
**UNIVERSIDAD POLITÉCNICA SALESIANA**

**SEDE CUENCA**

**CARRERA DE COMPUTACIÓN**

# IMPLEMENTACIÓN DE UN MIDDLEWARE BASADO EN GRAPHQL PARA LA PUBLICACIÓN Y VENTA DE PAQUETES TURÍSTICOS A TRAVÉS DEL USO DE MICROSERVICIOS

Trabajo de titulación previo a la obtención del título de Ingeniero en Ciencias de la Computación

AUTOR: DAVID ANDRÉS MORALES RIVERA

TUTOR: ING. DIEGO FERNANDO QUISI PERALTA

Cuenca - Ecuador

2022

**CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN**

Yo, David Andrés Morales Rivera con documento de identificación N° 0104493325 manifiesto que:

Soy el autor y responsable del presente trabajo; y, autorizo a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Cuenca, 07 de marzo del 2022

Atentamente,

David Andrés Morales Rivera
0104493325

**CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA**

Yo, David Andrés Morales Rivera con documento de identificación Nº 0104493325, expreso mi voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del Proyecto Técnico: "Implementación de un middleware basado en GraphQL para la publicación y venta de paquetes turísticos a través del uso de microservicios", el cual ha sido desarrollado para optar por el título de: Ingeniero en Ciencias de la Computación, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribo este documento en el momento que hago la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Cuenca, 07 de marzo del 2022

Atentamente,

David Andrés Morales Rivera
0104493325

**CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN**

Yo, Diego Fernando Quisi Peralta con documento de identificación N° 0104616461, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: IMPLEMENTACIÓN DE UN MIDDLEWARE BASADO EN GRAPHQL PARA LA PUBLICACIÓN Y VENTA DE PAQUETES TURÍSTICOS A TRAVÉS DEL USO DE MICROSERVICIOS, realizado por David Andrés Morales Rivera con documento de identificación N° 0104493325, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Cuenca, 04 de marzo del 2022

Atentamente,

Ing. Diego Fernando Quisi Peralta
0104616461

**DEDICATORIA**

*Esta tesis está dedicada a mis padres Luis Gustavo Morales Vélez y Ligia del Roció Rivera García, gracias a su infinito amor y su duro trabajo me es posible culminar este trabajo y esta etapa de mi vida siendo una persona con grandes valores. A mi compañera de vida, Tamara Doménica Fernández de Córdova López, por todo su amor y estar a mi lado en los momentos mas difíciles brindándome su tiempo y apoyo. A mis hermanos Luis Gustavo Morales Rivera, María Emilia Morales Rivera y Kevin Nicolás Morales Rivera por brindarme su apoyo incondicional y cariño.*

*David Andrés Morales Rivera*

**AGRADECIMIENTOS**

*Agradezco a mis padres por todo su amor, ejemplo, dedicación y trabajo que me han permitido culminar esta etapa de mi vida y así cumplir una meta más. A mi novia, Tamara Fernández de Córdova, por su apoyo e impulso para finalizar esta tesis.*

*Agradezco a mi tutor, Diego Quisi Peralta, por permitirme trabajar en conjunto con él, y por brindarme sus conocimientos, experiencia, tiempo y apoyo para realizar este trabajo de la mejor manera.*

*A los docentes de la Universidad Politécnica Salesiana, por compartir sus conocimientos conmigo y de esa manera lograr mi desarrollo en mi vida profesional.*

*Finalmente, agradezco a todos mis amigos que estuvieron presentes en este proceso, gracias por brindarme su apoyo y amistad a lo largo de estos años.*

*David Andrés Morales Rivera*

**RESUMEN**

La necesidad de obtener productos y servicios juegan un rol muy importante dentro de la sociedad, estos mecanismos han ido evolucionando con el paso del tiempo, y consecuentemente han migrado a internet para brindar a las empresas y personas mecanismos mucho mas ágiles y rápidos para obtener, vender y promocionar a gran escala productos o servicios.

La gran mayoría de estos servicios de comercio electrónico, se encuentran desarrollados con técnicas tradicionales que consisten en usar tecnologías centralizadas, provocando comúnmente, perdidas, caídas y ralentización de los servicios brindados, lo que se traduce en perdidas económicas.

Teniendo en cuenta estos antecedentes, en el presente trabajo se propone el desarrollo y la implementación en un ambiente de producción, de un middleware basado en la tecnología GraphQL, en conjunto de una aplicación web progresiva realizada en Flutter, abarcando como principal caso de uso una aplicación que permita un mercado electrónico mediante el mantenimiento, promoción y reserva de paquetes turísticos.

Una vez implementado el middleware se logró distribuir las diferentes tecnologías en microservicios, obteniendo como resultado un middleware que presenta alta disponibilidad, alto rendimiento, seguridad y tolerancia a fallos.

Para corroborar estas características se realizaron pruebas de carga al middleware GraphQL, simulando 50, 100, 1000, 1500 y 2000 solicitudes de usuarios de manera concurrente obteniendo resultados de mas del 98,5% sin errores de solicitudes y con latencias de menos de 3 segundos; y pruebas unitarias a la aplicación cliente.

**Palabras clave:** GraphQL, Flutter, Middleware, PWA, Microservicios, Turismo, Aplicaciones disruptivas.

**ABSTRACT**

The need to obtain products and services play a very important role in society, these mechanisms have evolved over time, and consequently have migrated to the Internet to provide companies and individuals with much more agile and faster mechanisms to obtain, sell and promote products or services on a large scale.

Most of these e-commerce services are developed with traditional techniques that consist of using centralized technologies, commonly causing losses, crashes and slowdowns in the services provided, which translates into economic losses.

Considering this background, this paper proposes the development and implementation in a production environment, of a middleware based on GraphQL technology, together with a progressive web application made in Flutter, covering as main use case an application that allows an electronic market through the maintenance, promotion, and reservation of tourist packages.

Once the middleware was implemented, it was possible to distribute the different technologies in microservices, obtaining as a result a middleware with high availability, high performance, security, and fault tolerance.

To corroborate these characteristics, load tests were performed on GraphQL middleware, simulating 50, 100, 1000, 1500 and 2000 concurrent user requests, obtaining results of more than 98.5% without request errors and with latencies of less than 3 seconds; and unit tests on the client application.

**Keywords:** GraphQL, Flutter, Middleware, PWA, Microservices, Tourism, Disruptive applications.

**Índice de contenido**


1. INTRODUCCIÓN ..... 13

1.1. DESCRIPCIÓN DEL PROBLEMA ..... 13

1.1.1. Antecedentes ..... 13

1.1.2. Importancia y alcances ..... 14

1.2. OBJETIVOS GENERALES Y ESPECÍFICOS ..... 15

1.2.1. Objetivo general ..... 15

1.2.2. Objetivos específicos ..... 15

2. MARCO DE REFERENCIA TEÓRICO ..... 16

2.1. MIDDLEWARE ..... 16

2.2. GRAPHQL ..... 16

2.2.1. Query ..... 17

2.2.2. Mutation ..... 17

2.2.3. Subscription ..... 17

2.2.4. Comparativa vs REST ..... 18

2.3. APOLLO ..... 18

2.4. PRISMA ..... 19

2.5. REDIS ..... 19

2.6. JWT ..... 20

2.7. EXPRESS.JS ..... 20

2.8. DOCKER ..... 20

2.9. APLICACIONES ..... 20

2.9.1. Aplicaciones web ..... 20

2.9.2. Aplicaciones nativas ..... 21

2.9.3. Aplicaciones hibridas ..... 21

3. ANÁLISIS Y REQUERIMIENTOS ..... 22

3.1. ELICITACIÓN DE REQUERIMIENTOS ..... 22

3.1.1. Requerimientos funcionales ..... 22

3.1.2. Requerimientos no funcionales ..... 25
4. METODOLOGÍA ..... 26
4.1. METODOLOGÍA DE DESARROLLO ÁGIL SCRUM ..... 26
4.2. FASES ..... 28
4.2.1. Primera fase ..... 29
4.2.2. Segunda fase ..... 33
4.2.3. Tercera fase ..... 38
4.3. ARQUITECTURA ..... 43
4.3.1. Backend ..... 44
4.3.2. Frontend ..... 45
5. RESULTADOS ..... 47
5.1. PRUEBAS DE CARGA ..... 47
5.2. PRUEBAS UNITARIAS ..... 49
CONCLUSIONES ..... 51
RECOMENDACIONES ..... 53
TRABAJOS FUTUROS ..... 53
REFERENCIAS BIBLIOGRÁFICAS ..... 54

## Índice de figuras

Figura 1. Query (Facebook Open Source, s.f). 17
Figura 2. Mutation (Facebook Open Source, s.f). 17
Figura 3. Apollo GraphQL (Apollo GraphQL, s.f). 19
Figura 4. Diagrama de SCRUM (Hayat et al., 2019). 27
Figura 5. Sprints del proyecto 29
Figura 6. Tecnologías del middleware aplicado. 33
Figura 7. Tecnologías Frontend 38
Figura 8. Interfaz para inicio de sesión 40
Figura 9. Interfaz para registro 41
Figura 10. Interfaz de Paquetes Turísticos 41
Figura 11. Interfaz detalle paquete turístico 42
Figura 12. Interfaz de perfil del usuario 42
Figura 13. Arquitectura del proyecto 43
Figura 14. Arquitectura Backend 44
Figura 15. Arquitectura Frontend 46
Figura 16. Estructura del proyecto 47
Figura 17. Pruebas de carga 48
Figura 18. Pruebas de carga - Latencia 48
Figura 19. Pruebas unitarias - Iniciar Sesión 50
Figura 20. Pruebas unitarias - Crear Paquete Turístico 50

## Índice de tablas

Tabla 1. GraphQL vs REST 18
Tabla 2. RF - Almacenamiento de Información 22
Tabla 3. RF - Mantenimiento de paquetes turísticos 22
Tabla 4. RF - Inicio de sesión 23
Tabla 5. RF - Verificación de cuenta 23

Tabla 6. RF - Reserva de paquetes turísticos ..... 24
Tabla 7. RF - Imagen de perfil ..... 24
Tabla 8. RNF - Tiempo de respuesta ..... 25
Tabla 9. RNF - Diseño adaptativo ..... 25
Tabla 10. RNF - Sistema disponible ..... 25
Tabla 11. RNF - Aplicación PWA ..... 26
Tabla 12. Estado del arte ..... 30
Tabla 13. GraphQL Shield - Regla ..... 36
Tabla 14. GraphQL Shield - Permisos de los resolvers. ..... 36
Tabla 15. Dockerfile ..... 37
Tabla 16. Resultados pruebas de carga ..... 49

# 1. Introducción

El internet y los grandes avances tecnológicos han permitido que las empresas y personas sean capaces de abrirse un nuevo camino para adquirir y vender sus productos. El comercio electrónico ha mejorado sustancialmente la forma en la que las personas hacen negocios, ya que las nuevas tecnologías han permitido el fácil acceso a la personalización de los sitios para brindar algún valor agregado a sus clientes (Walsh & Godfrey, 2000). Una de las mayores ventajas que presentan las ventas por internet, frente a los métodos de venta tradicionales, es la fácil y rápida expansión de los productos para llegar a más personas de diferentes ciudades, países e incluso continentes; esta ventaja, junto a la crisis de la emergencia sanitaria mundial generada por el COVID-19, han puesto a las tecnologías informáticas como las principales soluciones en todos los aspectos de la vida cotidiana, teniendo mayor impacto en servicios para compra y venta a través de internet.

Según (Bhatti et al., 2020) el 52% de los compradores ha evitado realizar sus compras en tiendas físicas, esto ha generado una demanda muy alta en los servicios informáticos de este tipo, llegando a incrementar hasta un 10% el uso del comercio electrónico solamente en el 2020 (Jílková & Králová, 2021). El problema principal se centra en cómo garantizar que los servicios informáticos que permiten la compra y venta de productos presenten un desempeño eficiente y de calidad frente a la gran demanda por el contexto actual que está atravesando la humanidad.

Por otro lado, pensando en la reactivación económica que surgirá en el tema turístico, se propone generar un espacio en el que las empresas y personas puedan promocionar y adquirir servicios y paquetes turísticos a través de aplicaciones multiplataforma basadas en servicios en la web.

## 1.1. Descripción del problema

### 1.1.1. Antecedentes

De acuerdo con los datos brindados por Internet World Stats en el 2017, Ecuador se encontraba catalogado como el país con la mayor penetración de

internet en los países más poblados de Latinoamérica, liderando la lista con el 81%.

Hootsuite, por su parte, en un censo (a enero de 2019) reportó que el 79% de la población ecuatoriana cuenta con acceso a internet. Con este porcentaje se puede determinar que el Ecuador es un país en donde el internet se ha convertido en un servicio de primera necesidad, cargando consigo las ventajas que este ofrece, como lo es el mercado electrónico.

Según los datos de Statista, a nivel mundial en el año 2017, se registraron 1.66 millones de compras en mercados digitales. Estos datos corresponden a que el 21.8% de la población mundial realiza sus compras por internet, y Statista según los datos recogidos, proyecta que para el 2021 supere los 2.14 millones de compras digitales (Alfonso & Boar & Frost & Gambacorta & Liu, 2021).

### 1.1.2. Importancia y alcances

El turismo toma un rol muy importante en temas económicos y de generación de empleo, dentro de empresas afines a esta actividad, ciudades y países. Como resultado de la pandemia del COVID-19, esta actividad fue una de las más afectadas, ya que, según el Anuario de Entradas y Salidas Internacionales del año 2020, emitido por el Instituto Nacional de Estadísticas y Censos, en Ecuador en junio del 2019 se registraron 477.117 arribos internacionales, mientras que en junio del 2020 se registraron 8.100 arribos internacionales (INEC, 2020). Por este motivo, personas, empresas privadas, municipios y gobiernos han impulsado el turismo, una vez que la humanidad se encuentra en una fase de transición hacia la nueva normalidad.

Una nueva normalidad que ha sugerido reinventar todas las actividades económicas, y en este caso el turismo no se queda atrás, dado esto es que se propone potenciar el turismo junto con tecnologías disruptivas permitiendo tener un mayor alcance, y de esta manera llegar a más personas con un proceso que sea simple de usar y estable para el consumidor final.

Por otro lado, las nuevas tendencias tecnológicas y el aumento de usuarios para los sistemas informáticos han obligado a los arquitectos informáticos y desarrolladores de aplicaciones tecnológicas a buscar nuevas tecnologías que ayuden a satisfacer las necesidades de interoperabilidad. Muchos servicios informáticos en la actualidad cuentan con arquitecturas que mantienen todas sus funcionalidades atadas, por tal motivo no soportan grandes volúmenes de peticiones realizadas por los usuarios, generando caídas o perdidas en los servicios.

Una de las nuevas tecnologías que busca solucionar estos problemas es GraphQL, un lenguaje de consultas que se enfoca en la generación de APIs que sean grandes y robustas. Otra de sus principales características es la capacidad para realizar múltiples peticiones a diferentes fuentes o endpoints con un solo llamado, es por esta razón, que en este trabajo se propone implementar una arquitectura orientada a microservicios, teniendo un middleware basado en GraphQL que sea capaz de ofrecer alta disponibilidad, escalabilidad, versatilidad y velocidad en las respuestas.

## 1.2. Objetivos generales y específicos

### 1.2.1. Objetivo general

Construir un middleware que permita publicar microservicios, para facilitar la venta y promoción de servicios y paquetes turísticos a través de tecnologías disruptivas utilizando GraphQL como pasarela, acceso y gestión de servicios.

### 1.2.2. Objetivos específicos

- **OE1.** Estudiar e investigar los fundamentos sobre Middleware, GraphQL y Microservicios.
- **OE2.** Definir e implementar una arquitectura tecnológica basada en GraphQL para generar un middleware.

- **OE3.** Desarrollar e implementar una aplicación para gestionar los planes y paquetes turísticos de una pequeña empresa a través de una web progresiva que permita consumir los servicios del middleware.
- **OE4.** Probar el middleware de gestión de los planes y paquetes turísticos a través de un cliente frontend multiplataforma.
- **OE5.** Validar y generar métricas de rendimiento para el consumo de los servicios GraphQL a través de la aplicación PWA.

# 2. Marco de referencia teórico

## 2.1. Middleware

El middleware es la capa de software que se encarga de la gestión de datos, servicios de aplicaciones, autenticación y gestión de APIs. Su principal característica es ocultar la heterogeneidad de las arquitecturas informáticas, los sistemas operativos, los lenguajes de programación y los protocolos de red para facilitar la construcción del desarrollo y la gestión de aplicaciones (Qilin & Mintian, 2010).

Se puede decir que se trata de una capa de software, que se encuentra por encima del sistema operativo, pero, por debajo de la aplicación que proporciona una abstracción de programación común dentro de un sistema distribuido.

## 2.2. GraphQL

Es un lenguaje que fue desarrollado por Facebook, como solución a varios problemas a los que se enfrentan al utilizar estilos de arquitectura estándar, como REST. GraphQL sirve como un lenguaje de consulta para la API, y también en el lado del servidor para ejecutar consultas utilizando los tipos de datos definidos por el desarrollador (Facebook Open Source, s.f).

El lenguaje comenzó a ganar impulso y comenzó a ser soportado por las principales APIs, entre ellas, GitHub, Airbnb, Netflix y Twitter (Brito & Valente, 2020), ya que

GraphQL evita las múltiples consultas al servidor y permite crear consultas para extraer datos de varias fuentes en una sola llamada a la API.

### 2.2.1. Query

Acción implementada por GraphQL que permite la consulta o lectura de datos dentro de la API. La esencia de GraphQL, se basa en las consultas, ya que el principal fundamento es que el resultado de la API sea lo que el cliente realmente quiere recibir.


Figura 1. Query (Facebook Open Source, s.f).

### 2.2.2. Mutation

Se trata de la acción implementada por GraphQL para permitir la escritura o manipulación de datos en la API. De la misma forma que en las consultas, si el campo de mutación devuelve un tipo de objeto, se pueden pedir únicamente los parámetros o propiedades que sean de interés para el cliente. Esta funcionalidad, ayuda a obtener el nuevo estado de un objeto después de una modificación.


Figura 2. Mutation (Facebook Open Source, s.f).

### 2.2.3. Subscription

Las suscripciones de GraphQL son operaciones que tienen la habilidad de cambiar su resultado. La principal diferencia de las suscripciones frente a las consultas es que son operaciones de larga duración, debido a que mantienen una

conexión activa con el servidor, de esta forma el servidor puede enviar actualizaciones del resultado esperado (Facebook Open Source, s.f).

El principal uso de esta acción es notificar al cliente en tiempo real los cambios en los datos, como la creación de un nuevo objeto o las actualizaciones de un campo importante.

### 2.2.4. Comparativa vs REST

En la actualidad, REST es considerado como el estándar para el diseño de APIs, ya que presenta ventajas como servidores sin estado y acceso estructurado a los recursos. Sin embargo, debido al avance de la tecnología, a dado como consecuencia nuevos requerimientos de los clientes, requerimientos en los que REST ha presentado dificultades para adaptarse (Facebook Open Source, s.f).

GraphQL se desarrolló para atender a estas necesidades, brindado APIs con mayor flexibilidad y eficiencia. Por esto, a continuación, se presentan las principales diferencias entre estas tecnologías.

**Tabla 1. GraphQL vs REST**

| GraphQL | REST |
|---|---|
| Una URL → Un recurso (Varios endpoints) | Una URL → Toda la información (Un endpoint) |
| No se puede elegir lo que se va a recibir en el JSON | Se puede elegir lo que se va a recibir en el JSON |
| Versionado | No existe versionamiento |
| No autodocumentado | Autodocumentado |
| Almacenamiento en cache | No tiene almacenamiento en cache |

## 2.3.Apollo

Es una plataforma de construcción de grafos de datos, desarrollada por Meteor Development Group Inc. Un grafo de datos es una capa que posibilita la comunicación entre las aplicaciones y los servicios internos.

Apollo ayuda a construir, consultar y gestionar un grafo de datos. Se trata de una capa de datos que permite a las aplicaciones interactuar con los datos de los almacenes de datos conectados y APIs externas. El grafo de datos se sitúa entre la aplicación y los servicios backend, facilitando el flujo de datos entre ellos (Apollo GraphQL, s.f).

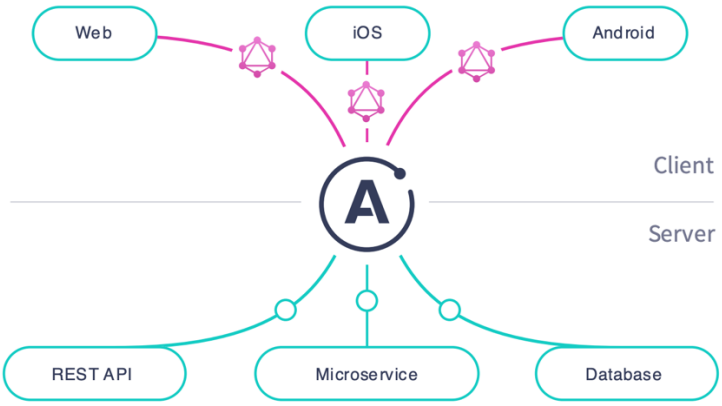


Figura 3. Apollo GraphQL (Apollo GraphQL, s.f).

## 2.4. Prisma

Es un conector de base de datos en tiempo real, su principal característica es convertir una base de datos en un API GraphQL.

Prisma funciona generando un servidor, que actúa como un proxy entre la base de datos y un motor de consultas de alto rendimiento, de esta manera se posibilita realizar consultas sobre la base de datos (Prisma, s.f).

## 2.5. Redis

Redis (Remote Dictionary Server) es una base de datos remota en memoria esto genera tiempos óptimos de respuesta en la recuperación de los datos (Carlson, 2013). Ofrece alto rendimiento principalmente a que tiene un motor de almacenamiento basado en clave-valor, como si se tratara de un diccionario de datos.

## 2.6. JWT

JWT (Json Web Token) es un estándar abierto utilizado para compartir información de seguridad entre dos partes, generalmente un cliente y un servidor. Cada JWT esta conformado por uno o varios objetos JSON codificados, incluyendo un conjunto de restricciones. Los JWT se firman utilizando un algoritmo criptográfico para garantizar que las restricciones no puedan ser alteradas después de la emisión del token.

## 2.7. Express.js

Express.js es un framework para Node.js. Está diseñado para la construcción de aplicaciones web y APIs, ayudando a manejar de manera sencilla las peticiones que ingresen al servidor.

Express.js, permite a los desarrolladores ahorrar tiempo y centrarse en otras tareas importantes como la lógica del negocio.

## 2.8. Docker

Docker es una plataforma de software que se fundamente en crear contenedores que sean ligeros y portables. Un contenedor se define como una unidad que incluye todo lo necesario para que una aplicación se ejecute, incluyendo bibliotecas y herramientas que se utilizan dentro del código y también en tiempo de ejecución.

Estos contenedores sirven para que las aplicaciones de software puedan ejecutarse en cualquier máquina que cuente con Docker instalado, independientemente del sistema operativo, facilitando de esta manera los despliegues.

## 2.9. Aplicaciones

### 2.9.1. Aplicaciones web

Para estas aplicaciones los datos y los archivos son procesados y almacenados dentro de la web, por este motivo se codifican en un lenguaje que sea soportado por los navegadores y que se pueda ejecutar en los mismos. Estas aplicaciones no necesitan ser instaladas en el dispositivo cliente.

### 2.9.2. Aplicaciones nativas

Dentro de estas aplicaciones están aquellas que son desarrolladas y destinadas para un sistema operativo en especifico. Este tipo de aplicaciones aprovecha por completo las funcionalidades que ofrece un sistema operativo y dispositivo tanto en hardware como software. La principal desventaja de este tipo de aplicaciones son los costos y tiempos de implementación, así como para su mantenimiento.

### 2.9.3. Aplicaciones hibridas

Las aplicaciones hibridas combinan elementos y características de las aplicaciones nativas y aplicaciones web. Estas aplicaciones tienen la capacidad de poder utilizar cualquier funcionalidad que brinde el sistema operativo y el dispositivo en el que se está ejecutando a través de un navegador integrado dentro de la aplicación. Dentro de la comunidad de desarrolladores este tipo de aplicaciones ha ganado popularidad, ya que permiten escribir código que se adapta a múltiples plataformas.

#### 2.9.3.1. Flutter

Flutter es un SDK[1] desarrollado y mantenido por Google que fue lanzado oficialmente al mercado en el 2018. El principal objetivo de Flutter es la creación de interfaces de software para diferentes plataformas.

Flutter dentro de su SDK provee al desarrollador de un motor de renderizado, diferentes componentes para la interfaz de usuario, frameworks para realizar tests, mantenimiento de rutas, etc., todo esto, bajo el lenguaje de programación Dart (Windmill, 2020).

[1] **SDK:** Software Development Kit, o en español Kit de Desarrollo de Software. Se denomina SDK al conjunto de herramientas que ofrece el fabricante, de un sistema operativo o lenguaje de programación, a su comunidad o clientes.

# 3. Análisis y requerimientos

## 3.1. Elicitación de requerimientos

### 3.1.1. Requerimientos funcionales

**Tabla 2. RF - Almacenamiento de Información**

| **Identificador:** RF01 | **Nombre:** Almacenamiento de Información. | **Funcional** |
|---|---|---|
| **Descripción:** El sistema debe registrar el nombre, correo, contraseña, teléfono y país de los usuarios. | **Categoría (Visible/No Visible):** No Visible | |
| **Objetivo:** Almacenar en una base de datos la información básica de los usuarios. | | |
| **Datos de entrada:** Datos personales del usuario | **Datos de salida:** | |
| **Criterios de aceptación:** El correo electrónico debe ser único en la base de datos. | | |
| **Precondición:** Contar con una interfaz grafica que permita al usuario el ingreso de los datos. | | |
| **Post condición:** Validar que el correo electrónico sea válido. | | |
| **Prioridad:** Alta | | |

**Tabla 3. RF - Mantenimiento de paquetes turísticos**

| **Identificador:** RF02 | **Nombre:** Mantenimiento de paquetes turísticos. | **Funcional** |
|---|---|---|
| **Descripción:** El sistema debe permitir la visualización, creación, edición y eliminar los paquetes turísticos. | **Categoría (Visible/No Visible):** Visible | |
| **Objetivo:** Tener control de los diferentes paquetes turísticos dentro de la aplicación. | | |
| **Datos de entrada:** Paquete turístico | **Datos de salida:** | |
| **Criterios de aceptación:** La información de los paquetes turísticos debe contener todos los campos requeridos. | | |
| **Precondición:** Únicamente los usuarios administradores podrán realizar los mantenimientos a los paquetes turísticos. | | |

| |
|---|
| **Post condición:** Los paquetes turísticos deben ser visualizados por los usuarios. |
| **Prioridad:** Alta |

**Tabla 4. RF - Inicio de sesión**

| | | |
|---|---|---|
| **Identificador:** RF03 | **Nombre:** Inicio de sesión. | **Funcional** |
| **Descripción:** El sistema debe permitir a los usuarios iniciar sesión mediante credenciales o redes sociales. | **Categoría (Visible/No Visible):** Visible | |
| **Objetivo:** Brindar al usuario varias opciones para que pueda ingresar al sistema. | | |
| **Datos de entrada:** Credenciales | **Datos de salida:** Token | |
| **Criterios de aceptación:** | | |
| **Precondición:** El usuario debe estar registrado en el sistema. | | |
| **Post condición:** | | |
| **Prioridad:** Alta | | |

**Tabla 5. RF - Verificación de cuenta**

| | | |
|---|---|---|
| **Identificador:** RF04 | **Nombre:** Verificación de cuenta. | **Funcional** |
| **Descripción:** El sistema debe enviar vía correo electrónico un código único para verificar la cuenta. | **Categoría (Visible/No Visible):** Visible | |
| **Objetivo:** Verificar que los correos electrónicos registrados existan y pertenezcan al usuario. | | |
| **Datos de entrada:** Credenciales | **Datos de salida:** Correo electrónico con código único. | |
| **Criterios de aceptación:** El código generado debe ser único para cada usuario. | | |
| **Precondición:** El usuario debe haberse registrado en el sistema. | | |
| **Post condición:** El usuario debe ingresar el código que fue enviado por correo electrónico en el sistema. | | |
| **Prioridad:** Alta | | |

**Tabla 6. RF - Reserva de paquetes turísticos**

| **Identificador:** RF05 | **Nombre:** Reserva de paquetes turísticos. | **Funcional** |
|---|---|---|
| **Descripción:** El sistema debe permitir a los usuarios reservar paquetes turísticos. | **Categoría (Visible/No Visible):** Visible | |
| **Objetivo:** Generar reservas en los diferentes paquetes turísticos. | | |
| **Datos de entrada:**<br>Paquete turístico.<br>Fecha de reserva | **Datos de salida:**<br>Listado de paquetes turísticos reservados | |
| **Criterios de aceptación:** Las reservas se visualizan en tiempo real en la interfaz del administrador | | |
| **Precondición:** El usuario debe agregar los paquetes turísticos a reservar en el carrito de compras. | | |
| **Post condición:** | | |
| **Prioridad:** Alta | | |

**Tabla 7. RF - Imagen de perfil**

| **Identificador:** RF06 | **Nombre:** Imagen de perfil. | **Funcional** |
|---|---|---|
| **Descripción:** El sistema debe permitir a los usuarios subir y actualizar su imagen de perfil. | **Categoría (Visible/No Visible):** Visible | |
| **Objetivo:** Personalizar el perfil del usuario. | | |
| **Datos de entrada:**<br>Imagen | **Datos de salida:**<br>Visualización de imagen cargada | |
| **Criterios de aceptación:** La imagen se guarda en el servidor y el cambio es reflejado en el perfil del usuario. | | |
| **Precondición:** El usuario debe estar registrado en el sistema. | | |
| **Post condición:** | | |
| **Prioridad:** Media | | |

### 3.1.2. Requerimientos no funcionales

**Tabla 8. RNF - Tiempo de respuesta**

| **Identificador:** RNF01 | **Nombre:** Tiempo de respuesta. | **No Funcional** |
|---|---|---|
| **Descripción:** El sistema no debe demorarse mas de 10 segundos en dar una respuesta al usuario. | **Categoría (Visible/No Visible):** Visible | |
| **Objetivo:** Mostrar al usuario que los diferentes procesos se están realizado para brindarle una mejor experiencia con el sistema. | | |
| **Datos de entrada:** | **Datos de salida:** | |
| **Criterios de aceptación:** Mostrar indicadores de carga cuando se este realizando un proceso e indicadores de que algún proceso finalizo. | | |
| **Precondición:** | | |
| **Post condición:** | | |
| **Prioridad:** Media | | |

**Tabla 9. RNF - Diseño adaptativo**

| **Identificador:** RNF02 | **Nombre:** Diseño adaptativo. | **No Funcional** |
|---|---|---|
| **Descripción:** Las interfaces del sistema deben acoplarse a los diferentes tamaños de los dispositivos. | **Categoría (Visible/No Visible):** Visible | |
| **Objetivo:** Brindar al usuario una experiencia optima en cualquier dispositivo que use. | | |
| **Datos de entrada:** | **Datos de salida:** | |
| **Criterios de aceptación:** El diseño de la aplicación debe ser adaptativo. | | |
| **Precondición:** | | |
| **Post condición:** | | |
| **Prioridad:** Media | | |

**Tabla 10. RNF - Sistema disponible**

| **Identificador:** RNF03 | **Nombre:** Sistema disponible. | **No Funcional** |
|---|---|---|
| **Descripción:** El sistema debe estar disponible las 24 horas del día. | **Categoría (Visible/No Visible):** No Visible | |

| | |
|---|---|
| **Objetivo:** Dotar de disponibilidad al sistema para que los usuarios y administradores puedan acceder al sistema en cualquier momento. | |
| **Datos de entrada:** | **Datos de salida:** |
| **Criterios de aceptación:** El sistema debe contar con alta disponibilidad. | |
| **Precondición:** Configurar el servidor para que levante una nueva instancia del sistema si es que esta llega a caerse. | |
| **Post condición:** | |
| **Prioridad:** Media | |

**Tabla 11. RNF - Aplicación PWA**

| | | |
|---|---|---|
| **Identificador:** RNF04 | **Nombre:** Aplicación PWA. | **No Funcional** |
| **Descripción:** El sistema debe permitir instalarse en los diferentes dispositivos que se este usando. | **Categoría (Visible/No Visible):** Visible | |
| **Objetivo:** Instalar el sistema dentro del dispositivo para que pueda tener un mejor rendimiento. | | |
| **Datos de entrada:** | **Datos de salida:** | |
| **Criterios de aceptación:** La aplicación se instala en el dispositivo y se puede usar sin problemas. | | |
| **Precondición:** Iniciar el sistema dentro de un navegador web para que este pueda ser instalado. | | |
| **Post condición:** | | |
| **Prioridad:** Baja | | |

# 4. Metodología

## 4.1. Metodología de desarrollo ágil SCRUM

Es una metodología ágil que proporciona la flexibilidad para controlar y gestionar los requerimientos levantados para un proyecto, así como para el desarrollo de software que busca entregar productos que tengan el máximo valor posible tanto productivo, como creativo. Dentro de esta metodología ágil se emplea un enfoque repetitivo y progresivo para optimizar el control de riesgos (Hayat et al., 2019).

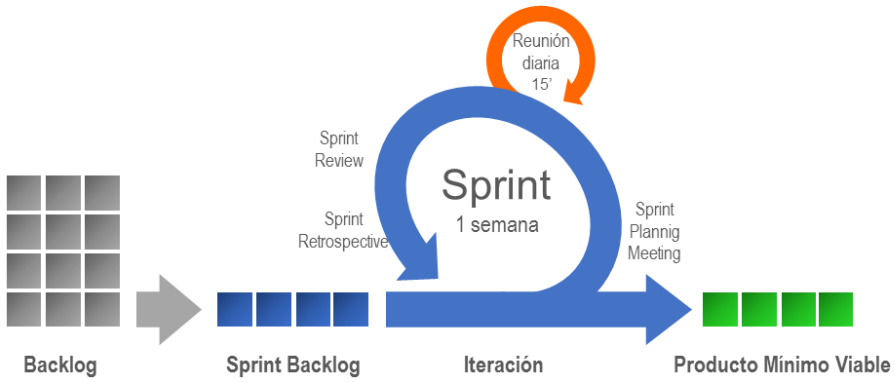


Figura 4. Diagrama de SCRUM (Hayat et al., 2019).

Dentro de SCRUM existen dos roles importantes, identificados como:

- **Product Owner. -** responsable de agigantar el valor del producto que se va a tener como resultado. Es el único responsable de gestionar el Product Backlog, que es una lista de todo lo que se conoce que es necesario en el producto (Mahalakshmi & Sundararajan, 2013).
- **SCRUM Máster. -** es el líder que tiene como tarea ayudar a todos los miembros del equipo a entender la teoría, prácticas, reglas y valores de SCRUM (Mahalakshmi & Sundararajan, 2013).

SCRUM, dentro de su metodología cuenta con eventos predefinidos, estos eventos sirven para crear una regularidad dentro del equipo.

- **Sprint. -** es la parte más importante dentro de la metodología SCRUM, y se trata de un lapso que tiene como máximo 4 semanas, aquí es en donde se trabaja con el equipo para realizar los objetivos y entregar un producto funcional.
- **Daily SCRUM. –** según (Mahalakshmi & Sundararajan, 2013), definen una daily SCRUM como una reunión diaria con duración de máximo 15 minutos en la que participa el equipo de desarrollo. En esta reunión se realiza una breve evaluación de los que se hizo y lo que se va a hacer, también en esta reunión se identifican los obstáculos que impiden avanzar con el cumplimiento de los objetivos del sprint.

Dentro de SCRUM, se tienen artefactos, los mismos que ayudan a dar valor a la transparencia de la información que es importante y que todos los miembros de SCRUM tengan el mismo entendimiento.

- **Product Backlog. -** o lista de producto, es la única fuente en donde se puede encontrar todo lo que es necesario dentro del proyecto. En esta lista se pueden encontrar características, funcionalidades, requisitos mejoras y correcciones sobre el proyecto para su entrega.

### 4.2. Fases

Para el desarrollo de este trabajo, se utilizó la metodología SCRUM, debido a que tiene como finalidad la entrega de productos que contengan algún valor en tiempos cortos.

Los roles que se identificaron dentro de la metodología son, Ing. Diego Quisi como Product Owner y Cliente, y David Morales como SCRUM Máster. Para la definición del sprint se tomaron en cuenta los objetivos específicos definidos.

Los sprints tuvieron una duración entre una y tres semanas, se definió de esta manera con el objetivo de mantener un contacto más cercano con el cliente y que al finalizar cada sprint, analizar las actividades cumplidas para redefinir el siguiente sprint en el caso que se lo requiera, esto con el sentido de determinar si es necesario revisar y mejorar un producto funcional, o dar paso al incremento de funcionalidades con el desarrollo de los nuevos requerimientos.

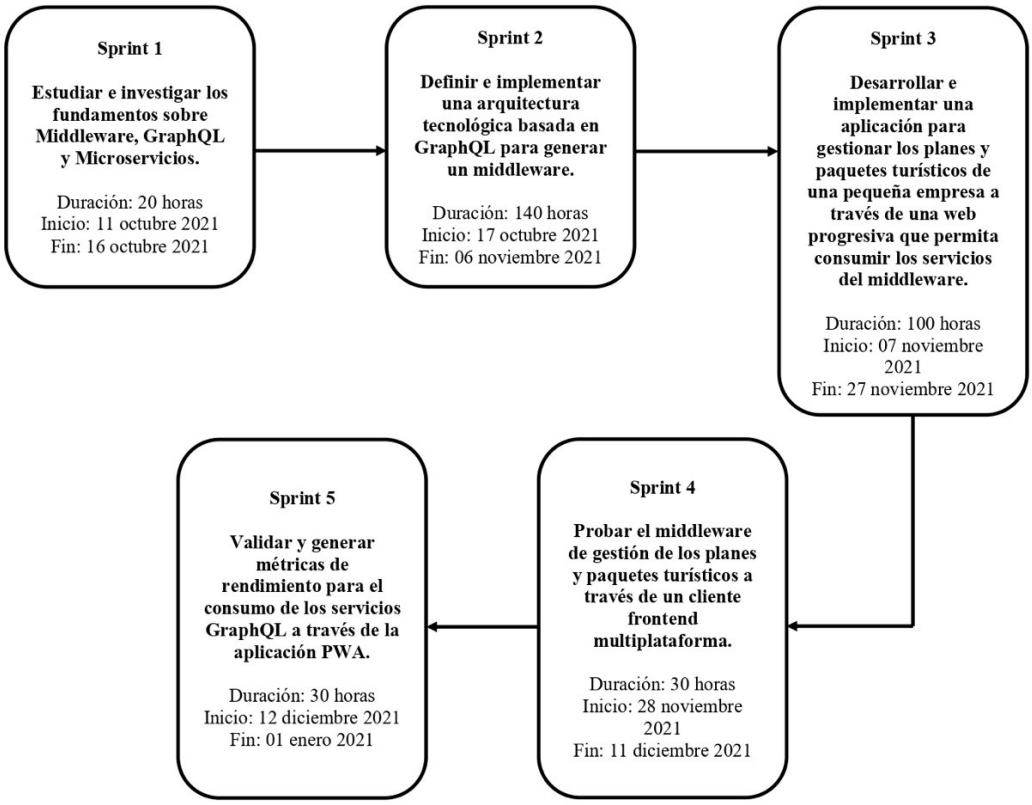


Figura 5. Sprints del proyecto

### 4.2.1. Primera fase

Consiste en estudiar los fundamentos y ventajas de las arquitecturas basadas en microservicios frente a otras arquitecturas, estudiar y dominar el lenguaje de consultas GraphQL, tener conceptos claros sobre los middlewares y sus categorías de integración con aplicaciones informáticas, e investigar y conocer tecnologías disruptivas que permitan el consumo de servicios.

Para esta fase, se recolectaron varios documentos para conocer y estudiar el estado del arte que se tiene actualmente referente a la problemática propuesta. Una vez que se estudiaron los documentos recopilados, se presenta un resumen de lo investigado, presentando el contenido mas relevante, con el fin de dar a conocer el estado del arte que se tiene al momento.

**Tabla 12. Estado del arte**

| Titulo del trabajo | Conclusión |
|---|---|
| A Survey of Middleware (Bishop & Karne, 2003). | En este trabajo el autor describe las ventajas que trae consigo un middleware, ente ellas proporcionan herramientas para mejorar la calidad del servicio, seguridad, los servicios de archivos, etc. Por otra parte, el autor reconoce que al haber muchas aplicaciones que corren bajo los middlewares han surgido varios tipos de estos, teniendo como los mas usados los middlewares orientados a procesos, middlewares orientados a objetos, middlewares para acceso a datos, middlewares basados en la web, etc. De esta forma al tener una cantidad grande de middlewares el autor concluye en que es importante clasificar los middlewares basándose en la integración de tecnologías que tendrá y el tipo de aplicación que brindará. |
| Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web (López & Maya, 2017). | La principal ventaja al utilizar una arquitectura basada en microservicios radica en la capacidad de publicar un sistema robusto como un conjunto de pequeñas aplicaciones que interactúan entre si y que se pueden desarrollar, desplegar y mantener de forma independiente obteniendo de esta manera una disminución en lo que refiere a costos, tiempos de desarrollo y despliegues, y una escalabilidad granular. |

| | |
|---|---|
| Semantics and Complexity of GraphQL (Hartig & Pérez, 2018) | En este trabajo los autores presentan los resultados y experiencia al realizar un estudio sistemático de GraphQL, formalizando la semántica este lenguaje de consulta. También presentan los puntos mas difíciles a los que hay que enfrentarse al momento de utilizar GraphQL, tales como problemas de evaluación, enumeración y cálculo del tamaño, mostrando que todos estos pueden resolverse de forma eficiente. |
| How fast GraphQL is compared to REST APIs (Oggier, 2020). | Aquí se realiza una comparativa de GraphQL frente a REST, realizando diferentes pruebas y analizando las métricas para determinar cual de estas tocologías brinda un mejor rendimiento frente a un escenario dado. El trabajo concluye en que GraphQL ofrece un tiempo de respuesta más frente a su oponente.<br>Después de analizar las métricas obtenidas en las diferentes pruebas el autor concluye que GraphQL es un 46% más rápido que REST para el escenario planteado. |
| React Native vs Flutter, cross-platform mobile application frameworks (Wu, 2018). | En este trabajo se realiza una comparativa de uno de los frameworks mas populares y robustos que existen en el mercado, React Native, frente a unos de los frameworks mas nuevos y que ha ido ganando popularidad, Flutter.<br>El autor después de presentar las ventajas de cada uno concluye en que la consistencia y limpieza a nivel de sintaxis y en el |

| | |
|---|---|
| | nivel del SDK, brinda un framework que resulta amigable con los desarrolladores. La eficiencia de Flutter deriva en que se puede aumentar considerablemente el tiempo de lanzamiento del producto al mercado; y finalmente reconoce que se da una pérdida de rendimiento no considerable frente a aplicaciones nativas. |

### 4.2.2. Segunda fase

Consiste en la generación e implementación de un middleware orientado a procesos, que estará basado en el lenguaje de consultas GraphQL, y permitirá publicar servicios autónomos. Este middleware será capaz de consumir e interactuar con información de diferentes fuentes, como servicios REST, bases de datos, lógicas de negocio ya implementadas, y consultas y funcionalidades propias de GraphQL, todo esto implementando métodos de autenticación. Con todas estas fuentes de información y servicios, el middleware será capaz de responder al cliente las solicitudes que requiera de una manera eficiente y centralizada.

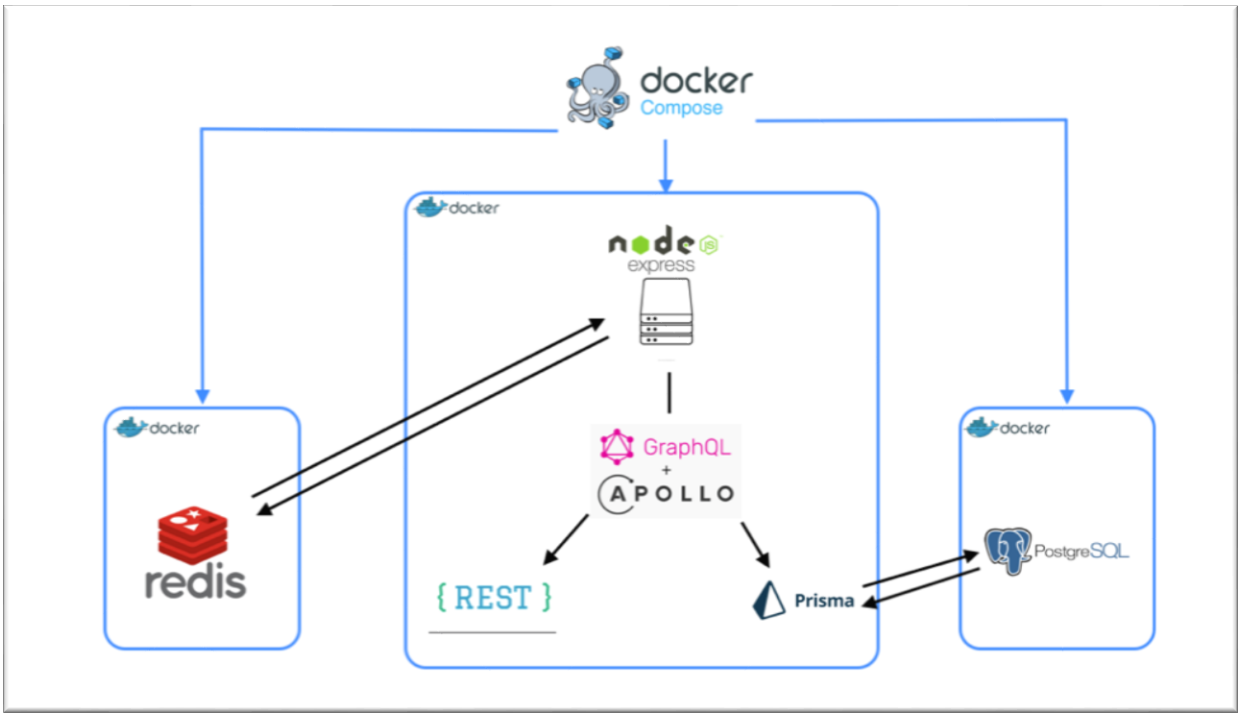


Figura 6. Tecnologías del middleware aplicado.

Como punto inicial, se plantea que los servicios que el middleware propuesto va a ofrecer deberán ser accedidos por medio de una API, en este caso bajo GraphQL, habilitando la comunicación con todos los servicios disponibles del

middleware a través de un único endpoint. Con esto se logra conectar varios servicios completamente distintos en una única capa.

Para la configuración y gestión de las peticiones del middleware se utiliza Apollo Server, un middleware disponible para implementar APIs de GraphQL con la ayuda de Express, Apollo Server habilitará al middleware a responder a las diferentes peticiones que se reciban mediante unas funciones conocidas como *resolvers*. De esta manera se obtendrá como resultado un servidor GraphQL.

Para acceder a este servidor de GraphQL se utiliza Express, debido a su escalabilidad, velocidad, rendimiento general, y debido a que Apollo Server se integra correctamente con este, posibilitando de esta manera un middleware basado en GraphQL.

Una vez que se tiene el servidor GraphQL, es necesario indicarle el esquema por el cual se regirá. Este esquema contendrá el tipo de datos en el que se manejará la información, y también las funciones que el cliente podrá usar para leer y escribir información del servidor GraphQL.

Con un esquema establecido, es momento de indicarle al servidor GraphQL de donde obtendrá la información. Apollo Server brinda la clase *DataSource* y su extensión *RestDataSource*, que habilitan al servidor a tener como fuentes de datos a bases de datos relacionales, bases de datos no relacionales, APIs REST, APIs GraphQL, o cualquier otra fuente que desempeñe el rol de fuente de datos. Cabe mencionar que Apollo Server no genera un cliente para bases de datos, ya sean estas relacionales o no relacionales. Para esto se utiliza Prisma, un tipo de ORM[2] que genera un cliente sobre la base de datos, dotando de esta manera, la habilidad de que el servidor GraphQL pueda leer y escribir información de una base de datos.

Sin embargo, el servidor no sabe como usar estas fuentes de datos para responder a las funciones definidas en el esquema. Es aquí donde entran los

[2] **ORM:** Modelo de programación que permite mapear las estructuras de una base de datos relacional sobre una estructura lógica de entidades con el objeto de simplificar y acelerar el desarrollo.

*resolvers*, funciones que se encargan de llenar los datos de un campo del esquema, es otras palabras, si el cliente requiere un campo en particular, esta función se encarga de buscarla en las fuentes de datos adecuada y devolverlo al cliente.

Una vez que el servidor GraphQL esta proveyendo la información requerida por el usuario, es momento de aplicar métodos de autenticación que servirán para validar la identidad del usuario; y métodos de autorización que ayudaran a determinar a que funciones y recursos tiene acceso el usuario dentro de la API.

Para el proceso de autenticación se implementa JWT, una vez que el usuario realice un inicio de sesión, que consta de ingresar correo electrónico y contraseña, el middleware realizará una llamada a un *resolver* de tipo *query*, ya que no es necesario realizar acciones de escritura, una vez validado el inicio de sesión, el servidor responderá con un token generado, este token contendrá información básica del usuario y que no compromete a su información personal. Este token servirá a manera de credencial dentro del servidor para identificar al usuario que esta realizando operaciones con los diferentes *resolvers*. El token generado contará con 24 horas de validez, una vez superado el tiempo establecido, el token quedará sin validez alguna para cualquier función que se quiera llevar a cabo.

Para el proceso de autorización se utiliza GraphQL Shield, una extensión que provee métodos para agregar reglas y permisos al esquema del servidor GraphQL. Por ejemplo, se desea que únicamente los usuarios que están autenticados dentro del servidor puedan consultar los paquetes turísticos que existen. Para esto primero se define la regla para identificar a los usuarios autenticados (figura 6), esto se logra comprobando si el token del usuario es válido.

**Tabla 13. GraphQL Shield - Regla**

```
const isAuthenticated = rule()(async (_, __, {token}) => {
    let info: any = new JWT().verify(token);
    if (info === 'Por favor, inicie session nuevamente.') {
        return false;
    }
    console.log('Usuario autenticado');
    return;
});
```

Y posteriormente se le indica al esquema que uno de sus *resolvers*, deberá cumplir con esta regla para poder ser llamado.

**Tabla 14. GraphQL Shield - Permisos de los resolvers.**

```
const permissions = shield({
    Query: {
        verPaquetesTuristicos: isAuthenticated,
    }
});
```

Una vez que se cuenta con un servidor GraphQL funcional y con los procesos para salvaguardar la integridad de los usuarios, es momento de encapsular el servidor dentro de un contenedor Docker. Este paso se implementa con el fin de que el middleware utilice los recursos necesarios del servidor en donde estará alojado. Para esto se define el fichero Dockerfile, en el cual se define que el contenedor utilizará la imagen de Node y copiará todos los ficheros definidos para la implementación del servidor GraphQL dentro del contenedor, para instalarlo y ejecutarlo a través del puerto definido.

**Tabla 15. Dockerfile**

```
FROM node:14
#Crear directorio de la app
WORKDIR /usr/src/app
#Instalar dependencias
COPY package*.json ./
COPY .env ./
COPY prisma ./prisma/
RUN npm install
COPY . .
EXPOSE 4000
CMD ["npm", "start"]
```

Finalmente, usando los servicios de Google Cloud Platform, se creó una instancia de una maquina virtual, para montar el servidor donde estará alojado el middleware implementado. Las principales características de la instancia de máquina virtual son:

- 2 CPU virtuales
- 4 GB de memoria RAM.
- S.O. Ubuntu 18.04 LTS
- 10 GB de almacenamiento SSD
- Región donde se encuentra la instancia, us-west4 (Las Vegas)

Con la instancia de maquina virtual lista, se realiza la instalación de la aplicación Docker para el sistema operativo Ubuntu. Con Docker se puede descargar la imagen del middleware creada anteriormente, para crear un nuevo contenedor con todas las configuraciones realizadas. De esta manera, se tiene como resultado el middleware montado en un servidor con una IP pública que provee Google Cloud Platform, posibilitando el acceso a este middleware desde cualquier lugar.

### 4.2.3. Tercera fase

Consiste en la creación de una aplicación progresiva, que es una aplicación basada en la web tradicional pero que incorpora varias características que la hacen parecer a una aplicación nativa para teléfonos móviles, la cual será desarrollada con tecnologías disruptivas y consumirá los servicios generados por el middleware para poder realizar el proceso de reserva de planes y paquetes turísticos.

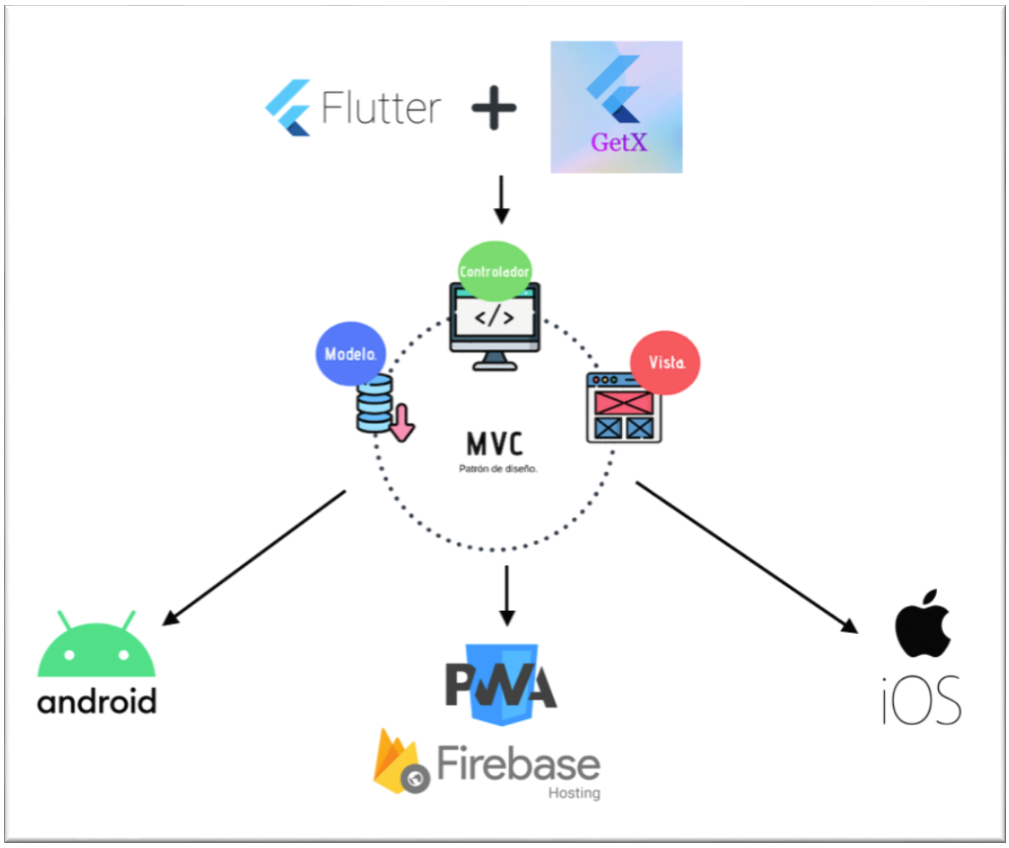


Figura 7. Tecnologías Frontend

El framework que se seleccionó para realizar la construcción de la aplicación progresiva fue Flutter, debido a que, a partir del mismo código, se pueden generar aplicaciones para los diferentes sistemas operativos, y a diferencia de otros frameworks, posibilita también la generación de código para aplicaciones web, incluyendo el soporte para aplicaciones progresivas.

Flutter maneja un concepto primordial para poder realizar el desarrollo de aplicaciones con este framework, el manejador de estados, que se la puede definir como el mecanismo con el que Flutter va a controlar todas las variables, tipos y widgets declarados, de esta manera Flutter puede controlar de manera global las actualizaciones de la UI[3] y todos los datos que la aplicación esta manejando.

Para la gestión de estados y enrutamiento de la aplicación, se utilizó un micro framework del ecosistema de Flutter, GetX, un paquete que como dice en su documentación (Get | Flutter Package, s. f.) "combina un administrador de estados de alto rendimiento, inyección inteligente de dependencias y gestión de rutas de forma rápida y práctica".

El patrón de diseño que se siguió para la aplicación fue MVC (Modelo-Vista-Controlador), con la aplicación de este patrón se tiene como resultado la separación de tres componentes dentro del proyecto, permitiendo tener una mejor gestión del proyecto, y realizar cambios sin afectar a otros componentes. Este patrón se adapta al gestor de estados GetX, ya que GetX actuará sobre los componentes vista y controlador, dotándolos de características para que se mantengan a la espera de cambios en y de esta manera se puedan actualizar datos que maneja la aplicación e interfaces de la aplicación.

La aplicación realizada en Flutter será el principal cliente del middleware GraphQL creado. Para que la aplicación sea capaz de realizar *querys*, *mutations y subscriptions* al servidor GraphQL, es necesario instalar la dependencia *graphql_flutter*, esta librería habilitará a la aplicación a crear una conexión hacia el servidor GraphQL y también proporcionará a la aplicación widgets ya diseñados para hacer uso de las diferentes acciones que nos brinda GraphQL.

Para la puesta en producción de la aplicación Flutter, se utilizó Firebase Hosting, un servicio de alojamiento que pertenece a Google y que proporciona contenido web estático como HTML, JavaScript, CSS, etc., de forma fácil, rápida y segura.

[3] **UI:** (User Interface), significa Interfaz de Usuario. Consiste de todos los elementos visuales que permiten al usuario interactuar con un sistema.

A continuación, se pueden visualizar algunas de las interfaces que se desarrollaron para la aplicación Flutter:

- **Iniciar sesión. –** Esta interfaz permite al usuario introducir sus credenciales para ingresar al sistema. También, dentro de esta interfaz se permite el inicio de sesión mediante las redes sociales Google y Facebook. Adicionalmente, se cuenta con un botón que lleva a la interfaz de registro, en el caso que no se tenga una cuenta para usar el sistema.

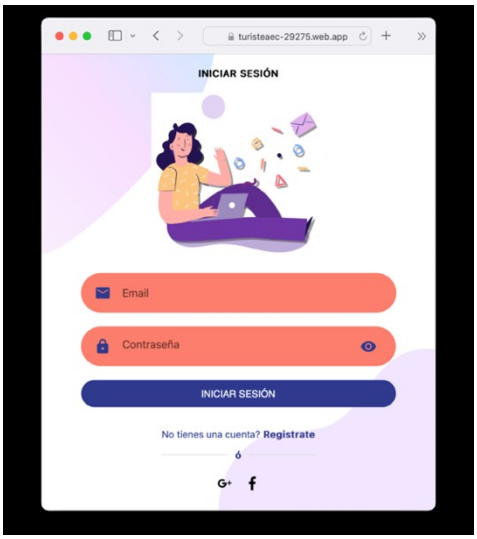


Figura 8. Interfaz para inicio de sesión

- **Registro. –** En esta interfaz se solicitan los datos personales del usuario, como nombres completos, teléfono, correo electrónico, contraseña y país de residencia. El formulario implementado cuenta con las validaciones respectivas. Adicionalmente, se cuenta con un botón que lleva a la interfaz de inicio de sesión, en el caso que ya se tenga una cuenta registrada para usar el sistema.

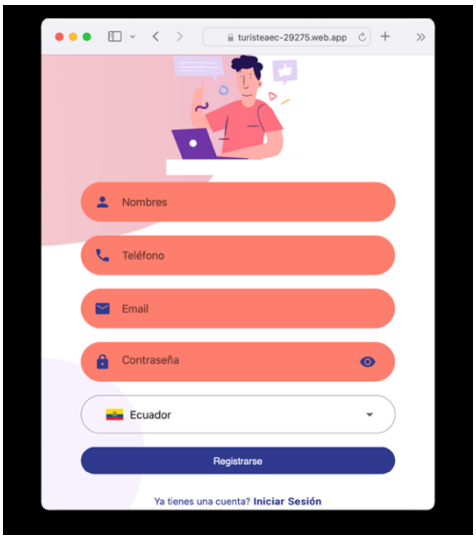


Figura 9. Interfaz para registro

- **Paquetes turísticos. –** En esta interfaz se muestran las diferentes categorías de paquetes turísticos. Dentro de cada categoría se pueden encontrar los diferentes paquetes turísticos que se están ofertando. Adicionalmente se tiene un botón que muestra el carrito donde se pueden agregar los paquetes turísticos que el usuario desea reservar.

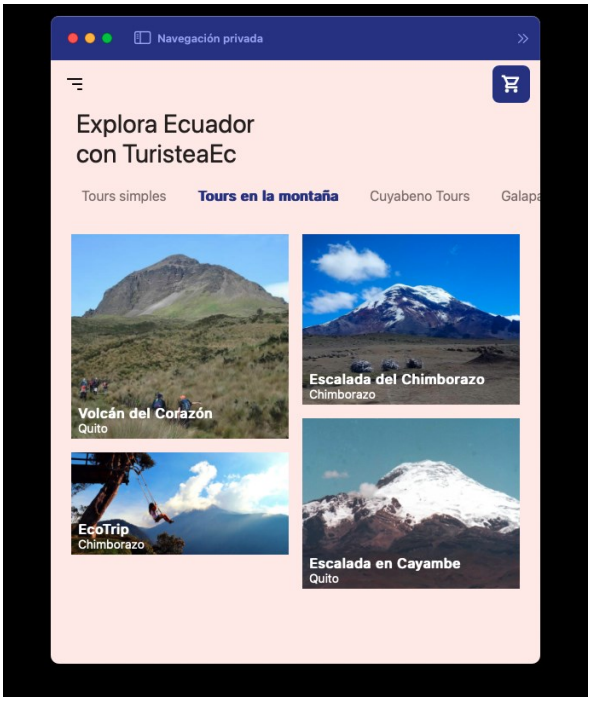


Figura 10. Interfaz de Paquetes Turísticos

- **Paquete turístico. –** En esta interfaz se muestra toda la información sobre el paquete turístico. En esta interfaz se incluyen botones para agregar el paquete turístico al carrito de compras.

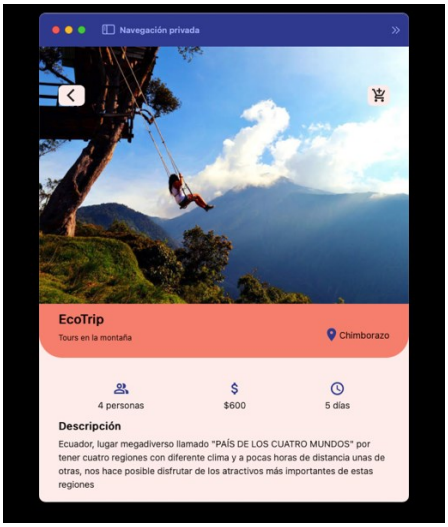


Figura 11. Interfaz detalle paquete turístico

- **Perfil. –** En esta interfaz se presenta la información acerca del usuario, desde aquí el usuario puede modificar su imagen de perfil, mediante una imagen que se tenga en la galería del dispositivo. También, en el caso que la cuenta del usuario no se encuentre verificada, se encuentra un botón que mostrará un dialogo donde se podrá ingresar el código de verificación.

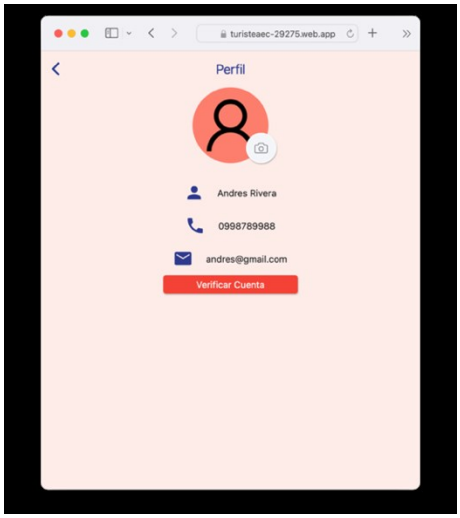


Figura 12. Interfaz de perfil del usuario

### 4.3. Arquitectura

Es el proceso que comprende en construir un diseño de la arquitectura, que tiene como principal objetivo la descomposición o modularización de un sistema en diferentes componentes, definiendo relaciones entre estos componentes para cumplir con los requisitos funcionales y no funcionales establecidos en el proyecto.

En la siguiente imagen se muestra la arquitectura a implementar, seccionando en los diferentes módulos que serán los microservicios, con las funciones que cumplirán dentro de la arquitectura y de que manera se interconectarán con el resto de los microservicios.

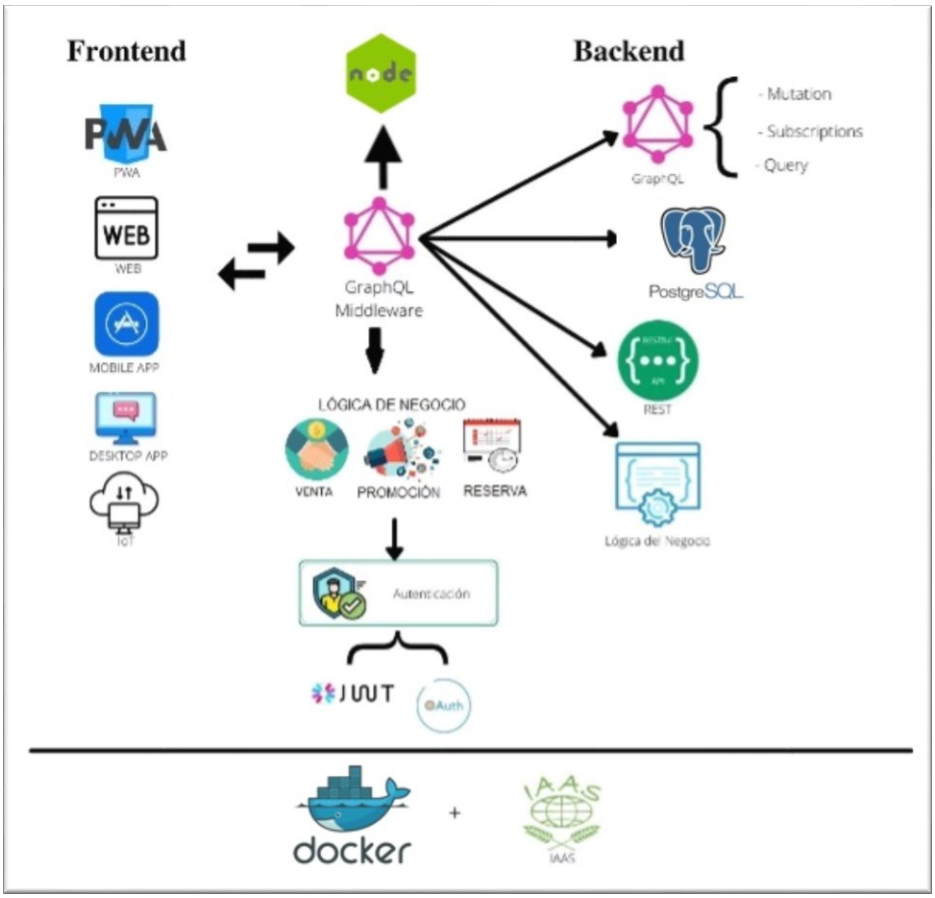


Figura 13. Arquitectura del proyecto

Seguidamente, se presenta más a fondo y con mejor detalle las arquitecturas utilizadas dentro de los dos grandes componentes que tendrá el sistema propuesto.

### 4.3.1. Backend

Cada vez que un usuario acceda a la aplicación realizada en Flutter, ya sea web o móvil, la aplicación debe conectarse con el servidor GraphQL para solicitar los datos que necesite.

Estos datos no se encuentran como tal en el servidor GraphQL, sino se encuentran en diferentes fuentes. Para el middleware desarrollado, estas fuentes son una base de datos Postgres y APIs REST.

Para que la aplicación Flutter pueda consumir estas fuentes de datos, en el servidor GraphQL se utiliza Express, que ayudara a gestionar las peticiones que provengan de la aplicación Flutter, obtener la información necesaria de las fuentes de datos y devolver dicha información de vuelta a la aplicación Flutter.

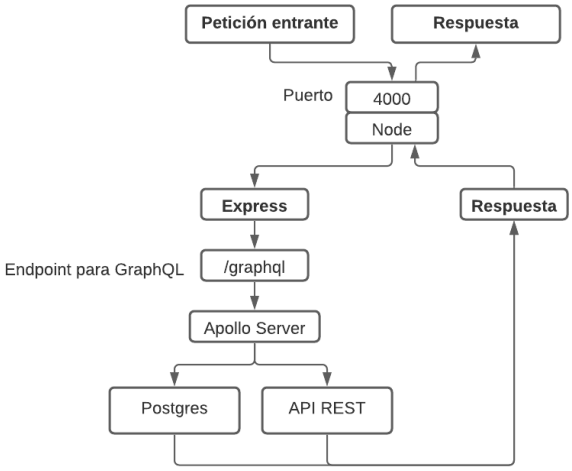


Figura 14. Arquitectura Backend

En la figura 13, se presenta la arquitectura implementada para el servidor GraphQL. El servidor va a estar a la espera de las peticiones que ingresen por el puerto definido en la aplicación Node, una vez que se tenga una petición entrante Express usará el endpoint de GraphQL, en donde Apollo llamará al resolver solicitado, obtendrá la información de las fuentes de datos definidas y enviará la respuesta a la aplicación Flutter.

### 4.3.2. Frontend

Para la construcción de la aplicación web, se utilizó la arquitectura MVC (Modelo-Vista-Controlador), debido a que, con este patrón de diseño de arquitectura de software, se permite una mejor separación de conceptos dentro del proyecto, de esta manera es mucho más fácil el manejo del proyecto y agregar cambios a cualquiera de los tres componentes sin afectar al resto de componentes, esto para que el desarrollo este estructurado de una mejor manera. Esta arquitectura consiste en la separación de los datos (Modelo), interfaces (Vista) y lógica del negocio (Controlador).

- **Modelo.** Representa los datos que se manejan dentro de la aplicación y dotan de información al usuario o a la aplicación misma. Dentro de la aplicación implementada, están representados por los tipos de datos que fueron definidos en el esquema del servidor GraphQL.
- **Vista.** En este componente constan todas las partes graficas del proyecto, las mismas que permiten la iteración del usuario con la aplicación. Dentro de la aplicación Flutter, entran todos los widgets definidos para el ingreso y salida de datos.
- **Controlador.** Este componente actúa como mediador entre los componentes de modelo y vista, ya que está al pendiente de las solicitudes que el usuario realiza mediante la vista, y según la lógica programada en este componente, realiza acciones sobre el modelo, retornando alguna respuesta por el componente de la vista.

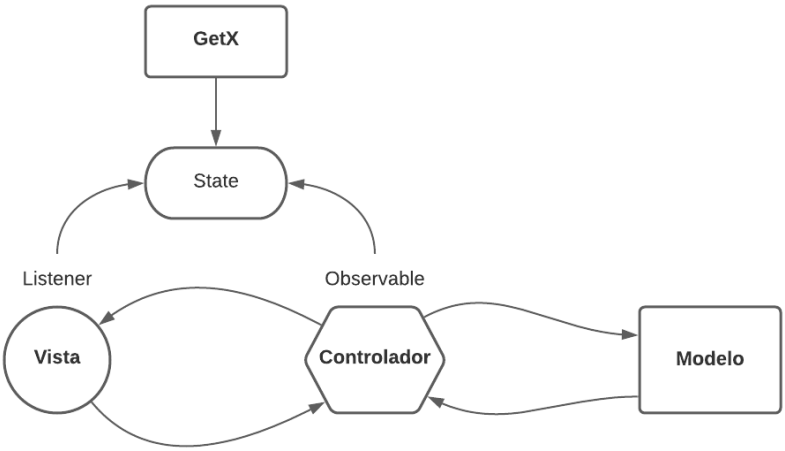


Figura 15. Arquitectura Frontend

Adicionalmente, dentro de esta arquitectura se incluye a GetX, el manejador de estados usado dentro de la aplicación Flutter. Con esta incorporación dentro de la arquitectura, el componente controlador incluye la característica de ser un observable, es decir, datos o valores futuros que se esperan recibir en algún punto del ciclo de vida de la aplicación o cuando sean accionados por una llamada.

A su vez, el componente de la vista adhiere la característica de actuar como oyente o listener, esta característica habilita a los componentes de la vista a estar pendientes de los cambios o nuevos datos que se manejen en el componente controlador.

Siguiendo el patrón de diseño Modelo-Vista-Controlador se estructuro el proyecto bajo los siguientes directorios

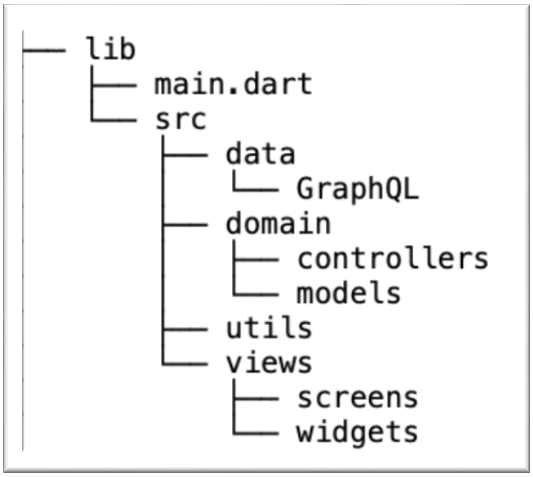


Figura 16. Estructura del proyecto

# 5. Resultados

## 5.1. Pruebas de carga

Las pruebas de carga se tratan de generar altos volúmenes de actividad dentro de un sistema para verificar como reacciona frente a múltiples solicitudes concurrentes, generalmente estas pruebas se centran en medir y validar la velocidad de respuestas del sistema durante interacciones concurrentes. Comúnmente estas pruebas deben realizarse cuando se tiene previsto que accedan al sistema una gran cantidad de usuarios al mismo tiempo.

Para realizar las pruebas de carga al middleware basado en GraphQL, se utilizó la herramienta JMeter, un software que permite automatizar y simular múltiples solicitudes.

Las pruebas que se realizaron se centraron en invocar a los diferentes tipos de resolvers, en este caso una *query* y *mutation*, para verificar el funcionamiento bajo alta demanda de estos resolvers. Se realizó la configuración de la herramienta JMeter para realizar las pruebas a estos resolvers, simulando 50, 100, 1000, 1500 y 2000 solicitudes de usuarios de manera concurrente.

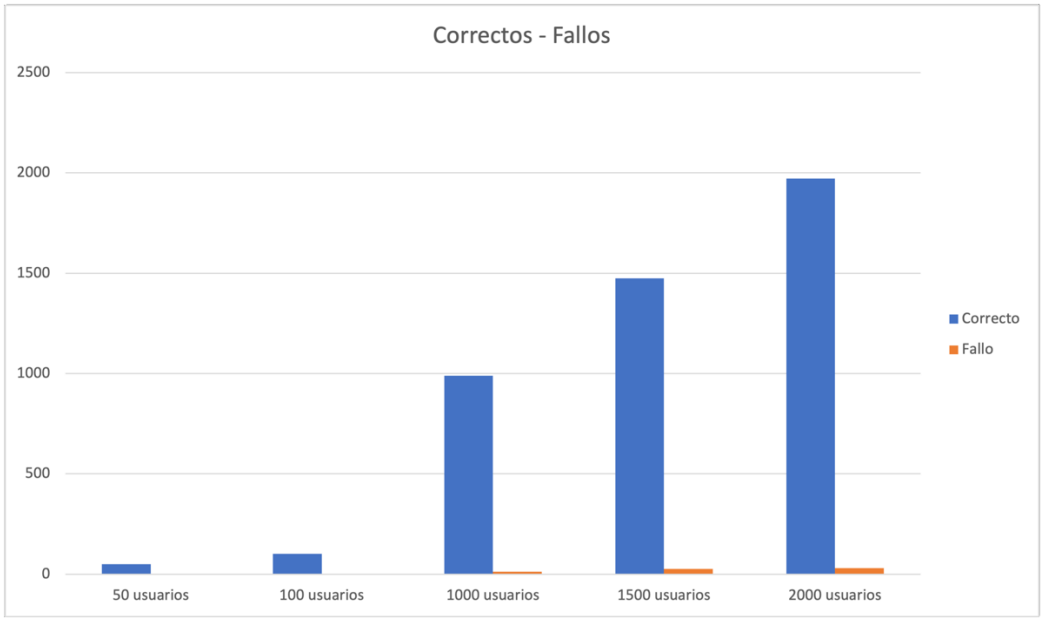


Figura 17. Pruebas de carga

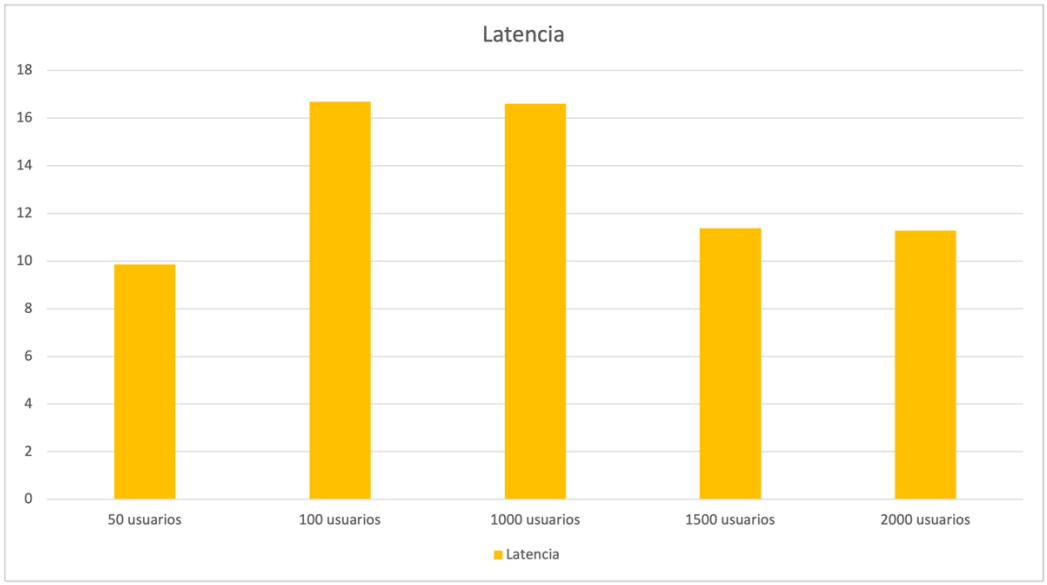


Figura 18. Pruebas de carga - Latencia

En las figuras 17 y 18, se aprecia que incluye el número de solicitudes que se realizaron correctamente y que no se realizaron, y también se presenta los resultados de

la latencia presentada en segundos (s) cuando se tiene diferentes números de usuarios realizando peticiones concurrentemente.

Con estos resultados se puede determinar que el middleware presenta una gran tolerancia a fallos ya que, la mayoría de las solicitudes fueron realizadas correctamente. En la siguiente tabla se muestra los datos obtenidos de las diferentes pruebas, incluyendo el numero de aciertos y tiempos mínimos y máximos de respuesta en milisegundos (ms).

**Tabla 16. Resultados pruebas de carga**

| Usuarios | Correcto | % Correctos | Fallos | % Fallos | Tiempo mínimo (ms) | Tiempo máximo (ms) |
|---|---|---|---|---|---|---|
| 50 | 50 | 100 | 0 | 0 | 123 | 1449 |
| 100 | 100 | 100 | 0 | 0 | 123 | 2588 |
| 1000 | 988 | 98.85 | 12 | 1.15 | 2463 | 105850 |
| 1500 | 1474 | 98.27 | 26 | 1.73 | 216 | 204070 |
| 2000 | 1972 | 98.5 | 28 | 1.5 | 358 | 312123 |

## 5.2. Pruebas unitarias

Las pruebas unitarias son pruebas automatizadas, que tiene como principal objetivo verificar el correcto funcionamiento de una unidad de código. Dentro de estas unidades de código se incluyen las partes mas pequeñas de una aplicación, como por ejemplo una función o un método.

La principal ventaja de realizar pruebas unitarias es que se puede seccionar la aplicación en varias unidades, permitiendo encontrar problemas en el código de una manera mas fácil y realizar modificaciones sin alterar otras partes o unidades de la aplicación.

Para ejemplificar esta prueba, se presentan los resultados de los métodos para iniciar sesión dentro de la aplicación, en donde se realizaron dos pruebas, el primero con credenciales correctas y el segundo con credenciales incorrectas; y el método para crear paquetes turísticos en donde se realizaron pruebas enviando un objeto válido y uno no

válido como se visualiza en las figuras 19 y 20 respectivamente, corroborando de esta manera la correcta funcionalidad de los métodos implementados.

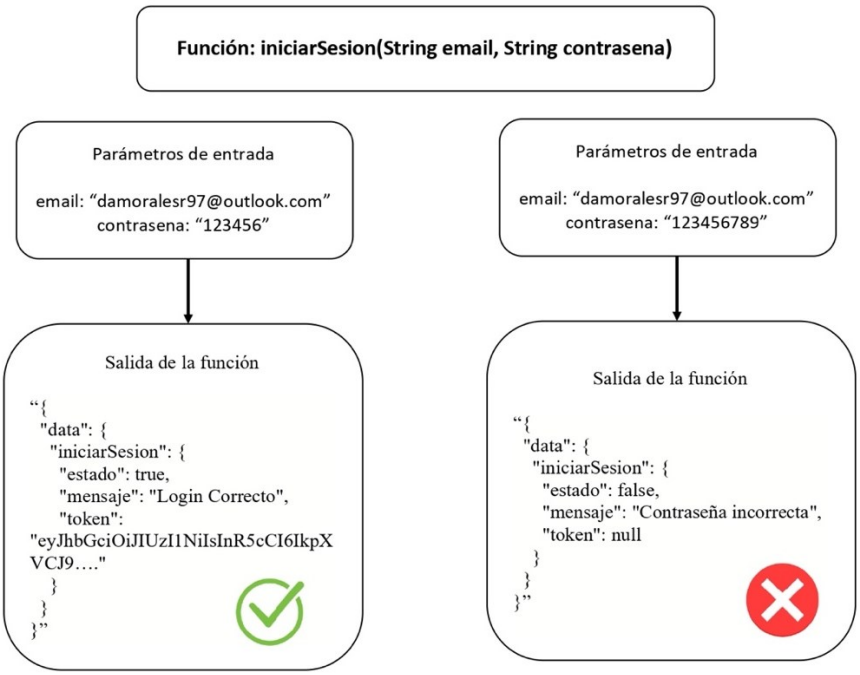


Figura 19. Pruebas unitarias - Iniciar Sesión

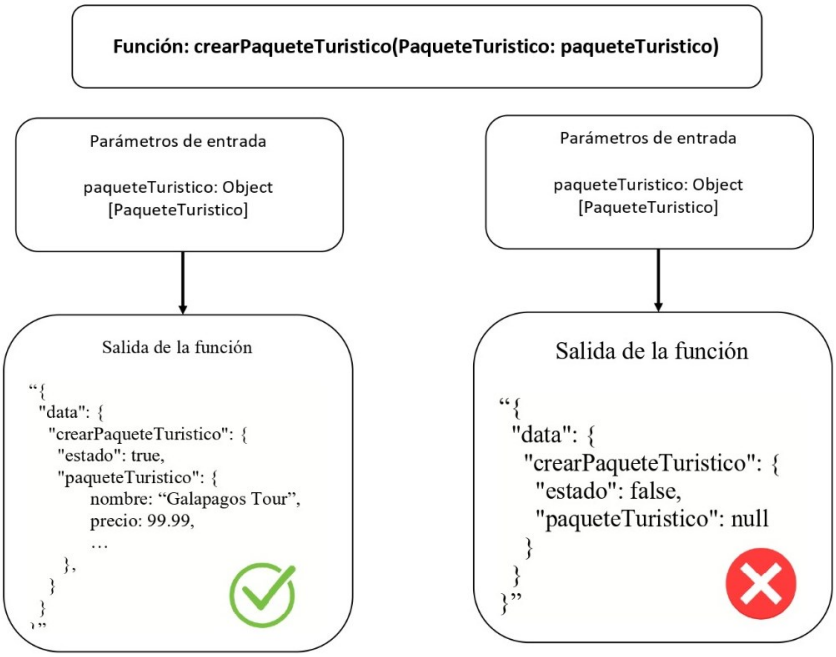


Figura 20. Pruebas unitarias - Crear Paquete Turístico

# Conclusiones

Estudiando acerca de los conceptos básicos para comenzar el desarrollo con GraphQL, se obtuvo una gran cantidad de información en sitios oficiales para poder introducirse a esta tecnología, con esta documentación se pudo abstraer los conceptos mas importantes para poder familiarizarse de manera rápida y con bases sólidas en esta tecnología. A su vez, en la documentación oficial de Apollo GraphQL se encontró ejemplos prácticos para montar un servidor basado en GraphQL, cubriendo secciones desde la creación del servidor, hasta despliegues en ambientes de producción.
El middleware implementado, se basó en los ejemplos propuestos por Apollo GraphQL, con este servidor GraphQL montado, se agregaron componentes como diferentes fuentes de datos, módulos para autenticación y autorización y módulos para manejar la lógica del negocio, con todos estos componentes se logró obtener una arquitectura robusta y de esta forma un middleware que cubre los requerimientos funcionales y no funcionales, en base a ello se tiene que las tecnologías seleccionadas para este trabajo presentan una gran abstracción y adherencia para trabajar en conjunto con otras tecnologías.
En base a las investigaciones sobre los diferentes frameworks del mercado, se seleccionó a Flutter debido a que presenta una curva de aprendizaje baja frente a sus competidores y que inicialmente genera aplicaciones para los sistemas operativos iOS y Android, incluyendo aquí el soporte para una aplicación web progresiva; ya que Flutter es mantenida por una comunidad grande, se encontraron extensiones que facilitan la conexión entre la aplicación y el servidor GraphQL, de esta manera se logro crear una aplicación que actúa como cliente del servidor GraphQL y cumple con las acciones básicas para realizar lecturas, registros, actualizaciones, eliminación y reservas de paquetes turísticos, así como el mantenimiento de un perfil de usuario.
Por otro lado, el cliente consumió satisfactoriamente los servicios generados por el middleware GraphQL, se realizaron pruebas de carga y pruebas unitarias. Con esto, se evidenció bajo métricas el rendimiento del middleware. Cabe mencionar que estas pruebas se realizaron con el middleware GraphQL montado en un ambiente de producción y con la aplicación Flutter corriendo sobre un hosting publicado en internet, en donde los resultados obtenidos fueron sobresalientes ya que al momento que se genero mayor trafico concurrente en el servidor, este continúo funcionando sin presentar perdidas de servicio significativos, ya que en los resultados obtenidos en las pruebas de

carga, se obtuvo como máximo porcentaje de fallas el 1.73% de las solicitudes realizadas, como mínimo tiempo de respuesta 123 milisegundos (ms) y como máximo tiempo de respuesta 312123 milisegundos (ms).

# Recomendaciones

Se evidenció que a pesar de que existe una gran cantidad de documentación para la implementación de las tecnologías usadas, se carece de una documentación detallada para el consumo de una API GraphQL usando Flutter.

Si bien Flutter presenta una fácil curva de aprendizaje para generar aplicaciones, después de realizar la aplicación se recomienda fuertemente especializarse en el desarrollo de interfaces graficas amigables y vistosas, ya que este puede llegar a ser el punto mas complejo dentro del framework.

En el lado del servidor GraphQL, al momento de desplegarlo en un ambiente de producción, se recomienda configurarlo con certificados SSL, caso contrario, las solicitudes pueden no realizarse, presentando problemas de comunicación entre el cliente y el servidor.

# Trabajos futuros

En el presente trabajo se implementó un middleware basado en GraphQL, que tiene como principales fuentes de datos APIs REST y bases de datos; adicionalmente para este middleware se utilizó como motor para el servidor GraphQL a Apollo Server: y finalmente se desplegó la aplicación haciendo uso de microservicios. Dicho esto, surgen algunas líneas que incluyen la utilización de diferentes tecnologías que podrían mejorar el rendimiento del middleware basado en GraphQL.

A continuación, se mencionan posibles trabajos futuros:

- Diseño e implementación de un middleware basado en GraphQL utilizando como motor GraphQL a Hasura.
- Implementación como fuente de datos para el servidor GraphQL a bases de datos Oracle.
- Diseño de una arquitectura para la integración y despliegue del middleware GraphQL mediante Kubernetes.
- Agregar integración continua (CI) y despliegue continuo (CD).

# Referencias bibliográficas


(1) Walsh, J., & Godfrey, S. (2000). The Internet: a new era in customer service. European Management Journal, 18(1), 85-92.

(2) Bhatti, A., Akram, H., Basit, H. M., Khan, A. U., Raza, S. M., & Naqvi, M. B. (2020). E-commerce trends during COVID-19 Pandemic. *International Journal of Future Generation Communication and Networking*, *13*(2), 1449-1452.

(3) Alfonso, V., Boar, C., Frost, J., Gambacorta, L., & Liu, J. (2021). E-commerce in the pandemic and beyond. *BIS Bulletin*, *36*(9).

(4) Jílková, P., & Králová, P. (2021). Digital Consumer Behaviour and eCommerce Trends during the COVID-19 Crisis. *International Advances in Economic Research*, 1-3.

(5) Bishop, T. A., & Karne, R. K. (2003, March). A Survey of Middleware. In Computers and Their Applications (pp. 254-258).

(6) López, D., & Maya, E. (2017). Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web.

(7) L. Qilin and Z. Mintian, "The State of the Art in Middleware," 2010 International Forum on Information Technology and Applications, Kunming, China, 2010, pp. 83-85, doi: 10.1109/IFITA.2010.118.

(8) Hartig, O., & Pérez, J. (2018, April). Semantics and complexity of GraphQL. In *Proceedings of the 2018 World Wide Web Conference* (pp. 1155-1164).

(9) Oggier, C. (2020). How fast GraphQL is compared to REST APIs.

(10) Wu, W. (2018). React Native vs Flutter, Cross-platforms mobile application frameworks.

(11) G. Brito and M. T. Valente, "REST vs GraphQL: A Controlled Experiment," 2020 IEEE International Conference on Software Architecture (ICSA), Salvador, Brazil, 2020, pp. 81-91, doi: 10.1109/ICSA47634.2020.00016.

(12) Carlson, J. (2013). *Redis in action*. Simon and Schuster.

(13) Windmill, E. (2020). *Flutter in action*. Simon and Schuster.

(14) Facebook Open Source. (s.f). *GraphQL*. Recuperado el enero de 2022, de GraphQL: https://graphql.org/

(15) Apollo GraphQL. (s.f). What is Apollo Server and what does it do? Recuperado el enero de 2022, de Apollo Docs: https://www.apollographql.com/docs/apollo-server/

(16) Prisma. (s.f). *Prisma Documentation.* Recuperado el enero de 2022, de Prisma Docs: https://www.prisma.io/docs

(17) Hayat, F., Rehman, A. U., Arif, K. S., Wahab, K., & Abbas, M. (2019). The Influence of Agile Methodology (Scrum) on Software Project Management. *Proceedings - 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2019*, 145–149. https://doi.org/10.1109/SNPD.2019.8935813

(18) Mahalakshmi, M., & Sundararajan, M. (2013). Traditional SDLC vs scrum methodology–a comparative study. *International Journal of Emerging Technology and Advanced Engineering*, *3*(6), 192-196.

(19) *get | Flutter Package*. (s. f.). Dart Packages. Recuperado enero de 2022, de https://pub.dev/packages/get