UNIVERSIDAD SANTO TOMAS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
T U N J A

Soluciones tecnológicas para procesos administrativos y clínicos en Grupo Prevenso.

PROPONENTE
Juan David Buitrago Lozano

DIRECTOR
Martha Susana Contreras Ortiz

Tunja
Fecha de presentación (20, diciembre, 2025)

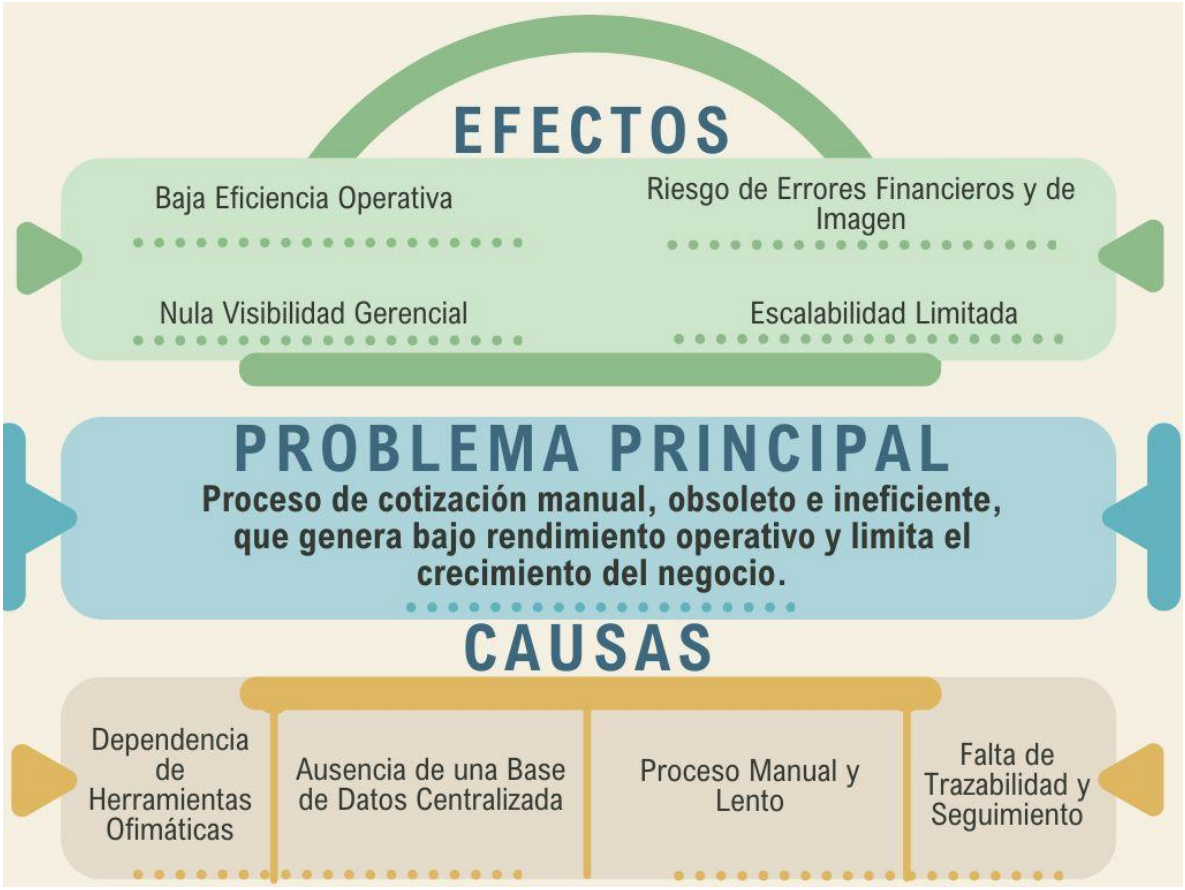
CONTENIDO

1. FICHA TÉCNICA DE LA PASANTÍA

Título	Soluciones tecnológicas para procesos administrativos y clínicos en Grupo Prevenso.
Nombre Estudiante	Juan David Buitrago Lozano
Correo electrónico estudiante	Juan.buitragol@usantoto.edu.co
Director	Martha Susana Contreras Ortiz
Entidad o sector donde realizará la pasantía	Grupo Prevenso LTDA
Lugar de ejecución de la pasantía	Remoto
Duración	4 meses
Los abajo firmantes confirman que todos los datos incluidos en la presente propuesta son correctos y verídicos, que no incumplen ninguna ley o norma vigente (incluir nombres y firmas de estudiantes y director).	
Firma del autor Juan David Buitrago Lozano Firma del director Martha Susana Contreras Ortiz	

2. PLANTEAMIENTO DEL PROBLEMA

Figura 1. Árbol de problemas



Fuente: Autor.

El árbol de problemas (Figura 1) expone cómo las causas, arraigadas en deficiencias operativas y tecnológicas, convergen en el problema central: un sistema de cotización manual que actúa como un cuello de botella. Esto genera diversos efectos negativos que no solo afectan el trabajo diario, sino que generan una desventaja en el área competitiva, la económica y de crecimiento estratégico. Este análisis resalta la importancia crítica de una solución tecnológica que automatice el proceso y organice la información.

3. OBJETIVOS

3.1. Objetivo General

Desarrollar una aplicación multiplataforma, tanto móvil como de escritorio, para la gestión interna de cotizaciones de la empresa Prevenso, mediante una metodología ágil, con el fin de optimizar la eficiencia y la trazabilidad del proceso.

3.2. Objetivos específicos

Tabla 1. Objetivos específicos

1	Actualizar el contenido visual y textual del sitio web en las secciones Inicio, Nosotros, Servicios y Contacto, utilizando herramientas de diseño web y lineamientos de comunicación institucional, para mejorar la experiencia del usuario y la coherencia visual de la plataforma.
2	Diseñar una arquitectura de software desacoplada y una estructura de base de datos normalizada en PostgreSQL, definiendo las capas de backend (Controller, Service, Repository) y la estructura del frontend (Feature-first) para garantizar un desarrollo escalable.
3	Implementar una API RESTful completa y segura utilizando Node.js, TypeScript y Express con la lógica de negocio para la gestión de usuarios, clientes, servicios, tarifas y cotizaciones.
4	Construir una interfaz de usuario completa e intuitiva con Flutter, conectando todas las pantallas a la API del backend para permitir la gestión de datos en tiempo real y asegurando su funcionamiento en dispositivos móviles y de escritorio.

4. DESCRIPCIÓN DE LAS ACTIVIDADES REALIZADAS

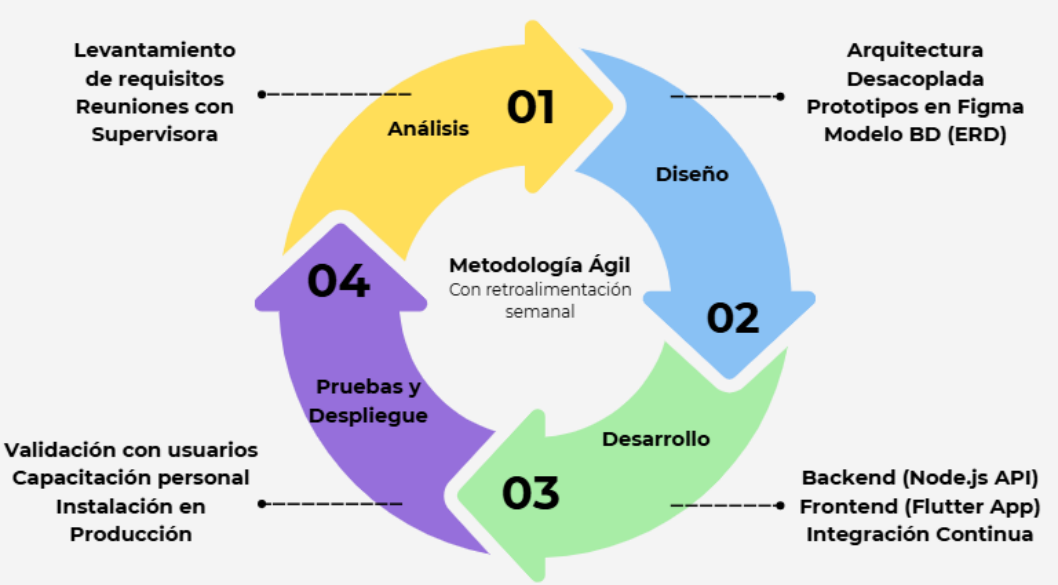
4.1. Metodología de Desarrollo

Para la ejecución del proyecto se adoptó una metodología de desarrollo de software Ágil, con un enfoque iterativo e incremental. Esta elección se fundamentó en la necesidad de adaptar el producto a los requerimientos cambiantes de la empresa y de entregar valor funcional desde las primeras etapas. Como señalan (Beck, Beedle, & van Bennekum, 2001), este enfoque permite a los equipos de desarrollo responder a la imprevisibilidad mediante la entrega de incrementos de trabajo funcionales y la colaboración constante con el cliente.

El ciclo de vida del desarrollo se estructuró en cuatro fases cíclicas, permitiendo una retroalimentación constante por parte de la tutora empresarial y la gerencia de Grupo Prevenso:

1. **Fase de Análisis y Requerimientos:** Levantamiento de información mediante reuniones semanales para identificar los puntos críticos del proceso manual y definir las reglas de negocio.
2. **Fase de Diseño y Arquitectura:** Elaboración de los planos técnicos (ERD, Arquitectura en Capas) y visuales (Mockups en Figma) antes de la codificación.
3. **Fase de Desarrollo (Codificación):** Construcción paralela del Backend (API) y Frontend (App), siguiendo una arquitectura desacoplada.
4. **Fase de Pruebas y Despliegue:** Verificación de funcionalidad, corrección de errores (bugs) y despliegue en entornos de prueba y producción.

Figura 2. Ciclo de la Metodología Ágil aplicada en el proyecto.



Fuente: Autor.

4.2. Actividades realizadas en el proyecto

Se debe relacionar el conjunto de actividades que realizó en el proyecto estableciendo fechas de los productos entregados.

Tabla 2. Descripción de actividades

Descripción de la actividad	Fecha inicio	Fecha entrega	Producto entregado
Levantamiento de requerimientos y análisis	1 de Julio	4 de julio	Documento con los requerimientos implícitos
Mantenimiento y actualización del sitio web	2 de Julio	7 de julio	Sitio web actualizado
Diseño de la arquitectura de software	7 de Julio	11 de julio	Mockups
Diseño del modelo de datos (PostgreSQL)	8 de Julio	14 de julio	Diagrama ER
Configuración del entorno (Node.js, TS)	14 de Julio	15 de julio	Entorno completamente configurado
Implementación del módulo de autenticación	15 de Julio	21 de julio	Módulo de autenticación finalizado
Desarrollo del CRUD de entidades	21 de Julio	1 de agosto	CRUD de entidades terminado y presentado en reunión
Implementación del módulo de	1 de agosto	15 de agosto	Reunión mostrando

cotizaciones			módulo de las cotizaciones terminado
Configuración del proyecto y estructura	18 de agosto	21 de agosto	Configuración y estructura del proyecto terminada
Desarrollo de pantallas de autenticación	22 de agosto	4 de septiembre	Reunión mostrando pantalla de login
Desarrollo de pantallas de gestión (CRUD)	5 de septiembre	17 de septiembre	Reunión mostrando todas las pantallas de la aplicación.
Implementación del formulario de cotización	22 de septiembre	9 de octubre	Reunión mostrando la creación y edición de una cotización
Implementación de generación de PDF	13 de octubre	17 de octubre	Pdf con los datos asignados en la app
Pruebas integrales de la aplicación (E2E)	20 de octubre	28 de octubre	Funcionamiento correcto de la aplicación
Despliegue del backend en Render	27 de octubre	3 de noviembre	Entrega de la aplicación a directiva de la empresa
Elaboración de documentación y entrega final	4 de noviembre	5 de noviembre	Instalación de la app a todos los equipos de la empresa.

La ejecución de la pasantía se estructuró en fases alineadas con los objetivos específicos propuestos. El seguimiento del proyecto se realizó mediante reuniones semanales de avance con la tutora empresarial asignada, en las cuales se presentaban los logros, se resolvían dudas y se planificaban las siguientes iteraciones. Esta sección describe en detalle el proceso llevado a cabo para el cumplimiento de cada objetivo, presentando una narrativa cronológica de las actividades, los desafíos técnicos encontrados, las metodologías aplicadas y los resultados obtenidos.

4.2.1. Actualización del Sitio Web Corporativo

4.2.1.1 Introducción y Fase de Diagnóstico (Semana 1)

El propósito inicial de la pasantía consistió en una primera etapa de inmersión en la infraestructura tecnológica del Grupo Prevenso.

La tarea inicial encomendada fue la de asegurar el acceso a la plataforma de gestión. El personal del Grupo Prevenso carecía de credenciales de administrador de WordPress, por ende, se solicitó recuperar ese acceso. Esta tarea implico la entrada al panel de administración del hosting (cPanel), este es suministrado por el proveedor. Ya en el

cPanel, se usó la herramienta de gestión de bases de datos para encontrar la tabla wp_users de la instalación de WordPress. A partir de ahí, se modificó el registro del usuario administrador, adjudicando una nueva contraseña encriptada.

Después de verificar el acceso al panel /wp-admin con las credenciales actualizadas, se entregaron de forma segura a la tutora empresarial.

Ese primer paso presento la posibilidad de hacer cambios y estableció también los protocolos de comunicación y entrega de resultados que durarían, durante el proyecto.

Con el acceso administrativo ya asegurado, el sitio web corporativo que usaba WordPress y estaba hospedado en un servidor Apache con cPanel, fue el primer campo de pruebas. Después, la tarea fue hacer una auditoría completa del sitio. Para esto se analizó a fondo las páginas más importantes Inicio, Nosotros, Servicios, Medicina Preventiva, Alturas y Contacto.

Incoherencia de Contenido: La información más básica, como los años de experiencia de la empresa, era inconsistente. La página de Inicio podía mostrar "más de 20 años" mientras que la de "Nosotros" no mencionaba el dato, generando confusión y restando credibilidad.

Errores Visuales (Bugs de UI): Se detectaron elementos de la interfaz de usuario que no renderizaban correctamente. Específicamente, en la página de Inicio, una sección de íconos de servicios aparecía con "cajas" rotas, debido a un conflicto en la carga de librerías de fuentes. Además, como se puede observar en la Figura 3, el sitio presentaba graves problemas de maquetación con imágenes y textos sobrepuestos, lo que dificultaba la lectura y afectaba negativamente la estética profesional de la empresa.

Contenido Desactualizado: Secciones críticas como el portafolio de servicios no reflejaban la oferta comercial actual de la empresa, y la información de contacto estaba incompleta.

Figura 3. Captura de pantalla del estado anterior de la página de Inicio

Fuente: Sitio web de Grupo Prevenso.

4.2.1.2 Fase de Planificación y Ejecución de Cambios (Semana 2)

Tras la auditoría, se sostuvo una reunión de alineación con el supervisor asignado en Grupo Prevenso para consolidar los requerimientos exactos. El resultado de esta reunión fue un documento formal (ver Informe de Actualizaciones del 7 de Julio de 2025) que sirvió como hoja de ruta.

El trabajo se ejecutó de manera metódica, abordando cada página según lo planificado. El proceso técnico implicó el acceso al panel de administración de WordPress y el uso del editor visual (Elementor) para aplicar los cambios.

Página de Inicio:

- **Actualización de Texto:** Se localizó el módulo de texto correspondiente y se actualizó el lema para reflejar "más de 25 años" de experiencia.
- **Corrección Visual:** La corrección de los íconos rotos requirió una investigación más profunda. Se determinó que el theme de WordPress había sido actualizado en algún momento, pero una dependencia (un *plugin* o una librería de íconos personalizada) no se actualizó con él. Esto provocaba que el CSS intentara llamar a clases de íconos que ya no existían o tenían un nombre diferente. La solución implicó la edición del CSS personalizado del sitio para forzar la carga de la librería correcta y re-mapear las clases de los íconos afectados.

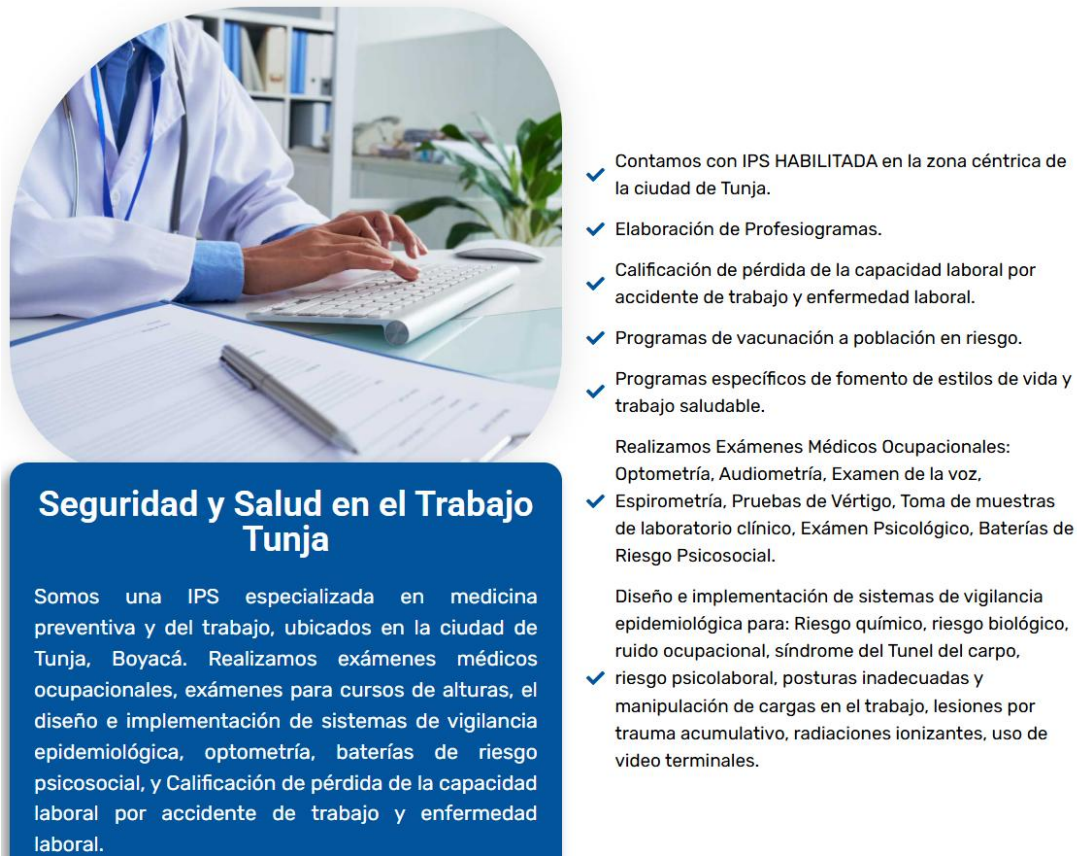
Página "Nosotros":

- En línea con el cambio anterior, se actualizó el texto en esta sección a "más de 25 años", asegurando la coherencia de la marca en todo el sitio.

Página "Medicina Preventiva" (Sección de Servicios): Esta página presentaba dos problemas: desactualización y diseño deficiente.

- **Alineación Visual:** Los listados de servicios utilizaban íconos de check (✅) que estaban desalineados verticalmente con el texto, dificultando la legibilidad. Se accedió al editor de CSS del *theme* y se aplicó una regla de CSS para corregirlo al selector de la lista.
- **Inclusión de Nuevos Servicios:** Se añadieron los servicios "Programas de vacunación" y "Toma de muestras de laboratorio clínico". Esto implicó duplicar la estructura de un servicio existente (imagen, título, descripción) y poblarla con la nueva información proporcionada por la empresa, asegurando que el nuevo bloque respetara el diseño de la cuadrícula existente.

Figura 4. Captura de pantalla de los iconos de checkbox alineados y servicios agregados



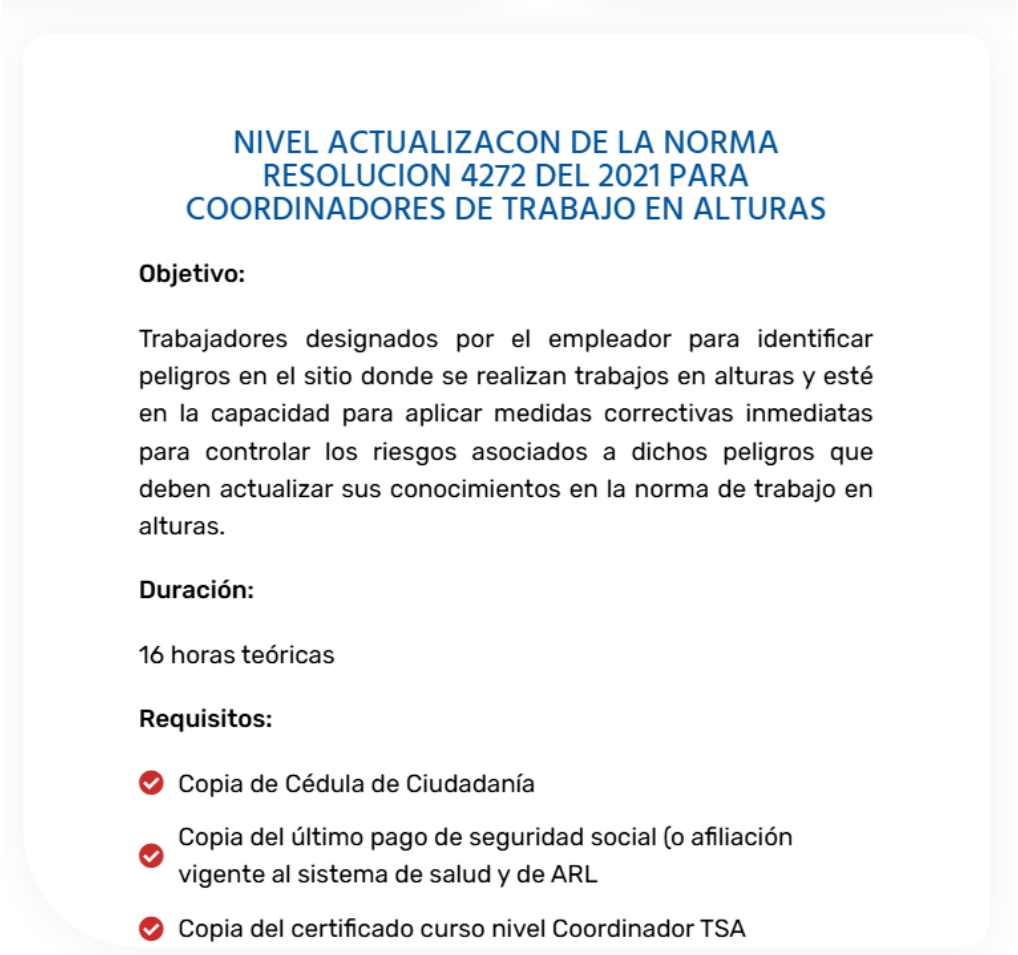
Fuente: Sitio web de Grupo Prevenso.

Como se evidencia en la (Figura 4), se logró la corrección de la alineación defectuosa de los íconos y se permitió la integración visualmente coherente de los nuevos servicios ofertados.

Página "Alturas" (Sección de Servicios): A diferencia de la sección anterior, esta requería la *eliminación* de contenido obsoleto.

- Se accedió al curso "Nivel Coordinador" y se eliminaron dos ítems de requisitos que ya no eran vigentes: "experiencia 50 horas" y "certificado curso 50 h".
- Más importante aún, se eliminó por completo la sección informativa del "Nivel de actualización de coordinador", ya que este servicio fue descontinuado por la empresa.

Figura 5. Captura de pantalla del servicio eliminado



Fuente: Sitio web de Grupo Prevenso.

Página "Contáctanos":

- **Adición de Información:** Se añadió un nuevo sub-apartado "Servicio de IPS" con los números de teléfono 310-337-7634 y 320-855-8707. (Figura 6)
- **Mejora de Visibilidad de Redes Sociales:** El requerimiento era "hacer las redes sociales más notorias". La solución fue instalar y configurar un *plugin* de WordPress especializado en "Social Media Icons". Se configuró dicho *plugin* para mostrar los íconos de TikTok y Facebook de forma prominente en esta página, enlazando a los perfiles oficiales de la empresa.

Figura 6. Captura de pantalla del apartado de contacto actualizado

Fuente: Sitio web de Grupo Prevenso.

4.2.1.3 Fase 4: Detección de Conflicto de Dependencia (Semana 3)

Una vez completadas todas las actualizaciones y verificadas visualmente, se procedió a marcar la tarea como finalizada. Sin embargo, pocas horas después, el personal de Grupo Prevenso reportó que una funcionalidad interna crítica había dejado de operar: el sistema de generación de certificados.

Se inició inmediatamente una fase de diagnóstico de emergencia. Al revisar los *logs* del servidor (logs de errores de Apache y PHP), se detectó una serie de "Errores Fatales" de PHP.

El análisis reveló una dependencia crítica que no había sido informada ni documentada previamente: el sistema de certificados era un desarrollo personalizado o un plugin antiguo que requería obligatoriamente que el servidor operara con la versión de PHP 7.3.

Las actualizaciones de plugins y del theme de WordPress, realizadas para asegurar la compatibilidad y seguridad (necesarias para arreglar los íconos y añadir las nuevas funciones de redes sociales), habían actualizado las dependencias a versiones que requerían PHP 8.0 o superior.

- **PHP 7.3:** Fundamental para los certificados, aun así, no es compatible con los plugins más recientes.
- **PHP 8.0+:** Indispensable para los plugins de hoy, aunque esto genera incompatibilidades con el sistema de certificados.

Esto representó la primera gran lección de la pasantía: el peligro de la deuda técnica. La empresa operaba sobre una infraestructura obsoleta (PHP 7.3 es una versión que alcanzó el fin de su vida útil en 2021) que impedía la modernización.

4.2.1.4 Fase 5: Gestión de Crisis y Restauración del Servicio (Semana 3)

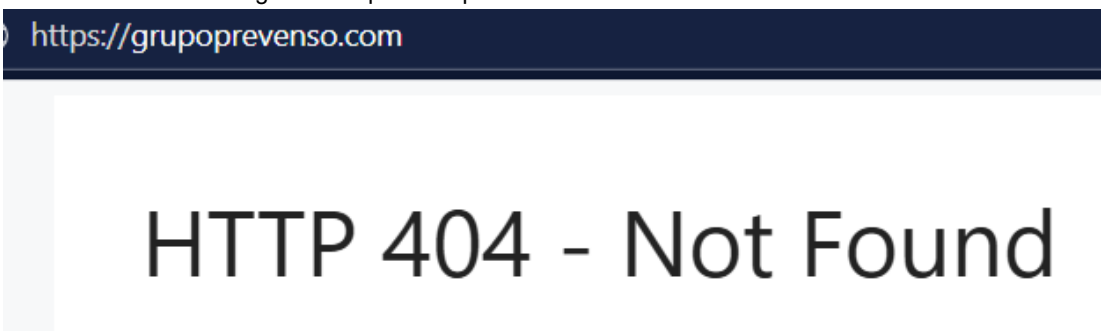
Ante la importancia vital de utilizar el sistema de certificados, se tomó la decisión ejecutiva de revertir todas las actualizaciones implementadas y devolver el sitio a su estado anterior.

Se procedió a acceder al cPanel del proveedor de hosting para utilizar la herramienta de restauración de copias de seguridad. Se seleccionó la copia de seguridad más reciente disponible antes de las modificaciones, correspondiente al 2 de julio de 2025.

Falla de Infraestructura del Hosting: Durante la ejecución del proceso de restauración automática, la herramienta proporcionada por el proveedor de hosting falló. El proceso se interrumpió a la mitad, dejando el sitio en un estado corrupto e inaccesible.

El sitio web y todos los servicios asociados, incluyendo el correo electrónico corporativo (@grupopevenso.com), quedaron completamente inoperativos, mostrando un error "HTTP 404 - Not Found" (Figura 7). El servicio web de la empresa estaba, en efecto, totalmente fuera de servicio.

Figura 7. Captura de pantalla del sitio web fuera de servicio



Fuente: Sitio web de Grupo Prevenso.

La situación escaló de un mantenimiento de rutina a una emergencia de infraestructura. Se contactó de inmediato al soporte técnico del proveedor de hosting, escalando el ticket a máxima. Se informó que su herramienta de restauración automática había fallado y destruido el sitio en producción.

El proveedor asumió la responsabilidad y, tras varias horas de tensión, realizó una restauración manual completa desde sus propias copias de seguridad internas (snapshots del servidor).

4.2.1.5 Conclusión y Re-ejecución del Objetivo (Semana 4)

Con el sitio restaurado a su estado original (con PHP 7.3), se adoptó un "enfoque quirúrgico". En lugar de actualizar plugins o el theme, se procedió a realizar todas las ediciones de texto y contenido (Páginas Inicio, Nosotros, Medicina Preventiva, Alturas, Contacto) utilizando únicamente el editor básico de WordPress, sin tocar ninguna dependencia de software.

El proceso técnico fue minucioso. En lugar de utilizar los constructores visuales que dependen de plugins (y que, al estar desactualizados, presentaban riesgos), se optó por editar el contenido directamente en el modo "HTML" o "Texto" del editor de WordPress siempre que fue posible. Para las secciones donde el constructor visual era ineludible, se trabajó con extrema precaución, modificando únicamente los módulos de texto sin aplicar ninguna actualización a los plugins que el sistema sugería.

El problema de los íconos rotos y la adición de redes sociales (que requerían *plugins* actualizados o la carga de nuevas librerías) se pospuso. Se catalogó como una mejora, la

cual estaría condicionada a la migración o actualización completa del sistema de certificados por parte de la empresa.

A finales de la semana 4, todos los cambios textuales y de contenido solicitados en el informe del 7 de julio estaban implementados en el sitio en producción. Aunque las mejoras visuales más complejas no pudieron implementarse debido a la fragilidad técnica del entorno, el objetivo de actualizar el contenido textual y mejorar la coherencia informativa se logró con éxito.

Esta primera fase concluyó con lecciones valiosas sobre la gestión de deuda técnica, la importancia de los entornos de desarrollo y prueba que se volverían fundamentales para el desarrollo del cotizador y la gestión de crisis bajo presión.

4.2.1.6 Análisis Post-Incidente y Lecciones Aprendidas (Semana 4)

La resolución del incidente crítico durante la Semana 3 no fue simplemente un retorno al estado anterior; constituyó una fuente fundamental de aprendizaje que redefinió la estrategia técnica para los siguientes objetivos del proyecto.

1. Análisis Causa Raíz: El análisis determinó dos causas raíz independientes que convergieron en el incidente:

- **Causa Raíz Técnica (Interna):** La existencia de deuda técnica crítica en forma de una dependencia de software no documentada (el sistema de certificados) anclada a una versión de PHP obsoleta y sin soporte (PHP 7.3). Esto creó un entorno de desarrollo frágil donde cualquier intento de modernización o parcheo de seguridad podía desencadenar un fallo catastrófico.
- **Causa Raíz Operacional (Externa):** La falla de la herramienta de restauración automatizada del proveedor de hosting. Esto reveló un punto único de fallo en la estrategia de recuperación ante desastres de la empresa; se confiaba ciegamente en una herramienta de terceros sobre la cual no se tenía control.

2. Análisis de Impacto al Negocio: El impacto de la caída del servicio fue clasificado como de "Alta Severidad".

- **Paralización de Operaciones:** La caída del servicio de correo electrónico (@grupopevenso.com) detuvo las comunicaciones formales con clientes, proveedores y personal interno.

- **Pérdida de Oportunidad Comercial:** La inaccesibilidad del sitio web (error 404) impidió que nuevos clientes potenciales descubrieran servicios o realizaran consultas.
- **Riesgo Reputacional:** Un sitio web inoperativo proyecta una imagen de inestabilidad técnica que puede afectar la confianza del cliente.

3. Justificación de la Arquitectura Futura: Este incidente no fue un contratiempo, sino un argumento contundente que validó el enfoque propuesto para el desarrollo de la nueva aplicación de cotizaciones:

- **La Necesidad de Entornos de Staging:** La lección más importante fue la absoluta necesidad de un entorno de pruebas. Quedó establecido que cualquier desarrollo futuro, comenzando por el cotizador, se realizaría en un subdominio o servidor de desarrollo completamente aislado, garantizando que ninguna prueba o despliegue pudiera afectar la operación en producción.
- **El Principio de Desacoplamiento:** El incidente demostró el peligro de construir una nueva aplicación (el cotizador) sobre la misma infraestructura fallida (el WordPress con PHP 7.3). Esto justificó plenamente la decisión de arquitectura (Objetivo 2) de desarrollar el cotizador como un sistema totalmente desacoplado, con su propia base de datos (PostgreSQL) y su propio backend (Node.js), en lugar de intentar hacerlo como un plugin de WordPress.
- **Estrategia de Respaldo Independiente:** Se recomendó a Grupo Prevenso implementar una política de copias de seguridad redundantes e independientes del proveedor de hosting (copias de seguridad descargadas y almacenadas en un servicio de nube externo).

4.2.2. Diseño de la Arquitectura de Software y Base de Datos

4.2.2.1 Introducción: La Necesidad Estratégica del Desacoplamiento

Las lecciones extraídas mientras se completaba el Objetivo 1 demostraron no ser solo un pequeño revés. Al contrario, Fueron el más potente argumento técnico, que realmente moldeó el diseño de la nueva aplicación. El problema con la versión de PHP, la dependencia no completamente explicada del sistema de certificados y esa fallida herramienta de restauración del hosting revelaron con dureza los peligros de una arquitectura de software tan intrincadamente unida.

En el sistema previo, todo, los componentes (interfaz de usuario, lógica del negocio, el contenido y las dependencias del servidor), estaban muy fusionados, a tal grado que una falla hasta en el plugin más insignificante, o la más mínima actualización en la infraestructura (como PHP) fácilmente podría desatar un fallo desastroso en el sistema completo, incluyendo servicios clave, como el correo electrónico.

Así, el principal pilar que guio el diseño para el Objetivo 2 fue la decisión de adoptar una arquitectura desacoplada. Este enfoque del diseño del software plantea que cada componente esencial del sistema (la interfaz de usuario o Frontend, la lógica del negocio o Backend y el almacenamiento de datos o Base de Datos) funcione como un servicio autónomo e independiente.

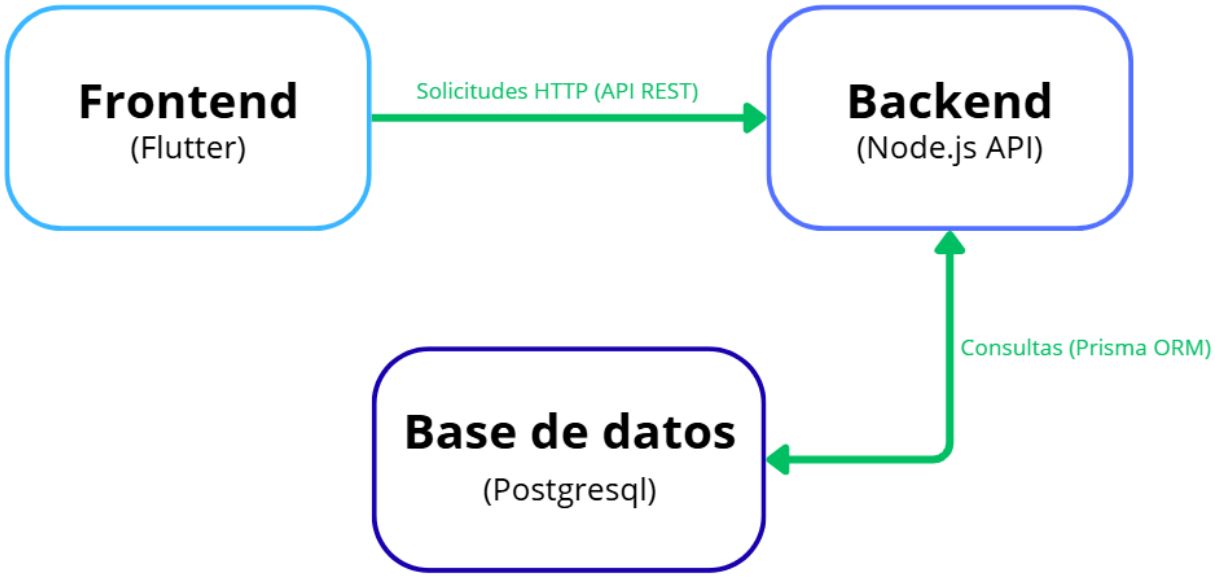
Estos servicios se comunican entre sí, valiéndose de interfaces claramente definidas, pero sin estar atados a cómo funcionan internamente los demás.

Las ventajas estratégicas que esta estrategia ofreció al proyecto fueron bien notables:

1. **Eficiencia Operativa:** A diferencia del incidente de WordPress, si la aplicación de escritorio (Frontend) fallara, el Backend y la Base de Datos seguirían operativos, sin afectar al resto de los usuarios.
2. **Mantenibilidad y Evolución Independiente:** Se podrían actualizar las tecnologías de cada componente por separado. Por ejemplo, sería posible migrar el Backend a una nueva versión de Node.js o incluso a un lenguaje diferente, sin que el Frontend de Flutter se vea afectado, siempre que el vínculo con la API se respete.
3. **Escalabilidad Selectiva:** Permite optimizar recursos escalando solo los componentes que lo necesitan. Si la aplicación experimenta un alto volumen de consultas, se puede escalar únicamente el servidor de la base de datos (PostgreSQL) sin necesidad de modificar el Frontend.
4. **Flexibilidad a Futuro:** El mismo Backend (API) diseñado para servir a la aplicación Flutter podría, en el futuro, servir a una aplicación web, integrarse con el sitio de WordPress o conectarse a otros sistemas de gestión que la empresa decida implementar, maximizando el retorno de la inversión del desarrollo.

La fase de diseño, por tanto, no consistió solo en elegir tecnologías, sino en definir un ecosistema de componentes resilientes e independientes.

Figura 8. Diagrama de Arquitectura Desacoplada de la Aplicación de Cotizaciones



Fuente: Autor.

4.2.2.2 *Fase de Planificación y Selección del "Stack" Tecnológico*

Con la estructura de desacoplamiento establecida (Figura 8), el siguiente paso, ejecutado durante las primeras semanas de julio, fue seleccionar el stack tecnológico óptimo para cada uno de estos componentes. La selección se basó rigurosamente en los requisitos del proyecto: una aplicación multiplataforma (móvil y escritorio), robustez e integridad de datos, y la capacidad de desarrollo ágil.

4.2.2.3 *Justificación de PostgreSQL (Base de Datos)*

Aun la infraestructura actual de Grupo Prevenso, que usaba MySQL (por el hosting de cPanel de WordPress) se tomó la decisión utilizar PostgreSQL para la nueva aplicación. Esta estrategia se apoyó en muchas razones técnicas, más importantes qué usar lo ya conocido:

1. **Robustez y Cumplimiento ACID:** PostgreSQL es bastante popular porque obedece estrictamente las propiedades ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad). Para la app, que tocará temas financieros (cotizaciones, precios, totales), esa certeza de qué las transacciones se terminen o no inicien es clave para cuidar la información.

2. **Un trato Superior de Lógica Compleja:** PostgreSQL sobresale, manejando lógica de negocios compleja. Da tipos de datos buenos (como JSONB, rangos y tipos geométricos) y funciones de ventana más poderosas y procedimientos guardados bien hechos, lo cual, hace una base mucho más resistente para después (como para módulos de facturación o inventario, por ejemplo).
3. **Escalabilidad y Rendimiento:** PostgreSQL es excelente porque ofrece un desempeño increíble en situaciones con mucha gente y datos grandes. De hecho, a menudo vence a MySQL al leer y escribir datos complicados, tal como uno esperaría en un sistema así.
4. **Ecosistema de Código Abierto:** Al ser de código abierto y tener una comunidad muy activa, nos garantiza que va a durar mucho, con seguridad constante y sin estar atado a una sola empresa.

4.2.2.4 Justificación de Node.js, Express y TypeScript (Backend)

Para crear el Backend, el "cerebro" que maneja la lógica y la API, se optó por una pila moderna basada en JavaScript.

1. **Node.js:** Se escogió este entorno por su forma de trabajar con entradas y salidas que no bloquean y son asíncronas. Esto es ideal para crear APIs, porque un solo hilo puede gestionar miles de conexiones a la vez, es decir, muchos usuarios pidiendo cosas, sin colapsar. Así, el servidor gasta menos recursos y responde super rápido.
2. **TypeScript:** En lugar de simplemente usar JavaScript, se decidio implementar TypeScript. Para un proyecto de grado y una aplicación empresarial, esta decisión fue fundamental. TypeScript, al ser un superconjunto de JavaScript con tipado estático, permite detectar una gran categoría de errores en tiempo de compilación (antes de que el código llegue a producción), facilita la refactorización a gran escala, mejora drásticamente la legibilidad del código y hace que el autocompletado en el IDE sea mucho más inteligente. Esto reduce las horas de debugging y aumenta la mantenibilidad a largo plazo.
3. **Express.js:** Se seleccionó como el *framework* web para construir la API sobre Node.js. Al ser bastante flexible permitió construir una API RESTful robusta, siguiendo exactamente los patrones de diseño definidos (la arquitectura en capas) sin sobrecargar el proyecto con características innecesarias.

4.2.2.5 Justificación de Prisma (ORM)

Para actuar como el "traductor" entre la lógica de negocio en TypeScript (Node.js) y la base de datos relacional (PostgreSQL), se eligió Prisma como ORM (*Object-Relational Mapper*). El ORM permite interactuar con la base de datos usando objetos y métodos de TypeScript (ej. prisma.cliente.create(...)) en lugar de escribir consultas SQL manualmente (ej. INSERT INTO Clientes...).

La elección de Prisma sobre otros ORMs fue por distintas razones como:

1. **Seguridad de Tipos:** Prisma es un ORM moderno diseñado explícitamente para TypeScript. Genera automáticamente tipos de TypeScript basados en el esquema de la base de datos, proporcionando un autocompletado y una seguridad de tipos excepcionales que previenen errores comunes de consultas como lo sería consultar una columna inexistente.
2. **Abstracción de Base de Datos:** Facilita la interacción con PostgreSQL, pero su esquema declarado en el proyecto (un archivo schema.prisma) otorga flexibilidad a la hora de migrar a otra base de datos (como MySQL o SQL Server) en el futuro con mínimos cambios en el código de la lógica de negocio.
3. **Gestión de Migraciones Integrada:** Prisma incluye un sistema robusto y declarativo para gestionar las migraciones de la base de datos, lo que permite evolucionar el esquema de datos (añadir tablas, modificar columnas) de forma controlada y versionada.

4.2.2.6 Justificación de Flutter (Frontend)

El requisito principal del proyecto, especificado por la empresa, era una aplicación que funcionara tanto en dispositivos móviles (para el personal en campo) como en computadores de escritorio (para el personal administrativo en la oficina). Flutter fue la elección tecnológica más eficiente para cumplir este requisito.

1. **Código Base Único:** Flutter permite con un solo código base (escrito en lenguaje Dart), compilar aplicaciones nativas de alto rendimiento para múltiples plataformas: iOS, Android, Windows, macOS y Linux. Esto reduce el tiempo de desarrollo y mantenimiento a una fracción de lo que costaría desarrollar y mantener dos, tres o más aplicaciones separadas (una para iOS, una para Android, una para Windows).

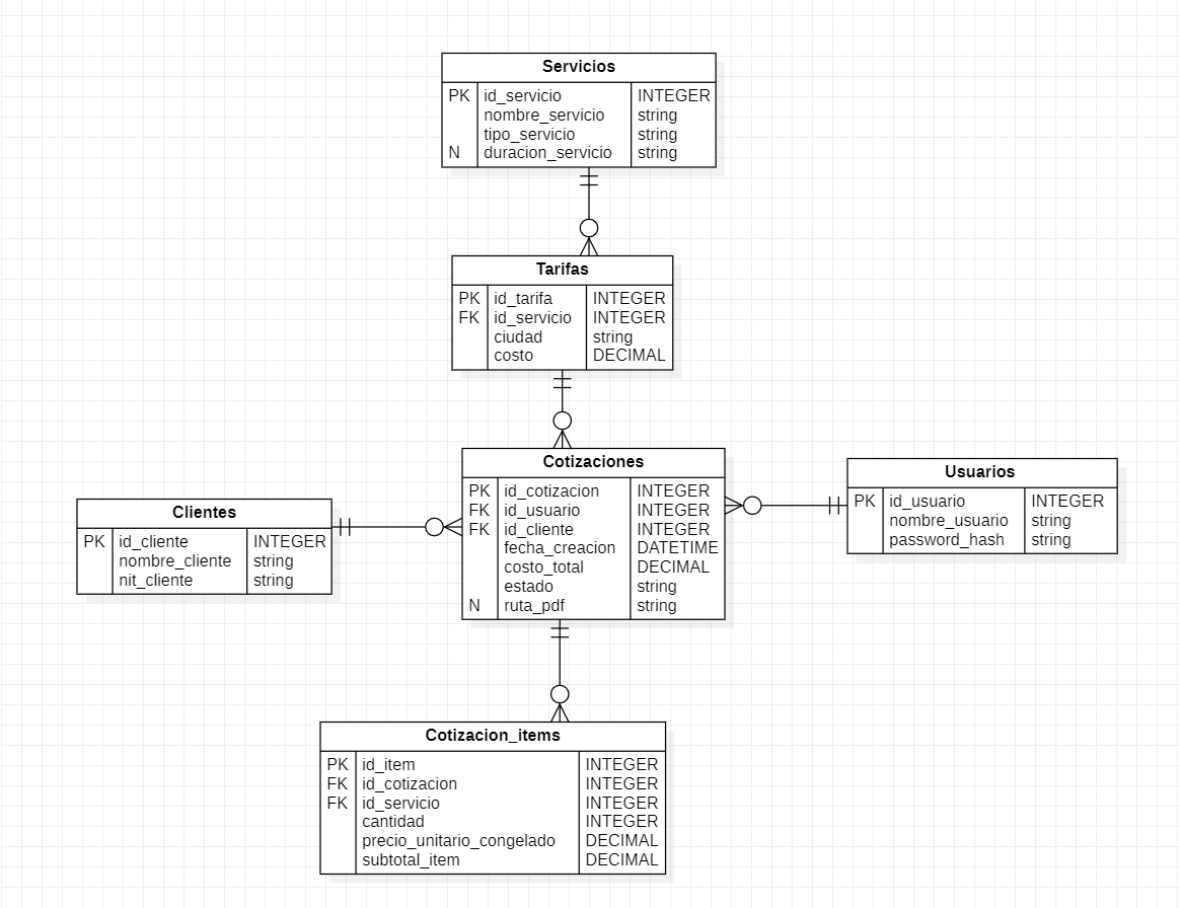
2. **Rendimiento Nativo:** A diferencia de otras soluciones híbridas que se ejecutan en un WebView, Flutter compila directamente a código máquina y dibuja su propia interfaz de usuario usando el motor gráfico Skia. Esto ofrece un rendimiento fluido indistinguible del de una aplicación nativa.
3. **Interfaz de Usuario Consistente:** Permite un control total (con una precisión de cada píxel) sobre la interfaz de usuario, asegurando que la aplicación, sus componentes y la identidad de marca de Grupo Prevenso se vean y se sientan exactamente igual en todas las plataformas.

4.2.2.7 Diseño de la Base de Datos Normalizada (PostgreSQL)

Tal como se menciona en el objetivo, no bastaba con elegir PostgreSQL; la base de datos debía estar normalizada. La normalización es un proceso de diseño de bases de datos que organiza los datos en tablas para minimizar la redundancia y mejorar la integridad de los datos, evitando anomalías de inserción, actualización o borrado.

Se diseñó e implementó un Esquema Entidad-Relación (ERD) que consta de seis tablas principales, logrando un diseño que cumple con la Tercera Forma Normal (3NF).

Figura 9. Diagrama Relacional inicial de la Aplicación de Cotizaciones



Fuente: Autor.

A continuación, se detalla el propósito, diseño y justificación de cada entidad dentro del esquema:

Tabla Usuarios

Propósito: Almacena la información del personal de Grupo Prevenso que tendrá acceso autorizado a la aplicación. Es la entidad base para la autenticación y la auditoría.

Campos Clave (según ERD):

- id_usuario (PK): Llave primaria, un identificador numérico único.
- nombre_usuario (string): Nombre del empleado para mostrar en la interfaz.
- password_hash (string): Se diseñó explícitamente para almacenar la contraseña encriptada usando un algoritmo hash seguro (con bcrypt), nunca en texto plano, garantizando el cumplimiento de las buenas prácticas de seguridad.

Relaciones: Uno-a-Muchos con Cotizaciones. Se establece que un usuario puede crear muchas cotizaciones, pero una cotización solo puede ser creada por un usuario.

Tabla Clientes

Propósito: Funciona como el directorio centralizado de los clientes de la empresa a quienes se les emiten las cotizaciones.

Campos Clave (según ERD):

- id_cliente (PK): Llave primaria única.
- nombre_cliente (string): Nombre del cliente o entidad.
- nit_cliente (string): NIT o documento de identificación, indexado para búsquedas rápidas.

Relaciones: Uno-a-Muchos con Cotizaciones. Un cliente puede tener muchas cotizaciones asociadas a lo largo del tiempo.

Tabla Servicios

Propósito: Es el catálogo principal de todos los productos y servicios (cursos, exámenes, capacitaciones) que ofrece Grupo Prevenso. Sirve como el único punto de información para la definición de servicios.

Campos Clave (según ERD):

- id_servicio (PK): Llave primaria única.
- nombre_servicio (string): Nombre descriptivo (ej. "Curso de Alturas Nivel Coordinador").
- tipo_servicio (string): Una categoría para filtrar (ej. "Curso", "Examen Médico", "Capacitación").
- duracion_servicio (string): Metadato del servicio (ej. "8 horas", "N/A").

Relaciones:

- Uno-a-Muchos con Tarifas.
- Uno-a-Muchos con Cotizacion_Items.

Tabla Tarifas

Propósito: Esta tabla es una pieza clave de la normalización. Desacopla el precio de un servicio de la definición del servicio en sí. Esto permite que un mismo servicio (ej.

"Examen de Optometría") tenga precios diferentes según la ciudad (ej. Tunja, Bogotá, Duitama), sin necesidad de duplicar el registro del servicio en la tabla Servicios.

Campos Clave (según ERD):

- id_tarifa (PK): Llave primaria única.
- id_servicio (FK): Llave foránea que referencia a Servicios.
- ciudad (string): El contexto que define el precio (ej. "Tunja").
- costo (decimal): El valor monetario del servicio en esa ciudad.

Relaciones: Muchos-a-Uno con Servicios.

4.2.3.5 Tabla Cotizaciones

Propósito: Almacena el "encabezado" o la información maestra de cada cotización generada. Contiene la información general del documento.

Campos Clave (según ERD):

- id_cotizacion (PK): Llave primaria única.
- id_usuario (FK): Referencia a quién creó la cotización.
- id_cliente (FK): Referencia a para quién es la cotización.
- fecha_creacion (datetime): Marca de tiempo de cuándo se generó, para auditoría y reportes.
- costo_total (decimal): El valor total calculado de la cotización, almacenado para consultas rápidas.
- estado (string): Estado del ciclo de vida del documento (ej. "Borrador", "Enviada", "Aprobada", "Rechazada").
- ruta_pdf (string, nullable): (Diseño a futuro) Campo para almacenar la ubicación del archivo PDF generado.

Relaciones:

- Muchos-a-Uno con Usuarios.
- Muchos-a-Uno con Clientes.
- Uno-a-Muchos con Cotizacion_Items.

Tabla Cotizacion_Items

Propósito: Esta es una tabla de unión que conecta Cotizaciones y Servicios. Es la pieza central que permite que una cotización contenga múltiples servicios. En lugar de

almacenar una lista o un JSON en la tabla Cotizaciones, se crea un registro en esta tabla por cada servicio añadido a la cotización.

Campos Clave (según ERD):

- id_item (PK): Llave primaria única del ítem.
- id_cotizacion (FK): A qué cotización pertenece este ítem.
- id_servicio (FK): Qué servicio se está cotizando.
- cantidad (integer): Cuántas unidades de este servicio.
- precio_unitario_congelado (decimal): Este es un campo de diseño vital. Almacena el precio al momento en que se generó la cotización. Esto es fundamental para la integridad histórica; si Grupo Prevenso decide subir el precio de un servicio en la tabla Tarifas en el futuro, las cotizaciones antiguas mantendrán el precio con el que fueron emitidas, evitando inconsistencias contables.
- subtotal_item (decimal): Campo calculado (cantidad * precio_unitario_congelado) para facilitar los cálculos.

Relaciones:

- Muchos-a-Uno con Cotizaciones.
- Muchos-a-Uno con Servicios.

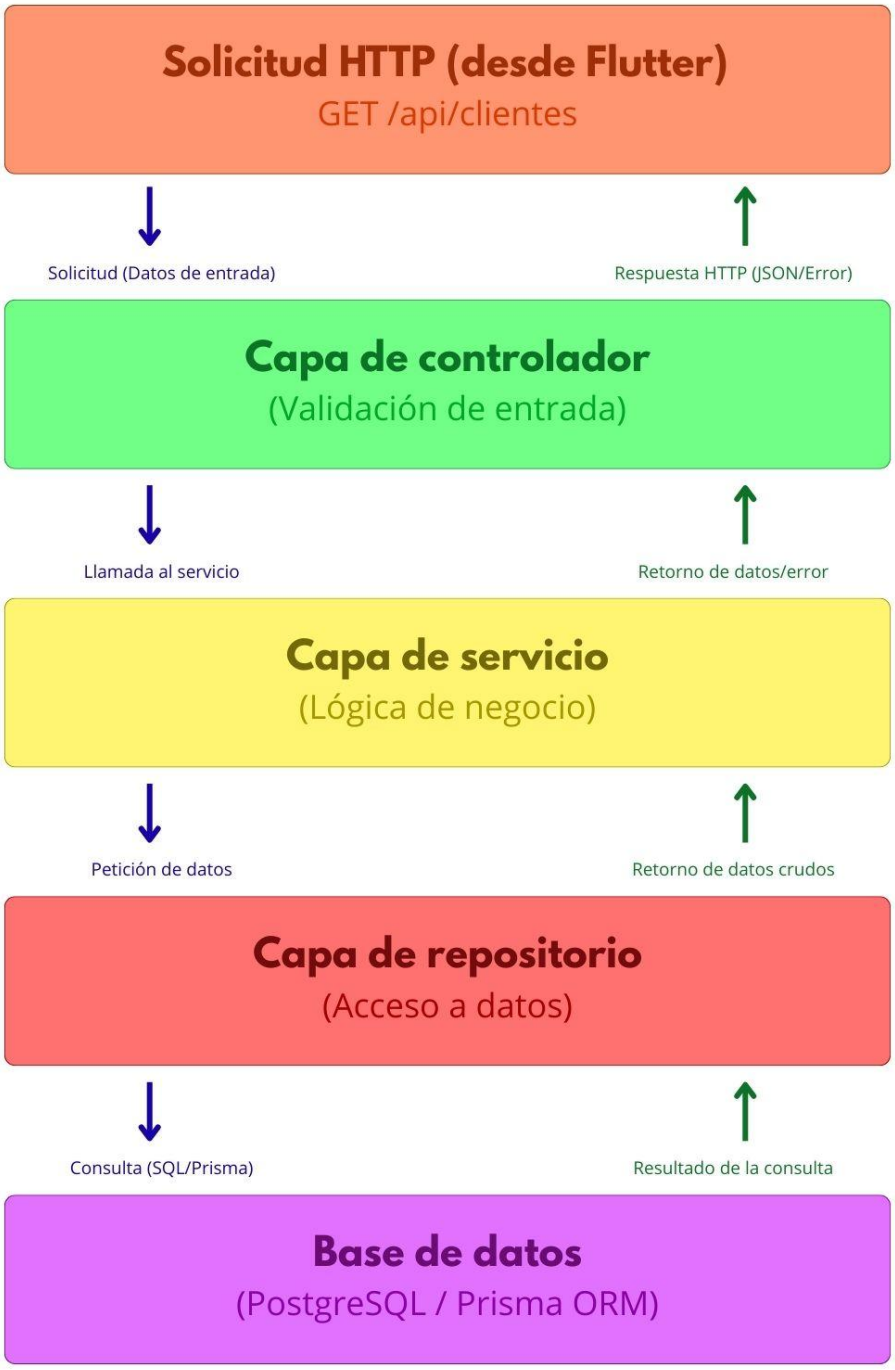
Este diseño de seis tablas cumple con la Tercera Forma Normal (3NF), elimina la redundancia de datos y garantiza que las dependencias sean lógicas, proveyendo una base de datos robusta y normalizada como lo exigía el objetivo.

4.2.2.8 Diseño de la Arquitectura Backend (Patrón en Capas)

Con la base de datos diseñada, el siguiente paso fue definir la arquitectura del "cerebro" de la aplicación: el backend. El objetivo exigía explícitamente definir las capas: **(Controller, Service, Repository)**.

Este patrón de arquitectura fue la base para organizar la lógica de Node.js. Su propósito es la Separación de Responsabilidades, un principio fundamental de la ingeniería de software que dicta que un sistema debe ser dividido en partes con responsabilidades distintas y bien definidas.

Figura 10. Diagrama de Flujo de la Arquitectura en Capas del Backend



Fuente: Autor.

4.2.2.9 Capa de Controladores (Controller)

- **Responsabilidad:** Es la única capa que interactúa directamente con el mundo exterior (la aplicación Flutter).
- **Funciones:**
  - Recibir las solicitudes HTTP (ej. POST /api/clientes).
  - Validar la sintaxis de los datos de entrada como por puede ser que el campo nit_cliente no esté vacío o que costo sea un número.
  - Llamar a la capa de servicio correspondiente para que ejecute la acción.
  - Recibir la respuesta (datos o error) del servicio y restablecerla como una respuesta HTTP estándar (ej. 201 Created con el nuevo cliente, o 400 Bad Request con un mensaje de error).
- **Restricción de Diseño:** Esta capa tiene prohibido contener lógica de negocio (ej. calcular un total) o comunicarse directamente con la base de datos.

4.2.2.10 Capa de Servicios (Service)

- **Responsabilidad:** Ser la mente de la aplicación. Aquí reside toda la lógica de negocio y las reglas específicas de Grupo Prevenso.
- **Funciones:**
  - Recibir la llamada del Controlador (ej. crearCliente(datosCliente)).
  - Dirigir la lógica: verificar si un cliente con ese NIT ya existe, formatear datos, etc.
  - Llamar a una o más capas de repositorio para obtener o persistir datos.
  - Devolver una respuesta de negocio (un objeto o un error) al Controlador.
- **Ejemplo (Módulo de Cotizaciones):** CotizacionService sería responsable de recibir una lista de ítems, buscar los precios actuales en Tarifas (llamando al TarifaRepository), calcular los subtotales, aplicar descuentos (si la lógica lo requiere), sumar el costo_total, y finalmente llamar al CotizacionRepository para guardar la cotización y sus Cotizacion_Items dentro de una transacción de base de datos.

4.2.2.11 Capa de Repositorios (Repository)

- **Responsabilidad:** Ser las "manos" de la aplicación. Es la única capa que tiene permitido interactuar directamente con la fuente de datos que en este caso son Prisma y PostgreSQL.
- **Funciones:**
  - Abstraer las consultas a la base de datos, proveyendo métodos simples para el servicio.
  - Recibir llamadas del Servicio (ej. crearNuevoCliente(datos)).
  - Ejecutar la consulta de Prisma correspondiente (ej. prisma.cliente.findUnique(...) o prisma.cliente.create(...)).
  - Devolver los datos puros de la base de datos al Servicio.
- **Restricción de Diseño:** Esta capa tiene prohibido contener lógica de negocio. Su única función es leer y escribir datos.

Este diseño en tres capas aseguró que el desarrollo (Objetivo 3) fuera ordenado, mantenible, escalable y fácil de testear, cumpliendo con la definición de una arquitectura robusta.

4.2.2.12 Diseño de la Arquitectura Frontend (Patrón Feature-First)

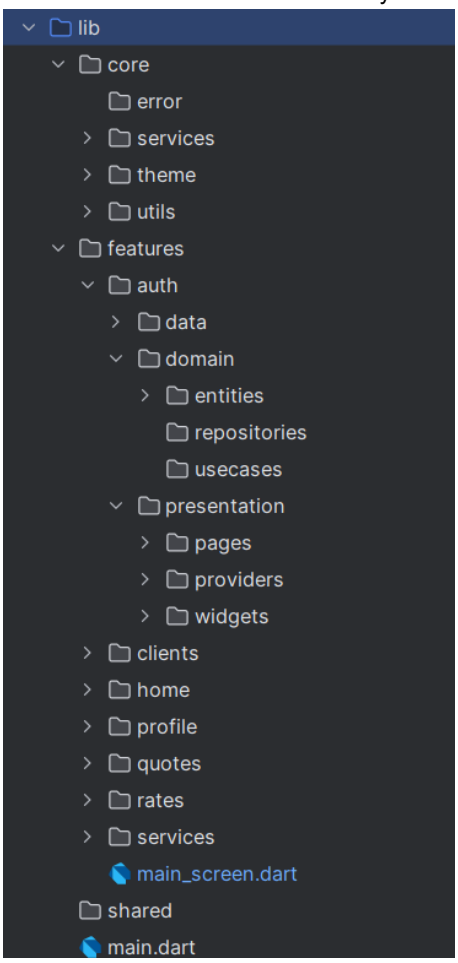
Finalmente, el objetivo requería definir la estructura del frontend en Flutter. Para un proyecto destinado a crecer y ser mantenido, una estructura de carpetas profesional es esencial. El proyecto se configuró en IntelliJ IDEA siguiendo el patrón "Feature-First" (primero la funcionalidad).

Este patrón contrasta con los patrones tradicionales como "Type-First", donde el código se agrupa por tipo de archivo (ej. una carpeta /screens para todas las pantallas, una carpeta /widgets para todos los widgets).

En una arquitectura Feature-First:

1. **Organización:** El código se agrupa en directorios que representan las funcionalidades del negocio.
2. **Estructura del Proyecto:** El directorio /lib (el corazón de una aplicación Flutter) se diseñó para verse de la siguiente manera:

Figura 11. Estructura de Directorios del Proyecto Frontend



Fuente: Autor.

3. **Ventajas del Diseño:**
    - **Escalabilidad:** Añadir una nueva funcionalidad como facturación solo requiere añadir una nueva carpeta /facturacion en /features, sin tocar las existentes.
    - **Mantenibilidad:** Facilita encontrar y corregir errores. Si hay un bug en la creación de clientes, todo el código relevante (pantalla, widgets, lógica) está contenido dentro de la carpeta /clients.

4.2.2.13 Diseño de la Experiencia de Usuario (UX) y Prototipado de Interfaz (UI)

Con la arquitectura de datos (ERD), la arquitectura de backend (Capas) y la arquitectura de frontend (Feature-First) definidas, el paso final de la fase de diseño consistió en establecer el aspecto de la aplicación.

El objetivo de la pasantía no era solo crear un sistema funcional, sino diseñar una Experiencia de Usuario (UX) que fuera intuitiva, rápida y que resolviera directamente las dificultades identificadas en el proceso manual de cotización de Grupo Prevenso. Un sistema técnicamente robusto pero difícil de usar no sería adoptado por el personal.

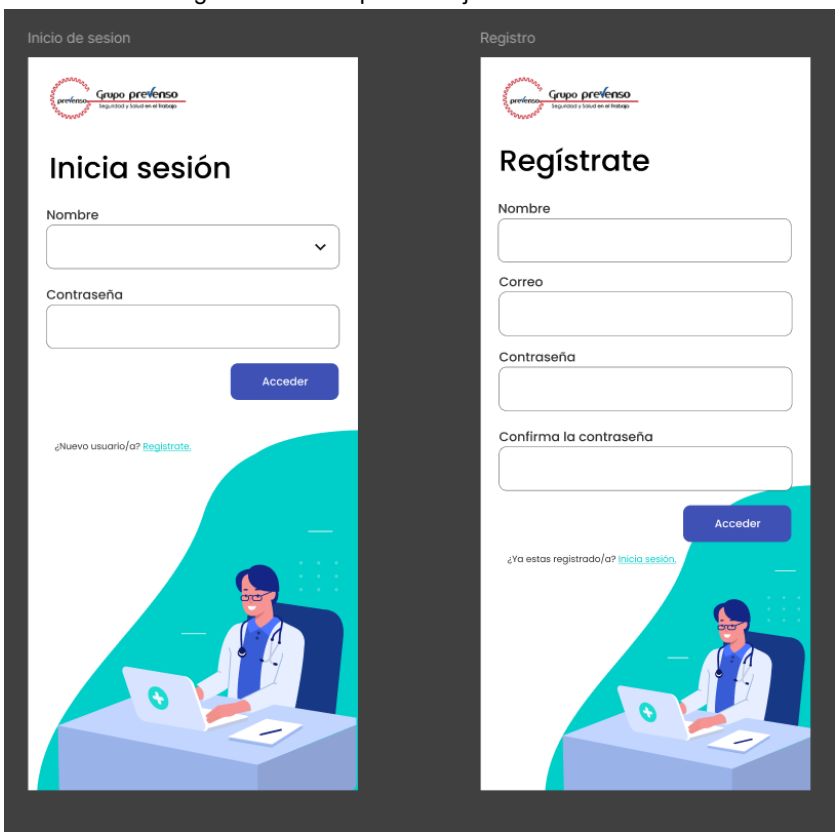
Para este fin, se utilizó la herramienta de diseño de interfaces Figma. Esta plataforma permitió crear prototipos de alta fidelidad (conocidos como mockups) que simulan de manera precisa la apariencia, los componentes visuales y el flujo de navegación de la aplicación final.

Estos prototipos funcionaron como soporte visual y un fuerte impulso de validación con la tutora empresarial. Diversas iteraciones de diseño fueron creadas para asegurar que cada pantalla fuese lógica y fácil de comprender antes de que la primera línea de código Flutter fuera escrita. Esto minimizó el riesgo de crear funciones que no coincidieran con las expectativas del usuario.

1. **Proceso de Autenticación y Acceso**

   Se refiere a las pantallas para iniciar sesión, el registro de nuevos usuarios, y el trámite de recuperar la contraseña. El diseño está hecho para ser simple e inspirado de otras aplicaciones actuales, para no afectar la experiencia al usuario.

Figura 12. Mockups del Flujo de Autenticación



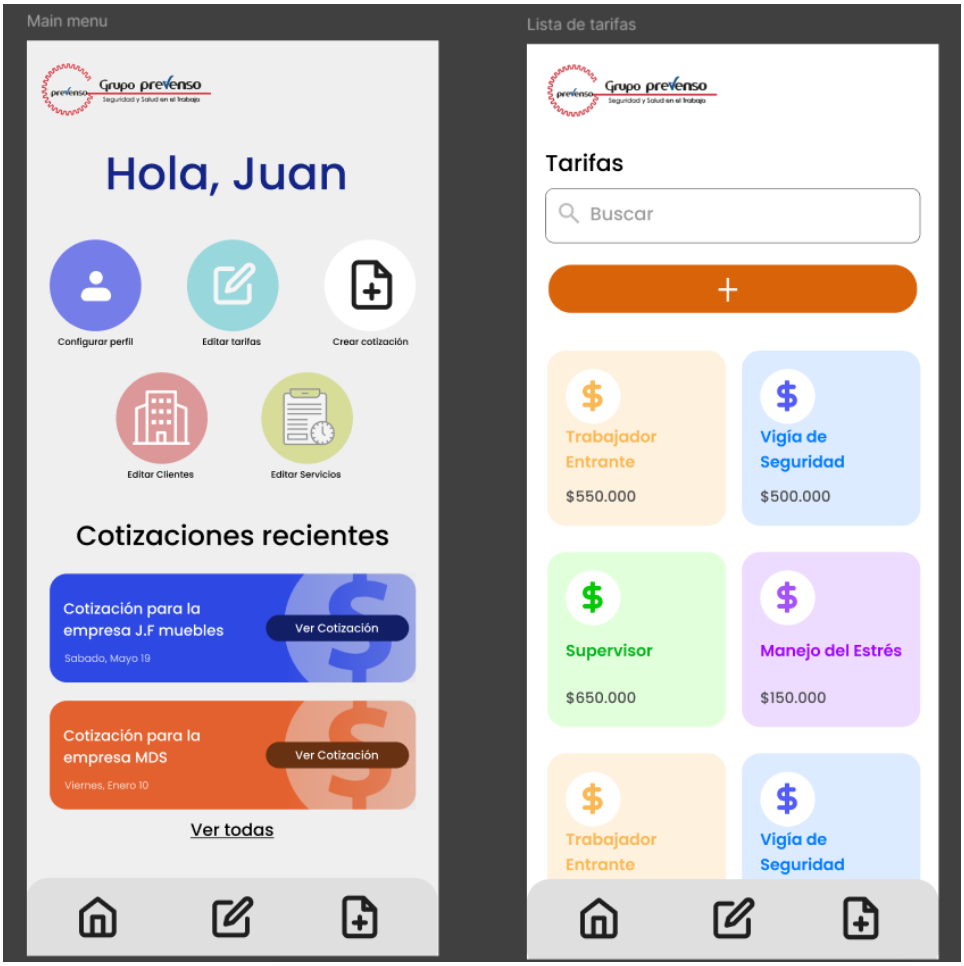
Fuente: Autor.

2. Pantalla Principal y Gestión de Tarifas

La pantalla principal (ver Figura 13) se diseñó como un dashboard centralizado. Saluda al usuario por su nombre y presenta accesos directos a las funciones clave (Cotizaciones, Clientes, Servicios), así como un resumen de cotizaciones recientes. Este diseño permite al usuario navegar a las secciones más importantes con un solo toque.

Junto a este flujo, se diseñó la gestión de "Tarifas" (Figura 13), una sección de configuración vital que permite a los administradores definir los precios de los servicios por ciudad, tal como se especificó en el diseño de la base de datos.

Figura 13. Mockups de la Pantalla Principal (Dashboard) y el Flujo de Gestión de Tarifas.

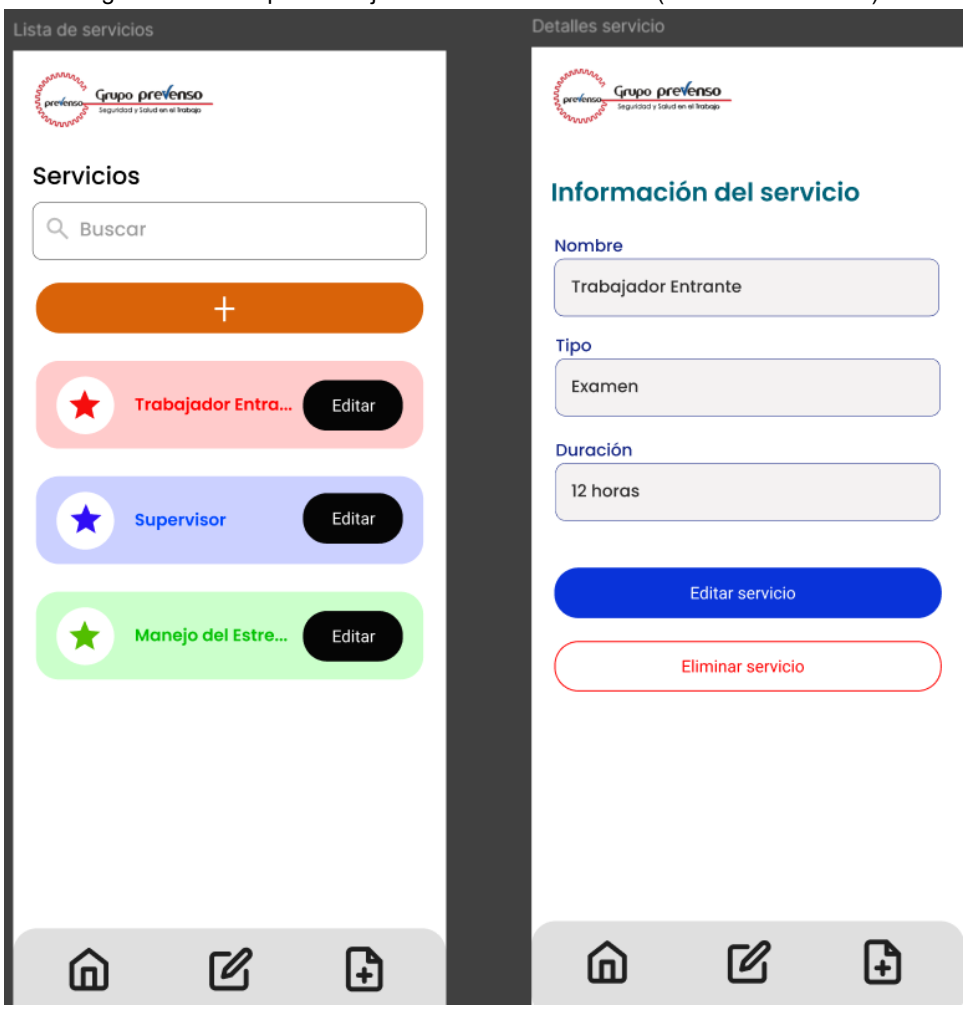


Fuente: Autor.

3. Flujo de Gestión de Servicios

Este flujo (Figura 13) permite al personal administrativo gestionar el catálogo de servicios de la empresa (cursos, exámenes, etc.). Incluye una pantalla de "Lista de servicios" con capacidad de búsqueda y filtrado, un formulario de "Crear servicio" y vistas de "Detalles" y "Edición". El diseño modular asegura que añadir o modificar un servicio sea una tarea atómica y libre de errores.

Figura 14. Mockups del Flujo de Gestión de Servicios (CRUD de Servicios).

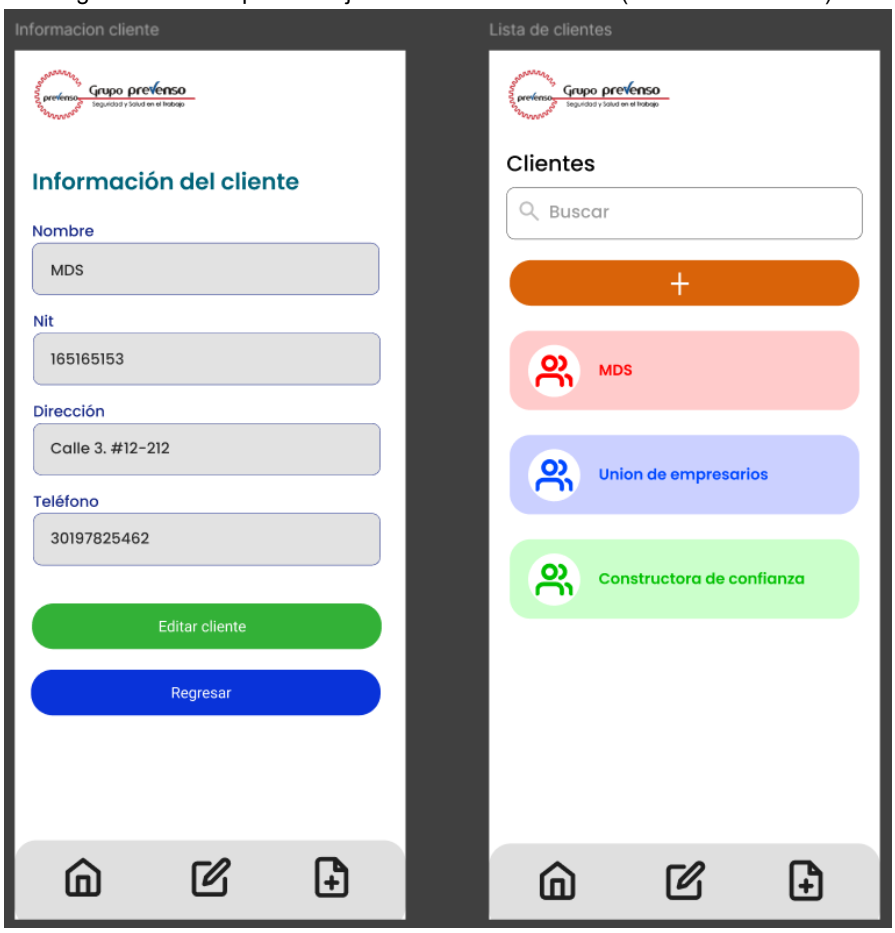


Fuente: Autor.

4. Flujo de Gestión de Clientes

De manera similar al flujo de servicios, la gestión de clientes (Figura 15) centraliza el CRM de la empresa. Permite listar, buscar, crear y editar la información de los clientes. Este diseño es fundamental para eliminar la redundancia de datos y los errores de digitación presentes en el proceso manual, donde los datos del cliente se escribían a mano en cada cotización.

Figura 15. Mockups del Flujo de Gestión de Clientes (CRUD de Clientes).



Fuente: Autor.

5. Flujo Principal: Creación de Cotizaciones

Este es el flujo más importante de la aplicación (Figura 16). Resuelve el problema central de la pasantía. El diseño guía al usuario paso a paso:

Lista de Cotizaciones: Muestra un historial de cotizaciones recientes.

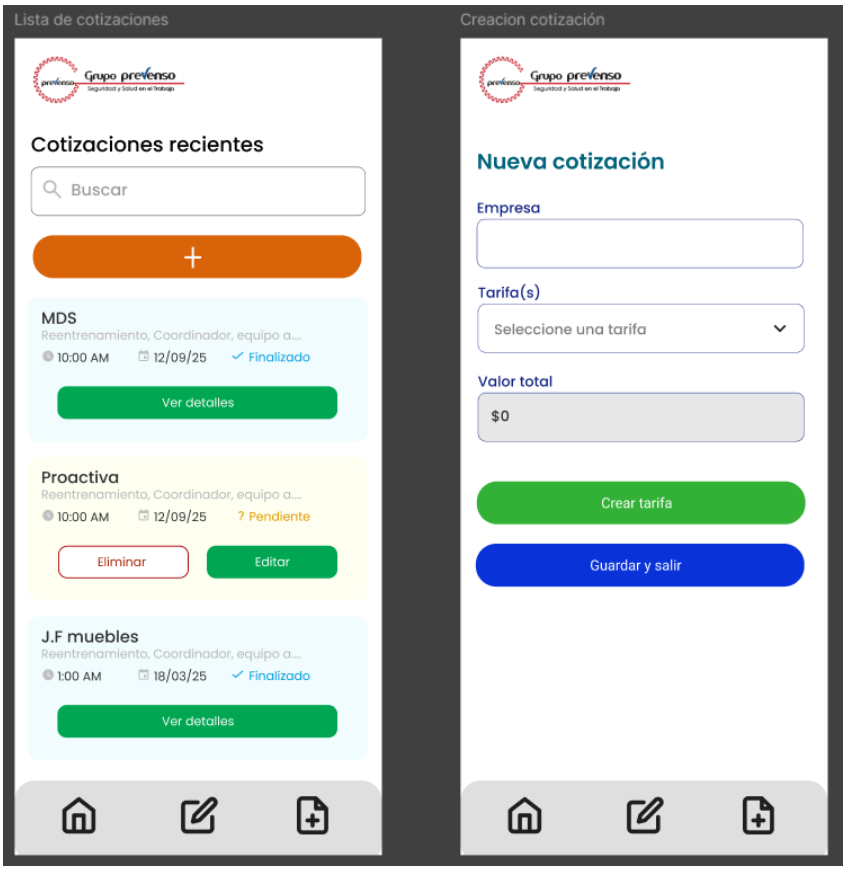
Creación Cotización: El usuario selecciona un cliente (del CRUD de clientes) y una empresa (si aplica).

Detalles Cotización: El usuario añade dinámicamente servicios (del CRUD de servicios), especifica cantidades y descuentos. El sistema calcula automáticamente los subtotales y el total, eliminando el 100% del cálculo manual.

Cotización Finalizada: Muestra un resumen de la cotización generada, lista para ser exportada a PDF.

Este diseño de formulario guiado transforma un proceso manual y propenso a errores (en Excel) en un flujo digital rápido, preciso y trazable.

Figura 16. Mockups del Flujo Principal de Creación de Cotizaciones.



Fuente: Autor.

6. Conclusión de la Fase de Diseño

Al finalizar esta fase, se había completado íntegramente el Objetivo Específico 2. Se contaba con un conjunto de planos detallados para cada componente del ecosistema de software:

- **Plano de Datos:** El ERD de PostgreSQL (Figura 9).
- **Plano de Backend:** La arquitectura en capas (Figura 10).
- **Plano de Código Frontend:** La estructura Feature-First (Figura 11).
- **Plano Visual y de UX:** Los mockups de Figma por funcionalidad (Figura 12 a la 16).

Este grupo de diseños aseguró un fundamento fuerte y un estímulo técnico definido para el inicio de la etapa de implementación y desenvolvimiento del código, especificado en los objetivos siguientes.

4.2.3. Implementación de la API RESTful

4.2.3.1 Introducción de la Arquitectura a la Implementación

Con los diseños arquitectónicos y de diseño mostrado en el Objetivo específico 2, la pasantía dio inició a toda la construcción del backend. Esta fase se desplegó principalmente durante el mes de julio y se enfocó en transformar todos los diagramas en un servicio de software funcional, protegido, y sólido.

Para guiar esta construcción, se adoptó el estilo arquitectónico REST (Representational State Transfer). Según (Fielding, 2000), una API RESTful se define como "una interfaz de programación de aplicaciones que se adhiere a las restricciones de arquitectura REST e interactúa con servicios web RESTful Esta aproximación aseguró la escalabilidad y la simplicidad necesarias para la comunicación entre el frontend y el backend.

Es importante destacar que, con la metodología ágil, la implementación no fue una simple línea recta desde el principio. Durante las reuniones semanales con la tutora de la empresa, surgieron nuevos requerimientos y se ajustaron distintas necesidades. Esto provocó cambios en el diseño, agregando lógica de negocio más compleja y cosas que no se habían planeado al inicio.

Esta sección explica como implementamos la API RESTful, pero no como algo fijo, sino un proceso en constante cambio, guiado por la retroalimentación del cliente. Esto permitió crear un producto final más útil y valioso.

4.2.3.2 Configuración del Entorno de Desarrollo

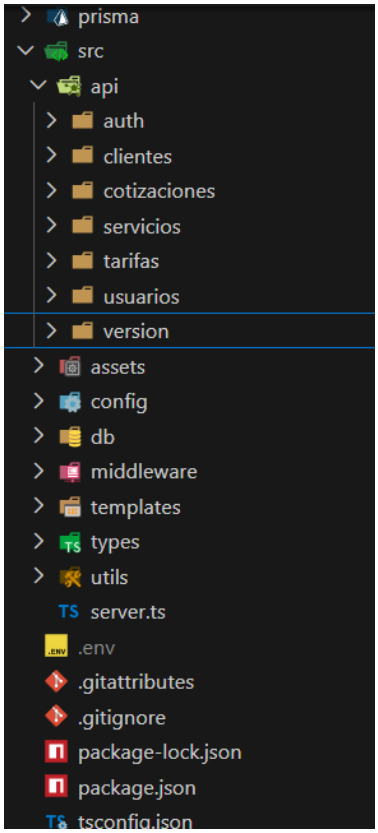
El primer paso era crear un entorno de desarrollo profesional, a nivel local, para que el proyecto pudiera mantenerse y crecer.

1. **Inicialización del Proyecto:** Se creó el directorio del proyecto y se inicializó un proyecto de Node.js.
2. **Adopción de TypeScript:** Se instalaron TypeScript y sus tipos de Node. js. Se configuró el archivo tsconfig. json, para definir las reglas del compilador, la carpeta de salida, el módulo, entre otras cosas.

3. Para el desarrollo local, fue ts-node-dev lo que se escogio, una herramienta muy útil con la cual se reinicia el servidor automáticamente cuando el código cambia, reduciendo drásticamente el proceso de depuración.
4. **Instalación de Express.js:** Se instaló el framework Express para gestionar el enrutamiento y las peticiones HTTP.
5. **Conexión a la Base de Datos con Prisma:** Se inicializó el cliente Prisma, y de ahí se creó el archivo schema.prisma. Este archivo es el unico referente y guia de la estructura de la base de datos, y se actualizó para que refleje el ERD inicial.

6. **Control de Versiones (Git y GitHub):** Se inicializó un repositorio local de Git. Se creó un repositorio privado en GitHub para resguardar el código fuente. Un paso crítico fue la creación de un archivo ".gitignore" robusto para excluir la carpeta node_modules, los archivos de entorno (.env) y los archivos compilados de JavaScript, asegurando que solo el código fuente relevante fuera versionado.

Figura 17. Estructura de Directorios del Proyecto Backend



Fuente: Autor.

4.2.3.3 Adaptación del Modelo de Datos

Apenas se comenzó a implementar el CRUD para las entidades básicas, las primeras sesiones de validación con la tutora empresarial revelaron necesidades más profundas que el modelo de datos inicial (Figura 9) no satisfacía. Esto llevó a la primera y más significativa evolución del proyecto.

1. **La Necesidad de un CRM Funcional**

- **Solicitud:** La tutora señaló que para que las cotizaciones fueran documentos verdaderamente útiles, el sistema no solo debía almacenar el nombre_cliente y nit_cliente sino también el email, teléfono y dirección en la ficha del cliente para que sean visibles en la cotización.
- **Impacto Técnico:** Modificación de la tabla Clientes para incluir los nuevos campos. Esto también implicaba actualizar la capa de validación en el ClienteController para asegurar que el email_cliente tuviera un formato válido.

2. **El Flujo de Recuperación de Contraseña**

- **Solicitud:** Aunque el diseño inicial incluía un inicio de sesión, se identificó un problema importante: si un empleado olvidaba su contraseña, el proceso de recuperación dependía de modificarla manualmente desde la base de datos, lo cual no era una solución viable. Por ello, se solicitó implementar un mecanismo adecuado para la gestión de contraseñas.
- **Impacto Técnico:** Esta fue una evolución funcional mayor. Se determinó la necesidad de un flujo de "Olvide mi contraseña". Esto requirió:
    1. Añadir un campo email a la tabla Usuarios, que no estaba en el diseño original.
    2. Diseñar un mecanismo seguro para los tokens de reinicio. Se creó una nueva tabla, PasswordResetToken, para almacenar un token temporal, su fecha de expiración y el userId al que estaba asociado. Esto es significativamente más seguro que almacenar el token en la misma tabla Usuarios.

3. **Lógica de Descuentos Compleja**

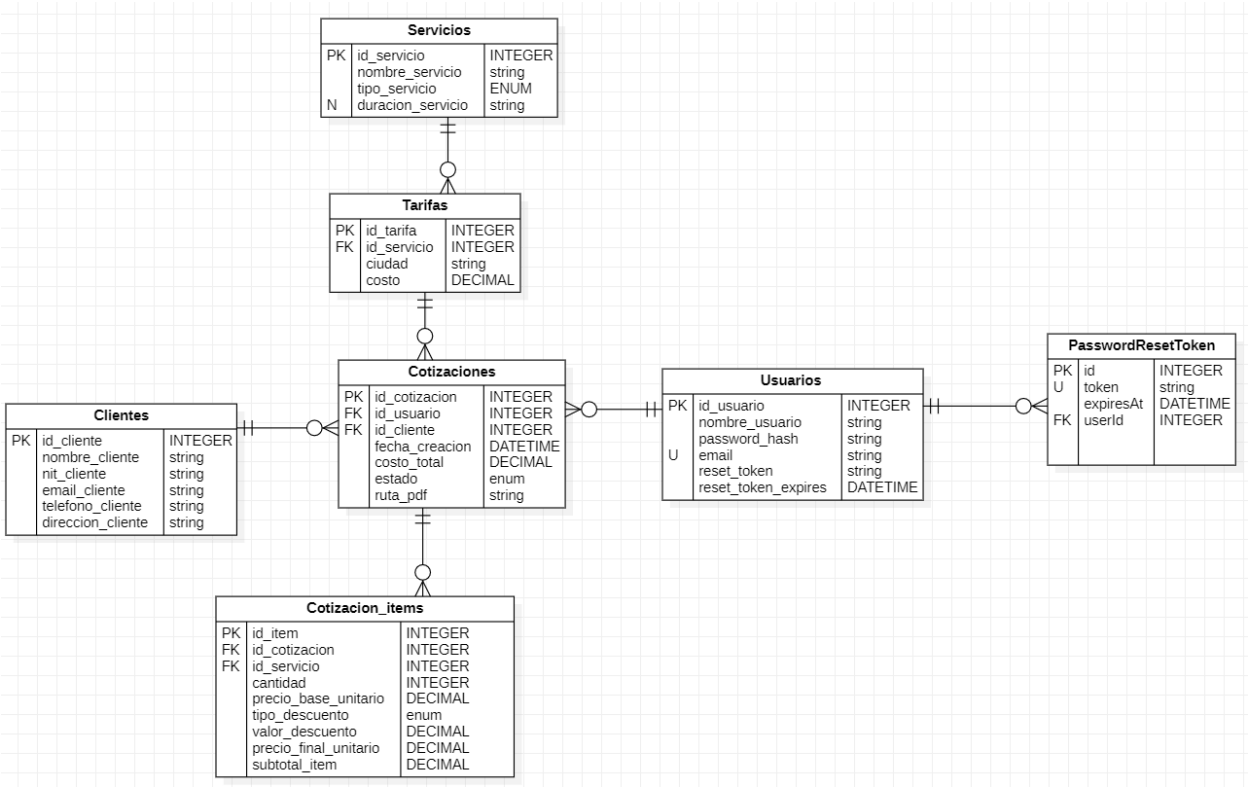
- **Solicitud:** Esta fue la solicitud de lógica de negocio más relevante. El diseño inicial de la tabla *Cotizacion_Items* solo incluía un precio unitario fijo y una cantidad. Sin embargo, se identificó que en Prevenso los descuentos no siguen un

único formato: en algunos casos se aplican porcentajes y en otros valores fijos sobre el precio base. Por ello, se requería que el sistema permitiera manejar ambos tipos de descuento.

- **Impacto Técnico:** Esta solicitud redefinió por completo la lógica del módulo de cotizaciones. La tabla Cotizacion_Items fue rediseñada para incluir:
  - precio_base_unitario: El precio original del servicio (antes precio_unitario_congelado).
  - tipo_descuento: Un campo ENUM (con valores NINGUNO, PORCENTAJE, FIJO).
  - valor_descuento: El número a aplicar (ej. 10 o 20000).
  - precio_final_unitario: El precio calculado después de aplicar el descuento.
  - subtotal_item: Calculado como precio_final_unitario * cantidad.

Tras estas sesiones de retroalimentación, el modelo de datos adoptó su forma final, la cual se usó para la implementación definitiva.

Figura 18. Diagrama Relacional Final



Fuente: Autor.

Este diagrama (Figura 18) muestra la evolución del proyecto con un proceso ágil, donde la estructura de datos se ajustó a tiempo para representar con mayor precisión las necesidades del negocio. Un ejemplo claro es el ajuste en duracion_servicio, configurado como opcional para adaptarse a la naturaleza de los exámenes médicos. Esta precisión en el modelado, realizada antes de que la implementación estuviera demasiado avanzada, ahorró incontables horas de refactorización futura. Con el modelo final aprobado y migrado a la base de datos, se procedió a la implementación de los endpoints.

4.2.3.4 Implementación del Módulo de Seguridad y Autenticación

Siguiendo la arquitectura en capas, se comenzó por el módulo más crítico: la seguridad. Una API "segura" requiere tres componentes: encriptación de contraseñas, autenticación (login) y autorización (protección de rutas).

1. **Encriptación de Contraseñas (bcrypt)**

Se instaló la librería bcrypt y se creó un módulo de utilidades con dos funciones: hashPassword (usada en el registro) y comparePassword (usada en el login). Se decidió usar un saltRounds de 10, que ofrece un balance estándar entre seguridad y rendimiento. Nunca se almacenaron contraseñas en texto plano.

2. **Flujo de Autenticación (JWT)**

Se implementaron los *endpoints* de autenticación siguiendo el patrón de 3 capas:

- **Registro:**
  - **Controller:** Recibe y valida el nombre_usuario, email y password.
  - **Service:** Llama a hashPassword y luego al Repository para crear el usuario.
  - **Repository:** Ejecuta la creación del registro.
- **Login:**
  - **Controller:** Recibe email y password.
  - **Service:** Busca al usuario por email. Si existe, usa comparePassword para verificar la contraseña. Si la verificación es exitosa, se instala jsonwebtoken y se genera un JSON Web Token (JWT). Este token contiene información no sensible como el id del usuario y está firmado con un secreto almacenado en el archivo .env.
  - **Controller:** Devuelve el JWT al cliente.

3. **Middleware de Autorización**

Este fue un componente clave para la seguridad. Se creó un middleware de Express.

- **Función:** Este middleware se añade a todas las rutas que requieren protección.
- **Operación:**
  - Revisa el encabezado Authorization de la solicitud.
  - Extrae el token.
  - Usa jwt.verify para validar el token contra el JWT_SECRET.
  - Si es válido, extrae el id_usuario del token y lo adjunta al objeto request.
  - Permite que la solicitud continúe a la capa de Controlador.
  - Si el token no es válido o no existe, rechaza la solicitud con un error 401 Unauthorized.

4.2.3.5 Implementación de los Módulos de Gestión (CRUDs)

Con la seguridad implementada, se procedió a la implementación de los endpoints para gestionar las entidades del negocio (Clientes, Servicios, Tarifas).

Estas rutas siguieron un patrón estándar de 3 capas, todas protegidas por el authMiddleware:

- GET /api/clientes: (Listar Clientes)
- POST /api/clientes: (Crear Cliente)
- PUT /api/clientes/:id: (Actualizar Cliente)
- DELETE /api/clientes/:id: (Eliminar Cliente)
- (Rutas análogas para /api/servicios y /api/tarifas)

En esta fase se implementaron validaciones de entrada robustas en la capa de Controlador, asegurando que datos malformados como un costo no numérico en una Tarifa nunca llegaran a la capa de servicio o base de datos.

4.2.3.6 Implementación del Módulo de Cotizaciones

Sin duda, este módulo resultó ser el más complejo, albergando la lógica del negocio, y por supuesto, la innovadora implementación de descuentos.

1. **Endpoint de Creación**

   Este endpoint no era un simple CRUD. Debía manejar una lógica de negocio compleja y ejecutarla de forma transaccional.

- **Controller:** Recibía desde el frontend un JSON con una estructura complicada, comprendiendo el id_cliente, asi como un conjunto de elementos, incluyendo el id_servicio, la cantidad, el tipo_descuento, además del valor_descuento.
- **Service (CotizacionService):** El servicio empezaba determinando el costo general de la cotización, después preparaba un listado interno donde se almacenarían los elementos que debían ser persistidos. Luego recorre cada uno de los ítems enviados en la solicitud y, para cada uno, consulta el precio base del servicio correspondiente a través del TarifaRepository. Esta consulta tiene en cuenta tanto el tipo de servicio como la ciudad del usuario, información que proviene del propio token de autenticación.

  Después entra en juego la lógica de descuentos, una de las funcionalidades clave del proyecto. En el sistema se selecciona si el descuento debe aplicarse como porcentaje o como un valor fijo, calcula el precio final unitario y luego obtiene el subtotal del ítem multiplicando dicho precio por la cantidad solicitada. Ese subtotal se suma al costo total general y se arma el objeto final del ítem, asegurando que queden guardados en la base de datos los valores “congelados” que corresponden a ese momento de la cotización.

  Finalmente, cuando todos los cálculos están listos, el servicio envía la información al repositorio y realiza el guardado mediante una transacción para mantener la integridad de los datos. Así se garantiza que todo el proceso sea consistente, preciso y robusto.

2. **Transacciones de Base de Datos (ACID)**

   Un aspecto clave de este módulo era garantizar la atomicidad de todo el proceso. Si el sistema llegara a guardar únicamente la cotización, pero fallara al registrar los ítems asociados, se correría el riesgo de generar información incompleta o inconsistente dentro de la base de datos.

   Para evitar este tipo de errores, se implementaron transacciones interactivas con Prisma mediante prisma.$transaction. Con esto, todas las operaciones relacionadas se ejecutan como una sola unidad: si alguna parte falla, nada se guarda. De esta manera, se asegura que la cotización y sus ítems siempre queden registrados de forma correcta y coherente.

4.2.3.7 Conclusión de la Fase de Backend

Al finalizar esta fase se obtuvo como resultado una API RESTful:

- **Completa:** Ofrecía *endpoints* para todas las entidades del negocio (Usuarios, Clientes, Servicios, Tarifas, Cotizaciones).
- **Segura:** Protegida por encriptación (bcrypt), autenticación (JWT) y autorización (middleware).
- **Robusta:** Construida sobre una arquitectura en capas, con validaciones estrictas y transacciones de base de datos para garantizar la integridad de los datos.
- **Adaptada al Negocio:** La API no solo seguía el plan original, sino que había mejorado para incorporar la lógica de negocio compleja (descuentos, recuperación de contraseña) solicitada por la empresa, demostrando la capacidad de adaptación del proyecto.

El backend fue desplegado exitosamente en Render, quedando preparado para el uso del cliente frontend, desarrollado en Flutter.

4.2.4. Implementación de la Interfaz Multiplataforma

4.2.4.1 Introducción: De los Prototipos a la Realidad

El logro de este objetivo concluyó el ciclo total del desarrollo, desde la planificación inicial hasta la entrega del producto a los usuarios finales. Esta fase se concentró en aplicar lo planeado meses atrás, mudando los prototipos de Figma, la arquitectura creada, y la lógica de negocio del backend en una aplicación funcional que usarían los empleados del Grupo Prevenso.

Este proceso requirió la sincronización de muchos componentes, entre ellos, la comunicación con la API, el manejo de sesiones seguras, realizar pruebas exhaustivas y resolver errores imprevistos, junto a optimizar el flujo de trabajo empresarial. En resumidas cuentas, esta fase consolidó la integración completa del ecosistema del software.

4.2.4.2 Fase 1: Construcción de la Interfaz y Lógica de Cliente

Inicialmente, se transformaron los prototipos de Figma (Figura 12 a 16) en vistas funcionales de Flutter, con el fin de conectarlas a la API.

Para este desarrollo se utilizó Flutter, definido como "un kit de desarrollo de software (SDK) de interfaz de usuario de código abierto creado por Google, utilizado para desarrollar aplicaciones multiplataforma para Android, iOS, Linux, Mac, Windows y la web desde una única base de código" (Google , 2024). Su aptitud multiplataforma resultó clave para satisfacer las exigencias del proyecto.

1. **Configuración del Proyecto**

Se creó un nuevo proyecto de Flutter y se configuró el archivo pubspec.yaml con las dependencias fundamentales para el funcionamiento del sistema:

- provider: gestión de estado declarativa.
- http: comunicación con la API REST.
- flutter_secure_storage: almacenamiento seguro del JWT.
- package_info_plus: lectura de la versión de la aplicación.
- path_provider y open_file: manejo de archivos descargados, especialmente los PDFs de cotización.

Además, se implementó una arquitectura Feature-First (Figura 11), organizando el proyecto en carpetas por funcionalidad (auth, clients, quotes, etc.). Cada módulo tenía su propia lógica y sus interfaces, dando orden y escalabilidad al proyecto.

2. **Implementación del Flujo de Autenticación**

Esta fue la primera prueba real de integración con el backend.

El proceso se desarrolló así:

- Pantalla de Login: Al presionar “Iniciar Sesión”, Flutter enviaba la petición al AuthService.
- ApiService centralizado: Se creó un servicio que envolvía las llamadas HTTP, permitiendo un punto único para manejar errores y encabezados.
- Gestión del JWT: Cuando el backend respondía con éxito, el token se guardaba en flutter_secure_storage.
- Navegación automática: El AuthProvider notificaba a la vista y redirigía al menú principal.
- Peticiones autenticadas: Desde ese momento, todas las solicitudes incluían automáticamente el encabezado Authorization: Bearer <token>.

Este fue el primer paso que conectó completamente Flutter con la lógica real del sistema.

3. **Integración de los Módulos CRUD**

Habiéndose asegurado el acceso al sistema, la integración de los módulos Clientes, Servicios, y Tarifas se ejecutó. La capacidad de búsqueda en tiempo real resultó ser una característica vital. Con un endpoint GET con parámetros variables, se pudo gestionar listas extensas de forma casi instantánea, mejorando la gestión con más rapidez y precisión.

4.2.4.3 Desafío 1: El Cuello de Botella del Generador de PDF

Después de interconectar los CRUDs, el primer gran obstáculo fue la creación del PDF de cotización, algo fundamental para la empresa. No obstante, esta función estaba ligada a la recepción de una plantilla final validada.

1. **Retraso en el Diseño**

El diseño sufrió retrasos, varios días el desarrollo se detuvo, el diseño oficial del documento tardo más tiempo del esperado. Esto evidenció como factores externos podrían impactar en el cronograma, sobre todo cuando el módulo dependía directamente de esa información.

2. **Implementación del Generador (Backend)**

una vez dispuesto el diseño, se empezó la parte técnica más complicada, dentro del backend. Se seleccionó EJS como el motor de plantillas.

- Se eligió EJS como motor de plantillas.
- Se recreó la plantilla del documento en formato HTML dinámico.
- Se utilizó Puppeteer para renderizar la plantilla en un navegador headless.
- Se creó el endpoint.
- El backend generaba el PDF en tiempo real y lo devolvía al cliente.

Este enfoque permitió entregar un documento profesional, consistente y totalmente automatizado.

3. **Resultado Final del PDF**

Este documento final representó con precisión la estética de la corporación, incorporando información variable extraída de la base de datos.

Figura 19. Resultado Final de la Plantilla de Cotización en EJS Renderizada como PDF.

Fuente: Autor.

4.2.4.4 Fase 3: Pruebas, Despliegue Inicial y Desafíos del Mundo Real

Con la aplicación ya operativa en desarrollo el momento de ponerla a prueba en dispositivos reales había llegado. En ese momento, surgieron dos problemas críticos e imprevistos al transitar del ámbito de desarrollo al de producción.

1. **Desafío 1: La Falla del Diseño Responsivo**

Al instalar la app en aparatos con pantallas pequeñas se percibió que la interfaz no se ajustaba adecuadamente:

- Elementos que no cabían en pantalla (RenderFlex overflowed).
- Botones fuera de alcance.
- Formularios imposibles de completar cuando aparecía el teclado.

La solución requirió una reformulación exhaustiva:

- Uso de MediaQuery y porcentajes en lugar de tamaños fijos.

- Adaptación mediante LayoutBuilder.
- Uso de Wrap, Flexible, Expanded y FittedBox para textos y contenedores dinámicos.

Esta imprevista fase se incrementó la robustez de la aplicación, y de igual modo aseguro su correcto desempeño en un amplio rango de dispositivos.

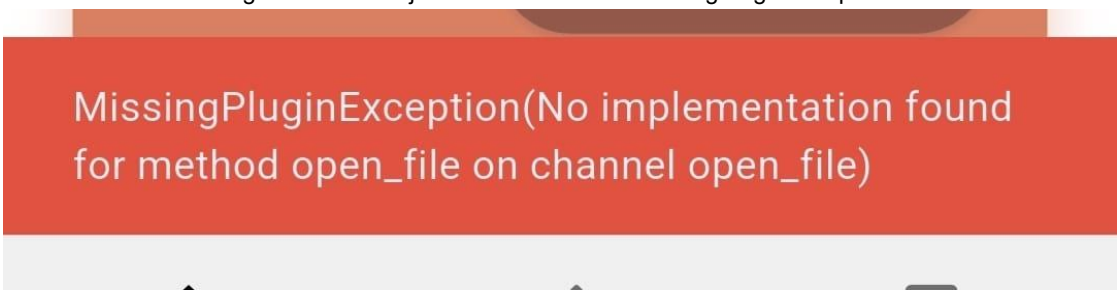
2. **Desafío 2: El "MissingPluginException" y la Falla de Shorebird**

Luego de resolver el diseño responsivo, surgió el mayor desafío técnico. Al intentar implementar actualizaciones mediante Shorebird (una herramienta externa que permitir actualizaciones rápidas sin reinstalar la aplicación los builds de producción dejaron de funcionar correctamente.

En los dispositivos reales:

- El PDF no abría.
- Los plugins nativos no respondían.

Figura 20. Mensaje de Error Crítico de MissingPluginException.



Fuente: Autor.

3. **Diagnóstico y Causa Raíz**

Tras varias jornadas de análisis se detectó que Shorebird interfería con el proceso de build, eliminando accidentalmente implementaciones nativas esenciales para la app.

4. **La Solución: Abandono de Shorebird**

Se decidió eliminarlo completamente:

- Limpieza de referencias en pubspec.yaml.
- Ejecución de flutter clean.
- Construcción del APK usando únicamente las herramientas oficiales de Flutter.

El resultado fue inmediato y la aplicación volvió a funcionar sin errores.

4.2.4.5 Fase 4: Creación de un Sistema de Actualización Personalizado

Al ya no poder utilizar Shorebird, fue necesario crear un sistema propio de actualizaciones.

1. **Diseño del Servicio de Actualización**

El sistema de actualización dentro de la app quedó estructurado así:

- Endpoint en el backend para obtener la versión.
- Comparación entre versión local y remota.
- Diálogo visual solicitando actualizar.
- Descarga del instalador.

2. **Implementación Multiplataforma (Android y Windows)**

- En Android, la app descarga un .apk y el usuario lo instala.
- En Windows, la app descarga un .zip y abre la carpeta de descargas para que el usuario actualice los archivos.

Figura 21. Alerta de actualización

Fuente: Autor.

Este sistema resolvió completamente la necesidad de actualizaciones sin depender de servicios externos.

4.2.4.6 Fase 5: Validación Final, Despliegue y Capacitación

La fase final consistió en validar el producto con los usuarios reales.

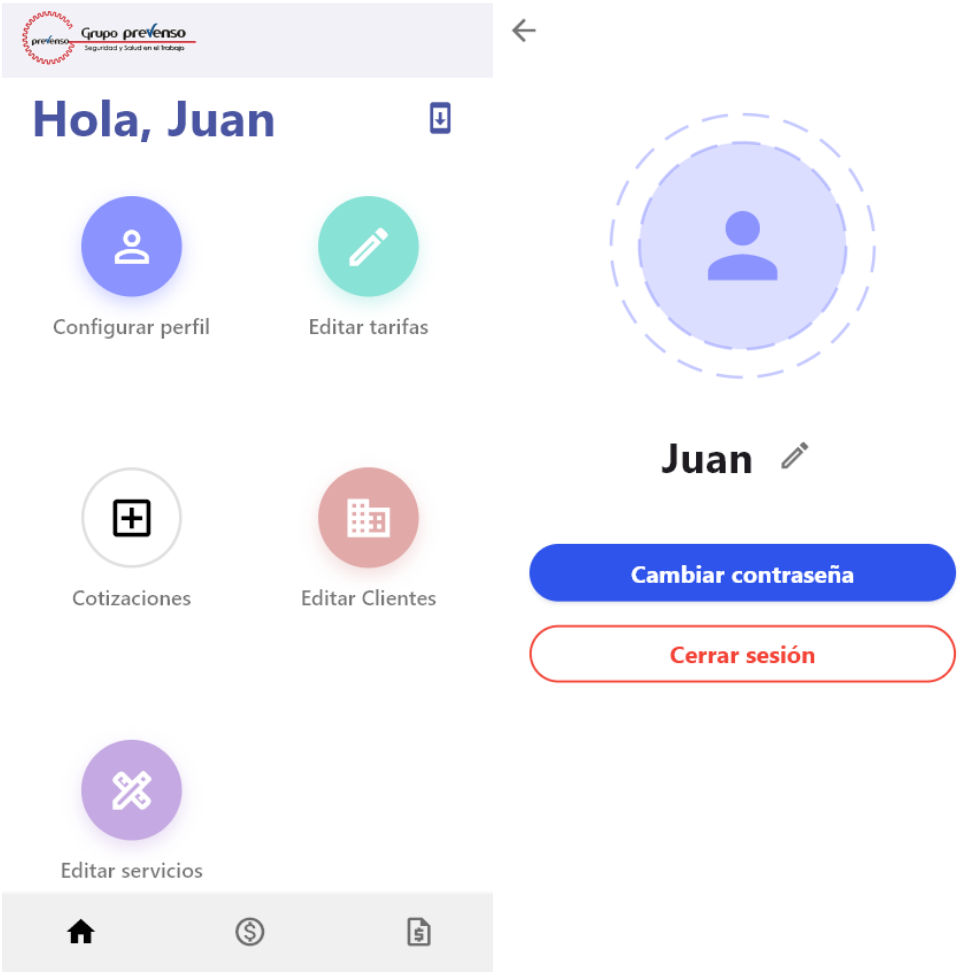
- Reunión de Aprobación: Se mostró el sistema a la tutora empresarial y a la gerente. Se verificó el flujo completo y se aprobó formalmente.
- Capacitación: Se entrenó al personal administrativo.
- Despliegue Total: La aplicación se instaló en todos los dispositivos de la empresa (Android y Windows), reemplazando el proceso manual anterior basado en Excel.

4.2.4.7 Resultado Final: Aplicación Multiplataforma en Producción

El resultado final de la pasantía es una aplicación funcional y desplegada. Esta sección presenta el producto final en ejecución y lo compara con los prototipos de Figma diseñados en la fase de arquitectura. Esta comparación es vital, ya que demuestra la evolución del diseño durante la implementación, un resultado esperado al usar una metodología ágil y el descubrimiento de mejores soluciones de UX/UI durante el desarrollo.

1. **Pantalla Principal (Dashboard) y Perfil**

Figura 22. Pantalla Principal (Dashboard) y Perfil de Usuario en Móvil.



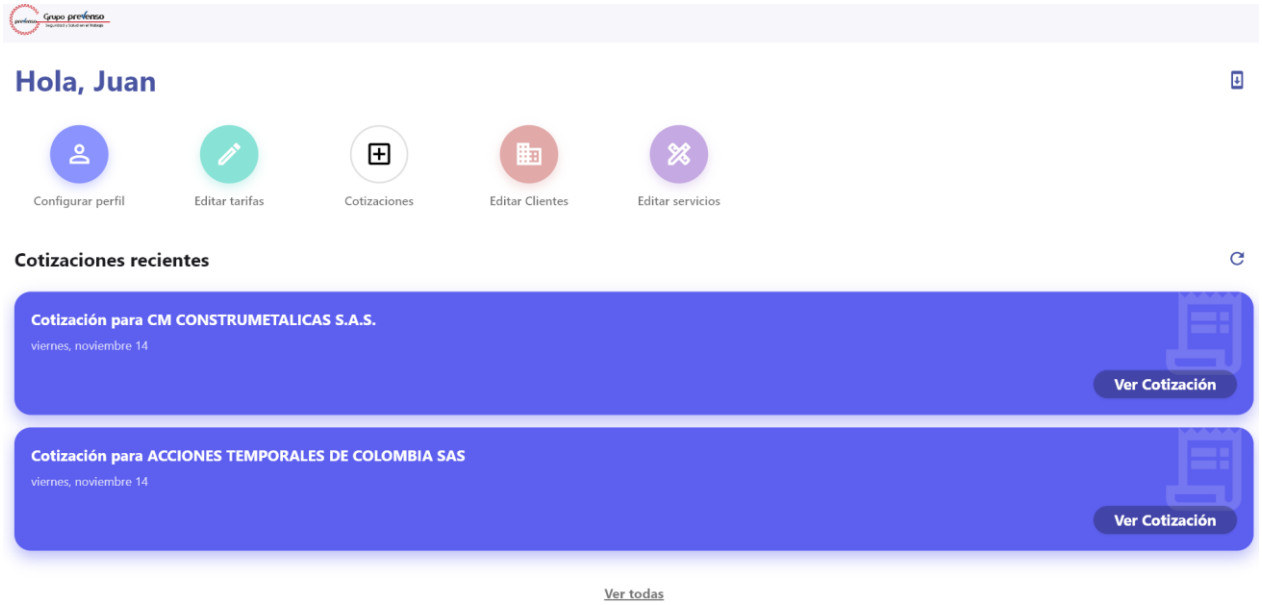
Fuente: Autor.

Análisis: El dashboard final (izquierda) es una evolución directa del prototipo. Mantiene la bienvenida ("Hola, Juan") y una cuadrícula de accesos directos. Sin embargo, se tomaron dos decisiones clave de mejora:

- **Iconografía Mejorada:** Se reemplazaron los íconos del prototipo con fondos circulares de colores, mejorando la legibilidad y el atractivo visual.
- **Barra de Navegación Inferior:** Se implementó una nueva barra de navegación inferior que no estaba en el prototipo inicial del dashboard. Esta se convirtió en el método de navegación secundario de la aplicación, una mejora ergonómica significativa para el uso móvil.

La pantalla de Perfil (derecha) se mantuvo casi idéntica al prototipo, lo que valida la solidez del diseño inicial para este flujo simple.

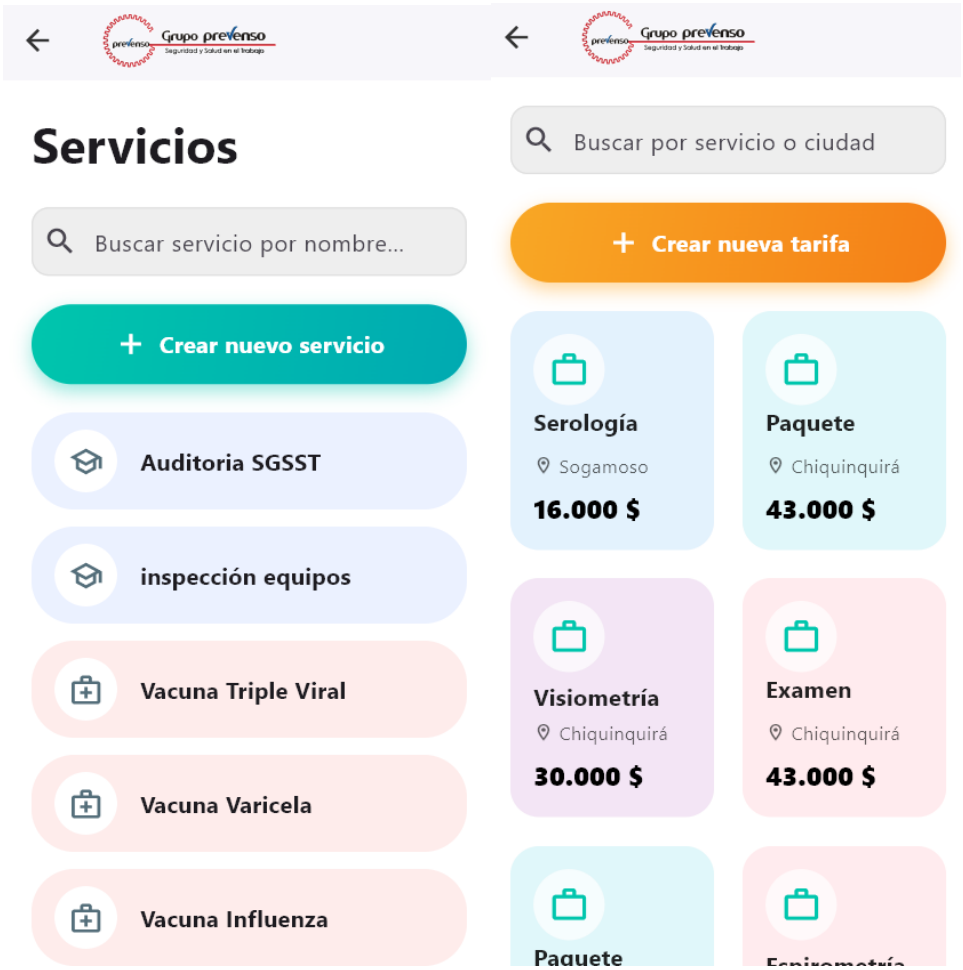
Figura 23. Pantalla Principal (Dashboard) en Versión de Escritorio (Windows).



Fuente: Autor.

2. Gestión de Servicios y Tarifas (Mejora de UX)

Figura 24. Pantallas de Servicios y Tarifas en Móvil.



Fuente: Autor.

Análisis: Estas pantallas representan la mejora de UX más significativa con respecto a los prototipos.

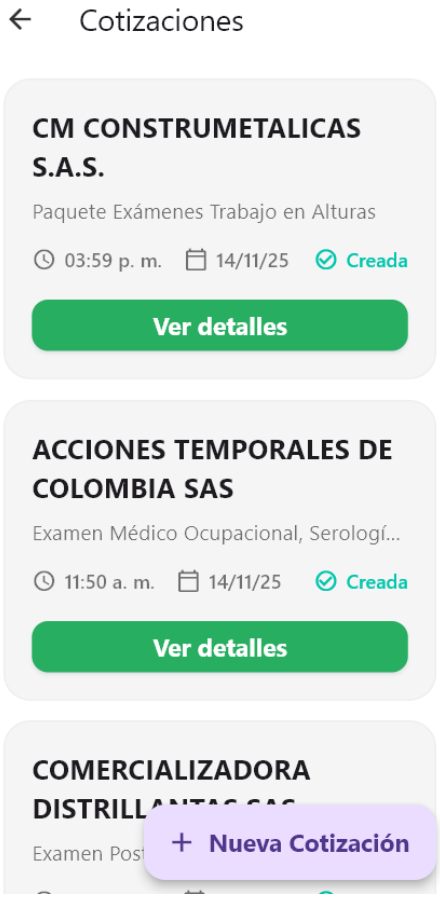
- **De Listas a Tarjetas:** Los prototipos originales mostraban simples listas de texto. Durante la implementación, esto se sintió poco intuitivo. Se rediseñó por completo para usar un sistema de "Tarjetas" (Cards) en una cuadrícula.
- **Información Visual:** La pantalla de Servicios (izquierda) ahora usa iconos (gorro de graduado para cursos, maletín para exámenes) y colores para categorizar visualmente los servicios, algo que no estaba en el prototipo.
- **Densidad de Información:** La pantalla de Tarifas (derecha) es un ejemplo de cómo condensar información. El prototipo era una lista simple; la versión final

muestra el servicio, la ciudad y el precio en una tarjeta visualmente atractiva, permitiendo al usuario escanear la información mucho más rápido. El botón de creación se movió a un botón interactivo, este diseño de Flutter mejora la usabilidad.

3. **Flujo de Cotizaciones (Móvil y Escritorio)**

Este es el corazón de la aplicación y donde el diseño multiplataforma de Flutter resalta, mostrando cómo el mismo código se adapta a diferentes factores de forma.

Figura 25. Listado de Cotizaciones en Dispositivo Móvil



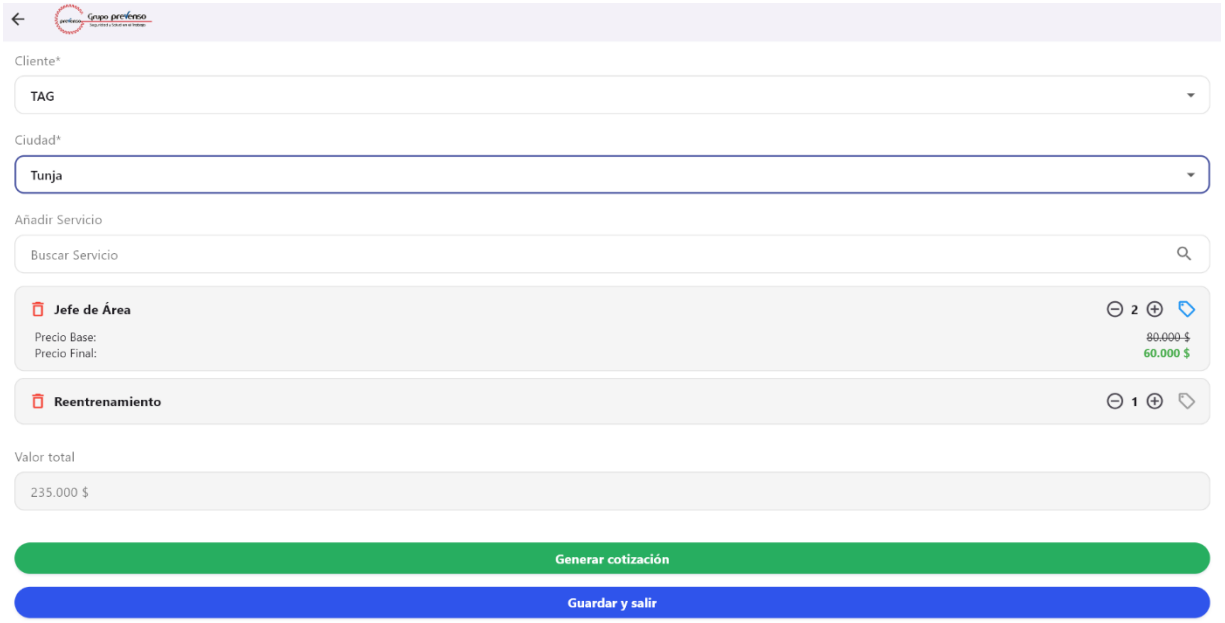
Fuente: Autor.

Análisis: La pantalla de Cotizaciones (Figura 25) es la implementación más fiel al prototipo original. Muestra una lista de tarjetas con el cliente, un resumen, la fecha, el estado y un botón de "Ver detalles". El botón flotante "Nueva Cotización" también se implementó tal cual. Esto indica que el diseño inicial para este flujo era robusto y no requirió modificaciones.

Figura 26. Creación de Cotización en Dispositivo Móvil

Fuente: Autor

Figura 27. Creación de Cotización en Versión de Escritorio (Windows).



Fuente: Autor

Análisis: Esta pantalla es la demostración más importante de la ventaja de la arquitectura elegida.

- **Diseño Responsivo:** El prototipo móvil (Figura 16) mostraba un flujo guiado de múltiples pasos para crear una cotización (seleccionar cliente, luego añadir ítems, luego ver total). Esto es excelente para una pantalla pequeña.
- **Versión de Escritorio:** En la versión de Windows (Figura 27), el mismo código fuente se reorganiza en una interfaz de una sola pantalla. No hay pasos. El usuario tiene el selector de cliente, el buscador de servicios, la lista de ítems (con la lógica de descuento visible) y el total, todo en una sola vista.
- **Validación de Lógica:** Esta captura también valida la implementación de la lógica de negocio compleja del Objetivo 3. Se puede ver el ítem "Jefe de Área" con un precio base de "80.000 $" y un precio final de "60.000 $", resultado de la lógica de descuento que se ejecuta en tiempo real.

4.2.4.8 Conclusión del Objetivo 4

Finalmente, el cuarto y último objetivo del proyecto, se materializó una interfaz multiplataforma que reproduce fielmente los diseños de Figma, con una sólida conexión a la API del backend, posibilitando el procesamiento de datos en tiempo real.

A lo largo de este proyecto, surgieron serias dificultades, tales como contratiempos con el diseño adaptativo en dispositivos físicos, fallos en las dependencias con Shorebird y la imperativa necesidad de idear un sistema autónomo para las actualizaciones. Tales inconvenientes demandaron la formulación de soluciones novedosas y operativas, optimizando la excelencia general del producto.

El análisis presentado en esta sección evidencia una evolución natural del diseño: aunque se respetaron los prototipos originales, se introdujeron mejoras visuales y ajustes según el dispositivo, logrando interfaces más claras, consistentes y fáciles de usar.

Finalmente, el proyecto cerró con la entrega oficial de la aplicación, su despliegue en todos los equipos de la empresa y la capacitación del personal, permitiendo la adopción completa del nuevo flujo digital dentro de Grupo Prevenso.

4.3. Herramientas utilizadas en el desarrollo del proyecto

Tabla 3. Herramientas utilizadas

Categoría	Herramienta / Tecnología	Propósito en el Proyecto
Gestión & Diseño	**Notion**	Seguimiento de tareas y cronograma.
	Figma	Diseño de prototipos de alta fidelidad (UI) y experiencia de usuario (UX).
	Git & GitHub	Control de versiones y resguardo del código fuente.
Backend (Servidor)	**Node.js**	Entorno de ejecución para la lógica del servidor.
	TypeScript	Lenguaje de programación tipado para mayor seguridad y mantenibilidad.
	Prisma ORM	Gestión de la comunicación y consultas con la base de datos.
	Puppeteer & EJS	Motor de renderizado para la generación dinámica de PDFs.
Base de Datos	**PostgreSQL**	Sistema de gestión de base de datos relacional.
Frontend	**Flutter**	Framework para el desarrollo multiplataforma

(Cliente)		(Móvil y Escritorio).
	Dart	Lenguaje de programación optimizado para interfaces de usuario.
	Visual Studio Code	Entorno de desarrollo integrado (IDE) principal.
	Android Studio	Herramientas de emulación y compilación para Android.
Pruebas	**Postman**	Pruebas de endpoints y validación de la API.

5. CONCLUSIONES

- La experiencia en Grupo Prevenso, terminó exitosamente completando el Objetivo General planteado que era el de desarrollar una aplicación multiplataforma para manejar las cotizaciones internas. Esto hizo más rápido, eficiente y transparente el proceso.
- El procedimiento manual previo, que dependía de Excel y Word, fue reemplazado por un sistema digital centralizado. El sistema moderno automatiza cálculos complejos, acorta el tiempo de elaboración de cotizaciones de minutos a segundos, y garantiza la exactitud de la información gracias a una base de datos relacional.
- Flutter resultó ser, sin lugar a duda, la herramienta perfecta para el desarrollo multiplataforma, proporcionando un desempeño sobresaliente y una experiencia visual consistente, tanto en Android como en Windows. Su capacidad para crear dos aplicaciones nativas a partir de una única fuente de código fue fundamental para cumplir los plazos.
- La pasantía no solo culminó con una aplicación que funcionaba. Sino también con una solución completa que el equipo de la empresa instaló, validó, y luego integró. Este proceso evidenció la habilidad del pasante para dominar el ciclo entero del desarrollo software: desde la recopilación de requisitos y el diseño de la arquitectura, hasta la ejecución, la depuración y la publicación final, que realmente trajo beneficios a la empresa.

6. BIBLIOGRAFÍA

- Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice.* Addison-Wesley Professional.
- Beck, K., Beedle, M., & van Bennekum, A. (2001). Obtenido de Agile Alliance: http://agilemanifesto.org/
- Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* University of California, Irvine.
- Google . (2024). Obtenido de Flutter.dev: https://flutter.dev/

- Pressman, R. (2010). *Ingeniería del Software. Un enfoque práctico.* USA: McGraw-Hill.

7. ANEXOS

7.1. Informe de Actualizaciones y Mejoras del Sitio Web

Fecha: 07 de julio de 2025

1. Resumen

El presente informe detalla la implementación exitosa de una serie de actualizaciones y mejoras en el sitio web de la empresa, de acuerdo con los requerimientos especificados. Los cambios abarcan la actualización de contenido informativo, la adición de nuevos servicios y la optimización de elementos visuales. Todas las tareas solicitadas han sido completadas y verificadas.

2. Detalle de Modificaciones Realizadas por Sección

A continuación, se presenta un desglose de los cambios aplicados en cada una de las páginas correspondientes:

Página de Inicio:

- **Texto de Experiencia:** Se actualizó el texto para reflejar **"más de 25 años"** de experiencia, fortaleciendo el mensaje de trayectoria de la empresa.
- **Corrección Visual:** Se solucionó un error que impedía la correcta visualización de los íconos en esta sección, restaurando la integridad del diseño.

Página "Nosotros":

- **Texto de Experiencia:** De manera consistente con la página de inicio, se actualizó el texto a **"más de 25 años"** para mantener la coherencia de la información en todo el sitio.

Página "Medicina Preventiva":

- **Alineación Visual:** Se corrigió la alineación de los íconos de check (✅) para mejorar la legibilidad y el aspecto profesional de los listados.
- **Inclusión de Nuevos Servicios:** Se amplió el portafolio de servicios visibles en esta sección, añadiendo:
    - Programas de vacunación.

- Toma de muestras de laboratorio clínico.

Página "Alturas":

- **Actualización de Contenido (Nivel Coordinador):** Se eliminaron los siguientes ítems que ya no son requisitos para el curso:
  - Ítem: "experiencia 50 horas".
  - Ítem: "certificado curso 50 h".
- **Eliminación de Servicio:** Se eliminó por completo la sección informativa del **"Nivel de actualización de coordinador"**, ya que este servicio ya no se ofrece.

Página "Contáctanos":

- **Adición de Información de Contacto:** Se añadieron los nuevos números de teléfono bajo el apartado "Servicio de IPS" para facilitar la comunicación directa con esa área:
  - 310-337-7634
  - 320-855-8707

3. Mejoras Generales Implementadas

Visibilidad de Redes Sociales:

- Para cumplir con el requerimiento de hacer las redes sociales más notorias, se implementó una solución integral:
  - **Iconos Sociales:** Se instaló y configuró un sistema de íconos sociales (incluyendo TikTok y Facebook) en la página de "Contacto", permitiendo a los usuarios acceder directamente a los perfiles de la empresa.

4. Detección de Conflicto de Dependencia

Tras la actualización, se identificó una dependencia crítica no informada previamente: un sistema interno de certificados requiere obligatoriamente que el servidor opere con la versión de PHP 7.3. Esta versión antigua es incompatible con los componentes actualizados del sitio web, lo que generó un conflicto técnico.

Para resolver la incompatibilidad y priorizar el funcionamiento del sistema de certificados, se tomó la decisión de revertir el sitio web a su estado anterior. Se procedió a restaurar la copia de seguridad más reciente disponible en el cPanel, correspondiente al 2 de julio.

5. Falla en la Infraestructura del Hosting

Durante el proceso de restauración, la herramienta automática proporcionada por el proveedor de hosting falló, dejando el sitio web y los servicios de correo electrónico asociados completamente inoperativos (error 404). El incidente fue escalado de inmediato

al equipo de soporte técnico del proveedor de hosting, informándoles de la falla en su sistema. El proveedor asumió la responsabilidad y realizó una restauración manual completa

8. MANUAL DE USUARIO

SISTEMA DE GESTIÓN DE COTIZACIONES MULTIPLATAFORMA

Fecha de Emisión: noviembre 2025

ELABORADO POR
Juan David Buitrago Lozano
1057570343
2310305

DIRIGIDO A
Personal Administrativo y Comercial de Grupo Prevenso LTDA.

Tunja, Boyacá 2025

TABLA DE CONTENIDO

1. INTRODUCCIÓN

Bienvenido al Sistema de Gestión de Cotizaciones de Grupo Prevenso. Esta aplicación multiplataforma ha sido diseñada para facilitar y agilizar la creación de cotizaciones comerciales, la gestión de clientes y la administración de servicios y tarifas.

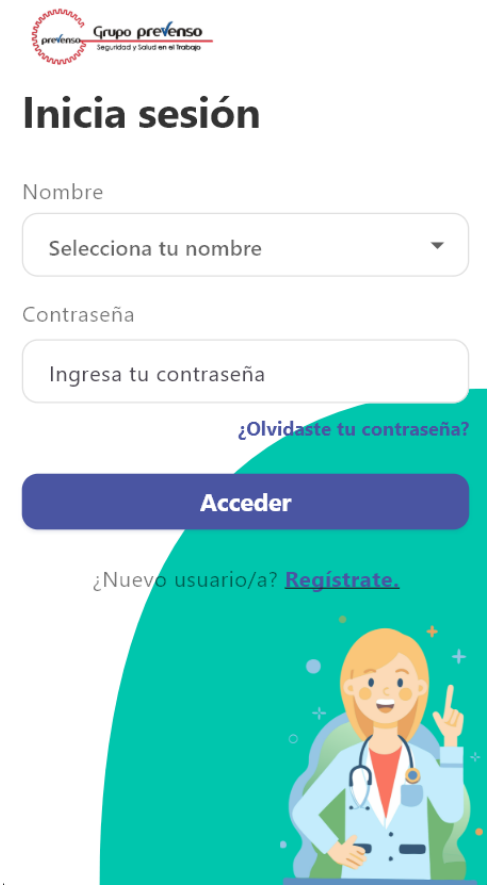
Este manual le guiará a través de las funcionalidades principales del sistema, tanto en su versión para dispositivos móviles (Android) como para computadores de escritorio (Windows).

2. ACCESO AL SISTEMA

2.1 Iniciar Sesión

Al abrir la aplicación, encontrará la pantalla de bienvenida.

1. Presione el botón "Iniciar Sesión".
2. Ingrese su correo electrónico corporativo y su contraseña asignada.
3. Si es la primera vez que ingresa, asegúrese de tener conexión a internet para validar sus credenciales.



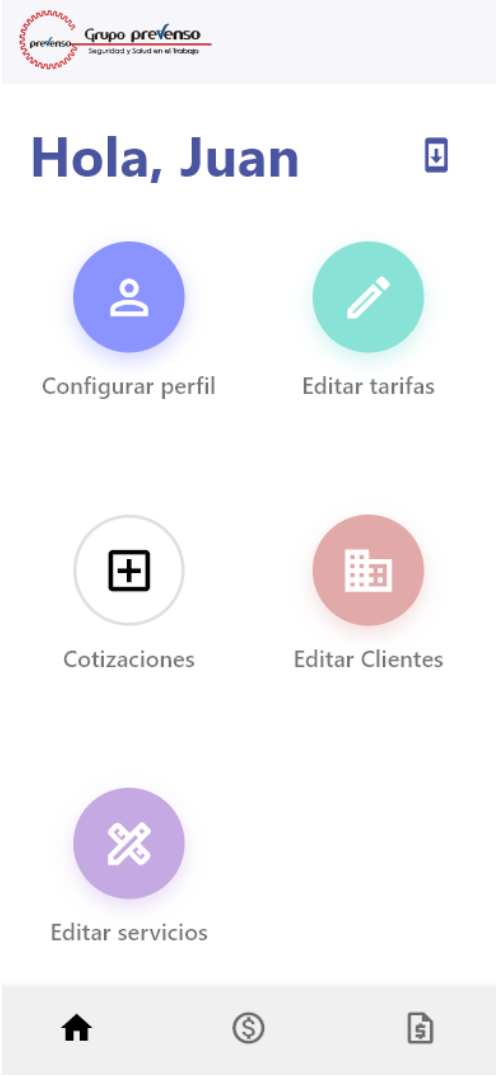
2.2 Recuperación de Contraseña

Si ha olvidado su contraseña:

1. En la pantalla de inicio, seleccione la opción "¿Olvidaste tu contraseña?".
2. Ingrese su correo electrónico. Recibirá un código de verificación.
3. Ingrese el código en la aplicación y establezca su nueva contraseña.

3. PANTALLA PRINCIPAL (DASHBOARD)

Una vez dentro, verá el Panel Principal. Este es su centro de control.



- **Hola, [Su Nombre]:** En la parte superior verá su perfil. Puede tocar su foto o nombre para cerrar sesión o cambiar su clave.
- **Accesos Rápidos:** Encontrará botones grandes para las funciones más usadas:
    - **Cotizaciones:** Para ver el historial o crear una nueva.
    - **Clientes:** Para administrar su base de datos de contactos.
    - **Servicios y Tarifas:** Para configurar el catálogo (solo administradores).

(Nota: La aplicación cuenta con una barra de navegación inferior para moverse rápidamente entre Inicio, Cotizaciones y Perfil).

4. GESTIÓN DE COTIZACIONES (MÓDULO PRINCIPAL)

Esta es la función más importante de la aplicación.

4.1 Crear una Nueva Cotización

1. Diríjase al módulo de Cotizaciones y presione el botón "Nueva Cotización".
2. **Seleccionar Cliente:** Busque el cliente por nombre o NIT. Si el cliente no existe, puede crearlo ahí mismo presionando "Nuevo Cliente".
3. **Seleccionar Ciudad:** Elija la ciudad para la cual aplica la cotización (esto ajustará las tarifas automáticamente).

4. **Agregar Servicios:**
   - Busque el servicio deseado (ej. "Curso de Alturas").
   - Indique la Cantidad de servicios.
   - El sistema mostrará el Precio Base automáticamente.

5. **Aplicar Descuentos (Opcional):**
    - Toque el ícono de etiqueta (🏷️) junto al servicio.
    - Seleccione el tipo de descuento: Porcentaje (%) o Monto Fijo ($).
    - Ingrese el valor y el sistema recalculará el total inmediatamente.

6. **Finalizar:** Revise el "Valor Total" en la parte inferior y presione "Generar Cotización".

4.2 Generar PDF

Una vez guardada la cotización, verá el resumen. Presione el botón "Descargar PDF". El documento se generará con el membrete oficial de la empresa y quedará listo para compartir por WhatsApp o Correo.

Grupo prevenso
Seguridad y Salud en el Trabajo
Nit. 820005100-6

NTC 6072:2014 BUREAU VERITAS Certification

DATOS DEL CLIENTE
Cliente: [redacted]
NIT/C.C: 9[redacted]
Dirección: [redacted]
Teléfono: 3[redacted]
Email: [redacted]

Cotización No.
CTGP0040
Válida por 30 días

DETALLE DE SERVICIOS

CÓDIGO	DESCRIPCIÓN DEL SERVICIO	CANTIDAD	VALOR UNITARIO	VALOR TOTAL	DESCUENTO	VALOR FINAL
0035	Examen Médico con Énfasis Osteomuscular y Cardiovascular	1	$ 30.000	$ 30.000	-$ 0	**$ 30.000**

Atendido por: Laura Rodriguez

Subtotal (sin descuentos):	$ 30.000
Ahorro total por descuentos:	-$ 0
TOTAL A PAGAR:	**$ 30.000**

MÉTODOS DE PAGO

Directamente en nuestra Sede Principal: Carrera 13 No. 22 - 81 Barrio Popular (costado occidental de la Secretaria de Salud de Boyacá, Parque Santander) en la ciudad de Tunja. - Consignación o transferencia a la cuenta de AHORROS del banco Davivienda No.176200017810 a nombre de GRUPO PREVENSO LTDA. NIT 820005100 - 6. - Consignación o transferencia a la cuenta de AHORROS de BANCOLOMBIA No. 258-290407-20 a nombre de GRUPO PREVENSO LTDA. NIT 820005100 - 6. - Transferencia a la cuenta de NEQUI (Bancolombia) 312 336 7478.

Escanéame

¡Contáctanos!

312 336 7478 • 3208558707 contacto@grupoprevenso.com
Cra. 13 No. 22-81, Barrio Popular, Tunja
www.grupoprevenso.com grupoprevenso grupoprevensoltda

MÁS DE 23 AÑOS DE EXPERIENCIA

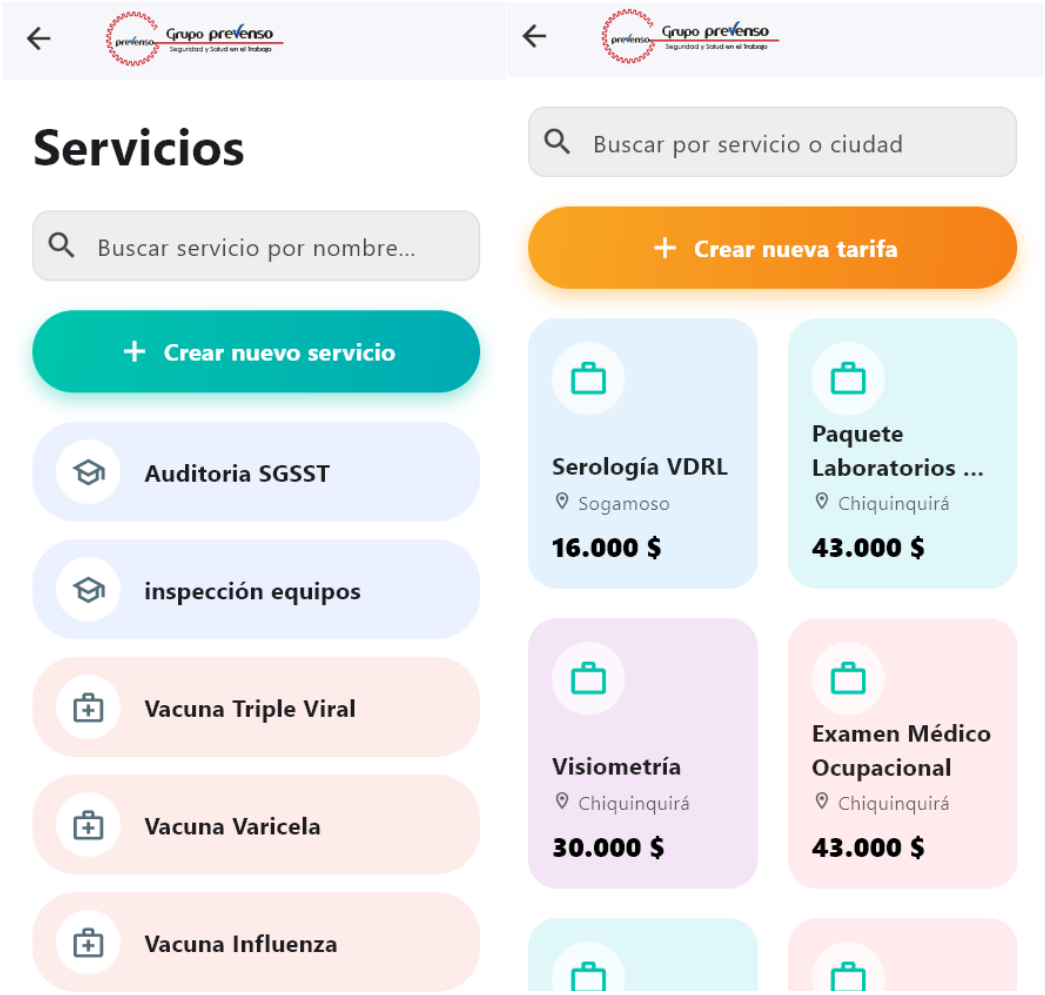
5. GESTIÓN DE CLIENTES Y SERVICIOS

5.1 Clientes

- Vaya a "Editar Clientes".
- Verá la lista de empresas registradas. Use el buscador para encontrar una rápidamente.
- Para crear uno nuevo, presione "Crear Cliente" y diligencie: Nombre, NIT, Teléfono, Dirección y Correo.
- *Importante:* Mantenga estos datos actualizados, ya que son los que aparecen en el PDF final.

5.2 Servicios y Tarifas (Solo Administradores)

- **Servicios:** Aquí define el catálogo (ej. "Examen Médico", "Serología"). Puede crear nuevos servicios especificando su nombre y tipo.
- **Tarifas:** Este módulo es vital. Aquí le dice al sistema cuánto cuesta cada servicio en cada ciudad.
    - Ejemplo: Puede configurar que la "Serología" cueste $16.000 en Sogamoso y $18.000 en Tunja. El sistema elegirá el precio correcto automáticamente al cotizar.



6. PREGUNTAS FRECUENTES Y SOPORTE

¿Cómo actualizo la aplicación?

La aplicación cuenta con un **Sistema de Actualización Automática**. Cada vez que abra la app, esta verificará si hay una nueva versión.

- Si hay una actualización, verá un mensaje en pantalla.
- Presione **"Descargar Ahora"**.
    - **En Android:** Se descargará y le pedirá permiso para instalar.
    - **En Windows:** Se abrirá la carpeta de Descargas con el nuevo archivo.

La aplicación se ve diferente en mi computador y mi celular.

Es normal. El sistema es **responsivo**.

- En el **Celular**, verá la información en tarjetas y listas para facilitar el uso táctil.

- En el **Computador**, verá tablas detalladas que aprovechan el tamaño de la pantalla para mostrar más datos a la vez.

¿Necesito internet?

Sí. Para garantizar que todos los empleados vean los mismos precios y clientes actualizados, la aplicación requiere conexión a internet para funcionar.