| Documento de Arquitectura de Software - DAS |
| --- |
| **Proyecto:** “Plataforma de Gestión de Servicios y Agenda Profesional” |
|  |
|  |
| Versión*: 0.1* |
| 03/11/2024 |

# Contenido

[**1. Contenido 1**](#_heading=h.d2xiuc4dl1oj)

[**2. Ficha del documento 3**](#_heading=h.gjdgxs)

[2.1. Versionamiento 3](#_heading=h.30j0zll)

[2.2. Integrantes 3](#_heading=h.1fob9te)

[**3. Introducción 4**](#_heading=h.3znysh7)

[**4. Objetivo de la Arquitectura 4**](#_heading=h.2et92p0)

[**5. Visión General de la Arquitectura 5**](#_heading=h.tyjcwt)

[**6. Vista Lógica 6**](#_heading=h.1t3h5sf)

[**7. Vista de Desarrollo 7**](#_heading=h.2s8eyo1)

[1. Organización del Código y Estructura de Carpetas 7](#_heading=h.grr7z62vor9b)

[Diagrama de Organización del Código 7](#_heading=h.ezcf2kqcww6e)

[Estructura de Carpetas y Descripción de los Paquetes 8](#_heading=h.ijzzj3gcd34y)

[2. Herramientas de Desarrollo 8](#_heading=h.p2vc8ghbdq99)

[3. Guía de Estándares de Codificación y Convenciones 8](#_heading=h.kjj93ddlxq0e)

[**8. Vista de Procesos 9**](#_heading=h.26in1rg)

[**9. Vista Física 9**](#_heading=h.1ci93xb)

[1. Infraestructura Física y Tecnologías de Infraestructura 10](#_heading=h.d3e4aavxbqvy)

[2. Diagrama de Despliegue 10](#_heading=h.6ep7971hftm)

[3. Conectividad y Flujo de Datos 11](#_heading=h.xa9sjkhiw47p)

[**10. Patrones de Diseño 11**](#_heading=h.3whwml4)

[1. Patrones Estructurales 12](#_heading=h.zf15dzvxd339)

[2. Patrones de Comportamiento 12](#_heading=h.ho9fjryo0nj4)

[Justificación General de los Patrones Seleccionados 13](#_heading=h.ta788pzfk8pq)

[**11. Consideraciones de Seguridad 13**](#_heading=h.2bn6wsx)

[1. Autenticación y Autorización 13](#_heading=h.p1lzoxrf633a)

[2. Cifrado de Datos en Tránsito y en Reposo 13](#_heading=h.8w5ag91po3d2)

[3. Protección contra Ataques 14](#_heading=h.id4t36mobv33)

[4. Gestión de Acceso a Recursos Sensibles y Políticas de Protección de Datos 14](#_heading=h.djcjrrrzm8hn)

[**12. Restricciones y Requerimientos Técnicos 14**](#_heading=h.qsh70q)

[12.1. Restricciones de Hardware y Software 15](#_heading=h.3as4poj)

[12.2. Dependencias Tecnológicas 15](#_heading=h.1pxezwc)

[12.3. Limitaciones de Rendimiento y Requisitos Específicos del Sistema 15](#_heading=h.49x2ik5)

[12.4. Requerimientos de Seguridad y Cumplimiento 16](#_heading=h.2p2csry)

[**13. Anexos 16**](#_heading=h.147n2zr)

# Ficha del documento

## Versionamiento

| Versión | Fecha | Modificación |
| --- | --- | --- |
| 1.0 | 13/11/2024 | Inicio Documento |
| 1.1 |  |  |
| 1.2 |  |  |
| 1.3 |  |  |
| 1.4 |  |  |
| 1.5 |  |  |

## Integrantes

| Integrantes del equipo | Rut |
| --- | --- |
| *Jairo Álvarez* | *21.055.328-2* |
| *Francisco López Avaria* | *20.883.087-2* |
| *Constanza Painevilo Jamett* | *19.561.395-8* |
| *Bastián Rodríguez* | *20.526.431-0* |

**Carrera:** Ingeniería en Informática

**Sede:** San Joaquín

# Introducción

Este Documento de Arquitectura de Software (DAS) presenta un análisis exhaustivo de los componentes estructurales y las interacciones fundamentales del sistema "Plataforma de Gestión de Servicios y Agenda Profesional". El objetivo de este documento es asegurar que el diseño de la arquitectura cumpla con los requisitos funcionales y no funcionales definidos para el proyecto, garantizando una estructura que sea escalable, segura, eficiente y adaptable a futuros cambios o expansiones.

La Plataforma de Gestión de Servicios y Agenda Profesional se diseñó para resolver la problemática de la dispersión de canales para la promoción y gestión de servicios. Al proporcionar una plataforma centralizada, facilita la búsqueda, comparación, y reserva de servicios por parte de los usuarios, y permite a los profesionales gestionar sus servicios y citas de manera organizada y eficiente.

Este documento es una referencia integral para los equipos de desarrollo, gestión, aseguramiento de calidad, y cualquier otra parte interesada en el proyecto. Su contenido incluye:

* Descripción y estructura de los componentes principales de la plataforma, como el cliente móvil, el servidor backend, y la base de datos.
* Modelos de interacción entre los componentes, detallando cómo los usuarios y profesionales interactúan con el sistema en tiempo real y cómo se gestionan las solicitudes y datos a lo largo de la arquitectura.
* Consideraciones de seguridad y de infraestructura que aseguran el resguardo de la información confidencial, el cumplimiento de los estándares de protección de datos, y el manejo seguro de transacciones y autenticaciones.
* Patrones de diseño empleados, como MVC y Factory, que permiten una estructura de código organizada y mantenible, facilitando tanto el desarrollo como la escalabilidad del sistema.
* Vista física y de despliegue, donde se describen los recursos y la infraestructura necesarios para soportar el sistema, asegurando una implementación eficiente y conectividad constante entre usuarios y el backend.

A través de la documentación en el DAS, se busca proporcionar una guía práctica y detallada para la implementación y mantenimiento de la plataforma, facilitando la alineación entre los equipos y asegurando que cada componente del sistema esté claramente definido y cumpla con los estándares de calidad establecidos. En resumen, este documento no solo especifica la arquitectura actual, sino que también establece las bases para la evolución futura del sistema, permitiendo adaptaciones y mejoras continuas en respuesta a las necesidades cambiantes de los usuarios y del mercado.

# Objetivo de la Arquitectura

El objetivo de la arquitectura de software para la "Plataforma de Gestión de Servicios y Agenda Profesional" es definir una estructura técnica robusta y organizada que garantice que el sistema cumpla con sus objetivos funcionales y no funcionales. Esta arquitectura debe asegurar una plataforma escalable, eficiente, segura y fácil de mantener, que soporte tanto el crecimiento en la cantidad de usuarios como la evolución de sus necesidades.

La arquitectura está diseñada para:

1. Optimizar el rendimiento del sistema, garantizando tiempos de respuesta rápidos y una experiencia de usuario fluida, independientemente de la cantidad de usuarios activos y de los servicios gestionados.
2. Facilitar la escalabilidad, permitiendo que el sistema crezca y se adapte conforme aumente la demanda o surjan nuevas funcionalidades sin afectar su estabilidad o rendimiento.
3. Garantizar la seguridad de la información, implementando medidas avanzadas para proteger los datos sensibles de los usuarios y profesionales, tales como autenticación segura, cifrado de datos y protección contra amenazas comunes en aplicaciones web y móviles.
4. Promover la mantenibilidad y adaptabilidad, mediante una organización modular y el uso de patrones de diseño que permiten modificaciones y actualizaciones en el sistema sin afectar su funcionamiento general. Esto facilita el trabajo del equipo de desarrollo en futuras mejoras o en la corrección de errores.
5. Asegurar la integración eficiente de componentes, permitiendo que todos los módulos (como autenticación, gestión de servicios y reserva de citas) interactúen de manera armónica y optimizada dentro del sistema.

En resumen, el objetivo de la arquitectura es ofrecer una base sólida y coherente que no solo cubra las necesidades actuales de los usuarios y profesionales, sino que también permita al sistema adaptarse al crecimiento y a las demandas futuras del mercado.

# Visión General de la Arquitectura

Esta sección describe los componentes principales del sistema y la forma en que interactúan para soportar las funcionalidades de la "Plataforma de Gestión de Servicios y Agenda Profesional". La arquitectura sigue un modelo cliente-servidor, en el que la aplicación móvil actúa como cliente y se conecta a un backend centralizado que gestiona la lógica de negocio, la autenticación y la comunicación con la base de datos.

Diagrama de arquitectura general: (Incluir un diagrama visual que represente la arquitectura y sus componentes clave).

**Componentes principales:**

1. Aplicación Móvil (Cliente): La interfaz de usuario, accesible desde dispositivos móviles, donde los clientes y profesionales pueden interactuar con el sistema. Esta aplicación permite a los usuarios autenticarse, explorar y reservar servicios, y gestionar su agenda, mientras que los profesionales pueden publicar servicios y gestionar sus citas.
2. Backend: El backend es el núcleo de la lógica de negocio de la plataforma. Gestiona las solicitudes de la aplicación móvil, maneja la autenticación y autorización de usuarios, y se comunica con la base de datos para almacenar y recuperar información. Este componente también coordina las notificaciones y la integración con servicios externos cuando es necesario.
3. Base de Datos: Una base de datos relacional y segura que almacena toda la información de los usuarios, servicios, y reservas de manera estructurada. Los datos son accesibles a través del backend y están organizados para soportar consultas rápidas y optimizadas. Además, la base de datos cuenta con medidas de seguridad para proteger la integridad y confidencialidad de los datos.
4. Servicios Externos: Servicios de terceros que complementan las funcionalidades de la plataforma, tales como autenticación externa (por ejemplo, a través de Google o Facebook) y notificaciones push para mantener a los usuarios informados de actualizaciones y recordatorios.
5. Flujo de datos clave: La comunicación entre la aplicación móvil y el backend se realiza de manera segura, con encriptación de los datos en tránsito utilizando HTTPS. Los datos personales y sensibles, como credenciales de usuario y detalles de reserva, se transmiten de forma encriptada desde el cliente al servidor. El backend valida y procesa estas solicitudes, interactuando con la base de datos y, cuando es necesario, con servicios externos para cumplir con las funcionalidades del sistema.

Esta arquitectura garantiza una interacción eficiente y segura entre los componentes, brindando a los usuarios una experiencia confiable y optimizada al utilizar la plataforma.

# Vista Lógica

La vista lógica detalla la estructura funcional del sistema, organizando los módulos, subsistemas y componentes principales que soportan las funcionalidades de la "Plataforma de Gestión de Servicios y Agenda Profesional". Esta sección proporciona una visión detallada de los elementos internos y de cómo interactúan para cumplir con los requisitos del sistema.

* Diagrama de componentes lógicos:

(Incluir un diagrama que represente la organización lógica del sistema, con los módulos y sus relaciones)

**Descripción de los Módulos y sus Responsabilidades**

1. Módulo de Autenticación: Responsable de gestionar el acceso seguro al sistema. Este módulo permite a los usuarios autenticarse mediante credenciales propias o autenticación de terceros (por ejemplo, Google, Facebook). También maneja la autorización de permisos para asegurar que cada usuario solo acceda a las funcionalidades correspondientes a su rol (cliente o profesional).
2. Módulo de Gestión de Servicios: Permite a los profesionales crear, actualizar y eliminar sus servicios dentro de la plataforma. Este módulo organiza la información de los servicios, incluyendo detalles como descripción, precios, y disponibilidad, y la hace accesible para que los usuarios puedan explorar las opciones disponibles.
3. Módulo de Búsqueda y Filtro: Este módulo facilita la búsqueda de servicios en la plataforma, permitiendo a los usuarios aplicar filtros como ubicación, tipo de servicio, rango de precios, y disponibilidad. Su objetivo es mejorar la experiencia del usuario ofreciendo resultados precisos y personalizables.
4. Módulo de Reserva de Citas: Gestiona las reservas entre clientes y profesionales. Permite a los usuarios seleccionar fecha y hora para la prestación de servicios y asegura la disponibilidad del profesional, actualizando la agenda automáticamente. Este módulo también coordina las notificaciones de confirmación y recordatorios de cita.
5. Módulo de Notificaciones: Responsable de enviar notificaciones automáticas a los usuarios para mantenerlos informados sobre el estado de sus reservas, confirmaciones de cita, y cualquier actualización importante. Este módulo se integra con el sistema de notificaciones push para garantizar la comunicación en tiempo real.
6. Módulo de Gestión de Agenda para Profesionales: Permite a los profesionales configurar y gestionar su disponibilidad dentro de la plataforma. A través de este módulo, los profesionales pueden organizar sus horarios, evitando conflictos en la programación de citas y optimizando su tiempo de atención a los clientes.

**Interacciones y Dependencias entre Módulos**

* **Dependencias entre módulos:** El módulo de autenticación interactúa con todos los demás módulos para validar el acceso de los usuarios a las funcionalidades correspondientes.
  + El módulo de reserva de citas depende de la información proporcionada por los módulos de gestión de servicios y gestión de agenda para asegurar la disponibilidad y gestionar la interacción entre clientes y profesionales.
  + El módulo de notificaciones se conecta con el módulo de reserva de citas y gestión de agenda para enviar recordatorios oportunos y mantener informados a los usuarios.
* **Interacciones clave:** Los módulos están diseñados para ser modulares e independientes en la medida de lo posible, lo que facilita el mantenimiento y la escalabilidad. Los datos se transfieren de forma segura entre módulos y se gestionan mediante el backend para optimizar el rendimiento del sistema.

La vista lógica permite entender cómo se estructura y organiza la funcionalidad de la plataforma, garantizando que cada módulo cumpla con sus responsabilidades de forma eficaz y que las interacciones entre ellos soporten las funcionalidades requeridas.

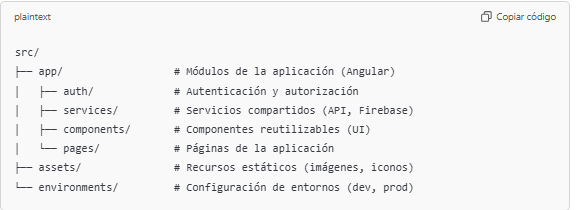
# Vista de Desarrollo

La vista de desarrollo proporciona una perspectiva técnica de la organización interna del sistema, permitiendo a los desarrolladores trabajar de manera estructurada y eficiente. La organización del código, herramientas de desarrollo y estándares de codificación aseguran consistencia y calidad en el desarrollo de la aplicación.

#### 1. Organización del Código y Estructura de Carpetas

##### Diagrama de Organización del Código

La aplicación se organiza en tres capas principales:



##### Estructura de Carpetas y Descripción de los Paquetes

* **app/auth**: Contiene la lógica relacionada con la autenticación de usuarios, incluyendo servicios que interactúan con Firebase Authentication.
* **app/services**: Define los servicios de la aplicación, como la comunicación con Firebase Firestore y la gestión de notificaciones push mediante Firebase Cloud Messaging.
* **app/components**: Incluye componentes de UI reutilizables, como botones, encabezados, y formularios de autenticación, que se utilizan en distintas páginas de la aplicación.
* **app/pages**: Cada página principal de la aplicación se organiza en módulos separados. Ejemplos de páginas son el módulo de inicio de sesión, el módulo de reservas y el perfil de usuario.
* **assets**: Carpeta para los recursos estáticos de la aplicación, tales como iconos y archivos multimedia.
* **environments**: Contiene los archivos de configuración de entornos (por ejemplo, environment.ts para desarrollo y environment.prod.ts para producción) que manejan variables de entorno, como las credenciales de Firebase.

#### 2. Herramientas de Desarrollo

* **Entorno de Desarrollo**: La aplicación se desarrolla principalmente en **Visual Studio Code**, que ofrece soporte para los principales frameworks y bibliotecas utilizados (Ionic, Angular, Node.js).
* **Control de Versiones**: Se utiliza **Git** para el control de versiones y la gestión del código fuente, con **GitHub** como plataforma de almacenamiento remoto. Las ramas se organizan siguiendo el flujo de trabajo Git Flow, separando ramas de desarrollo (develop), producción (main o master) y características (feature/\*).
* **Firebase CLI**: La herramienta de línea de comandos de Firebase (Firebase CLI) facilita la configuración y el despliegue de servicios de Firebase, permitiendo una integración fluida de autenticación, base de datos y notificaciones push.
* **Ionic CLI**: El CLI de Ionic se usa para la gestión de los entornos de desarrollo y despliegue de la aplicación, incluyendo la compilación y ejecución en emuladores o dispositivos físicos.

#### 3. Guía de Estándares de Codificación y Convenciones

**Estándares de Codificación**:

La aplicación sigue el **estándar de codificación de Angular**, que incluye la separación de lógica y presentación en componentes, uso de servicios para lógica de negocio, y enrutamiento modular.

Se emplea la convención de nombres en camelCase para variables y PascalCase para clases y componentes, asegurando consistencia en la nomenclatura.

**Convenciones de Nombres**:

**Componentes**: Los nombres de los componentes siguen el formato NombreComponent, para que LoginComponent indique claramente su propósito y ubicación.

**Servicios**: Los nombres de servicios siguen el formato NombreService (por ejemplo, AuthService) y se almacenan en la carpeta de servicios para fácil acceso.

**Archivos de Estilo**: Cada componente tiene un archivo .scss asociado para los estilos, permitiendo una personalización de estilo modular y específica para cada componente.

**Revisión y Calidad de Código**:

Se aplican herramientas de **linting** como **TSLint** para TypeScript y **ESLint** para JavaScript, garantizando que el código cumpla con las normas de estilo y evitando errores comunes.

La revisión de código se realiza mediante **pull requests** en GitHub, permitiendo que otros desarrolladores revisen y aprueben los cambios antes de integrarlos en la rama principal.

**Documentación en Línea**: Cada servicio y componente incluye documentación en línea utilizando comentarios JSDoc, lo cual facilita la comprensión y el mantenimiento del código.

# Vista de Procesos

[Describe cómo interactúan los componentes en tiempo de ejecución y la secuencia de eventos que se produce en el sistema.

Diagramas de secuencia y colaboración para mostrar el flujo de datos entre componentes en los casos de uso clave (ej. autenticación, reserva de citas).

Descripción de cómo se gestionan los eventos y transacciones en el sistema.

Explicación de la concurrencia y la sincronización de procesos, si aplica.]

# Vista Física

La infraestructura física para el despliegue del sistema está basada en una arquitectura en la nube, aprovechando los servicios de Firebase y Google Cloud Platform (GCP) para asegurar escalabilidad, rendimiento y seguridad en el entorno de producción.

#### 1. Infraestructura Física y Tecnologías de Infraestructura

* **Servidor en la Nube**: El sistema se despliega en los servidores de Google Cloud Platform (GCP), que ofrece servicios de alta disponibilidad y escalabilidad automática. Firebase actúa como backend, gestionando la autenticación, almacenamiento en tiempo real y la base de datos.
* **Servicios de Bases de Datos**:
  + **Firebase Firestore**: Base de datos NoSQL en tiempo real, optimizada para aplicaciones móviles y con sincronización automática entre dispositivos.
  + **Firebase Realtime Database** (opcional): Puede ser utilizado para casos específicos de sincronización instantánea, si es requerido en módulos específicos de la aplicación.
* **Servidor de Aplicaciones**: La lógica de backend está gestionada mediante Node.js, ejecutándose en entornos controlados de GCP que aseguran soporte y capacidad de respuesta, y que permiten implementar microservicios específicos.
* **Servicios Adicionales**: Firebase Hosting es utilizado para la entrega de contenido estático (HTML, CSS, JavaScript) del frontend. Firebase Cloud Messaging (FCM) gestiona las notificaciones push para asegurar la comunicación en tiempo real con los usuarios de la aplicación móvil.

#### 2. Diagrama de Despliegue

**Firebase Cloud Platform**:

**Firestore**: Contiene las colecciones de datos de la aplicación.

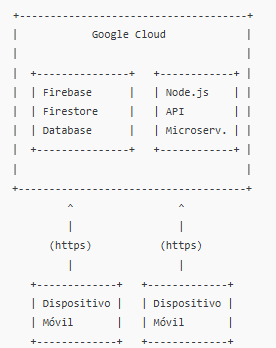
**Firebase Authentication**: Gestiona el control de acceso y las credenciales de usuario.

**Firebase Functions (Opcional)**: Permite ejecutar lógica backend para automatizar tareas como el procesamiento de pagos o la validación de datos, de ser necesario.

**Node.js en GCP**:

**Servidor de Autenticación**: Gestiona los tokens JWT y las sesiones de usuario.

**Microservicios**: Gestionan funcionalidades específicas como las consultas de datos y las notificaciones, interactuando directamente con Firebase.



#### 3. Conectividad y Flujo de Datos

* **Conectividad entre Dispositivos Móviles y Firebase**: Los dispositivos móviles se conectan a Firebase mediante HTTPS, asegurando que todas las comunicaciones de datos están cifradas en tránsito. La conexión permite la sincronización en tiempo real de datos entre Firebase y los dispositivos, lo cual es crítico para mantener la consistencia de la información en múltiples puntos de acceso.
* **Interacción con Node.js en GCP**: Las peticiones complejas o personalizadas, como la gestión avanzada de usuarios o las transacciones, son procesadas por microservicios en Node.js. Estos microservicios están protegidos y autenticados mediante tokens JWT, y reciben las solicitudes de los dispositivos móviles a través de HTTPS.
* **Flujo de Notificaciones Push**: Las notificaciones son gestionadas por Firebase Cloud Messaging (FCM), que permite enviar alertas y actualizaciones a los usuarios directamente en sus dispositivos móviles. FCM es una solución optimizada para entregar mensajes de forma eficiente y escalable, manteniendo a los usuarios informados sin saturar el servidor.

# Patrones de Diseño

En el diseño de la arquitectura de la aplicación móvil desarrollada con Ionic, Angular, Node.js y Firebase, se han implementado varios patrones de diseño para garantizar una arquitectura modular, eficiente y fácilmente mantenible. A continuación, se describen los patrones estructurales y de comportamiento seleccionados:

#### 1. Patrones Estructurales

* **Modelo-Vista-Controlador (MVC)**:
  + **Descripción**: El patrón MVC organiza la aplicación en tres componentes principales: el *Modelo* (que representa los datos y la lógica de la aplicación), la *Vista* (la interfaz de usuario), y el *Controlador* (que actúa como intermediario entre la vista y el modelo).
  + **Implementación**: En esta arquitectura, *Ionic y Angular* gestionan la vista y controladores, lo que permite una separación clara entre la lógica de negocio y la interfaz de usuario. El modelo de datos se gestiona con Firebase, que sincroniza y almacena la información.
  + **Justificación**: MVC facilita la mantenibilidad y la escalabilidad al permitir cambios en la interfaz sin afectar la lógica de negocio, y viceversa, lo cual es ideal para aplicaciones móviles que evolucionan rápidamente.
* **Repositorio**:
  + **Descripción**: Este patrón proporciona una capa intermedia entre la aplicación y la base de datos, permitiendo el acceso y la manipulación de datos sin que el resto del sistema sepa cómo se gestionan estos datos internamente.
  + **Implementación**: Se utiliza una clase de servicio en Angular que actúa como repositorio, encapsulando las operaciones de Firebase, como las consultas a la base de datos y las actualizaciones. Esto permite abstraer el acceso a los datos y facilita la reutilización de código.
  + **Justificación**: El patrón Repositorio reduce el acoplamiento entre la lógica de negocio y Firebase, permitiendo que el sistema pueda adaptarse a diferentes fuentes de datos en el futuro, si fuese necesario.
* **Microservicios** (aplicado en la capa backend):
  + **Descripción**: El patrón de microservicios divide la aplicación en servicios pequeños y especializados que interactúan entre sí mediante APIs. Cada servicio se puede desarrollar y desplegar de forma independiente.
  + **Implementación**: Node.js se utiliza para desarrollar microservicios ligeros que gestionan la autenticación, la administración de usuarios y otras funcionalidades específicas. Estos servicios se comunican con Firebase y responden a peticiones del frontend.
  + **Justificación**: La arquitectura de microservicios permite escalabilidad y flexibilidad, ya que cada servicio puede escalar de manera independiente según la demanda de la aplicación.

#### 2. Patrones de Comportamiento

* **Singleton**:
  + **Descripción**: Este patrón garantiza que una clase tenga una sola instancia y proporciona un punto de acceso global a dicha instancia.
  + **Implementación**: Se aplica en servicios de Angular para Firebase, como el servicio de autenticación, asegurando que solo exista una instancia de estos servicios en toda la aplicación.
  + **Justificación**: El patrón Singleton asegura que los servicios compartidos no creen múltiples instancias, lo que optimiza el uso de memoria y garantiza un control centralizado de operaciones como el inicio de sesión y la sincronización de datos.
* **Factory**:
  + **Descripción**: Este patrón proporciona una interfaz para crear objetos sin especificar las clases concretas que serán instanciadas.
  + **Implementación**: En la capa de servicios de Angular, se utiliza un servicio Factory que selecciona y devuelve diferentes configuraciones para las conexiones de Firebase dependiendo del entorno (desarrollo, prueba o producción).
  + **Justificación**: El patrón Factory permite centralizar y automatizar la creación de instancias de servicios, facilitando la configuración de la aplicación para diferentes entornos y garantizando consistencia en la inicialización de recursos.
* **Observer**:
  + **Descripción**: El patrón Observer define una relación uno-a-muchos entre objetos, donde si un objeto cambia, todos sus dependientes son notificados automáticamente.
  + **Implementación**: Este patrón se emplea en la interacción entre componentes de Angular y Firebase. Al actualizar los datos en Firebase, los observadores en Angular reciben las actualizaciones automáticamente, permitiendo que la vista refleje los cambios en tiempo real.
  + **Justificación**: El patrón Observer es fundamental para la sincronización en tiempo real de Firebase, permitiendo que los usuarios reciban actualizaciones instantáneas sin necesidad de recargar la aplicación.

#### Justificación General de los Patrones Seleccionados

Los patrones seleccionados ayudan a mantener el sistema modular, escalable y fácil de mantener. Al aplicar patrones estructurales como MVC y Repositorio, el código es más organizado y se minimiza el acoplamiento entre componentes. Los patrones de comportamiento como Singleton, Factory y Observer facilitan el uso eficiente de recursos y mejoran la experiencia de usuario mediante actualizaciones en tiempo real y la flexibilidad en la configuración de servicios.

# Consideraciones de Seguridad

Para proteger la integridad, confidencialidad y disponibilidad del sistema desarrollado con Ionic, Angular, Node.js y Firebase, se han aplicado diversas estrategias de seguridad en los siguientes aspectos:

#### 1. Autenticación y Autorización

* **Autenticación**: Se implementa Firebase Authentication, que soporta autenticación segura mediante proveedores como Google, Facebook, y correo electrónico. Firebase se encarga de gestionar y validar la autenticación del usuario.
* **Autorización mediante JWT**: Los tokens JSON Web Tokens (JWT) generados por Firebase permiten validar la identidad de los usuarios en cada solicitud. Estos tokens contienen información de autenticación que es verificable en el servidor, asegurando que solo usuarios autorizados puedan acceder a recursos sensibles.
* **Roles y permisos**: Utilizando las reglas de seguridad de Firebase, se configuran niveles de acceso según el rol del usuario, como administrador o usuario estándar. Esto asegura que solo los usuarios con los permisos adecuados puedan interactuar con ciertos datos o funcionalidades.

#### 2. Cifrado de Datos en Tránsito y en Reposo

* **Datos en Tránsito**: Todas las comunicaciones entre el cliente (aplicación móvil) y Firebase están protegidas mediante HTTPS, que garantiza el cifrado de los datos en tránsito, evitando la interceptación de información sensible.
* **Datos en Reposo**: Firebase aplica cifrado en reposo por defecto en los servidores de Google Cloud Platform, protegiendo los datos almacenados en Firestore y Realtime Database contra accesos no autorizados en caso de violaciones de seguridad del servidor.

#### 3. Protección contra Ataques

* **Inyección de Código**: Aunque Firebase no utiliza bases de datos SQL, se aplican las Firebase Security Rules para validar la integridad de los datos en el servidor, evitando inyecciones de código malicioso en los datos enviados.
* **Cross-Site Scripting (XSS)**: En el frontend, Ionic y Angular emplean métodos de sanitización de datos que previenen la ejecución de scripts maliciosos. Angular Sanitization se encarga de desinfectar las entradas de usuario en vistas y plantillas, mitigando así el riesgo de XSS.
* **Cross-Site Request Forgery (CSRF)**: Aunque el riesgo de CSRF es reducido en aplicaciones móviles, se mantiene la verificación de tokens de usuario en todas las solicitudes sensibles, evitando que un usuario malicioso ejecute acciones en nombre de otro usuario.
* **Limitación de Frecuencia de Solicitudes**: Se implementa un límite de frecuencia de solicitudes en endpoints críticos para mitigar ataques de fuerza bruta. Además, se considera la integración de Google reCAPTCHA en formularios de autenticación para prevenir ataques de bots.

#### 4. Gestión de Acceso a Recursos Sensibles y Políticas de Protección de Datos

**Reglas de Seguridad de Firebase**: Las reglas de seguridad de Firebase definen y restringen el acceso a datos según el nivel de autenticación y los permisos específicos de cada usuario. Las reglas se actualizan periódicamente para reflejar cambios en los requisitos de acceso.

**Principio de Menor Privilegio**: Los permisos se asignan basados en el principio de menor privilegio, otorgando a cada usuario solo el acceso necesario para cumplir sus funciones. Esto minimiza el riesgo de exposición de datos sensibles.

**Políticas de Protección de Datos**: Se implementan auditorías que registran eventos clave, como intentos de acceso a recursos sensibles y modificaciones en configuraciones de seguridad. Estos registros permiten detectar y responder ante posibles incidentes de seguridad.

**Almacenamiento Seguro en el Cliente**: Para proteger los datos en el dispositivo móvil, se utiliza el almacenamiento seguro de Ionic en lugar de LocalStorage, previniendo el acceso no autorizado a tokens o información sensible almacenada en el dispositivo.

# Restricciones y Requerimientos Técnicos

En el diseño y desarrollo de la arquitectura del aplicativo móvil desarrollado con Ionic, Angular, Node.js y Firebase, se han considerado las siguientes restricciones y requerimientos técnicos para asegurar el funcionamiento óptimo del sistema

## Restricciones de Hardware y Software

* Requisitos de Hardware para los Servidores: El backend y la base de datos de la plataforma deben ejecutarse en servidores con una configuración mínima.
  + Procesador de 4 núcleos o superior.
  + 16 GB de RAM para asegurar un rendimiento eficiente y capacidad de respuesta.
  + Almacenamiento SSD de al menos 250 GB para una carga rápida de datos y copias de seguridad.
* Requisitos para el Cliente Móvil: La aplicación móvil está diseñada para dispositivos con sistemas operativos:
  + Android versión 8.0 (Oreo) o superior.
  + iOS versión 13 o superior. Esto garantiza que la plataforma sea compatible con la mayoría de los dispositivos móviles actuales.

## Dependencias Tecnológicas

El sistema depende de tecnologías y bibliotecas específicas, las cuales deben estar actualizadas y compatibles entre sí para asegurar la estabilidad y seguridad de la plataforma.

* Backend:
  + Node.js en su versión 14 o superior, ya que ofrece una base sólida para la ejecución en el servidor y permite manejar grandes volúmenes de tráfico con concurrencia efectiva.
  + Express.js como framework para agilizar el desarrollo del backend y gestionar las solicitudes HTTP de manera eficiente.
  + Base de Datos: Se usa MongoDB o PostgreSQL para almacenar y gestionar los datos. La versión recomendada es 4.4 o superior para MongoDB y 12 para PostgreSQL, asegurando soporte para las características de seguridad y rendimiento requeridas.
* Frontend (Aplicación Móvil):
  + React Native o Flutter en sus versiones más recientes, para una experiencia de usuario fluida y desarrollo multiplataforma.
  + APIs de Integración de Autenticación (por ejemplo, Google OAuth2), para gestionar el acceso seguro de usuarios.
  + Bibliotecas de UI como Material UI para garantizar una interfaz moderna y accesible.

## Limitaciones de Rendimiento y Requisitos Específicos del Sistema

Para asegurar una experiencia de usuario fluida, el sistema debe cumplir con ciertos estándares de rendimiento:

* Tiempo de Respuesta Máximo: Las operaciones críticas, como autenticación, búsqueda de servicios y reserva de citas, deben completarse en un tiempo de respuesta inferior a 2 segundos para garantizar una interacción ágil.
* Concurrencia Máxima: El sistema debe ser capaz de manejar un mínimo de 1000 usuarios concurrentes sin afectar el rendimiento. Esto implica optimizaciones en el backend para gestionar múltiples solicitudes simultáneas mediante prácticas como el balanceo de carga y la implementación de colas de mensajes para solicitudes en espera.
* Escalabilidad Horizontal: La arquitectura debe soportar la escalabilidad horizontal mediante la adición de servidores adicionales en caso de que la demanda aumente, especialmente en eventos pico o en fases de crecimiento de usuarios.
* Capacidad de Almacenamiento: Dado el volumen de datos de usuarios, servicios y reservas, el sistema debe ser capaz de gestionar una base de datos de al menos 100,000 registros sin perder rendimiento. Esto implica una estructura de almacenamiento optimizada y procedimientos de limpieza y archivado para datos obsoletos o inactivos.

## Requerimientos de Seguridad y Cumplimiento

* Certificación de Seguridad y Cumplimiento: El sistema debe cumplir con normativas de seguridad y protección de datos (como GDPR para datos personales en el ámbito europeo), lo cual requiere controles estrictos sobre la recolección, almacenamiento y eliminación de datos de usuario.

Estas restricciones y requerimientos técnicos proporcionan una guía clara para asegurar que el sistema opere dentro de los límites óptimos de hardware, software, rendimiento y seguridad, y garantizan que la arquitectura pueda adaptarse a las necesidades y crecimiento de la plataforma.

# Anexos

[Cualquier información adicional que complemente el documento principal, tales como:

Glosario de términos técnicos.

Diagramas complementarios.

Documentación de APIs y servicios externos.

Referencias a documentación de herramientas y marcos utilizados.]