

UNIVERSIDAD DEL VALLE DE GUATEMALA

Ing. Cristián Rafael Muralles Salguero

Ingeniería de Software



Excelencia que trasciende

DEL VALLE
GRUPO EDUCATIVO

Corte 3. Análisis y Diseño

Angel Esquit

Javier España

José Merida

Karen Pineda

Guatemala, 17 de Marzo de 2025

Resumen

Este proyecto tiene como objetivo el desarrollo de un sistema de gestión médica (CRM médico) enfocado en la optimización de procesos administrativos en la clínica oftalmológica Optyma, ubicada en Antigua Guatemala. La plataforma permitirá gestionar citas, manejar historiales clínicos, automatizar recordatorios y mejorar la comunicación con los pacientes.

La digitalización de los consultorios médicos se ha vuelto una necesidad crítica debido a la ineficiencia y errores asociados con el uso de agendas físicas y registros en papel. Mediante este sistema, se busca reducir la carga administrativa, mejorar la eficiencia operativa y garantizar una mejor experiencia tanto para el personal médico como para los pacientes.

Introducción

Descripción de la entidad:

La clínica oftalmológica Optyma, ubicada en Antigua Guatemala, brinda servicios médicos especializados en oftalmología y óptica. Actualmente, la clínica gestiona sus procesos administrativos de manera manual, utilizando agendas físicas y registros en

papel, lo que genera problemas como: Errores en la programación de citas, duplicación de esfuerzos administrativos, pérdida o dificultad de acceso a historiales clínicos, y falta de automatización en la comunicación con los pacientes.

Descripción de la idea:

El proyecto consiste en el desarrollo de un sistema de gestión médica (CRM médico) para digitalizar los procesos administrativos de la clínica Optyma. La plataforma permitirá la gestión de citas, la administración de historiales clínicos, la automatización de recordatorios y la mejora en la comunicación con los pacientes, con un enfoque en usabilidad, seguridad y eficiencia.

A través de un diseño intuitivo y adaptable, el sistema facilitará el trabajo tanto para médicos con experiencia en software de gestión como para aquellos que aún utilizan métodos tradicionales.

Objetivo general:

Desarrollar una plataforma digital accesible e intuitiva para optimizar la gestión de datos médicos en consultorios oftalmológicos, mejorando la eficiencia operativa y reduciendo la carga administrativa.

Objetivos específicos

1. Implementar un módulo de historiales clínicos con acceso rápido y seguro a la información de los pacientes.
2. Automatizar tareas administrativas, eliminando la dependencia de documentos físicos y minimizando errores en la gestión.
3. Diseñar una interfaz intuitiva y fácil de usar, asegurando una rápida adopción del sistema por parte del personal médico y administrativo.
4. Mejorar la seguridad y privacidad de los datos clínicos, cumpliendo con estándares de protección de información médica.
5. Optimizar la comunicación con los pacientes mediante recordatorios automatizados y acceso remoto a la información relevante.

Prototipos

Prototipo 1 – Registro de exámenes

Iteración 1

REGISTRO DE EXAMENES

CLÍNICA
OPTALMOLÓGICA
OPTYMA

REGISTRAR EXAMENES PACIENTE

ID

489652

Paciente

JUAN CARLOS MOLINA

Q Buscar

ID

589632

Médico

GABRIELA MARÍA HERNANDEZ

Q Buscar

Examen

ABERROMETRÍA

+ Agregar

REGISTRAR REALIZACIÓN EXAMENES

Notas: Según las necesidades y observaciones del usuario

- Se añadió un nuevo campo para que se pueda indicar la fecha del examen
- También se implementó una sección donde se puedan observar los exámenes registrados. Se pueden eliminar los exámenes o también si ya está listo, mandar el pdf al correo electrónico

Iteración 2

REGISTRO DE EXAMENES

CLÍNICA
OFTALMOLÓGICA
OPTYMA

REGISTRAR EXAMENES PACIENTE

Fecha

02/05/2025

ID

489652

Paciente

JUAN CARLOS MOLINA

Q Buscar

ID

589632

Médico

GABRIELA MARÍA HERNANDEZ





Q Buscar

Examen

ABERROMETRÍA

+ Agregar

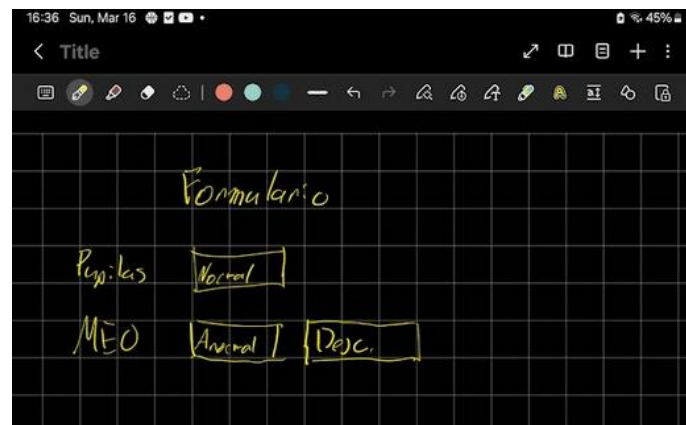
REGISTRAR REALIZACIÓN EXAMENES

ID	Examen	Acción
2	CURVA DE PRESIÓN OCULAR	 PDF 
1	ABERROMETRÍA	 PDF 

Prototipo 2 – Rellenado de Formularios

Este prototipo se realizó de manera muy cruda, iterando rápidamente con el Product Owner. Básicamente se buscaba establecer la manera que se rellenará el formulario, al iniciar la conversación se estableció que algunas evaluaciones simplemente se llenan con campos de “Normal” y “Anormal”. Luego, las observaciones “Anormales” deben tener un campo de texto para describir lo observado. Mientras que el valor predeterminado es de “Normal”, ya que la mayoría de las personas llegan por un problema en específico y no presentan anomalías en múltiples campos.

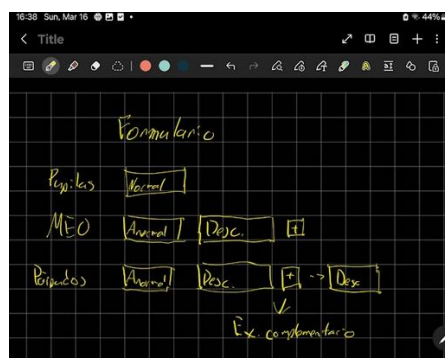
Primera Iteración



El feedback en cuanto a este prototipo fue el siguiente:

- Debe existir una manera de asociar exámenes a alguna sección del formulario. Por ejemplo, un MEO anormal puede llevar un examen complementario.

Segunda Iteración



Se buscó hacer más accesible recetar exámenes asociados con algún valor anormal asociado. Luego de este llenado de formulario, se conversó con el P.O sobre el flujo al recetar un examen. El cuál fue el siguiente:

- El examen se receta durante una consulta, luego de identificarse alguna anomalía

- Luego, durante otra consulta se vuelve a realizar la evaluación y se cubren los resultados de los exámenes.
- Es bastante útil poder ver valores anteriores fácilmente, puede ser con un split-screen-view o presentarlos al llenar el formulario.

Gracias a estas iteraciones rápidas y conversaciones logramos agregar algunos requisitos funcionales al programa.

- El sistema debe poder mostrar valores de consultas anteriores al llenar un formulario.
- El sistema debe de poder crear exámenes que se asocian con una cita cuándo se cubran.

Requisitos Funcionales

Lista de requisitos funcionales asociados con cada historia de usuario

Requisito Funcional	Actividad	Historia de Usuario
Ver lista de citas programadas	Manejar Citas	El sistema debe poder mostrar citas por día y por semana.
Agregar Citas	Manejar Citas	El sistema debe permitir a los médicos registrar nuevas citas en el calendario.
Actualizar Citas	Manejar Citas	El sistema debe permitir modificar la fecha y hora de una cita existente.
Cancelar Citas	Manejar Citas	El sistema debe permitir la cancelación de citas con notificación al paciente.
Buscar Paciente	Acceder a información del paciente	El sistema debe permitir la búsqueda de pacientes por nombre o ID.
Ver Historial	Acceder a información del paciente	El sistema debe mostrar el historial clínico completo de un paciente.
Agregar nueva ficha	Actualizar el Historial Clínico	El sistema debe permitir la creación de un nuevo expediente clínico para cada paciente.

Agregar examen a consulta	Actualizar el Historial Clínico	El sistema debe permitir asociar los resultados de un examen a una consulta específica, luego de que haya sido realizado
Agregar diagnóstico	Actualizar el Historial Clínico	El sistema debe permitir a los médicos registrar diagnósticos en los historiales clínicos.
Agregar tratamientos	Actualizar el Historial Clínico	El sistema debe permitir registrar tratamientos y medicamentos recetados.
Agregar exámenes	Actualizar el Historial Clínico	El sistema debe permitir a los médicos recetar exámenes a los pacientes durante una consulta
Almacenar exámenes	Actualizar el Historial Clínico	El sistema debe permitir a los usuarios agregar los resultados (PDF) de un examen y asociarlos a un usuario.
Actualizar datos personales del paciente	Actualizar el Historial Clínico	El sistema debe permitir actualizar la información personal del paciente.
Recordatorio de Citas	Comunicación con Pacientes	El sistema debe enviar notificaciones automáticas a los pacientes sobre sus próximas citas.
Actualizaciones de Tratamiento/Diagnóstico	Comunicación con Pacientes	El sistema debe notificar a los pacientes sobre cambios en su tratamiento o diagnóstico.

Diagramas

Diagrama de Clases

Entregado en archivo .UML en el repositorio de Git

Descripción de las Clases

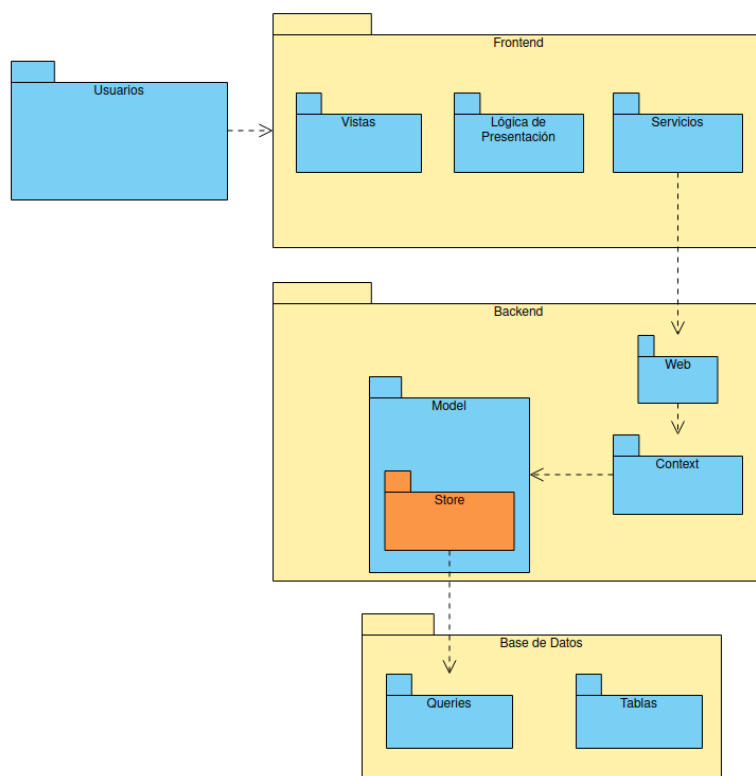
Paciente: Representa a un paciente del consultorio, tiene como atributos un ID para identificarlo, el nombre del paciente y teléfono.

Historial clínico: Almacena los registros médicos de un paciente. Tiene ID, paciente, diagnóstico, tratamiento y fecha.

Cita: Representa una cita médica. Cuenta con un ID, el paciente, el médico asignado, fecha, hora y estado.

Usuario: Representa un usuario del sistema, ya sea paciente, médico o administrador, que tiene atributos ID, nombre del usuario, contraseña y rol.

Diagrama de Paquetes



Descripción de cada Paquete y Componentes

Vistas

Se encarga de la presentación de las diferentes páginas dentro de la aplicación web, conteniendo contenido el contenido con el que interactúa directamente el usuario.

Lógica de Presentación

Se encarga de transformar información, manejar estados del UI, rutas de navegación, etc. En resumen, la lógica dentro del front-end de la aplicación.

Servicios

Se encarga de la comunicación con el back-end y la transformación de información.

Web

Maneja las solicitudes y respuestas HTTP, define rutas y handlers. Un paralelo en MVC sería un View, ya que se encarga de procesar solicitudes y respuestas sin enfocarse en la lógica de negocio interna.

Context

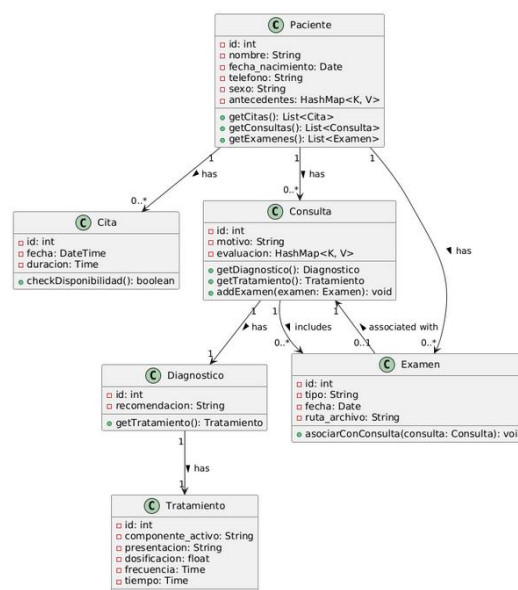
Es un módulo intermediario entre Web y Model, maneja la lógica de negocio y es una “interfaz” con la que interactúa Web. Un paralelo en MVC sería un controller, ya que es una capa intermedia que orquesta a las demás.

Store

Store se encarga de la comunicación con la base de datos de manera aislada.

Persistencia

Diagrama de Clases Persistentes

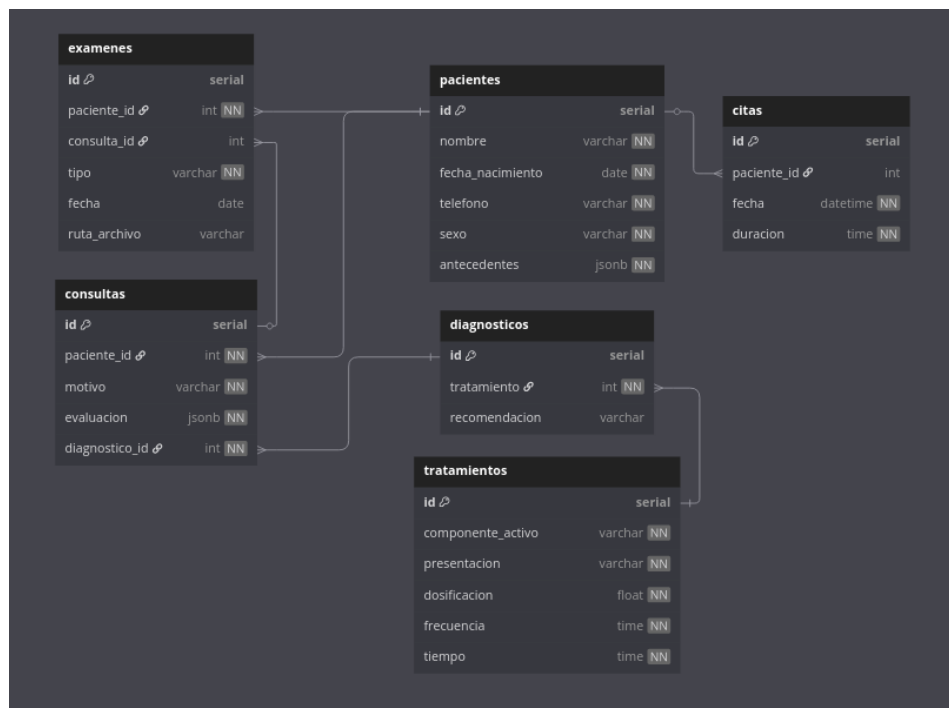


Tecnologías de Almacenamiento

Como equipo de desarrollo decidimos utilizar una base de datos relacional, ya que se adapta a la manera que se relaciona la información que deseamos almacenar. Observando el diagrama de clases persistentes, nos damos cuenta que está altamente relacionada y este tipo de DBMS es útil para este caso de uso específico. Luego, propusimos 4 DBMS por utilizar: SQLite, Microsoft SQL, Oracle Database y PostgreSQL.

Decidimos utilizar PostgreSQL para este proyecto debido a sus características avanzadas, garantía de seguridad, velocidad y flexibilidad. A diferencia de SQLite, PostgreSQL maneja un control de concurrencia más robusto sin comprometer el rendimiento. Además, pueden utilizarse tipos de datos avanzados como JSONB y modelado a través de objetos. Estos features pueden resultar cruciales al no tener una estructura muy claramente definida para, por ejemplo, los formularios médicos. Adicionalmente, observamos que Microsoft SQL Server y Oracle Database tienen costos asociados con licencias propietarias y pueden no ser compatibles con ciertas tecnologías. Por lo que descartamos estos DBMS.

Diagrama Entidad-Relación de la Base de Datos



Tecnología a Utilizar

Estimaciones

- 10-20 pacientes al día, resultando en alrededor de 4800 consultas anuales
- Máximo 3-4 usuarios interactuando simultáneamente
- La información debe ser almacenada de manera completamente segura, de lo contrario se podrían encontrar repercusiones legales.
- El sistema debe tener un uptime consistente, ya que se están manejando datos sensibles.

Backend

Lenguajes Considerados

Cómo equipo, decidimos seleccionar un lenguaje de programación antes de enfocarnos en Frameworks. Decidimos evaluar diferentes aspectos tales como:

- Legibilidad
- Escritura
- Fiabilidad
- Mantenibilidad
- Seguridad
- Comunidad / Ecosistema
- Curva de Aprendizaje

Por lo cuál nos decidimos enfocar en los siguientes lenguajes comúnmente utilizados en programación backend.

Go

Go es conocido por su simplicidad y velocidad, la concurrencia se maneja utilizando Goroutines y utiliza un GC para el manejo de memoria. Esto le da una ventaja en cuánto a tiempos de desarrollo, curva de aprendizaje y rendimiento. Además, los binarios generados por Go son considerablemente pequeños. Sin embargo, el GC genera rendimiento inconsistente y puede introducir pausas en las aplicaciones. Adicionalmente, el ecosistema es pequeño comparado con otros lenguajes y es posible que haya una falta de bibliotecas especializadas.

Rust

Rust destaca por su seguridad en la memoria sin utilizar un GC, por lo cuál es sumamente rápido y confiable. El sistema de ownership garantiza que no existan fugas de memoria o accesos inválidos. Además, está mencionado cómo uno de los lenguajes más seguros en una lista del NIST, lo cual se alinea con los requisitos no funcionales. Por otra parte, la curva de aprendizaje es bastante pronunciada y puede llegar a existir una deuda técnica dentro del equipo.

JavaScript / TypeScript

TypeScript es altamente flexible y tiene un ecosistema amplio, con una gran cantidad de frameworks y bibliotecas disponibles. Sin embargo, nos veríamos obligados a correr el Node, dónde el runtime corre single-threaded.

Java

Java es un lenguaje maduro con una gran cantidad de bibliotecas y herramientas. Sin embargo, el uso de un GC puede introducir pausas y Java puede llegar a ser muy verboso. Además, la configuración inicial en comparación a lenguajes más modernos como Go o Rust puede ser tediosa.

Cuadro Comparativo

Criterio	Go	Rust	JS / TS	Java
Legibilidad	Alta, la sintaxis es minimalista y clara.	Moderada, se complica un poco con el sistema de ownership.	Alta en JavaScript, mejorada con TypeScript.	Moderada, es algo verboso y puede dificultar la lectura
Escritura	Alta, la sintaxis es concisa y abstrae conceptos cómo concurrencia.	Moderada, requiere más código para garantizar seguridad.	Alta en JavaScript, mejorada con TypeScript.	Moderada, la verbosidad lo complica y la configuración es extensa.
Fiabilidad	Alta, el GC maneja la memoria de manera eficiente	Muy alta, provee garantías de seguridad en tiempo de compilación	Pobre en JavaScript, moderada en TypeScript por el tipado estático.	Alta, el GC maneja la memoria de manera eficiente.
Mantenibilidad	Alta	Alta	Moderada	Alta
Seguridad	Alta	Excelente	Moderada	Alta
Compatibilidad	Buena	En crecimiento	Excelente	Excelente
Comunidad / Ecosistema	Creciente	Creciente	Enorme	Madura y Estable
Curva de Aprendizaje	Baja	Alta	Baja	Moderada

Elección

Elegimos utilizar **Rust** debido a la necesidad de implementar un sistema robusto que cumpla con ciertos estándares de seguridad. Cualquier fallo dentro del sistema puede resultar en consecuencias fuertes (cómo acceso a información privada) y el rendimiento consistente es crucial.

Frameworks

En el ecosistema de Rust, existen varios frameworks para desarrollar backends. Empezando por Rocket, este es conocido por su facilidad de uso y sintaxis intuitiva. Es útil para prototipado y aplicaciones web más sencillas. Luego, tenemos Actix Web el cuál es un framework ideal para aplicaciones de alto tráfico y sistemas distribuidos. Sin embargo, Actix suele ser muy complejo y puede presentar desafíos para nuestro equipo de desarrollo.

Elección

Elegimos utilizar **Axum**, ya que nos da un balance entre rendimiento, facilidad de uso y flexibilidad.

Frontend

Frameworks / Librerías Propuestos

Ya que el proyecto consiste en una aplicación web, postulamos diferentes frameworks y librerías de JavaScript para manejar el Frontend.

React

React es una librería de JavaScript altamente flexible, ya que puede integrarse fácilmente con otras bibliotecas y herramientas. Cuenta con una comunidad muy grande (siendo la librería de JS más utilizada según múltiples encuestas) y el ecosistema es sumamente amplio. Sin embargo, la curva de aprendizaje se inclina un poco al introducir conceptos como JSX, hooks y manejo de estado. Adicionalmente, a pesar de ser potente no es un framework completo.

Svelte

Svelte es un framework moderno que compila el código a JavaScript puro en tiempo de compilación. Su mayor ventaja es el rendimiento, ya que no utiliza un DOM virtual y compila directamente a código eficiente. Además, su sintaxis es simple y requiere menos código boilerplate, lo que lo hace fácil de aprender y usar. Sin embargo, su ecosistema es más pequeño en comparación a React o Vue.

Angular

Angular es un framework completo para construir aplicaciones web escalables y empresariales. Su mayor ventaja es que incluye todo lo necesario para desarrollar aplicaciones complejas, como manejo de estado, routing, formularios y herramientas de testing. Además, implementa TypeScript de forma nativa. Sin embargo, esta amplia funcionalidad resulta en una curva de aprendizaje más pronunciada y puede ser verboso en comparación a otros frameworks.

Vue

Vue es un framework simple y flexible, se puede adaptar a proyectos de todos los tamaños. Su mayor ventaja es la facilidad de aprendizaje, tiene una sintaxis intuitiva y una documentación clara. Además, cuenta con un sistema de reactividad integrado que simplifica el manejo de estado. Mientras que su ecosistema sigue siendo más pequeño en comparación a React, tiene una comunidad activa y en crecimiento. Es ideal para proyectos que buscan equilibrio entre simplicidad y funcionalidad.

Elección

Decidimos elegir **Vue**, ya que es un framework bastante flexible. Al utilizar una tecnología más compleja en el backend, decidimos utilizar algo con una menor curva de aprendizaje. Además, la aplicación a desarrollar según los requisitos funcionales no es extremadamente compleja en términos de frontend.

Informe de Gestión del Tiempo

Desglose de la Investigación en Tareas:

Tarea	Miembro	Fecha	Tiempo	Tiempo Real

LOGT

José Mérida

Fecha	Inicio	Fin	Tiempo Interrupción	Delta Tiempo	Tarea	Comentarios

Karen Pineda

Fecha	Inicio	Fin	Tiempo Interrupción	Delta Tiempo	Tarea	Comentarios
14/03/2025					Prototipo 1	-----

Ángel Esquit

Fecha	Inicio	Fin	Tiempo Interrupción	Delta Tiempo	Tarea	Comentarios

Javier España

Fecha	Inicio	Fin	Tiempo Interrupción	Delta Tiempo	Tarea	Comentarios

Aspectos Positivos

Aspectos a Mejorar

