TALLER #2

ANDRES ARMANDO SANCHEZ MARTIN

ANDRES FELIPE DUARTE LEAL
JUAN MANUEL AGUIAR OROZCO
HUMBERTO RUEDA CATAÑO

FACULTAD DE INGENIERÍA

ARQUITECTURA DE SOFTWARE

PONTIFICIA UNIVERSIDAD JAVERIANA

BOGOTÁ D.C.

2024

I. INTRODUCCIÓN

El presente documento se los lineamientos y directrices para el desarrollo básico de una landing page para un portafolio. Este documento también tiene como objetivo crear una plataforma web no tan robusta, que sea ágil y de fácil navegación para los usuarios, priorizando la presentación clara y atractiva de los contenidos del portafolio. Además, se busca que la landing page sea altamente adaptable y compatible con diferentes dispositivos y navegadores, garantizando así una experiencia óptima para todos los visitantes. Además, establece una breve introducción a las tecnologías utilizadas en este taller como lo son, Golang, Grpc, MariaDB y Preact.

II. DEFINICIONES, HISTORIA Y EVOLUCIÓN

- ***Grpc:*** gRPC es un moderno marco de trabajo de llamada a procedimiento remoto (RPC), de código abierto y alto rendimiento, que puede ejecutarse en cualquier entorno. Puede conectar de manera eficiente servicios dentro y entre centros de datos con soporte plug-and-play para equilibrio de carga, trazabilidad, verificación de salud y autenticación. También es aplicable en el último tramo de la computación distribuida para conectar dispositivos, aplicaciones móviles y navegadores con servicios backend.

  gRPC fue creado inicialmente por Google, que ha utilizado una infraestructura de RPC general llamada Stubby para conectar la gran cantidad de microservicios en funcionamiento dentro y entre sus centros de datos durante más de una década. En marzo de 2015, Google decidió construir la próxima versión de Stubby y hacerla de código abierto. El resultado fue gRPC, que ahora es utilizado en muchas organizaciones fuera de Google para impulsar casos de uso desde microservicios hasta el "último tramo" de la computación (móvil, web e Internet de las cosas).

  Inicios:

  - 2005: Nace el concepto de microservicios, impulsando la necesidad de un sistema de comunicación eficiente.
  - 2011: Se consolida el término "microservicios" y su uso comienza a crecer en la industria.
  - 2015: Google crea gRPC como respuesta a la necesidad de un marco RPC eficiente para la comunicación entre microservicios.

Crecimiento y adopción:

- 2016: gRPC se lanza como código abierto, impulsando su adopción por parte de la comunidad de desarrolladores.
- 2017: Se crea la Fundación gRPC para promover la colaboración y el desarrollo del proyecto.
- 2018: gRPC se integra con Kubernetes, facilitando su uso en entornos de contenedores.
- 2019: gRPC se convierte en uno de los marcos RPC más populares del mundo, con una amplia adopción en diversas industrias.

Consolidación e innovación:

- 2020: gRPC se consolida como un estándar de facto para la comunicación entre microservicios.
- 2021: Se lanza gRPC Web, permitiendo la comunicación entre clientes web y servidores gRPC.
- 2022: gRPC se utiliza en una amplia gama de aplicaciones, desde la nube hasta el Internet de las Cosas (IoT).

- ***Golang:*** Golang, también conocido como Go, es un lenguaje de programación de código abierto desarrollado por Google. Fue creado por Robert Griesemer, Rob Pike y Ken Thompson en 2007 y se dio a conocer en 2009. Go se caracteriza por su eficiencia, concurrencia y simplicidad. Está diseñado para ser rápido en la compilación, fácil de leer y escribir, y adecuado para desarrollar software a gran escala, especialmente en entornos de computación distribuida y en la nube. Go también incluye características como recolección de basura, tipos estáticos y un sistema de manejo de errores sencillo. Es ampliamente utilizado en diversas áreas de desarrollo de software, desde sistemas de backend y servidores web hasta aplicaciones de escritorio y dispositivos integrados.

- ***Preact:*** Preact es una librería JavaScript que se describe a sí mismo como una alternativa más rápida a React con la misma API ES6. Preact nació en 2015, desarrollado por Jason Miller principalmente. El objetivo del desarrollo de Preact era hacer un framework de JavaScript mucho más ligero que React pero manteniendo las mismas funcionalidades con las que trabaja y está relacionado React. Preact tiene un tamaño total de 3kb, es decir, toda su librería/framekwork de JavaScript no ocupa casi nada en el tamaño del proyecto. A lo largo de los años Preact ha mejorado adquiriendo nuevas funcionalidades, una de sus más

características es poder traer código de React y adaptarlo fácilmente a Preact siempre y cuando no se esté usando una librería muy específica de React.

- ***MariaDB:*** MariaDB nació cuando en 2009, Oracle compró MYSQL y el fundador de MYSQL, Michael "Monty" Widenius dividió el proyecto porque no estaba de acuerdo con la administración de Oracle, este nuevo proyecto fue llamado MariaDB. Su nombre proviene de la segunda hija de Michael, Maria. Muchos de los desarrolladores originales entraron al nuevo proyecto y ha sido mantenido desde ese momento. MariaDB en sus primeras versiones seguía los mismos números de versión de MySQL, siguiendo esta norma hasta la 5.5, pero apartir de 2012, para reflejar las diferentes funcionalidades diferentes que poseía MariaDB y que MySQL, cambió su numeración de versión de la 5.5 a la 10.0. Hoy en día, la última versión de MariaDB estable con soporte a largo plazo 10.6.

- ***Estilo arquitectónico de Capas:*** El estilo arquitectónico de capas, también conocido como arquitectura en capas, es un enfoque de diseño de software en el que un sistema se divide en capas lógicas o niveles de abstracción, donde cada capa tiene una responsabilidad específica y se comunica únicamente con las capas adyacentes. Las capas suelen estar organizadas de manera jerárquica, con capas más internas que proporcionan servicios más básicos y capas externas que brindan funcionalidad más específica o de usuario. Este estilo facilita la modularidad, la reutilización del código y la escalabilidad del sistema al promover una clara separación de preocupaciones y una estructura bien definida.

  El estilo arquitectónico de capas surge en la década de 1970 como una respuesta a la necesidad de modularizar y estructurar sistemas complejos. En ese momento, los sistemas de software se estaban volviendo cada vez más grandes y complejos, lo que dificultaba su desarrollo y mantenimiento. La arquitectura de capas proporcionó una forma de dividir estos sistemas en partes más pequeñas y manejables.

  Evolución y adopción:

  En la década de 1980, se popularizaron los patrones de diseño como MVC (Model-View-Controller) y MVP (Model-View-Presenter), que se basan en la arquitectura de capas. Estos patrones proporcionaron una forma de

organizar el código dentro de las capas y definir las responsabilidades de cada capa.

En la década de 1990, la arquitectura de capas se convirtió en un enfoque fundamental para el desarrollo de software a gran escala. Esto se debe a que la arquitectura de capas ofrece una serie de beneficios, como modularidad, reutilización, escalabilidad y mantenimiento.

III. Casos de estudio

Golang:

Google utiliza Go para una gran variedad de aplicaciones, incluyendo Kubernetes, Docker y la infraestructura de Google Cloud Platform. Netflix emplea Go para su API de streaming, herramientas de análisis y otros sistemas internos. Uber implementa Go en su API de backend, sistema de gestión de flotas y otras aplicaciones críticas. Dropbox utiliza Go para su API de sincronización de archivos, infraestructura de almacenamiento y herramientas de DevOps. Twitter emplea Go para su API de streaming, sistema de procesamiento de tweets y herramientas de análisis. Algunos casos de uso específicos incluyen herramientas de línea de comandos como Docker, Terraform, Hugo y Kubernetes; APIs como Stripe, Twilio, Kong y Consul; microservicios como los de Netflix, Uber, Spotify y Lyft; aplicaciones web como las de New York Times, SoundCloud, Twitch y Heroku; y sistemas distribuidos como Google Cloud Platform, Kubernetes y Docker.

Preact:

Preact es empleado por varios sitios web, desde proyectos Open Source, hasta empresas multinacionales. Algunas de estas son: Domino's, Tencent QQ, Uber, The New York Times.

gRPC:

Netflix emplea gRPC: "En nuestro uso inicial de gRPC, pudimos extenderlo fácilmente para que viva dentro de nuestro ecosistema obstinado. Además, hemos tenido un gran éxito al realizar mejoras directamente en gRPC a través de solicitudes de extracción e interacciones con el equipo de Google que administra el proyecto. Esperamos ver muchas mejoras en la productividad de los desarrolladores y la capacidad de permitir el desarrollo en lenguajes que no sean JVM como resultado de la adopción de gRPC."

MariaDB:

Wikipedia, la enciclopedia online más grande del mundo, utiliza MariaDB para almacenar su vasta cantidad de datos. Google Cloud Platform ofrece MariaDB como una opción de base de datos para sus clientes. Red Hat, la empresa de software empresarial utiliza MariaDB en su plataforma OpenShift. Booking.com, el sitio web de reservas de viajes, emplea MariaDB para almacenar información sobre millones de hoteles y alojamientos. Verizon, la empresa de telecomunicaciones, utiliza MariaDB para almacenar datos de sus clientes.

En cuanto a casos de uso específicos, MariaDB se utiliza en sitios web y aplicaciones web para almacenar datos de usuarios, contenido, productos y pedidos. En aplicaciones de comercio electrónico, MariaDB se emplea para almacenar datos de productos, clientes, pedidos y pagos. En sistemas de gestión de contenidos (CMS), MariaDB se utiliza para almacenar contenido, usuarios y configuraciones. Para bases de datos de referencia, MariaDB se emplea para almacenar datos de referencia para aplicaciones. Finalmente, en aplicaciones móviles, MariaDB se utiliza para almacenar datos localmente en dispositivos móviles.

IV. Relación entre los temas

1. Arquitectura por Capas: Es un estilo de arquitectura de software que divide la aplicación en capas lógicas separadas. Cada capa tiene una responsabilidad específica dentro de la aplicación, como la presentación, la lógica de negocio, y el acceso a datos. Esta separación promueve la organización del código, la reusabilidad, y la facilidad para el mantenimiento y la escalabilidad.

2. Preact: Es una biblioteca de JavaScript para construir interfaces de usuario. Es similar a React, pero más ligera y con un enfoque en el rendimiento. Preact se utilizaría en la capa de presentación de una arquitectura por capas, permitiendo desarrollar interfaces de usuario ricas y dinámicas que interactúan con la lógica de negocio a través de llamadas a servicios web o APIs.

3. gRPC: Es un marco de comunicación remota de alto rendimiento que puede ser utilizado para conectar servicios en diferentes lenguajes de programación, incluyendo Go y otros. gRPC se basa en HTTP/2, lo que permite comunicaciones bidireccionales y eficientes. En una arquitectura por capas, gRPC podría ser

utilizado para la comunicación entre la capa de lógica de negocio y la capa de servicios, permitiendo una interacción eficiente y segura entre diferentes servicios y componentes de la aplicación.

4. Go (Golang): Es un lenguaje de programación compilado, concurrente, y con tipado estático diseñado por Google. Go es conocido por su eficiencia y simplicidad, y es comúnmente utilizado para desarrollar sistemas a gran escala, servicios web, y aplicaciones de microservicios. En una arquitectura por capas, Go podría ser utilizado para implementar la lógica de negocio y los servicios backend, aprovechando su rendimiento y facilidad para el manejo de concurrencia.

5. MariaDB: Es un sistema de gestión de bases de datos relacional, bifurcación de MySQL, que se utiliza para almacenar y gestionar los datos de la aplicación. En una arquitectura por capas, MariaDB se utilizaría en la capa de acceso a datos, proporcionando un almacenamiento persistente y eficiente para los datos que la aplicación necesita.

V. Situaciones y/o problemas donde se podrían aplicar los temas

1. Desarrollo de Aplicaciones Web de Alto Rendimiento

Situación: Necesidad de desarrollar una aplicación web interactiva y de alto rendimiento que maneje eficientemente grandes volúmenes de usuarios y datos en tiempo real.

Solución: Utilizar Preact en el frontend para aprovechar su ligereza y eficiencia, mejorando los tiempos de carga y la interactividad de la aplicación. Go puede ser utilizado en el backend para manejar la lógica de negocio y las operaciones concurrentes de manera eficiente, mientras que MariaDB almacena los datos. gRPC facilita la comunicación entre el frontend y el backend, así como entre diferentes servicios backend, con un protocolo de comunicación rápido y ligero.

2. Microservicios y Desacoplamiento de Aplicaciones

Situación: Modernización de una aplicación monolítica a una arquitectura de microservicios para mejorar la escalabilidad y la mantenibilidad.

Solución: Implementar microservicios utilizando Go, aprovechando su rendimiento y facilidad para el manejo de concurrencia. gRPC se utiliza para la comunicación entre microservicios debido a su eficiencia y soporte para múltiples lenguajes de programación, permitiendo un desacoplamiento efectivo y una interacción eficiente entre servicios. La arquitectura por capas asegura que cada microservicio se mantenga organizado y enfocado en una sola responsabilidad.

VI. Ventajas y desventajas

Arquitectura por Capas

Ventajas:

Separación de preocupaciones: Facilita el mantenimiento y la actualización de la aplicación al separar la lógica de negocio, la interfaz de usuario y el acceso a datos.

Reusabilidad: Las capas bien definidas permiten reutilizar componentes en diferentes partes de la aplicación o en proyectos futuros.

Flexibilidad: Facilita la sustitución o actualización de partes específicas del sistema sin afectar a otras.

Desventajas:

Complejidad: Puede introducir una complejidad adicional en la estructura del proyecto, especialmente en aplicaciones pequeñas donde podría no ser necesario.

Rendimiento: La comunicación entre capas puede aumentar la latencia, especialmente si no se maneja eficientemente.

Sobrecarga: La abstracción de cada capa puede llevar a una sobrecarga de desarrollo si no se implementa cuidadosamente.

Preact

Ventajas:

Rendimiento: Su pequeño tamaño (alrededor de 3kB) lo hace rápido de cargar y eficiente en el rendimiento, ideal para proyectos que necesitan ser ligeros.

Compatibilidad con React: Ofrece una API similar a React, lo que facilita la transición para los desarrolladores familiarizados con React.

Facilidad de uso: Proporciona una forma sencilla y eficiente de crear interfaces de usuario interactivas.

Desventajas:

Menos características: Al ser más pequeño y ligero que React, carece de algunas características avanzadas disponibles en React.

Menor comunidad: Aunque está creciendo, su comunidad es más pequeña que la de React, lo que puede resultar en menos recursos y soporte.

gRPC

Ventajas:

Alto rendimiento: Utiliza HTTP/2 para la comunicación, lo que mejora el rendimiento y la eficiencia en comparación con protocolos más antiguos.

Soporte para múltiples lenguajes: Funciona en varios lenguajes de programación, facilitando la integración en sistemas políglotas.

Comunicación bidireccional en tiempo real: Permite streaming bidireccional, ideal para aplicaciones que requieren comunicaciones en tiempo real.

Desventajas:

Curva de aprendizaje: Requiere una comprensión de los protocolos de comunicación y la definición de servicios, lo que puede ser complejo para los nuevos usuarios.

Soporte en navegadores: El soporte directo en navegadores es limitado, requiriendo soluciones alternativas para la comunicación cliente-servidor.

Go

Ventajas:

Concurrencia: Su modelo de concurrencia es una de las características más fuertes, facilitando la escritura de programas concurrentes.

Simplicidad: El lenguaje está diseñado para ser simple y eficiente, con una sintaxis clara y una herramienta de formato de código estándar.

Rendimiento: Compila directamente a código máquina, lo que significa que los programas son rápidos y eficientes.

Desventajas:

Gestión de dependencias: Aunque ha mejorado con la introducción de módulos, la gestión de dependencias ha sido un punto de crítica.

Genéricos: Hasta recientemente, la falta de genéricos fue vista como una limitación, aunque esto ha cambiado con las versiones más recientes de Go.

MariaDB

Compatibilidad con MySQL: Es un fork de MySQL, por lo que ofrece alta compatibilidad, facilitando la migración de bases de datos existentes.

Comunidad y soporte de código abierto: Tiene una comunidad activa y es completamente de código abierto.

Rendimiento y características: Ofrece características avanzadas y optimizaciones de rendimiento que superan a las versiones estándar de MySQL.

Desventajas:

Popularidad: Aunque es popular, algunas empresas prefieren soluciones de bases de datos más establecidas o con soporte comercial directo.

Curva de aprendizaje: Para usuarios nuevos en sistemas de gestión de bases de datos, puede haber una curva de aprendizaje para optimizar el rendimiento y la seguridad.

VII. ALCANCE

El proyecto tiene como objetivo principal diseñar, desarrollar e implementar una landing page para un portafolio, centrándose en dos módulos principales: uno destinado a facilitar el contacto con el creador del portafolio y otro diseñado para mostrar los proyectos realizados. La estructura de la landing page se basará en un patrón de arquitectura de capas, aprovechando tecnologías modernas como Preact para el Frontend, Grpc y Golang para el Backend. Además, se prevé la utilización de una base de datos relacional, integrando una base de datos MariaDB para una gestión eficiente de los datos.

VIII. REQUERIMIENTOS FUNCIONALES

1. Módulo de información personal y de contacto

Este módulo se centra en la presentación clara y accesible de la información personal y de contacto del aspirante, incluyendo nombre, correo electrónico, número de celular, ciudad y país. Permitirá a los visitantes del sitio comprender quién es el estudiante y cómo pueden ponerse en contacto con él.

Requerimientos Funcionales:

- Presentación de la información personal del aspirante, incluyendo nombre completo, ciudad y país.
- Presentación de la información de contacto, incluyendo correo electrónico y número de celular.
- Diseño responsivo para asegurar la accesibilidad y legibilidad en distintos dispositivos.

2. Módulo de proyectos de programación

Este módulo abarca la presentación y descripción detallada de los proyectos de programación realizados por el estudiante. Incluirá funcionalidades para agregar y visualizar proyectos, facilitando a los visitantes la comprensión de las habilidades y experiencias del estudiante en programación.

Requerimientos Funcionales:

- Visualización detallada de cada proyecto, incluyendo descripción, tecnologías utilizadas, y enlaces a repositorios o demos si están disponibles.
- Implementación de un diseño que permita la fácil exploración de los proyectos por parte de los visitantes.

IX. *REQUERIMIENTOS NO FUNCIONALES*

1. **Rendimiento y Tiempo de Respuesta:** Garantizar que la página cargue rápidamente y responda sin demoras a las interacciones del usuario, incluso durante picos de tráfico elevado.

2. **Diseño Responsivo y Adaptabilidad:** La página debe ser completamente accesible y funcional en una amplia gama de dispositivos, incluyendo computadoras de escritorio.

3. **Compatibilidad con Navegadores:** Asegurar que la landing page sea compatible con las versiones más recientes de los principales navegadores web, como Chrome, Firefox, Safari y Edge.

4. **Facilidad de Mantenimiento:** El código de la página debe ser limpio, bien documentado y organizado de manera que permita actualizaciones y mantenimiento eficiente.

5. **Carga Eficiente de Medios:** Optimizar imágenes y otros medios para asegurar una carga rápida de la página sin sacrificar la calidad visual.

X. RESTRICCIONES

- Se requiere el uso de la tecnología Preact exclusivamente para el desarrollo del Frontend.

- La tecnología Golang es la única opción aceptable para el desarrollo del Backend.

- Grpc se debe utilizar como único medio de comunicación entre el Frontend y el Backend.

- La base de datos debe ser MariaDB y no se consideran otras opciones.

- Se tiene que usar en el patrón de arquitectura de capas.

XI. RIESGOS

- Al no tener experiencia con las tecnologías a usar, puede que el desarrollo no cumpla con las buenas prácticas de programación.

- Al leer el enunciado de la presentación, se entendió que los atributos de calidad van desarrollados respecto al ejemplo práctico a realizar.
- Grpc puede no ser compatible con el protocolo de comunicación de Preact.

X. PRINCIPIOS SOLID APLICADOS

– Principio de responsabilidad única: al trabajar una arquitectura por capas, cada capa se encarga de una tarea en especifica, la capa de acceso a datos se encarga unicamente de hacer los llamados a la base de datos y traer la información
- Principio de segregación de interfaces: En Go, las interfaces son implícitamente satisfechas, lo que significa que se definen interfaces pequeñas y específicas que los tipos pueden implementar. Esto se refleja en cómo se definen interfaces para los repositorios o servicios, asegurando que cada componente solo necesite conocer y usar los métodos que son relevantes para sus necesidades.
- Principio de inversión de dependencias: Al utilizar interfaces para definir la lógica de acceso a datos y pasar concretamente estas abstracciones a la capa de lógica de negocio (como se hace al crear una nueva instancia del servidor con una referencia al repositorio), se está reduciendo las dependencias directas entre la lógica de negocio y la implementación específica de acceso a datos.

XII. ATRIBUTOS DE CALIDAD

1. **Eficiencia de desempeño:** Garantizar tiempos de carga rápidos.

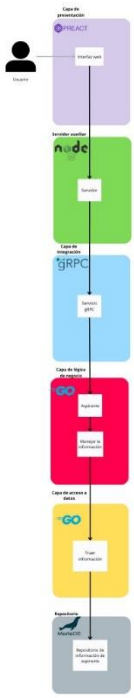
2. **Compatibilidad:** La página web debe ser accesible y funcional en todos los navegadores web y en dispositivos de escritorio (PC).

3. **Mantenibilidad:** Tener un código limpio y bien documentado para facilitar la comprensión, modificación y extensión para futuros desarrollos.

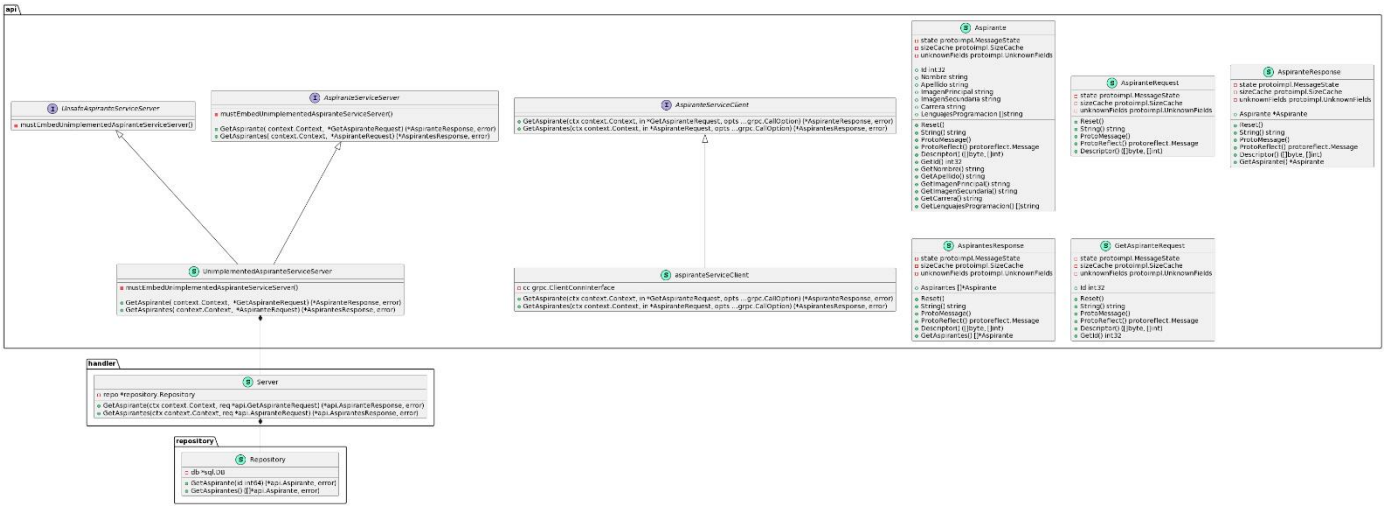
XIII. ESCENARIOS DE CALIDAD

Ref AC	Escenario	Prioridad (Cliente, Arquitecto)
Comportamiento Temporal	La landing page debe cargar completamente en menos de 2 segundos.	Alta, Alta
Utilización de Recursos	Durante la operación normal, el sistema debe mantener el uso de la CPU por debajo del 60% y el uso de memoria RAM por debajo del 75% en el servidor local.	Alta, Alta
Coexistencia	La landing page debe funcionar correctamente sin conflictos en un entorno con múltiples aplicaciones, sin afectar el rendimiento de otras aplicaciones en el PC.	Media, Media
Interoperabilidad	La landing page debe asegurar una interoperabilidad del 100% con los navegadores web modernos (Chrome, Firefox, Safari, Edge).	Alta, Alta
Modularidad	Actualizaciones de contenido o mejora en la funcionalidad no deben requerir más de 2 horas para su implementación y validación.	Media, Alta
Reusabilidad	Componentes desarrollados para la landing page deben ser diseñados para su reutilización en futuros proyectos o secciones, con una integración que no exceda las 2 horas.	Media, Media
Analizabilidad	En caso de fallas o bugs, el tiempo para identificar la causa raíz en el código no debe superar las 4 horas.	Alta, Alta

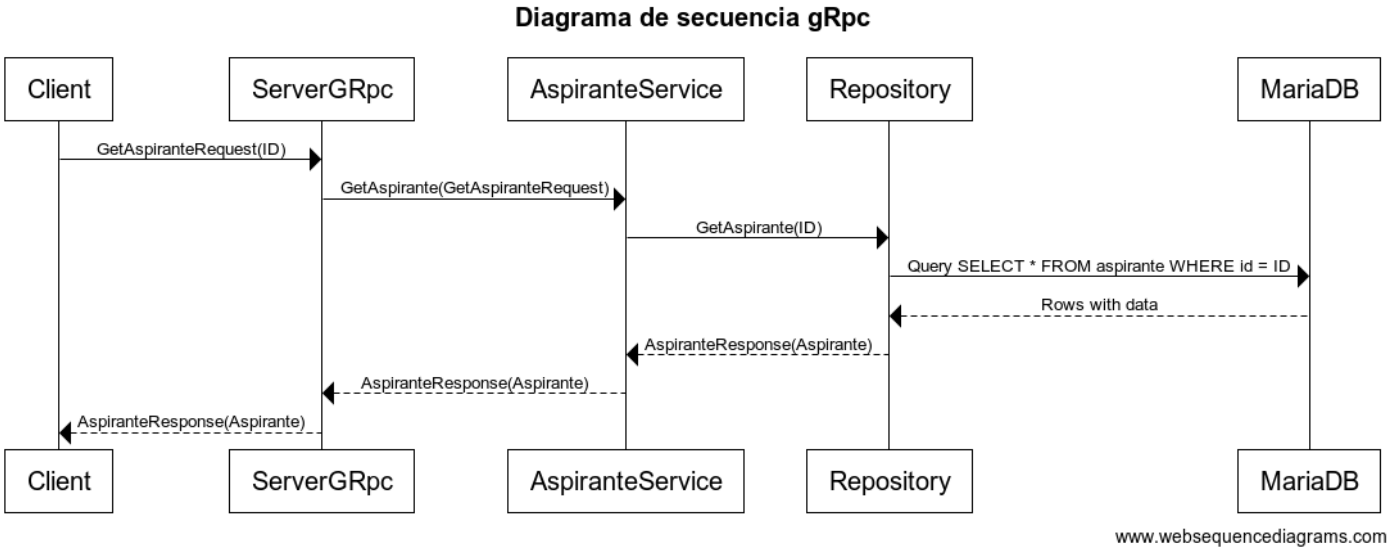
XIV. ARQUITECTURA DE ALTO NIVEL



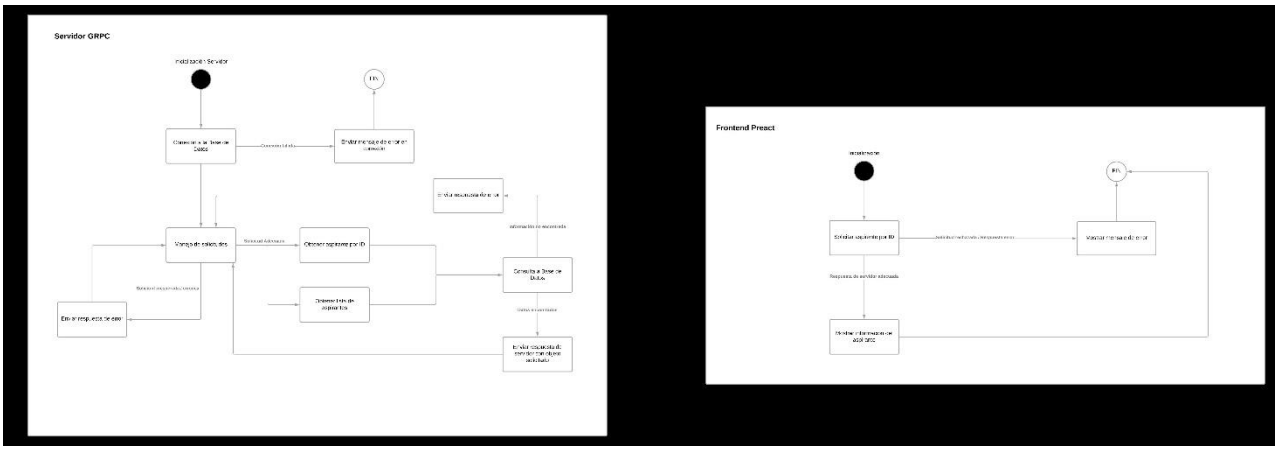
XV. DIAGRAMA DE CLASES



XVI. DIAGRAMA DE SECUENCIA



XVII. DIAGRAMA DE ESTADOS


Servidor GRPC
Frontend Preact