UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES E INFORMÁTICOS

TEMA:

"SISTEMA DE LOTERÍA DE APUESTAS DEPORTIVAS EN EL FÚTBOL CON SPRING FRAMEWORK PARA LA EMPRESA ALQUIMIASOFT S.A."

Trabajo de graduación. Modalidad: Proyecto de investigación, presentado previo la obtención del título de Ingeniero en Sistemas Computacionales e Informáticos.

LINEA DE INVESTIGACIÓN: Desarrollo de Software
SUBLINEA DE INVESTIGACIÓN: Aplicaciones WEB

AUTOR: Marcelo Daniel Sandoval Unapucha
TUTOR: Ing. Mg. Carlos Israel Núñez Miranda

Ambato – Ecuador
Enero/2020

APROBACIÓN DEL TUTOR

En mi calidad de Tutor del Trabajo de Investigación sobre el tema: "SISTEMA DE LOTERÍA DE APUESTAS DEPORTIVAS EN EL FÚTBOL CON SPRING FRAMEWORK PARA LA EMPRESA ALQUIMIASOFT S.A.", del señor Marcelo Daniel Sandoval Unapucha, estudiante de la Carrera de Ingeniería en Sistemas Computacionales e Informáticos, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, considero que el informe investigativo reúne los requisitos suficientes para que continúe con los trámites y consiguiente aprobación de conformidad con el numeral 7.2 de los Lineamientos Generales para la aplicación de Instructivos de las Modalidades de Titulación de las Facultades de la Universidad Técnica de Ambato.

Ambato enero, 2020

EL TUTOR

Ing. Mg. Carlos Israel Núñez Miranda

AUTORÍA

El presente Proyecto de Investigación titulado: **"SISTEMA DE LOTERÍA DE APUESTAS DEPORTIVAS EN EL FÚTBOL CON SPRING FRAMEWORK PARA LA EMPRESA ALQUIMIASOFT S.A.",** es absolutamente original, auténtico y personal, en tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato enero, 2020

Marcelo Daniel Sandoval Unapucha

CC: 050413878-5

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato, para que haga uso de este Trabajo de Titulación, como un documento disponible para la lectura, consulta y procesos de investigación.

Cedo los derechos de mi Trabajo de Titulación, con fines de difusión pública, además autorizo su reproducción dentro de las regulaciones de la Universidad.

Ambato enero, 2020

Marcelo Daniel Sandoval Unapucha

CC: 050413878-5

APROBACIÓN DE LA COMISIÓN CALIFICADORA

La Comisión Calificadora del presente trabajo conformada por los señores docentes: Ing. Mg. Fernando Ibarra, Ing. Mg. Marcos Benítez, revisó y aprobó el Informe Final del Proyecto de Investigación titulado "SISTEMA DE LOTERÍA DE APUESTAS DEPORTIVAS EN EL FÚTBOL CON SPRING FRAMEWORK PARA LA EMPRESA ALQUIMIASOFT S.A.", presentado por el señor Marcelo Daniel Sandoval Unapucha de acuerdo al numeral 9.1 de los Lineamientos Generales para la aplicación de Instructivos de las Modalidades de Titulación de las Facultades de la Universidad Técnica de Ambato.

Ing. Mg. Elsa Pilar Urrutia Urrutia
PRESIDENTA ENCARGADA DEL TRIBUNAL

Ing. Mg. Fernando Ibarra
DOCENTE CALIFICADOR

Ing. Mg. Marcos Benítez
DOCENTE CALIFICADOR

DEDICATORIA

Dedico esta tesis a mis padres, por haber confiado en mí y haberme apoyado día a día en este largo camino de formación profesional, con su amor, paciencia y esfuerzo que me han permitido llegar a cumplir hoy un sueño más. Gracias por inculcar en mí el ejemplo de dedicación, esfuerzo y superación, de no rendirme ante las adversidades y sobre todo de no olvidar de dónde vengo, siendo siempre humilde y honesto con mi vida profesional y vida personal.

A mis hermanos, por su cariño y apoyo incondicional, durante todo este proceso, por estar conmigo en todo momento.

Marcelo Daniel Sandoval Unapucha

AGRADECIMIENTO

Agradezco a todo el personal docente y administrativo de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial por haber compartido sus conocimientos y darme la oportunidad de formarme como profesional.

Agradezco a mi jefe Serguéi Proaño, por haberme abierto las puertas de Alquimia para realizar mis prácticas preprofesionales y ahora darme la oportunidad de iniciar mi vida profesional, por todo su apoyo durante este proceso, con su amistad y experiencia.

Agradezco a mis compañeros de equipo de desarrollo por compartir sus conocimientos y sobre todo por brindarme su amistad incondicional durante todo este proceso.

Agradezco a mis hermanos Mauricio y Edison por apoyarme durante mi formación profesional con su cariño y consejos.

Agradezco a mi hermana Alejandrina por ser una fuente de inspiración de esfuerzo, dedicación y humildad, por darme la inspiración para haber escogido esta carrera y por apoyarme en los momentos más difíciles durante mi formación profesional.

Por último, un sincero agradecimiento al Ing. Carlos Núñez quien me dio la oportunidad de conocer la empresa Alquimiasoft y llegar a donde estoy ahora, gracias por compartir sus conocimientos, por ser un excelente docente y por ser un gran guía en el proceso de desarrollo de este proyecto.

Marcelo Daniel Sandoval Unapucha

ÍNDICE GENERAL DE CONTENIDOS

PORTADA i
APROBACIÓN DEL TUTOR ii
AUTORÍA iii
DERECHOS DE AUTOR iv
APROBACIÓN DE LA COMISIÓN CALIFICADORA v
DEDICATORIA vi
AGRADECIMIENTO vii
RESUMEN EJECUTIVO xvii
CAPITULO I 1
MARCO TEÓRICO 1
1.1. Antecedentes investigativos 1
1.2. Objetivos 2
1.2.1. General 2
1.2.2. Específicos 2
1.3. Fundamentación teórica 3
1.3.1. Apuestas deportivas online 3
1.3.2. Framework 3
1.3.3. Spring Framework 4
1.3.4. Sistemas de Gestión de Base de Datos 14
1.3.5 Metodologías Ágiles 14
1.3.6. Metodología SCRUM 15
1.3.7. Docker 18
1.3.8. Integración Continua 19
CAPÍTULO II 20
METODOLOGÍA 20
2.1. Materiales 20
2.2. Métodos 20
2.2.1. Modalidad de la investigación 20
2.2.2. Población y Muestra 20
2.2.3. Recolección de Información 21
2.2.4. Procesamiento y análisis de datos 21
2.2.5. Metodología de Desarrollo 22
2.2.6. Desarrollo del proyecto 22

CAPÍTULO III 23
RESULTADOS Y DISCUSIÓN 23
3.1. Análisis y recolección de la información 23
3.1.1. Recolección de la información 23
3.1.2. Análisis de la información 23
3.2. Desarrollo de la metodología 29
3.2.1. Etapa preliminar 29
3.2.2. Análisis comparativo de las herramientas para el Desarrollo 30
3.2.3. Planificación Inicial 36
3.2.3.1. Product Backlog 36
3.2.3.2. Desarrollo de Sprints 38
Sprint 1 39
Sprint 2 50
Sprint 3 84
Sprint 4 93
Sprint 5 97
Sprint 6 104
Sprint 7 114
Sprint 8 119
3.3. Resultados SCRUM 121
3.3.1. Gráfica resultante del progreso del proyecto. 121
3.3.2. Análisis de la gráfica. 122
3.4. Pruebas 123
Conceptos 123
Instalación de librerías con Gradle 123
Crear clase de objetos simulados 125
3.4.1. Pruebas Unitarias. 126
Pruebas Unitarias *User Service* 126
Pruebas Unitarias *Card Service* 130
Pruebas Unitarias *Prognostic Service* 132
Pruebas Unitarias *Notification Service* 133
Pruebas Unitarias *Result Service* 135
3.4.2. Pruebas de Integración 136
Pruebas de Integración *User Controller* 136
Pruebas de Integración *Card Controller* 138

Pruebas de Integración *Prognostic Controller* 139
Pruebas de Integración *Result Controller* 140
Ejecución general de pruebas y reporte Jacoco 141
3.5. Implementación del sistema 142
CAPÍTULO IV 150
CONCLUSIONES Y RECOMENDACIONES 150
4.1. Conclusiones 150
4.2. Recomendaciones 151
BIBLIOGRAFÍA 152
ANEXOS 155
Anexo N.º 1 – Encuesta para determinar requerimientos y la factibilidad para el desarrollo del sistema de apuestas deportivas online. 155
Anexo N.º 2 - Especificación de Requisitos según el estándar de IEEE 830. 157
Recursos hardware 167
Recursos software 168
Otros 168
Anexo N.º 3 – Catálogo de las historias de usuario del sistema de lotería de apuestas deportivas en el fútbol. 169
Anexo N.º 4. Manual de Usuario Final. 182
Anexo N.º 5. Manual de Usuario Administrador. 191

ÍNDICE DE FIGURAS

Figura 1. Inyección basada en el constructor. 6
Figura 2. Inyección basada en el setter. 6
Figura 3. Inyección basada en el campo. 6
Figura 4. Arquitectura Spring Framework. 7
Figura 5. Clase Spring Application Context. 10
Figura 6. Configuración application.yml para DB Mysql. 11
Figura 7. Interfaz básica para una entidad User. 11
Figura 8. Métodos JpaRepository. 11
Figura 9. Fórmula para calcular muestra de una población. 21
Figura 10. Resultado encuesta pregunta 1. 23
Figura 11. Resultado encuesta pregunta 2. 24
Figura 12. Resultado encuesta pregunta 3. 25
Figura 13. Resultado encuesta pregunta 4. 26
Figura 14. Resultado encuesta pregunta 5. 27
Figura 15. Resultado encuesta pregunta 6. 27
Figura 16. Resultado encuesta pregunta 7. 28
Figura 17. Modelo Base de Datos - parte 1. 39
Figura 18. Modelo Base de Datos - parte 2. 40
Figura 19. Modelo Base de Datos - parte 3. 40
Figura 20. Inicializar proyecto con spring boot. 41
Figura 21. Dependencias del proyecto spring. 41
Figura 22. Acceder a eclipse marketplace. 42
Figura 23. Instalar spring en eclipse. 43
Figura 24. Importar proyecto de spring io. 43
Figura 25. Ejecutar proyecto spring. 44
Figura 26. Ejecución inicial del proyecto spring. 44
Figura 27. Configuración inicial Application.yml 45
Figura 28. Opciones para creación de base de datos postgres. 46
Figura 29. Creación de base de datos en postgres. 46
Figura 30. Instalación de docker en Windows. 47
Figura 31. Iniciar docker. 47
Figura 32. Archivo Dockerfile. 48
Figura 33. Configuración docker-compose.yml 48
Figura 34. Ejecutar script docker. 49
Figura 35. Crear package config. 50
Figura 36. Configuración spring security. 50
Figura 37. Implementar librería JWT. 51
Figura 38. Parámetros de JWT token. 52
Figura 39. Archivo de configuración JWT. 52
Figura 40. Crear package service. 53
Figura 41. Crear archivo token service. 53
Figura 42. Función generar token. 54
Figura 43. Funciones para verificar token. 54
Figura 44. Crear package dto. 55
Figura 45. Crear archivo SessionDTO. 55

Figura 46. Función obtener nuevo access token. 56
Figura 47. Crear archivo JWT Filter. 56
Figura 48. Crear función peticiones sin token. 57
Figura 49. Implementar método doFilter. 57
Figura 50. Crear package model. 58
Figura 51. Crear Entidad Catalog. 59
Figura 52. Crear Entidad Catalog Detail. 60
Figura 53. Crear Entidad Team. 61
Figura 54. Crear Entidad User. 62
Figura 55. Crear Entidad Role. 63
Figura 56. Crear Entidad AdminUser. 63
Figura 57. Crear clave compuesta AdminUserRoleId. 64
Figura 58. Crear Entidad AdminUserRole. 65
Figura 59. Añadir relación lista de roles en AdminUser. 65
Figura 60. Modelo base de datos Usuarios y Administradores. 66
Figura 61. Crear package repository. 67
Figura 62. Crear interfaz UserRepository. 67
Figura 63. Crear package operations. 68
Figura 64. Crear UserDTO. 69
Figura 65. Crear interfaz UserOperations. 69
Figura 66. Crear servicio UserService. 70
Figura 67. Implementar método saveUserData. 70
Figura 68. Implementar método saveUser – parte 1. 71
Figura 69. Implementar método saveUser – parte 2. 72
Figura 70. Crear interfaz AdminUserRepository. 72
Figura 71. Crear clase AdminUserDTO. 73
Figura 72. Crear interfaz AdminUserOperations. 73
Figura 73. Crear servicio AdminUserService. 74
Figura 74. Implementar método register en AdminUserService. 74
Figura 75. Método para cargar data en AdminUser. 75
Figura 76. Método para guardar data en AdminUserRole. 75
Figura 77. Crear package controller. 75
Figura 78. Crear controlador UserController. 76
Figura 79. Exponer servicio createUser. 76
Figura 80. Agregar end point saveUser en filtro JWT. 77
Figura 81. Crear controlador AdminUserController. 77
Figura 82. Exponer servicio register AdminUser. 78
Figura 83. Probar end point saveUser. 78
Figura 84. Probar end point register AdminUser. 79
Figura 85. Declarar método getUserLogin. 79
Figura 86. Implementar método getUserLogin – parte 1 80
Figura 87. Implementar método getUserLogin – parte 2. 80
Figura 88. Declarar método login en AdminUserOperations. 80
Figura 89. Implementar método login en AdminUserService. 81
Figura 90. Exponer servicio getUserLogin. 81
Figura 91. Exponer servicio login AdminUser. 82
Figura 92. Probar end point login User. 82

Figura 93. Probar end point login AdminUser. 83
Figura 94. Entregar balones por registro. 83
Figura 95. Crear entidad Card. 84
Figura 96. Crear entidad CardDetail. 85
Figura 97. Crear interfaz CardRepository. 86
Figura 98. Crear función loadCardDTO. 86
Figura 99. Crear servicio getCardValid. 87
Figura 100. Exponer servicio getCardValid. 87
Figura 101. Prueba end point /card. 88
Figura 102. Crear entidad Prognostic. 89
Figura 103. Crear entidad PrognosticDetail. 89
Figura 104. Función obtener pronostico a guardar. 90
Figura 105. Crear función savePrognostic. 90
Figura 106. Servicio guardar pronóstico usuario. 91
Figura 107. Crear end point save/prognostic. 91
Figura 108. Crear servicio updateUser parte –1. 92
Figura 109. Crear servicio updateUser parte – 2. 92
Figura 110. Crear end point user/update. 92
Figura 111. Crear entidad CardDetailStats. 93
Figura 112. Crear entidad LastMatchesStats. 94
Figura 113. Crear función setStatsDetailDTO. 95
Figura 114. Crear función setStatistics. 95
Figura 115. Actualizar DTO retorno getCardValid (estadísticas). 95
Figura 116. Crear entidad CardDetailComments. 96
Figura 117. Crear función loadComments. 96
Figura 118. Actualizar DTO retorno getCardValid (comentarios). 97
Figura 119. Crear clase EditCardRequestDTO. 97
Figura 120. Crear clase EditCardDetailRequestDTO. 98
Figura 121. Servicio createCard – parte 1. 98
Figura 122. Servicio createCard – parte 2. 99
Figura 123. Crear end point /create. 99
Figura 124. Crear función updateScores. 100
Figura 125. Crear servicio saveCardResults. 100
Figura 126. Crear end point /save en CardController. 100
Figura 127. Query getNotProccessPrognostics. 101
Figura 128. Validar prognostic value. 101
Figura 129. Crear función procesar partidos pronósticos. 101
Figura 130. Crear servicio processPrognostics. 102
Figura 131. Crear package schedule. 102
Figura 132. Crear componente batch. 103
Figura 133. Crear Schedule processBatch. 103
Figura 134. Crear entidad Notification. 105
Figura 135. Crear entidad NotificationDetail. 105
Figura 136. Crear interfaz NotificationRepository. 106
Figura 137. Crear servicio NotificationService – parte 1. 107
Figura 138. Crear servicio NotificationService – parte 2. 107
Figura 139. Crear servicio Aplicar Notificaciones – parte 1. 108

Figura 140. Crear servicio Aplicar Notificaciones – parte 2. 109
Figura 141. Crear controlador NotificationController. 110
Figura 142. Crear notificaciones por acierto. 111
Figura 143. Asignar balones a ganadores. 111
Figura 144. Crear servicio getResults. 112
Figura 145. Servicio para obtener resultados de cartillas. 113
Figura 146. Exponer servicios para resultados. 113
Figura 147. Crear servicio obtener cartillas por usuario. 114
Figura 148. Exponer servicio obtener cartillas. 114
Figura 149. Agregar configuraciones spring mail. 115
Figura 150. Archivo de configuración SpringMailConfig. 116
Figura 151. Envío de correos a ganadores – parte 1 117
Figura 152. Envío de correos a ganadores – parte 2 117
Figura 153. Función crear notificaciones Email. 118
Figura 154. Actualizar proceso cerrar cartilla envío de emails. 118
Figura 155. Servicio envío de correos de bienvenida. 119
Figura 156. Enviar correo de bienvenida en registro. 119
Figura 157. Crear servicio envío correo con clave temporal. 119
Figura 158. Servicio restaurar contraseña. 120
Figura 159. Crear end point restaurar contraseña. 120
Figura 160. Actualizar datos de un usuario 121
Figura 161. Crea end point actualizer datos de usuarios. 121
Figura 162. Grafica Burn Down SCRUM. 122
Figura 163. Implementar librerías test 123
Figura 164. Tareas gradle test y test integration. 124
Figura 165. Configuración Jacoco Report. 125
Figura 166. Clase Objects con objetos fake. 126
Figura 167. User Service Test - parte 1 127
Figura 168. User Service Test - parte 2. 127
Figura 169. User Service Test - parte 3. 128
Figura 170. User Service Test - parte 4. 129
Figura 171. Ejecutar User Service Test. 129
Figura 172. Reporte Jacoco User Service Test. 130
Figura 173. Card Service Test - parte 1. 130
Figura 174. Card Service Test - parte 2. 131
Figura 175. Card Service Test - parte 3. 131
Figura 176. Prognostic Service Test - parte 1. 132
Figura 177. Prognostic Service Test - parte 2. 132
Figura 178. Prognostic Service Test - parte 3. 133
Figura 179. Notification Service Test - parte 1. 133
Figura 180. Notification Service Test - parte 2. 134
Figura 181. Notification Service Test - parte 3. 134
Figura 182. Result Service Test - parte 1. 135
Figura 183. Result Service Test - parte 2. 135
Figura 184. Result Service Test - parte 3. 136
Figura 185. User Controller Integration Test - parte 1 136
Figura 186. User Controller Integration Test - parte 2 137

Figura 187. User Controller IntegrationTest - parte 3. 137
Figura 188. Card Controller Integration Test - parte 1. 138
Figura 189. Card Controller Integration Test - parte 2. 138
Figura 190. Card Controller Integration Test - parte 3. 139
Figura 191. Prognostic Controller Integration Test..... 139
Figura 192. Result Service Integration Test - parte 1..... 140
Figura 193. Result Service Integration Test - parte 2..... 140
Figura 194. Ejecutar todas las pruebas - parte 1. 141
Figura 195. Ejecutar todas las pruebas - parte 2. 141
Figura 196. Ejecutar todas las pruebas - parte 3. 142
Figura 197. Reporte Jacoco Test Total Cobertura..... 142
Figura 198. Configurar Jenkinsfile - parte 1. 143
Figura 199. Configurar Jenkinsfile - parte 2. 144
Figura 200. Configurar Jenkinsfile - parte 3. 145
Figura 201. Configurar Jenkins Pipeline - parte 1. 145
Figura 202. Configurar Jenkins Pipeline - parte 2. 146
Figura 203. Ejecución Jenkins Pipeline..... 146
Figura 204. Pantalla principal sistema. 147
Figura 205. Pantalla crear cuenta. 147
Figura 206. Pantalla principal sistema versión móvil..... 148
Figura 207. Pantalla crear usuario versión móvil..... 149

ÍNDICE DE TABLAS

Tabla 1. Aplicación escala de Likert pregunta 3. 25
Tabla 2. Aplicación escala de Likert pregunta 4. 26
Tabla 3. Aplicación escala de Likert pregunta 6. 28
Tabla 4. Análisis comparativo IDE's Java. 30
Tabla 5. Análisis comparativo SGBD. 32
Tabla 6. Análisis comparativo virtualización. 33
Tabla 7. Análisis comparativo Spring vs EJB. 33
Tabla 8. Análisis comparativo Metodología XP y SCRUM 35
Tabla 9. Esquema historias de usuario. 36
Tabla 10. Diseño de la Base de Datos. 37
Tabla 11. Estructura inicial del proyecto 37
Tabla 12. Construcción de scripts docker. 38

UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE INGENIERÍA EN SISTEMA COMPUTACIONALES E INFORMÁTICOS

"SISTEMA DE LOTERÍA DE APUESTAS DEPORTIVAS EN EL FÚTBOL CON SPRING FRAMEWORK PARA LA EMPRESA ALQUIMIASOFT S.A."

Autor: Marcelo Daniel Sandoval Unapucha.

Tutor: Ing. Mg. Carlos Israel Núñez Miranda.

Fecha: enero del 2020

RESUMEN EJECUTIVO

En el presente proyecto de investigación, se ha implementado un prototipo de un sistema de lotería de apuestas deportivas en el fútbol con Spring Framework en la empresa Alquimiasoft S.A. Para el desarrollo de la investigación se realizó un levantamiento de requerimientos con la empresa y el equipo de desarrollo, apoyados en la información obtenida de la aplicación de encuetas a una muestra de la población total del cantón Ambato. También se realizó un estudio teórico de los conceptos y métodos de aplicación de las herramientas tecnológicas utilizadas en el desarrollo, se estudió a profundidad el framework Spring en aspectos como: definiciones, características, ventajas, el uso de subproyectos tales como, Spring Boot, Spring Security y Spring Data, entre otros.

Se realizó el estudio de las demás tecnologías como Docker, Bases de Datos e Integración Continua. Además, se realizó el estudio de la aplicación de las metodologías agiles en el desarrollo de software, enfocado en la metodología SCRUM, que se aplicó al desarrollo del presente proyecto.

Para el sistema de lotería de apuestas deportivas en el fútbol, se impuso por parte de la empresa el desarrollo a base de pruebas (TDD), por tanto, se han realizado pruebas Unitarias y Pruebas de integración, que garantizan un software funcional y de calidad

CAPITULO I
MARCO TEÓRICO

1.1 Antecedentes investigativos

Peter Gomber, Peter Rohr y Uwe Schweickert en el año 2008, en su artículo *"Sports betting as a new asset class—current market organization and options for development"* demostraron que las apuestas deportivas son una nueva clase de activos basándose en su relevancia económica y el crecimiento del mundo tecnológico, delimitando su investigación al mercado europeo, y concluyeron que las apuestas deportivas constituyen un mercado económico relevante en Europa gracias a su ámbito legal, además de que la tecnología y los nichos legales han permitido internacionalizar la distribución y el mercado. Este antecedente investigativo aporta en gran parte a la justificación del proyecto, demostrando el potencial de las apuestas deportivas en el mercado mediante la tecnología y las leyes que las respaldan [1].

Álvaro Hernández publicó un artículo en su blog de startups titulado *"Startups españolas innovando en las apuestas online"* donde menciona a la empresa BetRocket, quienes crearon un sistema de apuestas deportivas en el año 2013 en España, justificando su proyecto por la gran demanda del mercado en plataformas online como lo son Bet 365 o Bwin. Este antecedente investigativo aporta a la investigación en el análisis para plantear la lógica de negocio del sistema de apuestas deportivas en el fútbol [2].

Hibai López, Frederic Guerrero, Ana Estévez y Mark Griffiths en su artículo *"Betting is Loving and Bettors are Predators: A Conceptual Metaphor Approach to Online Sports Betting Advertising"* mencionan que las apuestas deportivas en línea son una economía que crece a nivel mundial y que, en Europa en el año 2016, OSB representó un negocio anual de € 16.5 mil millones en ingresos brutos de juego. Además, el estado legal de OSB en la Unión Europea ha dado lugar a un gran número de operadores de apuestas que están legalmente disponibles para los consumidores [3].

1.2 Objetivos

1.2.1 General

Implementar un prototipo de un sistema de lotería de apuestas deportivas en el fútbol con Spring Framework en la empresa Alquimiasoft S.A.

1.2.2 Específicos

- Establecer conjuntamente con la empresa y el equipo de desarrollo los requerimientos para el prototipo del sistema de lotería de apuestas deportivas en el fútbol.

- Efectuar un estudio de las herramientas a utilizarse en el desarrollo del proyecto, profundizando la investigación en Spring Framework.

- Desarrollar un producto viable del sistema de lotería de apuestas deportivas en el fútbol.

Cada uno de los objetivos se ha cumplido durante el desarrollo del presente proyecto de investigación, mediante el análisis y recolección de la información a través de encuestas, obteniendo información crucial para establecer conjuntamente con la empresa los requerimientos del sistema.

Con la experiencia y conocimientos de los desarrolladores de la empresa Alquimiasoft en nuevas tendencias para el desarrollo de software, se han establecido las tecnologías que se utilizaron durante el desarrollo, aplicando los conocimientos obtenidos previamente durante un proceso de estudio de estas a través de cursos online, talleres dentro de la empresa y practicando la programación en pares.

El estudio previo del uso de las tecnologías para el desarrollo de software ha permitido que el equipo de desarrollo construya en poco tiempo un software de calidad y que cumple con los requerimientos establecidos al principio del proyecto.

Finalmente, se ha implementado el prototipo de un sistema de lotería de apuestas deportivas en el fútbol con Spring Framework en el ambiente de integración continua de la empresa Alquimiasoft con ayuda de la documentación y asesoría de los DevOps de la empresa.

1.3 Fundamentación teórica

1.3.1 Apuestas deportivas online

Las apuestas deportivas tuvieron su origen en el Reino Unido, con las carreras de caballos. Las carreras de caballos son una de las actividades más conocidas y con más antigüedad. Hoy en día, las apuestas deportivas han logrado instalarse a nivel mundial con la ayuda del internet. El internet ha permitido que las apuestas deportivas expandan su mercado de una manera rápida y fácil a través del mundo [4].

Casas de apuestas

Las *casas de apuestas* son los sitios oficiales donde se pueden realizar apuestas deportivas. Las casas de apuestas deportivas pueden ser físicas u online. Las casas de apuestas online tienen ventajas sobre las físicas, como son la comodidad de apostar desde cualquier lugar y la posibilidad de realizar apuestas con anticipación, entre otras. Las apuestas deportivas presenciales tienen la ventaja del anonimato del apostador, porque no se requiere de algún registro para permitirle apostar [4].

1.3.2 Framework

En informática, el término framework es muy utilizado durante el desarrollo de sistemas o aplicaciones y dependiendo del contexto en que éste se utilice puede interpretarse como una plataforma, marco de trabajo, infraestructura, armazón, entre otros. Un framework es una estructura genérica utilizada para colocar diferentes elementos según sean necesarios.

Un framework, es una abstracción de un código común que implanta una funcionalidad genérica, que puede ser modificada para desarrollar una nueva funcionalidad concreta, provisto por los desarrolladores o programadores de software [5].

Beneficio de utilizar un framework en el desarrollo de software

Un framework es una solución a los problemas recurrentes en el ámbito del desarrollo de sistemas, para reducir el tiempo que les toma a los programadores crear un producto funcional. Los framework permiten a los desarrolladores centrarse en la codificación

de funcionalidades específicas y ahorrar tiempo en la codificación de funcionalidades comunes como por ejemplo un proceso de login [5].

Framework, y la teoría de las zonas frías y calientes.

Teóricamente, los frameworks están conformados por dos zonas, las zonas calientes (*hot spots*) y las zonas frías (*frozen spots*).

Las zonas frías de un framework se lo pude relacionar como el *core* de un software, un core que no puede ser modificado porque a partir de él se crean las relaciones entre los distintos elementos que componen el framework y permiten su correcto funcionamiento. Los componentes básicos de un framework permanecen inalterados en cualquier instancia de este.

Las zonas calientes de un framework son los puntos donde los programadores tienen la libertad de añadir su propio código funcional, respetando la arquitectura del framework en el que se esté trabajando [5].

Existen diversos framework en el mundo de la informática, que permiten el desarrollo de sistemas en los diferentes lenguajes de programación que existen. Para el presente proyecto de investigación, la empresa Alquimiasoft impone que para el desarrollo del sistema de apuestas deportivas en el fútbol sea en el lenguaje Java en su versión 8 y con Spring Framework.

1.3.3 Spring Framework

Spring es un framework de desarrollo de aplicaciones para Java Enterprise (JEE), que fue escrito inicialmente por Rod Johnson. Spring framework fue lanzado por primera vez en junio del año 2003, pero oficialmente su lanzamiento se dio en marzo del 2004 como una plataforma Java de código abierto. Spring ha llegado a convertirse en el framework más popular para Java empresarial, que ayuda a construir código de alto rendimiento, liviano y reutilizable, además de la robustez que caracteriza a JEE [6].

Spring Framework es una plataforma que se encarga del manejo de toda la infraestructura que actúa de soporte para desarrollar aplicaciones Java, uniendo los componentes de la aplicación, manejando sus ciclos de vida y encargándose en la interacción entre ellos. Con Spring Framework los programadores se pueden centrar

en el desarrollo de la aplicación y dejar que el marco de trabajo Spring se encargue de la infraestructura [6].

Inyección de dependencias

La reutilización de código en programación consiste en utilizar funcionalidades de otras clases donde se lo necesite, sin la necesidad de reescribir el mismo código. El uso de clases y objetos es muy común en Java y éstas deben ser lo más independientes posibles de otras para aumentar las posibilidades de reutilizar su código, además esto permite testear las clases de manera independiente con pruebas unitarias. Spring Framework implementa la inyección de dependencias para juntar las clases independientes y mantenerlas; en lugar de que el desarrollador se preocupe por instanciar los objetos, Spring se encarga de todo el proceso [7].

Spring Framework en sus primeras versiones permitía la inyección de dependencias mediante configuraciones XML, pero era un trabajo muy extenso el tener que declarar los componentes de la aplicación y sus dependencias para que Spring lo interprete. Spring en sus últimas versiones permite el uso de anotaciones, y para implantar la inyección de dependencias basta con declarar la anotación **@Autowired** encima de un atributo, método setter o del constructor, dejando de lado el uso de XML [8].

Tipos de inyección de dependencias

Existen 3 tipos de inyección de dependencias:

1. **Inyección de independencias basada en el constructor (Constructor-based)**

   En este tipo de inyección, el constructor de la clase lleva la anotación **@Autowired,** además de incluir los parámetros que se requiera inyectar. Es muy aconsejable utilizar este tipo de inyección cuando se requiere declarar dependencias obligatorias [9].

```
@Component
public class TheClientBean{

  private final TheBean theBean;

  @Autowired
   public  TheClientBean (TheBean theBean) {
     this.theBean = theBean;
   }

   .....
}
```

Figura 1. Inyección basada en el constructor.
Fuente: https://www.logicbig.com/tutorials/spring-framework/spring-core/types-of-dependency-injection.html.

2. Inyección de dependencias basada en setter (Setter-based)

En este tipo de inyección, los métodos setters llevan la anotación **@Autowired.** El contenedor de Spring se encargará de llamar a los métodos setters luego de que el Bean se haya instanciado para inyectar sus dependencias [9].

```
public class TheClientBean{

  private TheBean theBean;

  @Autowired
   public  setTheBean (TheBean theBean) {
     this.theBean = theBean;
   }

   .....
}
```

Figura 2. Inyección basada en el setter.
Fuente: https://www.logicbig.com/tutorials/spring-framework/spring-core/types-of-dependency-injection.html.

3. Inyección de dependencias basada en campos (Field-based)

En este tipo de inyección, los campos de la clase llevan la anotación **@Autowired.** El contenedor de Spring se encargará de asignar un valor a los campos luego de que la clase se haya instanciado [9].

```
public class TheClientBean{

  @Autowired
   private TheBean  theBean;

   .....
}
```

Figura 3. Inyección basada en el campo.
Fuente: https://www.logicbig.com/tutorials/spring-framework/spring-core/types-of-dependency-injection.html.

Arquitectura Spring

Spring Framework tiene una arquitectura organizada en capas que consta de diferentes módulos. Cada módulo tiene su propia funcionalidad y conjuntamente forman una aplicación. Existen aproximadamente 20 módulos que se agrupan en ***Core Container, Data Access/Integration, Web, AOP, Instrumentation, Test.*** Esta arquitectura le da al desarrollador la libertad de elegir si un módulo es necesario o no dentro de la aplicación empresarial a construir. La arquitectura modular de Spring permite una integración fácil y rápida con otros marcos de trabajo [10].

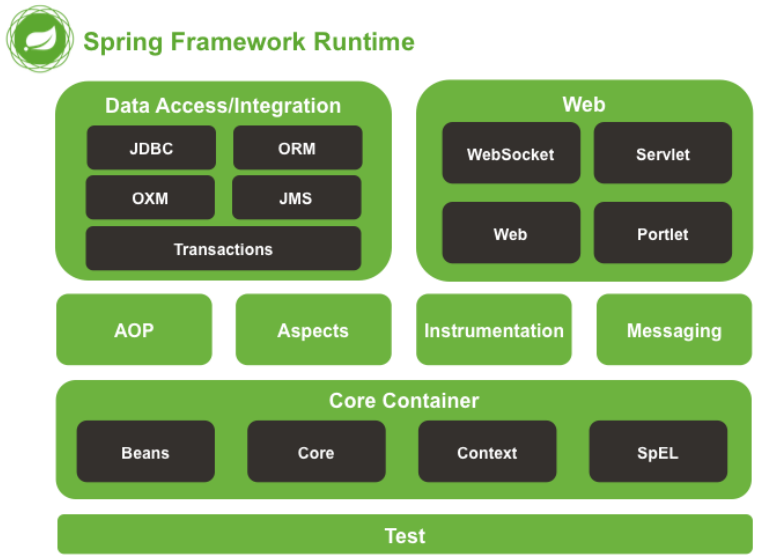


Figura 4. Arquitectura Spring Framework.
Fuente: https://data-flair.training/blogs/spring-framework-architecture/

En la figura 4 se aprecia la arquitectura modular de Spring, donde:

- **Core Container**

Este contenedor tiene 4 módulos.

1. **Spring Core:** Es el núcleo de Spring Framework. Proporciona implementación para las características como IoC (Inversión de control) o Inyección de dependencias.
2. **Spring Bean:** Proporciona una implementación para el patrón de diseño de fábrica a través del *BeanFactory*.
3. **Spring Context:** Está basado en el core y bean. Funciona como un medio para acceder a cualquier objeto definido y configurado. Su punto principal es la interfaz *ApplicationContext*.

4. **Spring Expression Languages (SpEL):** Proporciona una potente herramienta de lenguaje de expresión para consultar y manipular un gráfico de objeto [10].

- **Data Access/Integration**

Este contenedor tiene 5 módulos.

1. **JDBC:** Módulo que proporciona una capa de abstracción JDBC que elimina la necesidad de realizar la codificación JDBC y el análisis de códigos de error específicos del proveedor de base de datos.
2. **ORM:** Siglas de Object Relational Mapping. Este módulo se integra con las APIs de mapeo relacional de objetos como JPA, Hibernate, etc.
3. **OXM:** Siglas de Object XML Mappers. Módulo que se utiliza para convertir los objetos en formato XML y viceversa.
4. **JMS:** Siglas de Java Messaging Service. Este módulo contiene características para producir consumir mensajes entre varios clientes.
5. **Transactions:** Módulo para gestionar las transacciones y declaraciones de objetos POJO y clases que implementan interfaces especiales [10].

- **Spring Web**

Esta capa tiene 4 módulos.

1. **Web:** Este módulo tiene características de integraciones orientadas a la Web. Por ejemplo, funciones de carga de archivos e inicialización de IoC usando un servlet.
2. **Web-Servlet:** Este módulo implementa el patrón de diseño MVC (Modelo Vista Controlador) para aplicaciones Web.
3. **Web-Socket:** Este módulo proporciona soporte para la comunicación WebSocket y bidireccional entre el cliente y el servidor.
4. **Web-Portlet:** También conocido como Spring MVC Portlet. Proporciona el soporte para Portlets basados en Spring y refleja la funcionalidad de un módulo de Servlet Web [10].

- **Programación Orientada a Aspectos (AOP)**

Este es un módulo de Spring Framework que funciona como interceptor para agregar funcionalidades a la aplicación, como transacciones, seguridad, etc. [10]

- **Instrumentation**

Este módulo proporciona soporte de instrumentación de clase e implementaciones de carga de clase que se utiliza en ciertos servidores de aplicaciones [10].

- **Test**

Este módulo permite el desarrollo de pruebas de los componentes de la aplicación Spring con JUnit. También proporciona objetos simulados que ayudan a probar el código de manera aislada [10].

Spring Framework contiene subproyectos que añaden características específicas a una aplicación, como son la seguridad, transaccionalidad de datos, etc.
Entre los subproyectos más utilizados se encuentra:

Spring Boot

Es un subproyecto de Spring que se utiliza para configuraciones y proyectos de arranque. Spring Boot toma muy en serio la filosofía de la convención sobre la configuración, de tal manera que, para iniciar un proyecto con Spring, se requiere una configuración mínima, debido a que los valores predeterminados satisfarán las necesidades del desarrollador la mayor parte del tiempo [11].

Spring Boot viene con contenedores de servlets incrustados como Tomcat, Jetty o Undertow. Básicamente, se deduce de las dependencias del proyecto lo que va a usar y lo configura automáticamente para el mismo [11].

Spring Boot ha tenido éxito debido a las siguientes características que lo hacen fácil de utilizar:

- **Configuración:** Spring Boot dispone de un sofisticado módulo que autoconfigura todos los aspectos de una aplicación Spring, permitiendo ejecutar la aplicación sin tener que definir absolutamente nada.
- **Resolución de dependencias:** Spring Boot se encarga de resolver las librerías/dependencias para que la aplicación funcione, a partir del tipo de proyecto que se esté utilizando.
- **Despliegue:** Con Spring Boot se puede ejecutar aplicaciones Stand-alone, pero también es posible ejecutar aplicaciones web, mediante el despliegue de aplicaciones con un servidor web integrado, como es el caso de Tomcat, Jetty o Undertow.
- **Métricas:** Spring Boot cuenta con servicios que permiten consultar el estado de salud de una aplicación, consultando valores como el estado, memoria utilizada y disponible, número y detalle de los Beans creado por aplicación, etc.

- **Extensible:** Spring Boot permite la creación de complementos, ayudando a la comunidad de Software Libre a crear nuevos módulos que faciliten aún más el desarrollo [12].

Spring dispone de un servicio en línea llamado ***spring.io*** para la creación de proyectos utilizando Spring Boot, servicio que se utilizará para la creación del proyecto durante el desarrollo.

En la figura 5 se encuentra el código resultando de la creación de un proyecto con Spring Boot. El código es extraído de la clase Application Context, creada por el módulo Spring Context.

BalonazoApplication.java

```
1  package ec.com.alquimiasoft.balonazo;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.context.annotation.ComponentScan;
6  import org.springframework.scheduling.annotation.EnableScheduling;
7
8  @SpringBootApplication
9  @EnableScheduling
10 @ComponentScan(basePackages = "ec.com.alquimiasoft.balonazo")
11 public class BalonazoApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(BalonazoApplication.class, args);
15     }
16
17 }
```

Figura 5. Clase Spring Application Context.
Elaborado por: El investigador.

Spring Data

El manejo de base de datos relacionales o no relaciones ya no es un problema gracias a Spring Data. En el proceso de creación de una aplicación con Spring Boot se puede agregar el uso de JPA (API de persistencia de Java) que depende de Spring Data. El uso de Spring Data agilita la manipulación de cualquier base de datos mediante la interfaz *JpaRepository (*Interfaz de datos de Spring*)*, que dispone de funciones listas para la implementación CRUD de cualquier entidad.

Se puede ver en acción Spring Data luego de configurar el origen de datos en el archivo de configuración yml de una aplicación spring, y creando una Interfaz que extienda de *JpaRepository*.

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/ebook_chat
    username: root
    password: root
    testWhileIdle: true
    validationQuery: SELECT 1
  jpa:
    show-sql: true
    hibernate:
      ddl-auto: validate
      naming-strategy: org.hibernate.cfg.ImprovedNamingStrategy
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5Dialect
```

Figura 6. Configuración application.yml para DB Mysql.
Elaborado por: El investigador.

```
public interface UserRepository extends JpaRepository<User, String> {

}
```

Figura 7. Interfaz básica para una entidad User.
Elaborado por: El investigador

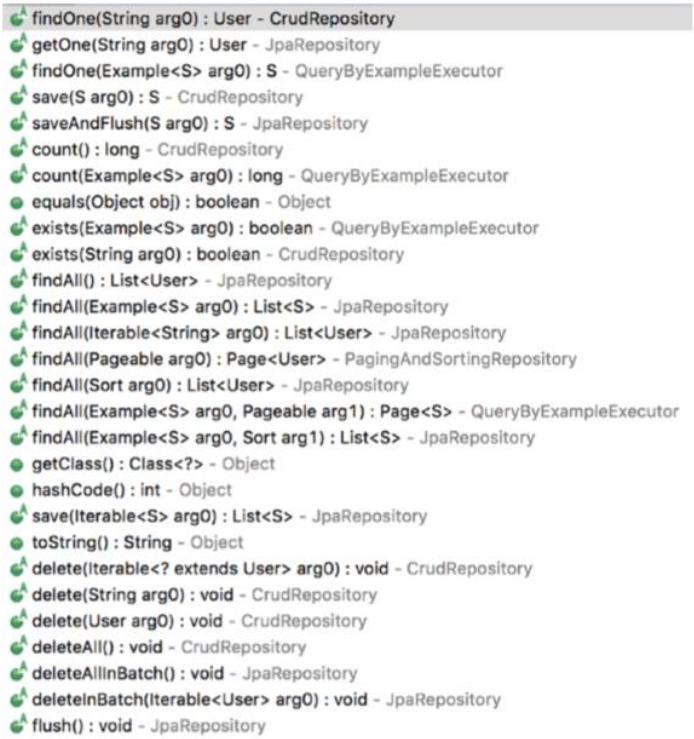


Figura 8. Métodos JpaRepository.
Elaborado por: El investigador.

Spring Security

Spring Security es un potente y altamente personalizable marco de autenticación y control de acceso. Es el estándar de fábrica para asegurar las aplicaciones basadas en Spring.

Spring Security es un marco que se enfoca en proporcionar tanto autenticación como autorización para aplicaciones Java. Como todos los proyectos de Spring, el verdadero poder de Spring Security se encuentra en la facilidad con que se puede extender para cumplir con los requisitos personalizados [13].

Spring Security actualmente soporta integración de autenticación con todas las siguientes tecnologías:

- HTTP BASIC authentication headers (an IEFT RFC-based standard).
- HTTP Digest authentication headers (an IEFT RFC-based standard).
- HTTP X.509 client certificate exchange (an IEFT RFC-based standard).
- LDAP (un enfoque muy comun para necesidades de autenticación multiplataforma, especificamente en entornos extensos).
- Form-based authentication (necesario para interfaces de usuario simples).
- OpenID authentication.
- Computer Associates Siteminder.
- JA-SIG Central Authentication Service.
- Transparent authentication context propagation for Remote Method Invocation (RMI) and HttpInvoker.
- Automatic "remember-me" authentication.
- Anonymous authentication.
- Run-as authentication.
- Java Authentication and Authorization Service (JAAS)
- Container integration with JBoss, Jetty, Resin and Tomcat (tambien podemos usar autenticación gestionada por el contenedor)
- Java Open Source Single Sign On (JOSSO)
- OpenNMS Network Management Platform
- AppFuse

- AndroMDA
- Mule ESB
- Direct Web Request (DWR)
- Grails
- Tapestry
- JTrac
- Jasypt
- Roller
- Elastic Plath
- Atlassian Crowd
- Sistemas de autenticación personalizados.

Ventajas de Spring Framework

Spring Framework obtuvo una inmensa popularidad y reconocimiento por la comunidad de desarrollo Java desde sus lanzamientos. Muchos desarrolladores utilizan Spring Framework para desarrollar código de alto rendimiento y reutilizable [14].

La siguiente lista [14] de ventajas son las razones por la que los desarrolladores prefieren Spring:

- Uso de POJO.
- Flexibilidad para configurar Spring.
- No necesita servidor.
- Uso de Spring AOP.
- No es necesario reinventar.
- Uso de modularidad.
- Facilidad de prueba.
- Consistencia en la gestión de transacciones.
- Marco Web bien diseñado (Spring MVC).
- Control de inversiones y APIs.

La empresa Alquimiasoft S.A implementa en sus proyectos de desarrollo Spring Framework en la capa de acceso a datos (Back End), justificando su uso en parte con las ventajas expuestas en el párrafo anterior y en la experiencia de sus desarrolladores

con el framework. Por tanto, la empresa impone el uso de Spring Framework para el desarrollo del presente proyecto.

1.3.4 Sistemas de Gestión de Base de Datos

Un Sistema de Gestión de Base de Datos (SGBD) permite la creación, gestión y administración de bases de datos. Es conocido también como un gestor de datos, y a través de él, se maneja todo acceso a la información en una base de datos. Un sistema de gestión de base de datos puede ser utilizado por cualquier persona con o sin conocimientos amplios de informática dentro de una empresa [15].

Se pueden clasificar los Sistemas de Gestión de Base de Datos por la forma en que administran los datos: **Relacionales (SQL)** y **No Relacionales (NoSQL).**

Sistemas Gestores de Bases de Datos Relaciones

Existen muchos SGBD actualmente, pero entro los más populares se encuentran:

- MySQL
- MariaDB
- SQLite
- PostgreSQL
- Microsoft SQL Server

PostgreSQL

Es un sistema de gestión de base de datos relacional orientada a objetos y fue publicada bajo la licencia BSD. Es un SGBD de código libre [15].

Sus principales **características** son:

- Control de Concurrencias multiversión (MVCC)
- Flexibilidad en cuanto a lenguajes de programación
- Multiplataforma
- Herramienta nativa PGAdmin para su administración.
- Robustez, Eficiencia y Estabilidad [15].

Para el presente proyecto se ha utilizado como gestor de base de datos a PostgreSQL por su robustez y todas las características mencionadas en el párrafo anterior.

1.3.5 Metodologías Ágiles

Las metodologías ágiles permiten a una empresa o equipo de desarrollo adaptar sus formas de trabajo a las condiciones del proyecto. El uso de estas metodologías en el desarrollo de software permite reducir los tiempos de entrega, reducir costos de desarrollo y mejorar la calidad del producto final [16].

Ventajas de implementar las metodologías ágiles.

Una de las características de las metodologías ágiles es involucrar al cliente durante el desarrollo del producto de software, para sumar su experiencia y conocimiento, además de mantener el enfoque del producto final y la satisfacción del cliente al conocer el estado del producto en todo momento [16].

El equipo de desarrollo también obtiene ventajas de las metodologías ágiles, mejorando su motivación, comunicación y productividad. Todos los miembros de un equipo de desarrollo están conscientes del estado del producto y existe un consenso al momento de tomar una decisión [16].

La empresa Alquimiasoft implementa en sus procesos de desarrollo metodologías ágiles y para el presente proyecto se ha establecido utilizar la metodología SCRUM, determinada en base al tamaño del proyecto y la cantidad de personas involucradas en el mismo.

1.3.6 Metodología SCRUM

La metodología SCRUM es comúnmente utilizado para el desarrollo de productos y software. SCRUM es adecuado para entornos que cambian de un momento a otro y el equipo tiene que reaccionar rápidamente y adaptarse a las situaciones. No es recomendable para entornos simples, estables y predecibles. SCRUM busca mejorar la productividad, reducir costos, controlar riesgos y entregar productos de calidad, mediante su enfoque empírico, iterativo e incremental. SCRUM, aunque es fácil de entender, puede ser difícil de dominar [17].

Sprint

El marco de trabajo de Scrum consta de una serie de iteraciones de desarrollo conocido como **Sprint.** Un sprint tiene una duración de tiempo máxima de 1 mes en el que se puede crear un producto utilizable e implementable. Cada Sprint tiene un objetivo alcanzable, el cual da como culminado el sprint y puede dar paso a uno nuevo. Durante el desarrollo de un Sprint, no se deben hacer alteraciones que pueden afectar potencialmente al objetivo de este [17].

Artefactos Scrum

- **Product Backlog**

Lista que contiene todos los requisitos que debe cumplir el producto final, generalmente llamados Historias de usuarios. Todos los ítems en el Product Backlog tienen una descripción, orden, estimación y valor.

- **Sprint Backlog**

Es una lista de las historias de usuario, que se deben cumplir durante el desarrollo del Sprint. Permiten realizar un seguimiento de la cantidad de trabajo faltante, también permite proyectar la probabilidad de alcanzar el objetivo del Sprint, por lo que son un medio de supervisión y administración del progreso.

- **Increment**

El resultado de culminar un Sprint debe aportar al progreso general del desarrollo del producto total. Un Sprint Backlog marcado como terminado debe incrementar a partir de los Sprints anteriores y tener una métrica y estimación de la finalización del desarrollo del producto [17].

Equipo Scrum

- **Scrum Master**

El Scrum Master no es el líder del equipo Scrum (el equipo se autoorganiza), su principal función es eliminar los obstáculos que impiden que el equipo Scrum logre alcanzar el objetivo del Sprint. El Scrum Master tiene la responsabilidad de ver y hacer cumplir las reglas y valores de una metodología ágil y Scrum. El Scrum Master ayuda en la interacción entre las personas ajenas al equipo y el Equipo Scrum.

- **Product Owner**

El Product Owner es el encargado de maximizar la productividad del equipo Scrum para entregar un producto de valor superior. El Product Owner también gestiona el Backlog del producto, priorizando los elementos que encuentre pertinentes para lograr objetivos establecidos y optimizar el valor del trabajo realizado por el equipo de desarrollo, y asegurarse de que el Product Backlog sea transparente y claro para todos.

- **Development Team**

El equipo de desarrollo es multifuncional y debe estar compuesto por al menos tres profesionales que trabajan para entregar un incremento del producto al final de cada Sprint. Todos los miembros del equipo de desarrollo son considerados como desarrolladores, independientemente del trabajo que realicen [17].

Roles Auxiliares Scrum

- **Usuarios**

Es a quien está dirigido el producto final y quien lo va a utilizar.

- **Stakeholders**

Son los clientes, proveedores, etc. Entidades quienes reciben un beneficio con el proyecto y formar parte de las revisiones del Sprint.

- **Managers**

Toma las decisiones finales participando en la sección de los objetivos y de los requisitos [17].

Eventos Scrum

- **Sprint Planning**

En este evento participa todo el equipo Scrum donde se determina el nuevo sprint a trabajar, el periodo de tiempo que va a durar y la cantidad de trabajo que se va a realizar. El equipo de desarrollo pronostica lo que puede desarrollarse de manera realista durante el Sprint. El Product Owner analiza y establece el objetivo del sprint que debe tener en cuenta el equipo de desarrollo.

- **Daily Scrum**

Es una reunión en la que forma parte el equipo de desarrollo y de ser necesario el Product Owner y Stakeholders. Cada día, y con una duración máxima de 15 minutos, se expresa ante todo el equipo el trabajo realizado en la última jornada por cada miembro, los impedimentos suscitados y el trabajo que se va ha realizar en la nueva jornada laboral. El Daily Scrum permite medir el progreso del Sprint y ver que se esté cumpliendo el objetivo de este.

- **Sprint Review**

Es una reunión que se realiza luego de terminar un Sprint, donde asisten el Equipo Scrum y las partes interesadas invitadas por el Product Owner. El Sprint Review tiene una duración aproximada de 4 horas. En esta reunión se discute y evalúa el trabajo realizado durante el Sprint, y el trabajo que va quedando pendiente. Durante la revisión del Sprint se revisa constantemente el Product Backlog para definir posibles elementos a ser tomados para el próximo Sprint.

- **Sprint Retrospective**

Es una reunión que se realiza después de culminado la revisión del sprint y se determinan los aspectos positivos y negativos que dejó el Sprint y establecer un plan de mejora continua del equipo de desarrollo [17].

1.3.7 Docker

Docker es una plataforma de software con la que un usuario tiene permitido crear, probar e implementar aplicaciones rápidamente. Docker utiliza contenedores como unidad estandarizada para empaquetar software que disponen de las librerías, herramientas y código necesario para su ejecución [18].

La tecnología Docker utiliza los contenedores como máquinas virtuales extremadamente livianas y modulares, que tienen la flexibilidad para crearlos, implementarlos, copiarlos y moverlos de un entorno a otro [18].

Docker Inc. dispone de un repositorio en la nube para almacenar las diferentes aplicaciones como sistemas operativos, gestores de base de datos, entre otros.

Ventajas de los contenedores Docker

- **Modularidad**

Al utilizar contenedores, un usuario tiene la posibilidad de implementar una parte de una aplicación, para actualizarla o repararla, sin tener que tomar la aplicación completa. Además, el uso de contenedores tiene un enfoque de micro servicios, permitiendo la comunicación entre aplicaciones similar al funcionamiento de una arquitectura orientada al servicio [18].

- **Control de versiones de imágenes y capas**

Un archivo de imagen Docker está construido por una serie de capas. Una capa se crea cuando la imagen se actualiza, y queda registrado ese cambio. Docker puede reutilizar estas capas para construir nuevos contenedores [18].

- **Restauración**

Mediante la utilización de capas para la creación de contenedores se pueden tener puntos de restauración y regresar a una versión específica en caso de requerirlo. Esto es con un enfoque de desarrollo ágil y permite tener **integración e implementación continua** desde una perspectiva de las herramientas [18].

- **Implementación rápida**

Una de las mayores ventajas de utilizar contendores docker es su implementación. Una aplicación se ejecuta en un contenedor creado a partir de una semilla que dispone de los recursos necesarios para su funcionamiento, esto permite optimizar los tiempos de implementación en aplicaciones complejas que requieren configuraciones extensas y una preparación previa de un sistema operativo [18].

1.3.8 Integración Continua

La práctica de una integración continua en el desarrollo de software consiste en integrar o combinar los cambios que se realizan al código en un repositorio centralizado de manera periódica, tras lo cual se ejecutan versiones y pruebas automáticas, apoyados de herramientas que automaticen dicho proceso [19].

Los objetivos claves de la integración continua consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que tarda en ser validado el lanzamiento de nuevas actualizaciones del producto [19].

CAPÍTULO II
METODOLOGÍA

2.1. Materiales

- Computadora de escritorio ACER
- Computadora portátil ASUS ROG STRIX
- Framework de desarrollo Spring
- Servidor Google Cloud

2.2. Métodos

2.2.1. Modalidad de la investigación

Las modalidades de investigación del proyecto son bibliográfica y de campo.

Investigación bibliográfica

Es una investigación bibliográfica porque se recolecta información de fuentes como libros, revistas, proyectos de investigación, etc. Para el desarrollo de la investigación.

Investigación de campo

Es una investigación de campo porque el proyecto se lo realiza a nivel empresarial y ésta aporta información fundamental al proyecto de investigación.

2.2.2. Población y Muestra

Dada la naturaleza de la investigación, se ha establecido como población al número de habitantes del cantón Ambato con un total de **329,856** personas

Dado el tamaño de la población se procedo a obtener una muestra en base a la siguiente fórmula:

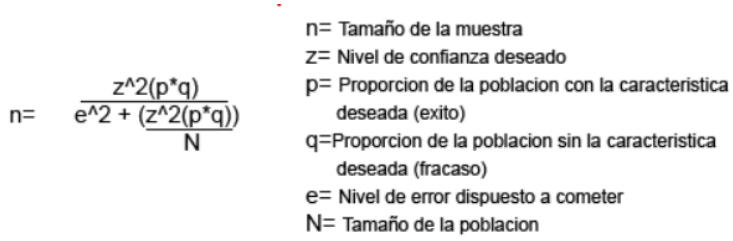


Figura 9. Fórmula para calcular muestra de una población.
Fuente: http://tesis.uson.mx/digital/tesis/docs/10705/Metodolog%C3%ADa.pdf

$$\boldsymbol{z = 1.65 = 90\%} \qquad \boldsymbol{p = q = 0.5}$$

$$\boldsymbol{e = 0.1 = 10\%}$$

$$\boldsymbol{n = \frac{(1.65)^2(0.5 * 0.5)}{(0.1)^2 + \frac{((1.65)^2(0.5 * 0.5))}{329856}}}$$

Tamaño total de la muestra obtenido: **68**

2.2.3. Recolección de Información

Para la recolección de información se utilizará las técnicas de encuestas.

2.2.4. Procesamiento y análisis de datos

- **Procesamiento de la Información**

  1) Elaboración de instrumentos para encuestas
  2) Aplicar la encuesta a los involucrados
  3) Tabulación de la información obtenida
  4) Estudio estadístico de datos para presentación de resultados

- **Análisis e interpretación de resultados**

  1) Análisis crítico de la información recolectada
  2) Análisis de los resultados estadísticos en base a los objetivos
  3) Interpretación de resultados
  4) Comprobación de los objetivos
  5) Establecimiento de conclusiones y recomendaciones

2.2.5. Metodología de Desarrollo

En el presente proyecto se ha aplicado la metodología **SCRUM** para el desarrollo del sistema de software, debido a que la empresa Alquimiasoft implementa esta metodología ágil para el desarrollo de sus aplicaciones.

Para el presente proyecto se han completado ***8 Sprints*** desarrollando sus respectivas actividades en lapsos de tiempo de 1 a 2 semanas por cada sprint.

2.2.6. Desarrollo del proyecto

Las actividades para llevar a cabo son las siguientes:

1) Levantar requerimientos del sistema.

2) Realizar el modelado de la Base de Datos

3) Construir scripts de Docker

4) Establecer la lógica del negocio

5) Desarrollar el sistema

6) Implementar conexión con FE

7) Realizar pruebas unitarias

8) Realizar pruebas de integración

9) Implementar el sistema desarrollado

CAPÍTULO III

RESULTADOS Y DISCUSIÓN

3.1. Análisis y recolección de la información

3.1.1. Recolección de la información

La recolección de la información para el desarrollo del presente proyecto se realizó a través de una encuesta, para determinar la lógica de negocio del sistema y establecer los requerimientos.

La encuesta fue dirigida a una muestra de 68 personas de la población total del cantón Ambato, lugar donde reside la empresa.

El instrumento utilizado para la recolección y análisis de la información se encuentra en el Anexo N.º 1.

3.1.2. Análisis de la información

La encuesta consta de 7 preguntas:

1) ¿Cuántas horas al día utiliza el servicio de internet?

En lo que respecta sobre el número de horas al día utilizando el internet, las personas opinaron que:

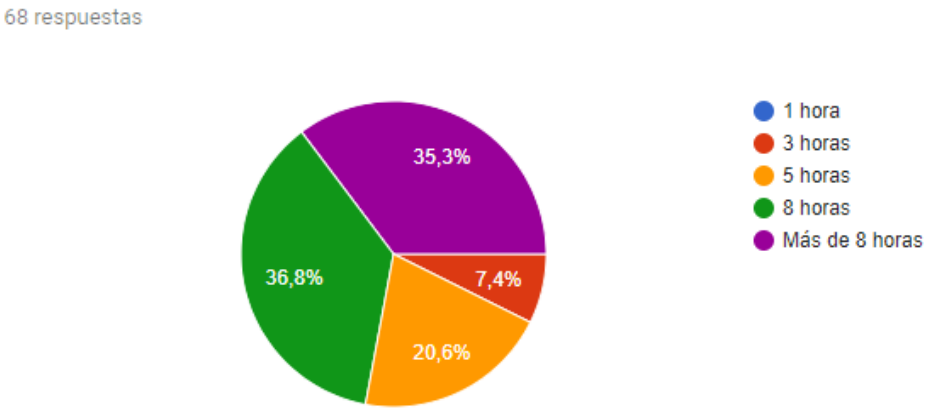


Figura 10. Resultado encuesta pregunta 1.
Elaborado por: El investigador.

Análisis: El 36.8% de los encuestados dicen que utilizan el servicio de internet unas 8 horas al día, el 35.3% dice que más de 8 horas, el 20.6% de las personas encuestadas dice que 5 horas y el 7.4% dice que 3 horas.

Conclusión: El 72.1% de las personas encuestadas utilizan el servicio de internet al menos 8 horas en el día, por lo tanto, es factible que el sistema de apuestas deportivas sea un servicio en línea.

2) **De la siguiente lista de dispositivos. ¿Cuáles utiliza usualmente para conectarse a internet?**

Con respecto a los dispositivos que usualmente se utilizan para acceder a la internet, las personas dicen que:

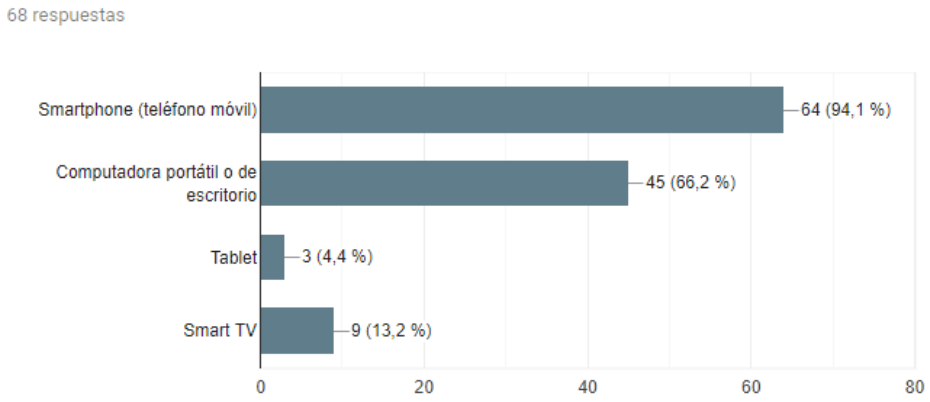


Figura 11. Resultado encuesta pregunta 2.
Elaborado por: El investigador.

Análisis: El 94.1% de los encuestados utiliza smartphone para acceder al servicio de internet y el 66.2% utiliza un computador, finalmente menos del 15% de encuestados utilizan otros medios como tablets, Smart TV u otros.

Conclusión: Los dispositivos móviles y computadores son los dispositivos más utilizados para acceder a la internet, por tanto, es factible que el sistema de apuestas se adapte a las distintas dimensiones de estos dispositivos.

3) ¿Qué tan importante es para usted el campeonato nacional de fútbol ecuatoriano?

Con respecto a lo importante que es el campeonato nacional de fútbol, las personas dicen que:

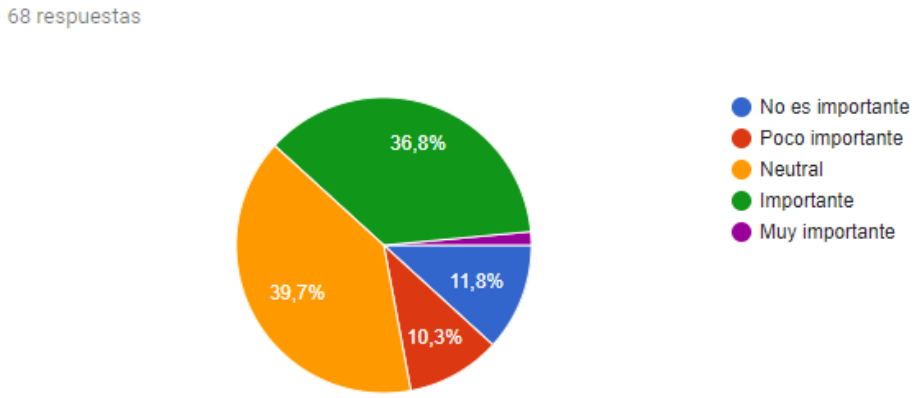


Figura 12. Resultado encuesta pregunta 3.
Elaborado por: El investigador.

Análisis: El 39.7% de los encuestados tiene afinidad neutral por el campeonato nacional de fútbol, el 36.8% dice que es importante, el 11.8% le parece que no es importante el campeonato nacional, el 10.3% dice que es poco importante y para el 1.5% es muy importante.

Aplicando una escala de Likert de 5 puntos, donde si el promedio total de respuestas es mayor a 3.5 es importante, si el valor se encuentra entre 3.0 y 3.5 es considerado regular y son respuestas con potencial a ser importante y si el valor es menor a 3.0 es considerado no importante.

Tabla 1. Aplicación escala de Likert pregunta 3.

	Valoración	Encuestados	Total
No es importante	1	8	8
Poco importante	2	7	14
Neutral	3	27	81
Importante	4	25	100
Muy Importante	5	1	5
		68	208
			3,06

Elaborado por: El investigador.

Conclusión: El valor promedio es 3.06, por tanto, el promedio de respuestas es considerado como propenso a ser importante el campeonato nacional de fútbol.

4) ¿Qué tan interesado está usted en el fútbol internacional?

Con respecto a lo importante que es el fútbol internacional, las personas dicen que:

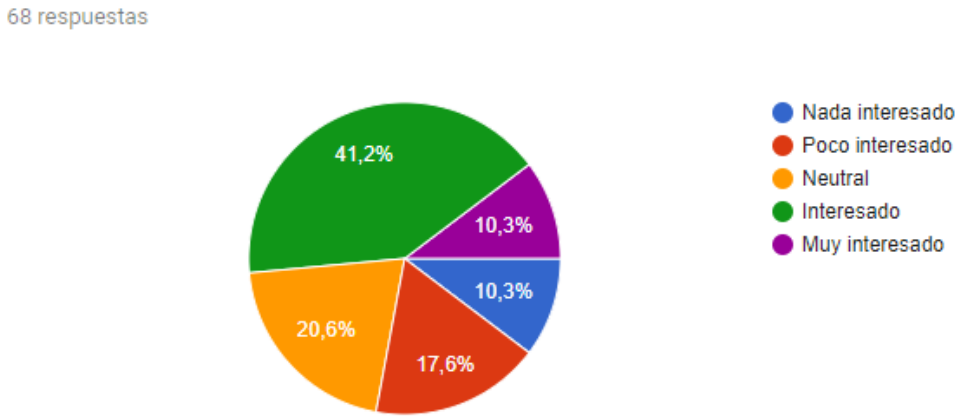


Figura 13. Resultado encuesta pregunta 4.
Elaborado por: El investigador

Análisis: El 41.2% de los encuestados tiene interés por el fútbol internacional, el 20.6% tiene afinidad neutral, el 17.6% está poco interesado en el fútbol internacional y el 10.3% dice que está nada interesado y también muy interesado, ambos casos con el mismo porcentaje.

Aplicando una escala de Likert de 5 puntos, donde si el promedio total de respuestas es mayor a 3.5 hay interés, si el valor se encuentra entre 3.0 y 3.5 es considerado regular y son respuestas con potencial a tener interés y si el valor es menor a 3.0 es considerado no interesado.

Tabla 2. Aplicación escala de Likert pregunta 4.

	Valoración	Encuestados	Total
Nada interesado	1	7	7
Poco interesado	2	12	24
Neutral	3	14	42
Interesado	4	28	112
Muy Interesado	5	7	35
		68	220
			3,24

Elaborado por: El investigador.

Conclusión: El valor promedio es 3.24, por tanto, el promedio de respuestas es considerado como propenso a estar interesado en el fútbol internacional.

5) ¿Conoce o escuchado sobre las apuestas en línea?

Con respecto al conocimiento sobre las apuestas deportivas en línea, las personas dicen que:

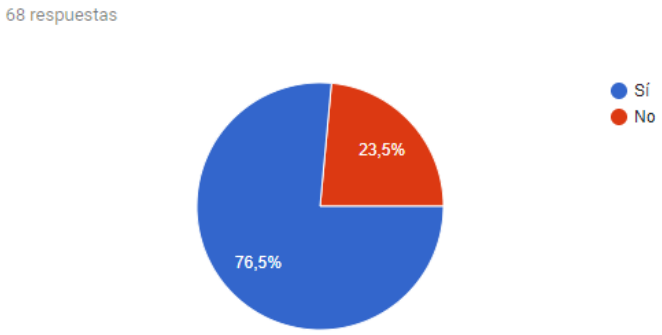


Figura 14. Resultado encuesta pregunta 5.
Elaborado por: El investigador

Análisis: El 76.5% de los encuestados tiene conocimiento sobre las apuestas deportivas en línea y el 23.5% lo desconoce.

Conclusión: Por tanto, un sistema de apuestas deportivas en línea no será tan desconocido para los usuarios en la actualidad.

6) ¿Qué tan probable es que usted participe en juegos de apuestas deportivas en línea?

Con respecto a la probabilidad de que participen en juegos de apuestas en línea, las personas dicen que:

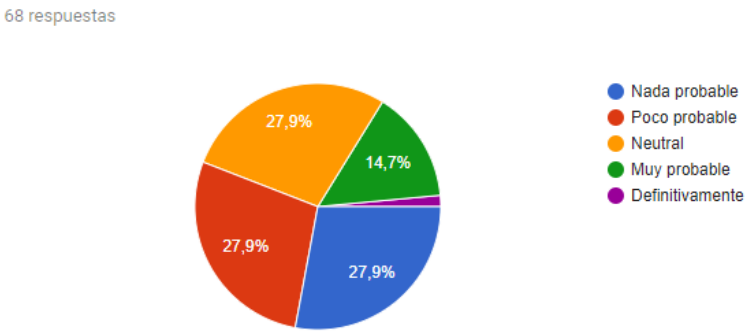


Figura 15. Resultado encuesta pregunta 6.
Elaborado por: El investigador

Análisis: El 27.9% dice que es poco probable, con el mismo porcentaje los encuestados tiene afinidad neutral y también dicen que es nada probable, el 14.7% dice que es muy probable y solo el 15% dice que definitivamente lo haría.

Aplicando una escala de Likert de 5 puntos, donde si el promedio total de respuestas es mayor a 3.5 es probable, si el valor se encuentra entre 3.0 y 3.5 es considerado regular y son respuestas con potencial a ser probable y si el valor es menor a 3.0 es considerado no probable.

Tabla 3. Aplicación escala de Likert pregunta 6.

	Valoración	Encuestados	Total
Nada probable	1	19	19
Poco probable	2	19	38
Neutral	3	19	57
Muy probable	4	10	40
Definitivamente	5	1	5
		68	159
			2,34

Elaborado por: El investigador.

Conclusión: El valor promedio es 2.34, por tanto, el promedio de respuestas es considerado como propenso a que no es probable participar en juegos de apuestas en línea.

7) ¿Cuánto dinero estaría dispuesto a apostar en juegos de apuestas deportivas?

Con respecto a la cantidad de dinero que estarían dispuestas a apostar, las personas dicen que:

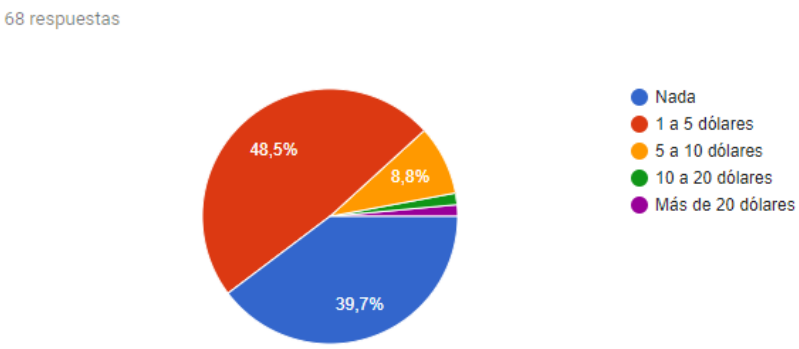


Figura 16. Resultado encuesta pregunta 7.
Elaborado por: El investigador

Análisis: El 48.5% dice que, de 1 a 5 dólares, el 39.7% dice que nada, el 8.8% dice que de 5 a 10 dólares y con el 1.5% los encuestados dicen que estarían dispuestos a apostar de 10 a 20 dólares o más.

Conclusión: Por tanto, las personas estarían dispuestas a invertir en el juego no más de 5 dólares.

3.2. Desarrollo de la metodología

3.2.1. Etapa preliminar

- **¿Qué y quién?**

La empresa Alquimiasoft S.A. desea desarrollar e implementar un sistema de lotería de apuesta deportivas en el fútbol con Spring Framework para el sistema en Back End y Angular para el sistema en Front End.

Los roles y miembros del equipo SCRUM se conforman de la siguiente manera:

Product Owner: Ing. Gonzálo Calahorrano.
Stakeholder: Ing. MSc. Serguéi Proaño
Scrum Master: Ing. MSc. Serguéi Proaño
Equipo de desarrollo:

- Ing. Christian Cartes (desarrollador Front End).
- Ing. Javier Fuentes (director de artes).
- Ing. MSc. Saúl Piña (DevOps).
- Ing. Pablo Solorzano (desarrollador Back End).
- Sr. Daniel Sandoval (desarrollador Back End).

- **¿Dónde y cuándo?**

El presente proyecto se lo desarrollará en las instalaciones de la empresa Alquimiasoft, ubicada en la ciudad de Ambato en la Avenida Miraflores.

Se estima un tiempo de desarrollo de 10 a 12 semanas a partir de la fecha de inicio del proyecto en octubre del 2018.

- **¿Por qué y cómo?**

El presente proyecto se lo desarrolla con fines económicos para ayudar en la sustentabilidad económica de la empresa.

Para el presente proyecto se han establecido distintas herramientas tecnológicas que faciliten el desarrollo e implementación de este. Las especificaciones de las herramientas a utilizar en el desarrollo se encuentran descritas en el Anexo 2.

3.2.2. Análisis comparativo de las herramientas para el Desarrollo

- **IDE de desarrollo**

Los IDE's de desarrollo son herramientas que permiten a los programadores escribir código en diferentes lenguajes de programación, brindando características de compilación, debug, integración con herramientas externas (Bases de Datos, Controlador de versiones, etc.), entre otros. [20]

Miguel Ángel Suma, en su artículo *Análisis comparativo de los diferentes IDE's para Java*, menciona los entornos de distribución libre como: Eclipse, NetBeans y también los entornos de distribución comercial como: IntelliJ IDEA.

En la tabla 4 se encuentra un análisis comparativo de las características requeridas para el desarrollo del presente proyecto.

Tabla 4. Análisis comparativo IDE's Java.

Parámetro	Variable	Eclipse	NetBeans	IntelliJ IDEA
Tipo de Licencia	Comunitaria	EPL	GPL	Apache 2.0
	Comercial	ninguna	ninguna	Trialware
Compilación	Capacidad para detectar errores	superior	buena	superior
	Capacidad para depurar errores	buena	buena	superior
Acceso a Base de Datos	Disponibilidad de Plugin	alta	alta	alta
	dificultad integración con DB	media	baja	media
Soporte técnico	Documentación	alta	alta	alta

	Compatibilidad con Spring Boot	si	si	si
	Spring Boot con Gradle	si	no	si
	Herramientas Spring Boot	instalable	instalable	integrado
Especificaciones Técnicas	RAM	4 GB	2 GB	8 GB
	SO	Multiplataforma	Multiplataforma	Multiplataforma
	Procesador	1.2 GH	2.6 GH	4 GH

Elaborado por: El investigador.

De la tabla 4 se puede concluir que los 3 IDE's tienen características muy similares, sin embargo, existen puntos en los que difieren y los cuales influyen en la decisión para elegir el IDE óptimo para el desarrollo del proyecto.

1) **Tipo de licencia:** La empresa Alquimiasoft utiliza herramientas de código abierto para el desarrollo de sus sistemas, por tanto, la disponibilidad de una licencia comunitaria es un punto para tomar en cuenta y las 3 herramientas cuentan con una.
2) **Capacidad para depurar errores:** El *debug* es una funcionalidad indispensable en el desarrollo de sistemas de software y la ayuda que brinde un IDE es importante para el programador. En este punto el IDE IntelliJ cuenta con un depurador de código superior a los demás.
3) **Compatibilidad con Spring Boot:** Las 3 herramientas son compatibles con Spring Boot, pero NetBeans solo es compatible con proyectos Spring con Maven. La empresa Alquimiasoft implementa la integración continua de sus proyectos utilizando Gradle, además de la ejecución de pruebas unitarias, por tal motivo es indispensable que un IDE sea compatible con Gradle.

Por lo expuesto anteriormente se concluye que los IDE's óptimos para el desarrollo del proyecto son Eclipse e IntelliJ, NetBeans fue descartado por no tener compatibilidad con Spring basado en Gradle. A pesar de que el IDE IntelliJ cuenta con mejores características en compilación, se decide utilizar el IDE Eclipse por sus requerimientos mínimos a nivel de Hardware.

- **Base de Datos**

Para el presente proyecto se requiere utilizar una base de datos relacional y de código abierto. Entre los sistemas de gestión de base de datos más populares se encuentran MariaDB y PostgreSQL.

En la tabla 5 se encuentra un análisis de los 2 sistemas de gestión de bases de datos, tomando en cuenta las características más importantes.

Tabla 5. Análisis comparativo SGBD.

Variable	MariaDB	PostgreSQL
Licencia	código abierto	código abierto
Ranking de motores DB	85.57	491.07
Modelo de base de datos	relacional	Relacional
Rendimiento	ligero y rápido. Ideal para consultas simples	robusto y estable. Ideal para proyectos grandes que requieren consultas complejas y largas
Tamaño de Base de Datos	pequeñas y medianas	medianas y grandes

Fuente: https://guiadev.com/postgresql-vs-mysql/

De la tabla 5 se puede concluir que el sistema de gestión de base de datos MariaDB es comúnmente utilizado para proyectos pequeños y medianos, mientras que PostgreSQL es utilizado en proyectos de mayor tamaño.

Por lo antes expuesto se concluye que la base de datos óptima para el proyecto es PostgreSQL debido a sus características de robustez y estabilidad.

- **Virtualización**

En la empresa Alquimiasoft se virtualizan los proyectos de desarrollo, con la finalidad de tener productos portables y fácil de desplegar en cualquier servidor.

Existen 2 formas de virtualización, mediante máquinas virtuales y contenedores docker. En la siguiente tabla se encuentran las principales diferencias entre las 2 tecnologías.

Tabla 6. Análisis comparativo virtualización.

Máquina virtual	Contenedor Docker
Aislamiento de procesos a nivel de hardware	Aislamiento del proceso a nivel del sistema operativo
Cada VM tiene un SO separado	Cada contenedor puede compartir OS
Inicio en minutos	Inicio en segundos
Las máquinas virtuales son de pocos GB	Los contenedores son livianos (KB / MB)
Las máquinas virtuales listas para usar son difíciles de encontrar	Los contenedores acoplables preconstruidos están fácilmente disponibles
Las máquinas virtuales pueden moverse al nuevo host fácilmente	Los contenedores se destruyen y se recrean en lugar de moverse
Crear VM lleva un tiempo relativamente más largo	Los contenedores se pueden crear en segundos.
Más uso de recursos	Menos uso de recursos

Fuente: https://geekflare.com/docker-vs-virtual-machine/

De la tabla 6 se concluye que el uso de la tecnología docker es la mejor opción para virtualizar aplicaciones. La empresa Alquimiasoft virtualiza todos sus proyectos para implementar la integración continua de manera rápida y fácil.

- **Framework de Desarrollo**

Para el presente proyecto se ha establecido utilizar el Framework Spring compatible con Java EE, pero existe otra tecnología compatible con esta tecnología y es EJB.

En la tabla 7 se encuentra un análisis comparativo de las principales características.

Tabla 7. Análisis comparativo Spring vs EJB.

Variables	EJB	Spring
Gestión de transacciones	EJB solo admite el administrador de transacciones JTA.	A través de su interfaz Platform Transaction

		Manager, Spring admite múltiples transacciones como JTA, Hibernate, JDO y JDBC.
Inyección de dependencias	Puede inyectar datos de EJB, recursos JMS, recursos JPA en el contenedor.	Se pueden inyectar listas, propiedades, mapas y recursos JNDI.
Persistencia	Admite la persistencia programática gestionada por beans y estrechamente acoplada a JPA	Proporciona un marco que admite la integración de varias tecnologías de persistencia como JDBC, Hibernate, JDO e iBATIS.
Gestión de estado	Admite beans de sesión con estado y contexto de persistencia extendida.	Admite la gestión de sesiones de contenedor web
Servicios Web	Admite beans de sesión con estado y contexto de persistencia extendida.	No admite ninguna integración directa de servicios web.
Seguridad	Admite soporte de seguridad tanto declarativo como programático a través de JAAS.	Proporciona seguridad declarativa a través del archivo de configuración de spring o metadatos de clase.
Computación distribuida	Proporciona llamadas a métodos remotos administrados por contenedor.	Proporciona soporte para llamadas remotas a través de RMI, JAX-RPC y servicios web.
Mensajería	Capacidades a través de beans controlados por mensajes.	Para el mensaje, se debe agregar la configuración de escuchas.
Scheduling	Proporciona programación simple a través del servicio EJB Timer	Para la programación es necesario agregar y configurar Quartz

Fuente: https://www.educba.com/ejb-vs-spring/

Con lo expuesto anteriormente se puede concluir que EJB es una especificación de Java EE que se implementó para cubrir y mejorar fallos en las primeras versiones de Java EE, mientras que Spring es un marco de trabajo que permite construir una aplicación funcional basada en Java EE.

Se ha escogido Spring para el desarrollo del presente proyecto por el simple hecho de ser un framework y el único que integra Java EE. Con un framework el programador

se puede centrar en el desarrollo funcional de la aplicación y dejar que el framework se encargue de la arquitectura de la aplicación.

- **Metodología para el Desarrollo**

En la ingeniería de software es común utilizar metodologías de desarrollo para garantizar un producto de calidad y reducir el tiempo que toma desarrollarlo.

Entre las metodologías más comunes y utilizadas se encuentran XP (Extreme Programming) y SCRUM. Las principales diferencias y características se encuentran en la siguiente tabla.

En la tabla 8 se encuentran las diferencias entre las 2 metodologías, de lo cual se concluye que la metodología XP se centra en la programación y SCRUM en la gestión del proyecto.

Tabla 8. Análisis comparativo Metodología XP y SCRUM

XP	SCRUM
Se centra en la programación y creación del producto.	Metodología enfocada a la administración del proyecto.
Se sigue estrictamente el orden de prioridad de las actividades definidas por el cliente.	Puede modificar el orden de prioridades establecido por el Product Owner en el Sprint Backlog.
Los miembros del equipo trabajan en parejas durante el proyecto.	Cada miembro del equipo Scrum trabaja de manera individual.
Su estructura es más cambiante y menos organizada.	Tiene una estructura más jerárquica y organizada.
Las iteraciones de entrega son de 1-3 semanas.	Los Sprint (iteraciones de entrega) se realizan cada 2-4 semanas.
Las tareas entregadas al cliente son susceptibles a modificaciones durante el	Al término de un Sprint, las tareas realizadas durante el Sprint Backlog y

proyecto, incluso si funcionan correctamente.	aprobadas por el cliente (Product Owner) no se vuelven a modificar.

*Fuente:*http://davidrtmetodosagiles.blogspot.com/2017/02/comparativa-entre-xp-y-scrum.html

Por lo antes expuesto se concluye que la metodología óptima para el desarrollo del presente proyecto es SCRUM, en parte por la cantidad de personas involucrados en el proyecto que ayudan en la gestionan e influyen en priorizar las tareas que se desarrollan en cada sprint, además de que cada programador trabaja de manera individual, tomando la programación en pares como opción válida para nivelar conocimientos.

3.2.3. Planificación Inicial

Como dicta la teoría de la metodología SCRUM, es necesario tener un planning inicial para la construcción de las historias de usuario que serán desarrolladas en los Sprints del proyecto. Las historias de usuario que se determinan en el planning inicial se colocan en el *product backlog*. A pesar de que las historias de usuario se han desarrollado tomando en cuenta la mayor cantidad de información del producto, es común que aparezcan nuevas historias durante el desarrollo y la metodología se adapta a estos sucesos.

3.2.3.1. Product Backlog

Esquema historias de usuario

Para el desarrollo de las historias de usuario se ha establecido el siguiente modelo:

Tabla 9. Esquema historias de usuario.

Historia de usuario	
ID	
Prioridad	
Riesgo	
Descripción	
Criterios de aceptación	

Elaborado por: El investigador

Donde:

Historia de usuario: Se reemplaza por el nombre asignado a la historia.
ID: Identificador de la historia (números enteros empezando por 1).
Prioridad: La importancia que tiene la tarea frente a las demás. (alta, media, baja).
Riesgo: Qué tanto impacto tiene al proyecto en caso de fallo (alto, medio, bajo).
Descripción: Una breve explicación de la historia de usuario (requerimientos, condiciones, etc.).
Criterios de aceptación: Son las condiciones que deben cumplirse para que la tarea se pueda dar por terminada.

Historias de Usuario

Tabla 10. Diseño de la Base de Datos.

Diseño de la Base de Datos	
ID	1
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que se diseñe el modelo de la base de datos para el sistema, tomando en cuenta los requerimientos de funcionalidad discutidos con anterioridad PARA almacenar y administrar la información que manejará el sistema.
Criterios de aceptación	• La base de datos tiene concordancia con los requerimientos establecidos.

Elaborado por: El investigador.

Tabla 11. Estructura inicial del proyecto.

Estructura inicial del proyecto	
ID	2
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que se cree la estructura inicial del proyecto en Spring Framework, tomando en cuenta el manejo de Spring Security, Spring

	Data y Pruebas PARA iniciar con el proceso de desarrollo del sistema. El repositorio por defecto para las librerías será Gradle y para las pruebas se utilizará JUnit en sus versiones más actuales.
Criterios de aceptación	• El proyecto tiene la estructura recomendada por Spring. • El proyecto cuenta con una sección de pruebas. • El proyecto usa Gradle como repositorio para las librerías del sistema. • El proyecto compila y se ejecuta sin problemas.

Elaborado por: El investigador.

Tabla 12. Construcción de scripts docker.

Construcción de scripts docker	
ID	3
Prioridad	Media
Riesgo	Medio
Descripción	YO como Product Owner QUIERO que se construya scripts docker del sistema en Spring Framework PARA tener la aplicación aislada en contenedores y mantener la estructura de desarrollo de proyectos de la empresa.
Criterios de aceptación	• El script se ejecuta y crea un contenedor con la aplicación Spring. • El contenedor se crea sin problemas.

Elaborado por: El investigador.

En la sección anterior (*Historias de usuario*) se encuentran descritas 3 historias de usuario como muestra del procedimiento de construcción de estas. El catálogo completo de las historias de usuario se encuentra en el Anexo 3.

3.2.3.2. Desarrollo de Sprints

Para el desarrollo de los Sprints, se ha establecido un tiempo de duración de 1 a 2 semanas para trabajar en cada Sprint.

- **Sprint 1**

- **Diseño de la base de datos**

Actividades:

- **Diseñar el modelo de la base de datos en Power Designer.**

Para el diseño de la base de datos se ha utilizado la herramienta DBeaver que permite modelar una base de datos de manera rápida, además es posible exportar scripts o archivos SQL para la creación de la base de datos. Spring Framework permite utilizar JPA para el manejo de datos, delegando la construcción de tablas y sus relaciones a las entidades dentro del proyecto Spring, tal como se muestra en las figuras 17, 18 y 19.

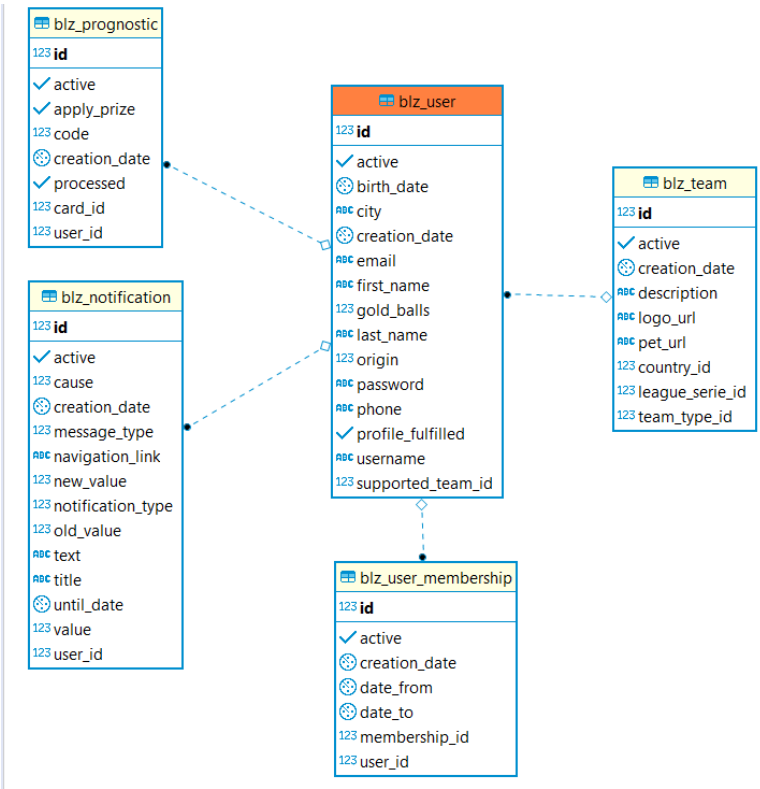


Figura 17. Modelo Base de Datos - parte 1.
Elaborado por: El investigador.

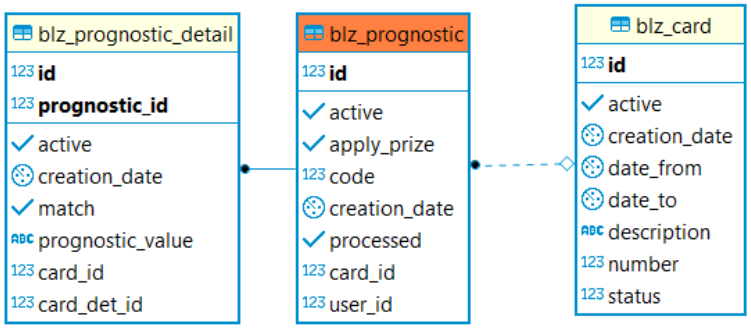


Figura 18. Modelo Base de Datos - parte 2.
Elaborado por: El investigador.

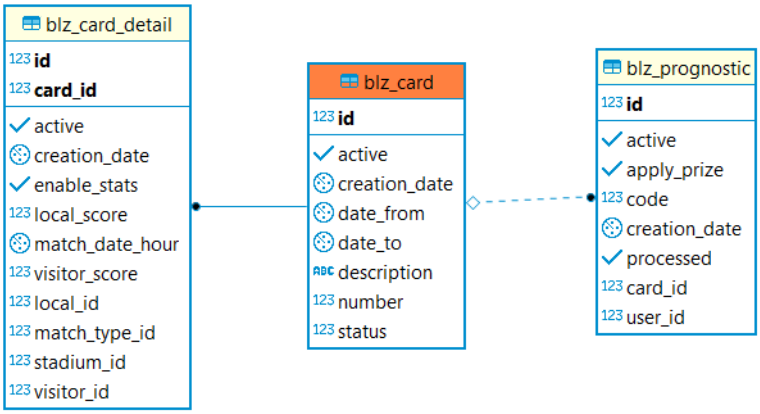


Figura 19. Modelo Base de Datos - parte 3.
Elaborado por: El investigador.

- **Estructura inicial del proyecto**

Actividades:

- **Crear el proyecto con spring io online**

Spring dispone de una herramienta en línea para la creación de proyectos, disponible en https://start.spring.io/. La herramienta online de Spring ejecuta Spring boot para la creación de la estructura base de un proyecto spring, además de permitir elegir el lenguaje de programación, el repositorio de dependencias y añadir distintas librerías al proyecto.

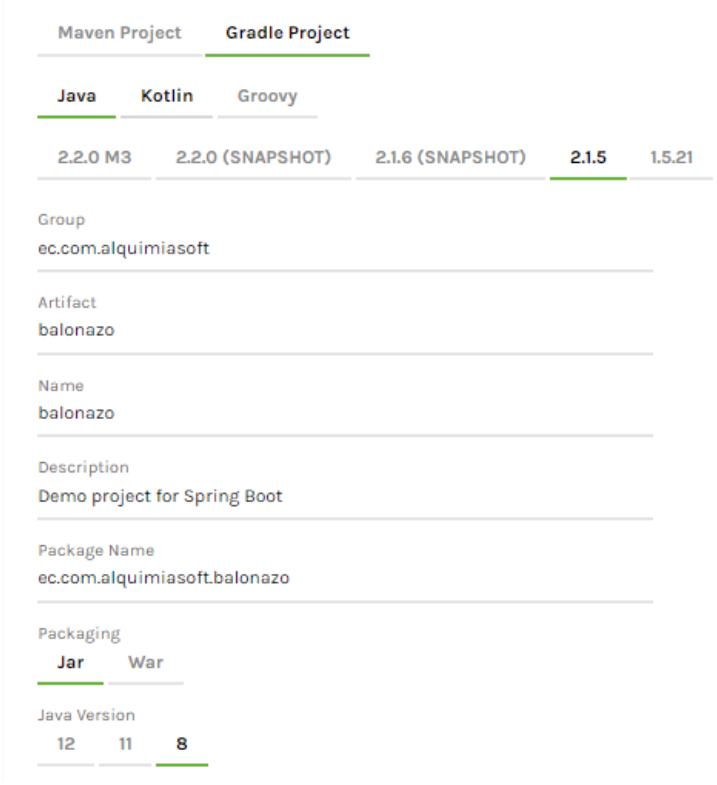


Figura 20. Inicializar proyecto con spring boot.
Elaborado por: El investigador

Para el presente proyecto se ha establecido utilizar el lenguaje de programación Java en su versión 8, además de utilizar el repositorio de dependencias Gradle y Spring Boot en su versión 2.1.5.

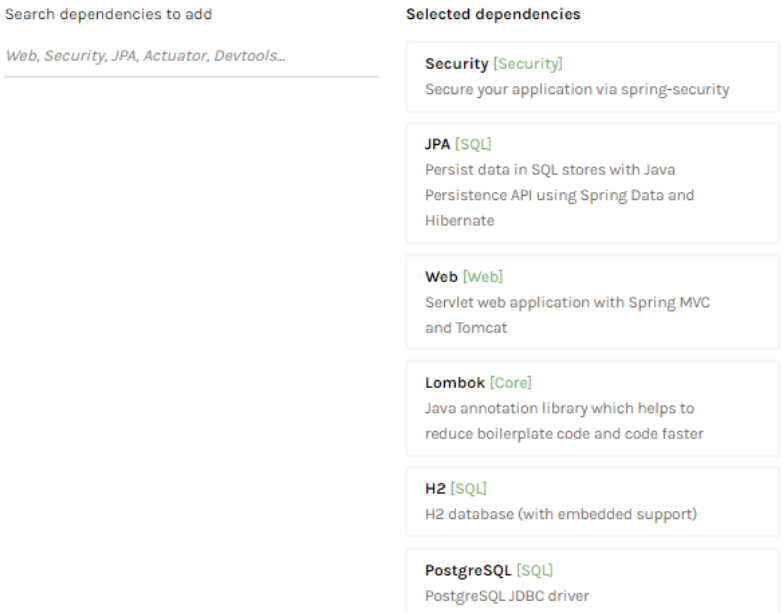


Figura 21. Dependencias del proyecto spring.
Elaborador por: El investigador.

Para el presente proyecto se han establecido utilizar las siguientes dependencias:

Spring security: Subproyecto de spring para el manejo de seguridades en un proyecto.
Spring MVC Web: Patrón de diseño MVC para el desarrollo de software Web.
Spring Data JPA: API de java para la administración de la base de datos.
Lombok: Librería que simplifica el manejo de Getters y Setters en las clases Java.
H2 Data Base: Base de datos relacional de Java. Es una base de datos muy liviana y se la puede utilizar a nivel de pruebas.
PostgreSQL: JDBC de la base de datos postgres. Es la base de datos escogida para el ambiente de desarrollo y producción en el presente proyecto.

Finalmente dar clic en ***generate the project*** para descargar el proyecto.

- **Instalar plugin spring en eclipse photon e importar el proyecto Spring.**

El IDE de programación eclipse dispone de un plugin para la ejecución de proyectos Spring, que se lo puede instalar desde Eclipse marketplace.

Para acceder a eclipse Marketplace se lo puede hacer desde el menú ***help*** ubicada en la barra de herramientas y finalmente seleccionar la opción ***Eclipse Marketplace,*** tal como se muestra en la figura 22**.**

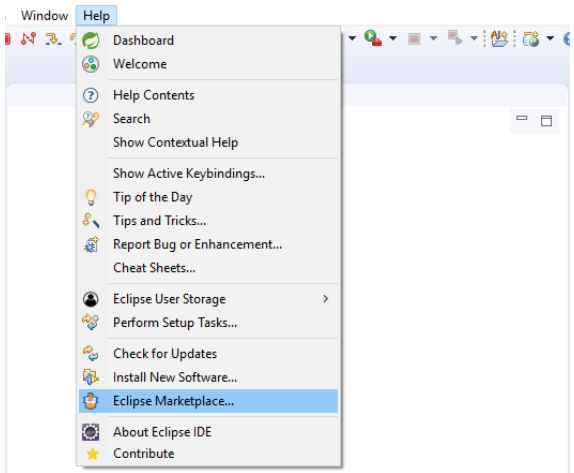


Figura 22. Acceder a eclipse marketplace.
Elaborado por: El investigador.

Dentro de eclipse marketplace proceder a buscar e instalar el plugin de ***spring.***

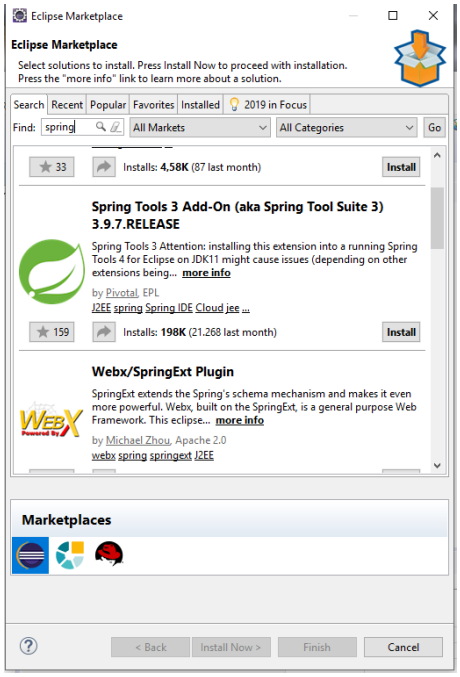


Figura 23. Instalar spring en eclipse.
Elaborado por: El investigador.

Aceptar los términos y condiciones y se instalará ***Spring tools*.**

- **Ejecutar proyecto spring.**

Para ejecutar el proyecto creado con spring io es necesario importar la carpeta con el proyecto. En eclipse se puede importar un proyecto desde la opción ***File*** y eligiendo ***importar*.** Para el presente proyecto se debe importarlo con Gradle para que pueda descargar las dependencias que necesite.

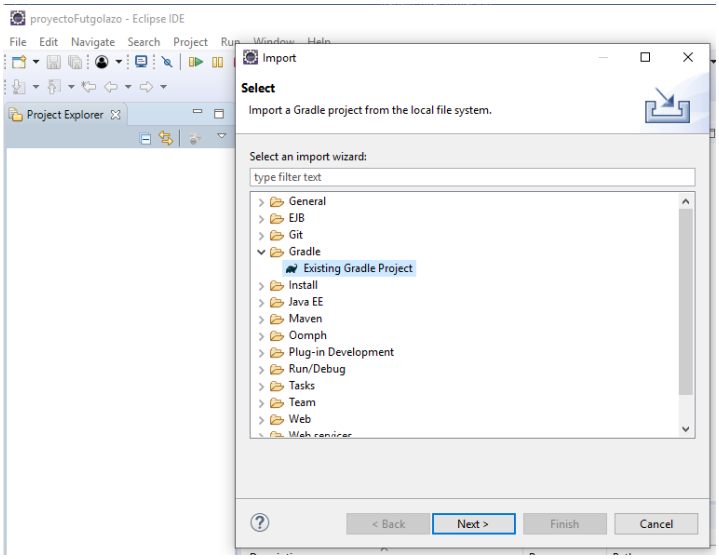


Figura 24. Importar proyecto de spring io.
Elaborado por: El investigador.

Finalmente, el proyecto ***spring*** se lo puede ejecutar dando clic derecho sobre el proyecto importado y en la opción ***run as*** seleccionar ***spring boot.***

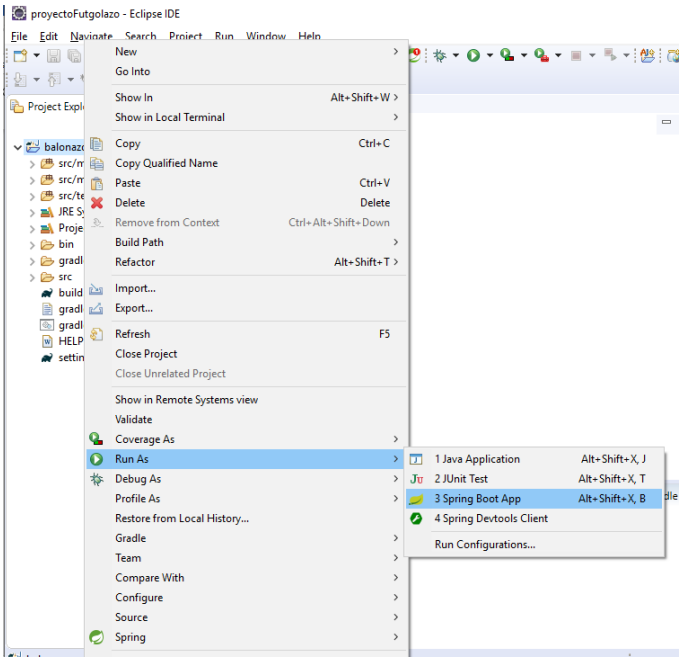


Figura 25. Ejecutar proyecto spring.
Elaborado por: El investigador.

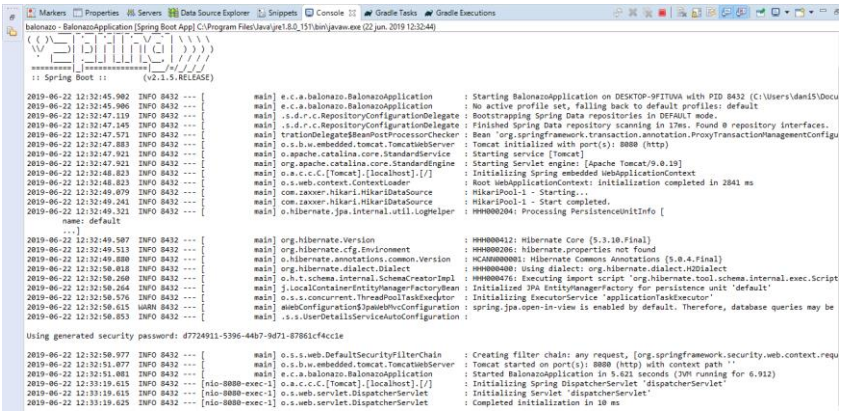


Figura 26. Ejecución inicial del proyecto spring.
Elaborado por: El investigador.

- **Configurar archivo de propiedades yaml de Spring.**

Spring permite configurar diferentes perfiles de ejecución, declaración de constantes y configuración de la base de datos. El archivo de propiedades se encuentra en el package recursos ***src/main/resources/application.properties.***

1) En primera instancia, cambiar la extensión del archivo properties por un archivo **YAML,** esto permite tener una configuración de propiedades más legible y jerárquica.

2) Para el presente proyecto se ha establecido 2 perfiles (desarrollo e integración), además de establecer la configuración de la base de datos postgres.

*application.yml

```yaml
spring:
   datasource:
      url: jdbc:postgresql://localhost:5432/balonazo
      username: postgres
      password: root

   jpa:
      properties:
         hibernate:
            dialect: org.hibernate.dialect.PostgreSQLDialect
            jdbc:
               lob:
                  non_contextual_creation: true
      hibernate:
         ddl-auto: update

logging:
    level:
        root: INFO

---

spring:
   profiles: int
   datasource:
      url: jdbc:postgresql://balonazo-db:5432/balonazo
      username: postgres
      password: postgres
   jpa:
      hibernate:
         ddl-auto: update
```

Figura 27. Configuración inicial Application.yml
Elaborado por: El investigador.

Los archivos YAML interpretan las propiedades configuradas por niveles en base a tabulaciones. En el archivo de configuración anterior se encuentra:

spring: Nivel 1 de configuración. Atributo que establece el bloque de configuración del ambiente spring.

datasource: Bloque de configuraciones para la conexión con la base de datos.

jpa: Bloque de configuración de Java Persistence API.

non_contextual_creation: Al establecer el valor ***true*** se creará el modelo de tablas desde spring con Spring Data.

ddl-auto: Al establecer el valor ***update*** las tablas y el resto de los elementos de la base de datos se actualizan al ejecutar la aplicación, de esta manera la base de datos va creciendo a la par con el desarrollo y los datos persisten.

---: Los 3 guiones permiten dividir los perfiles dentro de la configuración spring.

profiles: Permite determinar el perfil al que pertenece un bloque de configuraciones. Establecer el perfil ***int*** para las configuraciones en el ambiente de integración.

Finalmente, crear la base de datos en postgres con pgAdmin.

3) En PgAdmin III, dar clic derecho sobre ***Databases*** y elegir la opción ***new database..***

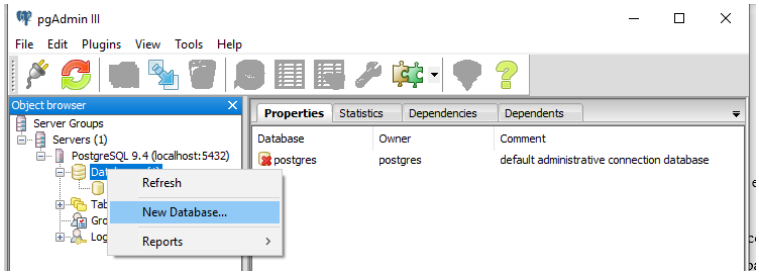


Figura 28. Opciones para creación de base de datos postgres.
Elaborado por: El investigador.

4) Establecer como propietario ***postgres*** y el nombre **balonazo.** Dar clic en ***ok*** y se creará la base de datos.

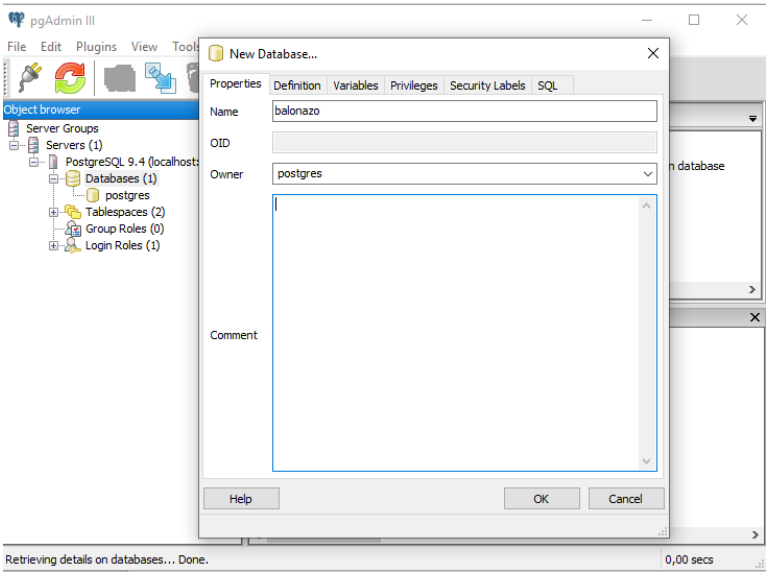


Figura 29. Creación de base de datos en postgres.
Elaborado por: El investigador.

- **Construcción de scripts docker**

Actividades:

- **Instalar docker**

Se puede descargar el instalador de docker para Windows desde su página oficial.

1) Ejecutar el instalador de docker para Windows.

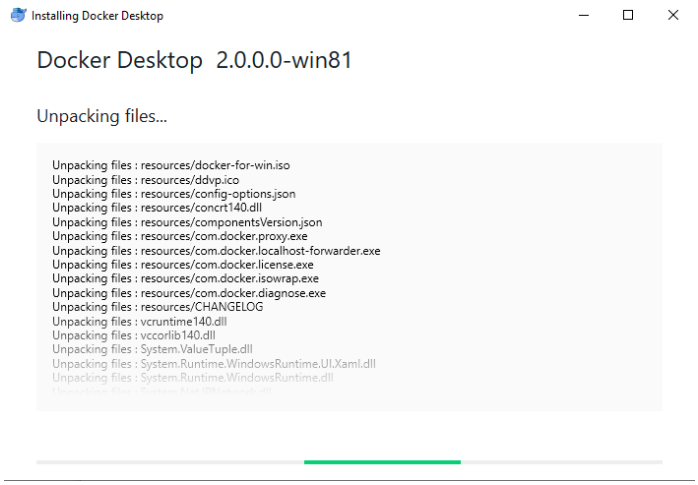


Figura 30. Instalación de docker en Windows.
Elaborado por: El investigador.

2) Después de la instalación, docker se inicia automáticamente.

Figura 31. Iniciar docker.
Elaborado por: El investigador.

- **Desarrollar scripts docker**

Usar docker en los proyectos de desarrollo tiene varias ventajas, como por ejemplo la portabilidad, el aislamiento entre proyectos, la administración, entre otros.

La herramienta docker compose permite construir scripts para levantar contenedores docker en forma de servicios de manera sencilla en cualquier dispositivo que tenga instalado docker.

1) Crear archivo Dockerfile sin extensión en la raíz del proyecto.

El archivo Dockerfile ejecuta una serie de instrucciones durante la creación del contenedor docker.

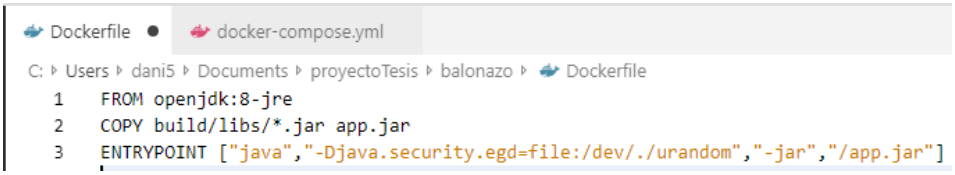


Figura 32. Archivo Dockerfile.
Elaborado por: El investigador.

En el archivo de configuración docker de la figura 32 se encuentra:

Línea 1: Instrucción ***FROM*** que indica la imagen a partir de la cual se creará el servicio y el contenedor docker.

Línea 2: Instrucción para copiar todas las librerías creadas después del ***build*** en el archivo ***app.jar*** que contiene todo el proyecto spring compilado.

Línea 3: Configuración recomendada por Docker para la ejecución de proyectos spring optimizando recursos.

2) Crear el archivo docker-compose.yml

Figura 33. Configuración docker-compose.yml
Elaborado por: El investigador.

En el script docker de la figura 33 se especifica:

version: La versión de docker mínima que debe estar instalada en el ordenador.

volumes: Volúmenes de datos que se pueden ubicar fuera de los contenedores.

services: Bloque de servicios a crear. Los contenedores se crean a manera de servicios y por tanto disponen de una dirección ip.

image: Instrucción para especificar la imagen del servicio a crearse. Las imágenes se encuentran en los repositorios docker.

balonazo-db y balonazo-back: Nombre de los contenedores. Los contendores se pueden comunicar entre sí con este nombre.

environment: Instrucción para establecer el ambiente de ejecución del servicio, determinando parámetros para su ejecución.

restart: Establecer el valor ***always*** para indicar que el contendor se debe volver a crear cuando se vuelva a ejecutar los comandos docker.

ports: Instrucción para indicar el puerto externo e interno del contenedor separados por dos puntos.

networks: Instrucción para indicar la red local que van a manejar los servicios y puedan tener conexión entre ellos.

depends_on: Instrucción para indicar si el servicio depende de otro y esperar a la creación del servicio antes de ejecutarse.

3) Ejecutar script docker.

Ejecutar el script docker con el comando ***docker-compose up*** en la consola de comandos desde la ubicación del archivo**.**

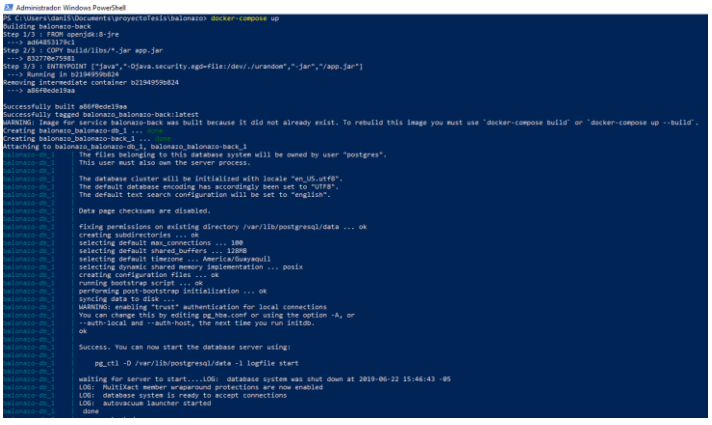


Figura 34. Ejecutar script docker.
Elaborado por: El investigador.

- **Sprint 2**

- **Configuración Spring Security**

Actividades:

- **Crear archivo de configuración Spring Security**

Spring security es uno de los subproyectos de spring para el manejo de seguridades y se puede bloquear o permitir el acceso a servicios desde un archivo de configuración.

1) Dar clic derecho sobre el proyecto y seleccionar la opción ***new package***. Crear un package de configuración.

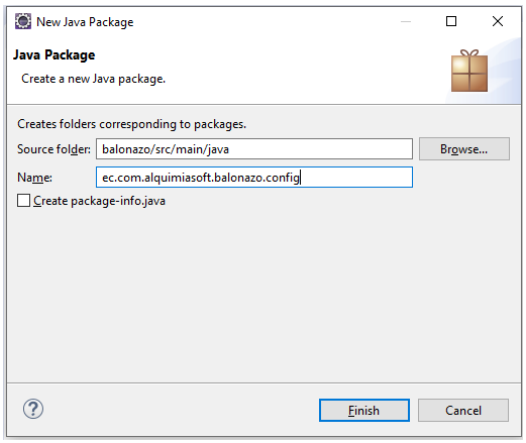


Figura 35. Crear package config.
Elaborado por: El investigador.

2) Crear archivo de configuración

```
SecurityConfiguration.java
1  package ec.com.alquimiasoft.balonazo.config;
2
3  import org.springframework.boot.context.properties.EnableConfigurationProperties;
4  import org.springframework.context.annotation.Configuration;
5  import org.springframework.security.config.annotation.web.builders.HttpSecurity;
6  import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
7  import org.springframework.security.config.http.SessionCreationPolicy;
8
9  @Configuration
10 @EnableConfigurationProperties
11 public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
12
13     @Override
14     protected void configure(HttpSecurity http) throws Exception {
15
16         http
17         .csrf()
18             .disable()
19         .authorizeRequests().antMatchers("/api/**").permitAll()
20          .anyRequest()
21          .authenticated()
22         .and()
23             .httpBasic()
24         .and()
25             .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
26     }
27
28 }
29
```

Figura 36. Configuración spring security.
Elaborado por: El investigador.

El modo de funcionamiento de spring security es a manera de capas de seguridad y en el archivo anterior se encuentra:

@Configuration: Anotación de spring para indicar que la clase es un archivo de configuración.

csrf().disable(): Instrucción para deshabilitar la falsificación de peticiones en sitios cruzados debido a que no se utilizará ningún token de csrf, en su defecto se utilizará tokens con jwt.

authorizeRequests().antMatchers("/api/").permitAll():** Instrucción para permitir el ingreso del tráfico a las peticiones cuyo uri contenga *"/api/"*.

- **Implementación JWT**

Una forma de agregar más seguridad a las peticiones http es el uso de tokens en la cabecera de los request. Para el presente proyecto se ha implementado el uso de JWT que permite agregar atributos extras en el json a codificar, además de permitir agregarle tiempo de expiración y una clave secreta como firma digital que se encuentra solo en Back End.

1) Implementar la librería JWT en el archivo ***build.gradle*** ubicado la raíz del proyecto.

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation('com.auth0:java-jwt:3.4.1')
    compileOnly 'org.projectlombok:lombok'
```

Figura 37. Implementar librería JWT.
Elaborado por: El investigador.

Añadir la librería ***jwt*** del paquete ***auth0*** en la sección de dependencias.

2) Agregar parámetros para jwt en Application.yml

En el archivo Application.yml es posible establecer constantes que se pueden usar en cualquier parte del proyecto, en este caso los parámetros para generar el token JWT.

```yaml
balonazo.security:
   tokenExpirationTime: 14400000
   tokenIssuer: balonazo.com.ec
   tokenSigningKey: D:XR+wj-E}fn7v:XRaei0nnr-ND\\=B+zX0Ui+q}}wkh%m
   refreshTokenExpTime: 31536000

spring:
```

Figura 38. Parámetros de JWT token.
Elaborado por: El investigador.

Dentro de los parámetros de jwt se encuentra:

tokenExpirationTime: Tiempo de vida del ***token*** JWT expresado en milisegundos.

tokenIssuer: editor del código.

tokenSigningKey: clave secreta para firma digital.

refreshTokenExpTime: Tiempo de vida del ***refresh token*** expresado en milisegundos.

3) Crear archivo de configuración jwt.

```java
package ec.com.alquimiasoft.balonazo.config;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Configuration
@ConfigurationProperties(prefix = "balonazo.security")
public class JwtSettings {
    /**
     * {@link JwtToken} will expire after this time.
     */
    private Integer tokenExpirationTime;

    /**
     * Token issuer.
     */
    private String tokenIssuer;

    /**
     * Key is used to sign {@link JwtToken}.
     */
    private String tokenSigningKey;

    /**
     * {@link JwtToken} can be refreshed during this timeframe.
     */
    private Integer refreshTokenExpTime;

    /************GETTERS Y SETTERS*****************/
    public Integer getRefreshTokenExpTime() {
        return refreshTokenExpTime;
    }

    public void setRefreshTokenExpTime(Integer refreshTokenExpTime) {
        this.refreshTokenExpTime = refreshTokenExpTime;
    }

    public Integer getTokenExpirationTime() {
        return tokenExpirationTime;
    }
```

Figura 39. Archivo de configuración JWT.
Elaborado por: El investigador.

En la figura 39, se especifica una clase de configuración con los parámetros de jwt en modo de variables y poder acceder a ellos por medio de los métodos ***getters*** y ***setters.***

Es necesario crear un servicio que se encargue de la creación de los tokens JWT.

4) Crear un package para servicios dando clic derecho sobre el proyecto.

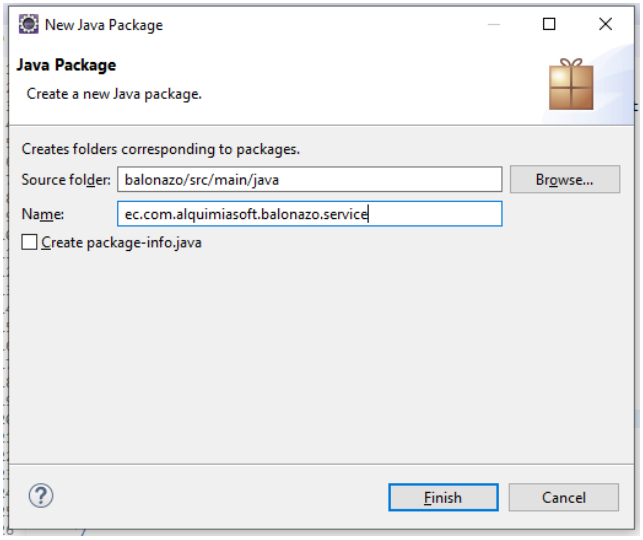


Figura 40. Crear package service.
Elaborado por: El investigador.

5) Crear el archivo ***TokenService*** dentro del ***package*** creado en el punto 3.

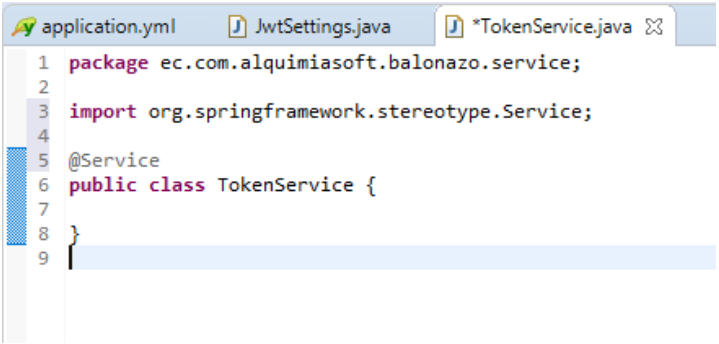


Figura 41. Crear archivo token service.
Elaborado por: El investigador.

Es necesario añadir la anotación ***@Service*** para indicarle a Spring que la clase es un servicio.

6) Crear función para generar token.

```java
@Service
public class TokenService {

    @Autowired
    private JwtSettings jwtSettings;

    public String createToken(String userId, boolean isRefreshToken) {
        try {
            Algorithm algorithm = Algorithm.HMAC256(jwtSettings.getTokenSigningKey());
            return JWT.create().withClaim("userId", userId.toString()).withClaim("createdAt", new Date())
                    .withExpiresAt(
                            new Date(System.currentTimeMillis() + (isRefreshToken ? jwtSettings.getRefreshTokenExpTime()
                                    : jwtSettings.getTokenExpirationTime())))
                    .sign(algorithm);

        } catch (JWTCreationException exception) {
        }
        return null;
    }

}
```

Figura 42. Función generar token.
Elaborado por: El investigador.

En la función para generar el token jwt de la figura 42 se encuentra:

Línea 14: Inyección de dependencia con el archivo de configuración JWT utilizando la anotación **@Autowired**.

Línea 22: Crear la cabecera del token con el algoritmo ***HMAC256*** y con la clave secreta establecida en los parámetros.

Línea 23: Retorno de la cadena token concatenando el ***userId,*** la fecha de creación.

Línea 25: Código para agregar a la cadena la fecha de expiración, condicionado por si se va a generar un token o un refresh token, por el motivo que el refresh token tiene un tiempo de vida mayor.

7) Crear funciones para validar el token.

Es necesario validar un token que viene incluido en el header para verificar que las peticiones sean de un usuario válido.

```java
    public String getUserIdFromToken(String token) {
        try {
            Algorithm algorithm = Algorithm.HMAC256(jwtSettings.getTokenSigningKey());
            JWTVerifier verifier = JWT.require(algorithm).build();
            DecodedJWT jwt = verifier.verify(token);
            return jwt.getClaim("userId").asString();
        } catch (JWTVerificationException exception) {
            return null;
        }
    }

    public boolean isTokenValid(String token) {
        String userId = this.getUserIdFromToken(token);
        return userId != null;
    }
```

Figura 43. Funciones para verificar token.
Elaborado por: El investigador.

En las funciones para verificar un token de la figura 43 se encuentran:

Líneas 39 y 40: Código para generar el algoritmo con la clave secreta y se construye el header del token JWT, esto equivale a crear un token en la función de generación del token.

Línea 41: Código para decodificar el token JWT con la clase ***JWTVerifier*** creado en la línea 40.

Línea 42: Código para retornar el ***userId*** que se encuentra encriptado en el token. En caso de no existir el ***userId*** se retorna ***null.***

Líneas 49 y 50: Código para obtener el ***userId*** del token y retorna ***true*** si el id del usuario no es nulo, caso contrario se retorna ***false.***

8) Crear package para los archivos DTO.

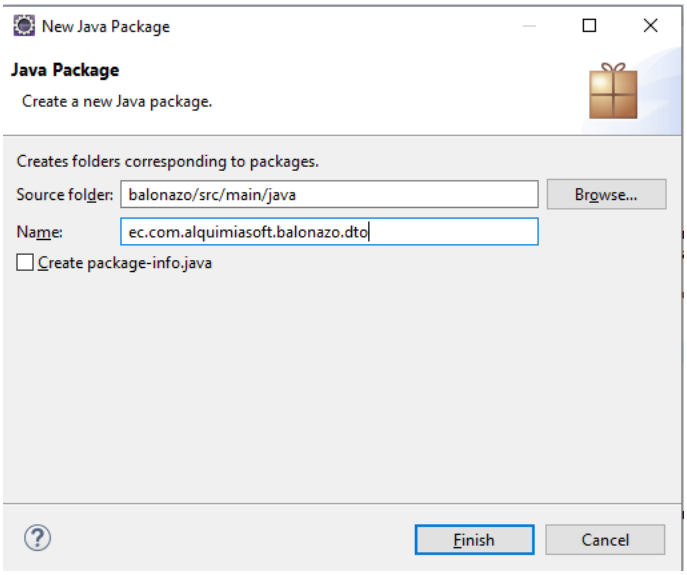


Figura 44. Crear package dto.
Elaborado por: El investigador.

9) Crear Session DTO para retornar tokens.

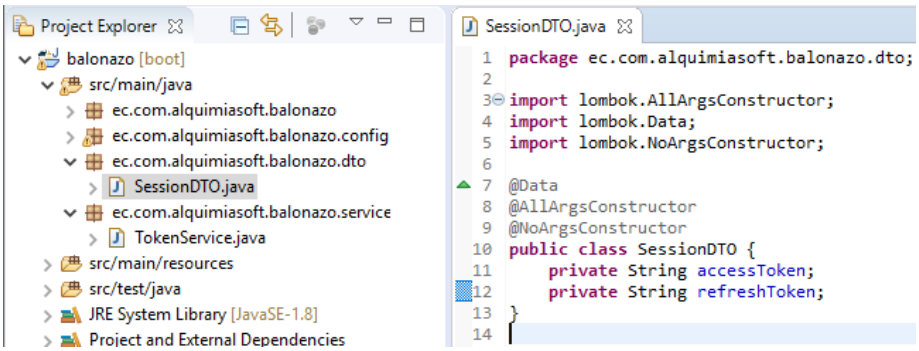


Figura 45. Crear archivo SessionDTO.
Elaborado por: El investigador.

En el archivo SessionDTO de la figura 45 se encuentra:

@Data: Anotación de Loombok para crear los métodos ***getters*** y ***setters*** de manera implícita.

@AllArgsConstructor: Anotación para crear constructor de manera implícita con todas las variables declaradas en la clase.

@NoArgsConstructor: Anotación para crear un constructor sin parámetros.

10) Crear función para generar un nuevo token al validar un refresh token.

Esta función permitirá crear un nuevo access token validando un refresh token, si el refresh token ya no tiene validez, la función retornará una excepción.

```
public ResponseEntity<SessionDTO> getNewAccessToken(SessionDTO sessionDTO, boolean isAdmin) {

    HttpHeaders httpHeaders = new HttpHeaders();

    if (sessionDTO.getRefreshToken() == null) {
        return new ResponseEntity<>(new SessionDTO(null, null), httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
    }

    String userIdFromToken = getUserIdFromToken(sessionDTO.getRefreshToken());
    if (userIdFromToken == null) {
        return new ResponseEntity<>(new SessionDTO(null, null), httpHeaders, HttpStatus.UPGRADE_REQUIRED);
    }

    String cad[] = userIdFromToken.split(",");

    String userId = cad[0];

    if (userId == null) {
        return new ResponseEntity<>(new SessionDTO(null, null), httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
    }

    if (!jwtSettings.getTokenIssuer().contains(cad[1])) {
        return new ResponseEntity<>(new SessionDTO(null, null), httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
    }

    return new ResponseEntity<>(new SessionDTO(createToken(userId, false), null), httpHeaders, HttpStatus.OK);

}
```

Figura 46. Función obtener nuevo access token.
Elaborado por: El investigador.

11) Crear archivo de filtros para peticiones http en el paquete config.

application.yml | JwtSettings.java | TokenService.java | JWTFilter.java

```
1  package ec.com.alquimiasoft.balonazo.config;
2
3  import java.io.IOException;
12
13 @Configuration
14 public class JWTFilter extends GenericFilterBean {
15
16     @Override
17     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
18             throws IOException, ServletException {
19         // TODO Auto-generated method stub
20
21     }
22
23 }
24
```

Figura 47. Crear archivo JWT Filter.
Elaborado por: El investigador.

La clase debe extender e implementar los métodos de la clase abstracta ***GenericFilterBean.***

12) Crear función para filtrar las peticiones que no requieren token, por ejemplo, el login.

```
@Configuration
public class JWTFilter extends GenericFilterBean {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        // TODO Auto-generated method stub

    }

    public boolean allowRequestWithoutToken(HttpServletRequest request) {
        return true;
    }

}
```

Figura 48. Crear función peticiones sin token.
Elaborado por: El investigador.

13) Implementar el filtro de peticiones http en el método doFilter.

```
@Autowired
private TokenService tokenService;

@Override
public void doFilter(ServletRequest req, ServletResponse res, FilterChain filterChain)
        throws IOException, ServletException {

    HttpServletRequest request = (HttpServletRequest) req;
    HttpServletResponse response = (HttpServletResponse) res;
    String token = request.getHeader("Authorization");

    if ("OPTIONS".equalsIgnoreCase(request.getMethod())) {
        response.sendError(HttpServletResponse.SC_OK, "success");
        return;
    }

    if (allowRequestWithoutToken(request)) {
        response.setStatus(HttpServletResponse.SC_OK);
    } else {
        if (token == null || !tokenService.isTokenValid(token)) {
            response.sendError(HttpServletResponse.SC_UNAUTHORIZED);
            return;
        }
    }

    filterChain.doFilter(req, res);
}
```

Figura 49. Implementar método doFilter.
Elaborado por: El investigador.

En la implementación del método ***doFilter*** de la figura 49 se encuentra:

Líneas 28 y 29: Código para realizar un casting de los ***servlets request*** y ***response*** a ***servlets http*** para obtener la información de la solicitud como, por ejemplo, POST, GET, HEAD, DELETE, etc.

Línea 30: Código para obtener el ***Authorization token*** que viene en el ***head*** de la solicitud.

Línea 37: Código para verificar si la solicitud pertenece a una url sin seguridad por token.

Línea 40: Código para verificar que el token que viene en la solicitud sea válido para procesar la solicitud, caso contrario retorna una excepción 401.

- **Login y registro**

Actividades:

- **Crear entidades para Usuarios y Administradores.**

Spring framework permite crear entidades y al ejecutar el proyecto, automáticamente se crearán las tablas y relaciones en la base de datos con Spring Data.

1) Crear package para entidades.

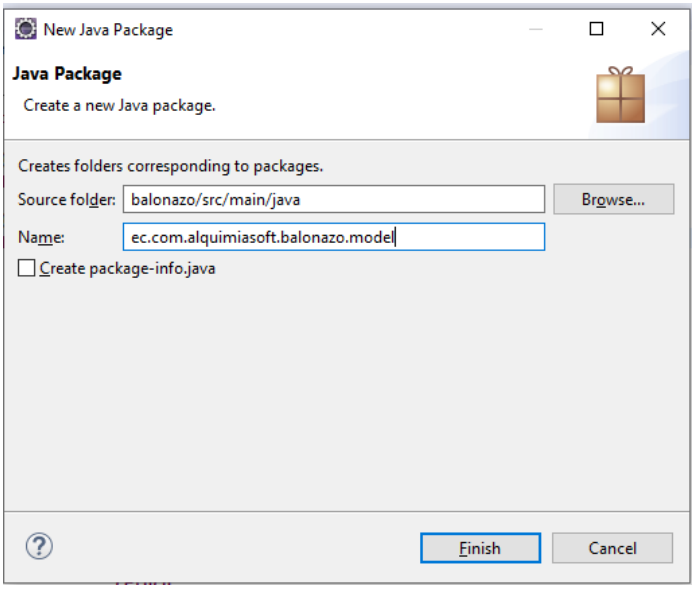


Figura 50. Crear package model.
Elaborado por: El investigador.

Dada la lógica de negocio, cada usuario podrá elegir ser hincha de un equipo y cada equipo será diferenciado por si es nacional o internacional, para ello es necesario crear entidades extras.

2) Crear entidad Catalog.

Catalog.java

```java
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;

import org.hibernate.annotations.ColumnDefault;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "blz_catalog")
@NoArgsConstructor
@AllArgsConstructor
@Data
public class Catalog implements Serializable {

    private static final long serialVersionUID = 6563035795524732355L;

    @Id
    @SequenceGenerator(name="pk_sequence_catalog",sequenceName="blz_catalog_seq", allocationSize=1)
    @GeneratedValue(strategy=GenerationType.SEQUENCE,generator="pk_sequence_catalog")
    private Long id;

    @NotNull
    private LocalDate creationDate;

    @ColumnDefault("'1'")
    private Boolean active;

    @NotNull
    private String tableName;

    @NotNull
    private String description;
}
```

Figura 51. Crear Entidad Catalog.
Elaborado por: El investigador.

Para la creación de cualquier entidad se debe implementar:

@Entity: Anotación que le dice a Spring que la clase es una entidad y la pueda mapear con la base de datos.

@Table: Anotación que permite agregar propiedades a la tabla al momento de crearse, por ejemplo, el nombre, agregar índices, entre otros.

Serializable: Interfaz que permite agregar un serializable ID a cada clase para que Java lo pueda mapear.

En la figura 51 anterior se puede encontrar:

@id: Anotación para identificar una variable como el ***id*** de la tabla.

@SequenceGenerator: Anotación para generar una secuencia en la base de datos.

@GeneratedValue: Anotación para asignar un secuencial a una columna de la tabla.

@NotNull: Anotación que le indica a Spring que cree el campo como requerido en la tabla.

@ColumnDefault: Anotación que permite asignar valores por defecto en una columna de la tabla.

3) Crear entidad Catalog Detail.

Catalog.java | CatalogDetail.java

```
14 import javax.validation.constraints.NotNull;
15
16 import org.hibernate.annotations.ColumnDefault;
17
18 import lombok.AllArgsConstructor;
19 import lombok.Data;
20 import lombok.NoArgsConstructor;
21
22 @Entity
23 @Table(name = "blz_catalog_detail")
24 @NoArgsConstructor
25 @AllArgsConstructor
26 @Data
27 public class CatalogDetail implements Serializable {
28
29     private static final long serialVersionUID = 5786386760984637765L;
30
31     @Id
32     @SequenceGenerator(name="pk_sequence_catalogdet",sequenceName="blz_catalog_detail_seq", allocationSize=1)
33     @GeneratedValue(strategy=GenerationType.SEQUENCE,generator="pk_sequence_catalogdet")
34     private Long id;
35
36     @NotNull
37     private LocalDate creationDate;
38
39     @ColumnDefault("'1'")
40     private Boolean active;
41
42     @NotNull
43     private String description;
44
45     @NotNull
46     @ManyToOne
47     @JoinColumn(name = "catalog_id")
48     private Catalog catalog;
49
50 }
```

Figura 52. Crear Entidad Catalog Detail.
Elaborado por: El investigador.

En la entidad Catalog Detail de la figura 52 se encuentra:

@ManyToOne: Anotación que permite relacionar entidades, en este caso se establece una relación de varios a uno con la entidad Catalog.

@JoinColumn: Anotación que indica la columna con la que se establece la relación. Con JPA se puede establecer una relación directa con la entidad, pero a nivel de la base de datos se relacionan los ***Ids.***

4) Crear entidad Team.

Team.java

```java
@NoArgsConstructor
@AllArgsConstructor
@Data
public class Team implements Serializable {

    private static final long serialVersionUID = 1737664294012984678L;

    @Id
    @SequenceGenerator(name="pk_sequence_team",sequenceName="blz_team_seq", allocationSize=1)
    @GeneratedValue(strategy=GenerationType.SEQUENCE,generator="pk_sequence_team")
    private Long id;

    @NotNull
    private LocalDate creationDate;

    @ColumnDefault("'1'")
    private Boolean active;

    @NotNull
    private String description;

    @NotNull
    @ManyToOne
    @JoinColumn(name = "team_type_id")
    private CatalogDetail teamtype;

    private String logoUrl;

    private String petUrl;

    @NotNull
    @ManyToOne
    @JoinColumn(name = "league_serie_id")
    private CatalogDetail leagueSerieId;

    @NotNull
    @ManyToOne
    @JoinColumn(name = "country_id")
    private CatalogDetail countryId;
}
```

Figura 53. Crear Entidad Team.
Elaborado por: El investigador.

La lógica de negocio determina que cada equipo de fútbol debe ser identificado por un tipo (nacional o internacional), liga o serie (Serie A, Serie B, etc.), y el país de origen.

5) Crear entidad para Usuarios.

La lógica de negocio dispone que aparte de los datos comunes de un usuario se agregue:

goldBall: Columna donde se almacena la moneda oficial de la plataforma.

origin: Columna para identificar si el usuario fue creado por medio de formulario o por acceso con Facebook.

supportedTeamId: Columna que almacena el ***id*** del equipo de fútbol que el usuario es hincha.

La lógica de negocio establece que existan usuarios administradores para manipular la información que se presenta en el juego, para ello se creará una tabla de roles.

En la figura 54 se muestra la entidad User creada con los aspectos mencionados en el párrafo anterior.

Team.java *User.java

```
24
25  @Entity
26  @Table(name = "blz_user")
27  @Data
28  @NoArgsConstructor
29  @AllArgsConstructor
30  public class User implements Serializable {
31
32      private static final long serialVersionUID = -4958588853471648123L;
33
34      @Id
35      @SequenceGenerator(name="pk_sequence",sequenceName="blz_user_seq", allocationSize=1)
36      @GeneratedValue(strategy=GenerationType.SEQUENCE,generator="pk_sequence")
37      private Long id;
38      @NotNull
39      private String username;
40      @NotNull
41      @Email
42      @Column(unique = true)
43      private String email;
44      @NotNull
45      private String password;
46      @NotNull
47      private LocalDate creationDate;
48      @NotNull
49      @ColumnDefault("'1'")
50      private Boolean active;
51
52      private Integer goldBalls;
53
54      @Size(max=50)
55      private String firstName;
56
57      private String lastName;
58      private LocalDate birthDate;
59      private String city;
60
61      @ManyToOne
62      @JoinColumn(name = "supported_team_id")
63      private Team supportedTeamId;
64
65      private String phone;
66
67      private Integer origin;
68  }
```

Figura 54. Crear Entidad User.
Elaborado por: El investigador.

6) Crear entidad para Role Administrador.

Role.java

```java
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;

import org.hibernate.annotations.ColumnDefault;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "blz_role")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Role implements Serializable {

    private static final long serialVersionUID = 1525944188286301297L;

    @Id
    @SequenceGenerator(name="pk_sequence_role",sequenceName="blz_role_seq", allocationSize=1)
    @GeneratedValue(strategy=GenerationType.SEQUENCE,generator="pk_sequence_role")
    private Long id;

    @NotNull
    private LocalDate creationDate;

    @NotNull
    @ColumnDefault("'1'")
    private Boolean active;

    @NotNull
    private String roleName;
}
```

Figura 55. Crear Entidad Role.
Elaborado por: El investigador.

7) Crear entidad Admin User.

Role.java | AdminUser.java

```java
@Table(name = "blz_admin_user")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class AdminUser implements Serializable {

    private static final long serialVersionUID = 1498127558841644015L;

    @Id
    @SequenceGenerator(name="pk_sequence_adm_user",sequenceName="blz_admin_user_seq", allocationSize=1)
    @GeneratedValue(strategy=GenerationType.SEQUENCE,generator="pk_sequence_adm_user")
    private Long id;

    @NotNull
    private LocalDate creationDate;
    @NotNull
    @ColumnDefault("'1'")
    private Boolean active;

    @NotNull
    @Column(unique = true)
    private String username;

    @NotNull
    @Email
    @Column(unique = true)
    private String email;

    @NotNull
    private String password;

    @NotNull
    private Integer type;

    private String firstName;
    private String lastName;
    private String identificationType;
    private String identification;
}
```

Figura 56. Crear Entidad AdminUser.
Elaborado por: El investigador.

8) Crear entidad para relación varios a varios entre Administrador y Role.

Debido a que existe una relación de varios a varios se creará una entidad que genera la tabla intermedia con una ***primary key*** compuesta por los ***Ids*** de cada tabla. Para la creación de claves compuestas en JPA se debe crear una clase con las columnas que conforman la ***primary key.***

AdminUserRoleId.java

```
1  package ec.com.alquimiasoft.balonazo.model;
2
3  import java.io.Serializable;
4
5  import javax.persistence.Embeddable;
6  import javax.persistence.JoinColumn;
7  import javax.persistence.ManyToOne;
8  import javax.validation.constraints.NotNull;
9
10 import lombok.AllArgsConstructor;
11 import lombok.Data;
12 import lombok.NoArgsConstructor;
13
14 @Embeddable
15 @NoArgsConstructor
16 @AllArgsConstructor
17 @Data
18 public class AdminUserRoleId implements Serializable {
19
20     private static final long serialVersionUID = 651129893768543116L;
21
22     @NotNull
23     @ManyToOne
24     @JoinColumn(name = "role_id")
25     private Role role;
26
27     @NotNull
28     @ManyToOne
29     @JoinColumn(name = "admin_user_id")
30     private AdminUser adminUser;
31 }
32
```

Figura 57. Crear clave compuesta AdminUserRoleId.
Elaborado por: El investigador.

Para la creación de la clave compuesta se implementa:

@Embeddable: Anotación que indica que la clase actual se puede embeber en otra, con el fin de crear los campos que componen la clave primaria en la tabla. Al no tener la notación **@Entity** la clase no es tratada como entidad y no se mapea con la base de datos.

Una vez creada la clase para la clave compuesta, crear la entidad que forma la tabla intermedia en la relación Administrador y Role.

AdminUserRoleId.java | AdminUserRole.java

```java
package ec.com.alquimiasoft.balonazo.model;

import java.io.Serializable;
import java.time.LocalDate;

import javax.persistence.EmbeddedId;
import javax.persistence.Entity;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;

import org.hibernate.annotations.ColumnDefault;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "blz_admin_user_role")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class AdminUserRole implements Serializable {

    private static final long serialVersionUID = 3596452520516958538L;

    @EmbeddedId
    private AdminUserRoleId pKey;

    @NotNull
    private LocalDate creationDate;

    @NotNull
    @ColumnDefault("'1'")
    private Boolean active;
}
```

Figura 58. Crear Entidad AdminUserRole.
Elaborado por: El investigador.

En la entidad AdminUserRole se encuentra:

@EmbeddedId: Anotación que embebe la clase con las columnas que forman la clave compuesta.

También se puede crear un campo relacionando la tabla intermedia con uno de sus extremos en cualquiera de las entidades, con el propósito de traer los datos de 2 tablas a la vez.

```java
    @OneToMany(mappedBy = "pKey.adminUser")
    List<AdminUserRole> adminUserRoleList;
}
```

Figura 59. Añadir relación lista de roles en AdminUser.
Elaborado por: El investigador.

Para la relación con la tabla intermedia se implementa:

mappedBy: Atributo que indica el campo o los campos de la clave compuesta que se mapean en la entidad. Este campo no existe explícitamente en la tabla que se crea en la base de datos y solo es accesible desde Java gracias a JPA.

9) Ejecutar proyecto spring para que se creen las tablas en la base de datos.

Al ejecutar el proyecto, comprobar en la base de datos que se han creado las tablas según las entidades declaradas en Spring. El resultado se muestra en la figura 60.

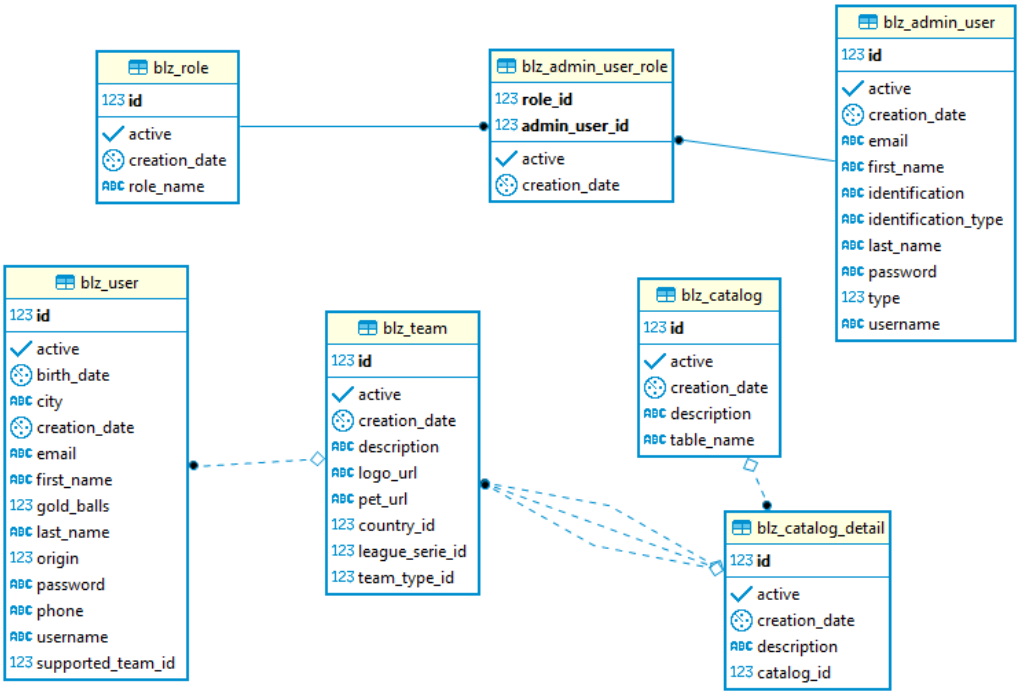


Figura 60. Modelo base de datos Usuarios y Administradores.
Elaborado por: El investigador.

- **Crear Servicio de registro para Usuarios y Administradores.**

Al usar el patrón de diseño MVC se debe crear las capas necesarias para el manejo de datos (***repository***)**,** la implementación de servicios (***service***) y la capa de presentación (***controller***). Spring recomienda exponer interfaces en lugar de las implementaciones para no comprometer el código fuente y tener todas las ventajas del uso de interfaces. (***operations***).

1) Crear package repository.

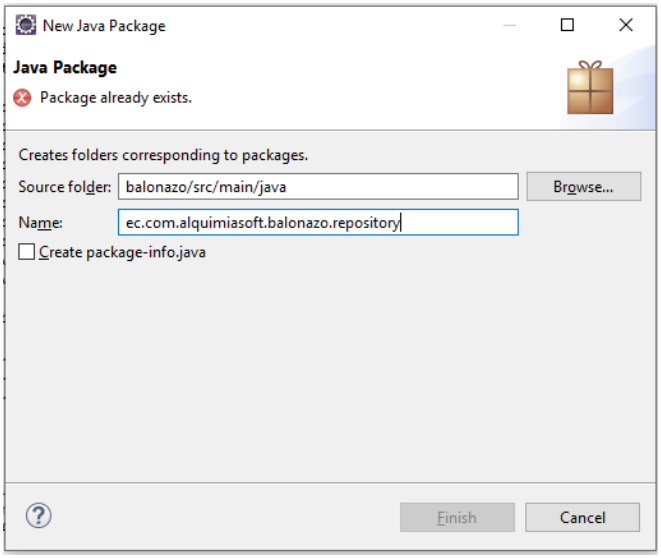


Figura 61. Crear package repository.
Elaborado por: El investigador.

2) Crear la interfaz UserRepository para la manipulación de la base de datos.

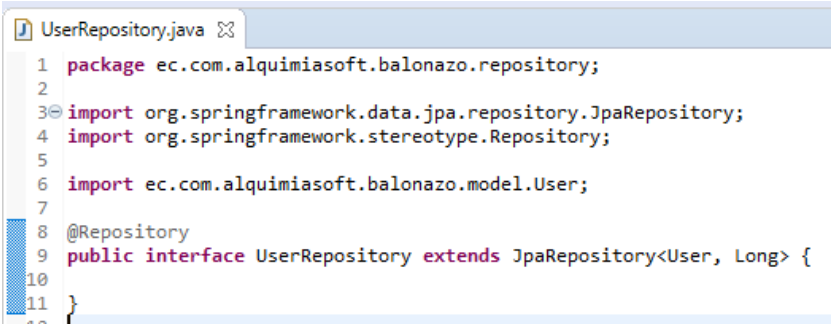


Figura 62. Crear interfaz UserRepository.
Elaborado por: El investigador.

Toda interfaz de tipo repositorio debe implementar:

@Repository: Anotación que indica que la interfaz es un repositorio.

JpaRepository: Clase que hereda el repositorio para manipular la información de la base de datos. La clase JpaRepository necesita que se le especifique un ***hash*** con el tipo de Objeto que retorna y el tipo de dato de su ***ID.***

Los repositorios implementan automáticamente CRUD de la data, además de métodos de consulta de la información en la base datos.

3) Crear package operations.

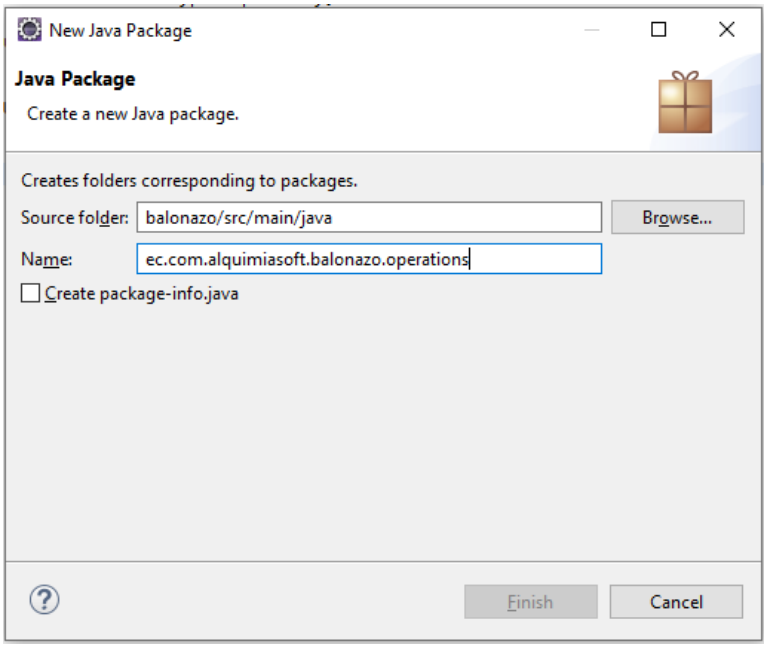


Figura 63. Crear package operations.
Elaborado por: El investigador.

4) Crear la interfaz UserOperations.

En la interfaz se declaran los métodos y funciones que se van a implementar y es recomendable usar DTOs en lugar de las entidades para manejar solo la data necesaria para el proceso. Crear la clase UserDTO.

La clase UserDTO creada en la figura 64, se utilizará para establecer los parámetros que necesita la consulta y también se utilizará como el objeto JSON que retornará el servicio luego de ejecutar la petición.

Se crea la interfaz UserOperations como se muestra en la figura 65 y se declara los métodos a implementar en el servicio.

En la interfaz UserOperations se encuentra:

ResponseEntity: Es un modelo para servicios REST donde se puede retornar una petición con ***head***, ***body*** y un código http; valores que ***Front End*** los puede interpretar y procesar.
BindingResult: Interfaz que captura las validaciones de los parámetros requeridos en una petición POST.

UserRepository.java | UserDTO.java

```java
package ec.com.alquimiasoft.balonazo.dto;

import javax.validation.constraints.Email;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class UserDTO {
    @NotNull(message = "username can't be null")
    @NotEmpty(message = "username can't empty")
    private String username;

    @NotNull(message = "email can't be null")
    @NotEmpty(message = "email can't be empty")
    @Email
    private String email;

    @NotNull(message = "password can't be null")
    @NotEmpty(message = "password can't be empty")
    private String password;

    private Integer goldCoins;

    @NotNull
    private Integer origin;

    private String message;

    private SessionDTO session;
}
```

Figura 64. Crear UserDTO.
Elaborado por: El investigador.

UserRepository.java | UserOperations.java

```java
package ec.com.alquimiasoft.balonazo.operations;

import org.springframework.http.ResponseEntity;

public interface UserOperations {
    User saveUserData(User user);

    ResponseEntity<UserDTO> saveUser(UserDTO userDTO, BindingResult bindingResult);

    User validateUser(String token);
}
```

Figura 65. Crear interfaz UserOperations.
Elaborado por: El investigador.

5) Crear servicio UserService.

UserService.java

```java
package ec.com.alquimiasoft.balonazo.service;

import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.validation.BindingResult;

import ec.com.alquimiasoft.balonazo.dto.UserDTO;
import ec.com.alquimiasoft.balonazo.model.User;
import ec.com.alquimiasoft.balonazo.operations.UserOperations;

@Service
public class UserService implements UserOperations {

    @Override
    public User saveUserData(User user) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public ResponseEntity<UserDTO> saveUser(UserDTO userDTO, BindingResult bindingResult) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public User validateUser(String token) {
        // TODO Auto-generated method stub
        return null;
    }

}
```

Figura 66. Crear servicio UserService.
Elaborado por: El investigador.

El servicio implementa la interfaz UserOperations, por tanto, la clase requiere que se implementen los métodos declarados en la interfaz.

6) Implementar método de registro de usuarios.

En primera instancia implementar el método para guardar la data en la base de datos.

```java
    @Override
    public User saveUserData(User user) {
        return userRepository.saveAndFlush(user);
    }
```

Figura 67. Implementar método saveUserData.
Elaborado por: El investigador.

UserRepository se encarga de guardar la información mediante el método ***saveAndFlush,*** que guarda los datos e inmediatamente retorna el objeto que se ingresó en la base de datos.

Implementar el método que valide la información antes de guardarla.

UserService.java

```
66      @Override
67      public ResponseEntity<UserDTO> saveUser(UserDTO userDTO, BindingResult bindingResult) {
68
69          HttpHeaders httpHeaders = new HttpHeaders();
70
71          UserDTO userDto = userDTO;
72          userDto.setMessage(null);
73
74          List<String> errors = null;
75
76          if (bindingResult.hasErrors()) {
77              errors = bindingResult.getAllErrors().stream().map(e -> e.getCode() + "." + e.getDefaultMessage())
78                      .collect(Collectors.toList());
79              userDto.setPassword("");
80              userDto.setMessage(errors.toString());
81              return new ResponseEntity<>(userDto, httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
82          }
83
84          if (userDTO.getOrigin() == UserOriginEnum.BALONAZO.getCode()
85                  && (userDTO.getPassword() == null || userDTO.getPassword().length() > maxPwdLength)) {
86              userDto.setMessage("Longitud del password debe tener un maximo de ".concat(maxPwdLength.toString())
87                      .concat(" caracteres"));
88              return new ResponseEntity<>(userDto, httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
89          }
90
91          errors = validateUniqueEmail(userDTO);
92
93          if (!errors.isEmpty()) {
94              User userOut = findByEmail(userDTO.getEmail());
95              if (userOut != null && userOut.getOrigin() != UserOriginEnum.BALONAZO.getCode()) {
96                  userDto.setOrigin(userOut.getOrigin());
97                  userDto.setMessage("El usuario ya esta registrado con otro canal");
98                  return new ResponseEntity<>(userDto, httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
99              }
100             userDto.setPassword("");
101             userDto.setMessage(errors.toString());
102             return new ResponseEntity<>(userDto, httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
103         }
```

Figura 68. Implementar método saveUser – parte 1.
Elaborado por: El investigador.

En la Fig. 68. que implementa el método saveUser se encuentra:

Línea 76: Código para verificar si la interfaz BindingResult capturó errores de validación en los parámetros de la petición.

Línea 84: Código para verificar el origen del usuario a guardar (Balonazo o Facebook), en el caso de usuario Balonazo se establece un máximo de caracteres para la contraseña.

Línea 91: Código para verificar si el correo electrónico ya ha sido utilizado para registrar otro usuario. En caso de ya existir se retorna un código de error y un mensaje.

Líneas 93 a 102: Código para verificar si el proceso de validación de correo duplicado retornó algún error y retorna un código de error y un mensaje.

```
104
105         if (userDto.getUsername().length() > maxUsernameLength) {
106             userDto.setPassword("");
107             userDto.setMessage("Username length must be less than ".concat(Integer.toString(maxUsernameLength)));
108             return new ResponseEntity<>(userDto, httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
109         }
110
111         User user = new User();
112
113         user.setUsername(userDTO.getUsername().toLowerCase());
114         user.setPassword(passwordEncoder.encode(userDTO.getPassword()));
115         user.setEmail(userDTO.getEmail().toLowerCase());
116         user.setActive(true);
117         user.setCreationDate(LocalDate.now());
118         user.setGoldBalls(Integer.valueOf(0));
119         user.setOrigin(userDTO.getOrigin());
120
121         user = userRepository.saveAndFlush(user);
122
123         userDto.setGoldCoins(Integer.valueOf(0));
124         userDto.setPassword("");
125         userDto.setMessage("user created successfully");
126         userDto.setSession(
127                 new SessionDTO(tokenService.createToken(user.getEmail(), TokenTypeEnum.ACCESS_TOKEN.getCode()),
128                         tokenService.createToken(user.getEmail().concat(",").concat(jwtSettings.getTokenIssuer()),
129                                 TokenTypeEnum.REFRESH_TOKEN.getCode())));
130
131         return new ResponseEntity<>(userDto, httpHeaders,HttpStatus.OK);
132     }
133
```

Figura 69. Implementar método saveUser – parte 2.
Elaborado por: El investigador.

En la Figura 69. que implementa el método saveUser se encuentra:

Líneas 104 a 108: Código para validar que el número de caracteres de username no excede el límite permitido, en caso de hacerlo se retorna un código de error y un mensaje.

Líneas 113 a 119: Código para agregar la data en el objeto User que se guardará en la base de datos.

Línea 121: Código para guardar la data User en la base datos de manera inmediata.

Líneas 126 a 129: Código para generar el access y refresh token del usuario con el email como ***userId*** y lo retorna en el DTO.

7) Crear la interfaz AdminUserRepository.

UserService.java | AdminUserRepository.java

```
1  package ec.com.alquimiasoft.balonazo.repository;
2
3  import org.springframework.data.jpa.repository.JpaRepository;
4  import org.springframework.stereotype.Repository;
5
6  import ec.com.alquimiasoft.balonazo.model.AdminUser;
7
8  @Repository
9  public interface AdminUserRepository extends JpaRepository<AdminUser, Long> {
10     AdminUser findByUsername(String username);
11
12     AdminUser findByEmail(String email);
13 }
```

Figura 70. Crear interfaz AdminUserRepository.
Elaborado por: El investigador.

En la interfaz AdminUserRepository se puede declarar métodos que ejecuten sentencias SQL en la base de datos.

8) Crear la interfaz AdminUserOperations.

En primera instancia crear la clase AdminUserDTO para el manejo de parámetros del servicio.

UserService.java | AdminUserDTO.java

```java
package ec.com.alquimiasoft.balonazo.dto;

import java.util.List;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class AdminUserDTO {
    private Long id;
    private String username;
    private String password;
    private String email;
    private List<RoleUserDTO> roles;
    private Integer identificationType;
    private String identificationNumber;
    private String firstName;
    private String lastName;
    private String message;
    private SessionDTO session;
}
```

Figura 71. Crear clase AdminUserDTO.
Elaborado por: El investigador.

En la interfaz AdminUserOperations declarar los métodos a implementar en el servicio.

AdminUserOperations.java

```java
package ec.com.alquimiasoft.balonazo.operations;

import org.springframework.http.ResponseEntity;

import ec.com.alquimiasoft.balonazo.dto.AdminUserDTO;
import ec.com.alquimiasoft.balonazo.model.AdminUser;

public interface AdminUserOperations {
    ResponseEntity<AdminUserDTO> register(AdminUserDTO adminUserDTO);

    AdminUser getAdminUserByUsername(String username);

    AdminUser getAdminUserByEmail(String email);
}
```

Figura 72. Crear interfaz AdminUserOperations.
Elaborado por: El investigador.

9) Crear servicio AdminUserService.

AdminUserService.java

```java
package ec.com.alquimiasoft.balonazo.service;

import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;

import ec.com.alquimiasoft.balonazo.dto.AdminUserDTO;
import ec.com.alquimiasoft.balonazo.model.AdminUser;
import ec.com.alquimiasoft.balonazo.operations.AdminUserOperations;

@Service
public class AdminUserService implements AdminUserOperations {

    @Override
    public ResponseEntity<AdminUserDTO> register(AdminUserDTO adminUserDTO) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public AdminUser getAdminUserByUsername(String username) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public AdminUser getAdminUserByEmail(String email) {
        // TODO Auto-generated method stub
        return null;
    }

}
```

Figura 73. Crear servicio AdminUserService.
Elaborado por: El investigador.

10) Implementar método de registro de administradores.

AdminUserService.java | AdminUserDTO.java

```java
    @Autowired
    private AdminUserRoleRepository adminUserRoleRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private RoleRepository roleRepository;

    @Override
    public ResponseEntity<AdminUserDTO> register(AdminUserDTO adminUserDTO) {

        HttpHeaders httpHeaders = new HttpHeaders();

        if (!validUserData(adminUserDTO, true)) {
            return new ResponseEntity<>(adminUserDTO, httpHeaders, HttpStatus.UNAUTHORIZED);
        } else {
            AdminUser adminUser = loadAdmUserData(adminUserDTO);
            adminUserRepository.saveAndFlush(adminUser);
            loadAdminRoles(adminUserDTO, adminUser);
        }

        return new ResponseEntity<>(adminUserDTO, httpHeaders, HttpStatus.OK);
    }

    @Override
    public AdminUser getAdminUserByUsername(String username) {
        return adminUserRepository.findByUsername(username);
    }

    @Override
    public AdminUser getAdminUserByEmail(String email) {
        return adminUserRepository.findByEmail(email);
    }
```

Figura 74. Implementar método register en AdminUserService.
Elaborado por: El investigador.

Al igual que en el registro de usuarios, en la implementación del método register se valida la información obtenida del AdminUserDTO y se procede a settear la información necesaria para guardar la data de AmdinUser.

```
75    private AdminUser loadAdmUserData(AdminUserDTO adminUserDTO) {
76        AdminUser adminUser = new AdminUser();
77        adminUser.setActive(true);
78        adminUser.setType(Integer.valueOf(1));
79        adminUser.setCreationDate(LocalDate.now());
80
81        adminUser.setUsername(adminUserDTO.getUsername());
82        adminUser.setPassword(passwordEncoder.encode(adminUserDTO.getPassword()));
83
84        adminUser.setIdentificationType(adminUserDTO.getIdentificationType().toString());
85        adminUser.setIdentification(adminUserDTO.getIdentificationNumber());
86        adminUser.setFirstName(adminUserDTO.getFirstName());
87        adminUser.setLastName(adminUserDTO.getLastName());
88        adminUser.setEmail(adminUserDTO.getEmail());
89        return adminUser;
90    }
```

Figura 75. Método para cargar data en AdminUser.
Elaborado por: El investigador.

Finalmente, se procede al ingreso de la data en la tabla intermedia ***AdminUserRole*** que relaciona la entidad ***AdminUser*** y la entidad ***Role***.

```
private void loadAdminRoles(AdminUserDTO adminUserDTO, AdminUser adminUser) {
    adminUserDTO.getRoles().forEach(r -> {
        AdminUserRole adminUserRole = new AdminUserRole();
        adminUserRole.setActive(true);
        adminUserRole.setCreationDate(LocalDate.now());
        Optional<Role> role = roleRepository.findById(r.getId());
        if (role.isPresent()) {
            adminUserRole.setPKey(new AdminUserRoleId(role.get(),adminUser));
            adminUserRoleRepository.saveAndFlush(adminUserRole);
        } else {
            adminUserDTO.setMessage(
                    MessageManager.formatMessages(MessageManager.getMessages(Messages.ADMIN_LOGIN_ROLE_NOT_FOUND)));
            throw new InvalidUserException();
        }
    });
}
```

Figura 76. Método para guardar data en AdminUserRole.
Elaborado por: El investigador.

11) Crear package controller.

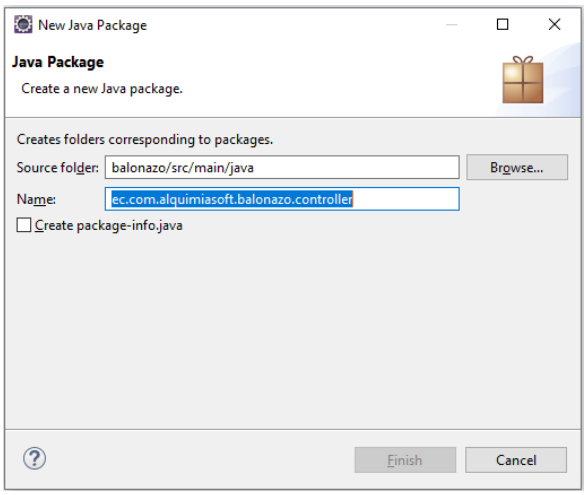


Figura 77. Crear package controller.
Elaborado por: El investigador.

12) Crear controlador UserController.

Un Controlador permite exponer servicios en modo de Web Services, en este caso se expondrán servicios REST.

UserController.java

```java
package ec.com.alquimiasoft.balonazo.controller;

import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api")
@CrossOrigin
public class UserController {

}
```

Figura 78. Crear controlador UserController.
Elaborado por: El investigador.

Para todo controlador es necesario implementar:

@RestController: Anotación que indica el tipo de servicio a exponer, en este caso Rest, pero también se puede exponer como servicio SOAP.
@RequestMapping: Anotación que establece el uri por donde ingresará el tráfico a los servicios, en este caso todas las direcciones url de los servicios empezaran con ***"/api".***
@CrossOrigin: Anotación que permite configurar las peticiones de orígenes cruzados, por defecto acepta cross origin desde cualquier sitio.

13) Exponer servicio registrar usuarios.

```java
@RestController
@RequestMapping("/api")
@CrossOrigin
public class UserController {

    @Autowired
    private UserOperations userOperations;

    public static final String USER_URI = "/user";

    @PostMapping(USER_URI)
    @ResponseBody
    public ResponseEntity<UserDTO> createUser(@Valid @RequestBody UserDTO userDTO, BindingResult bindingResult) {
        return userOperations.saveUser(userDTO, bindingResult);
    }
}
```

Figura 79. Exponer servicio createUser.
Elaborado por: El investigador.

En la *figura 79* se encuentra:

Línea 24: Código para inyectar la interfaz UserOperations con el fin de no exponer el código fuente de la implementación.

Línea 26: Código para definir una variable estática con el uri correspondiente para consumir el servicio expuesto, conocido como ***end point.***

@PostMapping: Anotación para indicar que el servicio necesita una petición POST, también se puede indicar peticiones como ***Get.***

@ResponseBody: Anotación para indicar que el servicio retorna información en el ***body*** de la petición.

@Valid: Anotación para indicar que se valide los parámetros de la petición y almacene el resultado en el parámetro BindingResult.

@RequestBody: Anotación para indicar que la petición tiene un ***body.***

Es necesario agregar el ***end point*** en el filtro JWT de las peticiones que no requieren ***access token,*** debido a que por defecto toda petición a requiere un ***access token.***

```
public boolean allowRequestWithoutToken(HttpServletRequest request) {
    return "/api".concat(UserController.USER_URI).equals(request.getRequestURI());
}
```

Figura 80. Agregar end point saveUser en filtro JWT.
Elaborado por: El investigador.

14) Crear controlador AdminUserController.

JWTFilter.java | *AdminUserController.java

```
1  package ec.com.alquimiasoft.balonazo.controller;
2
3  import org.springframework.web.bind.annotation.CrossOrigin;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import org.springframework.web.bind.annotation.RestController;
6
7  @RestController
8  @RequestMapping("/api")
9  @CrossOrigin
10 public class AdminUserController {
11
12 }
```

Figura 81. Crear controlador AdminUserController.
Elaborado por: El investigador.

15) Exponer servicio registrar administradores.

```java
@RestController
@RequestMapping("/api")
@CrossOrigin
public class AdminUserController {

    @Autowired
    private AdminUserOperations adminUserOperations;

    public static final String ADM_REGISTER_URI = "/adm/register";

    @PostMapping(ADM_REGISTER_URI)
    @ResponseBody
    public ResponseEntity<AdminUserDTO> register(@RequestBody AdminUserDTO adminUserDTO) {
        return adminUserOperations.register(adminUserDTO);
    }
}
```

Figura 82. Exponer servicio register AdminUser.
Elaborado por: El investigador.

16) Probar end point registrar usuarios.

Para probar la funcionalidad de los servicios se utiliza la herramienta PostMan que permite ejecutar peticiones http.

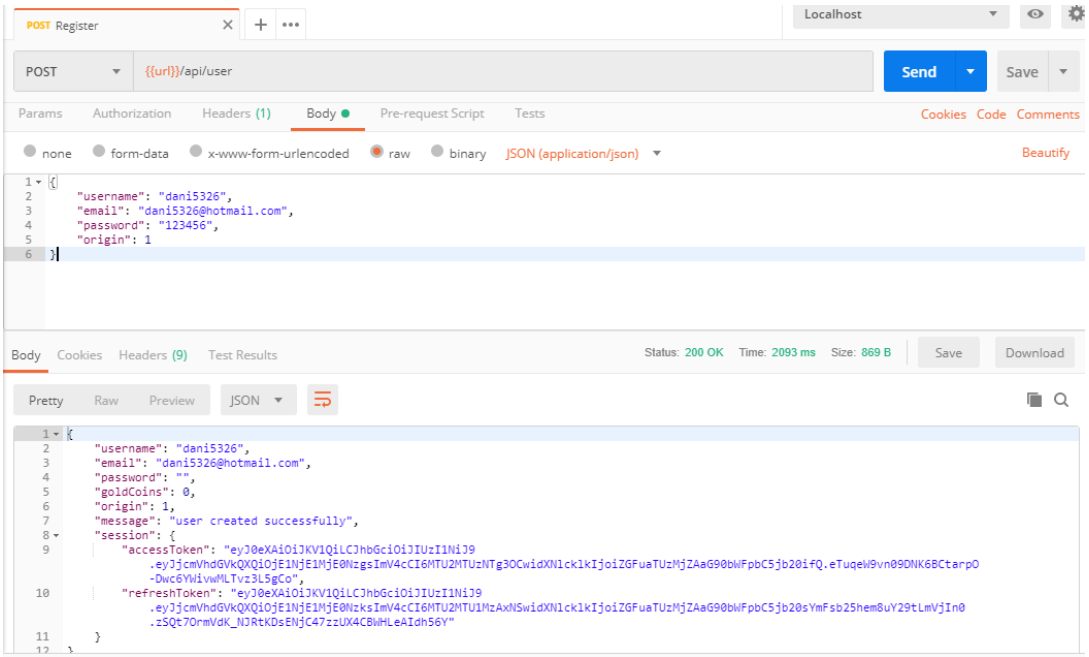


Figura 83. Probar end point saveUser.
Elaborado por: El investigador.

Con la herramienta PostMan se puede probar cualquier tipo de petición http y verificar el resultado de la petición, en este caso se utiliza formatos JSON.

Al probar el end point saveUser se guarda exitosamente la información del usuario y retorna el DTO con la información del usuario como por ejemplo el ***access y refresh token.***

Es necesario que se verifiquen todos los casos y validaciones en los servicios, lo cual se lo realiza con las pruebas unitarias.

17) Probar end point registrar administradores.

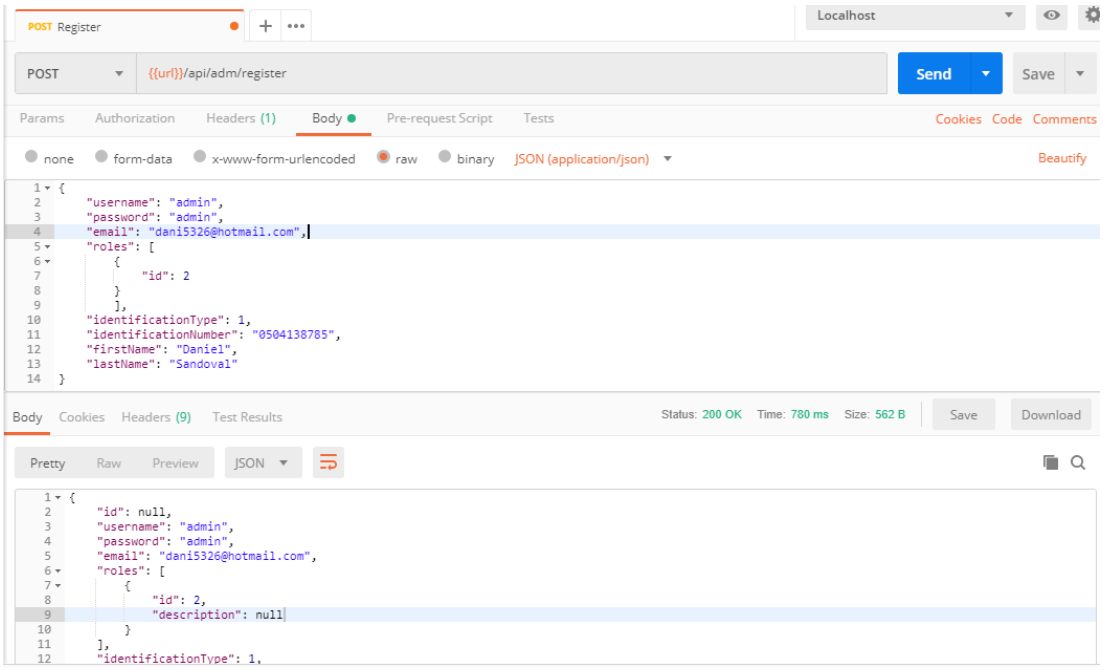


Figura 84. Probar end point register AdminUser.
Elaborado por: El investigador.

- **Crear Servicio de Login para Usuarios y Administradores.**

1) Crear método login en UserOperations.

```
User findByEmail(String email);

ResponseEntity<UserLoginDTO> getUserLogin(UserLoginDTO userLoginDTO, BindingResult bindingResult);
```

Figura 85. Declarar método getUserLogin.
Elaborado por: El investigador.

2) Implementar método login en UserService.

Como cualquier método de login, se valida que el correo electrónico y clave enviados en la petición, coincidan con los datos en la base de datos. Antes de comparar las credenciales se valida los parámetros requeridos y condiciones de longitud.

*UserService.java UserLoginDTO.java

```
56
57      @Override
58      public ResponseEntity<UserLoginDTO> getUserLogin(UserLoginDTO userLoginDTO, BindingResult bindingResult) {
59
60          HttpHeaders httpHeaders = new HttpHeaders();
61          UserLoginDTO userDto = new UserLoginDTO();
62          List<String> errors = null;
63
64          if (bindingResult.hasErrors()) {
65              errors = bindingResult.getAllErrors().stream().map(e -> e.getCode() + "." + e.getDefaultMessage())
66                      .collect(Collectors.toList());
67              userDto.setEmail(userLoginDTO.getEmail());
68              userDto.setMessage(errors.toString());
69              return new ResponseEntity<>(userDto, httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
70          }
71
72          if (userLoginDTO.getOrigin() == UserOriginEnum.BALONAZO.getCode()
73                  && (userLoginDTO.getPassword() == null || userLoginDTO.getPassword().length() > maxPwdLength)) {
74              userDto.setMessage("Longitud del password debe tener un maximo de ".concat(maxPwdLength.toString())
75                      .concat(" caracteres"));
76              return new ResponseEntity<>(userDto, httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
77          }
78
79          User user = findByEmail(userLoginDTO.getEmail().toLowerCase());
80
81          if (user == null) {
82              userDto.setEmail(userLoginDTO.getEmail());
83              userDto.setMessage("Usuario o contraseña incorrectos");
84              return new ResponseEntity<>(userDto, httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
85          }
86
87          if (user.getOrigin() != userLoginDTO.getOrigin()) {
88              userDto.setOrigin(user.getOrigin());
89              userDto.setMessage("Usuario ya registrado con otro canal");
90              return new ResponseEntity<>(userDto, httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
91          }
92
93          if (!passwordEncoder.matches(userLoginDTO.getPassword(), user.getPassword())) {
94              userDto.setEmail(userLoginDTO.getEmail());
95              userDto.setMessage("Usuario o contraseña incorrectos");
96              return new ResponseEntity<>(userDto, httpHeaders, HttpStatus.UNAUTHORIZED);
97          }
```

Figura 86. Implementar método getUserLogin – parte 1
Elaborado por: El investigador.

```
199         userDto.setEmail(user.getEmail());
200         userDto.setUsername(user.getUsername());
201         userDto.setPassword("");
202         userDto.setFirstName(user.getFirstName());
203         userDto.setLastName(user.getLastName());
204         userDto.setBirthDate(user.getBirthDate());
205         userDto.setPhone(user.getPhone());
206         userDto.setCity(user.getCity());
207
208         if (user.getSupportedTeamId() != null) {
209             userDto.setSupportedTeam(new TeamDTO(user.getSupportedTeamId().getId(),
210                     user.getSupportedTeamId().getDescription(), user.getSupportedTeamId().getLogoUrl(),
211                     user.getSupportedTeamId().getPetUrl(), user.getSupportedTeamId().getCountryId().getId(),
212                     user.getSupportedTeamId().getLeagueSerieId().getId(),
213                     user.getSupportedTeamId().getTeamtype().getId(), user.getSupportedTeamId().getActive()));
214         }
215
216         userDto.setGoldCoins(user.getGoldBalls());
217         userDto.setSession(
218                 new SessionDTO(tokenService.createToken(user.getEmail(), TokenTypeEnum.ACCESS_TOKEN.getCode()),
219                         tokenService.createToken(user.getEmail().concat(",").concat(jwtSettings.getTokenIssuer()),
220                                 TokenTypeEnum.REFRESH_TOKEN.getCode())));
221
222         return new ResponseEntity<>(userDto, httpHeaders, HttpStatus.OK);
223     }
```

Figura 87. Implementar método getUserLogin – parte 2.
Elaborado por: El investigador.

3) Crear método login AdminUserOperations.

```
15      ResponseEntity<AdminUserDTO> login(AdminUserDTO adminUserDTO);
16  }
```

Figura 88. Declarar método login en AdminUserOperations.
Elaborado por: El investigador.

4) Implementar método login en AdminUserService.

*AdminUserService.java

```
72      }
73
74      @Override
75      public ResponseEntity<AdminUserDTO> login(AdminUserDTO adminUserDTO) {
76          HttpHeaders httpHeaders = new HttpHeaders();
77
78          if (!validUserData(adminUserDTO, false)) {
79              return new ResponseEntity<>(adminUserDTO, httpHeaders, HttpStatus.UNAUTHORIZED);
80          } else {
81              AdminUser adminUser = adminUserRepository.findByUsername(adminUserDTO.getUsername());
82              if (adminUser != null) {
83                  if (!passwordEncoder.matches(adminUserDTO.getPassword(), adminUser.getPassword())) {
84                      adminUserDTO.setMessage(MessageManager
85                              .formatMessages(MessageManager.getMessages(Messages.ADMIN_LOGIN_WRONG_USER_PWD)));
86                      return new ResponseEntity<>(adminUserDTO, httpHeaders, HttpStatus.UNAUTHORIZED);
87                  }
88
89                  if (adminUser.getAdminUserRoleList() == null || adminUser.getAdminUserRoleList().isEmpty()) {
90                      adminUserDTO.setMessage(MessageManager
91                              .formatMessages(MessageManager.getMessages(Messages.ADMIN_LOGIN_UNASSIGNED_ROLES)));
92                      return new ResponseEntity<>(adminUserDTO, httpHeaders, HttpStatus.UNAUTHORIZED);
93                  }
94                  AdminUserRole adminUserRole = adminUser.getAdminUserRoleList().get(0);
95
96                  adminUserDTO.setRoles(new ArrayList<>());
97
98                  RoleUserDTO roleUserDTO = new RoleUserDTO(adminUserRole.getPKey().getRole().getId(),
99                          adminUserRole.getPKey().getRole().getRoleName());
00
01                  adminUserDTO.getRoles().add(roleUserDTO);
02                  adminUserDTO.setSession(new SessionDTO(
03                          tokenService.createToken(adminUser.getUsername(), TokenTypeEnum.ACCESS_TOKEN.getCode()),
04                          tokenService.createToken(
05                                  adminUser.getUsername().concat(",").concat(jwtSettings.getTokenIssuer()),
06                                  TokenTypeEnum.REFRESH_TOKEN.getCode())));
07
08              } else {
09                  adminUserDTO.setMessage(
10                          MessageManager.formatMessages(MessageManager.getMessages(Messages.ADMIN_LOGIN_USER_NOT_FOUND)));
11                  return new ResponseEntity<>(adminUserDTO, httpHeaders, HttpStatus.UNAUTHORIZED);
12              }
13          }
14
15          return new ResponseEntity<>(adminUserDTO, httpHeaders, HttpStatus.OK);
16      }
```

Figura 89. Implementar método login en AdminUserService.
Elaborado por: El investigador.

5) Exponer servicio login User.

Se agrega la uri para acceder al servicio de login y se implementa el servicio Post.

UserController.java

```
1  package ec.com.alquimiasoft.balonazo.controller;
2
3  import javax.validation.Valid;
18
19 @RestController
20 @RequestMapping("/api")
21 @CrossOrigin
22 public class UserController {
23
24     @Autowired
25     private UserOperations userOperations;
26
27     public static final String USER_URI = "/user";
28     public static final String LOGIN_URI = "/user/login";
29
30     @PostMapping(USER_URI)
31     @ResponseBody
32     public ResponseEntity<UserDTO> createUser(@Valid @RequestBody UserDTO userDTO, BindingResult bindingResult) {
33         return userOperations.saveUser(userDTO, bindingResult);
34     }
35
36     @PostMapping(LOGIN_URI)
37     @ResponseBody
38     public ResponseEntity<UserLoginDTO> getUserLogin(@Valid @RequestBody UserLoginDTO userLoginDTO,
39             BindingResult bindingResult) {
40         return userOperations.getUserLogin(userLoginDTO, bindingResult);
41     }
42 }
```

Figura 90. Exponer servicio getUserLogin.
Elaborado por: El investigador.

6) Exponer servicio login AdminUser.

AdminUserController.java

```java
package ec.com.alquimiasoft.balonazo.controller;

import org.springframework.beans.factory.annotation.Autowired;

@RestController
@RequestMapping("/api")
@CrossOrigin
public class AdminUserController {

    @Autowired
    private AdminUserOperations adminUserOperations;

    public static final String ADM_REGISTER_URI = "/adm/register";
    public static final String ADM_LOGIN_URI = "/adm/login";

    @PostMapping(ADM_REGISTER_URI)
    @ResponseBody
    public ResponseEntity<AdminUserDTO> register(@RequestBody AdminUserDTO adminUserDTO) {
        return adminUserOperations.register(adminUserDTO);
    }

    @PostMapping(ADM_LOGIN_URI)
    @ResponseBody
    public ResponseEntity<AdminUserDTO> login(@RequestBody AdminUserDTO adminUserDTO) {
        return adminUserOperations.login(adminUserDTO);
    }
}
```

Figura 91. Exponer servicio login AdminUser.
Elaborado por: El investigador.

7) Probar end point user login.

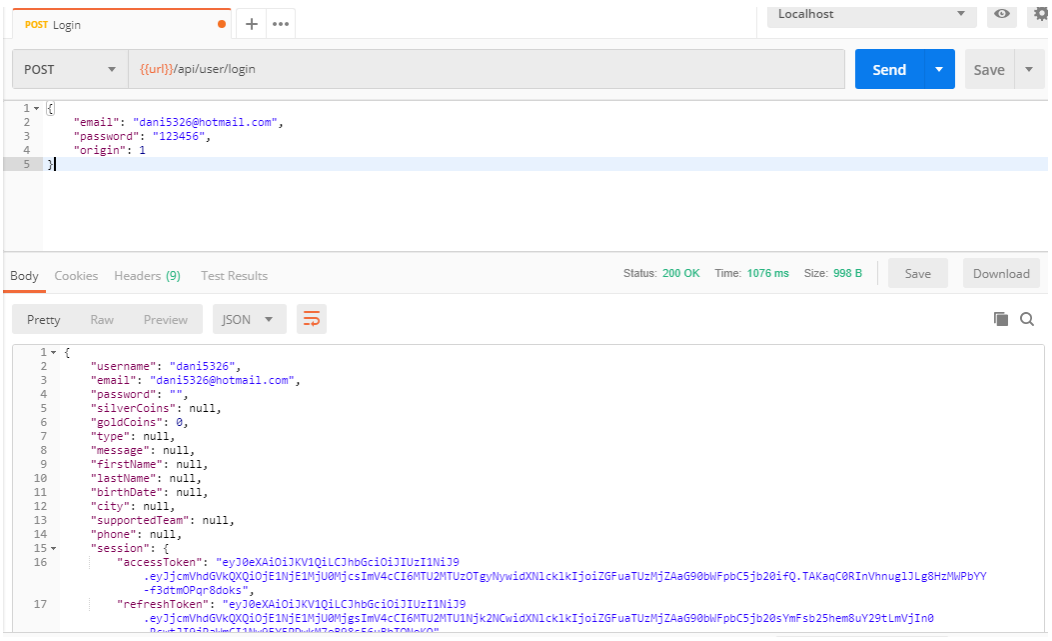


Figura 92. Probar end point login User.
Elaborado por: El investigador.

8) Probar end point admin user login.

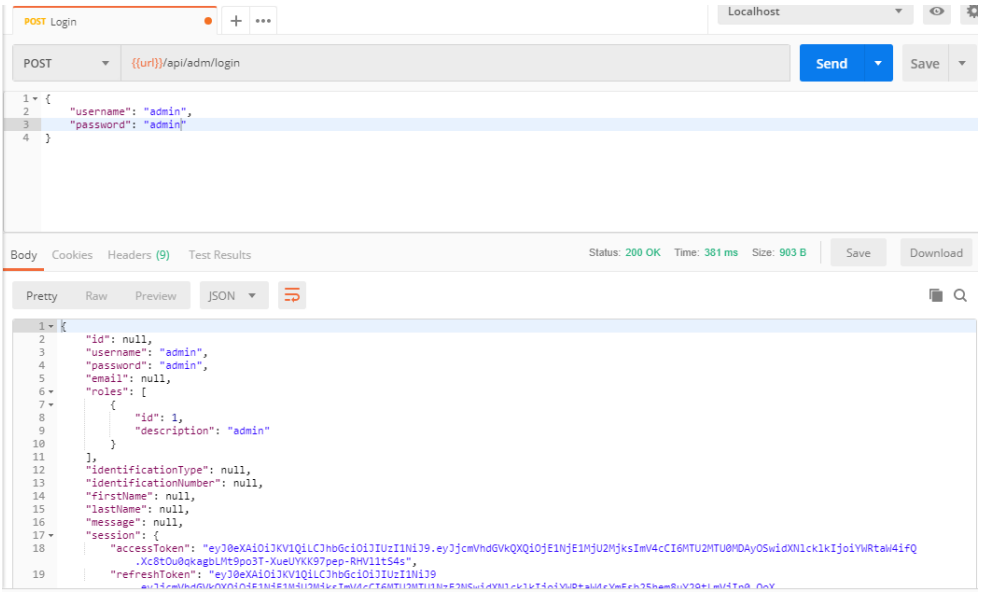


Figura 93. Probar end point login AdminUser.
Elaborado por: El investigador.

- **Entregar balones por registro**

La lógica de negocio establece que cada usuario registrado debe recibir un valor ***n*** de balones de oro como regalo.

1) Actualizar función registro de usuarios para el requerimiento.

```
123         user.setGoldBalls(Integer.valueOf(0));
124         user.setOrigin(userDTO.getOrigin());
125         user.setGoldBalls(GOLDBALLONS);
126
127         user = userReposit
128
129         userDto.setGoldCoi
130         userDto.setPasswor
131         userDto.setMessage("user created successfully");
132         userDto.setSession(
```

Integer ec.com.alquimiasoft.balonazo.service.UserService.GOLDBALLONS

@Value(value="${balonazo.user.goldcoins}")

Press 'F2' for focus

Figura 94. Entregar balones por registro.
Elaborado por: El investigador.

En la figura 94 se encuentra:

Línea 125: Código para settear en el objeto ***User*** la cantidad de balones de oro establecido en las propiedades ***appliacion.yml*** del proyecto.

- **Sprint 3**

- **Mostrar cartilla a jugar**

Actividades:

- **Crear entidades CardDetail y Card.**

Crear las entidades CardDetail y Card para que spring genere las entidades en la Base de Datos al ejecutar la aplicación.

1) Crear entidad Card.

Card.java

```
@Entity
@Table(name = "blz_card")
@NoArgsConstructor
@AllArgsConstructor
@Data
public class Card implements Serializable {
    private static final long serialVersionUID = 1335311502872202982L;

    @Id
    @SequenceGenerator(name="pk_sequence_card",sequenceName="blz_card_seq", allocationSize=1)
    @GeneratedValue(strategy=GenerationType.SEQUENCE,generator="pk_sequence_card")
    private Long id;
    @NotNull
    private LocalDate creationDate;

    @ColumnDefault("'1'")
    private Boolean active;

    @NotNull
    private LocalDateTime dateFrom;

    @NotNull
    private Integer number;
    @NotNull
    private LocalDateTime dateTo;
    @OneToMany(mappedBy = "pKey.card")
    private List<CardDetail> cardDetailList;
    private Integer status;
    private String description;
}
```

Figura 95. Crear entidad Card.
Elaborado por: El investigador.

2) Crear entidad CardDetail.

CardDetail.java

```
@Entity
@Table(name = "blz_card_detail")
@NoArgsConstructor
@AllArgsConstructor
@Data
public class CardDetail implements Serializable {
    private static final long serialVersionUID = 7913425490443828534L;
    @EmbeddedId
    CardDetailId pKey;
    @NotNull
    private LocalDate creationDate;
    @ColumnDefault("'1'")
    private Boolean active;
    @NotNull
    @ManyToOne
    @JoinColumn(name = "local_id")
    private Team localTeamId;
    @NotNull
    @ManyToOne
    @JoinColumn(name = "visitor_id")
    private Team visitorTeamId;
    @NotNull
    private LocalDateTime matchDateHour;
    @NotNull
    @ManyToOne
    @JoinColumn(name = "stadium_id")
    private CatalogDetail stadium;
    @NotNull
    @ManyToOne
    @JoinColumn(name = "match_type_id")
    private CatalogDetail matchType;

    private Integer localScore;

    private Integer visitorScore;

    @ColumnDefault("'1'")
    private Boolean enableStats;
}
```

Figura 96. Crear entidad CardDetail.
Elaborado por: El investigador.

Crear las interfaces ***Repository*** para cada entidad como se muestra en la figura 61.

- **Crear servicio para obtener la cartilla en juego.**

Todo servicio implementa una interfaz ***operations*** correspondiente a su entidad.

1) Crear interfaz ***CardRepository***

CardRepository.java

```java
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import ec.com.alquimiasoft.balonazo.model.Card;

@Repository
public interface CardRepository extends JpaRepository<Card, Long> {

    @Query("SELECT ca FROM Card ca WHERE ca.active = true "
            + " AND ca.status = 1 AND :date BETWEEN ca.dateFrom AND "
            + " (SELECT MIN(cd.matchDateHour) FROM ca.cardDetailList cd) AND "
            + " (SELECT MIN(cd2.matchDateHour) FROM ca.cardDetailList cd2) = :minDate")
    Card getCardValidBetweenDate(@Param("date") LocalDateTime date, @Param("minDate") LocalDateTime minDate);
```

Figura 97. Crear interfaz CardRepository.
Elaborado por: El investigador.

En la figura 97 se encuentra:

@Query: Anotación de spring data para establecer consultas sql complejas. JPA permite construir consultas utilizando los campos de las entidades como si fuese programación en Java Orientada a Objetos.

@Param: Notación de spring data para establecer los parámetros que se van a utilizar en la consulta sql.

2) Crear función ***loadCardDTO***

La función llena la información a retornar en un objeto DTO.

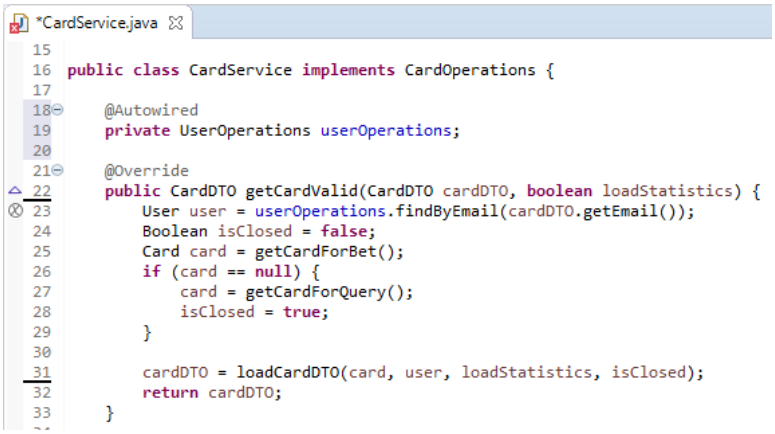
```java
@Override
public CardDTO loadCardDTO(Card card, User user, boolean loadStatistics, Boolean isClosed) {
    CardDTO cardDTO = null;
    if (card != null) {
        cardDTO = new CardDTO();
        cardDTO.setId(card.getId());
        cardDTO.setDateFrom(card.getDateFrom());
        cardDTO.setIsClosed(isClosed);
        cardDTO.setDescription(card.getDescription());

        Optional<CardDetail> minMatchDate = card.getCardDetailList().stream()
                .collect(Collectors.minBy(Comparator.comparing(CardDetail::getMatchDateHour)));
        if (minMatchDate.isPresent()) {
            cardDTO.setDateTo(minMatchDate.get().getMatchDateHour().minusMinutes(minutesBeforeDeadline));
        }
    }
    return cardDTO;
}
```

Figura 98. Crear función loadCardDTO.
Elaborado por: El investigador.

3) Crear servicio ***getCardValid*** en ***CardService.***

El servicio obtiene la última cartilla jugable, en base a la fecha y hora de inicio de juego, valor presente en la cartilla. El servicio devuelve la información necesaria para que Front pueda mostrar la cartilla final al usuario y pueda realizar su apuesta.



```java
*CardService.java
15
16 public class CardService implements CardOperations {
17
18     @Autowired
19     private UserOperations userOperations;
20
21     @Override
22     public CardDTO getCardValid(CardDTO cardDTO, boolean loadStatistics) {
23         User user = userOperations.findByEmail(cardDTO.getEmail());
24         Boolean isClosed = false;
25         Card card = getCardForBet();
26         if (card == null) {
27             card = getCardForQuery();
28             isClosed = true;
29         }
30
31         cardDTO = loadCardDTO(card, user, loadStatistics, isClosed);
32         return cardDTO;
33     }
```

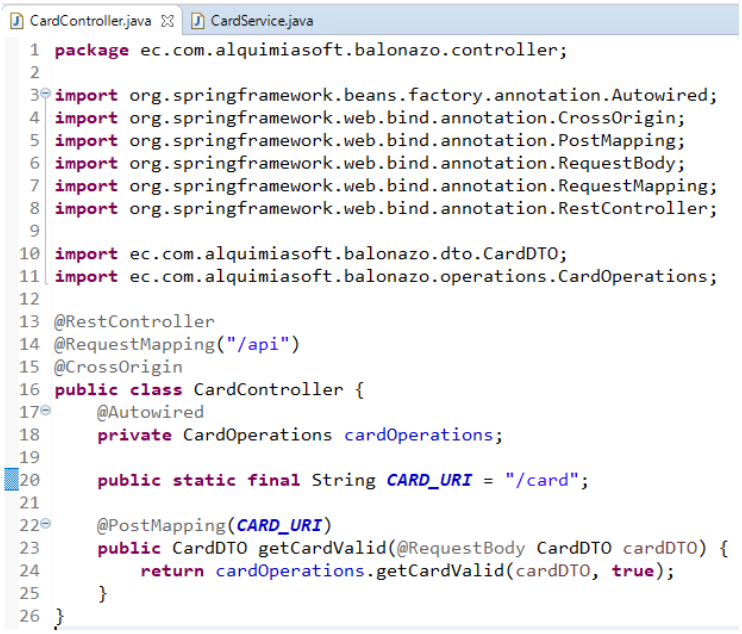
Figura 99. Crear servicio getCardValid.
Elaborado por: El investigador.

Crear las interfaces ***operations*** como se muestra en la figura 65.

- **Exponer Servicio getCardValid.**

Exponer el servicio getCardValid mediante la clase CardController.

1) Crear end point ***/card.***



```java
CardController.java  CardService.java
1  package ec.com.alquimiasoft.balonazo.controller;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.web.bind.annotation.CrossOrigin;
5  import org.springframework.web.bind.annotation.PostMapping;
6  import org.springframework.web.bind.annotation.RequestBody;
7  import org.springframework.web.bind.annotation.RequestMapping;
8  import org.springframework.web.bind.annotation.RestController;
9
10 import ec.com.alquimiasoft.balonazo.dto.CardDTO;
11 import ec.com.alquimiasoft.balonazo.operations.CardOperations;
12
13 @RestController
14 @RequestMapping("/api")
15 @CrossOrigin
16 public class CardController {
17     @Autowired
18     private CardOperations cardOperations;
19
20     public static final String CARD_URI = "/card";
21
22     @PostMapping(CARD_URI)
23     public CardDTO getCardValid(@RequestBody CardDTO cardDTO) {
24         return cardOperations.getCardValid(cardDTO, true);
25     }
26 }
```

Figura 100. Exponer servicio getCardValid.
Elaborado por: El investigador.

- **Probar servicio en PostMan.**

Llamar al end point ***/card*** con la herramienta PostMan y verificar el que el servicio funcione. La base de datos dispone de data precargada para probar el servicio.

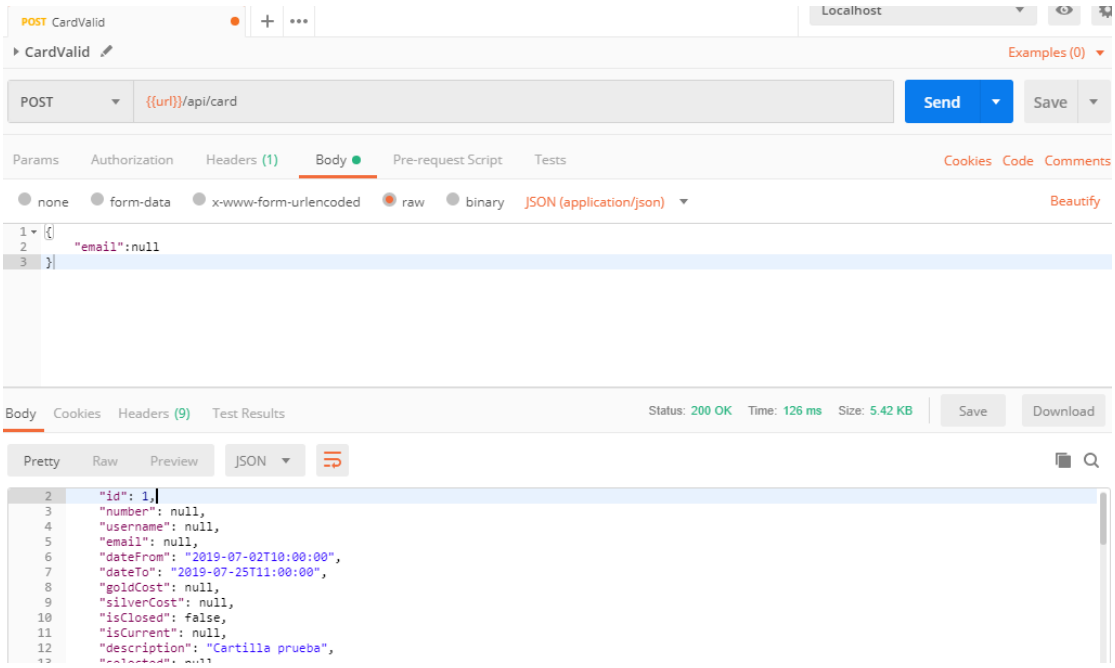


Figura 101. Prueba end point /card.
Elaborado por: El investigador.

- **Guardar pronóstico por Usuario**

Actividades:

- **Crear entidades Prognostic y PrognosticDetail.**

Crear interfaces ***repository*** de las entidades Prognostic y PrognosticDetail.

1) Crear entidad ***Prognostic*** como se muestra en la figura 102**.**
2) Crear entidad ***PrognosticDetail*** como se muestra en la figura 103**.**

```java
@AllArgsConstructor
public class Prognostic implements Serializable {
    private static final long serialVersionUID = -8724902444163706357L;

    @Id
    @SequenceGenerator(name = "pk_sequence_prog", sequenceName = "blz_prognostic_seq",
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "pk_sequence_prog")
    private Long id;
    @NotNull
    private LocalDateTime creationDate;
    @ColumnDefault("'1'")
    private Boolean active;
    @OneToMany(mappedBy = "pKey.prognostic")
    private List<PrognosticDetail> prognosticDetailList;

    @ColumnDefault("'0'")
    private Boolean processed;
    @NotNull
    @ManyToOne
    @JoinColumn(name = "card_id")
    private Card card;
    @NotNull
    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    @NotNull
    private Integer code;

    private Boolean applyPrize;
}
```

Figura 102. Crear entidad Prognostic.
Elaborado por: El investigador.

```java
@Entity
@Table(name = "blz_prognostic_detail")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class PrognosticDetail implements Serializable {
    private static final long serialVersionUID = 2761405986556499733L;

    @EmbeddedId
    private PrognosticDetailId pKey;

    @NotNull
    private LocalDate creationDate;

    @ColumnDefault("'1'")
    private Boolean active;

    @NotNull
    private String prognosticValue;

    @OneToOne(fetch = FetchType.LAZY)
    @NotNull
    @JoinColumns({
        @JoinColumn(name="card_det_id", referencedColumnName="id"),
        @JoinColumn(name="card_id", referencedColumnName="card_id")
    })
    private CardDetail cardDetail;

    @ColumnDefault("'0'")
    private Boolean match;
}
```

Figura 103. Crear entidad PrognosticDetail.
Elaborado por: El investigador.

- **Crear servicio para guardar un pronóstico por usuario.**

  1) Crear función para obtener el pronóstico a guardar.

  La función transforma el DTO proveniente de los parámetros de la petición a una entidad ***Prognostic*** para posteriormente guardar la ***data.***

```java
@Override
public Prognostic cardDTOToPrognostic(PrognosticDTO prognosticDTO, User user) {
    Optional<Card> optional = cardRepository.findById(prognosticDTO.getBook().getId());
    Prognostic prognostic = new Prognostic();

    if (optional.isPresent()) {
        Card card = optional.get();
        validateCard(card);
        prognostic.setUser(user);
        prognostic.setCard(card);
        prognostic.setPrognosticDetailList(new ArrayList<>());
        prognostic.setCreationDate(LocalDateTime.now());
        prognostic.setActive(true);
        prognostic.setProcessed(false);
        prognostic.setCode(prognosticRepository.getPrognosticCountPerCardAndUser(card.getId(), user.getId()) + 1);
        card.getCardDetailList()
                .forEach(cardDetail -> prognostic.getPrognosticDetailList()
                        .add(setPrognosticDetail(prognosticDTO.getBook(), prognostic, cardDetail,
                                Integer.valueOf(card.getCardDetailList().indexOf(cardDetail) + 1))));
    } else {
        return null;
    }

    return prognostic;
}
```

Figura 104. Función obtener pronostico a guardar.
Elaborado por: El investigador.

2) Crear función ***savePrognostic***

```java
85      @Override
86      public Prognostic savePrognostic(PrognosticDTO prognosticDTO, User user) {
87          validatePrognostic(prognosticDTO);
88          Prognostic prognostic = cardDTOToPrognostic(prognosticDTO, user);
89          if (prognostic != null) {
90              prognosticRepository.saveAndFlush(prognostic);
91              prognosticDetailRepository.saveAll(prognostic.getPrognosticDetailList());
92          }
93          return prognostic;
94      }
```

Figura 105. Crear función savePrognostic.
Elaborado por: El investigador.

En la imagen 105 se encuentra:

Línea 87: Código para validar que todos los campos sean diferentes de nulo antes de procesar a guardar la ***data.***

Línea 90 y 91: Código para guardar la información en las tablas ***Prognostic*** y ***PrognosticDetail.***

3) Crear servicio ***saveData***

El servicio se encarga de validar la información que viaja en la petición antes de guardarla para evitar conflictos. El servicio también actualiza el número de balones de oro del usuario, restando del valor total el costo de envío de una cartilla (1 balón de oro) establecido por la lógica de negocio.

```java
ognosticService.java   application.yml   SavePrognosticResponseDTO.java
import ec.com.alquimiasoft.balonazo.repository.PrognosticDetailRepository;
import ec.com.alquimiasoft.balonazo.repository.PrognosticRepository;

@Service
public class PrognosticService implements PrognosticOperations {

    @Value("${balonazo.card.goldcost}")
    private Integer goldCost;

    @Autowired
    private PrognosticRepository prognosticRepository;

    @Autowired
    private PrognosticDetailRepository prognosticDetailRepository;

    @Autowired
    private UserOperations userOperations;

    @Transactional
    public ResponseEntity<SavePrognosticResponseDTO> saveData(PrognosticDTO prognosticDTO, String authorizationToken) {
        ResponseEntity<SavePrognosticResponseDTO> response = null;

        User user = userOperations.validateUser(authorizationToken);
        Prognostic prognostic = savePrognostic(prognosticDTO, user);
        if (prognostic != null) {
            updateCostPerPrognostic(user);
            userOperations.saveUserData(user);

            /* Obtener datos actuales Usuario */
            SavePrognosticResponseDTO savePrognosticResponseDto = new SavePrognosticResponseDTO();
            savePrognosticResponseDto.setGoldCoins(user.getGoldBalls());

            response = new ResponseEntity<>(savePrognosticResponseDto, new HttpHeaders(), HttpStatus.CREATED);
        } else {
            response = new ResponseEntity<>(new SavePrognosticResponseDTO(), new HttpHeaders(), HttpStatus.NOT_FOUND);
        }

        return response;
    }
```

Figura 106. Servicio guardar pronóstico usuario.
Elaborado por: El investigador.

- **Exponer servicio savePrognostic.**

  1) Crear end point ***save/prognostic***

```java
PrognosticController.java
1  package ec.com.alquimiasoft.balonazo.controller;
2
3  import org.springframework.beans.factory.annotation.Autowired;
15
16 @RestController
17 @RequestMapping("/api")
18 @CrossOrigin
19 public class PrognosticController {
20     @Autowired
21     private PrognosticOperations prognosticOperations;
22
23     public static final String PROGNOSTIC_URI = "/save/prognostic";
24
25     @PostMapping(PROGNOSTIC_URI)
26     public ResponseEntity<SavePrognosticResponseDTO> savePrognostic(@RequestBody PrognosticDTO prognosticDTO,
27             @RequestHeader(value = "Authorization") String authorizationToken) {
28         return prognosticOperations.saveData(prognosticDTO, authorizationToken);
29     }
30 }
```

Figura 107. Crear end point save/prognostic.
Elaborado por: El investigador.

- **Actualizar datos de usuarios**

Actividades:

- **Crear servicio para actualizar datos de un usuario.**

  1) Crear servicio ***updateUser*** en la clase ***UserService*** como se muestra en las figuras 108 y 109**.**

```
public ResponseEntity<UserUpdateDTO> updateUser(UserUpdateDTO userUpdateDTO, String authorizationToken) {
    User user = validateUser(authorizationToken);

    Boolean profileNotCompletedBefore = (user.getProfileFulfilled() == null || !user.getProfileFulfilled())
            && (user.getFirstName() == null || user.getLastName() == null || user.getBirthDate() == null
                    || user.getPhone() == null || user.getCity() == null || user.getSupportedTeamId() == null);

    HttpHeaders httpHeaders = new HttpHeaders();
    user.setUsername(userUpdateDTO.getUsername() == null ? user.getUsername() : userUpdateDTO.getUsername());
    user.setFirstName(userUpdateDTO.getFirstName() == null ? user.getFirstName() : userUpdateDTO.getFirstName());
    user.setLastName(userUpdateDTO.getLastName() == null ? user.getLastName() : userUpdateDTO.getLastName());
    user.setBirthDate(userUpdateDTO.getBirthDate() == null ? user.getBirthDate() : userUpdateDTO.getBirthDate());
    user.setPhone(userUpdateDTO.getPhone());
    user.setCity(userUpdateDTO.getCity() == null ? user.getCity() : userUpdateDTO.getCity());

    if (userUpdateDTO.getSupportedTeamId() != null) {
        Team team = teamOperations.getTeam(userUpdateDTO.getSupportedTeamId());
        user.setSupportedTeamId(team);
    }
```

Figura 108. Crear servicio updateUser parte –1.
Elaborado por: El investigador.

```
    Boolean profileCompletedAfter = (user.getProfileFulfilled() == null || !user.getProfileFulfilled())
            && (user.getFirstName() != null && user.getLastName() != null && user.getBirthDate() != null
                    && user.getPhone() != null && user.getCity() != null && user.getSupportedTeamId() != null);

    if (profileNotCompletedBefore && profileCompletedAfter) {
        user.setProfileFulfilled(true);
    }
    saveUserData(user);

    return new ResponseEntity<>(userUpdateDTO, httpHeaders, HttpStatus.OK);
}
```

Figura 109. Crear servicio updateUser parte – 2.
Elaborado por: El investigador.

En el servicio updateUser se encuentra:

Línea 236: Código para validar el access token de la petición y da paso a ejecutar el servicio.

Línea 256: Código para verificar si la cuenta del usuario tiene todos los campos llenos y lo actualiza como cuenta completa.

- **Exponer servicio updateUser.**

  1) Crear end point ***user/update***

```
public class UserController {

    @Autowired
    private UserOperations userOperations;

    public static final String USER_URI = "/user";
    public static final String LOGIN_URI = "/user/login";
    public static final String USER_UPDATE_URI = "/user/update";

    @PostMapping(USER_UPDATE_URI)
    @ResponseBody
    public ResponseEntity<UserUpdateDTO> updateUser(@Valid @RequestBody UserUpdateDTO userUpdateDTO,
            BindingResult bindingResult, @RequestHeader(value = "Authorization") String authorizationToken) {
        return userOperations.updateUser(userUpdateDTO, authorizationToken);
    }
```

Figura 110. Crear end point user/update.
Elaborado por: El investigador.

- **Sprint 4**

- **Manejo de estadísticas por partido**

Actividades:

- **Crear entidades *CardDetailStats y LastMatchesStats***

La lógica de negocio determina que cada partido puede tener asignado estadísticas, tales como: posición en la tabla, probabilidades de victoria y empate, 3 últimos resultados de los partidos de cada equipo.

1) Crear entidad CardDetailStats como se muestra en la figura 111.
2) Crear entidad LastMatchesStats como se muestra en la figura 112.

CardDetailStats.java

```java
package ec.com.alquimiasoft.balonazo.model;

import java.io.Serializable;

@Entity
@Table(name = "blz_card_detail_stats")
@NoArgsConstructor
@AllArgsConstructor
@Data
public class CardDetailStats implements Serializable {

    private static final long serialVersionUID = -5095479485251954201L;

    @EmbeddedId
    private CardDetailStatsId pKey;

    @NotNull
    private LocalDate creationDate;

    @ColumnDefault("'1'")
    private Boolean active;

    @NotNull
    @ManyToOne
    @JoinColumn(name = "team")
    private Team team;

    private Integer position;

    private Integer probability;
}
```

Figura 111. Crear entidad CardDetailStats.
Elaborado por: El investigador.

LastMatchesStats.java

```java
package ec.com.alquimiasoft.balonazo.model;

import java.io.Serializable;

@Entity
@Table(name = "blz_last_matches_stats")
@NoArgsConstructor
@AllArgsConstructor
@Data
public class LastMatchesStats implements Serializable {

    private static final long serialVersionUID = -5580189492901328677L;

    @EmbeddedId
    private LastMatchesStatsId pKey;

    @NotNull
    private LocalDate creationDate;

    @ColumnDefault("'1'")
    private Boolean active;

    @NotNull
    @ManyToOne
    @JoinColumn(name = "team")
    private Team team;


    private Integer scoreLocal;

    private String result;

    private Integer scoreVisitant;

    private Integer matchType;

    private LocalDateTime matchDate;

}
```

Figura 112. Crear entidad LastMatchesStats.
Elaborado por: El investigador.

- **Actualizar servicio *getCardValid.***

Actualizar el servicio *getCardValid* para agregar estadísticas de los partidos en el DTO de respuesta del servicio.

1) Crear función para agregar estadísticas del equipo al Detail DTO.

La función obtiene las estadísticas de un equipo y retorna un DTO con los datos.

```
private StatsDetailDTO setStatsDetailDTO(Long team, CardDetail cardDetail) {
    CardDetailStats cardDetailStats = cardDetailStatsRepository
            .getStatsPerTeam(cardDetail.getPKey().getCard().getId(), cardDetail.getPKey().getId(), team);
    if (cardDetailStats != null) {
        StatsDetailDTO statsdetailDto = new StatsDetailDTO();
        statsdetailDto.setCardId(cardDetailStats.getPKey().getCardId());
        statsdetailDto.setId(cardDetailStats.getPKey().getId());
        statsdetailDto.setPosition(cardDetailStats.getPosition());
        statsdetailDto.setProbability(cardDetailStats.getProbability());
        setLastMatches(cardDetailStats.getPKey().getCardId(), cardDetailStats.getPKey().getCardDetailId(), team,
                statsdetailDto.getLastGames(), cardDetailStats.getPKey().getId());
        return statsdetailDto;
    } else {
        return null;
    }
}
```

Figura 113. Crear función setStatsDetailDTO.
Elaborado por: El investigador.

2) Crear función para agregar estadísticas del partido al DTO.

```
private StatsDTO setStatistics(Long teamLocal, Long teamVisitant, CardDetail cardDetail) {
    StatsDTO statsDTO = new StatsDTO();
    StatsDetailDTO local = setStatsDetailDTO(teamLocal, cardDetail);
    StatsDetailDTO visitant = setStatsDetailDTO(teamVisitant, cardDetail);

    if (local != null && visitant != null) {
        statsDTO.setLocal(local);
        statsDTO.setVisitant(visitant);
        statsDTO.setDraw(new DrawDTO(Integer.valueOf(8),
                (100 - ((local.getProbability() == null ? Integer.valueOf(0) : local.getProbability())
                        + (visitant.getProbability() == null ? Integer.valueOf(0) : visitant.getProbability()))),
                Integer.valueOf(10)));
    }

    return statsDTO;
}
```

Figura 114. Crear función setStatistics.
Elaborado por: El investigador.

La función agrega estadísticas del partido y retorna el DTO que se agrega en el servicio.

3) Agregar estadísticas en el DTO que retorna el servicio.

Agrega al DTO retorno las estadísticas del partido.

```
cardDetailDTO.setShowStats(applyStatistics);
cardDetailDTO.setStats(applyStatistics ? setStatistics(localId, visitantId, cardDetail) : null);
cardDetailDTO.setLocalBadgeImgUrl(cardDetail.getLocalTeamId().getLogoUrl());
cardDetailDTO.setVisitantBadgeImgUrl(cardDetail.getVisitorTeamId().getLogoUrl());
```

Figura 115. Actualizar DTO retorno getCardValid (estadísticas).
Elaborado por: El investigador.

- **Manejo de comentarios por partido**

- **Crear entidad *CardDetailComments.***

La lógica de negocio determina que cada partido puede tener asignado comentarios.

1) Crear entidad *CardDetailComments.*

CardDetailComments.java

```java
package ec.com.alquimiasoft.balonazo.model;

import java.io.Serializable;

@Entity
@Table(name = "blz_card_detail_comments")
@NoArgsConstructor
@AllArgsConstructor
@Data
public class CardDetailComments implements Serializable {
    private static final long serialVersionUID = 7600809419478605538L;

    @EmbeddedId
    CardDetailCommentsId pKey;

    @NotNull
    private LocalDate creationDate;

    @ColumnDefault("'1'")
    private Boolean active;

    @NotNull
    private String phrase;

    @NotNull
    @ManyToOne
    @JoinColumn(name = "journalist_id")
    private AdminUser journalist;


}
```

Figura 116. Crear entidad CardDetailComments.
Elaborado por: El investigador.

- **Actualizar servicio *getCardValid*.**

Actualizar el servicio *getCardValid* para agregar comentarios en el DTO de respuesta del servicio.

1) Crear función para cargar lista comentarios en un List DTO como se muestra en la figura 117.
2) Actualizar DTO retorno del servicio con lista de comentarios como se muestra en la figura 118.

```java
private List<CommentsDTO> loadComments(List<CardDetailComments> commentsList, Integer id) {

    List<CommentsDTO> commentsDTOList = new ArrayList<>();

    commentsList.stream().filter(c -> c.getPKey().getCardDetail().getPKey().getId() == id).forEach(item -> {
        commentsDTOList.add(new CommentsDTO(
                item.getPKey().getId(), item.getJournalist().getId(), item.getJournalist().getFirstName()
                        .concat(Constants.SPACE).concat(item.getJournalist().getLastName()),
                item.getPhrase(), true));
    });
    return commentsDTOList.isEmpty() ? null : commentsDTOList;
}
```

Figura 117. Crear función loadComments.
Elaborado por: El investigador.

```
cardDetailDTO.setVisitantPetImgUrl(cardDetail.getVisitorTeamId().getPetUrl());
cardDetailDTO.setComments(loadComments(commentsList, cardDetail.getPKey().getId()));

return cardDetailDTO;
```

Figura 118. Actualizar DTO retorno getCardValid (comentarios).
Elaborado por: El investigador.

- **Sprint 5**

- **Administrar proceso crear cartilla**

Actividades:

- **Crear DTO para CRUD en cartilla.**
  1) Crear clase *EditCardRequestDTO* como se muestra en la figura 119.
  2) Crear clase *EditCardDetailRequestDTO* como se muestra en la figura 120.

EditCardRequestDTO.java

```
1  package ec.com.alquimiasoft.balonazo.dto;
2
3  import java.time.LocalDateTime;
9
10 @Data
11 @NoArgsConstructor
12 @AllArgsConstructor
13 public class EditCardRequestDTO {
14     private Long id;
15
16     private LocalDateTime dateFrom;
17
18     private LocalDateTime dateTo;
19
20     private Boolean active;
21
22     private String message;
23
24     private String description;
25
26     private List<EditCardDetailRequestDTO> matchesList;
27
28 }
```

Figura 119. Crear clase EditCardRequestDTO.
Elaborado por: El investigador.

EditCardDetailRequestDTO.java

```java

@Data
@NoArgsConstructor
@AllArgsConstructor
public class EditCardDetailRequestDTO {
    private Integer id;

    private Long localId;
    private Long visitantId;

    private Long admUserId;

    private Integer localScore;
    private Integer visitantScore;

    private LocalDateTime matchDateHour;

    private Boolean active;

    private Long matchTypeId;
    private Long stadiumId;
    private Integer phraseId;
    private String phrase;
    private Boolean enableStats;
}
```

Figura 120. Crear clase EditCardDetailRequestDTO.
Elaborado por: El investigador.

- **Crear servicio para crear nueva cartilla.**

  1) Crear servicio *createCard* en *CardService.*

  El servicio permite crear una nueva cartilla para apostar en el juego. El servicio también valida la que la data necesaria para el proceso no sea nula.

```java
    @Transactional
    @Override
    public ResponseEntity<DefaultResponse> createCard(EditCardRequestDTO editCardRequestDTO, LocalDateTime dateTime,
            String authorizationToken) {

        adminUserOperations.validateAdminUser(authorizationToken);
        HttpHeaders httpHeaders = new HttpHeaders();
        DefaultResponse defaultResponse = new DefaultResponse("OK");

        LocalDateTime minMatchDateHour = validCardData(editCardRequestDTO, dateTime);

        Card card = new Card();
        card.setActive(true);
        card.setCreationDate(LocalDate.now());
        card.setDateFrom(editCardRequestDTO.getDateFrom());
        card.setStatus(CardStatusEnum.CURRENT.getCode());
        card.setDateTo(minMatchDateHour);
        card.setNumber(Integer.valueOf(2));
        card.setDescription(editCardRequestDTO.getDescription());
        card = this.saveCard(card);
```

Figura 121. Servicio createCard – parte 1.
Elaborado por: El investigador.

En la figura 121 se encuentra:

@Transactional: Anotación spring para revertir los cambios realizados en la base de datos en caso de producirse un error en el servicio padre y las funciones hijas.

Línea 115: Código para validar el access token de un usuario administrador.

Línea 125: Código para colocar en estado actual a la cartilla creada.

```
        for (EditCardDetailRequestDTO editCardDetailRequestDTO : editCardRequestDTO.getMatchesList()) {
            Map<Integer, Object> values = getValues(editCardDetailRequestDTO);
            Team localTeamId = (Team) values.get(LOCAL_TEAM_KEY);
            Team visitorTeamId = (Team) values.get(VISITOR_TEAM_KEY);
            CatalogDetail matchType = (CatalogDetail) values.get(MATCH_TYPE_KEY);
            CatalogDetail stadium = (CatalogDetail) values.get(STADIUM_KEY);

            CardDetail cardDetail = setCardDetailData(card, editCardDetailRequestDTO, localTeamId, visitorTeamId,
                    matchType, stadium, editCardRequestDTO.getMatchesList().indexOf(editCardDetailRequestDTO) + 1);

            cardDetail = cardDetailOperations.saveCardDetail(cardDetail);

            CardDetailStats cardDetailStatsVisitor = new CardDetailStats(
                    new CardDetailStatsId(Integer.valueOf(2), cardDetail.getPKey().getCard().getId(),
                            cardDetail.getPKey().getId()),
                    LocalDate.now(), true, visitorTeamId, Integer.valueOf(1), Integer.valueOf(1));

            cardDetailStatsRepository.saveAndFlush(cardDetailStatsVisitor);
        }

        return new ResponseEntity<>(defaultResponse, httpHeaders, HttpStatus.OK);

    }
```

Figura 122. Servicio createCard – parte 2.
Elaborado por: El investigador.

- **Exponer servicio *createCard.***

  1) Crear end point */create* en *CardController.*

```
public static final String CREATE_CARD_URI = "/create";

@PostMapping(CREATE_CARD_URI)
public ResponseEntity<DefaultResponse> createCard(@RequestBody EditCardRequestDTO editCardRequestDTO,
        @RequestHeader(value = "Authorization") String authorizationToken) {
    return cardOperations.createCard(editCardRequestDTO, LocalDateTime.now(), authorizationToken);
}
```

Figura 123. Crear end point /create.
Elaborado por: El investigador.

- **Administrar cierre de cartilla**

Actividades:

- **Crear servicio para actualizar el marcador de los partidos en la cartilla.**

  1) Crear función *updateScores.*

  La función obtiene los parámetros del detalle de la cartilla y coloca los marcadores en el partido correspondiente.

```
private void updateScores(SaveCardRequestDTO saveCardRequestDTO, Card card) {
    List<CardDetail> cardDetailList = card.getCardDetailList();
    cardDetailList.forEach(item -> {
        List<SaveCardDetailRequestDTO> detail = saveCardRequestDTO.getMatches().stream()
                .filter(p -> p.getId() == item.getPKey().getId()).collect(Collectors.toList());
        cardDetailList.get(cardDetailList.indexOf(item)).setLocalScore(detail.get(0).getLocalScore());
        cardDetailList.get(cardDetailList.indexOf(item)).setVisitorScore(detail.get(0).getVisitantScore());
    });
}
```

Figura 124. Crear función updateScores.
Elaborado por: El investigador.

2) Crear servicio *saveCardResults* en *CardService.*

El servicio actualiza el marcador de los partidos en la cartilla y actualiza el estado de la cartilla a listo para procesar.

```
@Override
@Transactional
public ResponseEntity<DefaultResponse> saveCardResults(SaveCardRequestDTO saveCardRequestDTO,
        String authorizationToken) {

    DefaultResponse defaultResponse = new DefaultResponse("OK");
    HttpHeaders httpHeaders = new HttpHeaders();
    adminUserOperations.validateAdminUser(authorizationToken);

    Optional<Card> value = cardRepository.findById(saveCardRequestDTO.getId());
    if (value.isPresent()) {
        Card card = value.get();
        updateScores(saveCardRequestDTO, card);
        card.setStatus(CardStatusEnum.READY_FOR_PROCESS.getCode());
        cardRepository.saveAndFlush(card);

        return new ResponseEntity<>(defaultResponse, httpHeaders, HttpStatus.OK);
    }

    return new ResponseEntity<>(defaultResponse, httpHeaders, HttpStatus.PRECONDITION_REQUIRED);
}
```

Figura 125. Crear servicio saveCardResults.
Elaborado por: El investigador.

- **Exponer servicio saveCardResults.**

1) Crear end point */save* en *CardController.*

```
public static final String CARD_SAVE_URI = "/save";

@PostMapping(CARD_SAVE_URI)
public ResponseEntity<DefaultResponse> saveCardResults(@RequestBody SaveCardRequestDTO saveCardRequestDTO,
        @RequestHeader(value = "Authorization") String authorizationToken) {
    return cardOperations.saveCardResults(saveCardRequestDTO, authorizationToken);
}
```

Figura 126. Crear end point /save en CardController.
Elaborado por: El investigador.

- **Procesar resultados**

Actividades:

- **Crear query en para obtener los pronósticos no procesados de una cartilla.**
  1) Crear query en *PrognosticRepository* para obtener los pronósticos no procesados de una cartilla.

```
@Query("SELECT p FROM Prognostic p "
        + "WHERE p.active = true AND p.card.id = :lastCardId AND p.processed = false")
List<Prognostic> getNotProccesedPrognostics(@Param("lastCardId") Long lastCardId);
```

Figura 127. Query getNotProccessPrognostics.
Elaborado por: El investigador.

- **Crear función para determinar aciertos en pronósticos de los usuarios.**
  1) Crear función para validar el resultado del partido con el valor guardado en el pronóstico.

  La función verifica si el valor del pronóstico es igual al resultado del partido y coloca *true* si es acierto y *false* caso contrario.

```
private Boolean isValidPrognosticValue(String prognosticValue, Integer localScore, Integer visitorScore) {
    return ((localScore > visitorScore && PrognosticValueEnum.LOCAL_WINNER.getCode().equals(prognosticValue))
            || (visitorScore > localScore && PrognosticValueEnum.VISITOR_WINNER.getCode().equals(prognosticValue))
            || (visitorScore == localScore && PrognosticValueEnum.EVEN.getCode().equals(prognosticValue)));
}
```

Figura 128. Validar prognostic value.
Elaborado por: El investigador.

  2) Crear función *processPrognosticMatches.*

  La función recorre el detalle de los pronósticos para validar los resultados.

```
private void processPrognosticMatches(Prognostic p) {
    List<PrognosticDetail> prognosticDetailList = p.getPrognosticDetailList();
    prognosticDetailList.forEach(item -> prognosticDetailList.get(prognosticDetailList.indexOf(item))
            .setMatch(isValidPrognosticValue(item.getPrognosticValue(), item.getCardDetail().getLocalScore(),
                    item.getCardDetail().getVisitorScore())));
    p.setProcessed(true);
}
```

Figura 129. Crear función procesar partidos pronósticos.
Elaborado por: El investigador.

- **Crear servicio para procesar los pronósticos de la cartilla cerrada.**
  1) Crear servicio *processPrognostics* en *PrognosticService.*

  El servicio obtiene la última cartilla en estado lista para procesar y procesa los pronósticos de esa cartilla.

```java
@Transactional
public Boolean processPrognostics() {
    System.out
            .println("INITIATE BATCH PROCESS".concat(this.getClass().getSimpleName()).concat("processPrognostics"));
    Boolean processDone = false;

    Long processCardId = cardRepository.getMaxCardId(CardStatusEnum.READY_FOR_PROCESS.getCode());
    System.out.println("processCardId=".concat(processCardId == null ? "null" : processCardId.toString())
            .concat(this.getClass().getSimpleName()).concat("processPrognostics"));
    if (processCardId != null) {
        List<Prognostic> prognosticList = prognosticRepository.getNotProccesedPrognostics(processCardId);

        prognosticList.forEach(p -> {
            processPrognosticMatches(p);
            prognosticRepository.saveAndFlush(p);

        });

        Card card = cardRepository.getCardInfo(processCardId);
        card.setStatus(CardStatusEnum.PROCESSED.getCode());
        cardRepository.saveAndFlush(card);
        processDone = true;

    }

    return processDone;
}
```

Figura 130. Crear servicio processPrognostics.
Elaborado por: El investigador.

- **Crear *Schedule* para el procesamiento de pronósticos automático.**

1) Crear package *Schedule.*

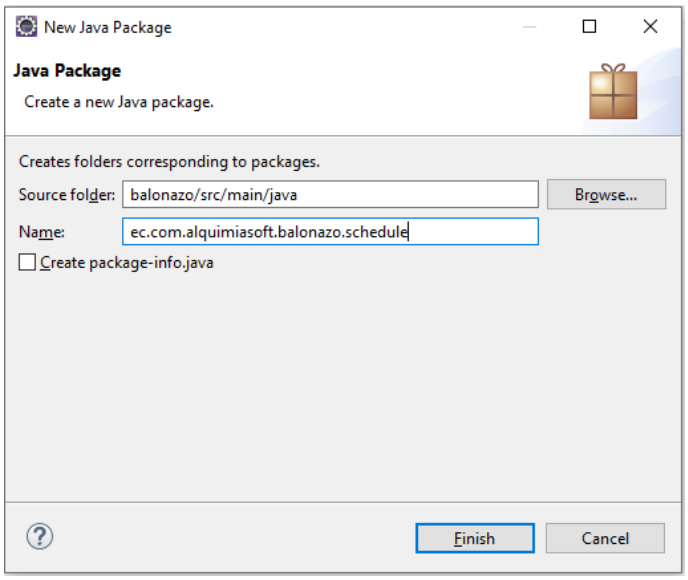


Figura 131. Crear package schedule.
Elaborado por: El investigador.

2) Crear componente *batch.*

Un componente forma parte del ciclo de vida del proyecto desde su ejecución.

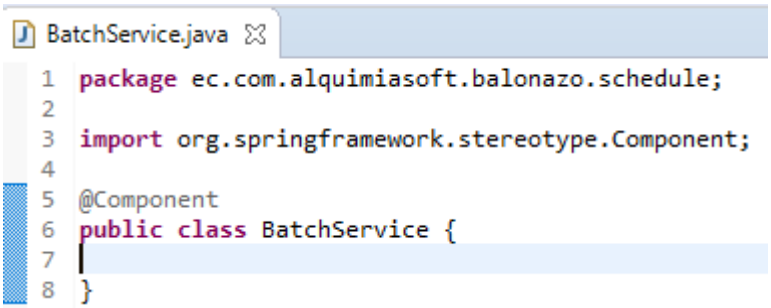


Figura 132. Crear componente batch.
Elaborado por: El investigador.

En la figura 132 se encuentra:

@Component: Anotación spring que indica que la clase es un componente.

3) Crear *Schedule* para procesar pronósticos.

Un *Schedule* ejecuta uno o varios procesos cada cierto tiempo. *Schedule* permite establecer parámetros de tiempo para sus periodos de ejecución (cada hora, cada día, cada mes, etc.)

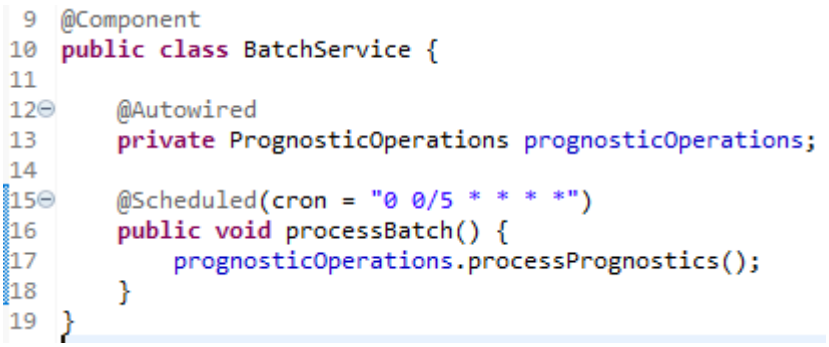


Figura 133. Crear Schedule processBatch.
Elaborado por: El investigador.

En la imagen anterior se encuentra:

@Scheduled: Anotación spring que indica que la función es un Schedule.

Línea 15: Código para establecer el parámetro CRON (representación de instantes de tiempo) para que el *Schedule* se ejecute cada 5 minutos, todos los días del mes, todos los meses y durante todos los años.

- **Sprint 6**

- **Sistema de notificaciones**

Actividades:

- **Crear entidad Notification.**

La lógica de negocio establece que debe existir un sistema de notificaciones para permitirle al usuario reclamar recompensas o notificar algún otro evento en el futuro.

La lógica de negocio establece que existen 3 tipos de notificaciones: Pop Up, Table y Notification, donde:

Pop Up: Notificación que se presenta al usuario y necesita una interacción del usuario para cerrarla.

Table: Notificaciones que recibe Front End y los muestra en una sección de notificaciones y requieren una acción del usuario para cerrarlas.

Notification: Notificaciones rápidas que no requieren interacción del usuario.

Para el sistema de notificaciones se creará una entidad ***Notification*** y su correspondiente detalle ***NotificationDetail,*** tal y como se muestra en las figuras 134 y 135**.**

En la figura 134 se encuentra la notación *@columnDefault* indicando que la columna tiene un valor por defecto.

En la figura 135 se encuentra la notación *@EmbeddedId* indicando que la columna clave primaria es una clave compuesta.

Notification.java

```java
package ec.com.alquimiasoft.balonazo.model;

import java.io.Serializable;

@Entity
@Table(name = "blz_notification")
@NoArgsConstructor
@AllArgsConstructor
@Data
public class Notification implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @SequenceGenerator(name = "pk_sequence_notif", sequenceName = "blz_notif_seq", allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "pk_sequence_notif")
    private Long id;
    @NotNull
    private LocalDate creationDate;

    private LocalDate untilDate;

    @ColumnDefault("'1'")
    private Boolean active;

    @NotNull
    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    private Integer notificationType;
    private Integer messageType;
    private String text;
    private Integer oldValue;
    private Integer newValue;
    private Integer value;
    private String navigationLink;
    private String title;
    private Integer cause;

}
```

Figura 134. Crear entidad Notification.
Elaborado por: El investigador.

NotificationDetail.java

```java
package ec.com.alquimiasoft.balonazo.model;

import java.io.Serializable;

@Entity
@Table(name = "blz_notification_detail")
@NoArgsConstructor
@AllArgsConstructor
@Data
public class NotificationDetail implements Serializable {

    private static final long serialVersionUID = 8978451540355483923L;

    @EmbeddedId
    private NotificationDetailId pKey;

    @NotNull
    private Integer hits;

    @NotNull
    private Integer code;

}
```

Figura 135. Crear entidad NotificationDetail.
Elaborado por: El investigador.

- **Crear Servicio Notification.**

1) Crear interfaz ***NotificationRepository***

Para los servicios de notificaciones se debe crear la interfaz repository para la manipulación de la información de la base de datos.

NotificationRepository.java

```
1  package ec.com.alquimiasoft.balonazo.repository;
2
3  import java.time.LocalDate;
12
13 @Repository
14 public interface NotificationRepository extends JpaRepository<Notification, Long> {
15     @Query("SELECT n FROM Notification n WHERE n.active = true AND n.user.id = :user AND n.notificationType = :notificationType "
16             + "AND current_date BETWEEN n.creationDate AND COALESCE(n.untilDate,current_date) ORDER BY n.id DESC")
17     List<Notification> findByUserId(@Param("user") Long user, @Param("notificationType") Integer notificationType);
18
19     @Query("SELECT DISTINCT n.id, n.user.email, n.user.firstName, n.user.username, n.text FROM Notification n WHERE n.active = true "
20             + "AND n.notificationType = :notificationType AND current_date BETWEEN n.creationDate AND COALESCE(n.untilDate,current_date)")
21     List<Object> getEmailWinners(@Param("notificationType") Integer notificationType);
22
23     @Query("SELECT n FROM Notification n WHERE n.active = :active AND n.user.id = :user AND n.notificationType = :notificationType "
24             + "AND :creationDate BETWEEN n.creationDate AND COALESCE(n.untilDate,current_date)")
25     List<Notification> findNotification(@Param("active") Boolean active, @Param("user") Long user,
26             @Param("notificationType") Integer notificationType, @Param("creationDate") LocalDate creationDate);
27
28     @Query("SELECT n FROM Notification n WHERE (n.active = :active OR :active is null) AND n.user.id = :user "
29             + "AND n.notificationType = :notificationType AND n.creationDate BETWEEN :dateFrom AND :dateTo")
30     List<Notification> findNotificationByDateRange(@Param("active") Boolean active, @Param("user") Long user,
31             @Param("notificationType") Integer notificationType, @Param("dateFrom") LocalDate dateFrom,
32             @Param("dateTo") LocalDate dateTo);
33
34 }
```

Figura 136. Crear interfaz NotificationRepository.
Elaborado por: El investigador

2) Crear servicio ***NotificationService***

Antes de crear la clase ***NotificationService*** se debe crear la interfaz ***NotificationOperations*** para declarar las funciones a implementar.

En la imagen 137 se encuentra:

Línea 46: Función para retornar un DTO con la información de las notificaciones de tipo ***Pop Up*** y ***Table***.

Línea 63: Función para retornar un DTO con la información de las notificaciones de tipo ***Notification***.

```java
@Service
public class NotificationService implements NotificationOperations {

    @Autowired
    private NotificationRepository notificationRepository;

    @Autowired
    private UserOperations userOperations;

    @Autowired
    private AdminUserOperations adminUserOperations;

    @Override
    public NotificationDTO getPopupMessagesForUser(Notification notification) {
        NotificationDTO notificationDTO = new NotificationDTO();
        notificationDTO.setId(notification.getId());
        notificationDTO.setTitle(notification.getTitle() != null ? notification.getTitle()
                : MessageManager.getMessages(Messages.POPUP_TITLE));
        notificationDTO.setDescription(notification.getText());
        notificationDTO.setIconImage(MessageManager.getMessages(Messages.POPUP_ICONIMAGE));
        notificationDTO.setNavigationLink(notification.getNavigationLink() != null ? notification.getNavigationLink()
                : MessageManager.getMessages(Messages.POPUP_NAVIGATIONLINK));
        notificationDTO.setNavigationLinkTitle(MessageManager.getMessages(Messages.POPUP_NAVIGATION_LINK_TITLE));
        notificationDTO.setNotificationType(notification.getNotificationType());
        notificationDTO.setMessageType(notification.getMessageType());
        notificationDTO.setValue(notification.getValue());
        return notificationDTO;
    }

    @Override
    public NotificationDTO getNotificationMessagesForUser(Notification notification) {
        NotificationDTO notificationDTO = new NotificationDTO();
        notificationDTO.setId(notification.getId());
        notificationDTO.setTitle(MessageManager.getMessages(Messages.NOTIFICATION_TITLE));
        notificationDTO.setDescription(notification.getText());
        notificationDTO.setIconImage(MessageManager.getMessages(Messages.NOTIFICATION_ICONIMAGE));
        notificationDTO.setNavigationLink(MessageManager.getMessages(Messages.NOTIFICATION_NAVIGATIONLINK));
        notificationDTO.setNavigationLinkTitle(MessageManager.getMessages(Messages.NOTIFICATION_NAVIGATION_LINK_TITLE));
        notificationDTO.setNotificationType(notification.getNotificationType());
        notificationDTO.setMessageType(notification.getMessageType());
        notificationDTO.setValue(notification.getValue());
        return notificationDTO;
    }
```

Figura 137. Crear servicio NotificationService – parte 1.
Elaborado por: El investigador.

```java
    @Override
    public ResponseEntity<List<NotificationDTO>> getAllNotifications(String authorizationToken) {
        User user = userOperations.validateUser(authorizationToken);

        List<NotificationDTO> notificationList = new ArrayList<>();
        notificationRepository.findByUserId(user.getId(), NotificationTypeEnum.POPUP.getCode()).forEach(n -> {
            notificationList.add(getPopupMessagesForUser(n));
        });

        notificationRepository.findByUserId(user.getId(), NotificationTypeEnum.TABLE.getCode()).forEach(n -> {
            notificationList.add(getPopupMessagesForUser(n));
        });

        notificationRepository.findByUserId(user.getId(), NotificationTypeEnum.NOTIFICATION.getCode()).forEach(n -> {
            notificationList.add(getNotificationMessagesForUser(n));
            n.setActive(false);
            saveNotification(n);
        });

        System.out.println("Get all notifications " + this.getClass().getSimpleName() + "getAllNotifications");

        return new ResponseEntity<>(notificationList, new HttpHeaders(), HttpStatus.OK);
    }
```

Figura 138. Crear servicio NotificationService – parte 2.
Elaborado por: El investigador.

En la imagen 138 se encuentra:

Línea 78: Servicio para obtener la lista de notificaciones pendientes de un usuario determinado. El servicio retorna todos los tipos de notificaciones en una lista de DTOs y Front End se encarga de filtrarlos.

3) Crear servicio para aplicar notificaciones.

El servicio para aplicar notificaciones validará que la notificación a aplicar se encuentre en estado válido. De ejecutarse con éxito el servicio retorna un estado OK, caso contrario un estado un código de error.

Las notificaciones tienen un estado para identificar si el usuario ya ha interactuado con ellas. El servicio para aplicar notificaciones permite cerrar la notificación y en caso de ser una notificación con bonus de balones, asignarlos al usuario.

```
123    @Override
124    public ResponseEntity<ApplyNotificationResponseDTO> applyBalloonsNotifications(
125            List<NotificationDTO> notificationsList, String authorizationToken) {
126        if (notificationsList != null && !notificationsList.isEmpty()) {
127            User user = userOperations.validateUser(authorizationToken);
128
129            Logger.getAnonymousLogger()
130                    .info("VERIFY NOTIFICATIONS" + this.getClass().getSimpleName() + "applyBalloonsNotifications");
131            List<Notification> notificationRepositoryList = notificationRepository.findNotification(true, user.getId(),
132                    NotificationTypeEnum.POPUP.getCode(), LocalDate.now());
133
134            List<Notification> tableNotificationList = notificationRepository.findNotification(true, user.getId(),
135                    NotificationTypeEnum.TABLE.getCode(), LocalDate.now());
136
137            notificationRepositoryList.forEach(n -> {
138                Logger.getAnonymousLogger().info("notification ID:".concat(n.getId().toString())
139                        + this.getClass().getSimpleName() + "applyBalloonsNotifications");
140                notificationsList.stream().filter(p -> p.getId().equals(n.getId())).forEach(q -> {
141
142                    Logger.getAnonymousLogger().info("applying notification ID:".concat(q.getId().toString())
143                            + this.getClass().getSimpleName() + "applyBalloonsNotifications");
144                    if (validateCondition(n)) {
145                        n.setOldValue(user.getGoldBalls());
146                        user.setGoldBalls(user.getGoldBalls() + q.getValue());
147                        n.setNewValue(user.getGoldBalls());
148                    }
149                    n.setActive(false);
150
151                    userOperations.saveUserData(user);
152                    saveNotification(n);
153                });
154            });
```

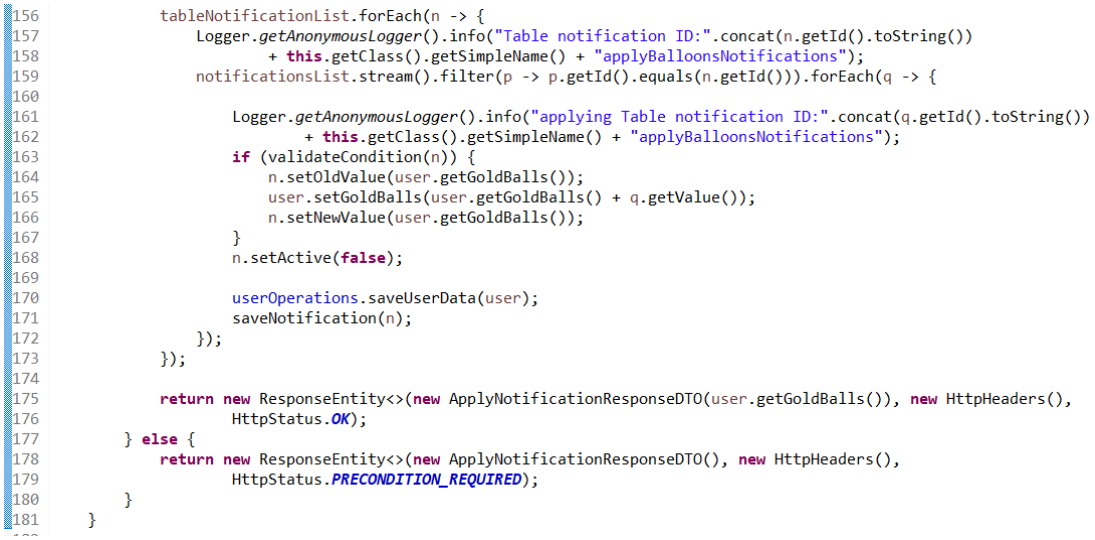
Figura 139. Crear servicio Aplicar Notificaciones – parte 1.
Elaborado por: El investigador.

En la imagen 139 se encuentra:

Línea 126: Código para verificar si la lista de notificaciones a cerrar o aplicar no se encuentre vacía o sea nula. En caso no cumplirse las condiciones retorna un estado de precondiciones requeridas.

Línea 131 y 134: Código para obtener las listas de notificaciones de tipo Pop Up y Table del usuario y aplicar un filtro de búsqueda con las notificaciones provenientes de la petición.

Líneas 144 a 148: Valida que las notificaciones sean de tipo bonus y asignar el número de balones correspondientes al usuario.



```
156             tableNotificationList.forEach(n -> {
157                 Logger.getAnonymousLogger().info("Table notification ID:".concat(n.getId().toString())
158                         + this.getClass().getSimpleName() + "applyBalloonsNotifications");
159                 notificationsList.stream().filter(p -> p.getId().equals(n.getId())).forEach(q -> {
160 
161                     Logger.getAnonymousLogger().info("applying Table notification ID:".concat(q.getId().toString())
162                             + this.getClass().getSimpleName() + "applyBalloonsNotifications");
163                     if (validateCondition(n)) {
164                         n.setOldValue(user.getGoldBalls());
165                         user.setGoldBalls(user.getGoldBalls() + q.getValue());
166                         n.setNewValue(user.getGoldBalls());
167                     }
168                     n.setActive(false);
169 
170                     userOperations.saveUserData(user);
171                     saveNotification(n);
172                 });
173             });
174 
175             return new ResponseEntity<>(new ApplyNotificationResponseDTO(user.getGoldBalls()), new HttpHeaders(),
176                     HttpStatus.OK);
177         } else {
178             return new ResponseEntity<>(new ApplyNotificationResponseDTO(), new HttpHeaders(),
179                     HttpStatus.PRECONDITION_REQUIRED);
180         }
181     }
```

Figura 140. Crear servicio Aplicar Notificaciones – parte 2.
Elaborado por: El investigador.

En la imagen 140 se encuentra:

Línea 156: Código para filtrar de la lista de notificaciones tipo table las notificaciones a cerrar que provienen de la petición y validarlas para proceder a aplicar la asignación de balones al usuario.

- **Exponer servicios de notificaciones.**

Para exponer los servicios que puede consumir Front End se procede a crear el controlador ***NotificationController.***

Los **end points** requieren de un access token para validar las peticiones de los usuarios. En cada en point que requiere un token se especifica el parámetro @RequestHeader en donde se encuentra el access y refresh token enviado desde el sistema Front.

```java
public class NotificationController {
    public static final String NOTIFICATIONS_URI = "/notif/all";
    public static final String NOTIFICATIONS_APPLY_URI = "/notif/apply";
    public static final String SURVEY_URI = "/notif/survey";

    @Autowired
    private NotificationOperations notificationOperations;

    @GetMapping(NOTIFICATIONS_URI)
    public ResponseEntity<List<NotificationDTO>> getAllNotifications(
            @RequestHeader(value = "Authorization") String authorizationToken) {
        return notificationOperations.getAllNotifications(authorizationToken);

    }

    @PostMapping(NOTIFICATIONS_APPLY_URI)
    public ResponseEntity<ApplyNotificationResponseDTO> applyNotifications(
            @RequestBody List<NotificationDTO> notificationsList,
            @RequestHeader(value = "Authorization") String authorizationToken) {
        return notificationOperations.applyBalloonsNotifications(notificationsList, authorizationToken);

    }

    @PostMapping(SURVEY_URI)
    public ResponseEntity<DefaultResponse> saveSurveyNotifications(@RequestBody NotificationDTO notificationDTO,
            @RequestHeader(value = "Authorization") String authorizationToken) {
        return notificationOperations.saveNotifications(notificationDTO.getNavigationLink(), notificationDTO.getTitle(),
                notificationDTO.getDescription(), notificationDTO.getValue(), notificationDTO.getDateTo(),
                PopupTypeEnum.SURVEY_NOTIFICATION.getCode(), authorizationToken);

    }
}
```

Figura 141. Crear controlador NotificationController.
Elaborado por: El investigador.

- **Asignación de balones a ganadores**

Actividades:

- **Proceso de asignación de balones a usuarios por pronósticos**

Para la asignación de balones de oro a los usuarios que han acertado en sus pronósticos, se debe crear una notificación de tipo ***Table*** para que el usuario reclame su premio y el proceso de asignación de balones sea transparante.

1) Crear función para crear notificación por acierto.

La función se crea en la clase ***PrognosticService.***

```
394    private void verifyPopupInWebPage(Prognostic p, MembershipHits m) {
395        Integer goldBalloonsGift = m.getGoldBalloonsGift() == null ? Integer.valueOf(0) : m.getGoldBalloonsGift();
396        if (m.getShowPopup() && goldBalloonsGift > 0) {
397            Logger.getAnonymousLogger()
398                    .info("Prognostic ".concat(p.getId().toString()).concat(" Apply balloons gift of ").concat(
399                            goldBalloonsGift.toString()) + this.getClass().getSimpleName() + "processPrognostics");
400
401            Notification notification = new Notification();
402            notification.setCreationDate(LocalDate.now());
403            notification.setActive(true);
404            notification.setUser(p.getUser());
405            notification.setNotificationType(NotificationTypeEnum.TABLE.getCode());
406            notification.setMessageType(PopupTypeEnum.EARNED_BALLOONS.getCode());
407            notification.setValue(goldBalloonsGift);
408            if (goldBalloonsGift == Integer.valueOf(1)) {
409                notification
410                        .setText(MessageManager.formatMessages(MessageManager.getMessages(Messages.POPUP_DESCRIPTION_S),
411                                String.format("%04d", p.getCode()), p.getCard().getDescription(), goldBalloonsGift));
412            } else {
413                notification
414                        .setText(MessageManager.formatMessages(MessageManager.getMessages(Messages.POPUP_DESCRIPTION_P),
415                                String.format("%04d", p.getCode()), p.getCard().getDescription(), goldBalloonsGift));
416            }
417
418            notificationRepository.saveAndFlush(notification);
419        }
420    }
```

Figura 142. Crear notificaciones por acierto.
Elaborado por: El investigador.

En la función para crear notificaciones por acierto se encuentra:

Líneas 401 a 407: Código para crear una notificación y settear los datos necesarios como el usuario al que pertenece, el título y el tipo Table.

Líneas 408 a 416: Código para asignar a la notificación el texto de la notificación. Verifica si es un bonus single o no y establece un mensaje en base a ello.

2) Actualizar servicio para procesar cartillas.

Actualiza en el servicio ***processPrognostics.***

```
232            prognosticList.forEach(p -> {
233
234                processPrognosticMatches(p);
235
236                List<MembershipHits> membershipHitsList = getHitsParam(p);
237
238                Logger.getAnonymousLogger()
239                        .info("prognostic: ".concat(Long.toString(p.getId())).concat(" matches -> ")
240                                .concat(Long.toString(
241                                        p.getPrognosticDetailList().stream().filter(progd -> progd.getMatch()).count()))
242                                + this.getClass().getSimpleName() + "processPrognostics");
243
244                membershipHitsList.forEach(m -> {
245                    Logger.getAnonymousLogger()
246                            .info("Prognostic ".concat(p.getId().toString()).concat(" --> apply prize for ")
247                                    .concat(m.getHitsNumber().toString()).concat(" hits ")
248                                    + this.getClass().getSimpleName() + "processPrognostics");
249
250                    verifyPopupInWebPage(p, m);
251                    p.setApplyPrize(true);
252                });
253
254                prognosticRepository.saveAndFlush(p);
255
256            });
```

Figura 143. Asignar balones a ganadores.
Elaborado por: El investigador.

En la imagen 143 se encuentra:

Línea 232: Código para recorrer la lista de pronósticos procesados.

Línea 250: Código para verificar si el pronóstico aplica a alguna recompensa por aciertos.

- **Servicio para obtener resultados**

Actividades:

- **Crear servicios para obtener resultados**

La lógica de negocio establece que debe existir un servicio que retorne los resultados de las cartillas enviadas por el usuario en base al número de aciertos.

1) Crear servicio ***ResultService.***
2) Crear servicio para obtener los resultados de la última cartilla procesada como se muestra en la figura 144.

El servicio obtiene los pronósticos enviados por el usuario en la última cartilla procesada y las retorna en un DTO.

```
31 @Service
32 public class ResultsService implements ResultsOperations {
33
34     @Autowired
35     private CardRepository cardRepository;
36
37     @Autowired
38     private UserOperations userOperations;
39
40     @Autowired
41     private PrognosticOperations prognosticOperations;
42
43     public ResultsResponseDTO getResults(ResultsRequest resultsRequest, String authorizationToken) {
44
45         ResultsResponseDTO resultsResponseDTO = new ResultsResponseDTO(new ArrayList<>(), null, null,null);
46
47         Long lastCard = cardRepository.getMaxCardId(CardStatusEnum.PROCESSED.getCode());
48
49         if (lastCard != null) {
50             cardRepository.findAll().stream()
51                     .filter(p -> p.getActive() && p.getStatus() == CardStatusEnum.PROCESSED.getCode())
52                     .sorted(Comparator.comparing(Card::getId).reversed()).limit(10).forEach(p -> {
53                         resultsResponseDTO.getCardList().add(new CardDTO(p.getId(), null, null, null, null, null, null,
54                                 null, null, lastCard == p.getId(), p.getDescription(), null, null,null,null));
55                     });
56             resultsResponseDTO.setCard(
57                     getCardResults(resultsRequest, resultsRequest.getId() != null ? resultsRequest.getId() : lastCard));
58         }
59
60         Logger.getAnonymousLogger().info("Get results "+ this.getClass().getSimpleName()+ "getResults");
61         return resultsResponseDTO;
62     }
```

Figura 144. Crear servicio getResults.
Elaborado por: El investigador.

La lógica de negocio determina que el usuario puede revisar los resultados de cartillas ya jugadas.

3) Crear servicio para obtener los resultados de una cartilla ya jugada.

El servicio obtiene los pronósticos enviados por un usuario en la cartilla que proviene del request y las retorna en un DTO.

```
64    public ResponseEntity<ResultsResponseDTO> loadPrognosticsPerResult(ResultsRequest resultsRequest, String authorizationToken) {
65
66        User user = userOperations.validateUser(authorizationToken);
67        ResultsResponseDTO resultsResponseDTO = null;
68
69        if (resultsRequest.getId() != null && resultsRequest.getPageSize() != null && resultsRequest.getPageSize() > 0
70                && resultsRequest.getPageNumber() != null && resultsRequest.getPageNumber() > 0) {
71
72            Integer count = prognosticOperations.getPrognosticCount(resultsRequest.getId(),user.getId());
73
74            int pageCount = MathUtil.getPageCount(resultsRequest.getPageSize(), count);
75
76            Integer offset = (resultsRequest.getPageNumber() - 1) * resultsRequest.getPageSize();
77
78            resultsResponseDTO = new ResultsResponseDTO(new ArrayList<>(), null, null,pageCount);
79            resultsResponseDTO.setPrognostics(prognosticOperations.loadPrognosticsForResults(user.getId(),
80                    resultsRequest.getId(), resultsRequest.getPageSize(), offset));
81
82            return new ResponseEntity<>(resultsResponseDTO, new HttpHeaders(), HttpStatus.OK);
83        }
84        return new ResponseEntity<>(resultsResponseDTO, new HttpHeaders(), HttpStatus.PRECONDITION_REQUIRED);
85    }
```

Figura 145. Servicio para obtener resultados de cartillas.
Elaborado por: El investigador.

- **Exponer servicios**

1) Crear end points para consumir los servicios.

Crear el controlador ***ResultController*** y exponer los servicios creados para resultados.

```
20 public class ResultsController {
21     public static final String RESULTS_URI = "/results";
22     public static final String PROGS_PER_RESULTS_URI = "/progresults";
23
24     @Autowired
25     private ResultsOperations resultsOperations;
26
27     @PostMapping(RESULTS_URI)
28     public ResultsResponseDTO getResults(@RequestBody ResultsRequest resultsRequest,
29             @RequestHeader(value = "Authorization") String authorizationToken) {
30         return resultsOperations.getResults(resultsRequest, authorizationToken);
31     }
32
33     @PostMapping(PROGS_PER_RESULTS_URI)
34     public ResponseEntity<ResultsResponseDTO> loadPrognosticsPerResult(@RequestBody ResultsRequest resultsRequest,
35             @RequestHeader(value = "Authorization") String authorizationToken) {
36         return resultsOperations.loadPrognosticsPerResult(resultsRequest, authorizationToken);
37     }
38
39 }
```

Figura 146. Exponer servicios para resultados.
Elaborado por: El investigador.

- **Sprint 7**

- **Servicio cartillas enviadas por usuario**

Actividades:

- **Crear servicio para obtener cartillas.**

La lógica de negocio establece que un usuario puede consultar sus pronósticos enviados en la cartilla vigente y también puede consultar los pronósticos de cartillas pasadas.

1) Crear servicio para obtener pronósticos por usuario.

El servicio obtiene los pronósticos enviados por un usuario según el ID de la cartilla que proviene del request.

```
294    public ResponseEntity<PrognosticsActiveUserResponseDTO> loadPrognosticsPerPage(CardDTO cardDTO,
295            String authorizationToken) {
296
297        User user = userOperations.validateUser(authorizationToken);
298        PrognosticsActiveUserResponseDTO prognosticsActiveUserResponseDTO = null;
299
300        if (cardDTO.getPageNumber() != null && cardDTO.getPageNumber() > 0 && cardDTO.getPageSize() != null
301                && cardDTO.getPageSize() > 0 && cardDTO.getId() != null) {
302
303            prognosticsActiveUserResponseDTO = new PrognosticsActiveUserResponseDTO();
304            Integer count = prognosticRepository.getPrognosticCountPerCardAndUser(cardDTO.getId(), user.getId());
305
306            int pageCount = MathUtil.getPageCount(cardDTO.getPageSize(), count);
307
308            Integer offset = (cardDTO.getPageNumber() - 1) * cardDTO.getPageSize();
309            List<Prognostic> prognosticList = prognosticRepository.getAllPrognostics(user.getId(), cardDTO.getId(),
310                    cardDTO.getPageSize(), offset);
311
312            prognosticsActiveUserResponseDTO.setPrognosticList(loadPrognosticResponseDTOList(prognosticList));
313            prognosticsActiveUserResponseDTO.setCount(pageCount);
314
315            return new ResponseEntity<>(prognosticsActiveUserResponseDTO, new HttpHeaders(), HttpStatus.OK);
316        }
317
318        return new ResponseEntity<>(prognosticsActiveUserResponseDTO, new HttpHeaders(),
319                HttpStatus.PRECONDITION_REQUIRED);
320    }
```

Figura 147. Crear servicio obtener cartillas por usuario.
Elaborado por: El investigador.

- **Exponer servicio.**

1) Crear end point para obtener las cartillas por usuario.

```
52    @PostMapping(PROGNOSTIC_JSON_URI)
53    @ResponseBody
54    public ResponseEntity<PrognosticsActiveUserResponseDTO> loadPrognosticsPerPage(@RequestBody CardDTO cardDTO,
55            @RequestHeader(value = "Authorization") String authorizationToken) {
56        return prognosticOperations.loadPrognosticsPerPage(cardDTO, authorizationToken);
57    }
```

Figura 148. Exponer servicio obtener cartillas.
Elaborado por: El investigador.

- **Servicio envío de correos a ganadores**

Actividades:

- **Agregar configuración spring mail**

Spring tiene servicios propios para el envío de emails y requieren una configuración que se las coloca en el archivo de propiedades.

1) Agregar configuración spring mail en Application.yml

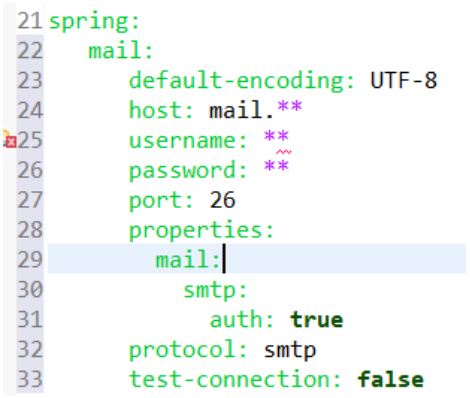


Figura 149. Agregar configuraciones spring mail.
Elaborado por: El investigador.

En la imagen 149 se encuentra:

Mail: Bloque para la configuración de spring mail con atributos tales como: host, username, password, puerto y protocolos.

- **Crear archivo de configuración para spring email**

La lógica de negocio establece que el sistema debe enviar un correo para notificar al usuario que ha ganado un premio con el acierto de sus pronósticos.

1) Crear archivo de configuración ***SpringMailConfig.***

El archivo configura el uso de plantillas para el envío de correos electrónicos.

```java
@Configuration
public class SpringMailConfig implements ApplicationContextAware, EnvironmentAware {

    public static final String EMAIL_TEMPLATE_ENCODING = "UTF-8";

    @Bean
    public SpringTemplateEngine emailTemplateEngine() {
        final SpringTemplateEngine templateEngine = new SpringTemplateEngine();
        templateEngine.addTemplateResolver(htmlTemplateResolver());
        return templateEngine;
    }

    private ITemplateResolver htmlTemplateResolver() {
        final ClassLoaderTemplateResolver templateResolver = new ClassLoaderTemplateResolver();
        templateResolver.setOrder(Integer.valueOf(2));
        templateResolver.setResolvablePatterns(Collections.singleton("html/*"));
        templateResolver.setPrefix("/mail/");
        templateResolver.setSuffix(".html");
        templateResolver.setTemplateMode(TemplateMode.HTML);
        templateResolver.setCharacterEncoding(EMAIL_TEMPLATE_ENCODING);
        templateResolver.setCacheable(false);
        return templateResolver;
    }
```

Figura 150. Archivo de configuración SpringMailConfig.
Elaborado por: El investigador.

En el archivo de configuración de la figura 150 se encuentra:

Línea 29: Función para configurar el path y la extensión de los archivos aceptados para las plantillas, en este caso .html.

Línea 23: Función para retornar la construcción de una plantilla para correos.

- **Crear servicio para enviar correos**

1) Crear servicio ***EmailService.***
2) Crear funciones para el envío de correos a ganadores como se muestra en las figuras 151 y 152.

En la imagen 151 se encuentra:

Línea 46: Código para obtener la lista de correos de los usuarios que han acertado en sus pronósticos y tienen premio en base a notificaciones generadas para el caso.

Línea 49: Código para recorrer la lista notificaciones de correos y armar el MailDTO que se envía a los servicios de spring mail.

```java
@Service
public class EmailService implements EmailOperations {
    @Value("${balonazo.contact-email}")
    private String contactEmail;

    @Value("${balonazo.email.maxsendingsperhour}")
    private Integer maxSendingsPerHour;

    @Autowired
    private NotificationRepository notificationRepository;

    @Autowired
    private NotificationDetailRepository notificationDetailRepository;

    @Autowired
    private AsyncEmailService asyncEmailService;

    @Override
    public void sendWinnersEmail(Boolean isSynchronous) {
        Logger.getAnonymousLogger().info("Sending email for winners"+ this.getClass().getSimpleName()+ "sendWinnersEmail");
        Stream<Object> notificationsList = notificationRepository.getEmailWinners(NotificationTypeEnum.EMAIL.getCode())
                .stream().limit(maxSendingsPerHour);

        notificationsList.forEach(n -> {
            Object[] obj = (Object[]) n;
            Long id = (Long) obj[0];
            String email = (String) obj[1];
            String firstName = (String) obj[2];
            String username = (String) obj[3];
            String description = (String) obj[4];
            firstName = firstName != null ? firstName : username;
            firstName = firstName.isEmpty() ? username : firstName;

            MailDTO mailDTO = new MailDTO();
            mailDTO.setFrom(MessageManager.getMessages(Messages.DEFAULT_EMAIL_ALIAS));
            mailDTO.setTo(email);
            mailDTO.setSubject(MessageManager.getMessages(Messages.EMAIL_SUBJECT));

            String content = MessageManager.formatMessages(MessageManager.getMessages(Messages.EMAIL_DESCRIPTION),
                    firstName);
```

Figura 151. Envío de correos a ganadores – parte 1.
Elaborado por: El investigador.

```java
            mailDTO.setContent(content);
            Map<String, Object> variables = new HashMap<>();
            variables.put("name", StringUtils.capitalize(firstName));
            variables.put("currentyear", Integer.toString(LocalDate.now().getYear()));
            variables.put("description", description);
            List<String> prognosticList = new ArrayList<>();

            notificationDetailRepository.getNotificationDetailById(id).forEach(item -> {
                prognosticList.add(
                        MessageManager.formatMessages(MessageManager.getMessages(Messages.NOTIFICATION_PRIZE_PER_HIT),
                                String.format("%04d", item.getCode()), item.getHits()));
            });

            variables.put("prognosticlist", prognosticList);
            if (isSynchronous) {
                asyncEmailService.sendMailWithTemplateSynch(variables, mailDTO, null, "html/hitsNotification");
            } else {
                asyncEmailService.sendMailWithTemplate(variables, mailDTO, null, "html/hitsNotification");
            }


            Optional<Notification> optional = notificationRepository.findById(id);
            if (optional.isPresent()) {
                Notification notif = optional.get();
                notif.setActive(false);
                notificationRepository.saveAndFlush(notif);
            }
        });

    }
```

Figura 152. Envío de correos a ganadores – parte 2.
Elaborado por: El investigador.

- **Actualizar servicio procesar cartillas.**

1) Crear función para generar notificaciones de tipo Email.

La función crea una notificación de tipo Email para enviar los correos a ganadores en un proceso asíncrono.

```
private void verifyEmailNotification(Prognostic p, MembershipHits m) {
    List<Notification> emailNotification = notificationRepository.findByUserId(p.getUser().getId(),
            NotificationTypeEnum.EMAIL.getCode());
    Notification notification = null;

    if (m.getSendWinnersEmail()) {
        if (emailNotification.isEmpty()) {

            notification = new Notification();
            notification.setCreationDate(LocalDate.now());
            notification.setActive(true);
            notification.setUser(p.getUser());
            notification.setText(p.getCard().getDescription());
            notification.setNotificationType(NotificationTypeEnum.EMAIL.getCode());
            notification.setMessageType(MessageTypeEnum.INFORMATION.getCode());

            notification = notificationRepository.saveAndFlush(notification);
        } else {
            notification = emailNotification.get(0);
        }

        Integer recordCount = notificationDetailRepository.getCount(notification.getId());
        Long hits = p.getPrognosticDetailList().stream().filter(item -> item.getMatch()).count();
        NotificationDetail notificationDetail = new NotificationDetail(
                new NotificationDetailId(recordCount + 1, notification),
                hits.intValue(), p.getCode());

        notificationDetailRepository.saveAndFlush(notificationDetail);
    }
}
```

Figura 153. Función crear notificaciones Email.
Elaborado por: El investigador.

2) Actualizar proceso cerrar cartilla

```
membershipHitsList.forEach(m -> {
    Logger.getAnonymousLogger()
            .info("Prognostic ".concat(p.getId().toString()).concat(" --> apply prize for ")
                    .concat(m.getHitsNumber().toString()).concat(" hits ")
                    + this.getClass().getSimpleName() + "processPrognostics");

    verifyEmailNotification(p, m);
    verifiNotificationInWebPage(p, m);
    verifyPopupInWebPage(p, m);
    p.setApplyPrize(true);
});
```

Figura 154. Actualizar proceso cerrar cartilla envío de emails.
Elaborado por: El investigador.

- **Envío de correos electrónicos por registro**

Actividades:

- **Actualizar método registro Usuario.**

La lógica de negocio establece que el sistema envíe correos de bienvenida a los nuevos usuarios.

1) Crear servicio para envió de correos de bienvenida.

```
@Override
public void sendWelcomeEmail(String email, String name, Boolean isSynchronous) {
    String template = null;
    MailDTO mailDTO = new MailDTO();
    mailDTO.setFrom(MessageManager.getMessages(Messages.DEFAULT_EMAIL_ALIAS));
    mailDTO.setTo(email);
    LocalDate now = LocalDate.now();
    if (isWithinRange(now)) {
        template = "html/founderSupporterNotification";
        mailDTO.setSubject(MessageManager.getMessages(Messages.WELCOME_EMAIL_FOUNDER_SUBJECT));
    } else {
        template = "html/welcome";
        mailDTO.setSubject(MessageManager.getMessages(Messages.WELCOME_EMAIL_SUBJECT));
    }
    Map<String, Object> variables = new HashMap<>();
    variables.put("name", StringUtils.capitalize(name));
    variables.put("currentyear", Integer.toString(LocalDate.now().getYear()));

    if (isSynchronous) {
        asyncEmailService.sendMailWithTemplateSynch(variables, mailDTO, null, template);
    } else {
        asyncEmailService.sendMailWithTemplate(variables, mailDTO, null, template);
    }
}
```

Figura 155. Servicio envío de correos de bienvenida.
Elaborado por: El investigador.

2) Actualizar método de registro de un usuario.

```
userMembershipOperations.saveMembershipForUser(
        new UserMembership(null, LocalDate.now(), true, user, newMembership, LocalDateTime.now(), null));

emailOperations.sendWelcomeEmail(userDTO.getEmail().toLowerCase(), userDTO.getUsername(),false);
```

Figura 156. Enviar correo de bienvenida en registro.
Elaborado por: El investigador.

- **Sprint 8**

- **Restaurar contraseña**

Actividades:

- **Crear servicio para envío de correo con clave temporal**

```
@Override
public void sendPasswordRecoverEmail(String email, String name, String password, Boolean isSynchronous) {
    MailDTO mailDTO = new MailDTO();
    mailDTO.setFrom(MessageManager.getMessages(Messages.DEFAULT_EMAIL_ALIAS));
    mailDTO.setTo(email);
    mailDTO.setSubject(MessageManager.getMessages(Messages.RECOVER_PWD_EMAIL_SUBJECT));
    Map<String, Object> variables = new HashMap<>();
    variables.put("name", StringUtils.capitalize(name));
    variables.put("password", password);
    variables.put("currentyear", Integer.toString(LocalDate.now().getYear()));

    if (isSynchronous) {
        asyncEmailService.sendMailWithTemplateSynch(variables, mailDTO, null, "html/recoverPwd");
    } else {
        asyncEmailService.sendMailWithTemplate(variables, mailDTO, null, "html/recoverPwd");
    }
}
```

Figura 157. Crear servicio envío correo con clave temporal.
Elaborado por: El investigador.

- **Crear servicio restaurar contraseña**

La lógica de negocio establece que un usuario puede obtener una contraseña temporal para acceder al sistema si no recuerda la contraseña original.

1) Crear servicio para restaurar contraseña.

```
public ResponseEntity<DefaultResponse> recoverPassword(UserUpdateDTO userUpdateDTO) {
    DefaultResponse defaultResponse = new DefaultResponse("OK");
    User user = findByEmail(userUpdateDTO.getEmail());
    if (user == null) {
        return new ResponseEntity<>(defaultResponse, new HttpHeaders(), HttpStatus.PRECONDITION_REQUIRED);
    }

    if (user.getOrigin() != UserOriginEnum.BALONAZO.getCode()) {
        return new ResponseEntity<>(defaultResponse, new HttpHeaders(), HttpStatus.FAILED_DEPENDENCY);
    }
    String passwordGenerated = StringUtil.generateRandomString(8, 15, 3, 3, 0, 0);
    user.setPassword(passwordEncoder.encode(passwordGenerated));
    saveUserData(user);

    Logger.getAnonymousLogger()
            .info("new password->".concat(passwordGenerated) + this.getClass().getSimpleName() + "recoverPassword");

    String firstName = user.getFirstName() != null ? user.getFirstName().split(" ")[0] : null;
    firstName = firstName == null || firstName.isEmpty() ? user.getUsername() : firstName;

    Logger.getAnonymousLogger().info("Sending password recover mail to ".concat(user.getEmail())
            + this.getClass().getSimpleName() + "recoverPassword");

    emailOperations.sendPasswordRecoverEmail(user.getEmail(), firstName, passwordGenerated, false);

    return new ResponseEntity<>(defaultResponse, new HttpHeaders(), HttpStatus.OK);
}
```

Figura 158. Servicio restaurar contraseña.
Elaborado por: El investigador.

2) Exponer servicio

```
@PostMapping(RECOVER_URI)
@ResponseBody
public ResponseEntity<DefaultResponse> recoverPassword(@RequestBody UserUpdateDTO userUpdateDTO) {
    return userOperations.recoverPassword(userUpdateDTO);
}
```

Figura 159. Crear end point restaurar contraseña.
Elaborado por: El investigador.

- **Actualización de los datos de un usuario**

Actividades:

- **Crear servicio para actualizar datos de un usuario.**

La lógica de negocio establece que un usuario puede actualizar su información adicional.

1) Crear servicio para actualizar los datos de un usuario.

El servicio valida la petición con el access token y al mismo tiempo obtiene el usuario para actualizar su información.

```
public ResponseEntity<UserUpdateDTO> updateUser(UserUpdateDTO userUpdateDTO, String authorizationToken) {
    User user = validateUser(authorizationToken);

    Boolean profileNotCompletedBefore = (user.getProfileFulfilled() == null || !user.getProfileFulfilled())
            && (user.getFirstName() == null || user.getLastName() == null || user.getBirthDate() == null
                    || user.getPhone() == null || user.getCity() == null || user.getSupportedTeamId() == null);

    HttpHeaders httpHeaders = new HttpHeaders();
    user.setUsername(userUpdateDTO.getUsername() == null ? user.getUsername() : userUpdateDTO.getUsername());
    user.setFirstName(userUpdateDTO.getFirstName() == null ? user.getFirstName() : userUpdateDTO.getFirstName());
    user.setLastName(userUpdateDTO.getLastName() == null ? user.getLastName() : userUpdateDTO.getLastName());
    user.setBirthDate(userUpdateDTO.getBirthDate() == null ? user.getBirthDate() : userUpdateDTO.getBirthDate());
    user.setPhone(userUpdateDTO.getPhone());
    user.setCity(userUpdateDTO.getCity() == null ? user.getCity() : userUpdateDTO.getCity());

    if (userUpdateDTO.getSupportedTeamId() != null) {
        Team team = teamOperations.getTeam(userUpdateDTO.getSupportedTeamId());
        user.setSupportedTeamId(team);
    }

    Boolean profileCompletedAfter = (user.getProfileFulfilled() == null || !user.getProfileFulfilled())
            && (user.getFirstName() != null && user.getLastName() != null && user.getBirthDate() != null
                    && user.getPhone() != null && user.getCity() != null && user.getSupportedTeamId() != null);

    if (profileNotCompletedBefore && profileCompletedAfter) {
        user.setProfileFulfilled(true);
    }
    saveUserData(user);

    return new ResponseEntity<>(userUpdateDTO, httpHeaders, HttpStatus.OK);
}
```

Figura 160. Actualizar datos de un usuario.
Elaborado por: El investigador.

2) Exponer servicio

```
@PostMapping(USER_UPDATE_URI)
@ResponseBody
public ResponseEntity<UserUpdateDTO> updateUser(@Valid @RequestBody UserUpdateDTO userUpdateDTO,
        BindingResult bindingResult, @RequestHeader(value = "Authorization") String authorizationToken) {
    return userOperations.updateUser(userUpdateDTO, authorizationToken);
}
```

Figura 161. Crea end point actualizer datos de usuarios.
Elaborado por: El investigador.

3.3. Resultados SCRUM

3.3.1. Gráfica resultante del progreso del proyecto.

En SCRUM se utiliza la gráfica Burn Down para medir el progreso de un proyecto al finalizar cada Sprint. En el presente proyecto se ha utilizado como unidad de medida los ***Story Points*** (Puntos de esfuerzo por historia) para graficar el progreso del proyecto.

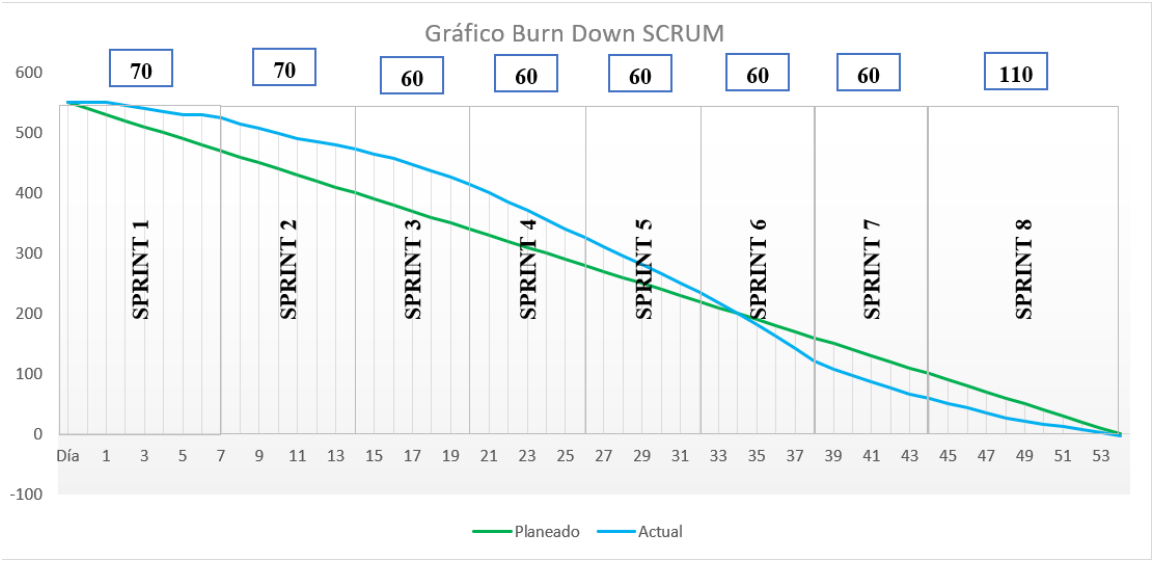


Figura 162. Grafica Burn Down SCRUM.
Elaborado por: El investigador.

3.3.2. Análisis de la gráfica.

En la figura 162 se puede observar la representación gráfica del progreso del proyecto al finalizar todos los Sprints.

El desarrollo del proyecto se planificó para 8 Sprints con duraciones de 1 a 2 semanas por cada uno. Se ha establecido como objetivo ideal cumplir 10 puntos de esfuerzo por día, obteniendo totales de 60 a 110 puntos por Sprint.

En la figura 161 se encuentran graficadas 2 líneas que representan el progreso ideal (línea verde) y el progreso real del proyecto (línea azul) de lo cual se puede concluir:

- El progreso del proyecto tuvo un retraso en los primeros Sprints debido al tiempo que se le dedicó al estudio y aprendizaje de las herramientas para el desarrollo como Spring Boot y Docker.

- A partir del Sprint 4 aumentaron los puntos de esfuerzo cumplidos con respecto a los sprint anteriores, mostrando que el equipo de desarrollo empieza a tener mejor dominio sobre el producto y las herramientas de software.

- Desde el sprint 6 se recupera y sobrepasa el umbral ideal, demostrando que se cumplido con el estudio y aprendizaje de las herramientas tomando un control absoluto sobre el producto y mejorando exponencialmente las métricas de esfuerzo.

3.4. Pruebas

- **Conceptos**

Pruebas Unitarias

Se ha utilizado TDD (Desarrollo Guiado por Pruebas) para el desarrollo del presente proyecto debido a que nos brinda la posibilidad de reducir el impacto de errores en producción, detectándolos fácilmente en las etapas de desarrollo e integración.

Las pruebas unitarias consisten en probar de manera aislada la lógica que contiene una determinada función, permitiendo detectar errores en la fase de desarrollo cuando se da mantenimiento a un sistema o se agregan funcionalidades que afecten la lógica de negocio.

Pruebas de Integración

Las pruebas de integración son otro tipo de pruebas diferentes a las unitarias. Este tipo de pruebas garantizan que todos los elementos probados unitariamente se integren correctamente y puedan ser utilizados por otra aplicación u otro proceso dentro de la misma.

- **Instalación de librerías con Gradle**

El repositorio de Gradle dispone de librerías útiles para realizar pruebas unitarias y de integración como: *spring boot test, junit, mocks, jacoco, etc.*

Implementar librerías test en *build.gradle*

```
dependencies {
    implementation('org.springframework.boot:spring-boot-starter')
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation('org.springframework.boot:spring-boot-starter-mail:1.2.0.RELEASE')
    implementation('org.apache.commons:commons-lang3:3.8.1')
    implementation('javax.xml.bind:jaxb-api:2.3.0')
    implementation('com.auth0:java-jwt:3.4.1')
    implementation('org.thymeleaf:thymeleaf:3.0.11.RELEASE')
    implementation('org.thymeleaf:thymeleaf-spring5:3.0.11.RELEASE')

    compileOnly 'org.projectlombok:lombok'
    runtimeOnly 'com.h2database:h2'
    runtimeOnly 'org.postgresql:postgresql'
    annotationProcessor 'org.projectlombok:lombok'

    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.security:spring-security-test'
    testImplementation('com.h2database:h2:1.4.197')
    testImplementation('org.assertj:assertj-core:3.11.1')
    testImplementation('org.assertj:assertj-joda-time:2.2.0')
    testImplementation('junit:junit:4.12')
    testImplementation('org.easymock:easymock:3.4')
    testImplementation('com.github.javafaker:javafaker:0.15')
    testImplementation('javax.xml.bind:jaxb-api:2.3.0')
}
```

Figura 163. Implementar librerías test
Elaborado por: El investigador.

En la figura 163 se encuentra:

Línea 40: Implementación de la librería *spring boot starter test* para testear aplicaciones Spring Boot con JUnit.

Línea 41: Implementación de la librería *spring security test* para testear aplicaciones con Spring Security.

Línea 45: Implementación de la librería JUnit para el desarrollo de pruebas.

Línea 46: Implementación de la librería *easymock* para crear objetos falsos que son utilizados durante la ejecución de las pruebas unitarias.

Configuración del runtime test en build.gradle

En Gradle se puede configurar las tareas que pueden ser ejecutadas desde Gradlew como son las pruebas unitarias y de integración.

```
51 test {
52     testLogging {
53         events 'passed', 'skipped', 'failed'
54         exceptionFormat = 'full'
55     }
56     exclude '**/ec/com/alquimiasoft/balonazo/integrationtest/*'
57 }
58
59 task integrationTest(type: Test) {
60     testLogging {
61         events 'passed', 'skipped', 'failed'
62         exceptionFormat = 'full'
63     }
64     include '**/ec/com/alquimiasoft/balonazo/integrationtest/*'
65 }
```

Figura 164. Tareas gradle test y test integration.
Elaborado por: El investigador.

En la figura 164 se encuentra:

Línea 51: Inicio de la tarea *test* para ejecutar pruebas unitarias con Gradle.

Línea 52: Bloque que contiene los eventos resultantes después de terminar la ejecución de las pruebas unitarias.

Línea 56: Se excluye los paquetes de clases que no serán ejecutadas durante la tarea, en este caso las pruebas de integración.

Línea 59: Inicio de la tarea *integrationTest* para ejecutar pruebas de integración con Gradle.

Línea 64: Se incluye los paquetes de clases que serán ejecutadas durante la tarea, en este caso las pruebas de integración.

Configuración de jacoco test en build.gradle

La librería Jacoco permite crear un reporte detallado del resultado de las pruebas unitarias, detallando el porcentaje de cobertura y resaltando el código faltante por probar.

```
67 jacocoTestReport {
68     reports {
69         xml.enabled false
70         csv.enabled false
71         html.destination file("${buildDir}/jacocoHtml")
72     }
73     afterEvaluate {
74         classDirectories = files(classDirectories.files.collect {
75             fileTree(dir: it, exclude: [
76                     '**/ec/com/alquimiasoft/balonazo/BalonazoApplication*'
77             ])
78         })
79     }
80 }
```

Figura 165. Configuración Jacoco Report.
Elaborado por: El investigador.

En la figura 165 se encuentra:

Línea 67: Inicio del bloque jacoco para generar el reporte de resumen de las pruebas unitarias.

Línea 68: Bloque para indicar el formato del archivo resultando al finalizar la ejecución de las pruebas unitarias, en este caso el reporte se genera en un archivo html.

Línea 73: Bloque para especificar configuraciones adicionales, como los paquetes de archivos a ser excluidos del test de cobertura.

- **Crear clase de objetos simulados**

Para probar el código de una aplicación se utilizan objetos simulados para reemplazar a los objetos reales durante la ejecución y continuar con el proceso que le corresponde. Una clase que provea de objetos simulados es de mucha utilidad para crear un objeto en cualquier lugar que se lo necesite.

En la siguiente figura se encuentra la clase *Objects* que se utiliza como proveedor de objetos simulados.

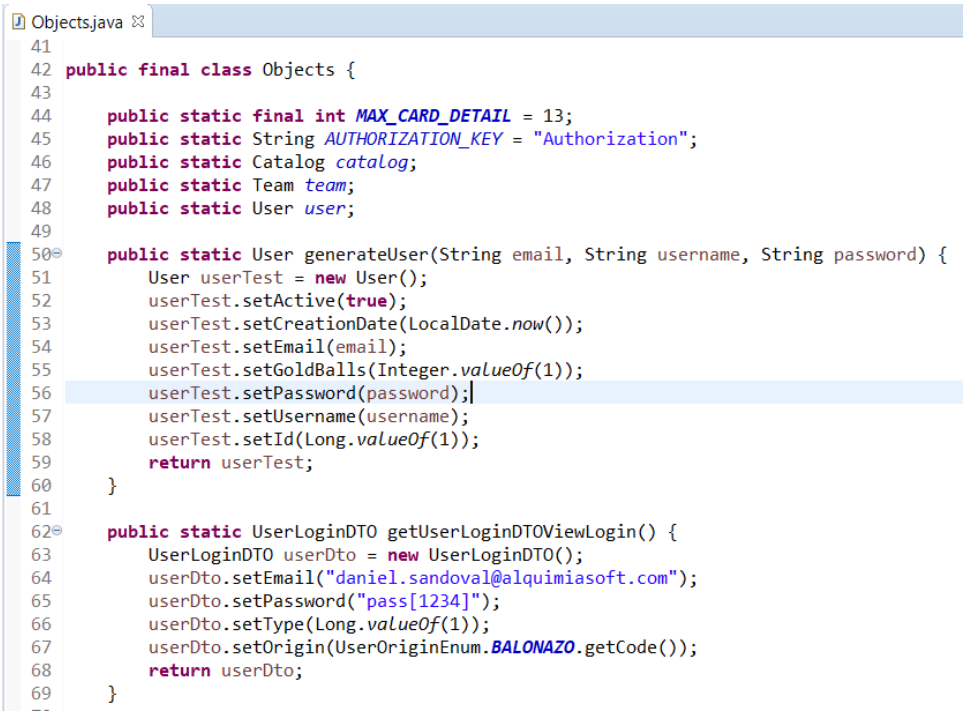


Figura 166. Clase Objects con objetos fake.
Elaborado por: El investigador.

3.4.1. Pruebas Unitarias.

- **Pruebas Unitarias *User Service***

Una Prueba unitaria permite probar de manera individual cualquier función y método que requiera validar lógica de negocio. El objetivo de las pruebas unitarias es validar la lógica de negocio de los servicios de una aplicación, para que, al momento de integrarlos con otros servicios, esté garantizado en gran parte su funcionamiento, reduciendo la aparición de errores y el impacto posdesarrollo, tal y como se muestra en las figuras 167 y 168.

En la figura 167 se encuentra:

@SpringBootTest: Notación spring para especificar que una clase es de tipo test e inicie su ejecución con la librería *spring boot started test.*

@Autowired: Notación spring que permite inyectar dependencias como servicios y operadores.

@MockBean: Notación spring boot test para agregar objetos simulados al contexto de la aplicación que reemplazará cualquier bean existente del mismo tipo.

Los servicios que van a ser probados se los inyectará en la clase, en este caso *User Service.* Los servicios, repositorios y operadores adicionales que forman parte de un proceso a ser probado se los inyectará como objetos simulados.

UserServiceTest.java

```java
package ec.com.alquimiasoft.balonazo.service;

import static org.assertj.core.api.Assertions.assertThat;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserServiceTest {
    private static User user = Objects.getUser();
    private static UserLoginDTO userDto = Objects.getUserLoginDTOViewLogin();
    private static UserDTO userDTO = Objects.getUserDTO();

    @Autowired
    private UserService userService;

    @MockBean
    private UserRepository userRepository;

    @MockBean
    private PrognosticService prognosticOperations;

    @MockBean
    private PasswordEncoder passwordEncoder;

    @MockBean
    private JavaMailSender emailSender;

    @MockBean
    private NotificationRepository notificationRepository;

    @MockBean
    private AsyncEmailService asyncEmailService;

    @MockBean
    private UserMembershipRepository userMembershipRepository;

    @MockBean
    private MembershipRepository membershipRepository;
```

Figura 167. User Service Test - parte 1
Elaborado por: El investigador.

UserServiceTest.java

```java
    @Before
    public void initialize() {
        tokenTest = tokenService.createToken("test@test.com", TokenTypeEnum.ACCESS_TOKEN.getCode());

        PrognosticsActiveUserResponseDTO prognosticsFake = new PrognosticsActiveUserResponseDTO();
        prognosticsFake.setPrognosticList(new ArrayList<>());
        Mockito.when(prognosticOperations.getPrognosticsActiveUser(Mockito.any(), Mockito.any(), Mockito.any()))
                .thenReturn(prognosticsFake);
    }

    /* Login */
    @Test
    public void testSuccessForLogin() {
        Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(user);
        Mockito.when(passwordEncoder.matches(Mockito.anyString(), Mockito.anyString())).thenReturn(true);
        List<UserMembership> value = Arrays.asList(new UserMembership(Long.valueOf(1), LocalDate.now(), true, user,
                Objects.createMembershipMock(), LocalDateTime.now(), null));
        Mockito.when(userMembershipRepository.getMembershipPerUser(Mockito.anyLong(), Mockito.any())).thenReturn(value);


        DataBinder data = new DataBinder(null);
        ResponseEntity<UserLoginDTO> result = userService.getUserLogin(userDto, data.getBindingResult());

        assertThat(result.getStatusCode()).isEqualTo(HttpStatus.OK);
    }
```

Figura 168. User Service Test - parte 2.
Elaborado por: El investigador.

En la figura 168 se encuentra:

@Before: Notación para indicar que un método debe ejecutarse antes de iniciar las pruebas.

Mockito: Librería que se utiliza para simular la ejecución de procesos y retornar valores específicos necesarios para las pruebas. La función *when* permite simular la ejecución de un método o función, adicionalmente se utiliza la función *thenReturn* para devolver valores falsos que cumplan condiciones específicas en las pruebas.

@Test: Notación spring boot test para indicar que un método es una prueba y deba ejecutarse al iniciar los tests.

assertThat: función que permite validar que un valor cumpla una condición específica, como validar la igualdad de dos valores, etc.

UserServiceTest.java

```
121    @Test
122    public void testWrongPassword() {
123        UserLoginDTO userDto = new UserLoginDTO();
124        Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(user);
125        Mockito.when(passwordEncoder.matches(Mockito.anyString(), Mockito.anyString())).thenReturn(false);
126        userDto.setEmail("daniel.sandoval@alquimiasoft.com.ec");
127        userDto.setPassword("pass[56478]");
128        userDto.setOrigin(UserOriginEnum.BALONAZO.getCode());
129
130        DataBinder data = new DataBinder(null);
131        ResponseEntity<UserLoginDTO> result = userService.getUserLogin(userDto, data.getBindingResult());
132
133        assertThat(result.getStatusCode()).isEqualTo(HttpStatus.UNAUTHORIZED);
134    }
135
136    @Test
137    public void testEmailNotExist() {
138        Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(null);
139        DataBinder data = new DataBinder(null);
140
141        ResponseEntity<UserLoginDTO> result = userService.getUserLogin(userDto, data.getBindingResult());
142
143        assertThat(result.getStatusCode()).isEqualTo(HttpStatus.PRECONDITION_REQUIRED);
144
145    }
146
147    @Test
148    public void testGetUserLoginHasErrorsValidateFields() {
149        DataBinder data = new DataBinder(null);
150        data.getBindingResult().addError(new ObjectError("validate", "Error in validate procesing"));
151
152        ResponseEntity<UserLoginDTO> result = userService.getUserLogin(userDto, data.getBindingResult());
153        assertThat(result.getBody().getMessage()).isNotNull();
154        assertThat(result.getStatusCode()).isEqualTo(HttpStatus.PRECONDITION_REQUIRED);
155    }
```

Figura 169. User Service Test - parte 3.
Elaborado por: El investigador.

UserServiceTest.java

```
156
157     /* Save User */
158     @Test
159     public void testSaveUserHasErrorsValidateFields() {
160         DataBinder data = new DataBinder(null);
161         data.getBindingResult().addError(new ObjectError("validate", "Error in validate procesing"));
162         ResponseEntity<UserDTO> result = userService.saveUser(userDTO, data.getBindingResult());
163         assertThat(result.getStatusCode()).isEqualTo(HttpStatus.PRECONDITION_REQUIRED);
164     }
165
166     @Test
167     public void testSaveUserSuccess() {
168         Mockito.when(userRepository.findByUsername(Mockito.any())).thenReturn(null);
169         Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(null);
170         List<UserMembership> value = Arrays.asList(new UserMembership(Long.valueOf(1), LocalDate.now(), true, user,
171                 Objects.createMembershipMock(), LocalDateTime.now(), null));
172         Mockito.when(userMembershipRepository.getMembershipPerUser(Mockito.anyLong(), Mockito.any())).thenReturn(value);
173
174         Optional<Membership> membershipMock = Optional.of(Objects.createMembershipMock());
175         Mockito.when(membershipRepository.findById(Mockito.any())).thenReturn(membershipMock);
176
177         Mockito.when(userRepository.saveAndFlush(Mockito.any(User.class))).thenReturn(Objects.createUser());
178
179         DataBinder data = new DataBinder(null);
180         ResponseEntity<UserDTO> result = userService.saveUser(userDTO, data.getBindingResult());
181         assertThat(result.getStatusCode()).isEqualTo(HttpStatus.OK);
182         assertThat(result.getBody().getPassword()).isEmpty();
183     }
184
```

Figura 170. User Service Test - parte 4.
Elaborado por: El investigador.

Ejecución de la prueba unitaria.

Para ejecutar las pruebas unitarias de una aplicación utilizando Gradle se utiliza el siguiente comando en una terminal de comandos:

gradlew test jacocoTestReport

```
> Task :test

ec.com.alquimiasoft.balonazo.BalonazoApplicationTests > contextLoads PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > sendEmailTest PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > verifyRecoverPassword_WhenEmailIsNull PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > testSaveUserHasErrorsValidateFields PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > testSaveUserSuccess PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > testSuccessForLoginAndPopup PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > testMaximunLengthForUsername PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > testGetUserLoginHasErrorsValidateFields PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > testSuccessForLogin PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > testSuccessForLoginAndNotificationsForPopup PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > testWrongPassword PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > verifyUpdateUser_withProfileFulfilled PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > testEmailNotExist PASSED

ec.com.alquimiasoft.balonazo.service.UserServiceTest > verifyRecoverPassword_WhenIsOk PASSED
```

Figura 171. Ejecutar User Service Test.
Elaborado por: El investigador.

Reporte Jacoco Test.

balonazo > ec.com.alquimiasoft.balonazo.service > UserService

UserService

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
saveUser(UserDTO, BindingResult)		80%		56%	5	9	11	59	0	1
updateUser(UserUpdateDTO, String)		79%		36%	26	27	0	30	0	1
loadUserByUsername(String)		0%		0%	2	2	5	5	1	1
getUserLogin(UserLoginDTO, BindingResult)		88%		72%	5	10	6	55	0	1
recoverPassword(UserUpdateDTO)		84%		50%	4	6	1	17	0	1
getDataFromUser(Long, String)		0%		n/a	1	1	2	2	1	1
getUserBalance(Long, String)		0%		n/a	1	1	4	4	1	1
validateUser(String)		65%		50%	4	5	3	11	0	1
validateUniqueEmail(UserDTO)		73%		50%	1	2	1	4	0	1
findAllUsers()		0%		n/a	1	1	1	1	1	1
emailExist(String)		77%		50%	1	2	0	1	0	1
lambda$getUserLogin$2(Notification)		88%		50%	1	2	0	1	0	1
getProfile(UserUpdateDTO)		100%		50%	1	2	0	14	0	1
sendContactEmail(String, String, String, String)		100%		n/a	0	1	0	8	0	1
lambda$getUserLogin$1(ObjectError)		100%		n/a	0	1	0	1	0	1
lambda$saveUser$0(ObjectError)		100%		n/a	0	1	0	1	0	1
saveUserData(User)		100%		n/a	0	1	0	1	0	1
findByEmail(String)		100%		n/a	0	1	0	1	0	1
lambda$getUserLogin$3(UserLoginDTO, Notification)		100%		n/a	0	1	0	2	0	1
UserService()		100%		n/a	0	1	0	1	0	1
Total	229 of 1.155	80%	60 of 114	47%	53	77	34	216	4	20

Figura 172. Reporte Jacoco User Service Test.
Elaborado por: El investigador.

- **Pruebas Unitarias *Card Service***

CardServiceTest.java

```
35 @RunWith(SpringRunner.class)
36 @SpringBootTest
37 public class CardServiceTest {
38     @Autowired
39     private CardOperations cardOperations;
40
41     @MockBean
42     private CardRepository cardRepository;
43
44     @MockBean
45     private CardDetailRepository cardDetailRepository;
46
47     @MockBean
48     private UserService userOperations;
49
50     @MockBean
51     private CardDetailCommentsRepository cardDetailCommentsRepository;
52
53     @MockBean
54     private PrognosticRepository prognosticRepository;
55     private Card card;
56
57     private List<CardDetailComments> commentsFake;
58
59     @MockBean
60     private UserMembershipRepository userMembershipRepository;
61
62     private User user;
63
64     @Before
65     public void initialize() {
66         card = Objects.generateCardMock();
67         commentsFake = Objects.generateComments(card);
68         user = new User(Long.valueOf(1), "test", "test@test.com", "prueba", LocalDate.now(),
69                 true, Integer.valueOf(0), null, null, null, null, null, null, null,
70                 UserOriginEnum.BALONAZO.getCode());
71     }
```

Figura 173. Card Service Test - parte 1.
Elaborado por: El investigador.

CardServiceTest.java

```java
    @Test
    public void validateMatches() {
        List<LocalDateTime> value = new ArrayList<>();
        value.add(LocalDateTime.now());
        Mockito.when(cardDetailRepository.getMinDate(Mockito.any())).thenReturn(value);
        Mockito.when(cardRepository.getCardValidBetweenDate(Mockito.any(), Mockito.any())).thenReturn(card);
        Mockito.when(userOperations.findByEmail(Mockito.any())).thenReturn(user);
        CardDTO cardDTOin = new CardDTO();
        cardDTOin.setEmail("test@test.com");
        Mockito.when(prognosticRepository.getPrognosticCountPerCardAndUser(Mockito.any(), Mockito.any()))
                .thenReturn(Integer.valueOf(0));

        List<UserMembership> value2 = Arrays.asList(new UserMembership(Long.valueOf(1),LocalDate.now(),
                true,Objects.createUser(),Objects.createMembershipMock(),LocalDateTime.now(),null));
        Mockito.when(userMembershipRepository.getMembershipPerUser(Mockito.any(),Mockito.any())).thenReturn(value2 );

        CardDTO cardDTO = cardOperations.getCardValid(cardDTOin, true);
        assertThat(cardDTO.getMatches().size()).isGreaterThanOrEqualTo(1);
    }

    @Test
    public void validateCardExistence() {
        List<LocalDateTime> value = new ArrayList<>();
        value.add(LocalDateTime.now());
        Mockito.when(cardDetailRepository.getMinDate(Mockito.any())).thenReturn(value);
        Mockito.when(cardRepository.getCardValidBetweenDate(Mockito.any(), Mockito.any())).thenReturn(null);
        Mockito.when(userOperations.findByEmail(Mockito.any())).thenReturn(user);
        CardDTO cardDTOin = new CardDTO();
        cardDTOin.setEmail("test@test.com");
        Mockito.when(prognosticRepository.getPrognosticCountPerCardAndUser(Mockito.any(), Mockito.any()))
                .thenReturn(Integer.valueOf(0));

        CardDTO cardDTO = cardOperations.getCardValid(cardDTOin, true);
        assertThat(cardDTO).isNull();
    }
```

Figura 174. Card Service Test - parte 2.
Elaborado por: El investigador.

CardServiceTest.java

```java
    @Test
    public void validateNationalMatchesExistence() {
        List<LocalDateTime> value = new ArrayList<>();
        value.add(LocalDateTime.now());
        Mockito.when(cardDetailRepository.getMinDate(Mockito.any())).thenReturn(value);
        Mockito.when(cardRepository.getCardValidBetweenDate(Mockito.any(), Mockito.any())).thenReturn(card);
        Mockito.when(userOperations.findByEmail(Mockito.any())).thenReturn(user);
        CardDTO cardDTOin = new CardDTO();
        cardDTOin.setEmail("test@test.com");
        Mockito.when(prognosticRepository.getPrognosticCountPerCardAndUser(Mockito.any(), Mockito.any()))
                .thenReturn(Integer.valueOf(0));

        List<UserMembership> value2 = Arrays.asList(new UserMembership(Long.valueOf(1),LocalDate.now(),
                true,Objects.createUser(),Objects.createMembershipMock(),LocalDateTime.now(),null));
        Mockito.when(userMembershipRepository.getMembershipPerUser(Mockito.any(),Mockito.any())).thenReturn(value2 );

        CardDTO cardDTO = cardOperations.getCardValid(cardDTOin, true);

        assertThat(cardDTO.getMatches()).isNotNull();

    }

    @Test
    public void verifyGetCard_whenThereIsAValue() {
        Optional<Card> cardFake = Optional.of(card);
        Mockito.when(cardRepository.findById(Mockito.anyLong())).thenReturn(cardFake);
        Card cardOut = cardOperations.getCard(Long.valueOf(1));
        assertThat(cardOut).isEqualTo(card);
    }

    @Test
    public void verifyGetCard_whenThereIsNotAValue() {
        Optional<Card> cardFake = Optional.empty();
        Mockito.when(cardRepository.findById(Mockito.anyLong())).thenReturn(cardFake);
        Card cardOut = cardOperations.getCard(Long.valueOf(1));
        assertThat(cardOut).isNull();
    }
```

Figura 175. Card Service Test - parte 3.
Elaborado por: El investigador

- **Pruebas Unitarias *Prognostic Service***

PrognosticServiceTest.java

```java
package ec.com.alquimiasoft.balonazo.service;

import static ec.com.alquimiasoft.balonazo.tools.StreamTools.singlResult;

@RunWith(SpringRunner.class)
@SpringBootTest
public class PrognosticServiceTest {
    private PrognosticDTO prognosticDTO;

    private static List<Prognostic> prognosticList = Objects.getPrognosticList();

    @Autowired
    private PrognosticOperations prognosticOperations;

    @Autowired
    private TokenService tokenService;

    @MockBean
    private CardRepository cardRepository;

    @MockBean
    private PrognosticRepository prognosticRepository;

    @MockBean
    private PrognosticDetailRepository prognosticDetailRepository;

    @MockBean
    private CatalogDetailRepository catalogDetailRepository;

    @MockBean
    private NotificationRepository notificationRepository;

    @MockBean
    private UserRepository userRepository;

    @MockBean
    private CardDetailRepository cardDetailRepository;

    @MockBean
    private UserMembershipRepository userMembershipRepository;
```

Figura 176. Prognostic Service Test - parte 1.
Elaborado por: El investigador.

PrognosticServiceTest.java

```java
    @Before
    public void arrange() {
        prognosticDTO = Randomic.getPrognosticDTO();
        mockingCardDetailRepository();

        tokenTest = tokenService.createToken("test@test.com", TokenTypeEnum.ACCESS_TOKEN.getCode());

        Membership membership = Objects.createMembershipMock();

        List<UserMembership> userMemberShipMock = Arrays.asList(new UserMembership(Long.valueOf(1), LocalDate.now(),
                true, Objects.createUser(), membership, LocalDateTime.now(), null));
        Mockito.when(userMembershipRepository.getMembershipPerUser(Mockito.anyLong(), Mockito.any()))
                .thenReturn(userMemberShipMock);

        List<MembershipHits> value = Arrays.asList(
                new MembershipHits(Long.valueOf(1), LocalDate.now(), true, 9, membership, true, false, false,
                        Integer.valueOf(1)),
                new MembershipHits(Long.valueOf(2), LocalDate.now(), true, 10, membership, true, false, false,
                        Integer.valueOf(2)),
                new MembershipHits(Long.valueOf(3), LocalDate.now(), true, 11, membership, true, false, false,
                        Integer.valueOf(3)),
                new MembershipHits(Long.valueOf(4), LocalDate.now(), true, 12, membership, false, true, true, null),
                new MembershipHits(Long.valueOf(5), LocalDate.now(), true, 13, membership, false, true, true, null));

        Mockito.when(membershipHitsRepository.getHitsPerMembership(Mockito.anyLong(), Mockito.any())).thenReturn(value);

        prognostic = Objects.generatePrognostic();
    }

    @Test
    public void validatePrognosticEnum() {
        assertThat(StringUtil.getPrognostic("V").toString()).isEqualTo("VISITOR_WINNER");
        assertThat(StringUtil.getPrognostic("E").toString()).isEqualTo("EVEN");
        assertThat(StringUtil.getPrognostic("L").toString()).isEqualTo("LOCAL_WINNER");
    }

    @Test(expected = IncompletePrognosticException.class)
    public void verifyValidPrognosticData() {
        prognosticOperations.validatePrognostic(prognosticDTO);
    }
```

Figura 177. Prognostic Service Test - parte 2.
Elaborado por: El investigador.

PrognosticServiceTest.java

```
231
232    @Test
233    public void shouldReturnAPrognosticWithTheDataOfCardDTO() {
234        User user = new User(Long.valueOf(1), "test", "test1@test.com", "pwdtest", LocalDate.now(), true,
235                Integer.valueOf(100), null, null, null, null, null, null, null, UserOriginEnum.BALONAZO.getCode());
236
237        List<LocalDateTime> datesMock = Arrays.asList(LocalDateTime.now());
238        Mockito.when(cardDetailRepository.getMinDate(Mockito.any())).thenReturn(datesMock);
239
240        Card cardMock = Objects.generateCardMock();
241        cardMock.setId(Long.valueOf(3));
242        Optional<Card> card = Optional.of(cardMock);
243        Mockito.when(cardRepository.findById(Mockito.any())).thenReturn(card);
244
245        Mockito.when(cardRepository.getCardValidBetweenDate(Mockito.any(), Mockito.any())).thenReturn(cardMock);
246
247        List<LocalDateTime> dateList = Arrays.asList(LocalDateTime.now());
248        Mockito.when(cardDetailRepository.getMinDate(Mockito.any())).thenReturn(dateList);
249
250        Mockito.when(prognosticRepository.getPrognosticCountPerCardAndUser(Mockito.anyLong(), Mockito.any()))
251                .thenReturn(Integer.valueOf(1));
252
253        prognosticDTO = cardDTOToTestTrasnformToPrognostic();
254
255        Prognostic transformed = prognosticOperations.cardDTOToPrognostic(prognosticDTO, user);
256        assertThat(transformed.getCard().getId()).isEqualTo(prognosticDTO.getBook().getId());
257        transformed.getPrognosticDetailList().forEach((PrognosticDetail prognosticDetail) -> {
258            PrognosticDetailDTO cardDetailBelongsToPrognosticDetailItem = prognosticDTO.getBook().getMatches().stream()
259                    .filter(match -> match.getId() == prognosticDetail.getCardDetail().getPKey().getId())
260                    .collect(singlResult());
261
262            if (StringUtil.getPrognostic(cardDetailBelongsToPrognosticDetailItem.getMatchResult()) == null) {
263                throw new AssertionError("Different Prognostic value");
264            }
265            assertThat(prognosticDetail.getPKey().getPrognostic()).isEqualTo(transformed);
266        });
267    }
268
269    @Test
270    public void sendEmptyParamMethodgetPrognosticsActiveUser() {
271        PrognosticsActiveUserResponseDTO result = prognosticOperations.getPrognosticsActiveUser(null,null, true);
272        assertThat(result).isNull();
273    }
```

Figura 178. Prognostic Service Test - parte 3.
Elaborado por: El investigador.

- **Pruebas Unitarias *Notification Service***

NotificationServiceTest.java

```
39 @RunWith(SpringRunner.class)
40 @SpringBootTest
41 public class NotificationServiceTest {
42     @Autowired
43     private NotificationOperations notificationOperations;
44     @Autowired
45     private TokenService tokenService;
46     @MockBean
47     private UserRepository userRepository;
48     @MockBean
49     private NotificationRepository notificationRepository;
50     private String authorizationToken;
51     private String adminAuthorizationToken;
52     @MockBean
53     private AdminUserRepository adminUserRepository;
54
55     @Before
56     public void initialize() {
57         authorizationToken = tokenService.createToken("prueba@prueba", TokenTypeEnum.ACCESS_TOKEN.getCode());
58         adminAuthorizationToken = tokenService.createToken("admin@prueba", TokenTypeEnum.ACCESS_TOKEN.getCode());
59     }
60
61     @Test
62     public void verifyNotifications_whenThereisDataAndUserIsValid() {
63         UserLoginDTO userLoginDTO = new UserLoginDTO();
64         userLoginDTO.setEmail("prueba@prueba");
65         User userFake = Objects.createUser();
66         Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(userFake);
67
68         List<Notification> notificationsFake = Arrays.asList(Objects.createNotification(Long.valueOf(1),
69                 NotificationTypeEnum.NOTIFICATION.getCode(), null, userFake));
70         Mockito.when(notificationRepository.findByUserId(Mockito.anyLong(),
71                 Mockito.eq(NotificationTypeEnum.NOTIFICATION.getCode()))).thenReturn(notificationsFake);
72         List<Notification> popupFake = Arrays.asList(Objects.createNotification(Long.valueOf(2),
73                 NotificationTypeEnum.TABLE.getCode(), PopupTypeEnum.EARNED_BALLOONS.getCode(), userFake));
74         Mockito.when(notificationRepository.findByUserId(Mockito.anyLong(),
75                 Mockito.eq(NotificationTypeEnum.POPUP.getCode()))).thenReturn(popupFake);
76         ResponseEntity<List<NotificationDTO>> response = notificationOperations.getAllNotifications(authorizationToken);
77
78         assertThat(response).isNotNull();
79         Mockito.verify(notificationRepository, Mockito.times(1)).saveAndFlush(Mockito.any());
80     }
```

Figura 179. Notification Service Test - parte 1.
Elaborado por: El investigador.

NotificationServiceTest.java

```java
    @Test(expected = InvalidUserException.class)
    public void verifyNotifications_whenUserIsNotValid() {
        UserLoginDTO userLoginDTO = new UserLoginDTO();
        userLoginDTO.setEmail("prueba@prueba");

        Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(null);

        notificationOperations.getAllNotifications(authorizationToken);
    }

    @Test
    public void verifyNotifications_whenThereisNoNotificationsButThereisPopup() {
        UserLoginDTO userLoginDTO = new UserLoginDTO();
        userLoginDTO.setEmail("prueba@prueba");

        User userFake = Objects.createUser();
        Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(userFake);

        Mockito.when(notificationRepository.findByUserId(Mockito.anyLong(),
                Mockito.eq(NotificationTypeEnum.NOTIFICATION.getCode()))).thenReturn(new ArrayList<>());

        List<Notification> popupFake = Arrays.asList(Objects.createNotification(Long.valueOf(2),
                NotificationTypeEnum.TABLE.getCode(), PopupTypeEnum.EARNED_BALLOONS.getCode(), userFake));

        Mockito.when(notificationRepository.findByUserId(Mockito.anyLong(),
                Mockito.eq(NotificationTypeEnum.POPUP.getCode()))).thenReturn(popupFake);

        ResponseEntity<List<NotificationDTO>> response = notificationOperations.getAllNotifications(authorizationToken);

        assertThat(response).isNotNull();
        Mockito.verify(notificationRepository, Mockito.never()).saveAndFlush(Mockito.any());
    }
```

Figura 180. Notification Service Test - parte 2.
Elaborado por: El investigador.

NotificationServiceTest.java

```java
    @Test
    public void verifyNotifications_whenThereisNotificationsButThereisNotPopup() {
        UserLoginDTO userLoginDTO = new UserLoginDTO();
        userLoginDTO.setEmail("prueba@prueba");

        User userFake = Objects.createUser();
        Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(userFake);

        List<Notification> notificationsFake = Arrays.asList(Objects.createNotification(Long.valueOf(1),
                NotificationTypeEnum.NOTIFICATION.getCode(), null, userFake));
        Mockito.when(notificationRepository.findByUserId(Mockito.anyLong(),
                Mockito.eq(NotificationTypeEnum.NOTIFICATION.getCode()))).thenReturn(notificationsFake);

        Mockito.when(notificationRepository.findByUserId(Mockito.anyLong(),
                Mockito.eq(NotificationTypeEnum.POPUP.getCode()))).thenReturn(new ArrayList<>());

        ResponseEntity<List<NotificationDTO>> response = notificationOperations.getAllNotifications(authorizationToken);

        assertThat(response).isNotNull();
        Mockito.verify(notificationRepository, Mockito.times(1)).saveAndFlush(Mockito.any());
    }

    @Test
    public void verifyNotifications_validateBonus() {
        User userFake = Objects.createUser();
        userFake.setGoldBalls(0);
        Mockito.when(notificationRepository.findNotification(Mockito.anyBoolean(), Mockito.anyLong(), Mockito.any(),
                Mockito.any())).thenReturn(new ArrayList<>());

        BalloonsBonusDTO balloonsBonusDTO = new BalloonsBonusDTO();
        balloonsBonusDTO.setProcessDate(LocalDate.now());
        balloonsBonusDTO.setQuantity(Integer.valueOf(5));
        balloonsBonusDTO.setZeroBalloonsOnly(true);

        notificationOperations.verifyGoldBalloonsBonus(balloonsBonusDTO, userFake);
        Mockito.verify(notificationRepository, Mockito.times(1)).saveAndFlush(Mockito.any());
    }
```

Figura 181. Notification Service Test - parte 3.
Elaborado por: El investigador.

- **Pruebas Unitarias *Result Service***

ResultsServiceTest.java

```
34 public class ResultsServiceTest {
35     @Autowired
36     private ResultsOperations resultsOperations;
37     @MockBean
38     private CardRepository cardRepository;
39     @MockBean
40     private PrognosticRepository prognosticRepository;
41     @MockBean
42     private PrognosticDetailRepository prognosticDetailRepository;
43     @MockBean
44     private UserRepository userRepository;
45     @Autowired
46     private TokenService tokenService;
47
48     private String tokenTest;
49
50     private String validate(Integer localScore, Integer VisitorScore) {
59
60     @Before
61     public void initialize() {
62         tokenTest = tokenService.createToken("prueba", TokenTypeEnum.ACCESS_TOKEN.getCode());
63         Prognostic prognostic = Objects.generatePrognostic();
64         prognostic.getPrognosticDetailList().get(0).getPKey().getPrognostic().getPrognosticDetailList().get(0)
65                 .setPrognosticValue(PrognosticValueEnum.LOCAL_WINNER.getCode());
66         prognostic.getPrognosticDetailList().get(0).getPKey().getPrognostic().getPrognosticDetailList().get(0)
67                 .setPrognosticValue(PrognosticValueEnum.VISITOR_WINNER.getCode());
68         prognostic.getPrognosticDetailList().get(0).setMatch(true);
69         prognostic.getPrognosticDetailList().get(1).setMatch(true);
70         prognostic.getPrognosticDetailList().get(2).setMatch(true);
71         prognostic.getPrognosticDetailList().get(3).setMatch(true);
72         prognostic.getPrognosticDetailList().get(4).setMatch(true);
73         prognostic.getPrognosticDetailList().get(5).setMatch(true);
74         prognostic.getPrognosticDetailList().get(6).setMatch(true);
75         prognostic.getPrognosticDetailList().get(7).setMatch(true);
76         prognostic.getPrognosticDetailList().get(8).setMatch(true);
77         prognostic.getPrognosticDetailList().get(9).setMatch(true);
78         prognostic.getPrognosticDetailList().get(10).setMatch(true);
79         prognostic.getPrognosticDetailList().get(11).setMatch(true);
```

Figura 182. Result Service Test - parte 1.
Elaborado por: El investigador.

ResultsServiceTest.java

```
91
92     @Test
93     public void verifyResults() {
94         ResultsRequest resultsRequest = new ResultsRequest();
95         resultsRequest.setId(Long.valueOf(1));
96
97         User value = Objects.generateUser("a@a.com", "prueba", "prueba");
98         Mockito.when(userRepository.findByUsername(Mockito.any())).thenReturn(value);
99         Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(value);
100
101        ResultsResponseDTO resultsResponseDTO = resultsOperations.getResults(resultsRequest, tokenTest);
102        assertThat(resultsResponseDTO.getCard()).isNotNull();
103        assertThat(resultsResponseDTO.getPrognostics()).isNull();
104
105        resultsResponseDTO.getCard().getMatches().stream().forEach(item -> {
106            assertThat(validate(item.getLocalScore(), item.getVisitantScore())).isEqualTo(item.getMatchResult());
107        });
108
109    }
110
111    @Test
112    public void verifyResultsWhenMaxCardIdIsNull() {
113        Mockito.when(cardRepository.getMaxCardId(Mockito.any())).thenReturn(null);
114        User value = Objects.generateUser("a@a.com", "prueba", "prueba");
115        Mockito.when(userRepository.findByUsername(Mockito.any())).thenReturn(value);
116        Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(value);
117        resultsOperations.getResults(new ResultsRequest(), tokenTest);
118        Mockito.verify(prognosticRepository, Mockito.never()).getLastPrognostics(Mockito.any(),
119                Mockito.any(),Mockito.any(), Mockito.any());
120    }
```

Figura 183. Result Service Test - parte 2.
Elaborado por: El investigador.

ResultsServiceTest.java

```java
@Test
public void loadPrognosticsPerResult_whenOk() {
    ResultsRequest request = new ResultsRequest(1L,10,1);

    User value = Objects.generateUser("a@a.com", "prueba", "prueba");
    Mockito.when(userRepository.findByUsername(Mockito.any())).thenReturn(value);
    Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(value);

    Mockito.when(prognosticRepository.getPrognosticCountPerCardAndUser(Mockito.anyLong(),Mockito.any())).thenReturn(13);

    ResponseEntity<ResultsResponseDTO> result = resultsOperations.loadPrognosticsPerResult(request , tokenTest);

    assertThat(result.getStatusCode()).isEqualTo(HttpStatus.OK);
    assertThat(result.getBody().getPageCount()).isEqualTo(2);
    assertThat(result.getBody().getPrognostics().size()).isEqualTo(1);

    Long totalHits = result.getBody().getPrognostics().get(0).getPrognostics().stream()
            .filter(item -> item.getHit() == true).count();
    assertThat(result.getBody().getPrognostics().get(0).getTotalHits()).isEqualTo(totalHits);
}

@Test
public void loadPrognosticsPerResult_whenPageIsEven() {
    ResultsRequest request = new ResultsRequest(1L,10,1);

    User value = Objects.generateUser("a@a.com", "prueba", "prueba");
    Mockito.when(userRepository.findByUsername(Mockito.any())).thenReturn(value);
    Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(value);

    Mockito.when(prognosticRepository.getPrognosticCountPerCardAndUser(Mockito.anyLong(),Mockito.any())).thenReturn(20);

    ResponseEntity<ResultsResponseDTO> result = resultsOperations.loadPrognosticsPerResult(request , tokenTest);

    assertThat(result.getStatusCode()).isEqualTo(HttpStatus.OK);
    assertThat(result.getBody().getPageCount()).isEqualTo(2);
    assertThat(result.getBody().getPrognostics().size()).isEqualTo(1);

    Long totalHits = result.getBody().getPrognostics().get(0).getPrognostics().stream()
            .filter(item -> item.getHit() == true).count();
    assertThat(result.getBody().getPrognostics().get(0).getTotalHits()).isEqualTo(totalHits);
}
```

Figura 184. Result Service Test - parte 3.
Elaborado por: El investigador.

3.4.2. Pruebas de Integración

- **Pruebas de Integración *User Controller***

UserControllerIntegrationTest.java

```java
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class UserControllerIntegrationTest {
    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private UserRepository userRepository;

    @MockBean
    private TeamOperations teamOperations;

    @MockBean
    private CardDetailRepository cardDetailRepository;

    @Autowired
    private TokenService tokenService;

    @Autowired
    private JwtSettings jwtSettings;

    private String tokenTest;
    private String refreshTokenTest;

    @Before
    public void initTest() {
        tokenTest = tokenService.createToken("prueba", TokenTypeEnum.ACCESS_TOKEN.getCode());
        refreshTokenTest = tokenService.createToken("pruebaAdm,".concat(jwtSettings.getTokenIssuer()),
                TokenTypeEnum.REFRESH_TOKEN.getCode());
    }
```

Figura 185. User Controller Integration Test - parte 1.
Elaborado por: El investigador.

UserControllerIntegrationTest.java

```
69    @Test
70    public void updateUserSuccessfulTest() throws Exception {
71        User user = new User();
72        user.setActive(true);
73        user.setBirthDate(LocalDate.now());
74        user.setCity("Ciudad");
75        user.setCreationDate(LocalDate.now());
76        user.setEmail("prueba@prueba.com");
77        user.setFirstName("Primer nombre");
78        user.setLastName("Primer Apellido");
79        user.setPhone("2333444");
80        user.setUsername("prueba");
81        Mockito.when(userRepository.findByUsername(Mockito.any())).thenReturn(user);
82        Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(user);
83        Mockito.when(userRepository.saveAndFlush(Mockito.any())).thenReturn(user);
84        Mockito.when(teamOperations.getTeam(Mockito.anyLong())).thenReturn(new Team());
85
86        List<LocalDateTime> value = new ArrayList<>();
87        value.add(LocalDateTime.now());
88        Mockito.when(cardDetailRepository.getMinDate(Mockito.any())).thenReturn(value);
89        UserUpdateDTO userUpdateDTO = new UserUpdateDTO();
90        userUpdateDTO.setUsername("username");
91        mockMvc.perform(post("/api".concat(UserController.USER_UPDATE_URI)).header(Objects.AUTHORIZATION_KEY, tokenTest)
92                .contentType(MediaType.APPLICATION_JSON).content(convertObjectToJsonBytes(userUpdateDTO)))
93                .andExpect(status().isOk());
94
95        Mockito.verify(userRepository, Mockito.atLeastOnce()).saveAndFlush(Mockito.any());
96    }
97
98    @Test
99    public void updateUserMissingUserNameTest() throws Exception {
100
101       Mockito.when(userRepository.findByUsername(Mockito.any())).thenReturn(null);
102       Mockito.when(teamOperations.getTeam(Mockito.anyLong())).thenReturn(new Team());
103
104       List<LocalDateTime> value = new ArrayList<>();
105       value.add(LocalDateTime.now());
106       Mockito.when(cardDetailRepository.getMinDate(Mockito.any())).thenReturn(value);
107       UserUpdateDTO userUpdateDTO = new UserUpdateDTO();
108       mockMvc.perform(post("/api".concat(UserController.USER_UPDATE_URI)).header(Objects.AUTHORIZATION_KEY, tokenTest)
109               .contentType(MediaType.APPLICATION_JSON).content(convertObjectToJsonBytes(userUpdateDTO)))
110               .andExpect(status().isUnauthorized());
111
112   }
```

Figura 186. User Controller Integration Test - parte 2.
Elaborado por: El investigador.

UserControllerIntegrationTest.java

```
126   @Test
127   public void getNewAccessToken_whenNoRefreshTokenTest() throws Exception {
128       SessionDTO sessionDTO = new SessionDTO(null,null);
129       mockMvc.perform(post("/api".concat(UserController.USER_NEW_ACCESS_TOKEN_URI))
130               .contentType(MediaType.APPLICATION_JSON).content(convertObjectToJsonBytes(sessionDTO)))
131               .andExpect(status().isPreconditionRequired());
132   }
133
134   @Test
135   public void getNewAccessToken_whenNoUserinDatabaseTest() throws Exception {
136       SessionDTO sessionDTO = new SessionDTO(null,refreshTokenTest);
137       Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(null);
138
139       mockMvc.perform(post("/api".concat(UserController.USER_NEW_ACCESS_TOKEN_URI))
140               .contentType(MediaType.APPLICATION_JSON).content(convertObjectToJsonBytes(sessionDTO)))
141               .andExpect(status().isPreconditionRequired());
142
143   }
144
145   @Test
146   public void getNewAccessToken_whenFakeRefreshTokenTest() throws Exception {
147       String fakeRefreshTokenTest = tokenService.createToken("pruebaAdm,".concat("fake"),
148               TokenTypeEnum.REFRESH_TOKEN.getCode());
149       SessionDTO sessionDTO = new SessionDTO(null,fakeRefreshTokenTest);
150       User user = Objects.generateUser("test@test.com","test","test");
151       Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(user);
152
153       mockMvc.perform(post("/api".concat(UserController.USER_NEW_ACCESS_TOKEN_URI))
154               .contentType(MediaType.APPLICATION_JSON).content(convertObjectToJsonBytes(sessionDTO)))
155               .andExpect(status().isPreconditionRequired());
156
157   }
158
159   @Test
160   public void getNewAccessToken_whenUserIsNotActiveTest() throws Exception {
161       SessionDTO sessionDTO = new SessionDTO(null,refreshTokenTest);
162       User user = Objects.generateUser("test@test.com","test","test");
163       user.setActive(false);
164       Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(user);
165
166       mockMvc.perform(post("/api".concat(UserController.USER_NEW_ACCESS_TOKEN_URI))
167               .contentType(MediaType.APPLICATION_JSON).content(convertObjectToJsonBytes(sessionDTO)))
168               .andExpect(status().isPreconditionRequired());
169
170   }
```

Figura 187. User Controller IntegrationTest - parte 3.
Elaborado por: El investigador.

- **Pruebas de Integración *Card Controller***

CardControllerIntegrationTest.java

```
@SpringBootTest
@RunWith(SpringRunner.class)
@AutoConfigureMockMvc
public class CardControllerIntegrationTest {
    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private CardRepository cardRepository;

    @Autowired
    private TokenService tokenService;

    @MockBean
    private UserService userOperations;

    @MockBean
    private AdminUserRepository adminUserRepository;

    @MockBean
    private LastMatchesStatsRepository lastMatchesStatsRepository;

    @MockBean
    private CardDetailStatsRepository cardDetailStatsRepository;

    private String tokenTest;
    private String adminTokenTest;

    @Before
    public void initialize() {
        tokenTest = tokenService.createToken("test",TokenTypeEnum.ACCESS_TOKEN.getCode());
        adminTokenTest = tokenService.createToken("admTest",TokenTypeEnum.ACCESS_TOKEN.getCode());
    }
```

Figura 188. Card Controller Integration Test - parte 1.
Elaborado por: El investigador.

CardControllerIntegrationTest.java

```
    @Test
    public void saveCardResultsWhenFindByIdReturnsEmptyTest() throws Exception {
        Optional<Card> value = Optional.empty();
        Mockito.when(cardRepository.findById(Mockito.any())).thenReturn(value);
        Mockito.when(userOperations.findByUsername(Mockito.any())).thenReturn(new User());
        AdminUser adminUser = new AdminUser(Long.valueOf(1),LocalDate.now(),true,"admTest","prueba@prueba.com",
                "test",Integer.valueOf(1),null,null,null,null,null);
        Mockito.when(adminUserRepository.findByUsername(Mockito.any())).thenReturn(adminUser );

        SaveCardRequestDTO saveCardRequestDTO = new SaveCardRequestDTO();
        mockMvc.perform(post("/api".concat(CardController.CARD_SAVE_URI)).contentType(MediaType.APPLICATION_JSON)
                .content(convertObjectToJsonBytes(saveCardRequestDTO)).header("Authorization", tokenTest))
        .andExpect(status().isPreconditionRequired()).andReturn();

    }

    @Test
    public void saveCardResultsTest() throws Exception {
        Card card = Objects.generateCardMock();
        Optional<Card> value = Optional.of(card);
        Mockito.when(cardRepository.findById(Mockito.any())).thenReturn(value);
        Mockito.when(cardRepository.getCardInfo(Mockito.any())).thenReturn(card);
        Mockito.when(userOperations.findByUsername(Mockito.any())).thenReturn(new User());

        AdminUser adminUser = new AdminUser(Long.valueOf(1),LocalDate.now(),true,"admTest","prueba@prueba.com",
                "test",Integer.valueOf(1),null,null,null,null,null);
        Mockito.when(adminUserRepository.findByUsername(Mockito.any())).thenReturn(adminUser );

        SaveCardRequestDTO saveCardRequestDTO = Objects.generateSaveCardRequestDTO(Long.valueOf(1));

        Mockito.when(cardRepository.getMaxCardId(Mockito.anyInt())).thenReturn(Long.valueOf(1));
        Card cardMock = Objects.generateCardMock();
        Mockito.when(cardRepository.getCardInfo(Mockito.anyLong())).thenReturn(cardMock);

        List<LastMatchesStats> matchesMock = Objects.getThreeLastMatches();
        Mockito.when(lastMatchesStatsRepository.getLastMatchesPerCardDetStat(Mockito.any(),
                Mockito.any(),Mockito.any())).thenReturn(matchesMock );

        List<CardDetailStats> statsMock = Arrays.asList(new CardDetailStats(
                new CardDetailStatsId(), LocalDate.now(), true, Objects.createTeam(),null,null));
        Mockito.when(cardDetailStatsRepository
        .getStatsPerMatch(Mockito.anyLong(),Mockito.any())).thenReturn(statsMock);
```

Figura 189. Card Controller Integration Test - parte 2.
Elaborado por: El investigador.

CardControllerIntegrationTest.java

```
        assertThat(saveCardRequestDTO.getMatches()).isNotNull();
        assertThat(saveCardRequestDTO.getMatches().size()).isNotZero();
        for (SaveCardDetailRequestDTO item : saveCardRequestDTO.getMatches()) {
            assertThat(item.getId()).isNotNull().isNotZero();
            assertThat(item.getLocalScore()).isNotNull().isNotNegative();
            assertThat(item.getVisitantScore()).isNotNull().isNotNegative();
        }

        mockMvc.perform(post("/api".concat(CardController.CARD_SAVE_URI)).contentType(MediaType.APPLICATION_JSON)
                .content(convertObjectToJsonBytes(saveCardRequestDTO)).header("Authorization", adminTokenTest))
        .andExpect(status().isOk()).andReturn();

        Mockito.verify(cardRepository, Mockito.times(1)).saveAndFlush(Mockito.any());
        assertThat(card.getStatus()).isEqualTo(CardStatusEnum.READY_FOR_PROCESS.getCode());

    }

    @Test
    public void previousCardWhenLastCardIdIsNull() throws Exception {
        Mockito.when(cardRepository.getMaxCardId(Mockito.any())).thenReturn(null);

        MvcResult mvcResult = mockMvc.perform(get("/api".concat(CardController.PREV_CARD_URI)))
                .andExpect(status().isOk()).andReturn();

        String content = mvcResult.getResponse().getContentAsString();
        assertThat(content).isEmpty();

    }

    @Test
    public void previousCardSuccessful() throws Exception {
        Card card = Objects.generateCardMock();
        Mockito.when(cardRepository.getPreviousCard(Mockito.any(),Mockito.any())).thenReturn(Arrays.asList(card));

        MvcResult mvcResult = mockMvc.perform(get("/api".concat(CardController.PREV_CARD_URI)))
                .andExpect(status().isOk()).andReturn();

        String content = mvcResult.getResponse().getContentAsString();
        assertThat(content).isNotEmpty();

    }
```

Figura 190. Card Controller Integration Test - parte 3.
Elaborado por: El investigador.

- **Pruebas de Integración *Prognostic Controller***

PrognosticControllerIntegrationTest.java

```
package ec.com.alquimiasoft.balonazo.integrationtest;

import static ec.com.alquimiasoft.balonazo.testtools.JsonBytes.convertObjectToJsonBytes;

@SpringBootTest
@RunWith(SpringRunner.class)
@AutoConfigureMockMvc
public class PrognosticControllerIntegrationTest {
    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private UserService userService;

    private String tokenTest;

    @Autowired
    private TokenService tokenService;

    @Before
    public void initTest() {
        tokenTest = tokenService.createToken("test",TokenTypeEnum.ACCESS_TOKEN.getCode());
    }

    @Test
    public void getPrognosticsPerUserTest() throws Exception {

        Mockito.when(userService.findByEmail(Mockito.anyString())).thenReturn(null);
        String email = "";
        MvcResult mvcResult =mockMvc.perform(post("/api".concat(PrognosticController.PROGNOSTIC_JSON_URI))
                .header(Objects.AUTHORIZATION_KEY, tokenTest)
                .contentType(MediaType.APPLICATION_JSON)
                .content(convertObjectToJsonBytes(email))
                ).andReturn();
        String content = mvcResult.getResponse().getContentAsString();
        assertThat(content).isEmpty();
    }
}
```

Figura 191. Prognostic Controller Integration Test.
Elaborado por: El investigador.

- **Pruebas de Integración *Result Controller***

ResultsControllerIntegrationTest.java

```java
@SpringBootTest
@RunWith(SpringRunner.class)
@AutoConfigureMockMvc
public class ResultsControllerIntegrationTest {
    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private CardRepository cardRepository;

    @MockBean
    private PrognosticRepository prognosticRepository;

    @Autowired
    private TokenService tokenService;

    @MockBean
    private UserRepository userRepository;

    private String tokenTest;

    @Before
    public void initTest() {
        tokenTest = tokenService.createToken("a@a.com",TokenTypeEnum.ACCESS_TOKEN.getCode());
        User value = Objects.generateUser("a@a.com", "prueba", "prueba");
        Mockito.when(userRepository.findByUsername(Mockito.any())).thenReturn(value);
        Mockito.when(userRepository.findByEmail(Mockito.any())).thenReturn(value);
    }

    @Test
    public void getResultsWhenMaxCardIdIsNull() throws Exception {
        ResultsRequest resultsRequest = new ResultsRequest();
        Mockito.when(cardRepository.getMaxCardId(CardStatusEnum.PROCESSED.getCode())).thenReturn(null);

        MvcResult mvcResult = mockMvc.perform(post("/api".concat(ResultsController.RESULTS_URI))
                .header(Objects.AUTHORIZATION_KEY, tokenTest)
                .contentType(MediaType.APPLICATION_JSON)
                .content(convertObjectToJsonBytes(resultsRequest))).andReturn();
        String content = mvcResult.getResponse().getContentAsString();
        assertThat(content).isNotEmpty();
    }
```

Figura 192. Result Service Integration Test - parte 1.
Elaborado por: El investigador.

```java
    @Test
    public void getResultsSuccessful() throws Exception {
        Mockito.when(cardRepository.getMaxCardId(CardStatusEnum.PROCESSED.getCode())).thenReturn(Long.valueOf(1));

        Card card = Objects.generateCardMock();
        card.getCardDetailList().get(0).setLocalScore(1);
        card.getCardDetailList().get(1).setLocalScore(3);
        card.getCardDetailList().get(2).setVisitorScore(5);
        card.getCardDetailList().get(2).setVisitorScore(6);
        Mockito.when(cardRepository.getCardInfo(Mockito.anyLong())).thenReturn(card);


        List<Prognostic> prognosticList = new ArrayList<>();
        for (int i = 0; i < 12; i++) {
            Prognostic prognostic = Objects.generatePrognostic();
            prognostic.setId(Long.valueOf(i+1));
            prognosticList.add(prognostic);
        }

        assertThat(prognosticList.size()).isNotZero();

        Mockito.when(prognosticRepository.getLastPrognostics(Mockito.any(), Mockito.anyLong(),
                Mockito.any(), Mockito.any())).thenReturn(prognosticList);

        ResultsRequest resultsRequest = new ResultsRequest();

        MvcResult mvcResult = mockMvc.perform(post("/api".concat(ResultsController.RESULTS_URI))
                .header(Objects.AUTHORIZATION_KEY, tokenTest)
                .contentType(MediaType.APPLICATION_JSON).content(convertObjectToJsonBytes(resultsRequest))).andReturn();
        String content = mvcResult.getResponse().getContentAsString();
        assertThat(content).contains("cardList");

    }
```

Figura 193. Result Service Integration Test - parte 2.
Elaborado por: El investigador.

- **Ejecución general de pruebas y reporte Jacoco**

En el desarrollo basado en pruebas, es primordial comprobar que todas las pruebas en conjunto se ejecuten con éxito y que cumplan los requerimientos previos como lo es un porcentaje de cobertura. Alquimiasoft impone que cada aplicación debe tener un porcentaje de al menos 80% en cobertura de pruebas.

Figura 194. Ejecutar todas las pruebas - parte 1.
Elaborado por: El investigador.

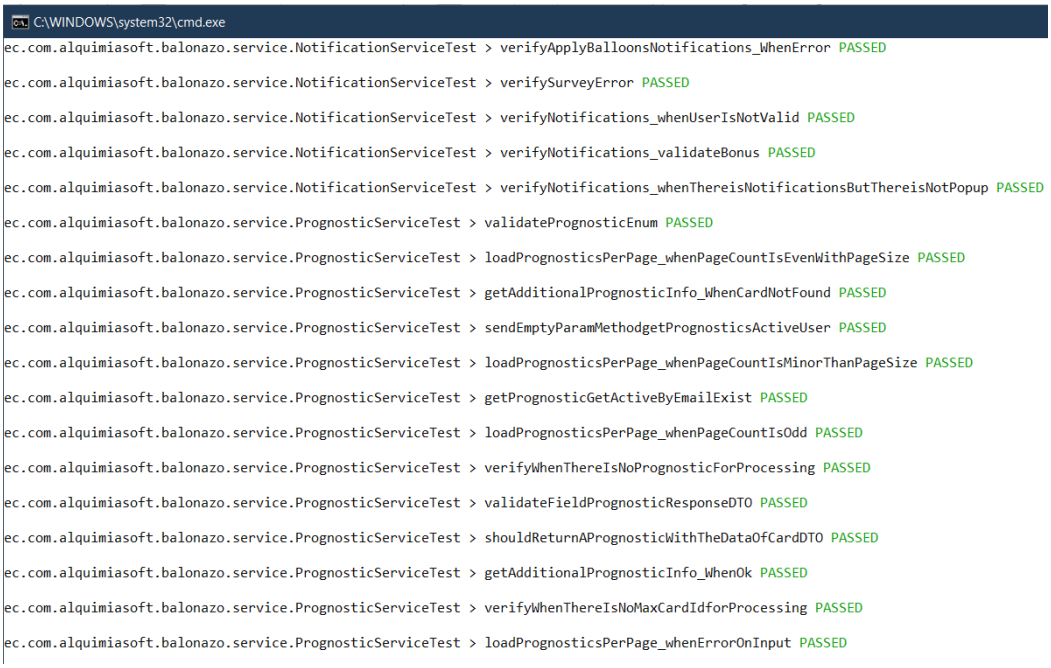


Figura 195. Ejecutar todas las pruebas - parte 2.
Elaborado por: El investigador.

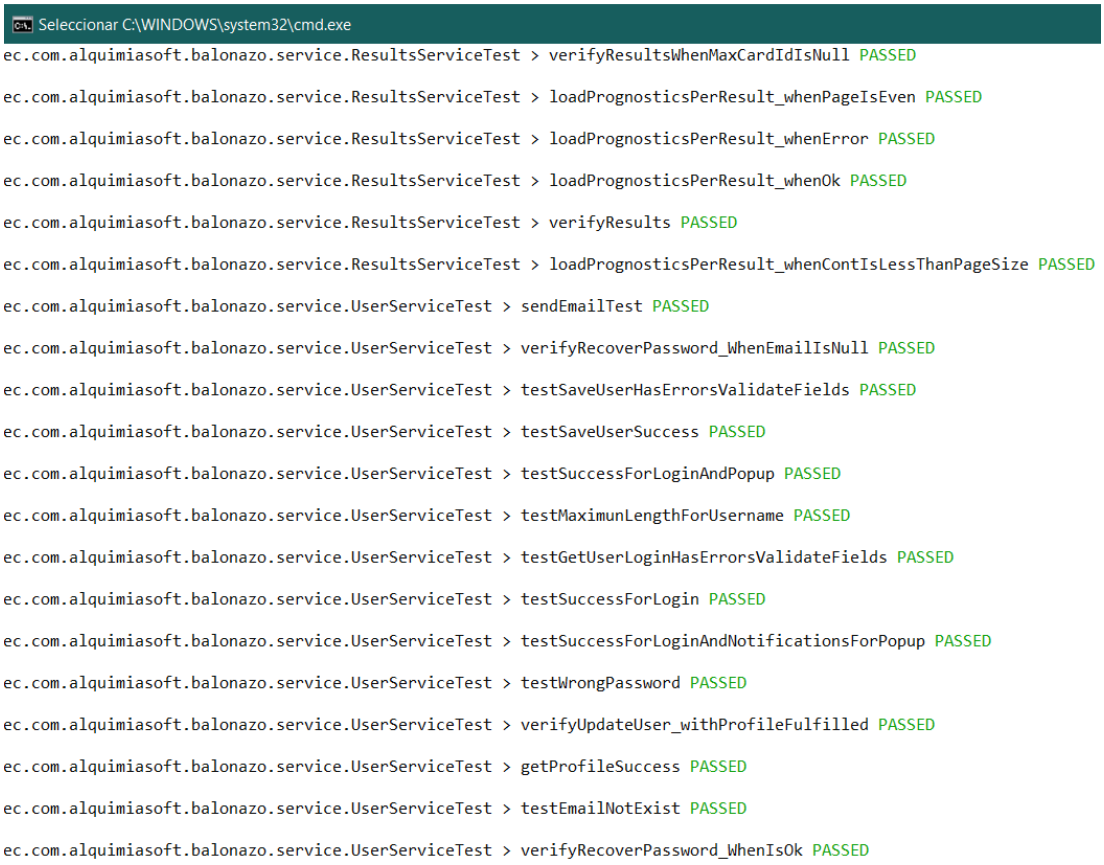
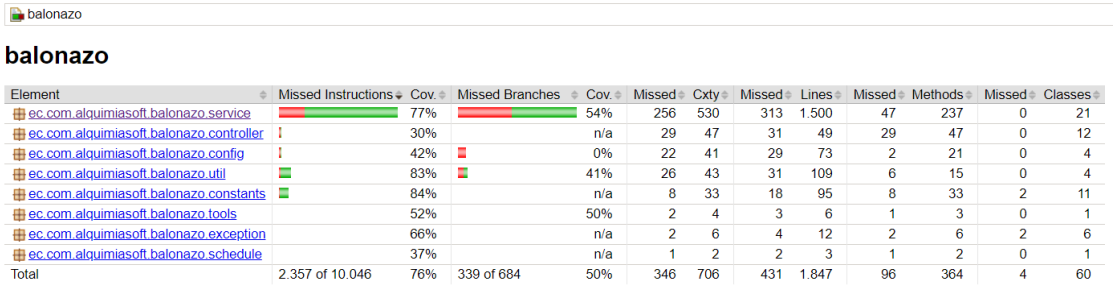


Figura 196. Ejecutar todas las pruebas - parte 3.
Elaborado por: El investigador.



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ec.com.alquimiasoft.balonazo.service		77%		54%	256	530	313	1.500	47	237	0	21
ec.com.alquimiasoft.balonazo.controller		30%		n/a	29	47	31	49	29	47	0	12
ec.com.alquimiasoft.balonazo.config		42%		0%	22	41	29	73	2	21	0	4
ec.com.alquimiasoft.balonazo.util		83%		41%	26	43	31	109	6	15	0	4
ec.com.alquimiasoft.balonazo.constants		84%		n/a	8	33	18	95	8	33	2	11
ec.com.alquimiasoft.balonazo.tools		52%		50%	2	4	3	6	1	3	0	1
ec.com.alquimiasoft.balonazo.exception		66%		n/a	2	6	4	12	2	6	2	6
ec.com.alquimiasoft.balonazo.schedule		37%		n/a	1	2	2	3	1	2	0	1
Total	2.357 of 10.046	76%	339 of 684	50%	346	706	431	1.847	96	364	4	60

Figura 197. Reporte Jacoco Test Total Cobertura.
Elaborado por: El investigador.

3.5. Implementación del sistema

La empresa Alquimiasoft dispone de un servidor centralizado para la implementación de los sistemas que desarrolla, con el fin de aplicar la integración continua y reducir el tiempo de lanzamiento de una aplicación a producción.

Continuos Integration

La empresa Alquimiasoft ha implementado un ambiente de integración continua para que los sistemas desarrollados se puedan integrar a una edad temprana y reducir los errores al momento de integrar varios servicios en una sola aplicación consolidada.

Para implementar una aplicación en integración continua se realizan los siguientes pasos:

1. Crear Jenkinsfile

Jenkinsfile

```
pipeline {
  options {
    disableConcurrentBuilds()
  }
  agent none
  environment {
    SPRING_PROFILES_ACTIVE = 'test'
    PROJECT_NAME = 'balonazo-back'
    SONAR_TOKEN = credentials('alquimiabot-sonar-token')
    REGISTRY_PASS = credentials('alquimiabot-registry-pass')
    ARTIFACTORY_PASS = credentials('alquimiabot-artifactory-pass')
  }
  stages {
    stage('Starting Tests') {
      agent {
          docker {
              image 'gradle:4.10.2'
              label 'docker'
          }
      }
      stages {
        stage('Unit Tests') {
          steps {
            sh 'gradle clean test jacocoTestReport'
            junit 'build/test-results/**/*.xml'
          }
          post {
                always {
                publishHTML target: [
                    allowMissing         : false,
                    alwaysLinkToLastBuild: false,
                    keepAll              : true,
                    reportDir            : 'build/jacocoHtml',
                    reportFiles          : 'index.html',
                    reportName           : 'Coverage Report'
                    ]
                }
           }
         }
        stage('Sonar Analysis') {
          when { anyOf { branch 'develop'; branch 'hotfix/*' } }
          steps {
            sh 'gradle sonarqube -x test -Dsonar.host.url=$SONAR_URL -Dsonar.login=$SONAR_TOKEN'
          }
        }
```

Figura 198. Configurar Jenkinsfile - parte 1.
Elaborado por: El investigador.

En la figura 198 se encuentra:

Stages: Valor para determinar un bloque de pasos que el pipeline va a ejecutar. Para especificar una etapa se utiliza la palabra en singular *stage.*

Enviroment: Bloque para determinar las configuraciones del ambiente en ejecución durante el pipeline.

Línea 14: Bloque de código para iniciar las pruebas unitarias, preparando el ambiente con un contenedor docker de Gradle.

Línea 22: Bloque de código para ejecutar las pruebas unitarias luego de que el ambiente esté preparado.

Línea 40: Bloque de código para validar la cobertura de las pruebas unitarias y el uso de buenas prácticas en el código. Cabe recalcar que si no se cumple los requisitos de sonarqube el pipeline falla.

Jenkinsfile

```
46        stage('Test Coverage') {
47              steps {
48                        sh 'gradle jacocoTestCoverageVerification -x test'
49                    }
50        }
51        stage('Integration Tests') {
52          steps {
53            sh 'gradle integrationTest'
54            junit 'build/test-results/**/*.xml'
55          }
56        }
57        stage('Build Jar') {
58          steps {
59            sh 'gradle bootJar -x test'
60            stash includes: 'build/libs/', name: 'build'
61          }
62        }
63      }
64    }
65    stage('Publish to Registry') {
66      agent { label 'docker' }
67      when { anyOf { branch 'develop'; branch 'hotfix/*' } }
68      steps {
69        unstash 'build'
70        sh 'docker login registry.alquimiasoft.com.ec -u $REGISTRY_USER -p $REGISTRY_PASS'
71        sh 'docker build -t
72        registry.alquimiasoft.com.ec/$PROJECT_NAME:latest -t registry.alquimiasoft.com.ec/$PROJECT_NAME:$GIT_COMMIT .'
73        sh 'docker push registry.alquimiasoft.com.ec/$PROJECT_NAME:latest'
74        sh 'docker push registry.alquimiasoft.com.ec/$PROJECT_NAME:$GIT_COMMIT'
75      }
76    }
77    stage('Tagging Release Image') {
78        agent { label 'integration' }
79        when {
80            branch 'master'
81        }
82        environment {
83            GIT_TAG = """${
84                sh(
85                        returnStdout: true,
86                        script: 'git tag --sort version:refname
87                        -points-at $GIT_COMMIT | grep -E -i -w "beta|release" | tail -n1'
88                )
89            }"""
90        }
```

Figura 199. Configurar Jenkinsfile - parte 2.
Elaborado por: El investigador.

En la figura 198 se encuentra:

Línea 51: Bloque de código para ejecutar las pruebas de integración.

Línea 57: Bloque de código para compilar la aplicación y generar un jar.

Línea 65: Bloque de código para iniciar la publicación de la imagen docker en el servidor registry propio de Alquimiasoft.

Línea 77: Bloque de código para publicar en el registry con un tag en específico, utilizado para descargar imagen en producción.

```
91        steps {
92            sh 'docker login registry.alquimiasoft.com.ec -u $REGISTRY_USER -p $REGISTRY_PASS'
93            sh 'docker pull registry.alquimiasoft.com.ec/$PROJECT_NAME:$GIT_COMMIT'
94            sh 'docker tag
95            registry.alquimiasoft.com.ec/$PROJECT_NAME:$GIT_COMMIT registry.alquimiasoft.com.ec/$PROJECT_NAME:$GIT_TAG'
96                sh 'docker push registry.alquimiasoft.com.ec/$PROJECT_NAME:$GIT_TAG'
97            }
98    }
99
100   stage('Deploy to Integration') {
101     agent { label 'integration' }
102     when { anyOf { branch 'develop'; branch 'hotfix/*' } }
103     steps {
104       sh 'docker login registry.alquimiasoft.com.ec -u $REGISTRY_USER -p $REGISTRY_PASS'
105       sh 'docker-compose -p balonazoBE -f docker-compose-int.yml up -d'
106       sh 'docker-compose -p balonazoBE -f docker-compose-int.yml ps'
107       sh 'docker-compose -p balonazoBE -f docker-compose-int.yml images'
108     }
109   }
110 }
111}
```

Figura 200. Configurar Jenkinsfile - parte 3.
Elaborado por: El investigador.

En la figura 200 se encuentra:

Línea 100: Bloque de código para ejecutar en integración el contenedor docker creado y que sea accesible para los desarrolladores.

2. Crear Multibranch Pipeline

Una vez que la aplicación contiene el archivo Jenkinsfile se procede a la configuración de la aplicación en Jenkins para que ejecute su respectivo pipeline.

La herramienta Jenkins es muy intuitiva y ya nos muestra directo la pantalla de configuración con solo poner el nombre del proyecto a desplegar, tal y como se muestra en las figuras 201 y 202.

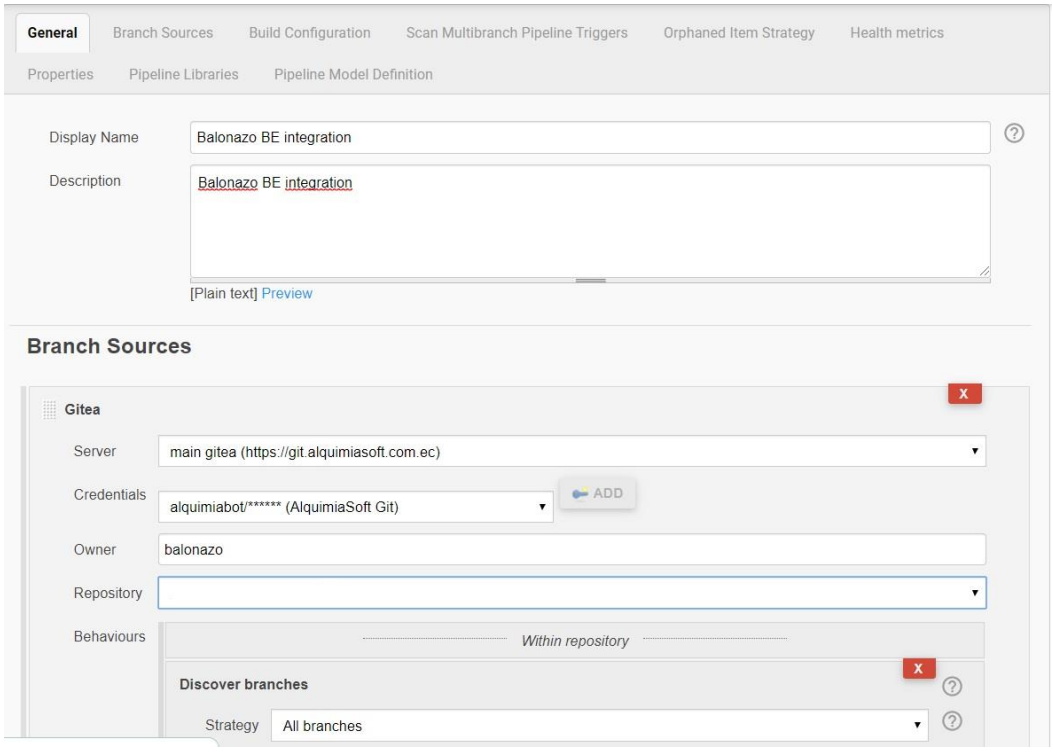


Figura 201. Configurar Jenkins Pipeline - parte 1.
Elaborado por: El investigador.

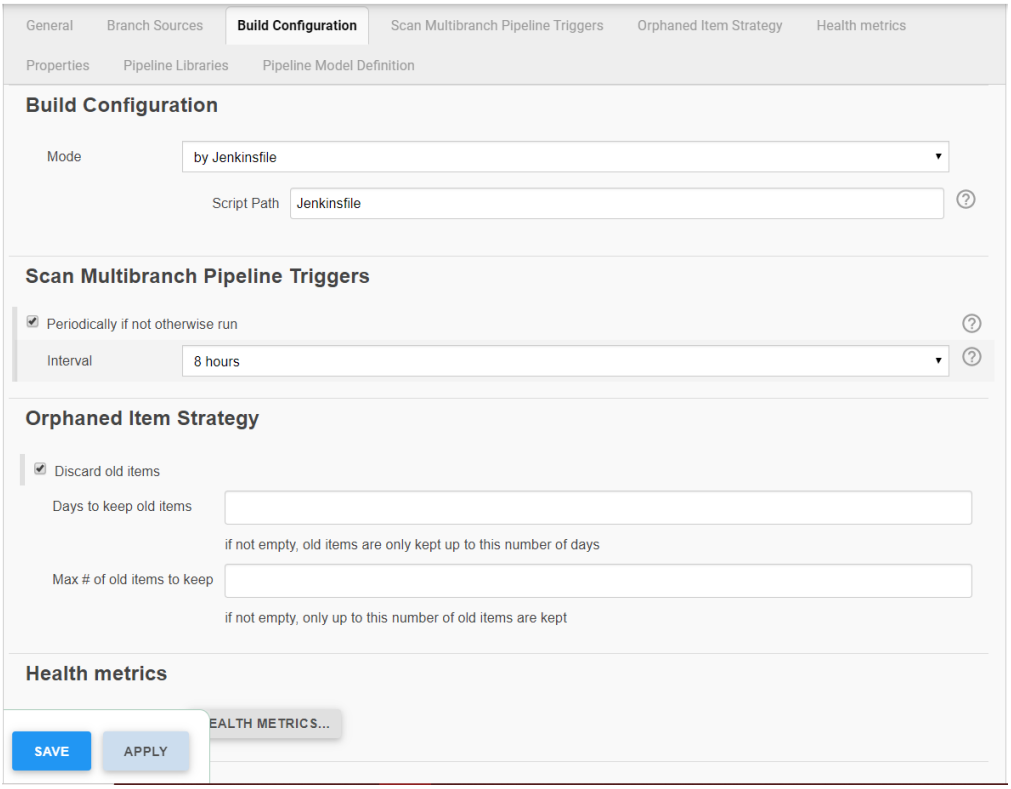


Figura 202. Configurar Jenkins Pipeline - parte 2.
Elaborado por: El investigador.

Los valores establecidos en las configuraciones están establecidos por la documentación proporciona por la empresa Alquimiasoft.

3. Verificar compilación exitosa en Jenkins

Al guardar la configuración del pipeline, Jenkins busca en el repositorio gitea de la empresa el proyecto y procede a ejecutar las etapas establecidas en el pipeline.

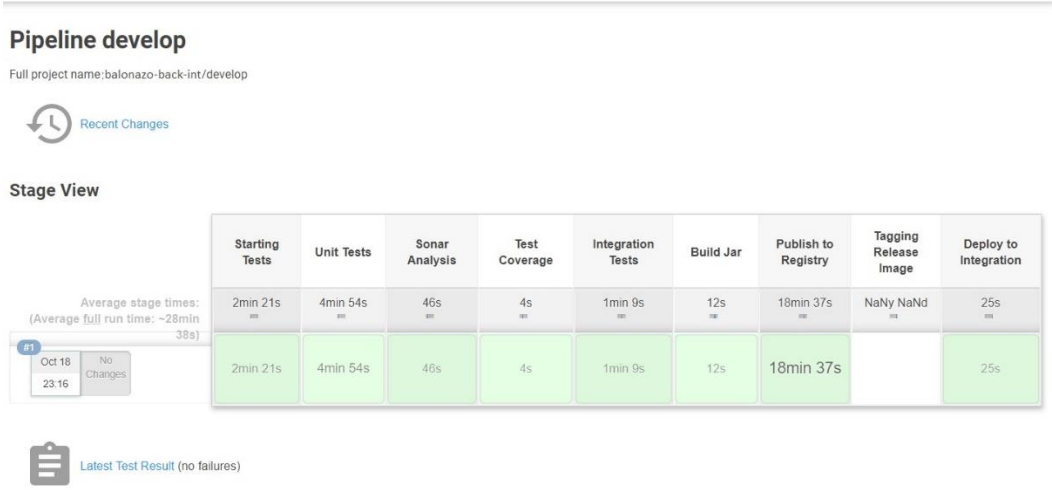


Figura 203. Ejecución Jenkins Pipeline.
Elaborado por: El investigador.

3.6. Resultado Final del Sistema

Luego de implementar el sistema en el ambiente de integración de la empresa Alquimiasoft se puede acceder a él con un puente al servidor.

El sistema es accesible desde la Web y puede ser utilizado en los navegadores Firefox y Google Chrome o cualquier navegador basado en Chromium.

La pantalla principal del sistema muestra la cartilla en juego o vigente, donde se encuentran 13 partidos, conformados por el campeonato nacional de fútbol, serie B, fútbol femenino y campeonatos internacionales. El usuario debe elegir si gana el Local, Empatan o gana el Visitante en cada partido.

Figura 204. Pantalla principal sistema.
Elaborado por: El investigador.

Para enviar una cartilla se debe crear una cuenta, la cual puede ser creada ingresando datos personales o conectándose con una cuenta de Facebook.

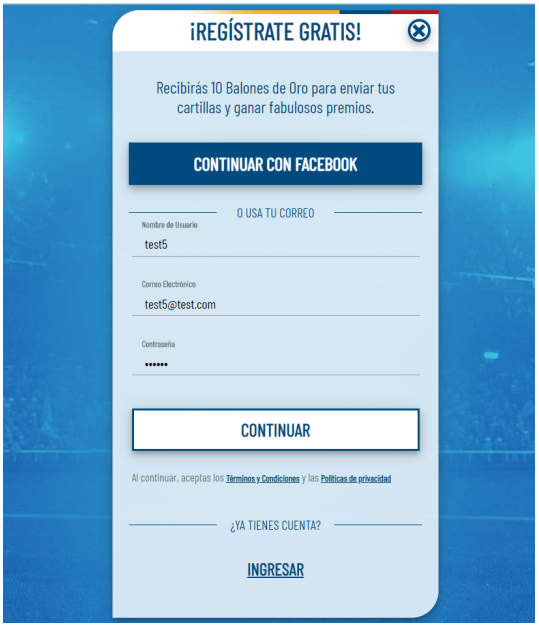


Figura 205. Pantalla crear cuenta.
Elaborado por: El investigador.

También se puede acceder al sistema desde un dispositivo móvil, gracias a que el sistema es responsivo y se adapta a las diferentes resoluciones de dispositivos electrónicos como se muestran en las figuras 206 y 207.

Todos los aspectos funcionales del sistema se encuentran descritos a detalle en el manual de usuario en el Anexo 4.

Figura 206. Pantalla principal sistema versión móvil.
Elaborado por: El investigador.

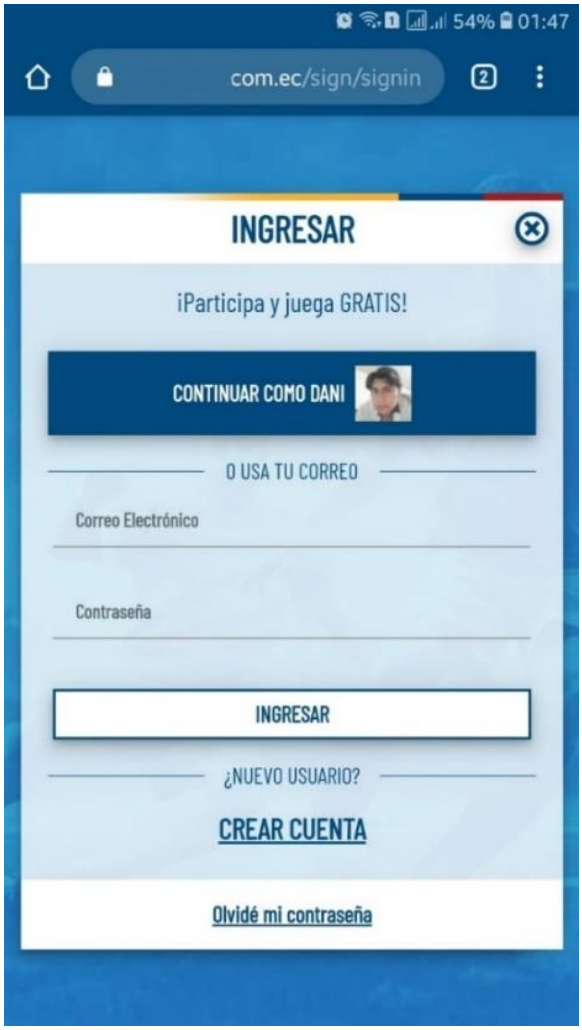


Figura 207. Pantalla crear usuario versión móvil.
Elaborado por: El investigador.

CAPÍTULO IV

CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

- Los resultados obtenidos con las encuestas aplicadas a una muestra de la población total del cantón Ambato ha permitido determinar en conjunto con los desarrolladores, empresa e interesados, los requerimientos funcionales para el desarrollo de un prototipo viable que puede ser implementado y utilizado.

- El tiempo de estudio y aprendizaje de las herramientas de software ha tenido un fuerte impacto en el progreso del proyecto que se pudo controlar con la adaptabilidad de la metodología SCRUM y cumplir con el desarrollo del sistema en los tiempos establecidos.

- Spring Framework se ha popularizado en el desarrollo de aplicaciones empresariales Java y al ser un framework que se encarga de toda su infraestructura, los desarrolladores se pueden centrar totalmente en la lógica de negocio, creando un producto de calidad, liviano, robusto y modular en poco tiempo.

- El desarrollo del sistema de apuestas deportivas en el fútbol con spring framework ha sido fácil y rápido gracias a la extensa documentación que dispone spring en su sitio oficial y también al apoyo de los desarrolladores experimentados de la empresa Alquimiasoft.

- El desarrollo basado en TDD ha permitido crear código funcionalmente_probado, mantenible y robusto en un periodo corto de tiempo, además de aportar a las métricas de calidad en la aplicación final alcanzando el objetivo de crear productos de calidad.

- Al utilizar el proyecto de código abierto Docker se ha logrado automatizar el despliegue de aplicaciones con contenedores, obteniendo los beneficios de portabilidad e integración rápida con otros servicios, además de facilitar la automatización con integración continua, conectando los servicios de Front, Back y Base de datos en un solo sistema funcional.

4.2. Recomendaciones

- Se recomienda investigar el desarrollo de aplicaciones Spring implementando la arquitectura de microservicios que ha tenido mejoras con las últimas actualizaciones de Spring.
- Se sugiere utilizar la herramienta Docker Compose para simplificar el uso de Docker y crear contenedores a base de scripts.
- Se aconseja utilizar sistemas de gestión de base de datos livianos para la ejecución de pruebas como por ejemplo H2, además se recomienda utilizar variables de entorno para establecer configuraciones específicas por ambientes.

BIBLIOGRAFÍA

[1] P. Gomber, P. Rohr, and U. Schweickert, “Sports betting as a new asset class—current market organization and options for development,” *Financ. Mark. Portf. Manag.*, vol. 22, no. 2, pp. 169–192, Jun. 2008.

[2] Álvaro Hernández, “Startups españolas innovando en las apuestas online,” *startupxplore*, 2017. [Online]. Available: https://startupxplore.com/es/blog/innovar-en-el-gallinero-de-las-apuestas-deportivas/. [Accessed: 22-Aug-2019].

[3] H. Lopez-Gonzalez, F. Guerrero-Solé, A. Estévez, and M. Griffiths, “Betting is loving and bettors are predators: A conceptual metaphor approach to online sports betting advertising,” *J. Gambl. Stud.*, vol. 34, no. 3, pp. 709–726, 2018.

[4] Apuestas-Deportivas.es, “Apuestas deportivas: explicación - Guía de apuestas deportivas,” *apuestas-deportivas.es*, 2017. [Online]. Available: https://www.apuestas-deportivas.es/guia-de-apuestas/apuestas-deportivas-explicacion.php. [Accessed: 10-Oct-2019].

[5] D. Gutierrez, “Frameworks y Componentes Universidad de los Andes,” Venezuela, 2010.

[6] Yanina Muradas M., “Conoce qué es Spring Framework y por qué usarlo | OpenWebinars,” *openwebinars.net*, 2018. [Online]. Available: https://openwebinars.net/blog/conoce-que-es-spring-framework-y-por-que-usarlo/. [Accessed: 06-May-2019].

[7] Dataflair Team, “Spring Dependency Injection – Types of Spring DI & Example,” *data-flair.training*, 2018. [Online]. Available: https://data-flair.training/blogs/spring-dependency-injection/. [Accessed: 18-May-2019].

[8] Mangrar, “Spring Framework: Introducción,” *www.genbeta.com*, 2011. [Online]. Available: https://www.genbeta.com/desarrollo/spring-framework-introduccion. [Accessed: 18-May-2019].

[9] LogicBig, “Spring - Different ways of injecting dependencies,”

www.logicbig.com, 2016. [Online]. Available: https://www.logicbig.com/tutorials/spring-framework/spring-core/types-of-dependency-injection.html. [Accessed: 18-May-2019].

[10] DATAFLAIR TEAM, "Spring Framework Architecture | 4 Modules of Spring Architecture - DataFlair," *data-flair.training*, 2018. [Online]. Available: https://data-flair.training/blogs/spring-framework-architecture/. [Accessed: 06-May-2019].

[11] J. Acetozi, "Pro Java clustering and scalability: Building real-time apps with Spring, Cassandra, Redis, WebSocket and RabbitMQ," *Pro Java Clust. Scalability Build. Real-Time Apps with Spring, Cassandra, Redis, WebSocket Rabbit.*, pp. 1–149, 2017.

[12] Oblancarte, "Qué es Spring Boot y su relación con los microservicios," 2018. [Online]. Available: https://www.oscarblancarteblog.com/2018/07/17/spring-boot-relacion-los-microservicios/. [Accessed: 10-Sep-2019].

[13] A. Ben and T. Luke, *Spring Security*, 3.0.8 RELE. docs.spring.io, 2017.

[14] Dataflair Team, "Advantages of Spring Framework With Limitations," 2018. [Online]. Available: https://data-flair.training/blogs/advantages-of-spring/. [Accessed: 29-Sep-2019].

[15] R. Marín, "Los gestores de bases de datos más usados en la actualidad.," 2019. [Online]. Available: https://revistadigital.inesem.es/informatica-y-tics/los-gestores-de-bases-de-datos-mas-usados/. [Accessed: 01-Oct-2019].

[16] V. R. Villán, "Las metodologías ágiles más utilizadas y sus ventajas dentro de la empresa," 2019. .

[17] L. Gonçalves, "Scrum The methodology to become more agile," *Schwerpkt. | Agil. Fram.*, pp. 40–42, 2018.

[18] Redhat, "¿Qué es DOCKER?" [Online]. Available: https://www.redhat.com/es/topics/containers/what-is-docker. [Accessed: 02-Oct-2019].

[19] A. Team, “Explicación de la integración continua.” [Online]. Available: https://aws.amazon.com/es/devops/continuous-integration/. [Accessed: 02-Oct-2019].

[20] Miguel Angel Suma Paucara, “Analisis_comparativo_de_los_IDES_de_Java,” academia.edu, Arequipa, Perú, 2017.

ANEXOS

Anexo N.º 1 – Encuesta para determinar requerimientos y la factibilidad para el desarrollo del sistema de apuestas deportivas online.

UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE INGENIERÍA EN SISTEMA COMPUTACIONALES E INFORMÁTICOS

Encuesta dirigida a la población del cantón Ambato

OBJETIVO:

Determinar los requerimientos y la factibilidad para el desarrollo de un sistema de apuestas deportivas online.

Instrucciones

- Al ser anónima la encuesta responda con toda libertad y sinceridad
- Antes de responder las preguntas, lea detenidamente, reflexione y luego de su opinión.
- Encierre en circulo la(s) respuesta(s) que usted considere apropiada(s).

CUESTIONARIO

1. **¿Cuántas horas al día utiliza el servicio de internet? (1 respuesta)**
    a) 1 hora
    b) 3 horas
    c) 5 horas
    d) 8 horas
    e) Más de 8 horas
2. **De la siguiente lista de dispositivos. ¿Cuáles utiliza usualmente para conectarse a internet? (1 o más respuestas)**
    a) Teléfono móvil
    b) Computadora portátil o de escritorio
    c) Tablet
    d) Smart TV
    e) Otro. Especifique: _____

3. **¿Qué tan importante es para usted el campeonato nacional de fútbol ecuatoriano? (1 respuesta)**
    a) No es importante
    b) Poco importante
    c) Neutral
    d) Importante
    e) Muy importante
4. **¿Qué tan interesado está usted en el fútbol internacional? (1 respuesta)**
    a) Nada interesado
    b) Poco interesado
    c) Neutral
    d) Interesado
    e) Muy interesado
5. **¿Conoce o escuchado sobre las apuestas en línea?**
    a) Sí
    b) No
6. **¿Qué tan probable es que usted participe en juegos de apuestas deportivas en línea? (1 respuesta)**
    a) Nada probable
    b) Poco probable
    c) Neutral
    d) Muy Probable
    e) Definitivamente
7. **¿Cuánto dinero estaría dispuesto a apostar en juegos de apuestas deportivas? (1 respuesta)**
    a) Nada
    b) 1 a 5 dólares
    c) 5 a 10 dólares
    d) 10 a 20 dólares
    e) Más de 20 dólares

Gracias por su colaboración

Anexo N.º 2 - Especificación de Requisitos según el estándar de IEEE 830.

UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE INGENIERÍA EN SISTEMA COMPUTACIONALES E INFORMÁTICOS

1. Estudio de Factibilidad

El estudio de Factibilidad nos permite recabar información relevante sobre el desarrollo del Proyecto para poder tomar decisiones en base a esta, para así poder decidir confiablemente si se procede con el desarrollo o no. Los tipos de factibilidad que se considera para el desarrollo del Sistema de Lotería de apuestas deportivas en el fútbol con spring framework para la empresa Alquimiasoft S.A., se describirán a continuación.

Factibilidad Técnica

Nos permite determinar si la solución propuesta puede ser implementada con los Recursos hardware, software y humanos disponibles. Por lo antes mencionado, se concluye que para el desarrollo del sistema de lotería de apuestas deportivas en el fútbol se cuenta con la mayoría de los recursos de hardware y software necesarios, a continuación, se detallará los recursos de hardware, software y humanos requeridos.

Hardware Existente

CANTIDAD	DESCRIPCIÓN	OBSERVACIÓN
1	Computador de escritorio ACER.	Herramienta para el desarrollo del sistema dentro de la empresa.
1	Laptop ASUS ROG G531GT	Herramienta para el desarrollo del sistema y su respectiva documentación fuera de la empresa.
1	Servidor Google Cloud	Servidor físico contratado por la empresa Alquimiasoft para

		la implementación Continua de sus proyectos.
1	Infraestructura de Red	Ya existente en el área de trabajo donde se llevará a cabo el desarrollo.
1	Impresora	Impresión de los informes o documentos necesarios.

Software Existente

NOMBRE	DESCRIPCIÓN	ESTADO	OBSERVACIÓN
Eclipse Photon	IDE de desarrollo	Legal	Licencia Libre
Windows 10	Sistema Operativo	Legal	Licencia digital adquirida por la empresa.
PostgreSQL 9.5	Gestor de Base de Datos	Legal	Licencia Libre
JDK 1.8	Kit de Desarrollo para JAVA	Legal	Licencia Libre
Jenkins versión 2.17	Software de Integración Continua	Legal	Licencia Libre

Cabe mencionar que las versiones de software utilizadas pueden variar según la disponibilidad de versiones, dado que en su mayoría son software libre se actualizarán sin incurrir en gastos adicionales.

Software Requerido

NOMBRE	DESCRIPCIÓN	ESTADO	OBSERVACIÓN
Angular versión 7	Framework de Desarrollo para aplicaciones Web con TypeScript	Legal	Licencia Libre

Spring Framework versión 2.1.5	Framework de Desarrollo para JAVA Empresarial	Legal	Licencia Libre

Recurso Humano Requerido

FUNCIÓN	FORMACIÓN	EXPERIENCIA
Desarrollador Back End	Egresado de la carrera de Ingeniería en Sistemas	Desarrollo en Java 8
Administrador de Base de Datos	Egresado de la carrera de Ingeniería en Sistemas	PostgreSQL
Desarrollador Front End	Ingeniero en Sistemas o afines	Desarrollo web en Angular y IONIC
Diseñador	Diseñador gráfico o afines	Diseño de sistemas Web

Factibilidad Operativa

Esta factibilidad nos permite determinar la probabilidad de que el nuevo Sistema se use de forma correcta por parte de los usuarios del Sistema.

Factibilidad Legal

El sistema de lotería de apuestas deportivas en el fútbol tiene reservado todos los derechos de autor, por lo que cualquier copia parcial o total debe ser autorizada por el autor según la Ley de Propiedad Intelectual. Según lo estipula en la legislación ecuatoriana que consta en el código penal con la ley de comercio electrónico, firmas electrónicas y mensajes de datos.

Con respecto al ámbito legal en el país por ser un sistema de apuestas, el desarrollo del proyecto es factible, ya que el gobierno ecuatoriano de acuerdo con el Art. 2 de la ley de ventas por sorteos y el Art. 5 del reglamento de la ley de ventas de bienes y sorteos, no prohíbe mediante un proscrito formal los juegos de lotería y azar por medios digitales (online).

Esto asegura que el Sistema que se está proponiendo es legalmente factible, ya que actualmente no existe ningún impedimento que este sistema se realizase.

2. Planificación y Análisis de Riesgos

Nomenclatura utilizada:

RP: Riesgo del Proyecto

RT: Riesgo Técnico

RN: Riesgo del Negocio

Identificación de Riesgos

Nº	**Riesgo**	**Categoría**	**Consecuencia**
R1	Las herramientas utilizadas para el desarrollo del proyecto se vuelvan complejas de aprender.	RT	Retraso en el proyecto
R2	El tiempo estimado para terminar el proyecto es muy poco y no se logra concluirlo.	RT	Demora en la entrega del proyecto, Software inadecuado
R3	La computadora donde se estaba elaborando el proyecto se dañó y la información no se puede recuperar.	RP	Cancelación del proyecto
R4	Al no definir correctamente los requerimientos, la solución software no cumplirá con las expectativas del cliente por lo que nuevamente se debe plantear los requerimientos.	RT	Planificación inadecuada
R5	Se cambian continuamente los requerimientos.	RP, RN	Demora en la entrega del proyecto

Categorización del Riesgo

Determinación de la probabilidad:

PORCENTAJE	DESCRIPCIÓN	VALOR
1% - 33%	Bajo	1
34% - 67%	Medio	2
68% - 99%	Alto	3

Determinación del Impacto:

IMPACTO	COSTO	RETRASO	IMPACTO TÉCNICO	VALOR
Bajo	< 1%	1 semana	Ligero impacto en el desarrollo del proyecto.	1
Medio	< 5%	2 semanas	Moderado impacto en el desarrollo del proyecto.	2
Alto	< 10%	1 mes	Severo impacto en el desarrollo del proyecto.	3
Critico	> 10%	> 1 mes	No se puede terminar el proyecto.	4

Determinación de la Exposición de riesgo

IMPACTO / PROBABILIDAD	BAJO=1	MEDIO=2	ALTO=3	CRÍTICO=4
Alta=3	3	6	9	12
Media=2	2	4	6	8
Baja=1	1	2	3	4

Código de Colores según la Exposición de Riesgo

EXPO. RIESGO	VALOR	COLOR
Baja	1 - 2	Verde
Media	3 - 4	Amarillo
Alta	>= 6	Rojo

Determinación de prioridad de riesgos

Riesgo	PROBABILIDAD			IMPACTO		EXPO. RIESGO	
	%	Valor	Prob.	Valor	Impacto	Valor	Expo.
R1	35%	2	Media	3	Alto	6	Alta
R3	40%	2	Media	3	Alto	6	Alta
R5	20%	1	Baja	3	Alto	3	Medio
R2	10%	1	Baja	3	Alto	3	Medio
R4	10%	1	Baja	2	Medio	2	Baja

Hojas de Riesgo

HOJA DE INFORMACION DEL RIESGO			
ID Riesgo: R1		**Fecha:** 05 de octubre del 2019	
Probabilidad: Media **Valor:** 2	**Impacto:** Alto **Valor:** 3	**Exposición:** Alta **Valor:** 6	**Prioridad:** 1
Descripción: Las herramientas utilizadas para el desarrollo del proyecto se vuelvan complejas de aprender o utilizar.			
Refinamiento: **Causas:** Poca información de las herramientas a utilizar, falta de interés por parte de los desarrolladores del proyecto.			

Consecuencia: Retraso en la entrega del Proyecto.
Reducción y supervisión: Buscar información acerca de las herramientas, pedir ayuda a terceras personas, poner empeño en aprender a utilizar las nuevas herramientas.
Gestión: Preparar al personal del proyecto mediante cursos, información bibliográfica y consultas acercas de las herramientas.
Estado actual: **Fase de Reducción Iniciada** ■ **Fase de Supervisión** □ **Fase de Gestión** □
Responsable: Determinado por la Empresa

HOJA DE INFORMACION DEL RIESGO			
ID Riesgo: R2		**Fecha:** 25 de marzo del 2015	
Probabilidad: Baja **Valor:** 1	**Impacto:** Alto **Valor:** 3	**Exposición:** Medio **Valor:** 3	**Prioridad:** 1
Descripción: El tiempo estimado para la terminar el proyecto es muy poco y no se logre concluirlo.			
Refinamiento: **Causas:** Mala planificación del proyecto.			
Consecuencia: Demora en la entrega del proyecto.			
Reducción y supervisión:			

Revisión y refinamiento de la planificación, cooperación entre todo el personal del proyecto para mejorar los tiempos de desarrollo.
Gestión: Informar a los dueños del proyecto y establecer la nueva fecha de entrega del proyecto.
Estado actual: **Fase de Reducción Iniciada** ☐ **Fase de Supervisión** ■ **Fase de Gestión** ☐
Responsable: Determinado por la Empresa

HOJA DE INFORMACION DEL RIESGO			
ID Riesgo: R3		**Fecha:** 25 de marzo del 2015	
Probabilidad: Media **Valor:** 2	**Impacto:** Alto **Valor:** 3	**Exposición:** Alta **Valor:** 6	**Prioridad:** 1
Descripción: La computadora donde se estaba elaborando el proyecto se daña y la información no se puede recuperar.			
Refinamiento: **Causas:** Corto circuito, daño en el disco duro, caída de la computadora al suelo.			
Consecuencia: Cancelación del Proyecto.			
Reducción y supervisión: Respaldar el proyecto con controladores de versiones.			
Gestión: Utilizar el software de controlador de versiones GIT privado de la empresa.			

Estado actual:	
Fase de Reducción Iniciada	■
Fase de Supervisión	□
Fase de Gestión	□
Responsable: Determinado por la Empresa	

HOJA DE INFORMACION DEL RIESGO			
ID Riesgo: R4		**Fecha:** 25 de marzo del 2015	
Probabilidad: Baja **Valor:** 1	**Impacto:** Medio **Valor:** 2	**Exposición:** Baja **Valor:** 2	**Prioridad:** 1
Descripción: Al no definir correctamente los requerimientos, la solución software no cumplirá con las expectativas del cliente por lo que nuevamente se debe plantear los requerimientos.			
Refinamiento: **Causas:** Mala Planificación del Proyecto.			
Consecuencia: Retraso en la elaboración del proyecto.			
Reducción y supervisión: Tener claro lo que el cliente realmente quiere que el proyecto solucione.			
Gestión: Al inicio del proyecto definir claramente lo requisitos, los que realmente necesita el cliente.			

Estado actual:	
Fase de Reducción Iniciada	■
Fase de Supervisión	□
Fase de Gestión	□
Responsable: Determinado por la Empresa	

HOJA DE INFORMACION DEL RIESGO			
ID Riesgo: R5		**Fecha:** 25 de marzo del 2015	
Probabilidad: Baja **Valor:** 1	**Impacto:** Alto **Valor:** 3	**Exposición:** Medio **Valor:** 3	**Prioridad:** 1
Descripción: Se cambian continuamente los requerimientos.			
Refinamiento: **Causas:** El cliente del producto no sabe lo que realmente quiere.			
Consecuencia: Perdida de tiempo, o no se realice el proyecto.			
Reducción y supervisión: Orientar al Cliente del producto de lo que el realmente necesita.			
Gestión: Mantener una constante comunicación con el cliente.			
Estado actual:			
Fase de Reducción Iniciada	■		
Fase de Supervisión	□		
Fase de Gestión	□		

Responsable: Determinado por la Empresa

3. Metodología de Desarrollo

Para el desarrollo del sistema se hará uso de la metodología ágil SCRUM, aplicando sus recomendaciones y adaptándolas a las gestiones de la empresa.

4. Personas y Roles del Proyecto

El equipo de desarrollo que ha llevado a cabo este proyecto es el siguiente:

Rol	**Persona**	**Responsabilidad**
Desarrollador Back End	Sr. Daniel Sandoval	Codificar
Desarrollador Front End	Ing. Christian Cartes	Desarrollar el sistema Web
Encargado de Pruebas (Tester)	Ing. MSc. Gonzalo Calahorrano	Ejecutar las pruebas funcionales.
Diseñador	Ing. Javier Fuentes	Diseñar interfaces del sistema
DevOps	Ing. MSc. Saúl Piña	Apoyo técnico para el desarrollo del proyecto
Desarrollador	Ing. MSc. Pablo Solorzano	Consultar interno con experiencia en Spring Framework

5. Recursos Empleados

Recursos hardware
Servidor Google Cloud
Computadores
Impresora
USB
DVD

Recursos software	
Sistema Operativo	Windows 10
Editor de Texto	Microsoft Office 365
Modelador de bases de datos	DBeaver Community
IDE de Desarrollo Back End	Eclipse Photon
IDE de Desarrollo Front EndV	Visual Code
Framework Front End	Angular 7
Framework Back End	Spring 2.1.5
Gestor de bases de datos	PostgreSql 9.5
Navegadores	Mozilla Firefox, Google Chrome.

Otros
Internet
Servicios básicos
Transporte

Anexo N.º 3 – Catálogo de las historias de usuario del sistema de lotería de apuestas deportivas en el fútbol.

Diseño de la Base de Datos	
ID	1
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que se diseñe el modelo de la base de datos para el sistema, tomando en cuenta los requerimientos de funcionalidad discutidos con anterioridad PARA almacenar y administrar la información que manejará el sistema.
Criterios de aceptación	• La base de datos tiene concordancia con los requerimientos establecidos.

Estructura inicial del proyecto	
ID	2
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que se cree la estructura inicial del proyecto en Spring Framework, tomando en cuenta el manejo de Spring Security, Spring Data y Pruebas PARA iniciar con el proceso de desarrollo del sistema. El repositorio por defecto para las librerías será Gradle y para las pruebas se utilizará JUnit en sus versiones más actuales.
Criterios de aceptación	• El proyecto tiene la estructura recomendada por Spring. • El proyecto cuenta con una sección de pruebas. • El proyecto usa Gradle como repositorio para las librerías del sistema. • El proyecto compila y se ejecuta sin problemas.

Construcción de scripts docker	
ID	3
Prioridad	Media
Riesgo	Medio

Descripción	YO como Product Owner QUIERO que se construya scripts docker del sistema en Spring Framework PARA tener la aplicación aislada en contenedores y mantener la estructura de desarrollo de proyectos de la empresa.
Criterios de aceptación	• El script se ejecuta y crea un contenedor con la aplicación Spring. • El contenedor se crea sin problemas.

Login y registro	
ID	4
Prioridad	Alto
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que un usuario pueda crear una cuenta dado un nombre de usuario, correo electrónico y contraseña, también que pueda ingresar al sistema si ya tiene una cuenta dado un correo electrónico y clave PARA permitirle a un usuario tener acceso a los servicios que ofrece el sistema. La información se valida en Front End y Back End y finalmente se guarda en la base de datos.
Criterios de aceptación	• Dado: que se ha ingresado un correo electrónico ya existente en DB. Cuando: el usuario se registra al sistema. Entonces: no registrar la información y devolver un código y mensaje de error a Front End. • Existen validaciones en Back End • Dado: que el usuario ha ingresado un correo electrónico inexistente en BD o una clave errónea. Cuando: el usuario intenta ingresar al sistema. Entonces: devolver a Front End un código de error y un mensaje de credenciales incorrectas.

Configuración Spring Security	
ID	5
Prioridad	Alta
Riesgo	Alto

Descripción	YO como Product Owner QUIERO que el sistema implemente las capas de seguridad propias de Spring Security PARA asegurar las peticiones al servidor y evitar ataques de hacking.
Criterios de aceptación	• El sistema tiene configurado reglas de Spring Security. • Las peticiones están protegidas por Spring Security

Implementación JWT	
ID	6
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que se implemente el manejo de tokens con JWT en las peticiones al servidor que lo requieran PARA validar las peticiones que llegan al servidor y poder bloquearlas desde las capas de Spring Security y las propias peticiones.
Criterios de aceptación	• Dado: se realice una petición a Back End sin access token a un servicio que lo requiera. Cuando: un usuario externo intenta acceder a un punto de acceso de datos en el servidor. Entonces: el sistema deberá responder con código de error 401. • Dado: se realice una petición a Back End sin access token a un servicio que no lo requiera. Cuando: un usuario accede un servicio del sistema sin restricción. Entonces: el sistema le permitirá ejecutar el proceso correspondiente en ese servicio.

Mostrar cartilla para jugar	
ID	7
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que se implemente un servicio que retorne la cartilla de partidos que se van a jugar en la semana vigente PARA mostrar al usuario la cartilla con la

	que puede realizar sus apuestas. La cartilla debe contemplar la fecha de vigencia.
Criterios de aceptación	• Dado: que Front End solicite la cartilla en juego. Cuando: el usuario acceda al home del sistema. Entonces: Back End debe retornar la cartilla de los partidos en juego en formato JSON. • El servicio de cartilla en juego no debe pedir un access token.

Guardar pronósticos por usuario	
ID	8
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que se guarde los pronósticos de los partidos de la cartilla en juego que un usuario haya jugado PARA almacenar y administrar la información dentro del sistema y procesar resultados más adelante. El sistema debe validar los campos como obligatorios. El costo de envío de un pronóstico será de 1 balón.
Criterios de aceptación	• Dado: el usuario haya llenado sus pronósticos de la cartilla y quiera guardar esa información. Cuando: el usuario de clic en ENVIAR. Entonces: el sistema valida el usuario y procede a guardar la información en DB, además restará de los balones totales del usuario el costo de envío del pronóstico. • El servicio de guardado de pronósticos debe pedir un access token. • Dado: el usuario haya llenado sus pronósticos de la cartilla y quiera guardar esa información. Cuando: el usuario no tenga balones para apostar y de clic en ENVIAR. Entonces: el sistema validará la disponibilidad de balones y retornará un código de error y un mensaje.

Actualización de los datos de un usuario	
ID	9
Prioridad	Baja
Riesgo	Bajo
Descripción	YO como Product Owner QUIERO que un usuario pueda editar su información personal desde el sistema PARA mantener la información de un usuario actualizada y recolectar más información de este. Los campos únicos como correo electrónico no se pueden actualizar. La información debe ser validada antes de guardar en DB. Todos los campos tienen carácter obligatorio.
Criterios de aceptación	• Dado: un usuario llene los campos de su información personal. Cuando: el usuario intenta actualizar su información personal. Entonces: el sistema valida el usuario y procede a guardar la información y notifica el suceso. • Dado: un usuario infringe en las condiciones de ingreso de información. Cuando: el usuario intenta actualizar su información personal. Entonces: el sistema valida la información y de no ser válida, el sistema retorna un código de error y un mensaje.

Manejo de estadísticas por partido	
ID	10
Prioridad	Media
Riesgo	Bajo
Descripción	YO como Product Owner QUIERO que la cartilla en juego tenga estadísticas de los partidos anteriores y de las probabilidades de victoria y empate de los equipos en cada partido PARA que el usuario tenga una ayuda al momento de elegir su pronóstico en los partidos que tenga dudas. Debe existir un campo para identificar que los pronósticos están activos.

Criterios de aceptación	• Dado: que Front End solicite la cartilla en juego. Cuando: el usuario acceda al home del sistema Entonces: Back End debe retornar la cartilla en juego adicionando las estadísticas por partido.

Manejo de comentarios por partido	
ID	11
Prioridad	Media
Riesgo	Bajo
Descripción	YO como Product Owner QUIERO que la cartilla en juego tenga comentarios de los partidos a jugarse PARA ayudarle al usuario a elegir su pronóstico en base a opiniones de expertos en el fútbol.
Criterios de aceptación	• Dado: que Front End solicite la cartilla en juego. Cuando: el usuario acceda al home del sistema Entonces: Back End debe retornar la cartilla en juego adicionando los comentarios por partido.

Entrega de balones por registro	
ID	12
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que un usuario que se registren en el sistema reciba como regalo 10 balones (la moneda virtual del sistema) PARA que pueda apostar sus cartillas dado que el costo de envío de un pronóstico será de 1 balón.
Criterios de aceptación	• Dado: que el usuario se haya ingresado por primera vez al sistema. Cuando: el usuario se haya registrado en el sistema. Entonces: El sistema añadirá en su cuenta 10 balones. Front End notificará del regalo.

Integración de Jacoco para pruebas unitarias	
ID	13
Prioridad	Media

Riesgo	Medio
Descripción	YO como Product Owner QUIERO que el sistema implemente pruebas unitarias con JUnit e implemente jacoco en el proceso PARA generar un reporte de cobertura de pruebas.
Criterios de aceptación	• Dado: que el sistema ejecuta pruebas unitarias. Cuando: el desarrollador o sistema ejecuten el comando de ejecución de pruebas unitarias Entonces: el sistema genera un reporte en html con la cobertura de pruebas unitarias.

Administración proceso crear cartilla	
ID	14
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que el sistema permita a un administrador ingrese los partidos de la cartilla a jugarse en la semana PARA que se pueda generar las cartillas a jugarse desde una interfaz gráfica. El sistema deberá validar los campos como obligatorios.
Criterios de aceptación	• Dado: que se ha ingresado la información de la nueva cartilla a jugarse y se procede a guardar la información. Cuando: el administrador intente crear una nueva cartilla a jugarse y de clic en ENVIAR Entonces: Back End deberá validar la información y proceder a guardar la información en DB.

Administración proceso cerrar cartilla	
ID	15
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que el sistema permita a un administrador ingrese los resultados de los partidos PARA que se pueda iniciar el proceso de cierre de la cartilla. El sistema deberá validar los campos como obligatorios.

Criterios de aceptación	• Dado: que se ha ingresado la información de los resultados de los partidos de la cartilla ya jugada y se procede a guardar la información. Cuando: el administrador intente cerrar una cartilla y de clic en ENVIAR Entonces: Back End deberá validar la información y proceder a actualizar la información de la cartilla en DB.

Procesar resultados	
ID	16
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que el sistema procese la información ingresada al momento de cerrar cartilla PARA que se pueda dar por cerrada y procesada una cartilla y se generen resultados. El proceso es automático.
Criterios de aceptación	• Dado: que se ha cerrado una cartilla Cuando: el administrador ingresó los resultados de la cartilla. Entonces: Back End deberá ejecutar automáticamente el proceso de cerrado de la cartilla y procesamiento de resultados.

Servicio envío de correo a ganadores	
ID	17
Prioridad	Media
Riesgo	Medio
Descripción	YO como Product Owner QUIERO que los usuarios que hayan obtenido un número n de aciertos reciban un correo electrónico PARA que sean notificados que son ganadores.
Criterios de aceptación	• Dado: que un usuario haya obtenido un número n de aciertos en sus pronósticos Cuando: se han procesado los resultados de la cartilla Entonces: el sistema enviará un correo electrónico de ganador a ese usuario.

Asignación de balones a ganadores	
ID	18
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que los usuarios que hayan obtenido un número n de aciertos se les asigne balones a sus respectivas cuentas PARA que aumente su número de balones y pueda seguir apostando en el juego. El número n de aciertos es parametrizable. Según el número de aciertos varía la cantidad de balones de premio.
Criterios de aceptación	• Dado: que un usuario haya obtenido un número n de aciertos en sus pronósticos Cuando: se han procesado los resultados de la cartilla Entonces: el sistema aumentará el número de balones en la cuenta del usuario.

Servicio de resultados	
ID	19
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que el sistema cuente con un servicio para obtener los resultados de los pronósticos de cada usuario PARA que los usuarios puedan consultar el número de aciertos obtenidos en sus pronósticos jugados.
Criterios de aceptación	• Dado: que Front End solicite los resultados de los pronósticos del usuario Cuando: el usuario ingrese a la sección resultados del sistema Entonces: Back End validará la petición del usuario y devolverá una lista con los resultados de los pronósticos de ese usuario.

Envío de correos electrónicos por registro	
ID	20
Prioridad	Media

Riesgo	Medio
Descripción	YO como Product Owner QUIERO que el sistema envié un correo electrónico a los usuarios que se registran en el sistema PARA que reciban un correo de bienvenida al juego.
Criterios de aceptación	• Dado: que un usuario haya ingresado por primera vez al sistema. Cuando: el usuario se haya registrado en el sistema Entonces: el sistema enviará un correo electrónico de bienvenida.

Restaurar contraseña	
ID	21
Prioridad	Media
Riesgo	Medio
Descripción	YO como Product Owner QUIERO que los usuarios puedan restaurar su contraseña en caso de olvido dado un correo electrónico PARA que puedan volver a acceder al sistema.
Criterios de aceptación	• Dado: que un usuario haya ingresado su correo electrónico Cuando: el usuario intente restaurar su contraseña Entonces: el sistema validará la existencia de la cuenta por correo electrónico y asignará una contraseña provisional aleatoria en la cuenta del usuario y enviará un correo electrónico al usuario con la nueva contraseña. • Dado: que un usuario haya ingresado su correo electrónico que no existe en la DB del sistema. Cuando: el usuario intente restaurar su contraseña. Entonces: el sistema validará la existencia de la cuenta por correo electrónico y retornará un código de error y un mensaje.

Servicio de contactos	
ID	22
Prioridad	Baja
Riesgo	Bajo

Descripción	YO como Product Owner QUIERO que el sistema cuente con un servicio de contacto PARA que un usuario pueda contactar a los administradores del sistema.
Criterios de aceptación	• Dado: que un usuario haya ingresado los datos en el formulario de contactos. Cuando: el usuario intente enviar un mensaje de contacto al administrador del sistema. Entonces: el sistema valida la información y envía un correo electrónico a la cuenta de correo de la empresa.

Administración de estadísticas	
ID	23
Prioridad	Media
Riesgo	Medio
Descripción	YO como Product Owner QUIERO que el sistema permita a un administrador ingresar y actualizar las estadísticas de cada partido de la cartilla en juego PARA que se pueda administrar las estadísticas de los partidos por medio de una interfaz gráfica.
Criterios de aceptación	• Dado: que el administrador ingrese la información de las estadísticas de un partido. Cuando: el administrador intente guardar la información de las estadísticas, Entonces: el sistema validará la información y procederá a guardar las estadísticas en la DB del sistema.

Administración de comentarios	
ID	24
Prioridad	Media
Riesgo	Medio
Descripción	YO como Product Owner QUIERO que el administrador pueda ingresar comentarios por cada partido de la cartilla en juego PARA que se pueda administrar los comentarios por medio de una interfaz gráfica. Los campos deben estar validados.

Criterios de aceptación	• Dado: que el administrador ingrese la información de los comentarios de un partido. Cuando: el administrador intente guardar la información de los comentarios. Entonces: el sistema validará la información y procederá a guardar los comentarios en la DB del sistema.

Ambiente jugable en la nube (Continuos Integration)	
ID	25
Prioridad	Alta
Riesgo	Medio
Descripción	YO como Product Owner QUIERO que el sistema esté implementado en el servidor de proyectos de la empresa PARA que el sistema pueda ser accesible desde el internet y los elementos del sistema se puedan integrarse de manera continua durante el desarrollo.
Criterios de aceptación	• El sistema tiene un archivo de configuración Jenkins • El sistema se compila en el servidor de Integración Continua de la empresa. • Se puede acceder al sistema desde cualquier computador mediante túnel.

Sistema de notificaciones	
ID	26
Prioridad	Alta
Riesgo	Alto
Descripción	YO como Product Owner QUIERO que el sistema maneje servicios de notificaciones PARA permitirle al usuario reclamar recompensas y notificarle de algún tipo de evento en el futuro.
Criterios de aceptación	• Dado: que Front End solicite las notificaciones pertenecientes a un usuario. Cuando: el usuario ingrese al sistema. Entonces: Back End responderá con una lista de notificaciones del usuario.

Pruebas de integración	
ID	27
Prioridad	Media
Riesgo	Medio
Descripción	YO como Product Owner QUIERO que el sistema cuente con pruebas de integración de los servicios REST PARA que el desarrollo cuente con pruebas de integración y reduca los daños críticos en caso de fallos.
Criterios de aceptación	• El sistema cuenta con pruebas de integración de sus servicios REST.

Servicio cartillas enviadas por usuario	
ID	28
Prioridad	Media
Riesgo	Medio
Descripción	YO como Product Owner QUIERO que el sistema cuente con un servicio para obtener los pronósticos enviados por un usuario en cualquier fecha PARA que el usuario tenga acceso a un histórico de sus pronósticos enviados.
Criterios de aceptación	• Dado: que Front End solicite el histórico de pronósticos de un usuario. Cuando: el usuario acceda a la sección mis cartillas enviadas. Entonces: el sistema validará la petición del usuario y retornará la lista de pronósticos del usuario dado una fecha concreta. • El servicio debe pedir un access token.

Anexo N.º 4. Manual de Usuario Final.

El presente manual tiene como objetivo mostrar el uso del sistema a los usuarios finales y conocer sus funcionalidades.

Registro de un Usuario.

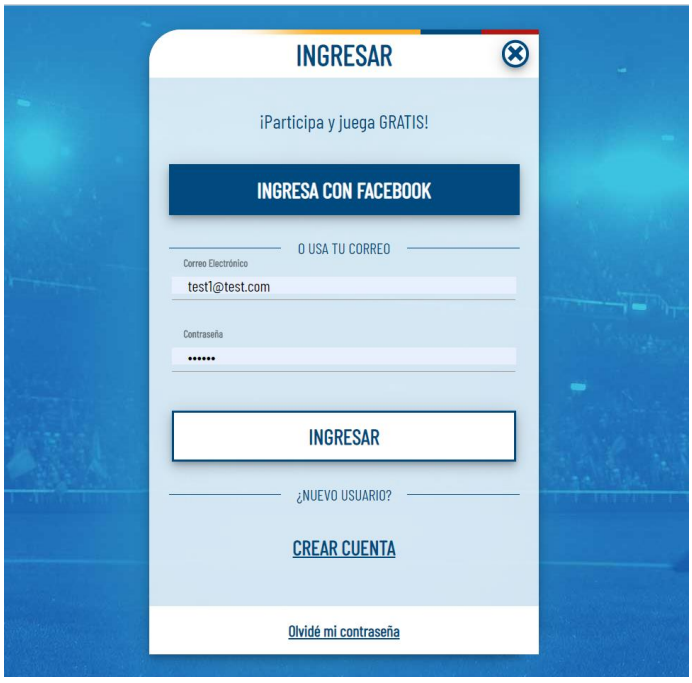
Se puede acceder al sistema desde cualquier navegador web desktop y mobile.

Existen 2 formas de acceder a la pantalla de ingreso y registro de un usuario.

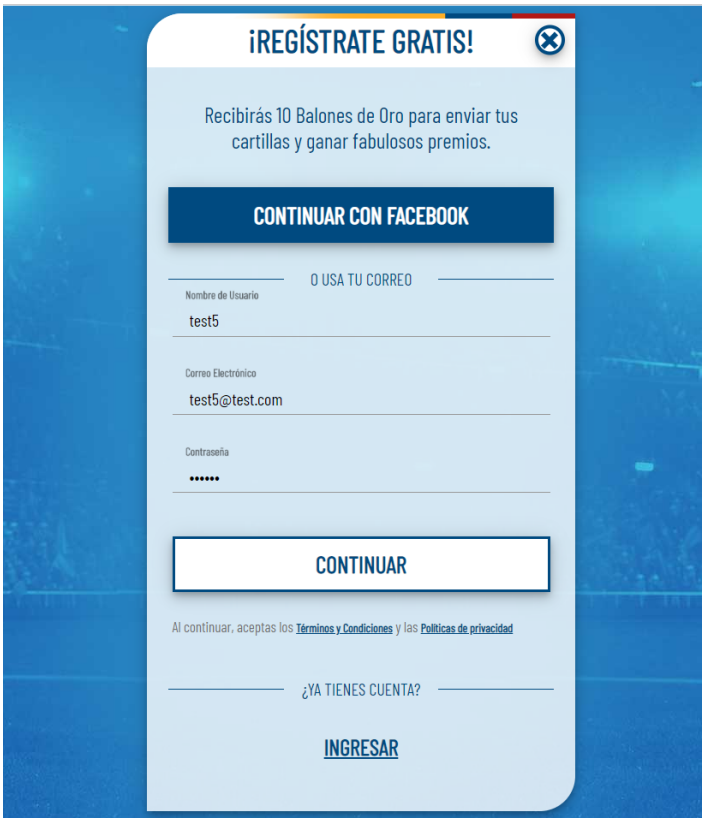
4) Enlace ingresar en la barra de menús.

En la parte superior derecha se encuentran las opciones de menú del sistema y entre ellas está *INGRESAR.*

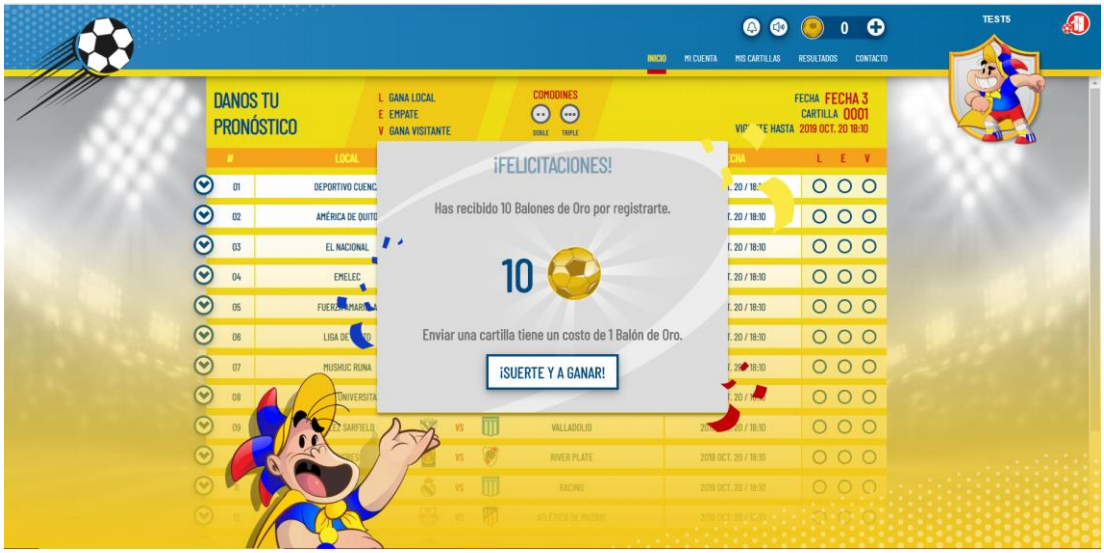
Al dar clic en el enlace se muestra la pantalla de ingreso y registro, si ya tiene una cuenta se procede a iniciar sesión con el correo y contraseña.



Para registrar un nuevo usuario se debe dar clic en el enlace CREAR CUENTA y llenar los datos requeridos (nombre de usuario, correo electrónico, contraseña).



Al crear un nuevo usuario, el sistema muestra un dialogo con 10 balones de oro como regalo.



Al dar clic en el botón se agregan los balones al total ubicado en la parte superior derecha.

5) Llenar un pronóstico y guardar los datos.

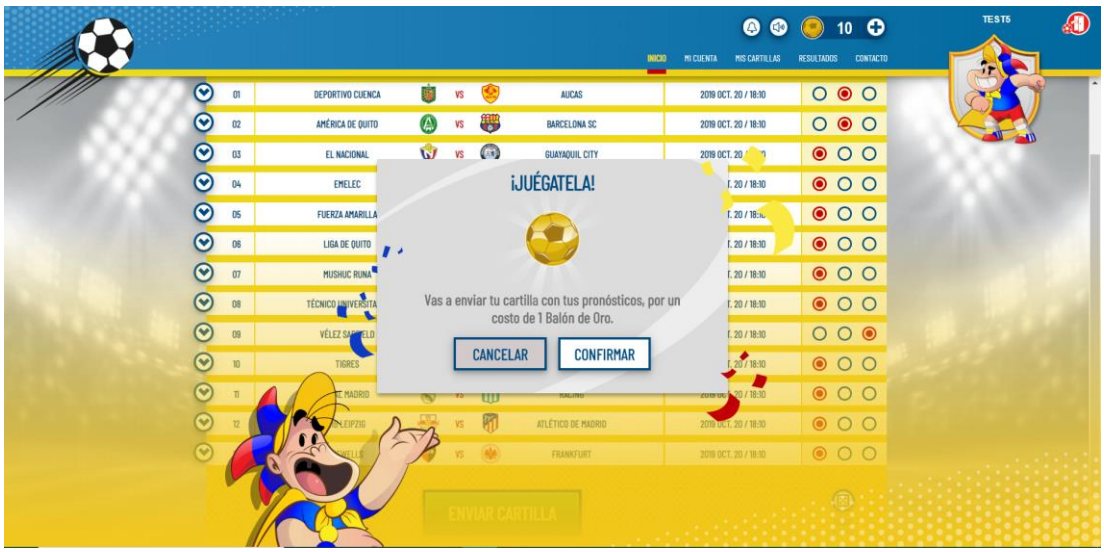
También se puede acceder a la pantalla de registro luego de llenar los datos de un nuevo pronóstico e intentar guardar los datos.

Al dar clic en *ENVIAR CARTILLA* se muestra la pantalla de ingreso y registro de usuarios y se procede a completar el proceso de registro descrito en la forma 1.

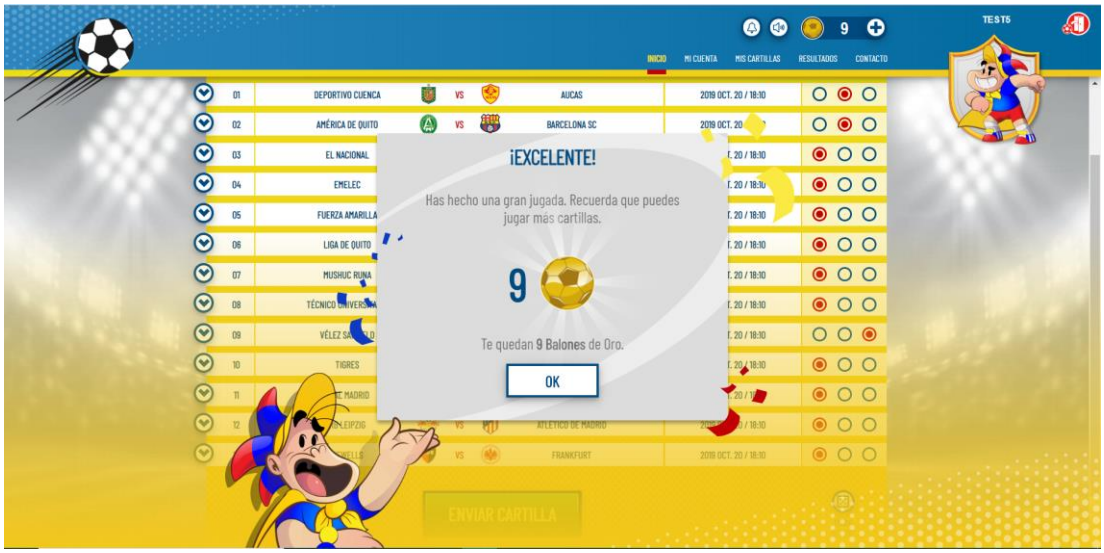
Enviar un pronóstico.

Para enviar una cartilla o pronóstico se debe llenar todos los campos y dar clic en ENVIAR CARTILLA.

Se muestra un dialogo informativo para confirmar el envío de la cartilla.

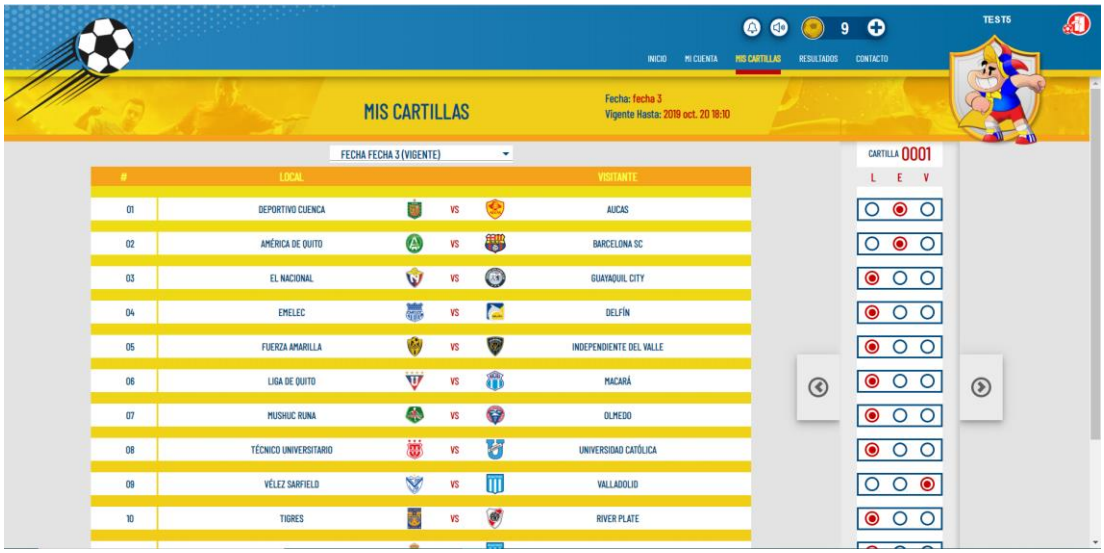


Al dar clic en CODNFIRMAR se ingresan los datos y se muestra un dialogo informativo con los balones de oro restantes. Cabe recordar que el costo de una cartilla es de 1 balón de oro.



Se puede revisar las cartillas enviadas desde el enlace MIS CARTILLAS en el menú de navegación.

Al dar clic en el enlace se muestra una pantalla con la cartilla actual en juego y en el lado derecho se muestran los pronósticos ingresados.

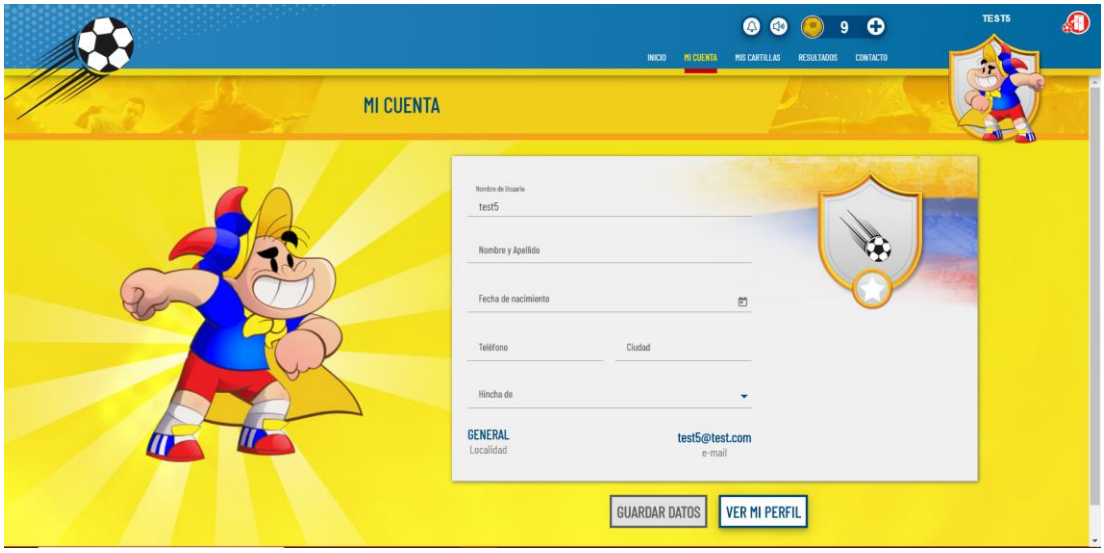


También se puede consultar los pronósticos de cartillas anteriores seleccionando la cartilla desde el combo box.

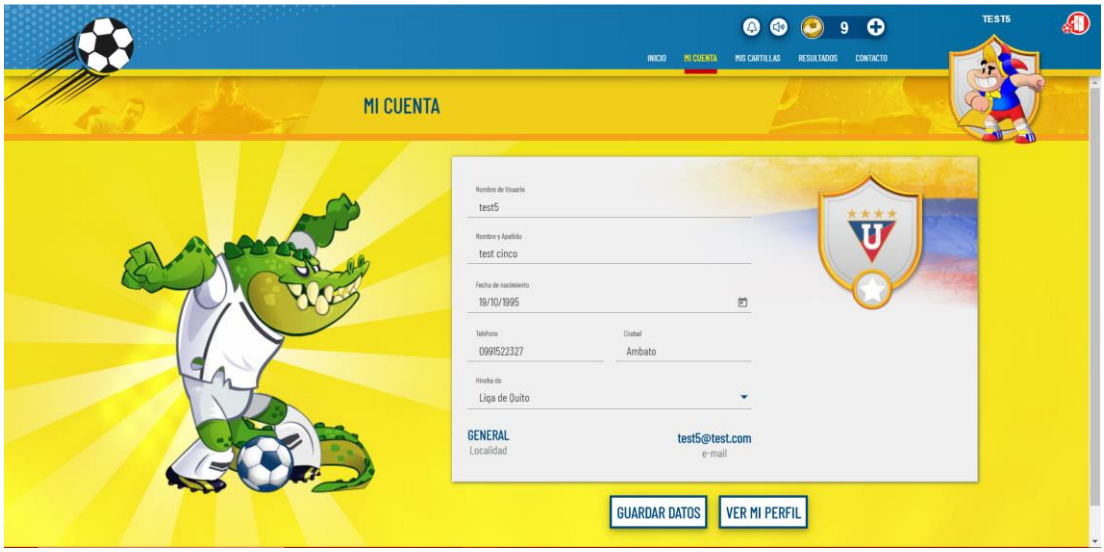
Actualizar datos de usuario.

Para actualizar los datos de una cuenta se debe acceder a la pantalla Mi Cuenta desde el enlace ubicado en el menú de navegación.

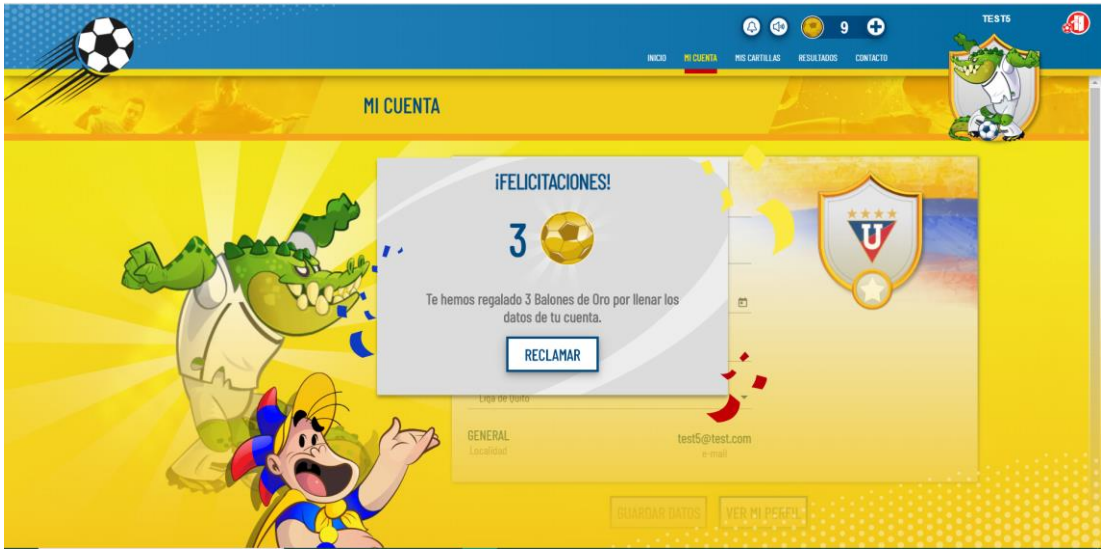
Se muestra la pantalla de cuenta privada donde se puede actualizar los datos personales. Si es una cuenta nueva los campos se encontrarán vacíos y la mascota por defecto.



Al llenar los datos personales se activa el botón de GUARDAR DATOS y al seleccionar ser hincha de un equipo, se actualiza la mascota en el lado izquierdo y en la tarjeta de usuario ubicado en la parte superior derecha.



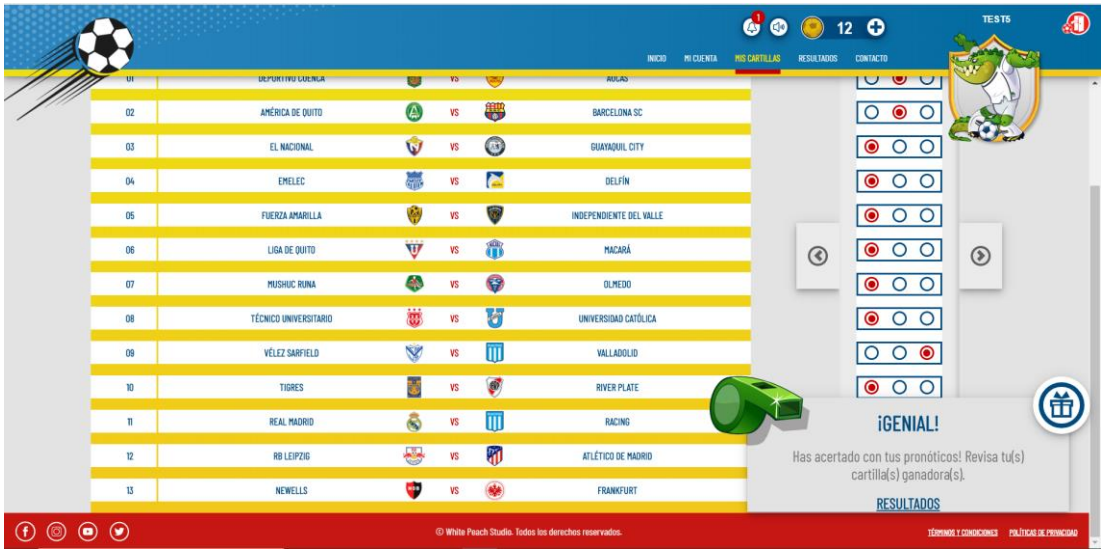
Al dar clic en GUARDAR DATOS se guarda la información. Si todos los datos personales están llenos, el sistema muestra un dialogo informativo para reclamar balones de oro como premio.



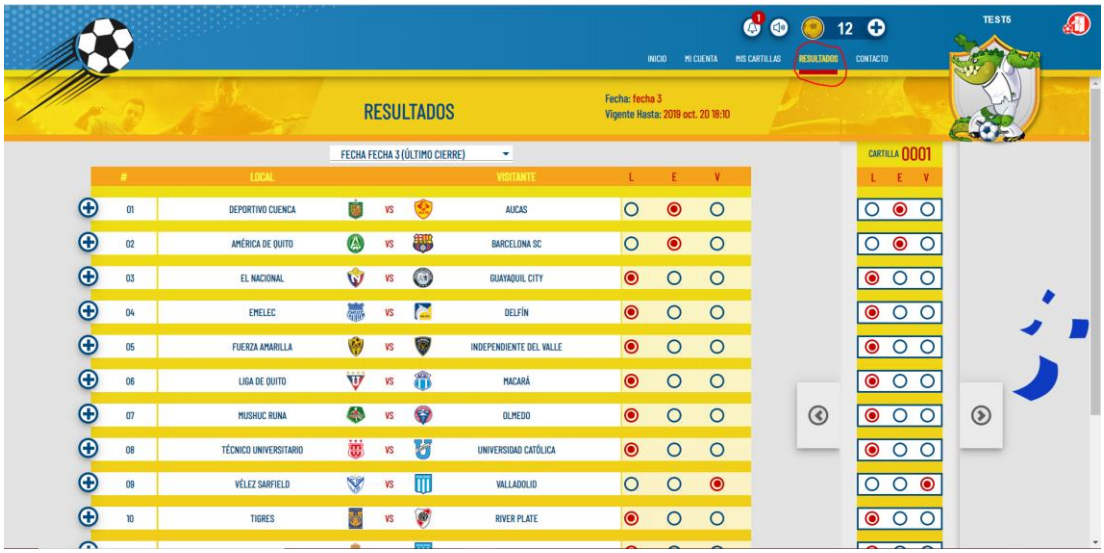
Reclamar balones de premio.

Cuando se ha procesado una cartilla, el sistema genera notificaciones para los usuarios que ha acertado los pronósticos.

El sistema muestra 2 notificaciones en el caso de tener premios por reclamar. La primera notificación es un mensaje informativo en la parte inferior derecha.



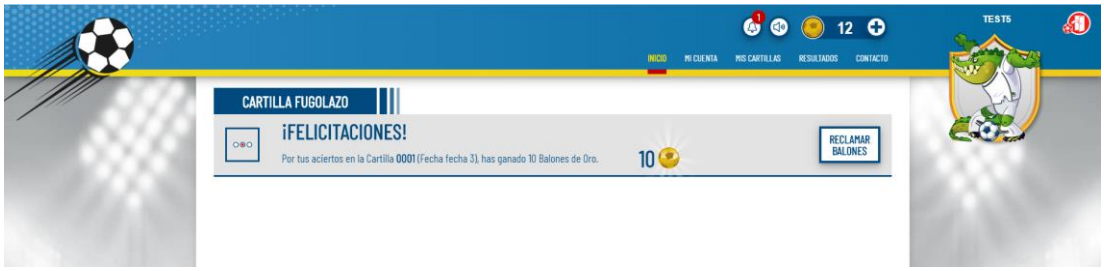
Al dar clic en RESULTADOS dentro de la notificación se redirecciona a la pantalla de resultados, también se puede llegar a la sección de resultados desde el enlace Resultados en el menú de navegación.



Se resalta de color amarillo los pronósticos que ganaron premios. También se puede consultar los resultados de las cartillas anteriores seleccionando la cartilla en el combo box.

La segunda notificación que se muestra es un contador en color rojo que se muestra en la parte superior derecha.

Al dar clic en la campana se muestra una pantalla con una lista de notificaciones, como los premios por cartillas u otras notificaciones de ese tipo.



En la lista de notificaciones se muestran botones para cerrar o reclamar premios, en este caso se debe reclamar balones de oro por aciertos en los pronósticos.

Al dar clic en RECLAMAR BALONES se actualiza el número total de balones en la parte superior derecha y se quita el contador en la campana de notificaciones.

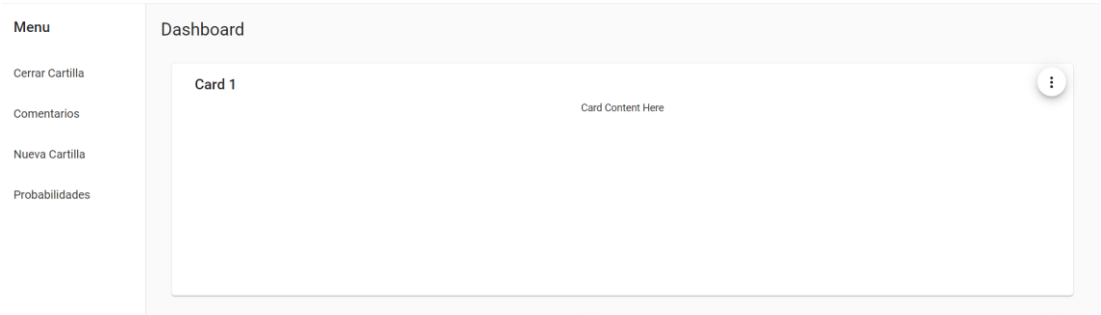
Anexo N.º 5. Manual de Usuario Administrador.

El presente manual tiene como objetivo mostrar el uso del sistema a los usuarios administradores y conocer sus funcionalidades.

Iniciar sesión.

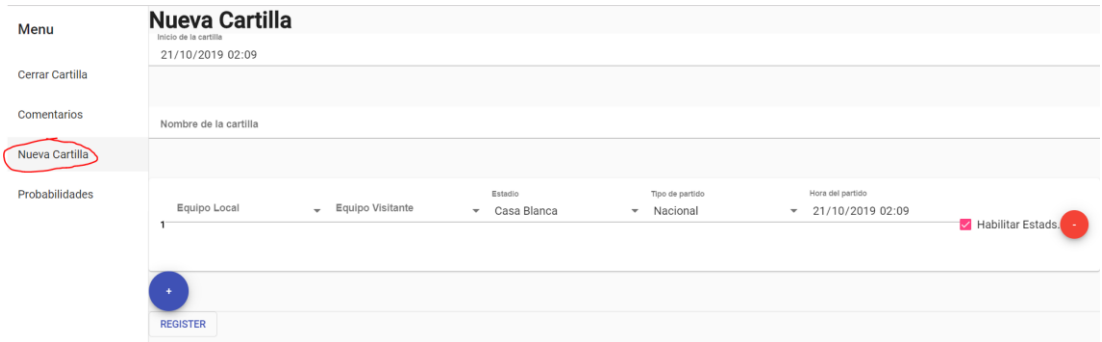
Para iniciar sesión como usuario administrador se debe ingresar las credenciales proporcionadas por el equipo de desarrollo. Por políticas de seguridad, la creación de usuarios administradores lo realiza el equipo de desarrollo.

Al iniciar sesión se muestra una pantalla minimalista con las opciones de administración en una barra de navegación al lado izquierdo.



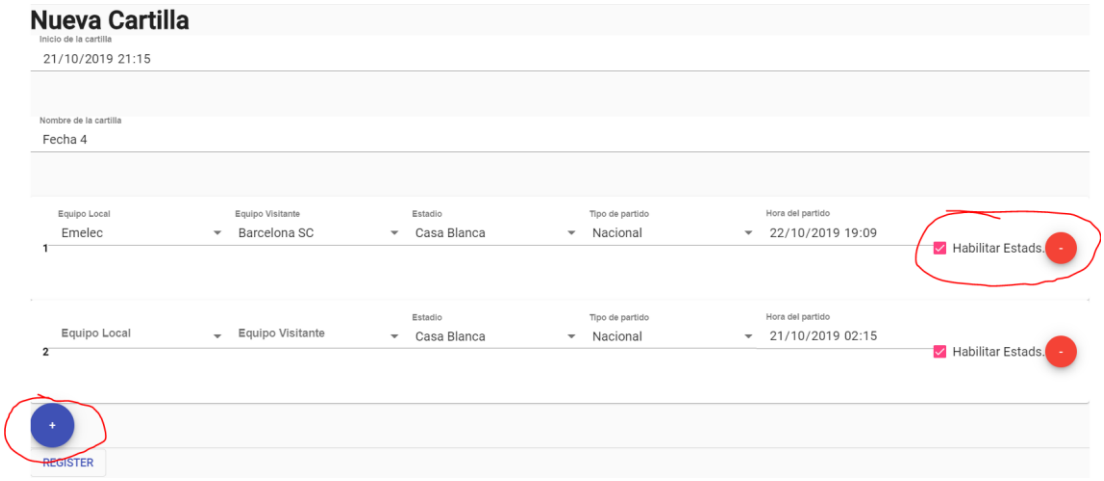
Crear nueva cartilla.

Para crear una nueva cartilla se debe dar clic en el enlace Nueva Cartilla en el menú de navegación.

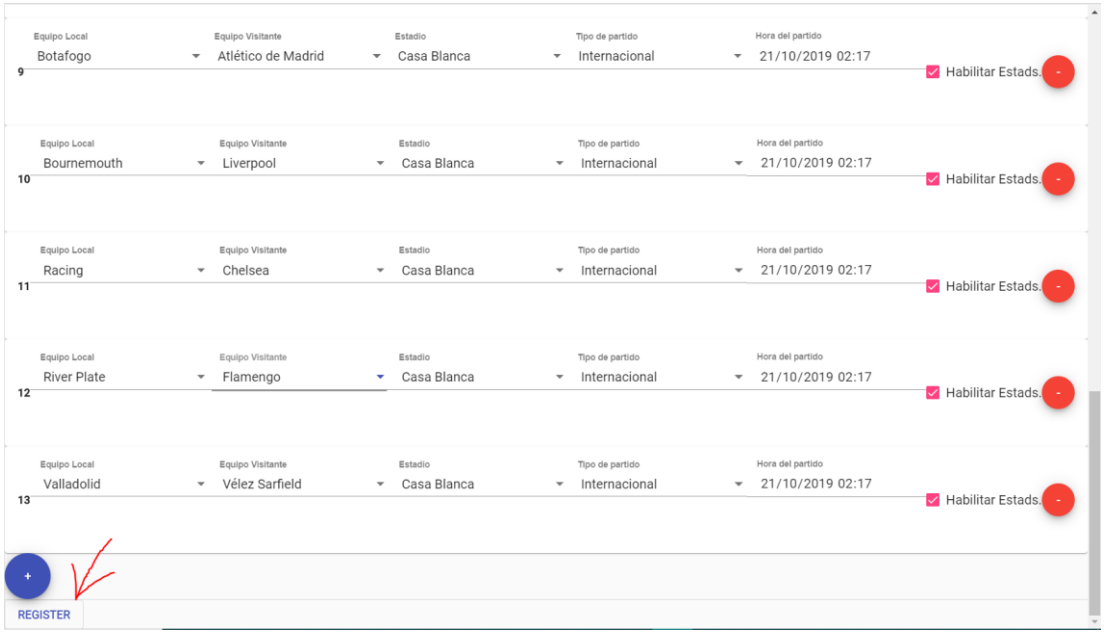


Para crear una nueva cartilla se debe especificar el nombre, fecha y hora que empieza su vigencia y los partidos que se jugarán. Para agregar un nuevo partido se debe dar

clic en el botón + de color azul, si se desea quitar un partido se debe dar clic en el botón – de color rojo. Si un partido permite mostrar estadísticas se debe dejar seleccionado la opción Habilitar Estads.



Al completar la cartilla con todos los partidos se procede guardar la información dando clic en el botón REGISTER ubicado al final del formulario. Al guardarse la cartilla correctamente se limpia el formulario, caso contrario se muestra un mensaje de error.



Al ingresar al sistema la nueva cartilla vigente se muestra en la pantalla de inicio.

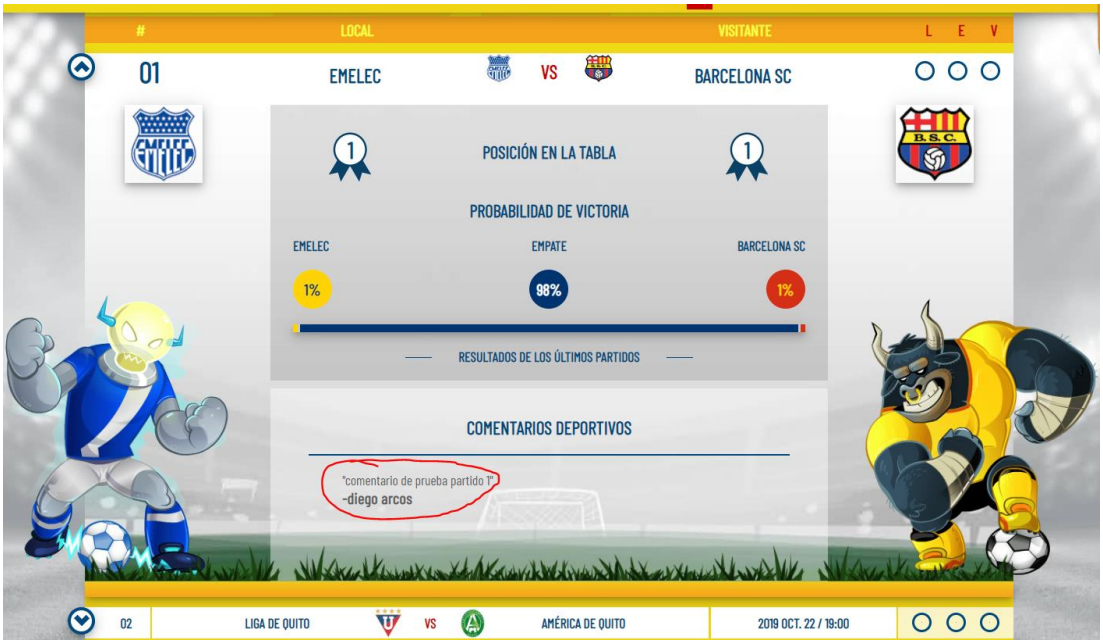
Insertar comentarios por partido.

Para insertar comentarios a los partidos de una cartilla se debe dar clic en el enlace Comentarios dentro del menú de navegación. En la sección comentarios se muestra la cartilla actual.

Al dar clic en el botón + de cada partido se despliega el formulario para ingresar comentarios y seleccionar el comentarista registrado en el sistema para que se muestra en la cartilla jugable.

Se puede agregar más comentarios dando clic en el botón + azul o eliminar comentarios dando clic en el botón – rojo. Para guardar la información se debe dar clic en el botón ENVIAR.

Al guardar la información se puede visualizar los cambios en el sistema desplegando las características del partido actualizado.



Insertar probabilidades por partido.

Para insertar las probabilidades de un partido se debe dar clic en el enlace Probabilidades dentro del menú de navegación. Al igual que en la sección de comentarios, se muestra la cartilla actual para actualizar los valores.

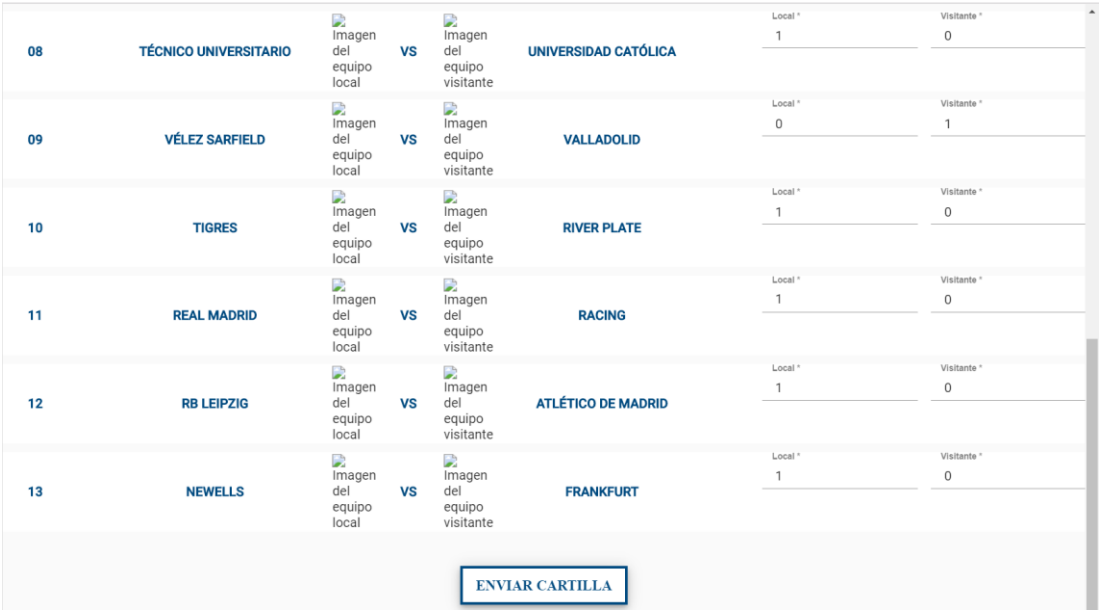
Al dar clic en el botón + se despliega la información a ser actualizada, donde se puede insertar la posición actual de los equipos, la probabilidad de victoria de cada uno e insertar los resultados de partidos anteriores.

Al guardar la información se puede visualizar los cambios en la cartilla actual en juego dentro del sistema.

Cerrar una cartilla.

Para insertar los resultados de las cartillas ya jugadas se lo debe hacer desde el enlace Cerrar Cartilla en el menú de navegación. En esta sección se muestra la última cartilla expirada con sus respectivos partidos.

Para cerrar una cartilla se debe insertar el marcado de cada partido en los campos de textos junto a cada partido de la cartilla.



Al guardar la información se muestra un mensaje de confirmación de la operación.

El proceso de validar los pronósticos de los usuarios se ejecuta en un cron cada 5 minutos y luego de que se ha procesado, se notifica a los usuarios si tiene premios por reclamar.