UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

Tema:

ARQUITECTURA MICRO-FRONTEND PARA OPTIMIZAR EL DESARROLLO DE APLICACIONES WEB TIPO SPA.

Trabajo de titulación modalidad Proyecto de Investigación, presentado previo a la obtención del título de Ingeniero en Tecnologías de la Información

ÁREA: Software.

LÍNEA DE INVESTIGACIÓN: Ingeniería de software

AUTOR: Pablo Alejandro Alvear Vaca

TUTOR: Ing. Dennis Vinicio Chicaiza Castillo, Mg

Ambato – Ecuador

agosto - 2023

APROBACIÓN DEL TUTOR

En calidad de tutor del trabajo de titulación con el tema: ARQUITECTURA MICRO-FRONTEND PARA OPTIMIZAR EL DESARROLLO DE APLICACIONES WEB TIPO SPA, desarrollado bajo la modalidad Proyecto de Investigación por el señor Pablo Alejandro Alvear Vaca, estudiante de la Carrera de Tecnologías de la Información, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, me permito indicar que el estudiante ha sido tutorado durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 17 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.3 del instructivo del reglamento referido.

Ambato, agosto 2023.

Firmado electrónicamente por:
DENNIS VINICIO
CHICAIZA CASTILLO

Ing. Dennis Vinicio Chicaiza Castillo, Mg

TUTOR

AUTORÍA

El presente trabajo de titulación titulado: ARQUITECTURA MICRO-FRONTEND PARA OPTIMIZAR EL DESARROLLO DE APLICACIONES WEB TIPO SPA es absolutamente original, auténtico y personal y ha observado los preceptos establecidos en la Disposición General Quinta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato. En tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato, agosto 2023

Pablo Alejandro Alvear Vaca

C.C. 1803542362

AUTOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato para que reproduzca total o parcialmente este trabajo de titulación dentro de las regulaciones legales e institucionales correspondientes. Además, cedo todos mis derechos de autor a favor de la institución con el propósito de su difusión pública, por lo tanto, autorizo su publicación en el repositorio virtual institucional como un documento disponible para la lectura y uso con fines académicos e investigativos de acuerdo con la Disposición General Cuarta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato.

Ambato, agosto 2023.

Pablo Alejandro Alvear Vaca

C.C 1803542362

AUTOR

APROBACIÓN DEL TRIBUNAL DE GRADO

En calidad de par calificador del informe final del trabajo de titulación presentado por el señor Pablo Alejandro Alvear Vaca, estudiante de la Carrera de Tecnologías de la Información, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, bajo la Modalidad Proyecto de Investigación, titulado ARQUITECTURA MICRO-FRONTEND PARA OPTIMIZAR EL DESARROLLO DE APLICACIONES WEB TIPO SPA , nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 19 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.4 del instructivo del reglamento referido. Para cuya constancia suscribimos, conjuntamente con la señora Presidente del Tribunal.

Ambato, agosto 2023.

Firmado electrónicamente por:
ELSA PILAR URRUTIA URRUTIA

Ing. Elsa Pilar Urrutia Urrutia, Mg.
PRESIDENTE DEL TRIBUNAL

Firmado electrónicamente por:
MARCO VINICIO GUACHIMBOZA VILLALVA

Ing. Marco Vinicio Guachimboza Villalva
PROFESOR CALIFICADOR

Firmado electrónicamente por:
SANTIAGO DAVID JARA MOYA

Ing. Santiago Jara Mora David
PROFESOR CALIFICADOR

DEDICATORIA

Quiero dedicar el presente trabajo de titulación a mi familia, principalmente a mis padres y a mi abuelita, que fueron los pilares fundamentales para que no me detenga en el camino y que con esfuerzo logre cumplir este sueño tanto en mi vida personal como académica. Gracias por hacerme la persona que soy.

A mi hermano Kevin que supo brindarme su apoyo cuando más lo necesitaba, gracias por brindarme la confianza y hacerme saber que si se podía.

AGRADECIMIENTO

Agradezco primeramente a mis padres por haber sido mi guía, durante todo el camino, y por no dejarme caer en el proceso. A mi abuelita que a pesar de su edad nunca dejo de velar por mí en los momentos difíciles.

A mis queridos amigos de la universidad que me acompañaron durante todos estos años y que finalmente vamos a cumplir este logro.

A mi profesor y tutor Dennis que, gracias a su conocimiento y experiencia, creo las bases adecuadas para que el camino no se me dificulte brindándome toda la ayuda necesaria.

ÍNDICE GENERAL DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE GRAFICOS

RESUMEN EJECUTIVO

En la actualidad, el mundo del desarrollo de aplicaciones web ha experimentado un cambio significativo y una de las tendencias más importantes es la adopción de la arquitectura microfrontend. Este modelo o enfoque en el diseño y desarrollo web permite descomponer un interfaz de usuario en componentes más pequeños e independientes, lo que ofrece varios beneficios en términos de rendimiento, mantenimiento y escalabilidad. El microfrontend abarca conceptos que buscan mejorar el modularidad y la reutilización de código de la parte frontend de la aplicación web. En lugar de manejar un solo código en único repositorio, la aplicación se puede dividir en módulos más pequeños y ser trabajados en equipos más pequeños.

El objetivo de este proyecto es implementar una arquitectura microfrontend para optimizar el desarrollo de aplicaciones web tipo SPA, lograr un desarrollo más eficiente y ágil, brindando a los usuarios una experiencia de alta calidad al interactuar con la aplicación web. Al aprovechar los beneficios del enfoque de microfrontend, las empresas pueden adaptarse rápidamente a las demandas cambiantes del mercado y ofrecer productos y servicios innovadores a sus usuarios finales. En consecuencia, se realizó una aplicación de prueba usando la arquitectura para evidenciar sus beneficios a través de métricas.

Para el desarrollo del proyecto se usó la metodología XP (Extreme Programming), enfocada a un desarrollo rápido, ya que fueron necesarias pruebas constantes de rendimiento para demostrar el objetivo principal. Para el desarrollo del Frontend se utilizaron las tecnologías de Angular y React, además se utilizó la librería de Module Federation para la aplicación de la arquitectura. Para el manejo del Backend se manejó un servicio basado en Nodejs utilizando una base de datos no relacional en un clúster de MongoDB, finalmente se aplicó pruebas de rendimiento con la herramienta devtools del navegador.

Palabras clave: Frontend, microfrontend, XP, angular, module federation.

ABSTRACT

Currently, the world of web application development has undergone a significant change, and one of the most important trends is the adoption of microfrontend architecture. This approach in web design and development allows for breaking down a user interface into smaller, independent components, offering several benefits in terms of performance, maintenance, and scalability. Microfrontend encompasses concepts that aim to improve modularity and code reuse in the frontend part of the web application. Instead of managing a single codebase in a unified repository, the application can be divided into smaller modules and worked on by smaller teams.

The objective of this project is to implement a microfrontend architecture to optimize the development of SPA-type web applications, achieving more efficient and agile development, and providing users with a high-quality experience when interacting with the web application. By leveraging the benefits of the microfrontend approach, companies can quickly adapt to the changing demands of the market and offer innovative products and services to their end users. Consequently, a test application was developed using the architecture to showcase its benefits through metrics.

For the project development, the XP (Extreme Programming) methodology was used, focusing on rapid development, as continuous performance testing was required to demonstrate the main objective. Angular and React technologies were used for frontend development, and the Module Federation library was utilized for implementing the architecture. A Node.js-based backend service was used, along with a non-relational database in a MongoDB cluster. Finally, Performance testing was finally applied using the browser's devtools tool.

Keywords: Frontend, microfrontend, XP, angular, module federation.

CAPÍTULO I.
MARCO TEÓRICO

1.1. Tema de investigación

ARQUITECTURA MICRO-FRONTEND PARA OPTIMIZAR EL DESARROLLO DE APLICACIONES WEB TIPO SPA.

1.1.1. Planteamiento del problema

Las páginas o aplicaciones web de tipo SPA (Single Page Application) son muy comunes hoy en día gracias a bibliotecas como Angular, React o Vue[1] Desarrollar una página web moderna se ha convertido en una tarea imprescindible para las empresas. Actualmente, se están buscando nuevas formas de implementar aplicaciones de forma más eficaz. Además, cada cierto tiempo surgen nuevos frameworks que brindan flexibilidad al momento de iniciar un nuevo proyecto. Gracias a todas las opciones y funcionalidades que ofrecen estas bibliotecas, surgen aplicaciones web más potentes y complejas, pero al mismo tiempo, surgen problemas al intentar realizar una escalabilidad [2].

Con el crecimiento de las páginas web en todo el mundo, modificar un solo módulo puede ser un gran problema para un equipo de desarrollo, ya que hay que implementar modificaciones asegurando siempre una escalabilidad a futuro. Por lo tanto, aplicar un nuevo enfoque se vuelve necesario. La aplicación de varias técnicas del lado del cliente y del servidor se vuelve muy pesada. La composición que se forma no está automatizada [3].

En Latinoamérica, la evolución en el marco de creación en el desarrollo web ha sido limitada. Sin embargo, en los últimos años, gracias al crecimiento de la digitalización, se ha destacado la labor de los desarrolladores Back-End y Front-End, lo que abre un camino para una buena proyección a futuro. En general, las páginas web actuales buscan crecer de manera acelerada. Pero, dado que la mayoría de ellas se basan en front-ends monolíticos, es decir, que la aplicación web está atada a un solo conjunto, pueden presentarse problemas en la escalabilidad y en el posible despliegue. Por lo tanto, existen otras formas de

estructurar una aplicación web para que funcionen como Webpack o GraphQL [4].

Existe ya una idea de desarrollo de componentes reutilizables en un marco de aplicaciones web, los proyectos contienen módulos sin afectar a otros de una forma independiente y por consiguiente pueden desplegarse, pero es importante establecer las practicas correctas que favorezcan a las características propias del navegador y se puedan realizar despliegues a diferentes servidores[5], Durante un tiempo las aplicaciones web han sido establecidas con un modelo o patrón Modelo-Vista- Controlador(MVC), dicho modelo establece separar la aplicación en 3 partes, un modelo donde se mapean los datos que se quieren manejar, una lógica y un controlador encargado de la comunicación, el siguiente paso es un proceso arquitectónico de microservicios en la parte Front-End del desarrollo, un proceso donde todo va por separado y de manera independiente en donde las vistas de las aplicaciones SPA devuelvan un resultado dividido en varias partes pero que va a ser procesado solo una vez.[6]

1.2. Antecedentes investigativos

Después de haber realizado un análisis de fuentes de investigación dentro de repositorios de Universidades Nacionales e Internacionales se han encontrado los siguientes temas similares al propuesto:

El proyecto realizado por Bonilla Adriano, el cual establece una aplicación WEB usando tecnologías SPA para la gestión de fichas médicas. Este proyecto concluyo que las aplicaciones web SPA y los servicios REST son tecnologías idóneas para el desarrollo de software que gestiona información sobre las historias clínicas de unidades médicas. Tecnologías que gracias a la arquitectura cliente servidor permiten obtener aplicaciones web más veloces al cargar la aplicación completa solo una vez, y evitar recargar con cada operación que el usuario realiza.[7]

La investigación realizada por Arango León, el cual habla sobre arquitecturas microfrontend, componentes y su construcción. Concluyo que los microfrontends son una realidad, y las arquitecturas en el frontend también son posibles e ir más allá gracias a la web moderna. Sin embargo, aún hay retos muy grandes que hacen

el desarrollo de estos más complejo, hay tecnologías que son realmente útiles y en un futuro cercano pueden cambiar completamente la forma en la que se desarrolla el frontend. Por otro lado, cuando hablamos de las tecnologías utilizadas es importante destacar a Single SPA, ya que es un framework que facilita mucho la implementación de una arquitectura en microfrontends, este framework, aunque aún le faltan cosas, es suficiente para satisfacer la necesidad básica de crearlos. AWS como nube es definitivamente la mejor opción, ya que permite desplegar frontends en S3 a un costo muy bajo y adicional a ello cuenta con otros servicios como Cloudfront y Route53 que facilitan el trabajo en gran medida. En cuanto a los frameworks de SPA como Vue, Angular y React funcionan muy bien.[3]

La investigación realizada por Garcia Mario la cual establece la implementación de una arquitectura microfrontend para optimizar el desarrollo y despliegue de una aplicación web. Concluyo que Se logró implementar la arquitectura micro-frontend en la aplicación web frontend del SIU mediante la tecnología webpack module federation, la cual funciona adecuadamente y una vez solucionados los errores reportados durante la marcha blanca, se concluye que esta implementación no afectó las funcionalidades que ya se encontraban desarrolladas. Además, Se dividió la aplicación web frontend del SIU en pequeñas aplicaciones, logrando disminuir los tiempos de compilación y despliegue de la aplicación web, lo cual supone una mejora importante en la reducción de tiempos en las fases de desarrollo y despliegue de la aplicación web del SIU.[8]

La Tesis realizada por linkola Lasse, en donde estableció una arquitectura modular microfrontend en una aplicación existente. Comprobó que una arquitectura modular en aplicaciones web es una buena opción para proyectos de larga duración, en los que la rotación de desarrolladores es alta. En la primera parte de la tesis, se realizó una introducción a las aplicaciones web de una sola página. En la segunda parte, se analizó una aplicación existente de una sola página, bastante tradicional, se elaboró un plan para migrar esta aplicación concreta a una arquitectura modular. Como base principal de la aplicación se eligió el marco de trabajo de una sola capa, una vez finalizado el proyecto, se determinó que la arquitectura de micro frontend es una opción válida a la hora de elegir el tipo de arquitectura de un proyecto de aplicación web.[9]

1.3. Fundamentación Teórica

Diseño Web:

El diseño web es una parte fundamental que suele confundirse comúnmente con el desarrollo web, pero la realidad es cada una tiene su lugar en el proceso de creación de un proyecto web, el encargado de la parte visual de la web era el programador o desarrollador, pero es importante que el encargado entienda el panorama completa de lo que va a ser la web, es decir conocer las capacitaciones y limitaciones.

La experiencia de usuario es lo más importante, el diseño web define las necesidades del proyecto y la arquitectura, además contiene toda la estructura estableciendo las plantillas y composiciones a través de WireFrames.[10]

Frameworks FrontEnd:

Un Framework es un estándar que sigue varias reglas que buscan facilitar la vida al momento del desarrollo, es decir brinda flexibilidad al momento de diseñar la lógica y las interfaces ya que existe una jerarquía.[11]

Arquitectura WEB:

Es la relación de los elementos de toda la estructura web, es decir es la planificación y diseño de los componentes, los desarrolladores se basan en esto para crear un plan de trabajo dependiendo del contexto y lo que busque la página:

- P2P - Peer to Peer.
- Modelo Cliente-Servidor.
- Modelo con servidor de aplicaciones
- Modelo con servidor de aplicaciones externo
- Modelo con varios servidores de aplicaciones [12]

Arquitectura MicroFront-End:

Es un tipo de arquitectura que divide el FrontEnd en porciones pequeñas independientes que trabajan de manera mutua, esta arquitectura está basado en los microservicios usados en el BackEnd. El Objetivo es que los equipos trabajen en áreas específicas y logren avances en menor tiempo, además de que el mantenimiento puede ser más optimo.[13]

Ingeniera de Software aplicada a la web:

La ingeniería de software aplicada a la web es la aplicación de los principios, métodos y técnicas de la ingeniería de software al desarrollo de aplicaciones web. Se trata de una disciplina que combina la informática, la ingeniería y la gestión de proyectos para crear productos web de alta calidad, seguros y eficientes.

Los ingenieros de software web utilizan un conjunto de herramientas y metodologías para desarrollar aplicaciones web. Estas herramientas incluyen lenguajes de programación, frameworks, bases de datos y servidores. Las metodologías, por su parte, proporcionan un marco para el desarrollo de software web de manera sistemática y eficiente.

Las principales tareas de los ingenieros de software web incluyen:

- **Análisis:** Comprender los requisitos del cliente y traducirlos a un plan de desarrollo.
- **Diseño:** Crear un diseño de la aplicación web que cumpla con los requisitos del cliente.
- **Desarrollo:** Escribir el código fuente de la aplicación web.
- **Pruebas:** Probar la aplicación web para garantizar su calidad.
- **Implementación:** Desplegar la aplicación web en un entorno de producción.

- **Mantenimiento:** Actualizar y mejorar la aplicación web de forma continua.

Los ingenieros de software web trabajan en una variedad de industrias, incluyendo tecnología, comercio electrónico, educación y servicios financieros. Pueden trabajar en empresas de todos los tamaños, desde pequeños startups hasta grandes corporaciones.

Los beneficios de la ingeniería de software aplicada a la web:

- **Mejor calidad**: La ingeniería de software ayuda a garantizar que las aplicaciones web sean de alta calidad, seguras y eficientes.
- **Mayor productividad:** La ingeniería de software puede ayudar a mejorar la productividad de los desarrolladores de software web.
- **Reducción de costes**: La ingeniería de software puede ayudar a reducir los costes de desarrollo y mantenimiento de aplicaciones web.
- **Mejora de la colaboración:** La ingeniería de software puede ayudar a mejorar la colaboración entre los desarrolladores de software web.[14]

Desarrollo de aplicaciones web:

Es la creación de herramientas para la interacción con el internet, que nos permitan obtener distintos tipos de información, las aplicaciones web son accesibles para la mayoría de público y el mantenimiento de estas es fácil ya que no hay problemas con las versiones.[15]

Tipos de Aplicaciones web:

Las aplicaciones web se diferencian según la necesidad de estas, es decir la gestión de contenidos que tendrán de acuerdo con su complejidad, por ejemplo:

- **Estáticas**: pensadas para no incluir a futuro nuevas actualizaciones.

- **Aplicación web Dinámica:** son más complejas gracias a que incluyen base de datos.
- **Portal web aplicación:** se basan en tener varios apartados donde pueda interactuar el usuario.
- **Web Animada:** Usan tecnologías flash, página que basa principalmente en mostrar animaciones.[16]

Aplicaciones web tipo SPA:

Es un tipo de aplicación que ejecuta todo el proyecto en una sola página, es decir que van ligados a un solo archivo index, el contenido de la aplicación se muestra en menor tiempo ya que carga el contenido de forma dinámica, esta al ejecutarse en el navegador es decir que para su frontEnd usa js se pueden usar librerías o frameworks como Angular o React.[17]

1.4. Objetivos

1.4.1. Objetivo general

Implementar una Arquitectura Micro-FrontEnd para optimizar el desarrollo de aplicaciones web tipo SPA.

1.4.2. Objetivos específicos

- Analizar las mejores tecnologías para el desarrollo web tipo SPA.
- Determinar cuáles son las técnicas para aplicar una Arquitectura Micro-FrontEnd.
- Desarrollar una aplicación web de tipo SPA que utilice Arquitectura MicroFrontEnd para optimizar su rendimiento y escalabilidad.

CAPÍTULO II

METODOLOGIA

2.1. Materiales

Para obtener una perspectiva más nítida del problema y el tema abordados, el proyecto de investigación empleó fichas bibliográficas para sintetizar tesis y documentos relacionados.

Ficha de Contenido

TEMA	
TESIS	
PROPOSITO	
IDEAS CENTRALES	
CONCEPTOS CLAVES	
CONCLUSIONES	
APORTE	

Tabla 1 Formato Ficha de Contenido

Elaborado por: el Investigador

2.2. Métodos

2.2.1 Modalidad de la investigación

La investigación se basará en un enfoque bibliográfico-documental, ya que se sustenta en la recopilación de información proveniente de diversas fuentes, tales como revistas, artículos, tesis y libros, entre otras. Esta información será utilizada para desarrollar una fundamentación teórica que abordará una variedad de temas, como el Diseño WEB, las Arquitecturas WEB, los Tipos de aplicaciones WEB, entre otros.

2.2.2 Población y muestra

Debido a que el proyecto es de tipo bibliográfico-documental no se va a utilizar población ni muestra.

2.2.3 Recolección de información

Las siguientes fichas bibliográficas hablan sobre la aplicación de una arquitectura microfrontend utilizando distintas tecnologías SPA como pueden ser Angular, Vue o React, mediante el uso de una correcta construcción de la arquitectura basándose en librerías como webpack, single-SPA, etc haciendo que, al dividir la interfaz de usuario en módulos independientes, se pueda desarrollar cada módulo como una SPA y luego integrarlos en una única aplicación web. Esto permite a los equipos trabajar de manera más eficiente y a los usuarios disfrutar de una experiencia de usuario más rápida y fluida.

Ficha de Contenido 1.

TEMA	IMPLEMENTACIÓN DE UNA ARQUITECTURA MICRO-FRONTEND PARA OPTIMIZAR EL DESARROLLO Y DESPLIEGUE DE LA APLICACIÓN WEB DEL SISTEMA DE INFORMACIÓN UNIVERSITARIA DE LA SUNEDU [8]
TESIS	El autor propone el uso de una arquitectura basada en microfrontends usando Angular para escalar el sistema.
PROPOSITO	reducir los tiempos de compilación y despliegue.

IDEAS CENTRALES	• La posibilidad de escalar un sistema. • Unión de diferentes tecnologías • Reducción de procesos
CONCEPTOS CLAVES	Contiene conceptos sobre node js, npm, angular, webpack, aplicaciones SPA, arquitectura de microservicios, arquitectura microfrontend.
CONCLUSIONES	Se dividió la aplicación web frontend del SIU en pequeñas aplicaciones, logrando disminuir los tiempos de compilación y despliegue de la aplicación web. Se crearon sitios web en el servidor que permiten alojar cada aplicación microfrontend del SIU, lo cual optimizó el procedimiento de despliegue de la aplicación web del SIU, logrando un despliegue independiente de app, esto sin que afecte las apps que ya están desplegadas. Al estar alojadas en un sitio web distinto, el despliegue de una app, no supone un riesgo para las apps que ya se encuentran en funcionamiento.
APORTE	• División de una aplicación web en partes iguales. • Escalamiento general de la aplicación. • Migración de la aplicación a nuevas versiones.

Tabla 2 Ficha de Contenido 1

Elaborado por: El Investigador

Ficha de Contenido 2.

TEMA	ARQUITECTURAS DE MICRO FRONTENDS, COMPONENTES WEB Y LIBRERÍAS DE COMPONENTES [3]
TESIS	El autor propone Elaborar una guía que permita construir arquitecturas basadas en microfrontends, componentes y librerías web, destacando los aspectos más relevantes en cada uno de estos.
PROPOSITO	Establecer las bases sólidas para implementar una arquitectura microfrontend.
IDEAS CENTRALES	• Hacer uso de los criterios, para definir arquitecturas adecuadas, patrones y antipatrones • prácticas para el diseño e implementación. • Implementar una prueba de concepto ejecutando los pasos y prácticas definidas en la guía.
CONCEPTOS CLAVES	Contiene conceptos sobre microfrontend, cómo funciona, formas de implementación.
CONCLUSIONES	Al hablar de las tecnologías utilizadas es importante destacar a Single SPA, ya que es un framework que facilita mucho la implementación de una arquitectura en microfrontends, este framework, aunque aún le faltan cosas, es suficiente para satisfacer la necesidad básica de crearlos. AWS como nube es definitivamente la mejor opción, ya que permite desplegar frontends en S3 a un costo muy bajo y

	adicional a ello cuenta con otros servicios como Cloudfront y Route53 que facilitan el trabajo en gran medida. En cuanto a los frameworks de SPA como Vue, Angular y React funcionan muy bien
APORTE	• Guía de construcción de un microfrontend • Uso correcto de las tecnologías tipo SPA.

Tabla 3 Ficha de Contenido 2

Elaborado por: El Investigador

Ficha de Contenido 3.

TEMA	MODELO DE ARQUITECTURA PARA FRONT-END (BASADO EN MICROFRONTENDS) APLICADO AL PRODUCTO DIGITAL DE FUERZA ESPECIALIZADA DE VIVIENDA DEL BANCO DE BOGOTÁ [18]
TESIS	El autor propone implementar una arquitectura de microfrontend que cumpla con las normas ISO-25010 e ISO-9126
PROPOSITO	Evitar los deterioros de las tecnologías en los productos digitales del Banco Bogotá.
IDEAS CENTRALES	• Evitar la complejidad en los proyectos frontend • Obtener información de la aplicación de la arquitectura microfrontend midiéndola con métricas que permitirán analizar correctamente la aplicación. • Mejorar la mantenibilidad de los proyectos frontend

CONCEPTOS CLAVES	Este proyecto contiene conceptos sobre microfrontend, beneficios de su uso, aplicaciones SPA, Angular, React, Vue, Componentes WEB
CONCLUSIONES	El principal objetivo de este proyecto era implementar una arquitectura frontend basada en la idea de los microservicios, que al lograr aplicar dicho modelo se pudo comprobar que es posible elaborar un proyecto en donde su composición se basa en pequeñas partes y que estas puedan ser mantenidas por diferentes equipos de desarrollo que a su vez se encargaran de crear nuevos microfrontends. Este proyecto busco afirmar que es posible reducir la complejidad del desarrollo ya que se usan mucho menos componentes en su estructura base. El proyecto se realizó una prueba de concepto del modelo microfrontend, ya que se implementó algunas funcionalidades para poder verificar el impacto.
APORTE	• Conceptos fundamentales para implementación de la arquitectura o modelo microfrontend • Interfaces estandarizadas para el correcto impacto del modelo

Tabla 4 Ficha de Contenido 3

Elaborado por: El Investigador

Ficha de Contenido 4.

TEMA	Micro frontend: Microservice implementation on Web Development [19]
TESIS	El autor propone examinar la aplicabilidad de la arquitectura microfrontend para el desarrollo frontend.
PROPOSITO	Documentar el uso de la arquitectura con ventajas y desventajas en contra de la arquitectura monolítica en una aplicación SPA en React.
IDEAS CENTRALES	• Migración de un proyecto monolítico a un proyecto con arquitectura Microfrontend. • Establecer las funcionalidades de un proyecto, para posteriormente dividirlas. • Adecuarse a los requisitos originales de la aplicación o proyecto de software
CONCEPTOS CLAVES	Esta investigación contiene conceptos importantes sobre Microfrontend, SPA, React, Api Rest, Axios
CONCLUSIONES	Esta investigación analiza completamente el desarrollo frontend, para demostrar que es posible la migración monolítica a microfrontend, donde se utilizó el framework single-spa para poder completar con satisfacción la migración.
APORTE	• Uso del framework single-spa para migración entre arquitectura monolítica y microfrontend

	• Uso de librerías y Frameworks frontend como: Angular,React y Vue para el desarrollo de la arquitectura microfrontend

Tabla 5 Ficha de Contenido 4

Elaborado por: El Investigador

Ficha de Contenido 5.

TEMA	Micro frontend architecture for cross framework reusability in practice[20]
TESIS	El autor propone en este estudio la posibilidad de aplicar una arquitectura microfrontend para crear funcionalidades reutilizables.
PROPOSITO	Reducir la carga de trabajo en el desarrollo frontend mediante el estudio y análisis de las técnicas para implementar una arquitectura microfrontend.
IDEAS CENTRALES	• Construcción de un Microfrontend Independiente. • Comunicación entre los distintos microfrontends.
CONCEPTOS CLAVES	Esta investigación contiene conceptos claves sobre Iframes, Javascript, Componentes Web, Integración del lado del servidor.
CONCLUSIONES	El estudio demuestra que una aplicación monolítica puede llegar a ser grande que se vuelve inútil, este hecho significo buscar algo similar a los microservicios en el frontend, esto está haciendo que las

	empresas empiecen a estructurar equipos internos y externos mediante una división vertical. Se encontraron desventajas en los microfrontends, sin embargo, se pueden controlar si la empresa se organiza para el manejo de la arquitectura.
APORTE	• Anidación de una arquitectura Microfrontend mediante el uso de librerías Angular y React. • Implementación de un microfrontend desacoplado.

Tabla 6 Ficha de Contenido 5

Elaborado por: El Investigador

Ficha de Contenido 6.

TEMA	Micro-frontends: application of microservices to web front ends[6]
TESIS	El artículo explora el concepto microfrontend en el desarrollo web, como la evolución lógica de la arquitectura del lado frontal de las aplicaciones web.
PROPOSITO	Crear aplicaciones fáciles de distribuir y entregar al usuario una experiencia cercana a las aplicaciones de escritorio.
IDEAS CENTRALES	• Desarrollo de funcionalidades basándose en una aplicación compleja • Simplificación del desarrollo en comparación a una arquitectura monolitica

CONCEPTOS CLAVES	El articulo contiene conceptos importantes sobre desarrollo web, frontend, microfrontends, arquitecturas web modernas, aplicaciones web estáticas y aplicaciones web SPA.
CONCLUSIONES	El concepto de arquitectura microfrontend es el enfoque de microservicios convertido al frontend, y es la nueva alternativa para el desarrollo de aplicaciones modulares frontend. El nuevo estilo de arquitectura microfrontend es una respuesta a la creciente complejidad de las aplicaciones de hoy en día.
APORTE	• Guía de implementación de una arquitectura microfrontend sobre una aplicación compleja

Tabla 7 Ficha de Contenido 6

Elaborador por: El Investigador

2.2.4 Procesamiento y análisis de datos

De acuerdo con la información recolectada de temas similares al planteado se puede decir que:

- No existe mucha información sobre la arquitectura microfrontend.
- Las librerías o frameworks frontend más usadas para la construcción de una arquitectura microfrontend son: Angular, React y Vue.
- Una arquitectura microfrontend puede ser implementada usando una o varias librerías o frameworks.
- Hay que evaluar todas las funcionalidades de una aplicación, antes de considerar una posible migración a una nueva arquitectura web.

- Es posible trabajar en pequeñas partes independientes al mismo tiempo por medio de varios equipos de desarrollo
- Existen tecnologías como single-spa y webpack que permiten la correcta construcción de una arquitectura microfrontend.
- Las tecnologías web SPA son muy populares y existe una extensa documentación que pueden ayudar en caso de que se presente un problema en el desarrollo.

CAPÍTULO III
RESULTADOS Y DISCUSIÓN

3.1 Análisis y discusión

3.1.1 Tecnologías SPA para construir una arquitectura microfrontend

Existen varias tecnologías SPA, que se pueden usar para el desarrollo frontend, de las cuales claramente las más populares son React, Angular y Vue como se pudo evidenciar en la información recolectada, entre las tecnologías SPA que se pueden usar para construir una arquitectura microfrontend se encuentran las siguientes:

3.1.1.1 React

React es una biblioteca de código abierto desarrollada por Facebook, diseñada para facilitar la creación de interfaces de usuario interactivas y reactivas. Su enfoque se basa en el uso de componentes reutilizables que encapsulan tanto la lógica como la presentación de la interfaz.

Una de las características destacadas de React es su utilización del DOM virtual (Virtual DOM). El DOM virtual es una representación en memoria de la estructura del DOM real, lo que permite a React realizar actualizaciones eficientes y rápidas en la interfaz de usuario. En lugar de actualizar todo el DOM directamente cada vez que se produce un cambio, React compara el DOM virtual anterior con el nuevo, determinando qué elementos necesitan ser actualizados. Esto resulta en un rendimiento optimizado y una mayor eficiencia en la manipulación del DOM.

React utiliza una sintaxis llamada JSX, que es una extensión de JavaScript que combina elementos HTML y código JavaScript en un solo archivo. Esta

sintaxis facilita la construcción de componentes al permitir que la lógica y la presentación coexistan en el mismo lugar.

Con JSX, puedes escribir código que se asemeje mucho al HTML tradicional, pero también puedes utilizar expresiones de JavaScript dentro de los elementos HTML. Esto significa que puedes insertar variables, ejecutar lógica condicional y realizar otras operaciones en línea dentro de tus componentes.

La sintaxis JSX ayuda a que el código sea más legible y declarativo, ya que puedes describir directamente cómo debería lucir la interfaz de usuario en lugar de tener que manipular manualmente el DOM. Esto simplifica el proceso de construcción de componentes y mejora la comprensión del código, lo que a su vez facilita el mantenimiento y la colaboración en proyectos de React.

React ofrece un flujo de datos unidireccional a través de las propiedades (props) y el estado (state). Las propiedades se utilizan para pasar datos de un componente padre a un componente hijo, mientras que el estado se utiliza para almacenar y controlar datos dentro de un componente. Cuando el estado o las propiedades cambian, React actualiza automáticamente la interfaz de usuario para reflejar estos cambios.

Otras características notables de React incluyen:

React Router: Una biblioteca adicional que facilita la creación de enrutamiento en aplicaciones React.

React Native: Una extensión de React que permite construir aplicaciones móviles nativas para iOS y Android utilizando JavaScript.

Redux: Una biblioteca de gestión de estado que se utiliza comúnmente con React para aplicaciones más grandes y complejas.

Amplia comunidad y ecosistema: React cuenta con una gran comunidad de desarrolladores, lo que significa que hay una amplia gama de recursos,

bibliotecas y componentes disponibles para su uso. Además, existen numerosos tutoriales, documentación y ejemplos en línea para facilitar el aprendizaje y la adopción de React.

React es una biblioteca de JavaScript ampliamente utilizada para construir interfaces de usuario interactivas y reactivas. Con su enfoque basado en componentes, rendimiento optimizado y amplio ecosistema, React se ha convertido en una opción popular para el desarrollo de aplicaciones web modernas y escalables.[21]

3.1.1.2 Angular

Angular es un framework de desarrollo de aplicaciones web de código abierto desarrollado por Google. Se utiliza para construir aplicaciones de una sola página (SPA) complejas y de gran escala. Angular adopta un enfoque completo para el desarrollo, proporcionando una estructura sólida y extensible que abarca todos los aspectos del desarrollo web, desde la creación de componentes hasta la manipulación de datos y la gestión del estado de la aplicación.

Una de las características clave de Angular es su enfoque en la arquitectura MVC (Modelo-Vista-Controlador) y el uso de TypeScript, un superset de JavaScript que añade tipado estático y otras características avanzadas al lenguaje. Esto proporciona un sistema de desarrollo más robusto y seguro, permitiendo la detección temprana de errores y una mejor organización del código.

Angular se basa en el concepto de componentes reutilizables, que encapsulan la lógica y la presentación de la interfaz de usuario. Estos componentes se combinan para formar la aplicación, y Angular se encarga de gestionar la comunicación y el flujo de datos entre ellos. Además, Angular proporciona un potente sistema de enrutamiento para crear una navegación fluida dentro de la aplicación y un sistema de inyección de dependencias que facilita la gestión de las dependencias entre los componentes.

Otras características importantes de Angular incluyen:

Enlace de datos bidireccional: Angular proporciona un enlace de datos bidireccional automático entre los componentes y las vistas, lo que facilita la actualización automática de los datos en ambas direcciones.

Pruebas unitarias integradas: Angular cuenta con un conjunto completo de herramientas y utilidades para realizar pruebas unitarias y de integración de manera eficiente.

Soporte completo para la accesibilidad: Angular ofrece características y directivas que facilitan la creación de aplicaciones web accesibles para personas con discapacidades.

CLI (Command Line Interface): Angular CLI es una herramienta de línea de comandos que proporciona una manera sencilla de crear, construir y probar aplicaciones Angular.

Amplia comunidad y ecosistema: Angular cuenta con una gran comunidad de desarrolladores, lo que significa que hay una amplia cantidad de recursos, bibliotecas y módulos disponibles para extender las capacidades de Angular.

Angular es un framework completo para el desarrollo de aplicaciones web SPA. Con su enfoque en la arquitectura MVC, componentes reutilizables y características como el enlace de datos bidireccional, Angular ofrece un entorno robusto y escalable para construir aplicaciones web modernas y complejas.[22]

3.1.1.3 Vue:

Vue.js es un framework progresivo de JavaScript para la construcción de interfaces de usuario interactivas. Es muy popular debido a su facilidad de uso, flexibilidad y rendimiento eficiente. Vue adopta un enfoque incremental, lo

que significa que puedes integrarlo fácilmente en proyectos existentes o utilizarlo para construir aplicaciones completas desde cero.

Una de las características distintivas de Vue es su capacidad para construir aplicaciones de una sola página (SPA). Utiliza una sintaxis declarativa y basada en componentes para definir la estructura de la interfaz de usuario. Los componentes de Vue encapsulan la lógica y la presentación de la interfaz de usuario, lo que permite la reutilización y modularidad del código.

Vue utiliza un sistema de reactividad para mantener los componentes sincronizados con los cambios en los datos subyacentes. Esto significa que cualquier cambio en los datos se reflejará automáticamente en la interfaz de usuario, lo que simplifica la manipulación y actualización de los elementos visuales.

Además, Vue ofrece una amplia gama de características y complementos que pueden ser agregados según las necesidades del proyecto. Estas características incluyen enrutamiento, gestión de estado con Vuex, renderizado del lado del servidor (SSR), pruebas unitarias y muchas más. Vue también proporciona herramientas como Vue CLI, que facilita la creación y configuración de proyectos de Vue de manera rápida y sencilla.

Otras características notables de Vue incluyen:

Directivas: Vue incluye un conjunto de directivas predefinidas, como v-if, v-for y v-bind, que permiten manipular y controlar fácilmente el DOM en función de los datos.

Animaciones y transiciones: Vue proporciona un sistema integrado para crear animaciones y transiciones suaves en los elementos de la interfaz de usuario.

Comunidad activa y ecosistema: Vue tiene una comunidad activa y creciente, lo que significa que hay una gran cantidad de recursos, bibliotecas y

complementos disponibles. También cuenta con una documentación completa y una amplia variedad de tutoriales y ejemplos en línea.

Vue.js es un framework JavaScript flexible y fácil de aprender que permite construir interfaces de usuario interactivas y eficientes. Con su enfoque basado en componentes, sistema de reactividad y amplia gama de características, Vue es una excelente opción para desarrolladores que desean construir aplicaciones SPA modernas y escalables.[22]

3.1.1.4 Svelte:

Svelte es un framework de desarrollo web de código abierto que se enfoca en la compilación anticipada (ahead-of-time compilation). A diferencia de otros frameworks como React, Vue o Angular, Svelte no se ejecuta en tiempo de ejecución en el navegador, sino que transforma el código en JavaScript optimizado durante la fase de compilación. Esto resulta en aplicaciones web más rápidas y eficientes, ya que no es necesario cargar una biblioteca de framework completa en el navegador.

Una de las características principales de Svelte es su enfoque en la simplicidad y la facilidad de uso. Svelte utiliza una sintaxis declarativa y descriptiva para definir la estructura de la interfaz de usuario. Los componentes en Svelte son autónomos y autocontenidos, lo que significa que el código del componente encapsula tanto la lógica como la presentación de la interfaz de usuario. Además, Svelte ofrece una reactividad intrínseca, lo que permite que los cambios en los datos se reflejen automáticamente en la interfaz de usuario sin necesidad de bibliotecas adicionales.

Svelte también ofrece un sistema de propiedades y eventos intuitivo para la comunicación entre componentes. Permite una gestión clara y eficiente de los datos y eventos, lo que facilita el desarrollo y el mantenimiento de aplicaciones complejas.

Otras características notables de Svelte incluyen:

Animaciones y transiciones: Svelte proporciona una sintaxis simple y potente para crear animaciones y transiciones fluidas en la interfaz de usuario.

Reutilización de código: Svelte permite reutilizar fácilmente componentes y fragmentos de código en diferentes partes de la aplicación.

Optimización del tamaño del archivo: Debido a su enfoque de compilación anticipada, Svelte genera código JavaScript optimizado y minimizado, lo que resulta en archivos más pequeños y tiempos de carga más rápidos.

Herramientas de desarrollo: Svelte cuenta con una variedad de herramientas de desarrollo, como el Svelte REPL (Read-Eval-Print Loop) y la extensión Svelte for VS Code, que facilitan la escritura, depuración y prueba de código.

Aunque Svelte es relativamente nuevo en comparación con otros frameworks, ha ganado popularidad debido a su enfoque único y eficiente en la creación de aplicaciones web rápidas y sencillas. La comunidad de Svelte está creciendo rápidamente y ofrece recursos, bibliotecas y ejemplos adicionales para ayudar a los desarrolladores a aprovechar al máximo esta tecnología.

Svelte es un framework de desarrollo web que se enfoca en la compilación anticipada y la generación de código optimizado. Con su enfoque en la simplicidad, la reactividad y la eficiencia, Svelte es una opción atractiva para desarrolladores que buscan crear aplicaciones web rápidas y eficientes sin agregar una gran sobrecarga de bibliotecas al código final.[23]

3.1.1.5 Preact

Preact es una biblioteca de JavaScript de código abierto que se inspira en React y ofrece una alternativa más ligera y rápida. Su objetivo principal es proporcionar una API compatible con React, pero con un tamaño de biblioteca

mucho más reducido. Esto permite una carga más rápida de la aplicación y un mejor rendimiento, especialmente en dispositivos con recursos limitados.

Preact se enfoca en brindar una experiencia de desarrollo similar a React, lo que facilita la transición para los desarrolladores familiarizados con esta biblioteca. Proporciona componentes reutilizables, un enfoque basado en props y state, y un sistema de renderizado virtual eficiente. Los componentes en Preact funcionan de manera similar a los de React, permitiendo una estructura organizada y modular para la construcción de interfaces de usuario.

Aunque Preact es más liviano que React, no sacrifica muchas de las características y funcionalidades esenciales. Ofrece soporte para el enrutamiento, gestión del estado y manipulación de eventos, lo que permite crear aplicaciones SPA completas y escalables.

Preact cuenta con una comunidad en crecimiento, lo que significa que hay cada vez más recursos y bibliotecas disponibles para su uso. Además, Preact se integra fácilmente con otras bibliotecas y frameworks, lo que le brinda flexibilidad y opciones para crear la arquitectura de su aplicación según sus necesidades específicas.[24]

3.1.2 Analisis comparativo de tecnologías SPA para construir un microfrontend

A continuación, se muestra una tabla comparativa entre frameworks o librerías que usan tecnología SPA con la finalidad de analizar las mejores para la implementación de una arquitectura de microfrontend.

Framework	**Ventajas**	**Desventajas**	**Compatible con Microfrontends**
Angular	Amplio ecosistema y	Mayor curva de aprendizaje.	Sí

	documentación. Soporte para aplicaciones de gran escala. Integración con herramientas y librerías.	Tamaño del bundle más grande en comparación. Complejidad en proyectos pequeños.	
Vue	Curva de aprendizaje moderada. Tamaño del bundle pequeño. Flexibilidad para diferentes proyectos. Buena integración con herramientas y librerías.	Comunidad algo más pequeña. Menor soporte para proyectos de gran escala.	Sí
React	Amplia popularidad y comunidad activa. Gran rendimiento y eficiencia. Enfoque en reutilización de componentes. Amplio ecosistema y librerías disponibles.	Mayor curva de aprendizaje para principiantes. Responsabilidad en elección de herramientas adicionales. Algunos conceptos pueden ser confusos.	Sí

Svelte	Excelente rendimiento y tamaño de bundle pequeño. Curva de aprendizaje moderada. Enfoque en simplicidad y código conciso. Mejor integración con el navegador y optimización en tiempo de compilación.	Comunidad y ecosistema en crecimiento. Menos recursos y librerías en comparación. Requiere configuraciones adicionales para características avanzadas.	Sí
Preact	Excelente rendimiento y tamaño de bundle mínimo. Curva de aprendizaje baja. Compatibilidad total con API de React. Buen soporte de comunidad y documentación.	Ecosistema y comunidad más pequeños en comparación. Menos herramientas y librerías disponibles. Algunas características pueden requerir configuraciones adicionales.	Sí

Tabla 8 Tecnologías SPA

Elaborador por: El Investigador

Analisis

React y Angular son dos de los frameworks de JavaScript más populares para el desarrollo de aplicaciones web. Ambos ofrecen una serie de ventajas y desventajas que pueden hacer que sean la elección adecuada para su proyecto.

Angular es un framework completo que proporciona todo lo que necesita para desarrollar aplicaciones web complejas. Incluye soporte para componentes, directivas, servicios, inyección de dependencias y más. Angular es una buena opción para proyectos que requieren una estructura sólida y bien definida.

React es una biblioteca de componentes ligera que es ideal para aplicaciones web de gran tamaño. Es eficiente en el uso de los recursos y puede escalar para soportar un alto tráfico de usuarios. React es una buena opción para proyectos que requieren una interfaz de usuario altamente interactiva y receptiva.

3.1.3 Técnicas para la construcción de una arquitectura web microfrontend

La arquitectura de Micro-FrontEnd se centra en la descomposición de una aplicación web en piezas más pequeñas, autónomas y escalables, conocidas como microfrontends. A continuación, se describen algunas técnicas para aplicar una arquitectura Micro-FrontEnd:

Single-SPA: Single-SPA (Single-Page Application) es un marco de trabajo que facilita la construcción de aplicaciones de una sola página compuestas por varios microfrontends, cada uno desarrollado con su propia tecnología (por ejemplo, React, Angular, Vue.js).

Brinda un enrutador compartido que se encarga de cargar y mostrar los microfrontends en función de la URL actual. Cada microfrontend tiene su propio ciclo de vida y puede ser desarrollado y desplegado de forma independiente. Además, ofrece funcionalidades como la compartición de

estado entre microfrontends y la carga dinámica de los mismos en tiempo de ejecución.

Module Federation: Module Federation es una funcionalidad de Webpack, un paquete de herramientas utilizado para el empaquetado de aplicaciones web, que permite la integración de varios microfrontends en una sola aplicación.
Module Federation permite compartir código y dependencias entre los microfrontends, evitando la duplicación innecesaria y optimizando la carga de la aplicación. Cada microfrontend puede exportar y consumir módulos de otros microfrontends, lo que facilita la comunicación y colaboración entre ellos.

Web Components ofrece un estándar para crear componentes reutilizables, Single-SPA proporciona un marco de trabajo para construir aplicaciones de una sola página con varios microfrontends y Module Federation permite la integración y compartición de código entre los microfrontends utilizando Webpack. Estas tecnologías son herramientas poderosas para construir arquitecturas de Micro-FrontEnd flexibles y escalables.

3.1.4 Analisis de las técnicas para la construcción de una arquitectura web microfrontend

Métricas	Web Components	Single-SPA	Module Federation (Webpack)
Reutilización de código	Alta	Alta	Alta
Comunidad y soporte	Amplia comunidad y soporte	Amplia comunidad y soporte	Amplia comunidad y soporte
Encapsulación	Alta	Alta	Alta

Interoperabilidad	Compatible con diferentes tecnologías (HTML, CSS, JS)	Compatible con diferentes tecnologías (React, Angular, Vue)	Compatible con diferentes tecnologías (Webpack, JS Modules)
Compartición de estado	Depende de la implementación	Admite compartición de estado	Admite compartición de estado
Carga dinámica	Depende de la implementación	Admite carga dinámica de microfrontends	Admite carga dinámica de microfrontends
Escalabilidad	Alta	Alta	Alta
Rendimiento	Depende de la implementación	Buen rendimiento	Buen rendimiento

Tabla 8 Analisis comparativo técnicas construcción microfrontend

Elaborador por: El Investigador

Analisis

Module Federation es una gran herramienta debido a los beneficios que tiene en cuanto se refiere escalabilidad, reutilización de código, y modularidad. Module Federation permite crear proyectos web que se componen de varios módulos que se pueden unir con varios frameworks o librerías Frontend, esto hace que las aplicaciones sean más fáciles de manejar ya que el despliegue es independiente ya que la ejecución tiene su propio espacio. Module Federation permite un rápido desarrollo y una arquitectura compleja.

Utilizar una arquitectura de microfrontends híbrida con Angular y React en un solo proyecto brinda flexibilidad al aprovechar las fortalezas de cada framework, permitiendo la reutilización de componentes y optimizando la experiencia del equipo de desarrollo. Sin embargo, es importante considerar la complejidad adicional y los recursos necesarios para coordinar múltiples frameworks en el proyecto.

3.1.5 Analisis comparativo de las metodologías compatibles con el desarrollo de una aplicación web.

Característica	Scrum	Kanban	XP (Extreme Programming)
Posibilidad de uso	Recomendado	Recomendado	Recomendado
Ventajas	Entrega incremental, flexibilidad, colaboración y adaptación continua	Visualización del flujo de trabajo, gestión de prioridades y adaptabilidad	Prácticas ágiles y enfoque en la calidad del código
Desventajas	Requiere un compromiso sólido del equipo	Menor estructura y enfoque en plazos fijos	Requiere una comunicación y colaboración intensiva
Fases del proyecto	Planificación, sprint, revisión y retrospectiva	No hay fases definidas, flujo continuo de trabajo	No hay fases definidas, enfoque en prácticas ágiles
Tamaño del equipo	Idealmente de 5 a 9 personas (incluyendo el Product Owner y Scrum Master)	Variable, puede adaptarse a equipos de diferentes tamaños	Idealmente de 2 a 12 personas

Tabla 9 Analisis comparativo metodologías

Elaborador por: El Investigador

Analisis

La metodología XP (Extreme Programming) es una opción sólida para un proyecto de microfrontend debido a varias razones. En primer lugar, XP se centra en la colaboración y la comunicación cercana entre los miembros del equipo, lo que es esencial en un entorno de desarrollo distribuido y en constante evolución de microfrontends. XP fomenta la interacción directa entre desarrolladores, diseñadores y usuarios finales, lo que facilita la toma de decisiones rápidas y la adaptación a los cambios. Además, XP promueve prácticas como la integración continua, las pruebas automatizadas y el diseño simple, lo que es fundamental en un proyecto de microfrontends donde se requiere una entrega rápida y frecuente de componentes independientes. La retroalimentación constante y el enfoque en la calidad del código garantizan la estabilidad y el mantenimiento eficiente de los microfrontends a lo largo del tiempo. En resumen, la metodología XP ofrece un marco sólido para la gestión ágil de un proyecto de microfrontend, fomentando la colaboración, la adaptabilidad y la entrega de valor de manera continua.

3.2 Desarrollo de la propuesta

El siguiente proyecto se desarrolló aplicando la metodología XP, que consta de las siguientes fases: planificación, diseño, codificación y pruebas.

3.2.1 Fase I: Planificación

3.2.1.1 Levantamiento de la información.

Los resultados obtenidos mediante la herramienta de recolección utilizada determinaron las funcionalidades exactas con las que cuenta una arquitectura microfrontend. Además, se determinaron las mejores técnicas para una correcta implementación de esta arquitectura, ya que su finalidad es demostrar

la optimización y escalabilidad de una aplicación web SPA. Por lo tanto, es necesario desarrollar dos aplicaciones web SPA manejando arquitecturas distintas: una que demuestre la implementación de una arquitectura microfrontend con el uso de Module Federation y otra aplicación web de tipo SPA con una arquitectura monolítica.

Aquí hay algunos detalles adicionales sobre cada aplicación:

La aplicación SPA microfrontend utilizará Module Federation para combinar diferentes módulos en una sola aplicación. Esto permitirá que la aplicación se divida en componentes más pequeños y fáciles de administrar, lo que hará que sea más escalable y fácil de mantener.
La aplicación SPA con arquitectura monolítica será una aplicación más tradicional en la que todos los componentes se encuentran en un solo archivo.

El tipo de aplicación que se va a usar para demostrar la arquitectura es una aplicación de tipo e-commerce.

3.2.1.2 Arquitectura de la aplicación

Se desarrollaron dos aplicaciones web para la propuesta, una de las cuales se creó utilizando el framework Angular, que emplea la tecnología de Aplicación de Página Única (SPA), pero siguiendo una arquitectura monolítica. Para el lado del backend de esta aplicación, se utilizó un servicio basado en Node.js, junto con una base de datos no relacional MongoDB.

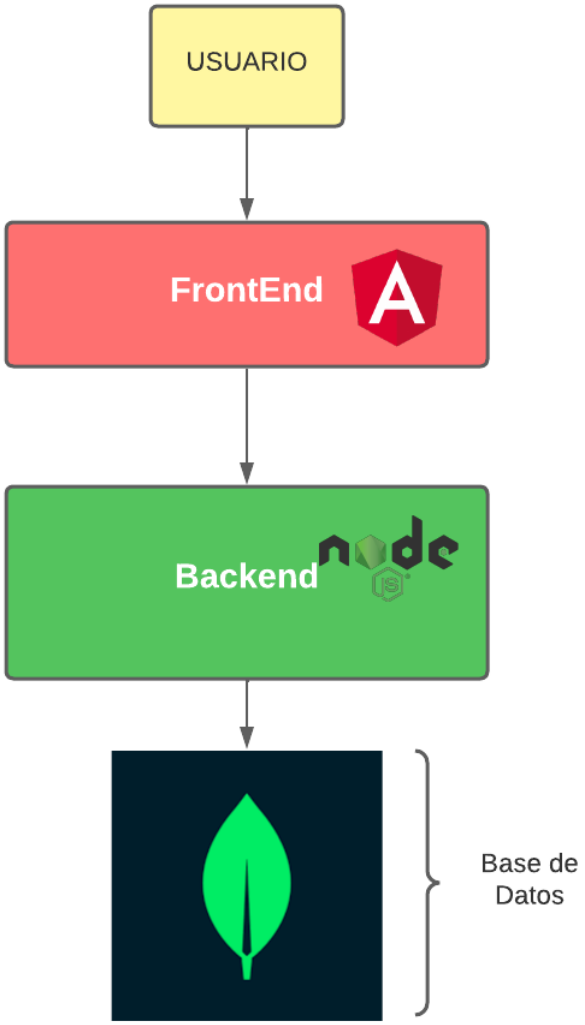


Figura 1 Arquitectura aplicación monolítica

Elaborador por: El Investigador

En caso de la segunda aplicación se usó las tecnologías de Angular y React que son compatibles con SPA, pero en este caso empleando una arquitectura microfrontend con el uso de Module Federation.

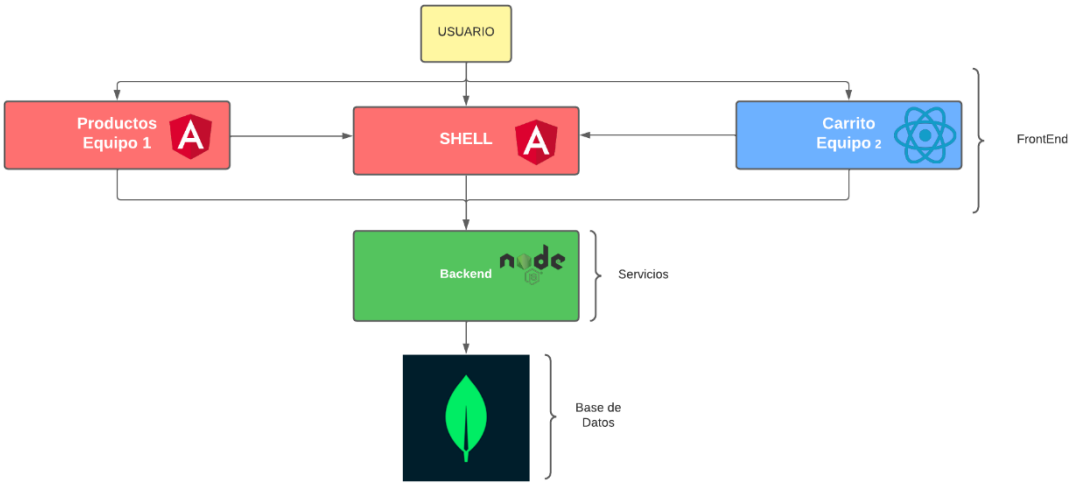


Figura 2 Arquitectura aplicación microfrontend

Elaborador por: El Investigador

3.2.1.3 Requerimientos de software

Para el desarrollo se utilizaron las siguientes herramientas:

Node.js: Es necesario tener Node.js instalado en el sistema, para crear un entorno de ejecución basado en JavaScript.

Gestor de paquetes npm: Junto con Node.js, npm (Node Package Manager) se instala automáticamente. Es utilizado para instalar y gestionar las dependencias del proyecto.

Module Federation: es una herramienta de Webpack 5, que permite la integración de varios módulos.

Tailwind: Framework de diseño que consta con una amplia cantidad de clases para el desarrollo de aplicaciones web.

Angular CLI: Angular CLI (Command Line Interface) es una herramienta de línea de comandos que se utiliza para crear, desarrollar y gestionar proyectos Angular.

MongoDB: Sistema de gestión de base de datos no relacional.

3.2.1.4 Roles del proyecto

Los roles del proyecto se presentan en la siguiente tabla:

Nombre	Roles	Función
Pablo Alvear	Programador	Responsable de tomar todas las decisiones técnicas del proyecto
Ing. Dennis Chicaiza	Tracker	Hacer el seguimiento de acuerdo con la planificación
	Tester	Revisar pruebas y avances

Tabla 10 Roles en el proyecto

Elaborador por: El Investigador

3.2.1.5 Historias de usuario para la aplicación microfrontend.

En el caso de un proyecto microfrontend, es necesario definir la utilización de los módulos y componentes que se usaron para exponer la aplicación.

Para la elaboración de las historias de usuario se realizó el siguiente modelo:

Historia de Usuario	
Numero:	
Nombre de la historia	
Prioridad:	
Programador Responsable:	

Microfrontend:	
Descripción:	

Tabla 11 Modelo Historias de Usuario aplicación microfrontend

Elaborador por: El Investigador

Descripción de los elementos de la historia de usuario.

3.1 Numero: Identificador numérico que contiene cada historia.

3.2 Nombre de la historia: Titulo que describe la historia.

3.3 Prioridad: Valor de la importancia de la historia, en este caso para la funcionalidad de la aplicación.

3.4 Programador Responsable: Persona encargada de cumplir el desarrollo de la historia de usuario.

3.5 Microfrontend: Componente o modulo que se expone en la aplicación web

Historia de Usuario	
Numero: 1	
Nombre de la historia:	Analisis de la Arquitectura de la aplicación web tipo SPA con microfrontend.
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Microfrontend:	Ninguno
Descripción: Identificar tecnologías a usar	

Tabla 12 Historia de usuario para las tecnologías de la aplicación

Elaborador por: El Investigador

Historia de Usuario	
Numero: 2	
Nombre de la historia:	Generar estructura de la aplicación
Prioridad:	Alta

Programador Responsable:	Pablo Alvear
Microfrontend:	Ninguno
Descripción: Se crea toda la estructura de la aplicación, generando un espacio de trabajo en angular, que en este caso se va a encargar de manejar todos los módulos que se expongan de la aplicación angular y react.	

Tabla 13 Historia de usuario estructura de la aplicación

Elaborador por: El Investigador

Historia de Usuario	
Numero: 3	
Nombre de la historia:	Configuración de module federation en el espacio de trabajo.
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Microfrontend:	Ninguno
Descripción: Se implementa todo el esquema de webpack con las dependencias de module federation en todo el espacio de trabajo, estableciendo el Shell y los remotes (propiedades que es establecen para el uso de microfrontend mediante los módulos que exponen)	

Tabla 14 Historia de usuario implementación module federation

Elaborador por: El Investigador

Historia de Usuario	
Numero: 4	
Nombre de la historia:	Configuración proyecto shell
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Microfrontend:	Shell
Descripción: Configuración del proyecto principal de la aplicación, que se va a encargar de contener y exponer las demás aplicaciones	

Tabla 15 Historia de usuario configuración proyecto Shell

Elaborador por: El Investigador

Historia de Usuario	
Numero: 5	
Nombre de la historia:	Configuración proyecto productos
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Microfrontend:	productos
Descripción: Configuración del proyecto productos, para exponerlo hacia la aplicación shell.	

Tabla 16 Historia de usuario configuración proyecto productos

Elaborador por: El Investigador

Historia de Usuario	
Numero: 6	
Nombre de la historia:	Desarrollo de la interfaz de productos.
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Microfrontend:	productos
Descripción: Diseñar la interfaz de productos aplicando las clases de tailwind	

Tabla 17 Historia de usuario desarrollo de la interfaz de productos

Elaborador por: El Investigador

Historia de Usuario	
Numero: 7	
Nombre de la historia:	Buscador de productos
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Microfrontend:	productos

Descripción: Buscador que filtre los distintos productos

Tabla 18 Historia de Usuario de buscador de productos

Elaborador por: El Investigador

Historia de Usuario	
Numero: 8	
Nombre de la historia:	Paginación de productos
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Microfrontend:	productos
Descripción: Paginador de productos para desplazarse entre todo el catalogo	

Tabla 19 Historia de usuario de paginación de productos

Elaborador por: El Investigador

Historia de Usuario	
Numero: 9	
Nombre de la historia:	Generar aplicación en react
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Microfrontend:	ninguno
Descripción: Generar aplicación en react, en este caso para el carrito y el navbar.	

Tabla 20 Historia de Usuario de aplicación en react

Elaborador por: El Investigador

Historia de Usuario	
Numero: 10	
Nombre de la historia:	Configuración proyecto react
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Microfrontend:	carrito

Descripción: Configurar proyecto en react para exponerlo y se pueda consumir en el shell.

Tabla 21 Historia de usuario de configuración proyecto de react

Elaborador por: El Investigador

Historia de Usuario	
Numero: 11	
Nombre de la historia:	Desarrollo interfaz navbar
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Microfrontend:	carrito
Descripción: Diseñar la interfaz navbar, que contendrá el módulo de la lógica del carrito.	

Tabla 22 Historia de usuario de desarrollo de la interfaz nabvar

Elaborador por: El Investigador

Historia de Usuario	
Numero: 12	
Nombre de la historia:	Modulo carrito de compras
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Microfrontend:	ninguno
Descripción: Desarrollar módulo de carrito que se encarga de gestionar los carritos seleccionados para realizar las compras.	

Tabla 23 Historia de usuario de módulo de carrito de compras

Elaborador por: El Investigador

3.2.1.6 Estimación historias de usuario aplicación microfrontend.

Numero de historia	Historia de usuario	Dias	Horas

1	Analisis de la Arquitectura de la aplicación web tipo SPA con microfrontend.	4	20
2	Generar estructura de la aplicación	2	10
3	Configuración de module federation en el espacio de trabajo.	3	15
4	Configuración proyecto Shell.	1	5
5	Configuración proyecto productos	1	5
6	Desarrollo de la interfaz de productos.	2	10
7	Buscador de productos	1	5
8	Paginación de productos	1	5
9	Generar aplicación en react	1	5
10	Configuración proyecto react	2	10
11	Desarrollo interfaz navbar	2	10
12	Modulo carrito de compras	3	15
Estimado:		23	115

Tabla 24 Estimación historia de usuario aplicación microfrontend

Elaborador por: El Investigador

3.2.1.7 Plan de entrega aplicación microfrontend

Numero de historia	Historia de usuario	Dias	Horas	Numero de Iteración
1	Analisis de la Arquitectura de la aplicación web tipo SPA con microfrontend.	4	20	1
2	Generar estructura de la aplicación	2	10	1
3	Configuración de module federation en el espacio de trabajo.	3	15	1
4	Configuración proyecto Shell.	1	5	1
5	Configuración proyecto productos	1	5	1
6	Desarrollo de la interfaz de productos.	2	10	2
7	Buscador de productos	1	5	2
8	Paginación de productos	1	5	2
9	Generar aplicación en react	1	5	3
10	Configuración proyecto react	2	10	3
11	Desarrollo interfaz navbar	2	10	3
12	Modulo carrito de compras	3	15	3
Estimado:		23	115	

Tabla 25 Plan de entrega aplicación microfrontend

Elaborador por: El Investigador

3.2.1.8 Plan de iteraciones de la aplicación microfrontend

Completada la fase de desarrollo de las historias de usuario con su respectiva iteración, a continuación, se muestra el cumplimiento de las actividades.

Iteración 1

Numero de Historia	**Historia de usuario**	**Prioridad**	**Riesgo**
1	Analisis de la Arquitectura de la aplicación web tipo SPA con microfrontend.	Alta	Alto
2	Generar estructura de la aplicación	Alta	Alto
3	Configuración de module federation en el espacio de trabajo.	Alta	Alto
4	Configuración proyecto Shell.	Alta	Alto
5	Configuración proyecto productos	Alta	Alto

Tabla 26 Iteración 1 aplicación microfrontend

Elaborador por: El Investigador

1. **Analisis de la Arquitectura de la aplicación web tipo SPA con microfrontend.**

Se debe definir correctamente la arquitectura de toda la aplicación, basada en el microfrontend.

Numero de historia: 1	**Programador responsable**: Pablo Alvear
Actividades: • Probar la arquitectura propuesta • Realizar cualquier cambio necesario para la instalación de la estructura	

Tabla 27 Actividades Historia 1 aplicación microfrontend

Elaborador por: El Investigador

2. Generar estructura de la aplicación

Se crea toda la estructura de la aplicación, generando un espacio de trabajo en angular, que en este caso se va a encargar de manejar todos los módulos que se expongan de la aplicación angular y react.

Numero de historia: 2	**Programador responsable**: Pablo Alvear
Actividades: • Establecer todo el espacio de trabajo • Configuraciones iniciales del espacio de trabajo	

Tabla 28 Actividades Historia 2 aplicación microfrontend

Elaborador por: El Investigador

3. Configuración de module federation en el espacio de trabajo.

Se implementa todo el esquema de webpack con las dependencias de module federation en todo el espacio de trabajo, estableciendo el Shell y los remotes (propiedades que es establecen para el uso de microfrontend mediante los módulos que exponen)

Numero de historia: 3	**Programador responsable**: Pablo Alvear
Actividades: • Crear 2 proyectos en el espacio de trabajo • Establecer proyecto Shell • Establecer proyecto remote • Realizar cualquier cambio en la configuración	

Tabla 29 Actividades Historia 3 aplicación microfrontend

Elaborador por: El Investigador

4. Configuración proyecto Shell.

Configuración del proyecto principal de la aplicación, que se va a encargar de contener y exponer las demás aplicaciones

Numero de historia: 4	**Programador responsable**: Pablo Alvear
Actividades: • Configurar proyecto Shell • Probar proyecto Shell	

Tabla 30 Actividades Historia 4 aplicación microfrontend

Elaborador por: El Investigador

5. Configuración proyecto productos

Configuración del proyecto productos, para exponerlo hacia la aplicación shell.

Numero de historia: 5	**Programador responsable**: Pablo Alvear
Actividades: • Configurar proyecto productos • Configurar módulo de exposición • Probar proyecto productos	

Tabla 31 Actividades Historia 5 aplicación microfrontend

Elaborador por: El Investigador

Iteración 2

Numero de Historia	**Historia de usuario**	**Prioridad**	**Riesgo**
6	Desarrollo de la interfaz de productos.	Alta	Medio
7	Buscador de productos	Alta	Medio
8	Paginación de productos	Alta	Medio

Tabla 32 Iteración 2 aplicación microfrontend

Elaborador por: El Investigador

6. Desarrollo de la interfaz de productos.

Diseñar la interfaz de productos aplicando las clases de tailwind.

Numero de historia: 6	**Programador responsable**: Pablo Alvear
Actividades: • Crear prototipo de la interfaz de productos	

• Agregar clases de diseño de tailwind • Ingresar productos desde la base de datos

Tabla 33 Actividades Historia 6 aplicación microfrontend

Elaborador por: El Investigador

7. Buscador de productos

Buscador que filtre los distintos productos

Numero de historia: 7	**Programador responsable**: Pablo Alvear
Actividades: • Crear prototipo del componente buscador • Implementar servicio de búsqueda en el backend • Probar funcionalidad	

Tabla 34 Actividades Historia 7 aplicación microfrontend

Elaborador por: El Investigador

8. Paginación de productos

Paginador de productos para desplazarse entre todo el catálogo.

Numero de historia: 8	**Programador responsable**: Pablo Alvear
Actividades: • Crear prototipo del componente paginador • Implementar lógica de paginación • Probar funcionalidad	

Tabla 35 Actividades Historia 8 aplicación microfrontend

Elaborador por: El Investigador

Iteración 3

Numero de Historia	Historia de usuario	Prioridad	Riesgo
9	Generar aplicación en react	Alta	Alto
10	Configuración proyecto react	Alta	Alto
11	Desarrollo interfaz navbar	Alta	Medio
12	Modulo carrito de compras	Alta	Medio

Tabla 36 Iteración 3 aplicación microfrontend

Elaborador por: El Investigador

9. Generar aplicación en react.

Generar aplicación en react, en este caso para el carrito y el navbar.

Numero de historia: 9	**Programador responsable**: Pablo Alvear
Actividades: • Crear aplicación en react • Instalación de dependencias necesarias	

Tabla 37 Actividades Historia 9 aplicación microfrontend

Elaborador por: El Investigador

10. Configuración proyecto react.

Configurar proyecto en react para exponerlo y se pueda consumir en el shell.

Numero de historia: 10	**Programador responsable**: Pablo Alvear
Actividades: • Configurar module federation en la aplicación • Configurar exposición remota	

Tabla 38 Actividades Historia 10 aplicación microfrontend

Elaborador por: El Investigador

11. Desarrollo interfaz navbar

Diseñar la interfaz navbar, que contendrá el módulo de la lógica del carrito.

Numero de historia: 11	**Programador responsable**: Pablo Alvear
Actividades: • Configurar prototipo interfaz navbar • Agregar clases de diseño tailwind	

Tabla 39 Actividades Historia 11 aplicación microfrontend

Elaborador por: El Investigador

12. Modulo carrito de compras

Desarrollar módulo de carrito que se encarga de gestionar los productos seleccionados para realizar las compras.

Numero de historia: 12	**Programador responsable**: Pablo Alvear
Actividades: • Desarrollar lógica del componente carrito de compras • Configurar proyecto para exponer el módulo carrito de compras	

Tabla 40 Actividades Historia 12 aplicación microfrontend

Elaborador por: El Investigador

3.2.1.9 Historias de usuario para la aplicación monolítica

Para la elaboración de las historias de la aplicación monolítica de usuario se realizó el siguiente modelo:

Historia de Usuario	
Numero:	
Nombre de la historia	
Prioridad:	
Programador Responsable:	
Descripción:	

Tabla 41 Modelo Historias de Usuario aplicación monolítica

Elaborador por: El Investigador

Historia de Usuario	
Numero: 1	
Nombre de la historia	Generar espacio de trabajo
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Descripción: Generar espacio de trabajo en Angular, en este caso va a ser el único ambiente de desarrollo.	

Tabla 42 Historia de usuario 1 Generar espacio de trabajo

Elaborador por: El Investigador

Historia de Usuario	
Numero: 2	
Nombre de la historia	Desarrollo interfaz productos

Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Descripción: Diseñar la interfaz de productos aplicando las clases de tailwind	

Tabla 43 Historia de usuario 2 desarrollo interfaz productos

Elaborador por: El Investigador

Historia de Usuario	
Numero: 3	
Nombre de la historia	Buscador de productos
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Descripción: Buscador que filtre los distintos productos	

Tabla 44 Historia de usuario 3 buscador de productos

Elaborador por: El Investigador

Historia de Usuario	
Numero: 4	
Nombre de la historia	Paginación de productos
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Descripción: Paginador de productos para desplazarse entre todo el catalogo	

Tabla 45 Historia de usuario 4 paginación de productos

Elaborador por: El Investigador

Historia de Usuario	
Numero: 5	
Nombre de la historia	Desarrollo interfaz navbar
Prioridad:	Alta
Programador Responsable:	Pablo Alvear

Descripción: Diseñar la interfaz de navbar aplicando las clases de tailwind

Tabla 46 Historia de usuario 5 desarrollo interfaz navbar

Elaborador por: El Investigador

Historia de Usuario	
Numero: 6	
Nombre de la historia	Desarrollo modulo carrito de compras
Prioridad:	Alta
Programador Responsable:	Pablo Alvear
Descripción: Desarrollar módulo de carrito que se encarga de gestionar los productos seleccionados para realizar las compras**.**	

Tabla 47 Historia de usuario 6 desarrollo modulo carrito de compras

Elaborador por: El Investigador

3.2.1.10 Estimación historias de usuario aplicación monolítica.

Numero de historia	**Historia de usuario**	**Dias**	**Horas**
1	Generar espacio de trabajo	1	5
2	Desarrollo interfaz productos	2	10
3	Buscador de productos	1	5
4	Paginación de productos	1	5
5	Desarrollo interfaz navbar	1	5
6	Desarrollo modulo carrito de compras	2	10
Estimado:		8	35

Tabla 48 Estimación historia de usuario aplicación monolítica

Elaborador por: El Investigador

3.2.1.11 Plan de entrega aplicación monolítica

Numero de historia	Historia de usuario	Dias	Horas	Numero de Iteración
1	Generar espacio de trabajo	1	5	1
2	Desarrollo interfaz productos	2	10	1
3	Buscador de productos	1	5	1
4	Paginación de productos	1	5	1
5	Desarrollo interfaz navbar	1	5	2
6	Desarrollo modulo carrito de compras	2	10	2
Estimado:		8	35	

Tabla 49 Plan de entrega aplicación monolítica

Elaborador por: El Investigador

3.2.1.8 Plan de iteraciones de la aplicación monolítica

Igualmente, que en el desarrollo de historias de la aplicación microfrontend, se presentan las iteraciones con sus respectivas actividades para la aplicación monolítica.

Iteración 1

Numero de Historia	Historia de usuario	Prioridad	Riesgo
1	Generar espacio de trabajo	Alta	Alto
2	Desarrollo interfaz productos	Alta	Alto
3	Buscador de productos	Alta	Alto
4	Paginación de productos	Alta	Alto

Tabla 50 Iteración 1 aplicación monolítica

Elaborador por: El Investigador

1. **Generar espacio de trabajo**

Generar espacio de trabajo en Angular, en este caso va a ser el único ambiente de desarrollo**.**

Numero de historia: 1	**Programador responsable**: Pablo Alvear
Actividades: • Generar el ambiente de desarrollo en angular a través de angular cli	

Tabla 51 Actividades historia 1 aplicación monolítica

Elaborador por: El Investigador

2. **Desarrollo interfaz productos**

Diseñar la interfaz de productos aplicando las clases de tailwind.

Numero de historia: 2	**Programador responsable**: Pablo Alvear
Actividades: • Desarrollar interfaz de productos aplicando las clases de tailwind • Cargar productos desde el servicio de backend	

Tabla 52 Actividades historia 2 aplicación monolítica

Elaborador por: El Investigador

3. Buscador de productos

Buscador que filtre los distintos productos

Numero de historia: 3	**Programador responsable**: Pablo Alvear
Actividades: • Crear prototipo del componente buscador • Implementar servicio de búsqueda en el backend • Probar funcionalidad	

Tabla 53 Actividades historia 3 aplicación monolítica

Elaborador por: El Investigador

4. Paginación de productos

Paginador de productos para desplazarse entre todo el catálogo.

Numero de historia: 4	**Programador responsable**: Pablo Alvear
Actividades: • Crear prototipo del componente paginador • Implementar lógica de paginación • Probar funcionalidad	

Tabla 54 Actividades historia 4 aplicación monolítica

Elaborador por: El Investigador

Iteración 2

Numero de Historia	Historia de usuario	Prioridad	Riesgo
5	Desarrollo interfaz navbar	Alta	Alto
6	Desarrollo modulo carrito de compras	Alta	Alto

Tabla 55 Iteración 2 aplicación monolítica

Elaborador por: El Investigador

5. Desarrollo interfaz navbar

Diseñar la interfaz navbar, que contendrá el módulo de la lógica del carrito

Numero de historia: 5	**Programador responsable**: Pablo Alvear
Actividades: • Configurar prototipo interfaz navbar • Agregar clases de diseño tailwind	

Tabla 56 Actividades historia 5 aplicación monolítica

Elaborador por: El Investigador

6. Desarrollo modulo carrito de compras

Desarrollar módulo de carrito que se encarga de gestionar los productos seleccionados para realizar las compras**.**

Numero de historia: 6	**Programador responsable**: Pablo Alvear
Actividades: • Desarrollar lógica del componente carrito de compras	

Tabla 57 Actividades historia 6 aplicación monolítica

Elaborador por: El Investigador

3.2.2 Fase II: Diseño

3.2.2.1 Tarjetas CRC

Para el proyecto se realizó las tarjetas CRC, que modelan el comportamiento del sistema, en caso solo usando las historias de usuario de la aplicación microfrontend que detallan en si el objetivo principal del proyecto.

Análisis de la arquitectura de la aplicación web tipo SPA con microfrontend	
Responsabilidades	**Colaboradores**
• Definir la arquitectura de la aplicación web tipo SPA con microfrontend. • Identificar los componentes de la aplicación.	Arquitectura

Tabla 58 Tarjeta CRC para el análisis de la arquitectura

Elaborador por: El Investigador

Generar estructura de la aplicación	
Responsabilidades	**Colaboradores**
• Crear los componentes de la aplicación	Arquitectura

• Conectar los componentes de la aplicación • Probar la estructura de la aplicación	

Tabla 59 Tarjeta CRC para la estructura de la aplicación

Elaborador por: El Investigador

Configuración de module federation en el espacio de trabajo	
Responsabilidades	**Colaboradores**
• Crear dos proyectos en el espacio de trabajo: un proyecto principal y un proyecto secundario	Arquitectura

Tabla 60 Tarjeta CRC para la configuración de module federation

Elaborador por: El Investigador

Configuración proyecto shell.	
Responsabilidades	**Colaboradores**
• Exponer proyecto principal, para establecerla como host	Arquitectura

Tabla 61 Tarjeta CRC para la configuración del proyecto Shell

Elaborador por: El Investigador

Configuración proyecto productos	
Responsabilidades	**Colaboradores**
• Exponer proyecto para que sea remote y se pueda	Arquitectura

comunicar con la aplicación principal.	

Tabla 62 Tarjeta CRC para la configuración proyecto productos

Elaborador por: El Investigador

Desarrollo de la interfaz de productos	
Responsabilidades	**Colaboradores**
• Mostrar listado de los productos. • Botones para agregar posteriormente productos al carrito.	Arquitectura Servicio

Tabla 63 Tarjeta CRC para el desarrollo de la interfaz de productos

Elaborador por: El Investigador

Buscador de productos	
Responsabilidades	**Colaboradores**
• Mostrar listado de productos según la búsqueda	Arquitectura Servicio

Tabla 64 Tarjeta CRC para el buscador de productos

Elaborador por: El Investigador

Paginación de productos	
Responsabilidades	**Colaboradores**
• Mostrar listado de productos segmentada por paginas	Arquitectura Servicio

Tabla 65 Tarjeta CRC para la paginación de productos

Elaborador por: El Investigador

Generar aplicación en react	
Responsabilidades	**Colaboradores**
• Espacio de trabajo listo para el desarrollo.	Arquitectura

Tabla 66 Tarjeta CRC para la generar aplicación de react

Elaborador por: El Investigador

Desarrollo interfaz navbar	
Responsabilidades	**Colaboradores**
• Mostrar navbar junto con el icono del carrito de compras	Arquitectura

Tabla 67 Tarjeta CRC para el desarrollo de la interfaz navbar

Elaborador por: El Investigador

Modulo carrito de compras	
Responsabilidades	**Colaboradores**
• Mostrar Listado de productos provenientes de la aplicación productos	Arquitectura

Tabla 68 Tarjeta CRC para el módulo carrito de compras

Elaborador por: El Investigador

3.2.2.2 Iteracion 1

Al ser una aplicación de prueba la iteración 1 contiene el primer microfrontend, en este caso la estructura principal Shell y la aplicación productos

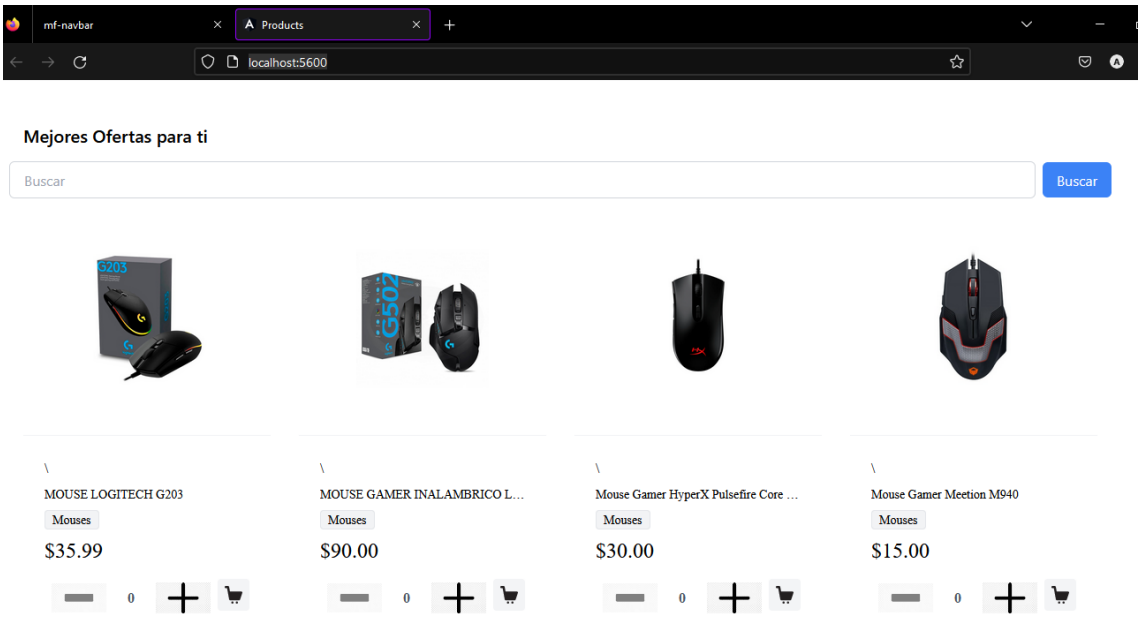


Figura 3 Iteración 1

Elaborador por: El Investigador

3.2.2.2 Iteracion 2

La iteración 2 contiene el componente navbar y la lógica del carrito de compras

Figura 4 Iteración 2

Elaborador por: El Investigador

3.2.3 Fase de codificación

3.2.3.1 Desarrollo del Backend

Se creo un servicio para asignar los productos de prueba para las 2 aplicaciones, se realizaron las siguientes configuraciones en las dependencias:

Figura 5 Configuración Dependencias

Elaborador por: El Investigador

3.2.3.2 Configuración app.js

app.js > ...

```
1   const express= require('express');
2
3   const app= express();
4   const bodyParser=require('body-parser');
5
6   const cors=require('cors');
7
8   app.use(bodyParser.urlencoded({
9       extended:false
10  }));
11
12  app.use(bodyParser.json());
13
14  app.use(cors());
15
16
```

Figura 6 App js

Elaborador por: El Investigador

const express = require('express'); Importa el marco Express al código. Express es un marco de aplicaciones web para Node.js que facilita la creación de aplicaciones web.

const app = express(); Crea una nueva aplicación Express. La variable app se utilizará para configurar y iniciar la aplicación Express.

const bodyParser = require('body-parser'); Importa el middleware body-parser. El middleware body-parser se utiliza para analizar el cuerpo de la solicitud en un objeto JavaScript.

const cors = require('cors'); Importa el middleware cors. El middleware cors se utiliza para habilitar la compartición de recursos de origen cruzado (CORS).

CORS permite que las aplicaciones web hagan solicitudes a otras aplicaciones web que se alojan en dominios diferentes.
app.use(bodyParser.urlencoded({ extended: false })); Configura el middleware body-parser para analizar el cuerpo de la solicitud como datos codificados en URL. La opción extended se establece en false para deshabilitar el análisis de parámetros de consulta extendidos.

app.use(bodyParser.json()); Configura el middleware body-parser para analizar el cuerpo de la solicitud como datos JSON.

app.use(cors()); Configura el middleware cors para habilitar CORS.

3.2.3.3 Configuración conexión a la base de datos

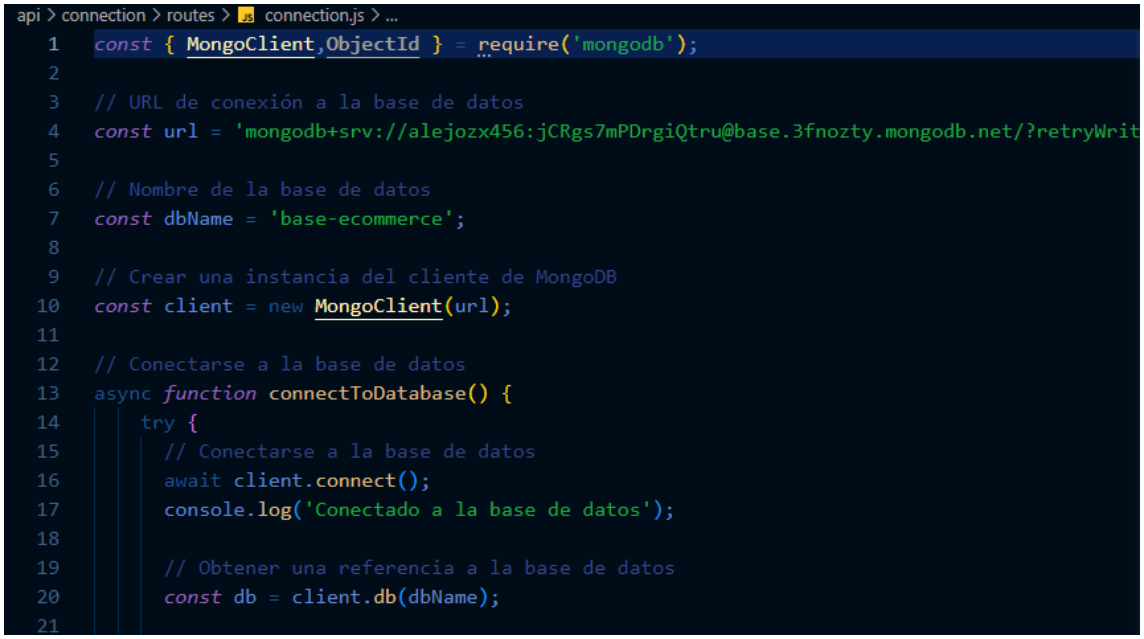


Figura 7 Conexión base de datos

Elaborador por: El Investigador

La función connectToDatabase() comienza importando el módulo mongodb. El módulo mongodb proporciona el cliente de MongoDB que se utilizará para conectarse a la base de datos.

A continuación, la función define la URL de conexión a la base de datos. La URL de conexión incluye el nombre de host, el nombre de usuario, la contraseña y el nombre de la base de datos.

La función luego define el nombre de la base de datos. El nombre de la base de datos es el nombre de la base de datos a la que se conectará la función.

La función luego crea una instancia del cliente de MongoDB. El cliente de MongoDB es el objeto que se utilizará para conectarse a la base de datos.

La función luego llama al método connect() del cliente de MongoDB. El método connect() se utiliza para conectarse a la base de datos.

Después comprueba si la conexión se realizó correctamente. Si la conexión se realizó correctamente, la función imprime un mensaje a la consola.

La función luego obtiene una referencia a la base de datos. La referencia a la base de datos es el objeto que se utilizará para interactuar con la base de datos.

3.2.3.4 Configuración Controlador Productos

```
const express =require('express');
const router=express.Router();
const {  ObjectId } = require('mongodb');
const connectToDatabase=require('./connection');

const jwt=require('jsonwebtoken');
const qs = require('qs');
```

Figura 8 Controlador Productos

Elaborador por: El Investigador

La primera línea, const express = require('express');, importa el marco Express al código. Express es un marco de aplicaciones web para Node.js que facilita la creación de aplicaciones web.

La segunda línea, const router = express.Router();, crea un nuevo enrutador Express. Un enrutador se utiliza para enrutar las solicitudes a diferentes partes de una aplicación.

La tercera línea, const { ObjectId } = require('mongodb');, importa la clase ObjectId del controlador de MongoDB. La clase ObjectId se utiliza para representar ObjectID de MongoDB en JavaScript.

La cuarta línea, const connectToDatabase = require('./connection');, importa la función connectToDatabase del archivo connection. La función connectToDatabase se utiliza para conectarse a la base de datos MongoDB.

La quinta línea, const jwt = require('jsonwebtoken');, importa la biblioteca jsonwebtoken. La biblioteca jsonwebtoken se utiliza para crear y verificar tokens web JSON (JWT).

La sexta línea, const qs = require('qs');, importa la biblioteca qs. La biblioteca qs se utiliza para analizar cadenas de consulta.

3.2.3.5 Ruta get productos

```
router.get('/', async (req, res) => {
  try {
    const db = await connectToDatabase();

    // Obtén una referencia a la colección "productos"
    const collection = db.collection('productos');

    // Realiza la consulta en MongoDB
    const products = await collection.find().toArray();

    res.json(products);
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Error al obtener los productos' });
  }
});
```

Figura 9 Ruta productos

Elaborador por: El Investigador

La ruta toma una solicitud y una respuesta como parámetros. La ruta luego conecta a la base de datos, obtiene una referencia a la colección y realiza una consulta en la colección. Una vez que la consulta se ha completado con éxito, la ruta devuelve los resultados a la respuesta.

La ruta comienza importando la función connectToDatabase. La función connectToDatabase se utiliza para conectarse a la base de datos.

3.2.3.6 Ruta search productos

```
router.get('/search', async (req, res) => {
  try {
    const db = await connectToDatabase();

    // Obtén una referencia a la colección "productos"
    const collection = db.collection('productos');

    // Obtén los parámetros de búsqueda de la URL
    const query = qs.parse(req.query);

    // Construye una expresión regular para la búsqueda
    const searchRegex = new RegExp(query.search, 'i');

    // Realiza la consulta en MongoDB con la expresión regular de búsqueda en el campo
    const products = await collection.find({ title: searchRegex }).toArray();

    res.json(products);
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: 'Error al buscar productos' });
  }
});
```

Figura 10 Ruta buscar productos

Elaborador por: El Investigador

La ruta toma una solicitud y una respuesta como parámetros. La ruta luego conecta a la base de datos, obtiene una referencia a la colección, obtiene los parámetros de búsqueda de la URL, construye una expresión regular para la búsqueda y realiza una consulta en MongoDB con la expresión regular de búsqueda en el campo title. Una vez que la consulta se ha completado con éxito, la ruta devuelve los resultados a la respuesta.

3.2.3.7 Desarrollo aplicación microfrontend

Por medio del uso de la tecnología module federations, se logro implementar la arquitectura microfrontend correctamente, en primer lugar, se creó el espacio de trabajo en angular con 2 proyectos.

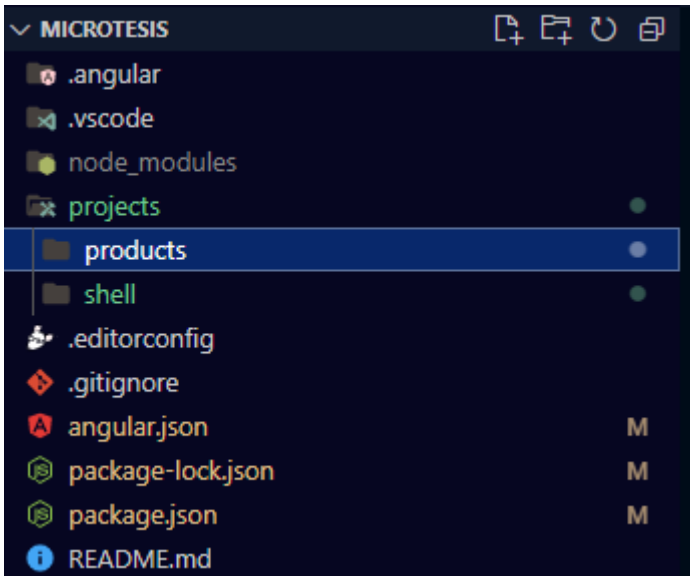


Figura 11 Desarrollo aplicación microfrontend

Elaborador por: El Investigador

3.2.3.8 Module federation proyecto Shell

Mediante el uso de @angular-architects/module-federation, a través de npm se pueden agregar sus dependencias.

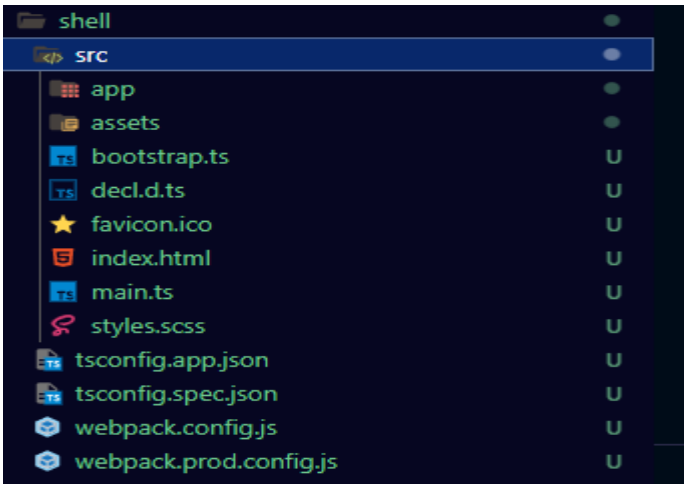


Figura 12 Module federation shell

Elaborador por: El Investigador

3.2.3.9 Configuración module federation proyecto productos

Igualmente, en el proyecto productos instalar las dependencias a través de @angular-architects/module-federation.

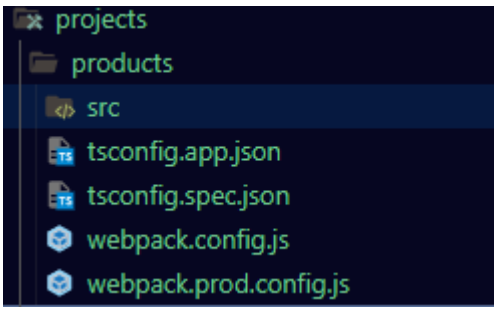


Figura 13 Module federation productos

Elaborador por: El Investigador

3.2.3.10 Package.json

Se puede observar las dependencias en el package json del espacio de trabajo principal del proyecto.

Figura 14 Package.json

Elaborador por: El Investigador

3.2.3.11 Configurar entrada remota aplicación Shell

En la aplicación Shell se configura la entrada remota de la aplicación products de la siguiente manera:

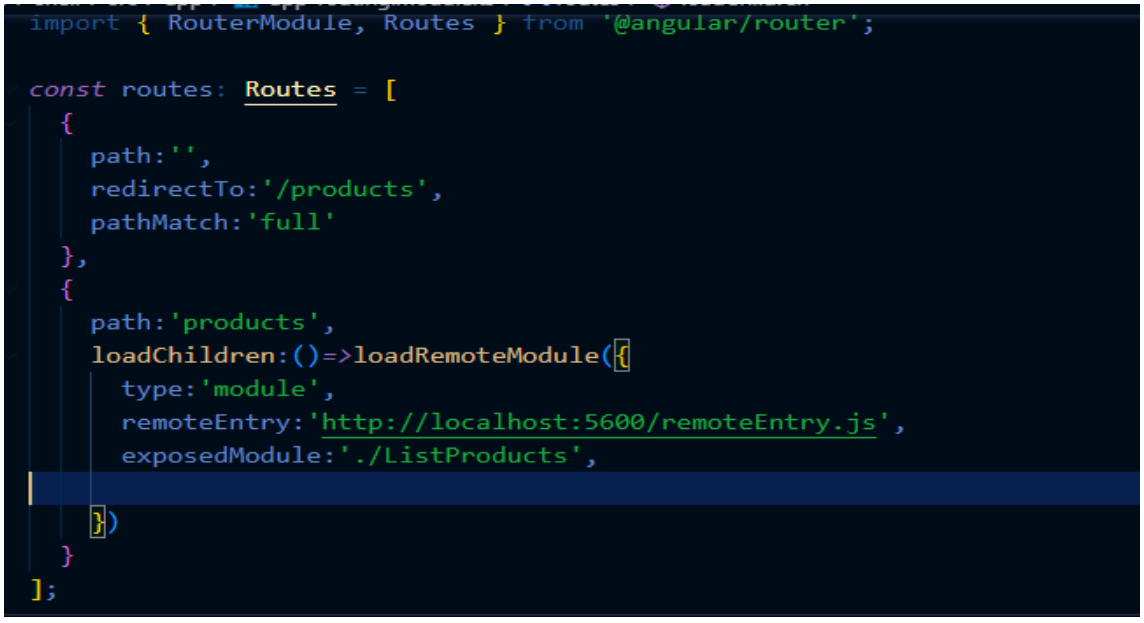


Figura 15 Entrada remota aplicación Shell

Elaborador por: El Investigador

La primera ruta en la definición de rutas es una ruta de redirección. Esta ruta redirige todas las solicitudes a la ruta /products. La propiedad pathMatch de la ruta de redirección se establece en full. Esto significa que la redirección solo ocurrirá si la ruta de la solicitud coincide exactamente con la ruta de la ruta de redirección.

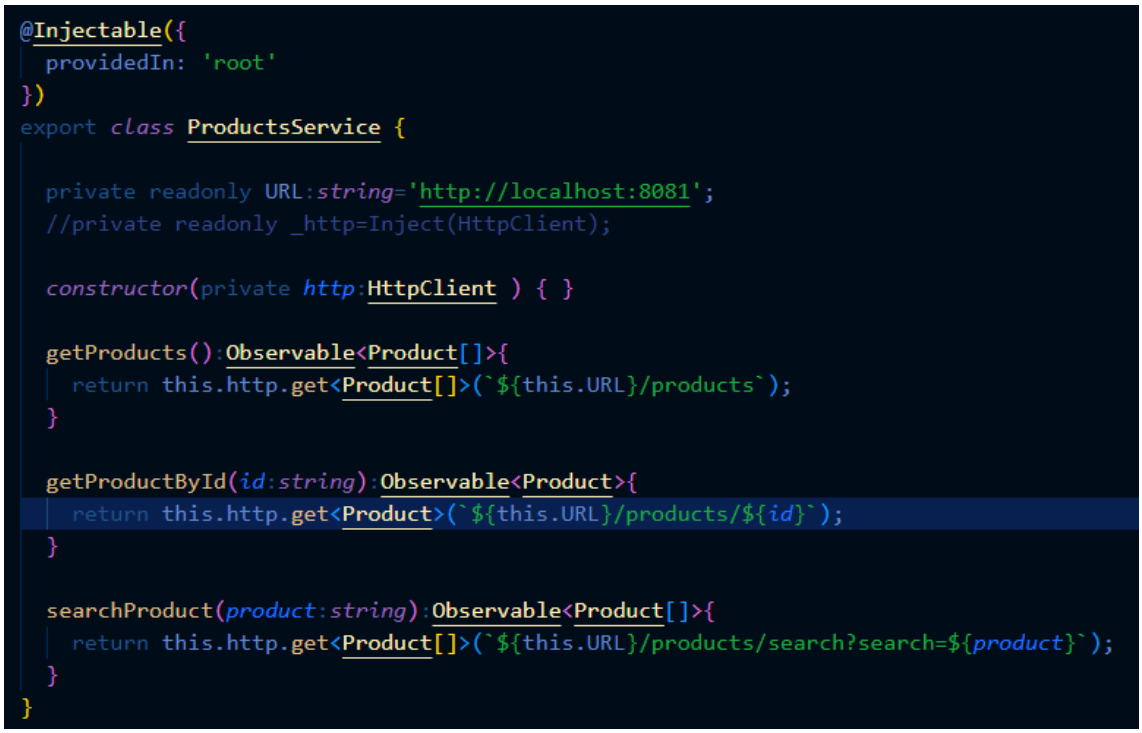
La segunda ruta en la definición de rutas es una ruta remota. Esta ruta carga un módulo remoto de la URL http://localhost:5600/remoteEntry.js. La propiedad exposedModule de la ruta remota se establece en ./ListProducts. Esto significa que el módulo ListProducts del módulo remoto será expuesto a la aplicación Angular.

La función loadRemoteModule se utiliza para cargar un módulo remoto. La función loadRemoteModule toma dos argumentos: el tipo del módulo remoto y la URL del módulo remoto.

3.2.3.12 Interfaz aplicación Productos

Servicio productos

A través de un componente servicio, mediante la clase HttpClient, se crean los métodos para traer a la API creada en express



```
@Injectable({
  providedIn: 'root'
})
export class ProductsService {

  private readonly URL:string='http://localhost:8081';
  //private readonly _http=Inject(HttpClient);

  constructor(private http:HttpClient ) { }

  getProducts():Observable<Product[]>{
    return this.http.get<Product[]>(`${this.URL}/products`);
  }

  getProductById(id:string):Observable<Product>{
    return this.http.get<Product>(`${this.URL}/products/${id}`);
  }

  searchProduct(product:string):Observable<Product[]>{
    return this.http.get<Product[]>(`${this.URL}/products/search?search=${product}`);
  }
}
```

Figura 16 Interfaz aplicacion productos

Elaborador por: El Investigador

Componente card

El componente card contendrá el producto

```
<div
    class="group py-3 transition ease-in-out duration-200 rounded-lg cursor-pointer card
    <div class="flex p-6 justify-center overflow-hidden h-60">
        <img class="mb-6 rounded-lg h-70 object-contain filter"
        [src]="product.image" width="260" height="260" alt="product.title">

    </div>
    <div class="relative flex flex-col p-6 gap-2 font-poppins w-full border-t border-gr
      <div class="relative flex justify-between">
        <h2 class="text-sm truncate">
            {{product.title}}
        </h2>
      </div>
      <span class="px-2 text-sm font-light border bg-gray-200/50 w-fit rounded">
        {{product.category|titlecase}}
      </span>
      <span class="font-medium text-2xl">
        {{product.price|currency}}
      </span>
      <div class="flex bg-white w-full justify-between items-center content-center">
        <app-quantity (quantityProductChanged)="onQuantityProductChange($event)"
        [productId]="product._id"></app-quantity>
        <app-add-to-cart
```

Figura 17 Componente card

Elaborador por: El Investigador

Componente producto

El componente producto contiene el componente card, a través de clases de tailwind se crear el flex para ordenar los productos.

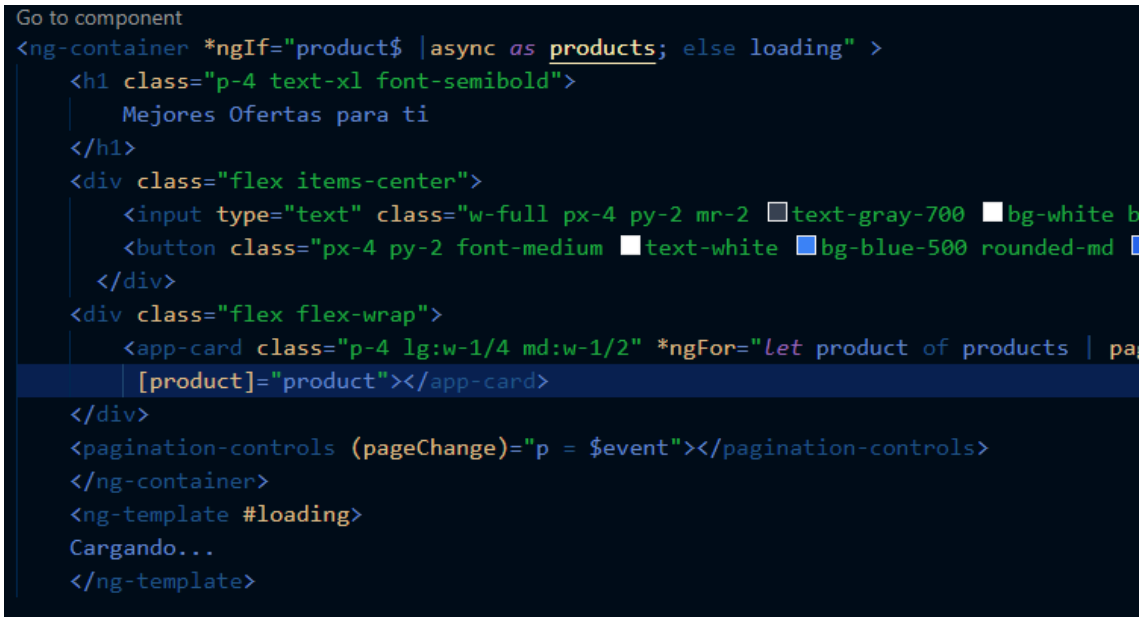


Figura 18 Componente producto

Elaborador por: El Investigador

3.2.3.13 Lógica aplicación productos

Modulo productos

Crear modulo productos para la exposición remota, que apunta directamente hacia el componente productos

Figura 19 lógica aplicación productos

Elaborador por: El Investigador

Modulo principal

El módulo principal se encarga de manejar la ruta principal proveniente del módulo productos

```
const routes:Routes=[
  {path:'',loadChildren:()=>import('./pages/products/products.module')},
]

@NgModule({
  declarations: [
    AppComponent,

  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ProductsModule,
    RouterModule.forRoot(routes)
  ],
```

Figura 20 Modulo principal

Elaborador por: El Investigador

lógica componente card.

Se encuentran los métodos para manejar la cantidad de productos a enviar y finalmente agregarlas al carrito de compras.

```
onQuantityProductChange(productAndCount: QuantityProductChanged) {
  this.productAndCount = productAndCount;
  console.log(this.onQuantityProductChange,productAndCount)
}
onAddProduct(productId: string) {
  if (this.productAndCount.productId === productId) {
    this.showProductAddedText();
    this.mostrarMensaje = true;
    setTimeout(() => {
      this.mostrarMensaje = false;
    }, 3000);
    document.dispatchEvent(
      new CustomEvent('store:addToCart', {
        detail: {
          data: {
            product: this.product,
            count: this.productAndCount.count,
          },
        },
      })
    );
  }
}
```

Figura 21 lógica componente card

Elaborador por: El Investigador

la función onAddProduct envía un evento personalizado llamado store:addToCart. El evento store:addToCart se utiliza para notificar a otras partes de la aplicación que se agregó un producto al carrito.

lógica componente producto

En el componente se inyecto el servicio de productos, además se implementaron los métodos para obtener los productos y buscar productos

```
export class ProductsComponent implements OnInit {
  product$!:Observable<Product[]>;

  private readonly productsApi=inject(ProductsService);
  p=1;
  search!:string;

constructor(){

}
ngOnInit(): void {
  this.getData();
}

getData(){
  this.product$=this.productsApi.getProducts();

}

searchProduct(product:string){
  this.product$=this.productsApi.searchProduct(product);
}
}
```

Figura 22 lógica componente producto

Elaborador por: El Investigador

3.2.3.14 Configurar exposición module federation aplicación products

Se exponen las funciones withModuleFederationPlugin y shared

```
const { shareAll, withModuleFederationPlugin } = require('@angular-architects/module-f

const productsMF = withModuleFederationPlugin({

  name: 'products',

  exposes: {
    './ListProducts': './projects/products/src/app/pages/products/products.module.ts',
  },

  shared: {
    ...shareAll({ singleton: true, strictVersion: true, requiredVersion: 'auto' }),
  },

});

productsMF.output.publicPath='http://localhost:5600/';

module.exports=productsMF;
```

Figura 23 Exposición module federation

Elaborador por: El Investigador

Se define una configuración de federación de módulos llamada productsMF. La configuración productsMF define el nombre del módulo, los módulos que se exponen y los módulos que se comparten.

La propiedad name de la configuración productsMF se establece en products. Este es el nombre del módulo que será cargado por la aplicación.

La propiedad exposes de la configuración productsMF define los módulos que se exponen a la aplicación. La propiedad exposes es un objeto que mapea desde un nombre de módulo hasta la ruta del archivo de módulo. En este caso, la propiedad exposes mapea el nombre ./ListProducts a la ruta ./projects/products/src/app/pages/products/products.module.ts.

3.2.3.15 Proyecto React

Igualmente, el proyecto de react en su estructura cuenta con module federation para la exposición de módulos.

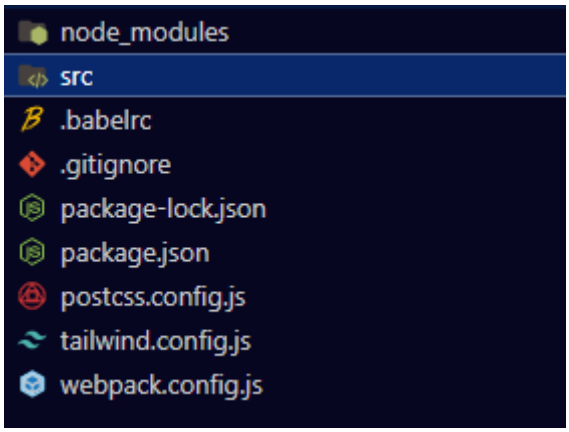


Figura 24 Proyecto react

Elaborador por: El Investigador

3.2.3.16 Interfaz aplicación React

lógica componente nabvar

Componente principal que se encarga de lógica para manejar los eventos que vienen de la aplicación angular

```
const Navbar = () => {
  const [cart, setCart] = useState([]);
  const [quantityProduct, setQuantityProduct] = useState(0);
  const [isHovering, setIsHovering] = useState(false);

  const shouldRenderSummary = () => isHovering && quantityProduct > 0;

  const updateProductCount = (currentCart, product) => {
    const existingProduct = currentCart.find((item) => item.product._id === product.produ

    if (existingProduct) {
      existingProduct.count = product.count;
      return currentCart;
    }

    return [...currentCart, product];
  };

  const calculateProductQuantity = (cart) => {
    return cart.reduce((sum, item) => sum + item.count, 0);
  };
```

Figura 25 lógica componente navbar

Elaborador por: El Investigador

Eventos

El componente navbar recibe la lógica de los eventos que se crearon en angular.

```
useEffect(() => {
  function handleEvent(event) {
    addToCart(event.detail.data);
  }

  document.addEventListener('store:addToCart', handleEvent);

  return () => {
    document.removeEventListener('store:addToCart', handleEvent);
  };
}, []);
```

Figura 26 Manejo de eventos

Elaborador por: El Investigador

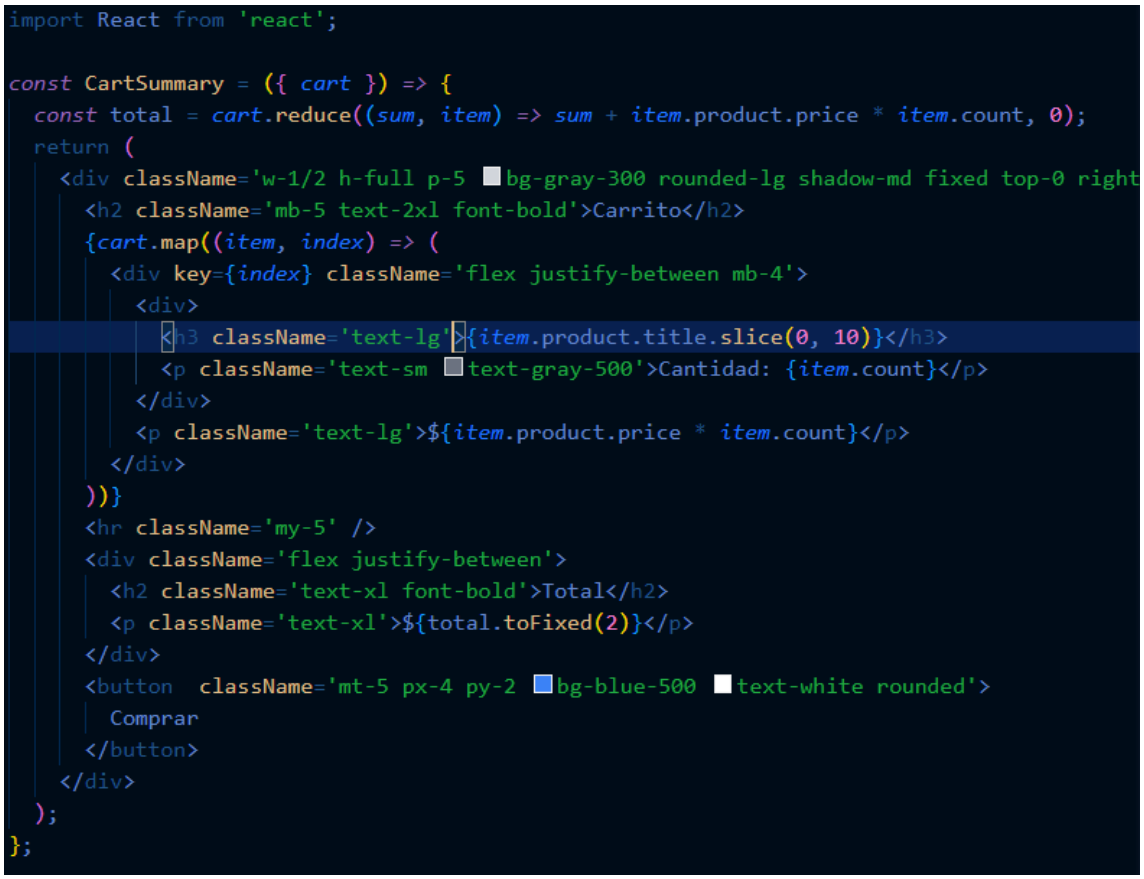
El código primero define una función llamada handleEvent. La función handleEvent se llama cuando se emite el evento store:addToCart. El evento store:addToCart es un evento que se emite cuando se agrega un producto al carrito.

La función handleEvent luego llama a la función addToCart. La función addToCart es una función que agrega un producto al carrito.

El código luego usa el método document.addEventListener para escuchar el evento store:addToCart. El método document.addEventListener toma dos argumentos: el nombre del evento y la función manejadora de eventos. La función manejadora de eventos es la función que se llamará cuando se emita el evento.

Componente carrito

El componente carrito muestra un resumen del carrito donde se encuentran todos los productos



```
import React from 'react';

const CartSummary = ({ cart }) => {
  const total = cart.reduce((sum, item) => sum + item.product.price * item.count, 0);
  return (
    <div className='w-1/2 h-full p-5 bg-gray-300 rounded-lg shadow-md fixed top-0 right
      <h2 className='mb-5 text-2xl font-bold'>Carrito</h2>
      {cart.map((item, index) => (
        <div key={index} className='flex justify-between mb-4'>
          <div>
            <h3 className='text-lg'>{item.product.title.slice(0, 10)}</h3>
            <p className='text-sm text-gray-500'>Cantidad: {item.count}</p>
          </div>
          <p className='text-lg'>${item.product.price * item.count}</p>
        </div>
      ))}
      <hr className='my-5' />
      <div className='flex justify-between'>
        <h2 className='text-xl font-bold'>Total</h2>
        <p className='text-xl'>${total.toFixed(2)}</p>
      </div>
      <button  className='mt-5 px-4 py-2 bg-blue-500 text-white rounded'>
        Comprar
      </button>
    </div>
  );
};
```

Figura 27 Componente carrito

Elaborador por: El Investigador

El componente CartSummary muestra un resumen del carrito, incluyendo los productos en el carrito, la cantidad de cada producto y el precio total. Después una única propiedad, cart, que es un array de objetos que representan los productos en el carrito. El componente CartSummary usa la función reduce para calcular el precio total del carrito. La función reduce toma un valor inicial, 0, e itera a través del array cart, agregando el precio del producto * cantidad al valor inicial para cada producto.

3.2.3.17 Configurar exposición module federation aplicación products

Renderizar aplicación react

A través de la constante fetch_el se puede referenciar el componente app, para posteriormente poder exportarla mediante la función mount

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

const fetch_el = document.getElementById('app');
const mount = (el) => {
  ReactDOM.render(<App />, el);
}

if (fetch_el) {
  mount(fetch_el);
}

export { mount }
```

Figura 28 Renderizar aplicación react

Elaborador por: El Investigador

Exposición del modulo

Se expone la aplicación de microfrontend de react

Figura 29 Exposición del modulo

Elaborador por: El Investigador

La propiedad library en la configuración de ModuleFederationPlugin especifica el tipo de biblioteca que será el microfrontend. En este caso, el microfrontend será una biblioteca de módulo. El microfrontend se cargará como un módulo y se puede importar.

La propiedad name en la configuración de ModuleFederationPlugin especifica el nombre del microfrontend. Este nombre se utilizará para identificar el microfrontend cuando se cargue por otros el Shell de angular.

La propiedad filename en la configuración de ModuleFederationPlugin especifica el nombre del archivo JavaScript que se utilizará para cargar el microfrontend. Este archivo contendrá el código del microfrontend.

La propiedad exposes en la configuración de ModuleFederationPlugin especifica los módulos que se exponen por el microfrontend.

La propiedad shared en la configuración de ModuleFederationPlugin especifica los módulos que son compartidos por el microfrontend y otros microfrontends. Estos módulos se cargarán una vez y estarán disponibles para todos los microfrontends. En este caso, los módulos react y react-dom están compartidos.

3.2.3.18 Consumir microfrontend react en el shell

Componente nabvar

Mediante un div con id se establece el espacio donde se va a configurar el microfrontend

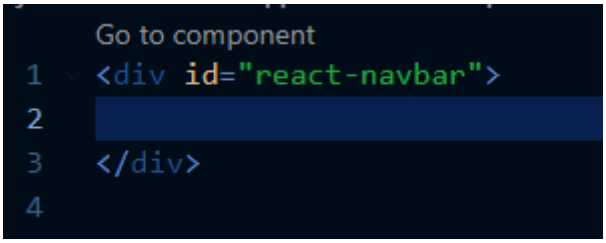


Figura 30 Componente navbar

Elaborador por: El Investigador

lógica componente nabvar

Importar el componente de react a través del nombre que se le dio a la función, en este caso mount

```
import { AfterContentInit, Component } from '@angular/core';

import {mount} from 'mf_navbar/reactAPP';

@Component({
  selector: 'app-navbar-react',
  templateUrl: './navbar-react.component.html',
  styleUrls: ['./navbar-react.component.scss'],
  standalone:true
})
export class NavbarReactComponent implements AfterContentInit {

  ngAfterContentInit(): void {
    const fetch_el=document.getElementById('react-navbar');
    mount(fetch_el);
  }
}
```

Figura 31 lógica componente navbar

Elaborador por: El Investigador

La clase NavbarReactComponent implementa la interfaz AfterContentInit. Esto significa que el método ngAfterContentInit() se llamará después de que se haya inicializado el contenido del componente.

El método ngAfterContentInit() primero obtiene el elemento react-navbar del DOM. Luego, llama a la función mount() para montar el componente de React en el elemento react-navbar.

La propiedad standalone: true en el decorador @Component especifica que el componente NavbarReactComponent es un componente independiente. Esto significa que el componente se puede usar independientemente de otros componentes de Angular.

3.2.3.19 Aplicación monolítica

Se creo un espacio de trabajo en Angular a través de angular cli, igualmente se van a usar las clases de tailwind

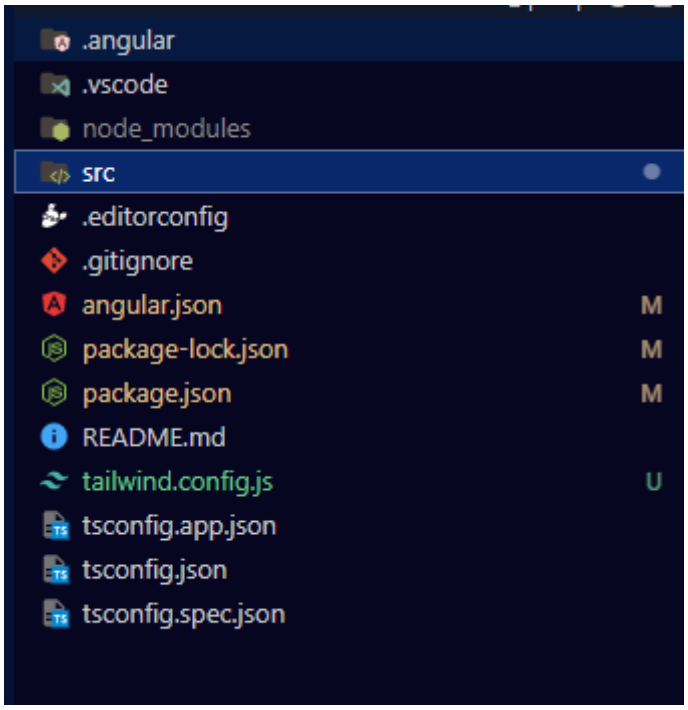


Figura 32 Aplicación monolítica

Elaborador por: El Investigador

3.2.3.20 Interfaz productos

Servicio productos

Igualmente, que en la aplicación de productos se generan servicios, en este caso como es una solo aplicación se usan servicios para los productos y carrito

```
export class StoreService {
  constructor(private httpClient: HttpClient) {}

  getAllProducts(
    limit = '12',
    sort = 'desc',
    category?: string
  ): Observable<Array<Product>> {
    return this.httpClient.get<Array<Product>>(
      `${API_ECOMMERCE}/products${
        category ? '/category/' + category : ''
      }?sort=${sort}&limit=${limit}`
    );
  }
```

Figura 33 Servicio productos

Elaborador por: El Investigador

```
@Injectable({
  providedIn: 'root',
})
export class CartService {
  cart = new BehaviorSubject<Cart>({ items: [] });

  constructor(private _snackBar: MatSnackBar) {}

  addToCart(item: CartItem): void {
    const items = [...this.cart.value.items];

    const itemInCart = items.find((_item) => _item.id === item.id);
    if (itemInCart) {
      itemInCart.quantity += 1;
    } else {
      items.push(item);
    }

    this.cart.next({ items });
    this._snackBar.open('1 item added to cart.', 'Ok', { duration: 3000 });
  }
```

Figura 34 Servicio carrito

Elaborador por: El Investigador

Componente productos

Componente que se encarga de mostrar los productos

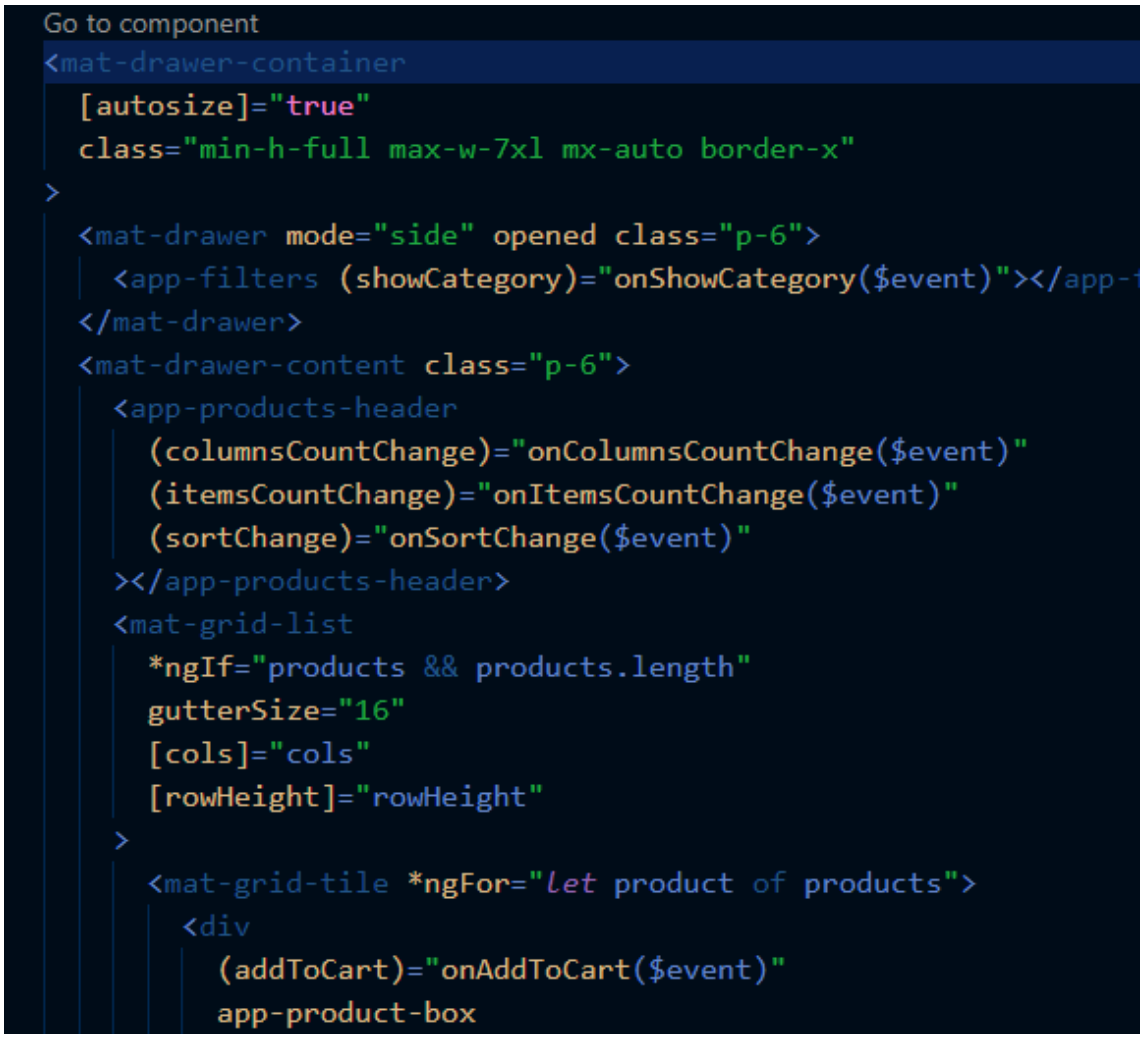


Figura 35 Componente productos

Elaborador por: El Investigador

Lógica productos

Métodos para obtener los productos y agregar al carrito

```
ngOnInit(): void {
  this.getProducts();
}

onColumnsCountChange(colsNum: number): void {
  this.cols = colsNum;
  this.rowHeight = ROWS_HEIGHT[colsNum];
}

onItemsCountChange(count: number): void {
  this.count = count.toString();
  this.getProducts();
}

onSortChange(newSort: string): void {
  this.sort = newSort;
  this.getProducts();
}
```

Figura 36 lógica productos

Elaborador por: El Investigador

Componente carrito

Componente que se encarga de manejar la cantidad de productos

```
<mat-card *ngIf="cart.items.length" class="max-w-7xl mx-auto">
  <table mat-table [dataSource]="dataSource" class="mat-elevation-z8 w-full">
    <ng-container matColumnDef="product">
      <th mat-header-cell *matHeaderCellDef>Product</th>
      <td mat-cell *matCellDef="let element">
        <img src="{{ element.product }}" alt="product" class="w-[100px] my-5" />
      </td>
      <td mat-footer-cell *matFooterCellDef>
        <button mat-raised-button routerLink="/home">Continue Shopping</button>
      </td>
    </ng-container>
    <ng-container matColumnDef="name">
      <th mat-header-cell *matHeaderCellDef>Name</th>
      <td mat-cell *matCellDef="let element">
        <p class="truncate max-w-xs">{{ element.name }}</p>
      </td>
      <td mat-footer-cell *matFooterCellDef></td>
    </ng-container>
    <ng-container matColumnDef="price">
      <th mat-header-cell *matHeaderCellDef>Price</th>
      <td mat-cell *matCellDef="let element">{{ element.price | currency }}</td>
      <td mat-footer-cell *matFooterCellDef></td>
    </ng-container>
```

Figura 37 Componente Carrito

Elaborador por: El Investigador

Lógica carrito

Métodos para obtener los totales y remover productos

```
ngOnInit(): void {
  this.cartSubscription = this.cartService.cart.subscribe((_cart: Cart) => {
    this.cart = _cart;
    this.dataSource = _cart.items;
  });
}

getTotal(items: CartItem[]): number {
  return this.cartService.getTotal(items);
}

onAddQuantity(item: CartItem): void {
  this.cartService.addToCart(item);
}

onRemoveFromCart(item: CartItem): void {
  this.cartService.removeFromCart(item);
}

onRemoveQuantity(item: CartItem): void {
  this.cartService.removeQuantity(item);
}
```

Figura 38 lógica carrito

Elaborador por: El Investigador

3.2.3 Fase de pruebas

A continuación, se muestran las pruebas necesarias para el cumplimento del objetivo principal.

Prueba de aceptación	
Numero: 1	**Historia de Usuario: 1**
Nombre: Analisis de la Arquitectura de la aplicación web tipo SPA con microfrontend.	
Resultado Esperado: la aplicación web se asegura de que esté bien estructurada y fácil de entender.	

Evaluación de Prueba: Satisfactoria

Tabla 69 Prueba de aceptación 1

Elaborador por: El Investigador

Prueba de aceptación	
Numero: 2	**Historia de Usuario: 2**
Nombre: Generar estructura de la aplicación	
Resultado Esperado: La estructura de la aplicación debe ser escalable y adaptable.	
Evaluación de Prueba: Satisfactoria	

Tabla 70 Prueba de aceptación 2

Elaborador por: El Investigador

Prueba de aceptación	
Numero: 3	**Historia de Usuario: 3**
Nombre: Configuración de module federation en el espacio de trabajo.	
Resultado Esperado: El espacio de trabajo debe estar bien configurado para usar module federation.	
Evaluación de Prueba: Satisfactoria	

Tabla 71 Prueba de aceptación 3

Elaborador por: El Investigador

Prueba de aceptación	
Numero: 4	**Historia de Usuario: 4**
Nombre: Configuración proyecto Shell.	
Resultado Esperado: El proyecto shell debe estar bien configurado para poder integrar los distintos modulos	
Evaluación de Prueba: Satisfactoria	

Tabla 72 Prueba de aceptación 4

Elaborador por: El Investigador

Prueba de aceptación	
Numero: 5	**Historia de Usuario: 5**
Nombre: Configuración proyecto productos	
Resultado Esperado: El proyecto productos debe estar bien configurado para poder exponer los distintos módulos	
Evaluación de Prueba: Satisfactoria	

Tabla 73 Prueba de aceptación 5

Elaborador por: El Investigador

Prueba de aceptación	
Numero: 6	**Historia de Usuario: 6**
Nombre: Desarrollo de la interfaz de productos**.**	
Resultado Esperado: La interfaz de productos debe ser fácil de usar y navegar, además de contar con un diseño agradable.	
Evaluación de Prueba: Satisfactoria	

Tabla 74 Prueba de aceptación 6

Elaborador por: El Investigador

Prueba de aceptación	
Numero: 7	**Historia de Usuario: 7**
Nombre: Buscador de productos	
Resultado Esperado: El buscador debe mostrar resultados precisos	
Evaluación de Prueba: Satisfactoria	

Tabla 75 Prueba de aceptación 7

Elaborador por: El Investigador

Prueba de aceptación	
Numero: 8	**Historia de Usuario: 8**
Nombre: Paginación de productos	

Resultado Esperado: Se debe poder desplazar entre los productos normalmente
Evaluación de Prueba: Satisfactoria

Tabla 76 Prueba de aceptación 8

Elaborador por: El Investigador

Prueba de aceptación	
Numero: 9	**Historia de Usuario: 9**
Nombre: Generar aplicación en react	
Resultado Esperado: La aplicación react debe estar bien estructurada y fácil de entender	
Evaluación de Prueba: Satisfactoria	

Tabla 77 Prueba de aceptación 9

Elaborador por: El Investigador

Prueba de aceptación	
Numero: 10	**Historia de Usuario: 10**
Nombre: Configuración proyecto react	
Resultado Esperado: El proyecto React debe ser escalable y adaptable.	
Evaluación de Prueba: Satisfactoria	

Tabla 78 Prueba de aceptación 10

Elaborador por: El Investigador

Prueba de aceptación	
Numero: 11	**Historia de Usuario: 11**
Nombre: Desarrollo interfaz navbar	
Resultado Esperado: La interfaz navbar debe ser visualmente atractiva	
Evaluación de Prueba: Satisfactoria	

Tabla 79 Prueba de aceptación 11

Elaborador por: El Investigador

Prueba de aceptación	
Numero: 12	**Historia de Usuario: 12**
Nombre: Modulo carrito de compras	
Resultado Esperado: Se debe poder agregar productos a su carrito de compras. Se debe poder ver el contenido de su carrito de compras.	
Evaluación de Prueba: Satisfactoria	

Tabla 80 Prueba de aceptación 12

Elaborador por: El Investigador

3.2.4.1 Pruebas de optimización

Se realizaron distintas pruebas para demostrar la optimización y escalabilidad al momento de desarrollar una aplicación microfrontend con tecnologías SPA.
la aplicación microfrontend tienen varias ventajas sobre las aplicaciones monolíticas en términos de tiempo de desarrollo, flexibilidad, escalabilidad, soporte, seguridad y costo. Esto se debe a que las arquitecturas de microfrontends dividen el código en componentes más pequeños y autónomos, lo que facilita el desarrollo, la prueba y la implementación de cambios. Además, las arquitecturas de microfrontends permiten a los equipos trabajar de forma independiente en diferentes componentes, lo que puede acelerar el tiempo de desarrollo.

Mediante el uso del devtools, se obtuvieron las siguientes métricas:

Aplicación Monolítica

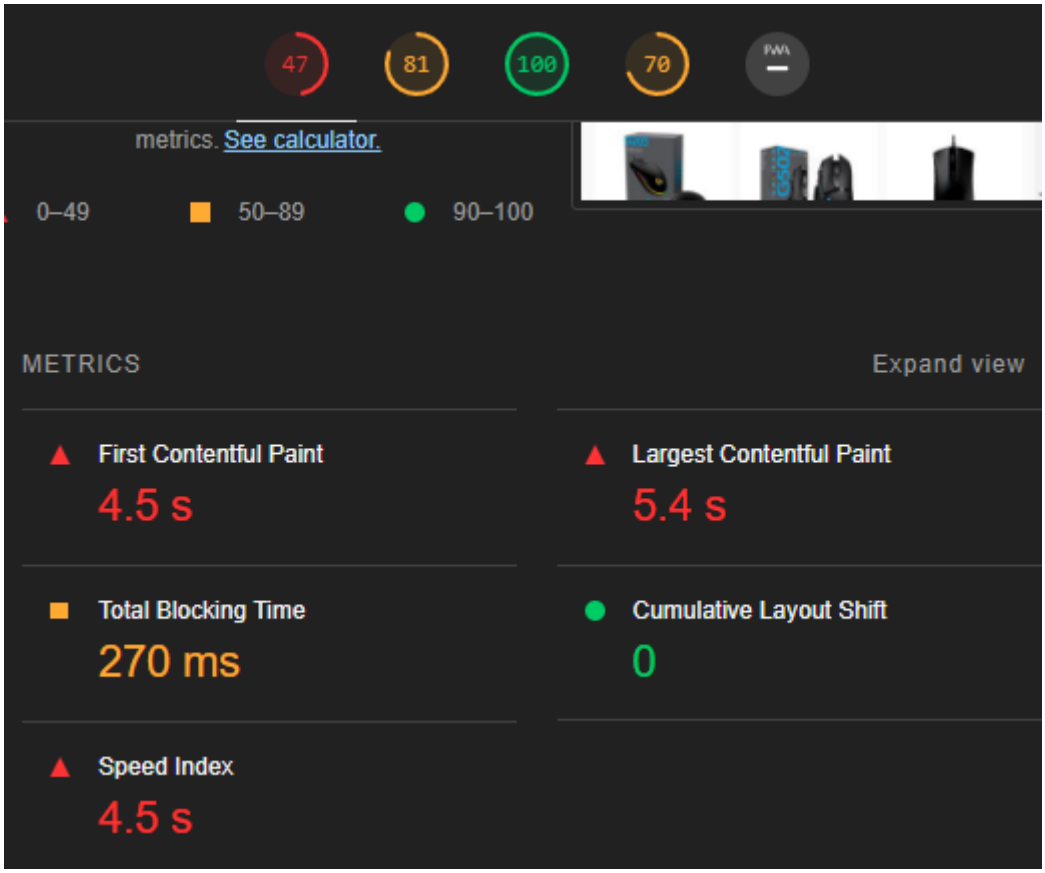


Figura 39 Aplicación Monolítica Métricas 1

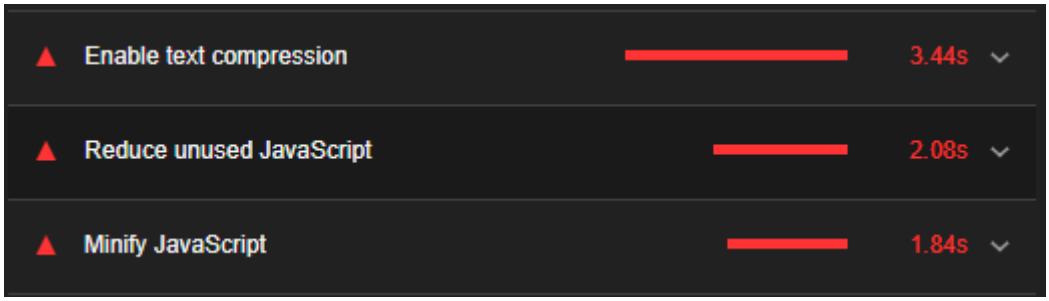


Figura 40 Aplicación Monolítica Métricas 2

Aplicación MicroFrontEnd

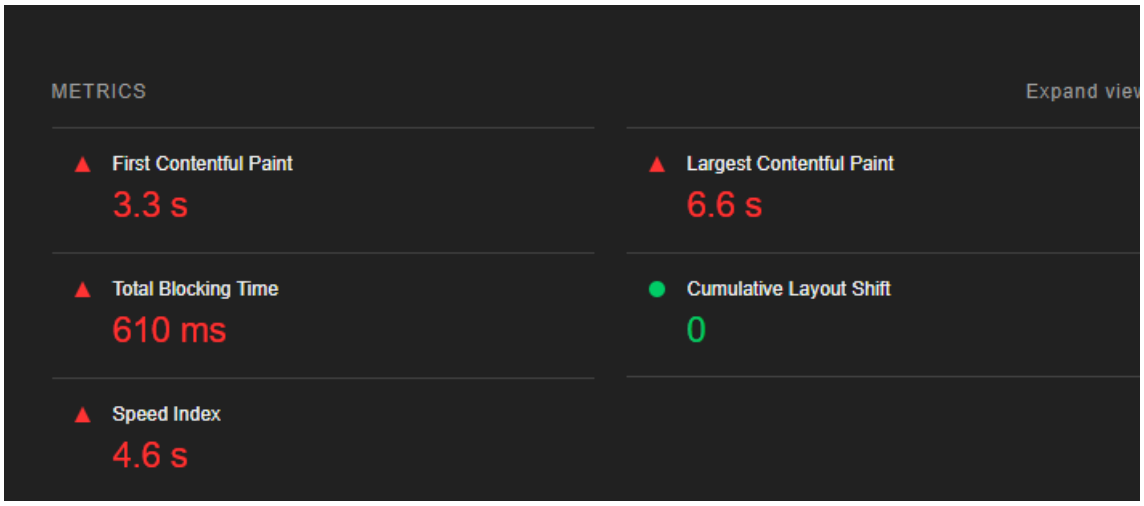


Figura 41 Aplicación MicroFrontend Métricas 1

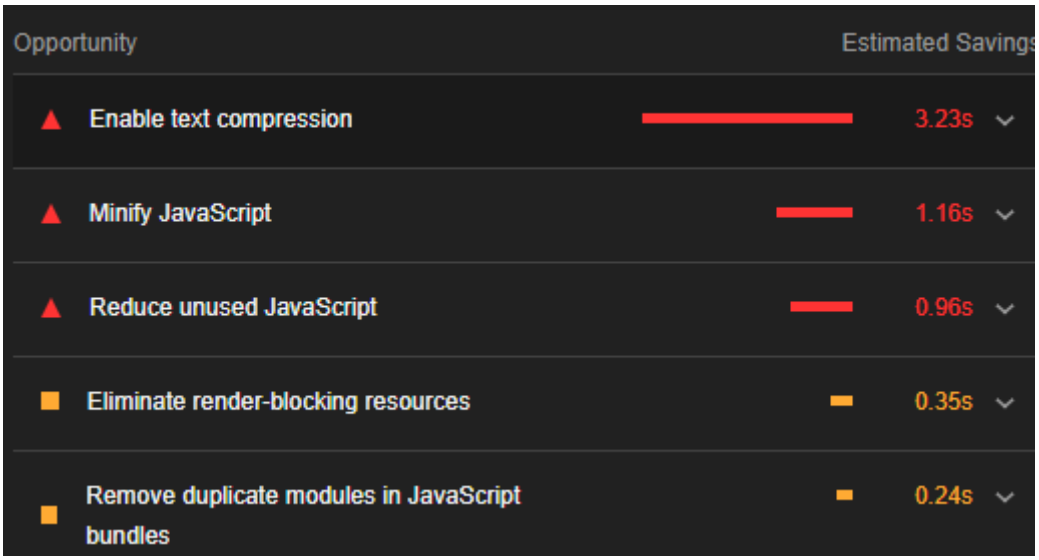


Figura 42 Aplicación Microfrontend Métricas 2

Como se puede ver, la aplicación de microfrontend tienen un tiempo de carga promedio más levemente más rápido principalmente al cargar la página. Esto se debe a que la aplicación de microfrontend se dividió el código en componentes más pequeños y autónomos, lo que facilita la carga de los componentes que se necesitan. Además, la aplicacion de microfrontends pueden utilizar diferentes tecnologías para diferentes componentes, lo que puede mejorar aún más el rendimiento.

CAPÍTULO IV

CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- Angular y React son perfectas para el desarrollo web ya que cuentan con una extensa documentación y comunidad, además de que son compatibles con una amplia gama de librerías que permiten manejar varias características en el proyecto.
- Gracias al instrumento de recolección aplicado se puede concluir que para aplicar una arquitectura microfrontend se necesita una estructura compleja y varios equipos de desarrollo.
- Es posible implementar varios frameworks o librerías de tipo SPA para construir una aplicación microfrontend como angular o react.
- Module Federation es una gran herramienta para el desarrollo de microfrontends ya que demuestra que se pueden trabajar con varios marcos de desarrollo al mismo tiempo.
- La aplicación microfrontend es escalable ya que está compuesta de componentes independientes que pueden ser desarrollados, probados e implementados de forma independiente. Esto permitirá a futuro trabajar de forma más sencilla si es que se desea continuar o extender el proyecto.
- La aplicación microfrontend cuenta con tiempos de carga levemente menores al momento de cargar la página en comparación a una aplicación monolítica

4.2 Recomendaciones

- Se sugiere que antes de empezar a desarrollar una aplicación microfrontend, es necesario tener un plan claro sobre cómo estructurar las aplicaciones. Esto ayudará a evitar problemas más adelante.
- En caso de tener un caso de estudio hay varias herramientas que se pueden utilizar para desarrollar y administrar microfrontends. Es importante elegir las que se adapten a las necesidades y preferencias del equipo desarrollo o la empresa.

- Definir los puntos de integración, son los lugares donde los diferentes microfrontends se comunicarán entre sí. Es importante identificar estos puntos de integración temprano en el proceso de desarrollo para que se los puedan diseñar correctamente.

BIBLIOGRAFIA

[1] B. Amin and S. Kumar, “A Mindtree Whitepaper,” 2021.

[2] S. Peltonen, L. Mezzalira, and D. Taibi, “Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review,” *Inf Softw Technol*, vol. 136, Aug. 2021, doi: 10.1016/j.infsof.2021.106571.

[3] L. Dario Arango Amaya, J. Alexander Diosa Giraldo, and M. en Ingeniería Frank Alexis Castrillon Giraldo, “ARQUITECTURAS DE MICRO FRONTENDS, COMPONENTES WEB Y LIBRERÍAS DE COMPONENTES,” 2021. [Online]. Available: www.udea.edu.co

[4] R. A. Pinto and D. A. Silva, “A Micro Frontends Solution-Analyzing quality attributes,” 2021.

[5] B. Curin, “Informe Final CurinOpt.pdf-PDFA,” 2022.

[6] A. Pavlenko, N. Askarbekuly, S. Megha, and M. Mazzara, “Micro-frontends: Application of microservices to web front-ends,” *Journal of Internet Services and Information Security*, vol. 10, no. 2, pp. 49–66, May 2020, doi: 10.22667/JISIS.2020.05.31.049.

[7] C. De, A. David, B. Adriano, M. Milton, P. López, and R. Riobamba -Ecuador, “APLICACION WEB SPA PARA LA GESTION DE FICHAS MEDICAS EN EL HOSPITAL UNIVERSITARIO ANDINO UTILIZANDO SERVICIOS REST.”

[8] T. De and S. Profesional, “Implementación de una arquitectura micro-frontend para optimizar el desarrollo y despliegue de la aplicación web del sistema de información universitaria de la SUNEDU,” 2022.

[9] M. Thesis, T. Kilamo, and K. Systä, “DESIGN AND IMPLEMENTATION OF MODULAR FRONTEND ARCHITECTURE ON EXISTING APPLICATION,” 2021.

[10] A. De, C. Presentado, J. Pedro, B. Soler, and F. B. Francés, “TFG DISEÑO Y DESARROLLO WEB,” 2013.

[11] N. Rodrigo, J. Tutor, M.-R. Ramón, and J. Nieto Rodrigo, “‘Desarrollo de una aplicación web, con Front-end y Back-end, para compraventa de segunda mano’ TRABAJO FINAL DE GRADO.”

[12] “arquitectura web”.

[13] “MicroFrontEnd y Arquitecturas Web.” [Online]. Available: www.arquitecturajava.com

[14] F. José and G. Peñalvo, “Introducción a la Ingeniería Web Procesos y Métodos de Modelado para la Ingeniería Web y Web Semántica.” [Online]. Available: http://grial.usal.eshttp://twitter.com/frangpContenidos

[15] R. Palacios, “DESARROLLO DE APLICACIONES WEB,” *Universidad Pontificia ICAI ICADE*, 2021.

[16] J. Perez, “Estudio y clasificación de tipos de aplicaciones Web y determinación de atributos de usabilidad más relevantes,” *Universidad Politécnica de Valencia*.

[17] “Arquitectura SPA.” www.arquitecturajava.com

[18] F. Nariño and J. Atuesta, “MODELO DE ARQUITECTURA PARA FRONT-END (BASADO EN MICRO- FRONTENDS) APLICADO AL PRODUCTO DIGITAL DE FUERZA ESPECIALIZADA DE VIVIENDA DEL BANCO DE BOGOTÁ,” *UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS FACULTAD DE INGENIERÍA BOGOTÁ*, 2019.

[19] Son Bui, “Micro frontend: Microservice implementation on Web development,” *Micro frontend: Microservice implementation on Web development*, 2021.

[20] T. De Smet, “Micro frontend architecture for cross framework reusability in practice.” [Online]. Available: https://martinfowler.com/articles/microservices.html

[21] “React Documentation,” *https://react.dev/*.

[22] “Documentación Angular,” *https://docs.angular.lat/docs*.

[23] “Svelte Documentation,” *https://svelte.dev/docs*.

[24] “Preact Documentation,” *https://preactjs.com/guide/v10/getting-started*.

[25] Michael Geers, *Micro-Frontends-in-Action*.