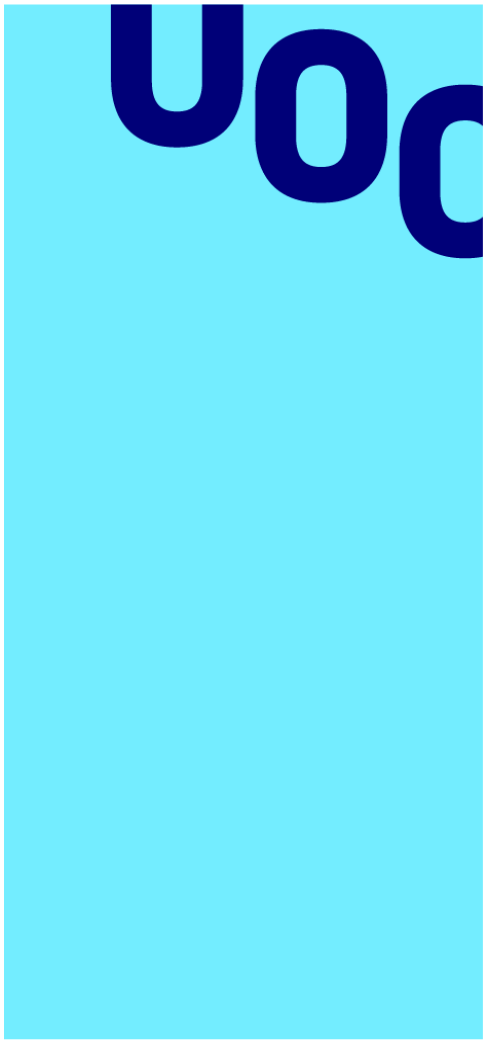
Desarrollo de una Aplicación Web para Gestión de Consultas Médicas



Universitat Oberta de Catalunya

Kevin Nava Argos

Trabajo Final de Grado Desarrollo Web

Profesor Tutor:
Gregorio Robles Martínez
Profesor Responsable:
Santi Caballé Llobet

Enero de 2024

Créditos/Copyright

Dedicatoria

En el momento de escribir estas palabras, me encuentro lleno de gratitud y emoción al completar mi Trabajo de Fin de Grado (TFG). Hoy, no puedo dejar pasar la oportunidad de expresar mi más profundo agradecimiento, en especial a ti, mi padre.

A lo largo de este arduo camino académico, has sido mi guía y apoyo incondicional. Tus sabias palabras, tu paciencia infinita y tu constante motivación han sido el motor que me ha impulsado a superar cualquier obstáculo. Tu dedicación hacia mi crecimiento intelectual y personal no tiene límites, y por ello, quiero dedicar este TFG a ti, como un símbolo de mi profunda admiración y gratitud.

También quiero extender mi agradecimiento a mi querida madre y hermana, quienes han estado a mi lado en cada paso de este camino. Su amor, aliento y comprensión han sido fundamentales para mantener mi determinación y confianza en mí mismo. Gracias por siempre creer en mí y por ser mi mayor fuente de inspiración.

Kevin Nava Argos

Ficha del Trabajo Final

Título del trabajo:	Desarrollo de una aplicación web para gestión de consultas médicas
Nombre del autor/a:	Kevin Nava Argos
Nombre del Tutor/a de TF:	Gregorio Robles Martínez
Nombre del/de la PRA:	Santi Caballé Llobet
Fecha de entrega:	01/2024
Titulación o programa:	Grado de Ingeniería Informática
Área del Trabajo Final:	Desarrollo Web
Idioma del trabajo:	Castellano
Palabras clave	Desarrollo web, aplicación web, consultas médicas, historia clínica, *React*, *Node.js*.

Resumen del Trabajo

El desarrollo web se basa en la creación de herramientas informáticas que permiten la integración de contenidos, facilitando el acceso a la información desde cualquier dispositivo. Las aplicaciones web surgen para abordar diversas necesidades, como la gestión de recursos y el acceso a información precisa. En el ámbito de la salud, las limitaciones tecnológicas dificultan la eficiencia, afectando a clínicas que aún utilizan métodos tradicionales.

La investigación propone desarrollar una aplicación web para la gestión de consultas médicas y otras especialidades. Utilizando tecnologías modernas como JavaScript, React y Node.js, la herramienta permitirá almacenar datos clínicos, programar consultas y reducir la dependencia de llamadas y gestiones manuales. Con segmentos de login diferenciados para profesionales, administradores y pacientes, la aplicación ofrecerá una interfaz con lista de consultas, historias clínicas y planner visualizable mensual, semanal y diariamente.

El enfoque es eficientizar la planificación diaria, semanal y mensual, reduciendo el tiempo de gestión para profesionales y recepcionistas. La aplicación se propone como software libre, permitiendo adaptaciones según necesidades individuales y facilitando la gestión eficiente incluso para aquellos en inicio de consultas sin capacidad para contratar programadores específicos.

Abstract

Web development is based on the creation of computer tools that enable content integration, facilitating access to information from any device. Web applications arise to address various needs, such as resource management and access to accurate information. In the field of health, technological limitations hinder efficiency, affecting clinics that still use traditional methods.

The research proposes developing a web application for the management of medical consultations and other specialties. Using modern technologies like JavaScript, React, and Node.js, the tool will allow storing clinical data, scheduling appointments, and reducing reliance on manual calls and management. With differentiated login segments for professionals, administrators, and patients, the application will provide an interface with a list of appointments, medical histories, and a planner that can be viewed monthly, weekly, and daily.

The focus is on streamlining daily, weekly, and monthly planning, reducing management time for professionals and receptionists. The application is proposed as open-source software, allowing adaptations according to individual needs and facilitating efficient management even for those starting consultations without the ability to hire specific programmers.

Índice

Lista de Figuras

Lista de Tablas

Lista de Códigos

1. Introducción

1.1. Resumen de la Propuesta

El desarrollo web se fundamenta en la creación de herramientas informáticas con las que es posible establecer una integración de contenidos y transformar los conocimientos en recursos al alcance de cualquier persona. Así, es posible que estas herramientas estén al alcance de cualquier persona y se reduzcan las limitaciones que tienen los medios físicos, facilitando a la población revisar informaciones desde sus computadoras o teléfonos, independientemente del lugar en el que se encuentren. (Hernández-Rodríguez, 2014; Redacción Guide IONOS, 2020).

Realizar aplicaciones web nace por la premisa de cubrir determinadas necesidades del público objetivo. Por ejemplo, carencia de recursos para administración y gestión, carencia de resúmenes de contenido, dificultades para acceder a información certera, necesidades de empresas, entre otras circunstancias. Todo esto lleva a que los programadores tengan que disponer de tecnologías específicas que se ajusten a las exigencias del mercado y permitan elaborar las aplicaciones deseadas. (Escudero Cuevas, 2019)

Un campo en el que muchas veces existen limitaciones en cuanto a las herramientas tecnológicas disponibles es el de las ciencias de la salud, en donde las aplicaciones pueden ser costosas, requerir de un manejo complejo que dificulta su uso, o hasta implicar que tengan deficiencias en algunos de los recursos que ponen a disposición. Esto ha dificultado que en muchas clínicas el abordaje de métodos tradicionales persista, limitando el acceso a herramientas que puedan gestionar las actividades de los profesionales de la salud de forma más eficiente.

Para entender esto mejor es fundamental tener presente que la salud es un modelo de negocio que puede ofrecer muchos beneficios al insertarse dentro de dicho modelo, debido a que no sólo es posible ganar a través de la formación profesional dentro de la salud propiamente dicha, sino que también es posible ofrecer herramientas y servicios para optimizar la práctica profesional. (Eryılmaz, 2019; Osorio et al., 2010)

Y es justamente en este contexto en el que encaja la presente investigación, ya que establece la posibilidad de programar una aplicación web adecuada para la gestión de consultas médicas, pero que pueda extenderse a las consultas y/o terapias ofrecidas por otros profesionales de la salud. Y en donde sea posible que se almacenen los datos clínicos del paciente y que son requeridos por el profesional, pero también esté al alcance la posibilidad de agendar nuevas consultas por parte del paciente, reduciendo el uso de llamadas a los centros de salud y planners de gestión manual que puedan implicar la necesidad de que las consultas sean por orden de llegada.

Así, el trabajo final que se pretende realizar tendrá por enfoque la construcción de una herramienta digital que utilice tecnologías modernas, incluyendo JavaScript, React y Node.js, para gestionar eficientemente la planificación de las consultas médicas. Esto

incluirá un segmento de login para el acceso de usuario (diferenciado en profesional, administrador y paciente), una página principal en la que pueda observarse una lista de las consultas programadas en orden específico, un área de historias clínicas en la que sea posible acceder a información de otras consultas realizadas al paciente y un planner que podrá visualizarse mensual, semanal y diariamente, en caso de que el profesional quiera reprogramar alguna consulta por problemas específicos.

Esta aplicación permitirá la reducción en los tiempos de planificación diaria, semanal y mensual tanto de los profesionales como de los recepcionistas del consultorio, además que reducirá también el uso de metodologías limitativas de gestión. Además, se espera que la aplicación sea de software libre, para que otras personas puedan ajustarla a sus necesidades y tener una gestión eficiente de sus actividades, incluso aunque sus consultas estén iniciando y no tengan la posibilidad de contratar programadores que elaboren algo específico para ellos.

1.2. Justificación, Interés y Relevancia de la Propuesta

La salud es una necesidad humana, si una persona no se encuentra saludable es muy difícil que pueda cubrir otras necesidades, por ello los entornos de la salud cada vez más están requiriendo la integración de herramientas digitales para que los problemas que antes se presentaban sean menos frecuentes. (Fernández Silano, 2014; Terrón et al., 2021) Ejemplo de ello fue la pandemia por COVID-19, en donde se vio la necesidad de implementar entornos digitales para gestionar las consultas médicas, reduciendo así la posibilidad de contagio al tener que asistir a los consultorios simultáneamente; sin embargo, pese a que se ha avanzado mucho en este ámbito, todavía existe mucho camino por recorrer. (Organización Internacional del Trabajo, 2021).

A raíz de eso, se están buscando nuevas estrategias que mejoren lo que actualmente se ha logrado y que permitan realmente controlar eficientemente las actividades dentro de los entornos médicos. Y esto fue principalmente lo que despertó el interés en el uso de la informática como herramienta para gestionar más eficientemente, al tiempo que se ofrece un servicio de calidad a quienes se desempeñan como profesionales sanitarios.

Esto ha llevado a plantear la realización de un estudio orientado a desarrollar una aplicación web para gestión de consultas médicas, que permitirá demostrar los conocimientos adquiridos en el ámbito del Desarrollo Web. De modo que se torna relevante tanto por sus aportes sanitarios, por sus aportes teóricos y metodológicos y por sus aportes académicos para los desarrolladores al ofrecer un software libre que permita la gestión de este tipo de consultas.

Con base en lo anterior, los aportes sanitarios estarán vinculados con la facilidad que otorgará a los profesionales de la salud en su actuación profesional. Los aportes teóricos estarán fundamentados en la información actualizada sobre la gestión de aplicaciones web y los elementos del frontend y del backend que puedan ser implementados en la aplicación diseñada. Los aportes metodológicos en lo referente a la realización de una herramienta

práctica. Y lo académico con la proyección del código, que estará disponible para cualquier desarrollador web o estudiante que desee realizar este tipo de prácticas.

1.3. Propuesta

Lo que se propone con este trabajo final es la creación de una aplicación web destinada a la gestión y planificación de consultas médicas. Todo ello con base en la posibilidad de reducir los tiempos de espera y el uso de herramientas anticuadas que no ofrecen la misma precisión durante la planificación de las actividades realizadas por los profesionales de la salud.

Teniendo en cuenta lo anterior, la motivación que me llevó a seleccionar justamente este tema fue saber que, durante la pandemia por COVID-19, murieron muchas personas por la cantidad de contagios que hubo, y que existen países en los que este problema fue mayor por no contar con una gestión eficiente. En algunos lugares del mundo, incluso, no se tenía un protocolo para evitar que permanecieran simultáneamente muchos pacientes en una misma sala de espera de consulta, en gran medida debido a que es costoso elaborar herramientas de gestión propias cuando se trabaja en lugares donde el acceso a la salud no es del todo eficiente.

Considerando todo esto, mi interés se despertó en ofrecer un recurso que pueda alcanzar cualquier región del mundo, que sea de fácil acceso y que se considere software libre, de modo que pueda servir tanto por su finalidad asistencial, como por el hecho de ser una herramienta con la que otros programadores podrían realizar prácticas profesionales para mejorar sus habilidades.

1.4. Objetivos

1.4.1. Objetivos Principales

Realizar el Trabajo Fin de Grado (TFG) sobre una aplicación web médica es un proyecto valioso que puede combinar conocimientos teóricos y habilidades prácticas en el campo de la medicina y la tecnología.

Los objetivos principales que me han llevado a desarrollar esta aplicación web médica son los siguientes:

- Analizar y desarrollar una aplicación web médica que aborde una necesidad específica dentro del campo de la salud, que mejore un área en la atención médica y que ofrezca soluciones innovadoras.
- Identificar y analizar las necesidades y problemas en el ámbito médico que la aplicación busca resolver.
- Validación técnica y funcional de la aplicación médica. Análisis exhaustivo de los

requisitos técnicos necesarios para desarrollar la aplicación, así como una evaluación de su funcionalidad y usabilidad y cómo se adapta a las necesidades de los profesionales de la salud y de los pacientes.

- Diseñar una interfaz de usuario intuitiva y atractiva que garantice una experiencia de usuario positiva.
- Implementar medidas de seguridad robustas para proteger la información médica y garantizar el cumplimiento de las normativas de privacidad.
- Documentar el proceso de desarrollo, la metodología utilizada y los resultados obtenidos.

1.4.2. Objetivos Personales

En este trabajo final de grado sobre la realización de una aplicación web médica, se persiguen los siguientes objetivos personales:

- Comprender en profundidad los desafíos y oportunidades que conlleva el desarrollo de soluciones tecnológicas en el ámbito de la salud.
- Adquirir conocimientos prácticos en el diseño y la implementación de aplicaciones web de calidad en el campo médico.
- Contribuir al mejoramiento de la atención médica a través de la creación de una aplicación web innovadora y efectiva para profesionales de la salud y pacientes.
- Ampliar y consolidar la experiencia previa con tecnologías conocidas, para perfeccionar mis habilidades y conocimientos en su aplicación en el ámbito de la salud.
- Explorar y aprender el uso de nuevas tecnologías relacionadas con el desarrollo de aplicaciones web médicas, con el fin de estar al tanto de las últimas tendencias y herramientas disponibles en el mercado.
- Adquirir conocimientos y experiencia en despliegues automatizados, para optimizar la implementación y actualización de la aplicación web en entornos de producción de manera eficiente y sin interrupciones.
- Obtener una valiosa experiencia en la gestión de un proyecto real, enfrentando desafíos y tomando decisiones relacionadas con la planificación, el diseño, el desarrollo y la evaluación de la aplicación web médica.

Estos objetivos personales son fundamentales para enriquecer mi formación académica y profesional, así como para adquirir las habilidades necesarias que me permitan destacar como desarrollador de aplicaciones web médicas capacitado y proactivo en el futuro.

1.4.3. Objetivos del Usuario

En el marco de este trabajo final de grado sobre la creación de una aplicación web médica, se establecen los siguientes objetivos para los usuarios de la aplicación:

- Proporcionar a los profesionales de la salud una herramienta innovadora y eficiente que les permita mejorar y agilizar la atención médica a sus pacientes, optimizando los procesos de diagnóstico, tratamiento y seguimiento.
- Facilitar a los pacientes el acceso a información médica confiable y actualizada, brindándoles la posibilidad de monitorear su estado de salud y gestionar citas médicas.
- Garantizar la seguridad y confidencialidad de los datos de los pacientes, implementando altos estándares de protección de la privacidad y cumpliendo con las regulaciones y leyes aplicables en materia de protección de datos de salud.
- Fomentar la colaboración y comunicación fluida entre los profesionales de la salud y los pacientes, permitiendo el intercambio de información relevante de manera segura y eficiente.
- Adaptar la aplicación web médica a las necesidades y requerimientos del usuario, brindando una interfaz intuitiva y fácil de usar, así como ofreciendo opciones de personalización para adaptarse a las preferencias individuales.

En resumen, los objetivos para los usuarios de la aplicación web médica se centran en mejorar la calidad de la atención médica, promover la participación activa de los pacientes en su cuidado de salud y garantizar la seguridad y privacidad de los datos personales.

1.4.4. Objetivos Secundarios

En la creación de la aplicación web médica, se plantean los siguientes objetivos secundarios:

- Elaborar una documentación exhaustiva y útil que sirva como referencia para el desarrollo, mantenimiento y utilización de la aplicación web médica, contribuyendo a la comprensión y facilitando futuras actualizaciones y mejoras.
- Establecer un sistema altamente automatizado para los despliegues de la aplicación web, con el objetivo de agilizar y simplificar el proceso de implementación en un entorno de producción, minimizando errores y tiempos de espera.
- Realizar pruebas rigurosas y completas para asegurar el correcto funcionamiento de la aplicación web en todos los aspectos, incluyendo funcionalidades, rendimientos y seguridad, garantizando así una experiencia óptima para los usuarios finales.
- Implementar un sistema de backup de datos confiable y seguro, con el propósito de resguardar la información generada y almacenada en la aplicación web médica, evitando la pérdida de datos y garantizando la continuidad del servicio.
- Brindar la posibilidad de personalización de la aplicación web médica según las preferencias y necesidades de cada usuario, permitiendo la configuración de elementos como colores, menús, preferencias de visualización, proporcionando así una experiencia más personalizada y agradable para los usuarios.

Estos objetivos secundarios contribuyen directamente al logro de los objetivos generales del proyecto, complementando el enfoque principal en la calidad asistencial y la experiencia del usuario en el ámbito de la atención médica.

1.5. Impacto en Sostenibilidad, Ético-social y de Diversidad

La creación de una aplicación web médica no solo implica el desarrollo de una herramienta tecnológica, sino también la oportunidad de generar un impacto significativo en diferentes aspectos de la sociedad.

En primer lugar, desde la perspectiva de la sostenibilidad, la creación de una aplicación web médica brinda la oportunidad de impulsar la sostenibilidad al reducir la dependencia de recursos físicos y minimizar el impacto ambiental. Al eliminar la necesidad de papel y otros materiales en los procesos médicos, se disminuye la deforestación y el consumo excesivo de recursos naturales. Además, al facilitar la telemedicina y la comunicación remota entre pacientes y profesionales de la salud, se reducen los desplazamientos y las emisiones de carbono asociadas. Esta transición hacia una práctica más eco amigable contribuye a la construcción de un futuro sostenible y respetuoso con el medio ambiente

Además, en términos ético-sociales, la aplicación web médica tiene un impacto significativo en términos ético-sociales, ya que garantiza un acceso más equitativo y accesible a los servicios de salud. Superando las barreras geográficas, los pacientes pueden acceder a servicios médicos independientemente de su ubicación física. Esto es especialmente beneficioso para las áreas rurales o remotas, donde la accesibilidad a la atención médica puede ser limitada. Además, al facilitar la comunicación entre pacientes y profesionales de la salud, la aplicación web fomenta una relación más directa y colaborativa, basada en la confianza y la participación activa del paciente en su propio cuidado.

Por último, la creación de la aplicación web médica también se enfoca en fomentar la diversidad en el ámbito de la salud. Al permitir la interacción entre pacientes y profesionales de diferentes culturas y antecedentes, se promueve una atención médica más holística y culturalmente sensible. Esto implica considerar las perspectivas individuales, creencias y prácticas culturales para ofrecer una atención médica personalizada y respetuosa. Además, al otorgar a los pacientes la posibilidad de elegir entre una amplia gama de profesionales de la salud, se promueve la inclusión y la igualdad de oportunidades en el campo médico.

En resumen, la creación de una aplicación web médica no solo representa un avance tecnológico, sino también una oportunidad para generar un impacto positivo en términos de sostenibilidad, ética social y diversidad. Esta herramienta permite avanzar hacia una atención médica más sostenible, equitativa y centrada en el paciente, superando barreras geográficas y culturales para proporcionar cuidados a todas las personas, independientemente de su ubicación o trasfondo.

1.6. Metodología

"En la era digital actual, la tecnología ha transformado muchos aspectos de nuestras vidas, incluido el ámbito de la atención médica. La creciente demanda de soluciones efectivas y accesibles para el cuidado de la salud ha llevado al desarrollo de aplicaciones web médicas, que brindan a los profesionales de la salud y a los pacientes herramientas innovadoras para mejorar la calidad de la atención y facilitar la gestión de la información clínica.

En este trabajo de grado, se analizará una metodología para el desarrollo de la aplicación web médica con el objetivo de ofrecer una guía práctica y sistemática para aquellos que deseen embarcarse en este desafiante proceso. Se explorarán las etapas clave del desarrollo, desde la planificación inicial hasta la implementación y evaluación final, abordando aspectos como el diseño de la interfaz de usuario, la seguridad de los datos y la integración con sistemas de información existentes.

Después de analizar el concepto inicial, nos hemos enfocado en identificar los criterios fundamentales para la creación de la aplicación. Hemos comenzado a generar la documentación necesaria y a llevar a cabo una planificación minuciosa del proyecto. Esta etapa marca el inicio oficial del desarrollo y sentará las bases para el éxito del mismo. Durante esta fase inaugural, nos aseguraremos de establecer los objetivos claros y definir los requerimientos específicos para guiar todo el proceso de desarrollo de la aplicación.

Una vez establecidas las bases y con todo claro en la fase anterior, además de toda la información recopilada comenzaremos la fase de análisis de requerimientos: En esta etapa inicial, se realiza una evaluación exhaustiva de los requerimientos y necesidades específicas de la aplicación web médica. Esto implica identificar los objetivos del proyecto, las funcionalidades requeridas, los usuarios finales y los requisitos legales y regulatorios aplicables.

A continuación, realizaremos la etapa de diseño de la interfaz de usuario: En esta etapa, se crean los diseños y prototipos de la interfaz de usuario de la aplicación web. Se busca una interfaz intuitiva y fácil de usar para los usuarios finales, teniendo en cuenta factores como la accesibilidad, la usabilidad y la experiencia del usuario.

Durante el proyecto se realiza el desarrollo del backend: En esta etapa se desarrollan los componentes y la lógica del lado del servidor de la aplicación web. Esto implica manejar la lógica de negocio, la gestión de bases de datos y la integración con otros sistemas de información médica existentes, como sistemas de gestión de pacientes o registros electrónicos de salud.

Otras de los pasos realizados es el desarrollo del frontend: En esta etapa se trabaja en el desarrollo de la interfaz de usuario de la aplicación web. Se utilizan tecnologías de diseño web, como HTML, CSS y JavaScript, para crear una interfaz atractiva y responsiva. También se incorporan frameworks y la biblioteca React para facilitar el desarrollo del frontend.

Durante la fase de desarrollo y creación de la aplicación web, se han realizado diferentes

pruebas y depuración: Una vez que la aplicación web está completa, se han llevan a cabo pruebas exhaustivas para detectar y corregir errores y garantizar su correcto funcionamiento. Las pruebas que ser realizaron abarcaban diferentes aspectos relacionados con pruebas de funcionalidad, pruebas de rendimiento, pruebas de seguridad y pruebas de compatibilidad en diferentes navegadores y dispositivos.

Como parte última del proyecto se realiza la implementación: En esta etapa, la aplicación web médica se despliega en un entorno de producción para su uso por parte de los usuarios finales. Esto puede implicar la configuración de servidores, la instalación de bases de datos y la integración con otros sistemas.

1.7. Fundamentos Teóricos

1.7.1. HTML

El HTML es un lenguaje de marcas que define la estructura del contenido en una página web mediante elementos que encierran y formatean el contenido. Estos elementos constan de una etiqueta de apertura, una etiqueta de cierre y contenido, y pueden tener atributos para proporcionar información adicional. Los elementos pueden anidarse creando una estructura jerárquica, y algunos elementos son "vacíos" y no requieren etiqueta de cierre (Kolade, 2021; Mozilla Foundation, 2023a; Willard, 2009).

Además, los elementos pueden tener atributos que proporcionan información adicional sobre el elemento, pero no se muestra en el contenido visible de la página web. Estos atributos se representan con un nombre y un valor, separados por un signo igual, y se encuentran dentro de las etiquetas de apertura. (Kolade, 2021; Mozilla Foundation, 2023a).

1.7.2. CSS

CSS (*Cascading Style Sheet*), o Hojas de Estilo en Cascada, es un lenguaje de marcado que se utiliza para controlar la presentación y el diseño de documentos HTML en un navegador web. Mientras que el HTML se encarga de marcar y estructurar el contenido de una página web, el CSS permite especificar cómo se verá ese contenido. Con CSS, los desarrolladores pueden definir cómo se muestran los encabezados, los párrafos, los enlaces y otros elementos HTML en términos de colores, fuentes, tamaños y diseños (Geneves et al., 2012; Mozilla Foundation, 2023c).

Un archivo CSS está compuesto por reglas que describen cómo deben estilizarse elementos HTML específicos. Cada regla consta de un selector que apunta a un elemento HTML y una serie de declaraciones que especifican cómo ese elemento debe lucir. En CSS, existen muchas propiedades y valores diferentes que se pueden utilizar para personalizar la apariencia de los elementos HTML. CSS se organiza en módulos, y cada módulo se centra en un aspecto particular de la presentación, como fondos y bordes, tipografía, colores, entre otros. Cada módulo tiene su propia especificación que detalla cómo se debe implementar. Los navegadores web implementan estas especificaciones de

manera gradual, lo que significa que no todos los navegadores admiten todas las características de CSS al mismo tiempo (Geneves et al., 2012; Mozilla Foundation, 2023c).

1.7.3. JavaScript

JavaScript es un lenguaje de programación esencial en el desarrollo web, utilizado para agregar dinamismo a sitios y aplicaciones. A diferencia de HTML y CSS, que son lenguajes de marcado, JavaScript permite realizar cálculos, modificar dinámicamente el contenido HTML en el DOM, aplicar estilos en tiempo real y más. Se puede incorporar en una página web de varias formas, ya sea a través de código en línea en atributos de etiquetas HTML, directamente en la página con la etiqueta <script>, o en un archivo externo vinculado a la página. JavaScript admite diversos tipos de datos y variables, y ofrece operadores de declaración (Kantor, 2022; Mozilla Foundation, 2023d).

En el desarrollo web moderno, los frameworks de JavaScript desempeñan un papel crucial para crear aplicaciones escalables y mantenibles. La elección del framework adecuado depende de las necesidades y objetivos del proyecto. Algunos de los frameworks populares incluyen AngularJS, Angular, React y Vue.js, cada uno con sus propias características y ventajas. AngularJS y Angular son desarrollos de Google, React es una biblioteca de Facebook que se basa en componentes, y Vue.js se centra en el patrón Modelo-Vista-Controlador (MVC) y se integra eficazmente con otras bibliotecas y proyectos existentes. La elección de un framework depende de consideraciones específicas del proyecto y los objetivos del desarrollo web (Duvander y Romhagen, 2019).

1.7.4. TypeScript

TypeScript se presenta como un estricto superset de JavaScript, permitiendo la implementación de cualquier cosa factible en JavaScript, al tiempo que ofrece características adicionales. Este lenguaje de programación orientado a objetos de código abierto y fuertemente tipado facilita la integración en proyectos de JavaScript, ya que el código TypeScript puede transpilarse a código JavaScript. Al optar por TypeScript, especialmente en proyectos a gran escala, los desarrolladores pueden lograr un software más sólido en comparación con las aplicaciones tradicionales de JavaScript, aunque no garantiza un código sin errores. TypeScript sobresale en la prevención de errores relacionados con tipos y mejora la funcionalidad de IntelliSense, contribuyendo a un código más confiable (Microsoft, 2013; Redacción GeeksforGeeks, 2023).

La evolución de JavaScript ha llevado a estándares cambiantes, y TypeScript surgió para enriquecer JavaScript con características avanzadas. Proporciona los bloques de construcción básicos de JavaScript y transpila todo el código de TypeScript a su equivalente en JavaScript para su ejecución. El soporte de TypeScript para clases y objetos es un factor clave en su creciente popularidad, ya que lo hace más accesible para comprender e implementar conceptos de programación orientada a objetos en comparación con el enfoque nativo de JavaScript basado en prototipos. Esto ha posicionado a TypeScript como una elección valiosa, especialmente dentro de los populares marcos de JavaScript y bibliotecas (Redacción GeeksforGeeks, 2023).

1.7.5. React

React es una versátil biblioteca que no se limita a aplicaciones web, ya que se puede utilizar con otras bibliotecas como React Native para aplicaciones móviles o React 360 para aplicaciones de realidad virtual. Aunque React se describe como una biblioteca, en muchos casos se utiliza como un framework para abordar desafíos similares a los marcos tradicionales de desarrollo web. Su objetivo principal es minimizar errores comunes en la construcción de interfaces de usuario mediante el uso de componentes, unidades lógicas y autónomas que describen partes específicas de la interfaz. React maneja eficientemente el renderizado de componentes, lo que simplifica la actualización y renderizado cuando los datos cambian, mejorando la predictibilidad del código y facilitando la depuración (Meta Platforms Inc., 2017; Mozilla Foundation, 2023b).

React se destaca por la construcción de componentes encapsulados que gestionan su propio estado, lo que permite crear interfaces de usuario más complejas y mantener el estado fuera del DOM. La biblioteca no impone restricciones en el conjunto tecnológico utilizado y permite desarrollar nuevas funcionalidades sin necesidad de reescribir el código existente. React es compatible con el renderizado en el servidor usando Node.js y el desarrollo de aplicaciones móviles con React Native. Los componentes en React implementan un método "render()" que toma datos de entrada y devuelve la representación visual, y se utiliza una sintaxis similar a XML llamada JSX. Además, React puede integrarse con bibliotecas externas, como "remarkable," para tareas específicas, lo que ofrece una gran flexibilidad en el desarrollo de aplicaciones (Meta Platforms Inc., 2017; Mozilla Foundation, 2023b).

1.7.6. Node.js

En cuanto a Node.js, este es un entorno de ejecución de JavaScript que está diseñado para aplicaciones de red escalables y utiliza un enfoque asincrónico y basado en eventos para gestionar múltiples conexiones de manera concurrente. Su característica distintiva es su modelo de concurrencia, que evita bloqueos en el proceso y permite desarrollar sistemas escalables sin bloqueos en las operaciones de entrada/salida (I/O) (OpenJS Foundation, 2011; Semah, 2022).

A diferencia de otros sistemas similares, Node.js entra automáticamente en el bucle de eventos después de ejecutar el script de entrada y sale cuando no hay más devoluciones de llamada por atender, siguiendo un comportamiento similar al JavaScript del navegador (OpenJS Foundation, 2011; Semah, 2022).

Node.js se destaca por su enfoque en HTTP, diseñado para la transmisión y baja latencia, lo que lo hace adecuado como base para bibliotecas o frameworks web. A pesar de su diseño sin hilos, Node.js permite aprovechar múltiples núcleos en un entorno mediante procesos secundarios y el módulo de agrupación (cluster module) (OpenJS Foundation, 2011; Semah, 2022).

Este entorno también permite a los desarrolladores crear aplicaciones tanto del lado del cliente como del servidor en JavaScript, y fue lanzado en 2009 por Ryan Dahl. Node.js es

de código abierto y multiplataforma, ejecutándose en sistemas operativos como Linux, macOS y Windows (Semah, 2022).

En cuanto a las diferencias con el navegador, mientras el navegador permite acceso al DOM, Node.js se enfoca en proporcionar acceso a recursos del sistema y operaciones de archivo. Además, Node.js ofrece un mayor control sobre las versiones de tiempo de ejecución y admite módulos CommonJS y ES, a diferencia de los navegadores que tienen limitaciones con los módulos ES (Semah, 2022).

1.7.7. Express

Express.js, el marco web más destacado para Node.js, está diseñado para crear aplicaciones web y APIs y es considerado el estándar de facto en términos de marcos de servidor para Node.js. Simplifica el desarrollo de aplicaciones *backend* al proporcionar un conjunto robusto de características. Desde el enrutamiento de solicitudes HTTP hasta la gestión de *middleware*, Express agiliza la creación de aplicaciones al reducir la necesidad de escribir código inicial repetitivo (OpenJS Foundation, 2010).

Su flexibilidad permite a los desarrolladores estructurar su código de la manera que prefieran y ofrece una gama de métodos y middleware para administrar solicitudes y respuestas. Esto facilita la creación de rutas y la implementación de lógica empresarial en aplicaciones web (OpenJS Foundation, 2010; Redacción Codecademy, 2020).

Un aspecto esencial de Express es su capacidad para manejar solicitudes HTTP y definir rutas de manera eficiente lo cual simplifica la creación de rutas y reduce la carga de trabajo de los desarrolladores al proporcionar un flujo de trabajo más rápido y efectivo. Además, Express permite la aplicación de *middleware*, que se puede personalizar y encadenar según las necesidades específicas del proyecto. (Redacción Codecademy, 2020).

1.7.8. MySQL

MySQL es una destacada base de datos relacional de código abierto ampliamente utilizada en todo el mundo. Forma parte integral de la pila LAMP (Linux, Apache, MySQL, PHP/Perl/Python), que respalda muchas aplicaciones y servicios populares. Clasificada como la segunda base de datos más popular a nivel mundial, su versatilidad y fiabilidad son altamente valoradas (Google, 2023).

MySQL es un sistema de gestión de bases de datos relacionales de código abierto que almacena datos en tablas compuestas por filas y columnas. Utiliza el lenguaje de consulta estructurada (SQL) para definir, manipular y consultar datos (Google, 2023).

Siendo de código abierto, MySQL ha evolucionado en colaboración con la comunidad de usuarios durante más de 25 años, y su disponibilidad gratuita ha dado lugar a tenedores como *MariaDB* y *Percona Server for MySQL*. Es considerada una base de datos relacional, que organiza datos en relaciones predefinidas almacenadas en tablas, y estas relaciones son fundamentales para entender la estructura de los datos. Este sistema se utiliza en diversas aplicaciones, incluyendo aplicaciones web, procesamiento de transacciones en

línea y comercio electrónico, destacándose por su facilidad de uso y eficiencia en la gestión de datos (Google, 2023).

1.7.9. MongoDB

MongoDB es una base de datos no relacional de código abierto que se diferencia de MySQL al almacenar datos en documentos flexibles en lugar de tablas, lo que brinda flexibilidad y escalabilidad para aplicaciones multiplataforma. Es ideal para situaciones donde la disponibilidad y velocidad son prioritarias, y su esquema dinámico permite adaptarse a cambios en la estructura de datos. MongoDB es especialmente adecuado para aplicaciones móviles y análisis en tiempo real, y se destaca en sistemas de gestión de contenido y almacenamiento de datos empresariales cuando se combina con Apache Hadoop (IBM, 2021).

Entre sus características notables, MongoDB ofrece equilibrio de carga para garantizar disponibilidad y rendimiento, la capacidad de realizar consultas ad hoc sin esquemas predefinidos y soporte para múltiples lenguajes de programación. Estas cualidades hacen de MongoDB una elección sólida para empresas que buscan una base de datos versátil y escalable (IBM, 2021).

1.8. Planificación

El desarrollo del proyecto se ha estructurado en cinco fases principales, las cuales incluyen las cuatro Pruebas de Evaluación Continua (PEC) y la Defensa Final del proyecto.

La siguiente programación temporal ha sido elaborada considerando una serie de requisitos generales que la aplicación debe cumplir. Este plan de trabajo puede sufrir modificaciones a lo largo del proyecto, con el fin de adaptarse a los cambios necesarios para brindar un mayor valor al usuario final.

En el diagrama de Gantt (figura 1), se han establecido hitos destacados correspondientes a las entregas de las diferentes fases del proyecto divididas en las siguientes PECs (Pruebas de Evaluación Continua):

- **PEC1: Plan de Trabajo.**

PEC 1	27/09/2023	11/10/2023	Propuesta de trabajo: - Plan de trabajo. - Temática a desarrollar. - Explicación de la aplicación que se quiere construir. - Planificación detallada. - Herramientas y frameworks que se utilizarán.

Tabla 1. Planificación del trabajo PEC1

- **PEC2: Análisis y Diseño.**

PEC 2	12/10/2023	11/11/2023	Análisis y diseño: - Elaboración de la memoria que incluya los requerimientos, el análisis y diseño. - Estudio tecnología escogida, software, frameworks. - Elaboración de la base del estado del arte. - Instalación y pruebas de funcionamiento.

Tabla 2. Planificación del trabajo PEC2

- **PEC3: Implementación y Pruebas.**

PEC 3	12/11/2023	02/01/2024	Implementación y pruebas: - Desarrollo y descripción del alcance y los objetivos. - Desarrollo de la aplicación. - Despliegue de la aplicación. - Entrega aplicación.

Tabla 3. Planificación del trabajo PEC3

- **PEC4: Memoria y Presentación.**

PEC 4	03/01/2024	19/01/2024	Memoria y presentación: - Cierre y presentación memoria. - Presentación virtual del trabajo fin de grado.

Tabla 4. Planificación del trabajo PEC4.

- **Defensa.**

Defensa	29/01/2024	02/02/2024	Defensa: - Ante el Tribunal que evaluará el trabajo realizado. - Contestación a las preguntas que plantee el Tribunal.

Tabla 5. Planificación del trabajo defensa

Este enfoque de planificación asegurará una gestión eficiente del tiempo y una correcta secuencia de actividades, garantizando la satisfacción de los requisitos establecidos y la entrega exitosa de todos los productos y documentación requeridos en cada etapa del proyecto.

A continuación, se presenta la planificación del proyecto basada en la asignación de tres horas diarias para su realización. Se ha elaborado la siguiente tabla que muestra el detalle de las distintas fases del proyecto, incluyendo las fechas de inicio y finalización de cada ítem:

NOMBRE ACTIVIDAD	DÍAS	INICIO	FINAL	HORAS
PROPUESTA DEL TRABAJO	**15**	**27-09-2023**	**11-10-2023**	**60**
Búsqueda y organización de información	5	27-09-2023	01-10-2023	20
Elaboración resumen propuesta, justificación, interés y relevancia de la propuesta	5	02-10-2023	06-10-2023	20
Creación índice y enfoque del trabajo	2	07-10-2023	08-10-2023	8
Planificación del trabajo	2	09-10-2023	10-10-2023	8
Entrega y finalización PEC 1	1	11-10-2023	11-10-2023	4
ANÁLISIS Y DISEÑO	**31**	**12-10-2023**	**11-11-2023**	**124**
Redacción memoria	3	12-10-2023	14-10-2023	12
Configuración de Node.js	2	15-10-2023	16-10-2023	8
Configuración base de datos	10	17-10-2023	26-10-2023	40
Lógica de la APP	7	27-10-2023	03-11-2023	28
APIs REST	3	03-11-2023	05-11-2023	12
Creación del Middleware	5	06-11-2023	10-11-2023	20
Testing y entrega PEC 2	1	11-11-2023	11-11-2023	4
IMPLEMETACIÓN Y PRUEBAS	**52**	**12-11-2023**	**02-01-2024**	**208**
Configuración de React y Java Script	8	12-11-2023	19-11-2023	32
Desarrollo de interfaces	7	20-11-2023	26-11-2023	28
Conexión con Backend	4	27-11-2023	30-11-2023	16
Testing	2	01-12-2023	02-12-2023	8
Integración Frontend-Backend	5	03-12-2023	07-12-2023	20
Testing completo	4	08-12-2023	11-12-2023	16
Evaluación de usabilidad	2	12-12-2023	13-12-2023	8
Redacción memoria	2	14-12-2023	15-12-2023	8
Preparación producción	5	16-12-2023	20-12-2023	20
Lanzamiento producción	5	21-12-2023	25-12-2023	20
Preparación video demostración	2	26-12-2023	27-12-2023	8
Grabación video demostración	1	28-12-2023	28-12-2023	4
Preparación y entrega PEC 3	5	29-12-2023	02-01-2024	20
MEMORIA Y PRESENTACION	**17**	**03-01-2024**	**19-01-2024**	**68**
Pruebas	1	03-01-2024	03-01-2024	4
Redacción memoria	8	04-01-2024	11-01-2024	32
Preparación power point	3	12-01-2024	14-01-2024	12
Preparación y grabación video presentación	4	15-01-2024	18-01-204	16
Preparación y entrega PEC	1	19-01-2024	19-01-2024	4
DEFENSA	**5**	**29-01-2024**	**02-02-2024**	**20**
PREPARACIÓN Y DEFENSA	5	29-01-2024	02-02-2024	20

Tabla 6. Planificación del trabajo detallada

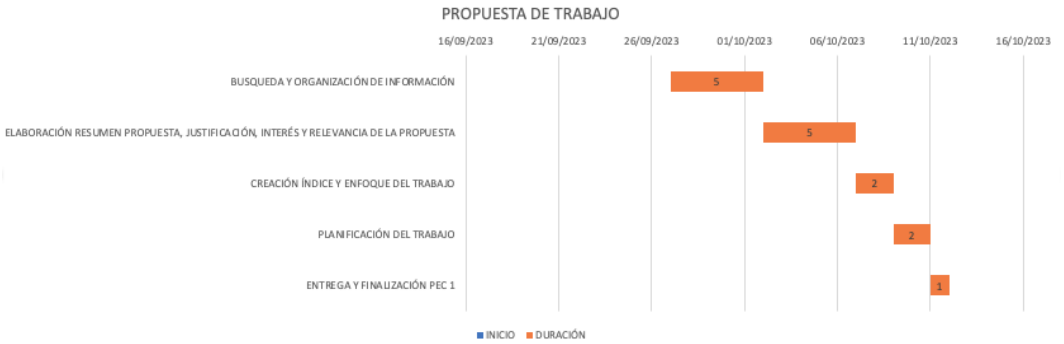


Figura 1. Propuesta de Trabajo

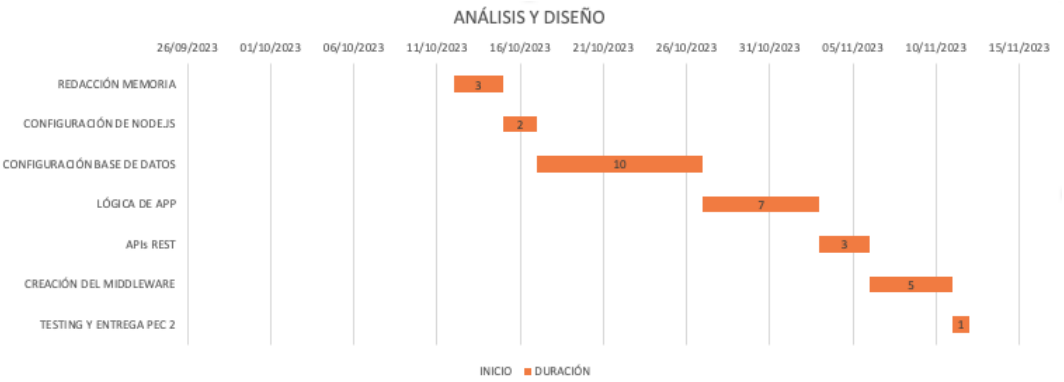


Figura 2. Análisis y Diseño

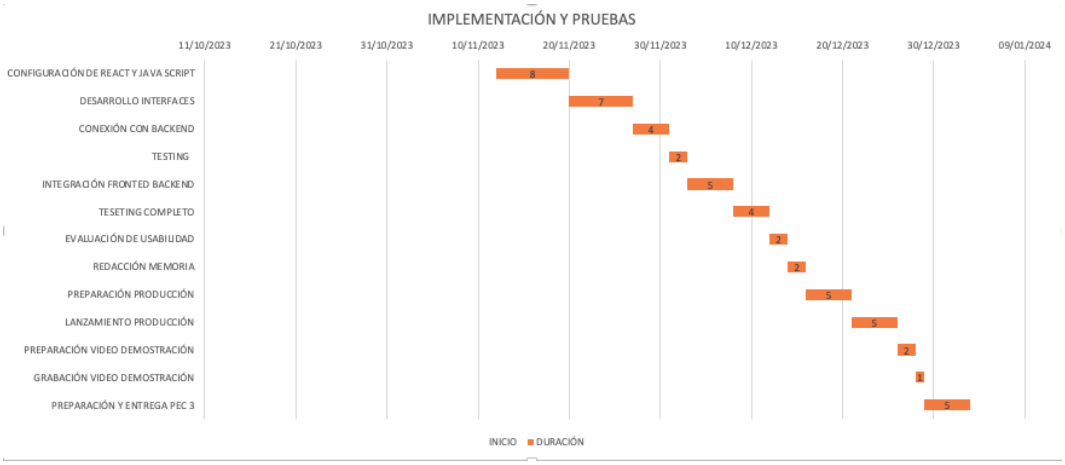


Figura 3. Implementación y Pruebas

Figura 4. Memoria, Presentación y Defensa

1.9. Evaluación de Riesgos

A continuación, se presenta una evaluación de riesgos que se pueden presentar durante la confección del proyecto.

- Riesgo de falta de conocimiento técnico: Si el estudiante no cuenta con un conocimiento sólido de las tecnologías web y médicas necesarias, puede haber dificultades al desarrollar la aplicación. Para mitigar este riesgo, es recomendable dedicar tiempo a investigar y aprender sobre las tecnologías y estándares relevantes, así como buscar recursos educativos y/o contar con la asesoría de profesionales de la industria.
- Riesgo de falta de acceso a datos médicos: Para desarrollar una aplicación web médica, es importante contar con datos médicos reales o simulados para su implementación y pruebas. Si no se tiene fácil acceso a estos datos, puede retrasar el progreso del proyecto. Para mitigar este riesgo, se puede explorar la posibilidad de colaborar con instituciones médicas o grupos de investigación que estén dispuestos a proporcionar datos ficticios o anonimizados para su uso en el proyecto.
- Riesgo de integración con sistemas existentes: Si la aplicación web médica necesita integrarse con sistemas o infraestructuras existentes en entornos de atención médica, puede ser un desafío técnico complejo. Para mitigar este riesgo, se debe investigar y comprender las interfaces y estándares utilizados en estos entornos, y trabajar en estrecha colaboración con profesionales y administradores de sistemas médicos para garantizar una correcta integración.
- Riesgo de seguridad de datos: Es fundamental garantizar la seguridad de los datos médicos en la aplicación web. Existen riesgos como el robo de información o la vulnerabilidad a ciberataques. Para mitigar este riesgo, se deben incorporar medidas de seguridad adecuadas, como el cifrado de datos, autenticación de usuarios, gestión de permisos de acceso y seguimiento de las mejores prácticas de seguridad informática en el desarrollo y despliegue de la aplicación.
- Riesgo de cumplimiento normativo: Las aplicaciones web médicas deben cumplir con las regulaciones y normativas de protección de datos, privacidad y seguridad médica. Para mitigar este riesgo, es fundamental investigar y comprender las obligaciones legales y normativas pertinentes y asegurarse de que la aplicación cumpla con ellas, incluyendo la obtención de los consentimientos necesarios y la implementación de políticas de privacidad adecuadas.

2. Contenidos

La estructura de la aplicación web seguirá un formato tradicional compuesto por tres secciones principales: la parte superior, parte lateral izquierda y el área central.

La parte superior se compone por la cabecera, en ella se distingue los siguientes elementos:

- Logo de la aplicación web.
- Botón de notificaciones.
- Botón de inicio.
- Botón ajustes.
- Botón de cerrar sesión.

Actualmente los botones de ajustes y notificaciones no están funcionando debido a que aún no han sido implementados. Esta funcionalidad específica se ha dejado para futuras actualizaciones.

En la sección central de la pantalla, encontramos dos áreas interactivas: la parte izquierda y el centro de la zona que se componen de los siguientes elementos:

- La parte lateral izquierda formado por un menú el que contiene: botón de perfil, botón historia y botón de consulta.
- En la zona central: zona de visualización de datos, formularios y gestor de consulta en tiempo real.

La estructura establecida se mantendrá a lo largo de todo el proyecto en las diferentes páginas de la aplicación web. Al seguir esta estructura y proporcionar una navegación intuitiva entre las diferentes páginas, la aplicación web de gestión de consultas médicas garantizará un flujo de trabajo fluido y una gestión óptima de la información médica de los pacientes.

3. Análisis Funcional y Diseño Técnico

Para realizar el análisis funcional de la aplicación web médica implica la identificación y definición de los requerimientos funcionales y los requerimientos no funcionales.

3.1. Análisis Funcional

Definición del problema:

- Identificación y compresión de las necesidades y problemas que la aplicación médica pretende abordar.
- Análisis de las deficiencias en la atención médica actual que la aplicación busca resolver.

1. **Requisitos del usuario:**
   - Entrevistas con los profesionales de la salud y potenciales usuarios para recopilar requisitos.
   - Creación de un documento de requisitos que incluya funciones esenciales y características deseadas.
2. **Diagramas de flujo:**
   - Desarrollo de diagramas que representen la interacción del usuario con la aplicación.
   - Visualización de los procesos internos de la aplicación para comprender su flujo de trabajo.
3. **Historias de usuario:**
   - Desglose de los requisitos en historias de usuario específicas.
   - Establecimiento de criterios de aceptación para cada historia de usuario.
4. **Prototipos y maquetas:**
   - Creación de prototipos de la interfaz de usuario para visualizar la apariencia y la disposición de los elementos.
   - Interacción y refinamiento del diseño con base en la retroalimentación de los usuarios.

3.2. Casos de Uso

Se dispone de 2 roles diferentes. Dentro del rol de usuarios tenemos dos actores (Usuarios Autenticados y Usuarios No Autenticados) con diferentes acciones. Los actores serán los siguientes:

- Los Pacientes, que podrán visualizar la historia médica, modificar la historia médica,

acceder a las consultas, visualizar las consultas, crear nuevas consultas, acceder al chat con el médico.

- Los Médicos, que podrán acceder al panel de inicio para médicos, pantalla de consulta detallada.
- Los Usuarios No Autenticados, que podrán iniciar sesión o registrarse.
- Los Usuarios Autenticados, que podrán acceder al perfil, modificar los datos en el perfil o cerrar sesión.

3.2.1. Casos de uso Usuarios No Autenticados

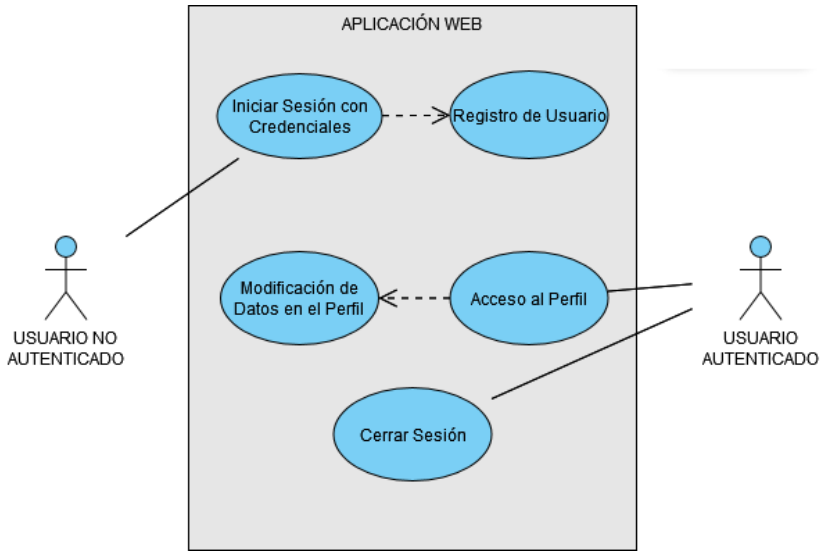


Figura 5. Caso de Uso de Usuario No Autenticado y Usuario Autenticado

Caso de Uso	Registro de Usuario	Identificador: **CU-01**
Actores	Usuario No Autenticado	
Tipo	Primario	
Referencias	Requiere acceso a la funcionalidad de registro del sistema.	
Precondición	El usuario no debe tener una cuenta registrada en el sistema.	
Postcondición	El usuario obtiene una cuenta registrada en el sistema y puede acceder a las funcionalidades personalizadas.	
Descripción	1. El usuario selecciona la opción de registro desde la pantalla principal de la aplicación. 2. Completa los campos obligatorios: a. Usuario: [Campo de texto] b. Correo electrónico: [Campo de texto] c. Contraseña: [Campo de contraseña] d. Soy médico: [Selección Si/No] 3. Si el usuario selecciona “Si” en la opción “Soy médico”, se le otorga el rol de médico. 4. El sistema verifica que la dirección de correo electrónico no esté asociada a otra cuenta existente.	

	5. Si la verificación es exitosa, el sistema crea una cuenta nueva para el usuario con el rol correspondiente. 6. Una vez verificado, el usuario puede iniciar sesión con las credenciales proporcionadas durante el registro.
Resumen	Usuarios no registrados crean cuentas con nombre, correo y contraseña. Pueden indicar si son médicos para obtener un rol especial. Después del registro, pueden acceder a sus cuentas.

Tabla 7. CU-01 – Registro de Usuario

Caso de Uso	Iniciar Sesión con Credenciales	Identificador: **CU-02**
Actores	Usuario No Autenticado	
Tipo	Primario	
Referencias	Requiere acceso a la funcionalidad de autenticación del sistema.	
Precondición	El usuario debe tener una cuenta registrada en el sistema.	
Postcondición	El usuario obtiene acceso a las funciones protegidas del sistema o recibe un mensaje de error si la autentificación falla.	
Descripción	1. El usuario ingresa su nombre de usuario y contraseña en los campos correspondientes de la pantalla de inicio de sesión. 2. El sistema verifica la autenticidad de las credenciales del usuario. 3. Si las credenciales son válidas, el sistema otorga acceso al usuario a las funciones protegidas. 4. Si las credenciales son inválidas, se muestra un mensaje de error y se da al usuario la opción de intentar nuevamente.	
Resumen	Usuarios registrados acceden con sus credenciales para disfrutar de funciones personalizadas y protegidas, siendo verificadas las credenciales para garantizar el acceso.	

Tabla 8. CU-02 – Iniciar Sesión con Credenciales

3.2.2. Casos de uso Usuarios Autenticados

Caso de Uso	Cerrar Sesión	Identificador: **CU-03**
Actores	Usuario Autenticado	
Tipo	Primario	
Referencias	Se activa al hacer clic en el icono de cerrar sesión en el menú.	
Precondición	El usuario debe haber iniciado sesión en el sistema.	
Postcondición	El usuario cierra la sesión y regresa a la pantalla de inicio de sesión.	
Descripción	1. El usuario, ya sea un paciente o un médico, accede al sistema e inicia sesión. 2. En la esquina superior derecha de la interfaz, el usuario visualiza el icono de cerrar sesión en el menú. 3. El usuario hace clic en el icono de cerrar sesión. 4. El sistema cierra la sesión actual del usuario. 5. El usuario es redirigido a la pantalla de inicio de sesión. 6. La información de la sesión anterior se borra y el usuario debe volver a autenticarse para acceder a las funciones del sistema.	
Resumen	Usuarios (pacientes o médicos) pueden cerrar sesión de forma segura al hacer clic en el icono correspondiente, siendo redirigidos a la pantalla de inicio de sesión para garantizar privacidad y seguridad.	

Tabla 9. CU-03 – Cerrar Sesión

Caso de Uso	Acceso al Perfil	Identificador: **CU-04**
Actores	Usuario Autenticado	
Tipo	Primario	
Referencias	Requiere acceso a la funcionalidad del perfil en el sistema.	
Precondición	El usuario debe haber iniciado sesión.	
Postcondición	El usuario accede a su perfil y puede visualizar la información asociada.	
Descripción	1. El usuario hace clic en el botón “Perfil” en el menú de navegación. 2. El sistema carga la página del perfil, que muestra la información asociada al usuario, como ID de usuario, nombre de usuario y correo electrónico. 3. Se presentan opciones adicionales en el perfil, como un botón para modificar los datos.	
Resumen	Usuarios autenticados pueden ver su perfil desde el dashboard haciendo clic en "Perfil", donde se muestra información como ID de usuario, nombre y correo electrónico.	

Tabla 10. CU-04 – Acceso al Perfil

Caso de Uso	Modificación de Datos en el Perfil	Identificador: **CU-05**
Actores	Usuario Autenticado	
Tipo	Primario	
Referencias	Requiere acceso a la funcionalidad de modificación de perfil en el sistema.	
Precondición	El usuario debe haber iniciado sesión. El usuario está en la página del perfil.	
Postcondición	Los datos modificados se actualizan en la base de datos del sistema.	
Descripción	1. Desde la página del perfil, el usuario hace clic en el botón “EDITAR INFORMACIÓN”. 2. Se le presenta un formulario prellenado con la información actual, incluyendo el ID de usuario, el nombre de usuario (que permanecen bloqueados y no se pueden modificar) y el correo electrónico. 3. El usuario puede modificar el campo del correo electrónico, si así lo desea. 4. Después de realizar los cambios, el usuario hace clic en el botón "GUARDAR". 5. El sistema verifica la validez de los datos modificados y actualiza la información en la base de datos. 6. El usuario recibe una confirmación de que los cambios se realizaron correctamente.	
Resumen	Usuarios autenticados pueden editar su perfil, incluyendo el cambio de correo electrónico. Al hacer clic en "EDITAR", completan un formulario prellenado y al dar "GUARDAR", el sistema actualiza la información en la base de datos, confirmando la acción. Modificar ID y nombre de usuario está bloqueado por razones de integridad del sistema.	

Tabla 11. CU-05 – Modificación de Datos en el Perfil

3.2.3. Casos de Uso Pacientes

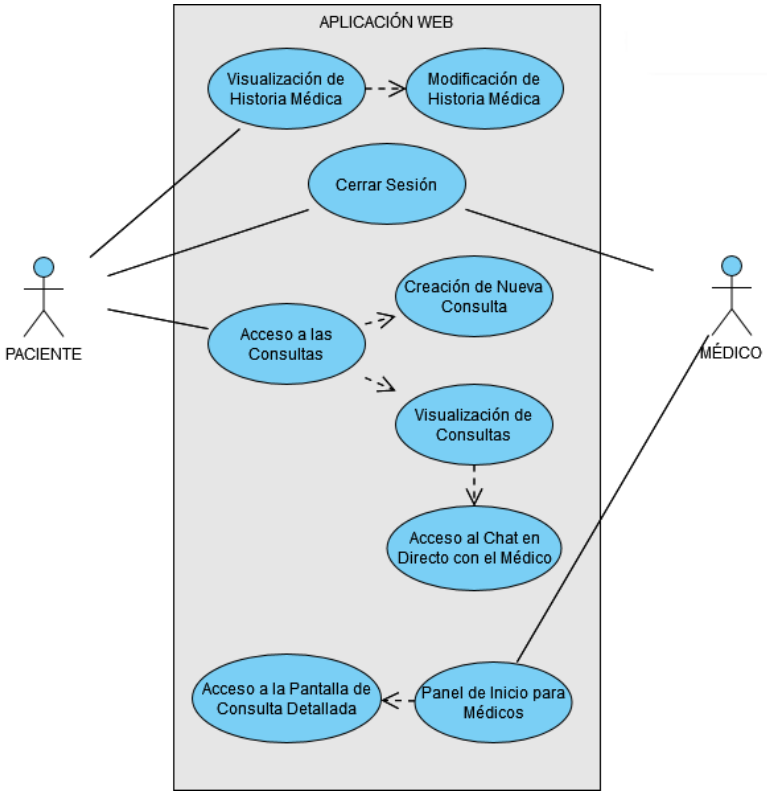


Figura 6. Caso de Uso de Pacientes y Médicos

Caso de Uso	Visualización de Historia Médica	Identificador: **CU-06**
Actores	Paciente (Usuario Autenticado)	
Tipo	Primario	
Referencias	Requiere acceso a la funcionalidad de historia médica en el sistema.	
Precondición	El paciente debe haber iniciado sesión.	
Postcondición	El paciente visualiza su historia médica.	
Descripción	1. El paciente hace clic en el botón “Historia” en el menú de navegación. 2. La página de historia médica se carga con campos predeterminados. 3. El paciente visualiza los campos correspondientes según las selecciones realizadas. 4. Puede observar y revisar la información almacenada en cada campo. 5. El paciente tiene la opción de modificar los datos.	
Resumen	Los pacientes acceden a su historial médico desde el dashboard al hacer clic en "Historia", donde pueden ver información detallada de su historial médico.	

Tabla 12. CU-06 – Visualización de Historia Médica

Caso de Uso	Modificación de Historia Médica	Identificador: **CU-07**
Actores	Paciente (Usuario Autenticado)	
Tipo	Primario	
Referencias	Requiere acceso a la funcionalidad de modificar la historia médica en el sistema.	
Precondición	El paciente debe haber iniciado sesión. Se activa al pulsar el botón "EDITAR" desde la sección de historia médica.	
Postcondición	Los cambios realizados en la historia médica se actualizan en la base de datos del sistema.	
Descripción	1. El paciente accede a la sección de historia médica desde el menú de navegación. 2. Antes de pulsar el botón "EDITAR", los campos de la historia médica están bloqueados y no se pueden editar. 3. Una vez que el paciente pulsa el botón "EDITAR": a. Los campos se desbloquean y se vuelven editables. b. En la primera fila, el paciente puede activar o desactivar la visualización de campos específicos marcando o desmarcando los checkbox asociados a "Alergias", "Medicamentos", "Alcohol", "Tabaquismo" y "Drogas". c. La página se actualiza para mostrar u ocultar campos según las selecciones realizadas. d. El paciente modifica la información en los campos de la historia médica: i. Campos obligatorios: "Nombre y apellidos", "Edad", "Sexo", "Raza", "Altura", "Peso" y “Comorbilidades”. ii. Campos opcionales: "Indica las alergias", "Consumo de cigarros/día", "Frecuencia de ingesta de alcohol", "Droga, cantidad y frecuencia" y "Uso de medicamentos y frecuencia". e. Después de realizar los cambios, el paciente hace clic en el botón "SALVAR". 4. El sistema verifica la validez de los datos modificados, asegurándose de que los campos obligatorios estén completos. 5. Si la verificación es exitosa, el sistema actualiza la información en la base de datos. 6. El paciente recibe una confirmación de que los cambios se realizaron correctamente y puede regresar a la página de Historia médica.	
Resumen	En la sección de historia médica, los pacientes pueden editar campos bloqueados al pulsar "EDITAR". Se pueden personalizar campos y modificar información. Se requiere completar campos obligatorios. Tras hacer clic en "SALVAR", el sistema verifica y actualiza la información si es válida, proporcionando una confirmación al paciente.	

Tabla 13. CU07 – Modificación de Historia Médica

Caso de Uso	Acceso a las Consultas	Identificador: **CU-08**
Actores	Paciente (Usuario Autenticado)	
Tipo	Primario	
Referencias	Requiere acceso a la funcionalidad de consulta en el sistema.	
Precondición	El paciente debe haber iniciado sesión.	
Postcondición	El paciente visualiza la pantalla "Consulta" con un listado de consultas médicas si las hay, y la opción de crear una nueva consulta.	
Descripción	1. El paciente selecciona la opción "Consulta" desde el menú de navegación. 2. La pantalla "Consulta" se carga, mostrando un listado de consultas médicas si existen. 3. Si hay consultas, se presentan con detalles como status, identificador, médico, fecha, motivo y un botón para ver más. 4. Además, se muestra el botón "Nueva Consulta" que redirecciona al paciente a un formulario para crear una nueva consulta médica. 5. El paciente puede navegar entre las consultas existentes o crear una nueva según sus necesidades.	
Resumen	Desde "Consulta", los pacientes pueden ver sus consultas con detalles como estado, identificador, médico, fecha y motivo. Pueden crear nuevas consultas con "Nueva Consulta". Esto facilita la gestión eficiente de interacciones médicas.	

Tabla 14. CU-08 – Acceso a las Consultas

Caso de Uso	Visualización de Consultas	Identificador: **CU-09**
Actores	Paciente (Usuario Autenticado)	
Tipo	Primario	
Referencias	Requiere acceso a la funcionalidad de consultas desde la pantalla "Consulta".	
Precondición	El paciente debe haber iniciado sesión. Deben existir consultas médicas asociadas al paciente.	
Postcondición	El paciente visualiza las consultas médicas en la pantalla "Consulta".	
Descripción	1. El paciente accede a la pantalla "Consulta" desde el menú de navegación. 2. Si existen consultas médicas asociadas al paciente, se muestran en la pantalla. 3. Cada consulta se presenta con los siguientes detalles: a. Status (espera: amarillo, abierta: verde, cerrada: rojo) b. Identificador de la consulta c. ID del médico d. Fecha de la consulta e. Motivo de la consulta f. Botón "Ver Más" para redireccionar al paciente a la consulta detallada con el médico.	
Resumen	En "Consulta", los pacientes ven un listado detallado de sus consultas médicas con información como estado, identificador, médico, fecha y motivo. El botón "Ver Más" facilita obtener detalles adicionales,	

	simplificando el seguimiento y gestión de las interacciones médicas de los pacientes.

Tabla 15. CU-09 – Visualización de Consultas

Caso de Uso	Acceso al Chat en Directo con el Médico	Identificador: **CU-10**
Actores	Paciente (Usuario Autenticado)	
Tipo	Primario	
Referencias	Se activa al pulsar el botón "Ver Más" desde la pantalla "Consulta".	
Precondición	El paciente debe haber iniciado sesión. Debe existir al menos una consulta médica asociada al paciente.	
Postcondición	El paciente accede al chat en directo con el médico desde la consulta seleccionada.	
Descripción	1. El paciente selecciona una consulta específica desde la pantalla "Consulta". 2. Al pulsar el botón "Ver Más", el sistema redirecciona al paciente a una página dedicada al chat en directo con el médico asociado a esa consulta. 3. Se establece una interfaz de chat en tiempo real donde el paciente puede comunicarse por escrito con el médico. 4. El paciente puede hacer preguntas, proporcionar información adicional o discutir aspectos relacionados con la consulta. 5. La comunicación en el chat se realiza de manera síncrona, permitiendo una interacción inmediata entre el paciente y el médico. 6. Al finalizar la conversación, el paciente tiene la opción de cerrar la ventana del chat y regresar a la pantalla "Consulta" o a otras secciones según sea necesario.	
Resumen	Al pulsar "Ver Más" en la pantalla "Consulta", los pacientes acceden a un chat en directo con su médico, facilitando una comunicación inmediata y eficiente sobre detalles adicionales o preguntas relacionadas con la consulta médica.	

Tabla 16. CU-10 – Acceso al Chat en Directo con el Médico

Caso de Uso	Creación de Nueva Consulta	Identificador: **CU-11**
Actores	Paciente (Usuario Autenticado)	
Tipo	Primario	
Referencias	Se activa al pulsar el botón "Nueva Consulta" desde la pantalla "Consulta".	
Precondición	El paciente debe haber iniciado sesión. Debe existir una historia médica asociada al paciente.	
Postcondición	Se crea una nueva consulta médica en el sistema si se cumplen las condiciones.	
Descripción	1. El paciente selecciona la opción "Nueva Consulta" desde la pantalla "Consulta". 2. Si no dispone de ninguna historia médica, se muestra el mensaje: "Para poder realizar consultas necesita rellenar su historia médica en la sección de Historia".	

	3. Si tiene una historia médica, se presenta un formulario con los siguientes campos: a. ID personal: Campo bloqueado que muestra el ID del paciente. b. ID Médico: Campo de entrada donde el paciente debe ingresar el ID del médico. c. Motivo de la consulta: Campo de entrada para que el paciente ingrese el motivo de la consulta. 4. Después de introducir los datos, el sistema verifica: a. Si el ID del médico ingresado corresponde a algún médico registrado en el sistema. En caso contrario, se notifica un error. b. Si el motivo de la consulta contiene mínimo 10 palabras. En caso contrario, se notifica un error. 5. Si todas las verificaciones son exitosas, se crea una nueva consulta médica en el sistema asociada al paciente y al médico correspondientes. 6. El paciente recibe una confirmación de que la nueva consulta se ha creado con éxito.
Resumen	Desde "Consulta", al pulsar "Nueva Consulta", los pacientes pueden crear una nueva consulta médica con campos como ID personal, ID del médico y motivo. Se realizan verificaciones para validar los datos y se notifican errores si es necesario.

Tabla 17. CU-11 – Creación de Nueva Consulta

3.2.4. Casos de Uso Médicos

Caso de Uso	Panel de Inicio para Médicos	Identificador: **CU-12**
Actores	Médico (Usuario Autenticado)	
Tipo	Primario	
Referencias	Se activa al iniciar sesión como médico.	
Precondición	El médico debe haber iniciado sesión en el sistema.	
Postcondición	El médico accede al panel de inicio con información y opciones relevantes.	
Descripción	1. El médico inicia sesión en el sistema utilizando sus credenciales. 2. Después de una autenticación exitosa, el sistema redirige al médico al panel de inicio. 3. En el panel de inicio, se presenta una tabla con las siguientes columnas: a. Status: Indica el estado de la consulta (verde para abierta, amarillo para espera, rojo para cerrada). b. Identificador: Muestra el ID de la consulta. c. Paciente: Muestra el nombre del paciente asociado a la consulta. d. Fecha: Indica la fecha programada para la consulta. e. Motivo: Describe el motivo de la consulta. 4. Las consultas se presentan en filas, mostrando múltiples consultas en la tabla. 5. El médico puede hacer clic en cualquier fila de la tabla para acceder a detalles adicionales de la consulta o realizar acciones	

	específicas, como abrir la consulta para interactuar con el paciente.
Resumen	En el panel de inicio, los médicos ven una tabla con detalles de consultas pendientes, incluyendo estado, ID de consulta, nombre del paciente, fecha y motivo. Esto facilita una rápida gestión de actividades clínicas.

Tabla 18. CU-12 – Panel de Inicio para Médicos

Caso de Uso	Acceso a la Pantalla de Consulta Detallada	Identificador: **CU-13**
Actores	Médico (Usuario Autenticado)	
Tipo	Primario	
Referencias	Se activa al hacer clic en una consulta específica desde el panel de inicio.	
Precondición	El médico debe haber iniciado sesión en el sistema. Debe existir al menos una consulta programada para el médico.	
Postcondición	El médico accede a la pantalla de consulta detallada.	
Descripción	1. El médico accede al panel de inicio después de iniciar sesión. 2. En el panel de inicio, el médico visualiza la tabla con las consultas. 3. El médico hace clic en una consulta específica en la tabla. 4. El sistema redirige al médico a la pantalla de consulta detallada. 5. En la parte izquierda de la pantalla, se presenta la historia médica del paciente asociado a la consulta seleccionada. 6. Si la consulta es nueva (status espera), el médico deberá cambiarla a estado abierta. 7. En la parte derecha de la pantalla, se muestra un chat en directo, permitiendo la comunicación síncrona con el paciente. 8. Durante la consulta, el médico puede visualizar la información de la historia médica mientras se comunica en tiempo real con el paciente. 9. Al finalizar la consulta, el médico tiene la opción de cambiar el status de la consulta a cerrada. 10. Se proporciona la opción de regresar al panel de inicio o a otras secciones del sistema al finalizar la consulta.	
Resumen	Al hacer clic en una consulta desde el panel de inicio, los médicos acceden a una pantalla dividida: 1. Izquierda: Muestra la historia médica detallada del paciente. 2. Derecha: Proporciona un chat en directo para la comunicación síncrona durante la consulta.	

Tabla 19. CU-13 – Acceso a la Pantalla de Consulta Detallada

3.3. Requerimientos Funcionales

Los requerimientos funcionales se refieren a las funciones específicas que deseamos que la aplicación realice. Aquí, debemos definir cómo queremos que el portal web reaccione ante las acciones de los usuarios y cómo queremos que interactúe con los diferentes sistemas. Algunos de los requerimientos funcionales de nuestra aplicación web médica son:

Requerimiento Funcional	Descripción	ID Caso De Uso Asociado
RF-01. Registro de Usuario	Los usuarios deben poder registrarse proporcionando información como nombre. correo electrónico y contraseña.	**CU-01** – Registro de Usuario
RF-02: Inicio de Sesión	Los usuarios, tanto médicos como pacientes, deben poder iniciar sesión con credenciales válidas y cerrar sesión cuando sea necesario.	**CU-02** – Iniciar Sesión con Credenciales **CU-03** – Cerrar Sesión
RF-03: Gestión de Perfil de Usuario	Los usuarios deben poder acceder a sus perfiles, visualizar información y tener la capacidad de modificar ciertos datos, como el correo electrónico.	**CU-04** – Acceso al Perfil **CU-05** – Modificación de Datos en el Perfil
RF-04: Creación de Consultas Médicas	Los pacientes deben poder crear nuevas consultas médicas.	**CU-11** – Creación de Nueva Consulta
RF-05: Visualización de Consultas Pendientes	Los médicos deben poder ver de manera clara y organizada las consultas pendientes en el panel de inicio.	**CU-12** - Panel de Inicio para Médicos
RF-06: Chat en Tiempo Real	Los médicos y pacientes deben poder comunicarse en tiempo real a través de un chat durante las consultas.	**CU-10** – Acceso al Chat en Directo con el Médico **CU-13** – Acceso a la Pantalla de Consulta Detallada

Tabla 20. Requerimientos Funcionales

3.4. Requerimientos No Funcionales

En cuanto a los requerimientos no funcionales se refieren a las propiedades o características del sistema, en lugar de las funciones específicas. Aquí, debemos tener en cuenta aspectos como el rendimiento, la seguridad y la disponibilidad. Algunos de los requerimientos funcionales de nuestra aplicación web médica son:

- **Seguridad de Datos:** La aplicación debe implementar medidas de seguridad para proteger la información confidencial del usuario, como contraseñas y datos médicos.
- **Usabilidad:** La interfaz de usuario debe ser intuitiva y fácil de usar para médicos y pacientes, favoreciendo una experiencia de usuario positiva.

- **Rendimiento:** La aplicación debe ser capaz de manejar un número considerable de usuarios concurrentes sin degradación significativa del rendimiento.
- **Disponibilidad:** La aplicación debe tener una alta disponibilidad, minimizando el tiempo de inactividad y asegurando que los usuarios puedan acceder a las funciones críticas en todo momento.
- **Escalabilidad:** La arquitectura de la aplicación debe ser escalable para manejar un crecimiento futuro en términos de usuarios y funciones.
- **Compatibilidad:** La aplicación debe ser compatible con diferentes navegadores web y dispositivos para garantizar una accesibilidad amplia.
- **Privacidad y Cumplimiento Normativo:** La aplicación debe cumplir con las regulaciones de privacidad y normativas médicas para garantizar la confidencialidad y seguridad de la información del usuario.
- **Mantenibilidad:** El código de la aplicación debe ser modular y fácil de mantener para permitir actualizaciones y correcciones de manera eficiente.

4. Arquitectura de la Aplicación

La aplicación desarrollada lleva por nombre "MedNet", en la cual confluyen las tecnologías anteriormente descritas para su pleno funcionamiento. De esta forma, en el frontend se ha empleado JavaScript y TypeScript en conjunto con React para crear las interfaces de usuario, mientras que HTML y CSS se utilizan para la estructura y el diseño de las páginas respectivamente.

Por otro lado, en el backend, se ha optado por Express, un framework de Node.js, para gestionar las operaciones del servidor. En cuanto a las bases de datos, se han empleado tanto bases de datos relacionales como MySQL para almacenar datos relacionados con registros y formularios, como información de usuarios y registros médicos, como bases de datos no relacionales como MongoDB para gestionar la información de las conversaciones en tiempo real, como los chats entre médicos y pacientes.

La aplicación ofrece una serie de funcionalidades clave, como la gestión de usuarios, la creación y visualización de registros médicos, la generación de consultas médicas y la gestión de salas de chat en tiempo real.

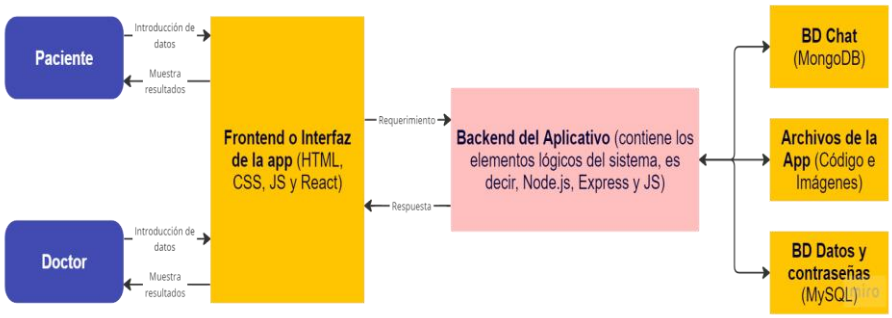


Figura 7. Arquitectura general de la aplicación

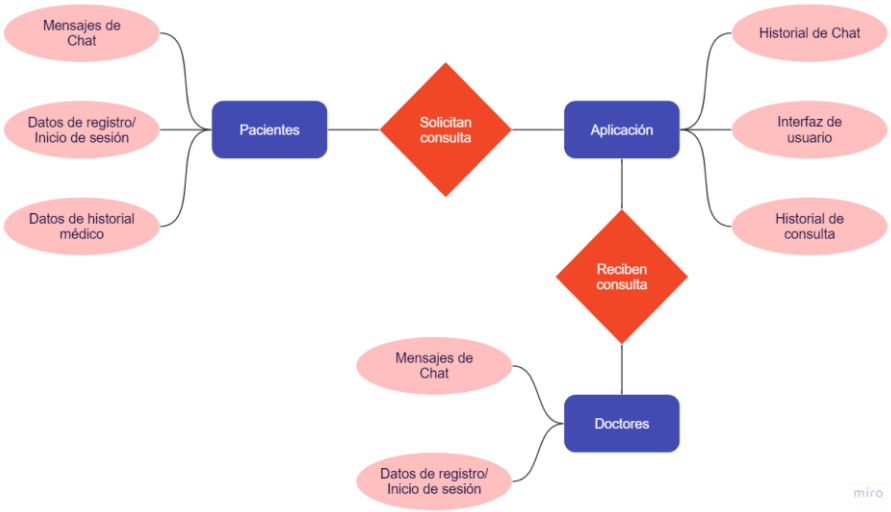


Figura 8. Modelo Entidad-Relación

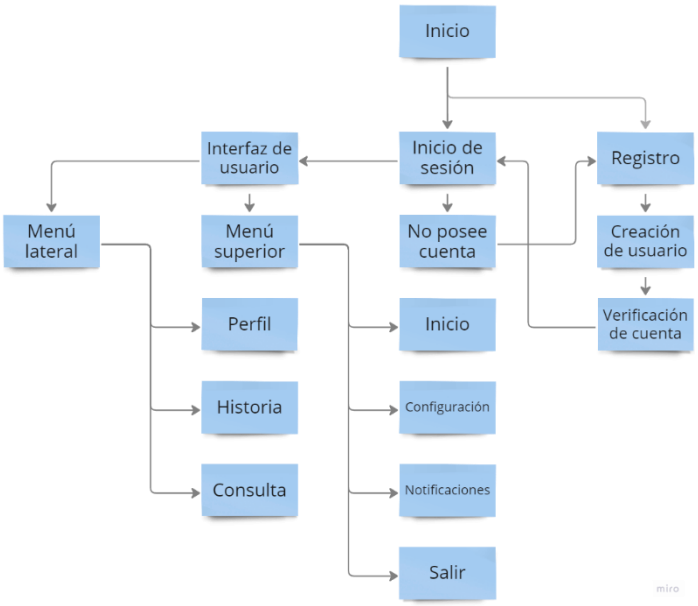


Figura 9. Diagrama de Funciones de la Aplicación

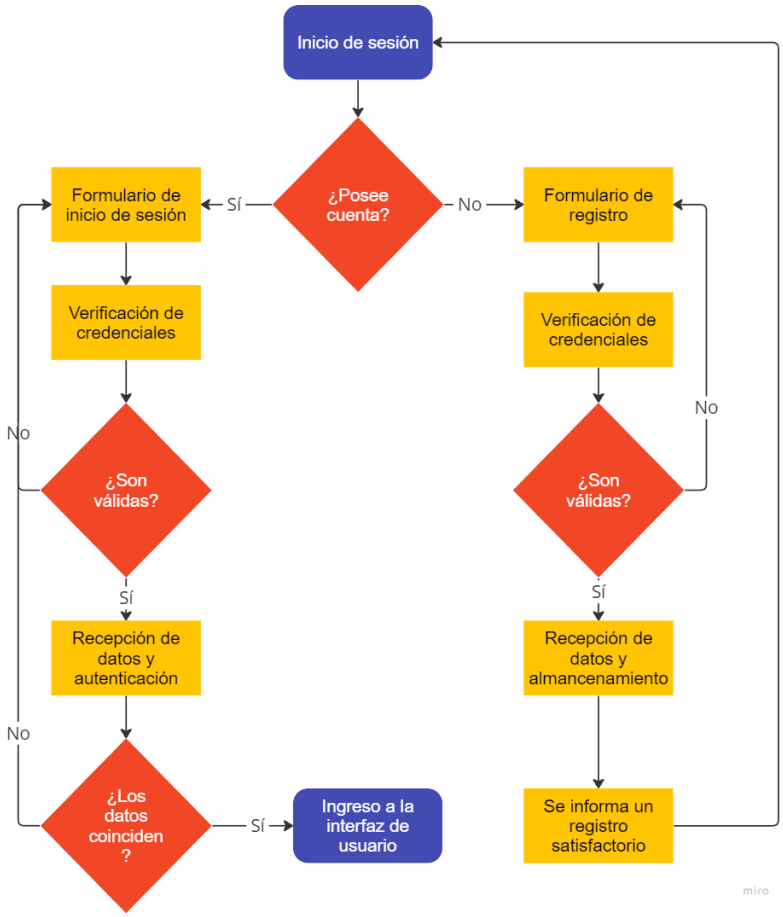


Figura 10. Diagrama de Funcionamiento de Inicio de Sesión y Registro

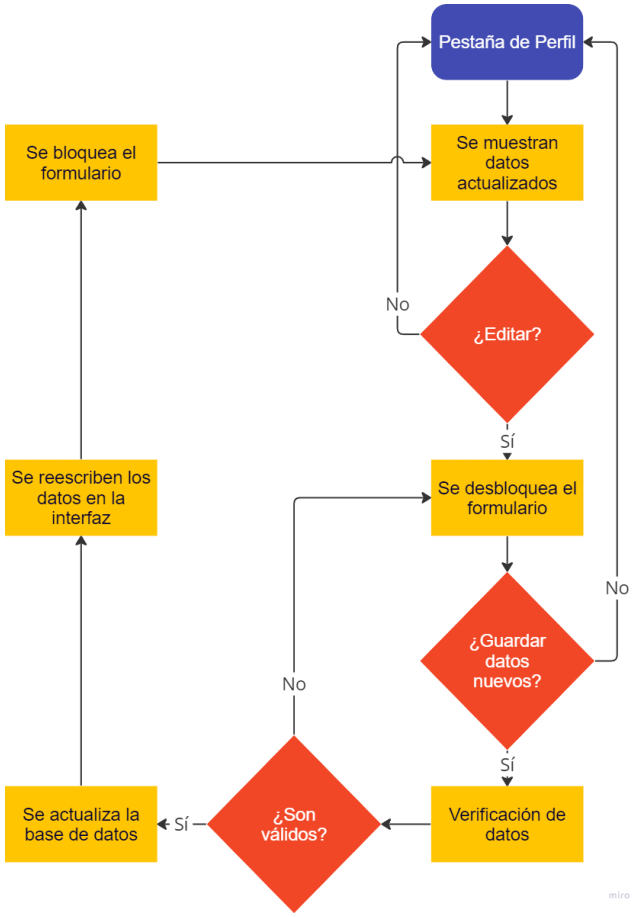


Figura 11. Diagrama de Funcionamiento de Edición de Datos de Usuario

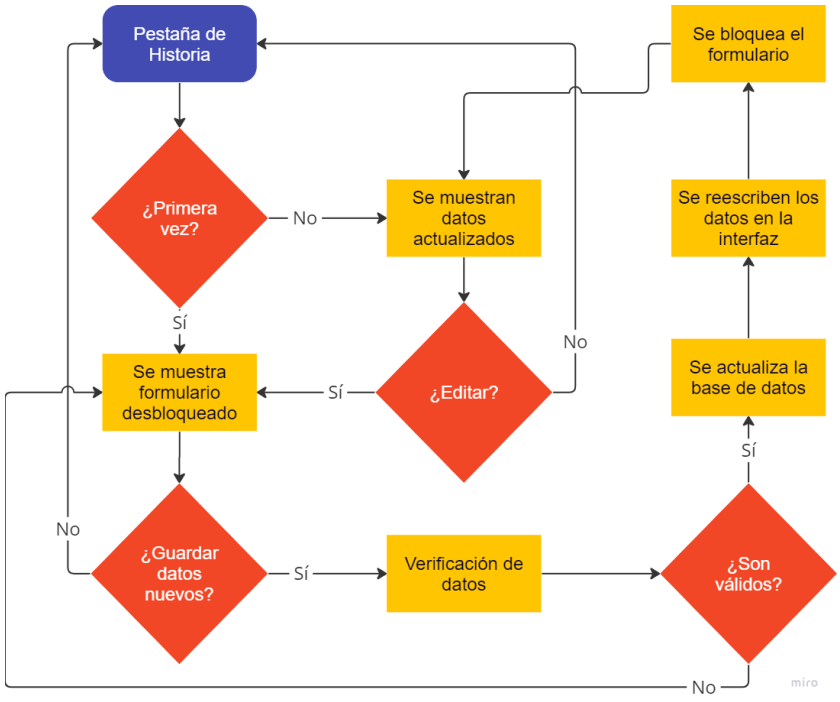


Figura 12. Diagrama de Funcionamiento de Añadir/Modificar el Historial Médico

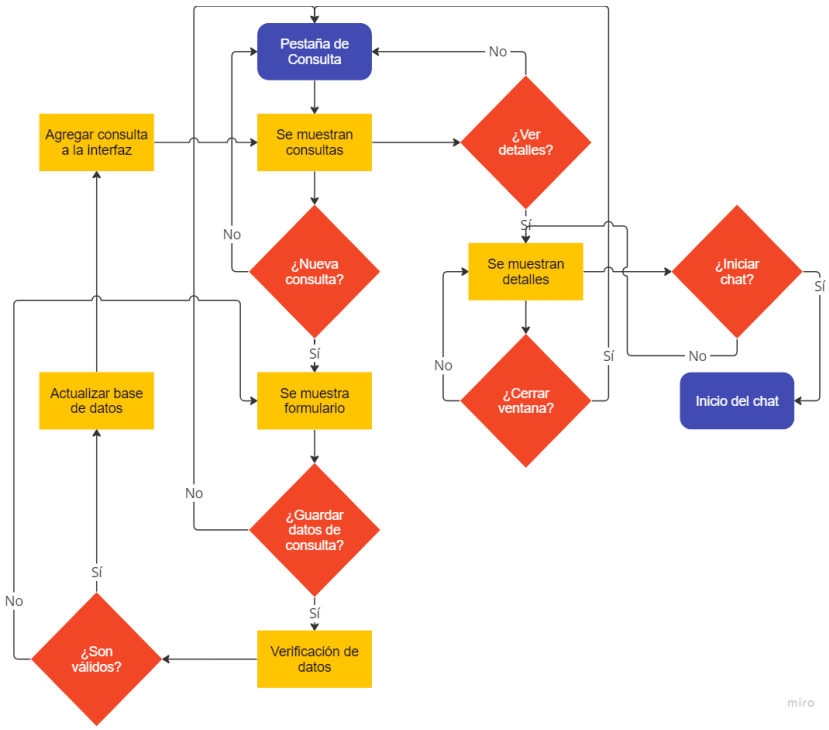


Figura 13. Diagrama de Funcionamiento del Chat

4.1. Modelo Base de Datos

A continuación, se muestra el modelo de datos diseñado inicialmente para desarrollar la aplicación web.

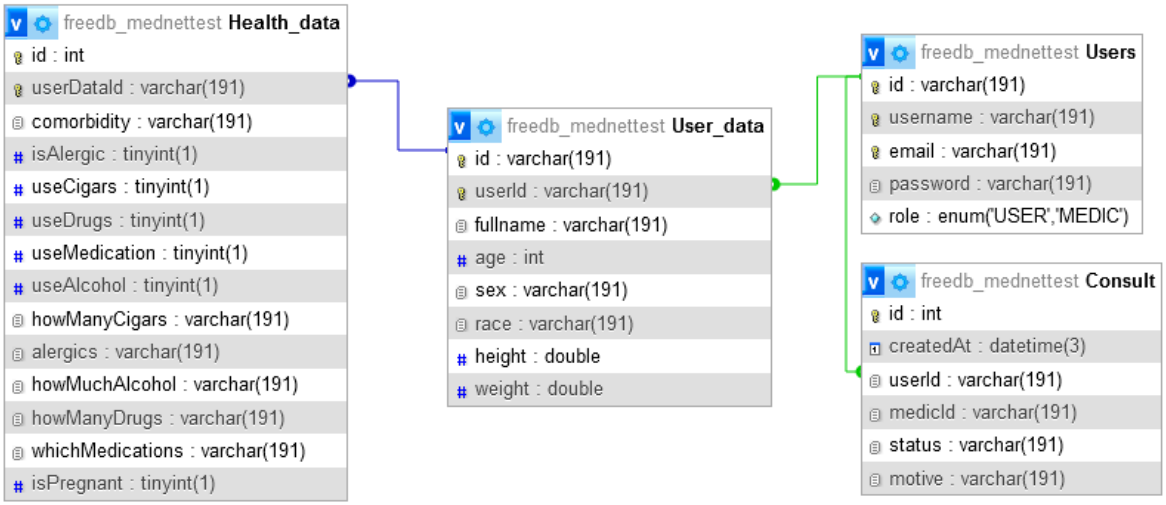


Figura 14. Modelo de base de datos de la aplicación

Columna	Tipo	Atributos	Nulo	Predeterminado	Extra	Enlaces a	Comentarios	MIME
id	int		No		auto_increment			
createdAt	datetime(3)		No	CURRENT_TIMESTAMP(3)	DEFAULT_GENERATED			
userId	varchar(191)		No			-> Users.id ON UPDATE CASCADE ON DELETE RESTRICT		
medicId	varchar(191)		No					
status	varchar(191)		No					
motive	varchar(191)		No					

Figura 15. Tabla Consult

Columna	Tipo	Atributos	Nulo	Predeterminado	Extra	Enlaces a	Comentarios	MIME
id	int		No		auto_increment			
userDataId	varchar(191)		No			-> User_data.id ON UPDATE CASCADE ON DELETE RESTRICT		
comorbidity	varchar(191)		No					
isAlergic	tinyint(1)		No					
useCigars	tinyint(1)		No					
useDrugs	tinyint(1)		No					
useMedication	tinyint(1)		No					
useAlcohol	tinyint(1)		No					
howManyCigars	varchar(191)		No					
alergics	varchar(191)		No					
howMuchAlcohol	varchar(191)		No					
howManyDrugs	varchar(191)		No					
whichMedications	varchar(191)		No					
isPregnant	tinyint(1)		No	0				

Figura 16. Tabla Health_data

Columna	Tipo	Atributos	Nulo	Predeterminado	Extra	Enlaces a	Comentarios	MIME
id	varchar(191)		No					
username	varchar(191)		No					
email	varchar(191)		No					
password	varchar(191)		No					
role	enum('USER', 'MEDIC')		No	USER				

Figura 17. Tabla Users

Columna	Tipo	Atributos	Nulo	Predeterminado	Extra	Enlaces a	Comentarios	MIME
id	varchar(191)		No					
userId	varchar(191)		No			-> Users.id ON UPDATE CASCADE ON DELETE RESTRICT		
fullname	varchar(191)		No					
age	int		No					
sex	varchar(191)		No					
race	varchar(191)		No					
height	double		No					
weight	double		No					

Figura 18. Tabla User_data

5. Plataforma de Desarrollo

Para el desarrollo del proyecto de nuestra aplicación web médica, se han empleado una gran variedad de recursos tecnológicos para garantizar el éxito y la eficiencia del proceso de creación de la aplicación web médica.

A continuación, se muestra el Software, Hardware y Web app utilizadas en el desarrollo de la aplicación.

SOFTWARE	
IDE Desarrollo	Visual Studio Code
Servidor local	Node.Js
Servidor web	Heroku
Maquetado de la documentación	Microsoft Word
Diseño Gráfico	Photoshop
Navegadores para realizar testeo	Chrome/ Firefox
HARDWARE	
Ordenadores	MSI Nightblade - Intel Core i5
Pantalla auxiliar	Benq Senseye 3 Led
WEB UP	
Control de versiones	GitHub
Respaldo de documentación	Google Drive

Tabla 21. Recursos tecnológicos para la creación de la aplicación web

6. Prototipos

La elaboración de prototipos lo-fi e hi-fi es una etapa crucial en el desarrollo de una aplicación web médica. A continuación, se describe brevemente cada tipo de prototipo:

6.1. Lo-Fi

Los prototipos Lo-Fi son representaciones básicas y de bajo nivel de fidelidad de la aplicación web. Estos prototipos se centran en la funcionalidad y estructura, sin preocuparse demasiado por el diseño visual o los detalles estéticos. Los prototipos Lo-Fi suelen ser esbozos o bocetos realizados a mano o utilizando herramientas de diseño sencillas.

La elaboración de prototipos Lo-Fi permite realizar pruebas rápidas y obtener comentarios tempranos de los usuarios y las partes interesadas. Estos prototipos son útiles para iterar rápidamente y realizar ajustes en el flujo de navegación, la arquitectura de la información y el diseño general. Al ser rápidos y económicos de producir, los prototipos Lo-Fi son ideales para probar conceptos y validar ideas antes de invertir tiempo y recursos en el desarrollo completo de la aplicación.

A continuación, se muestran algunos de los prototipos más importantes de la aplicación.

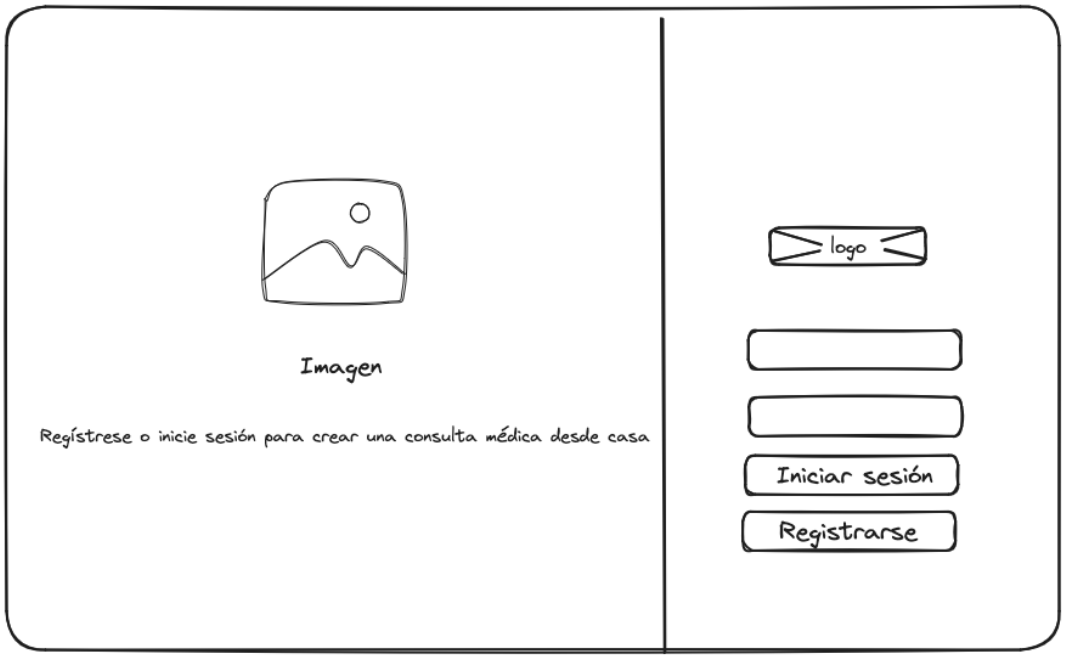


Figura 19. Prototipo Lo-Fi Escritorio – Home

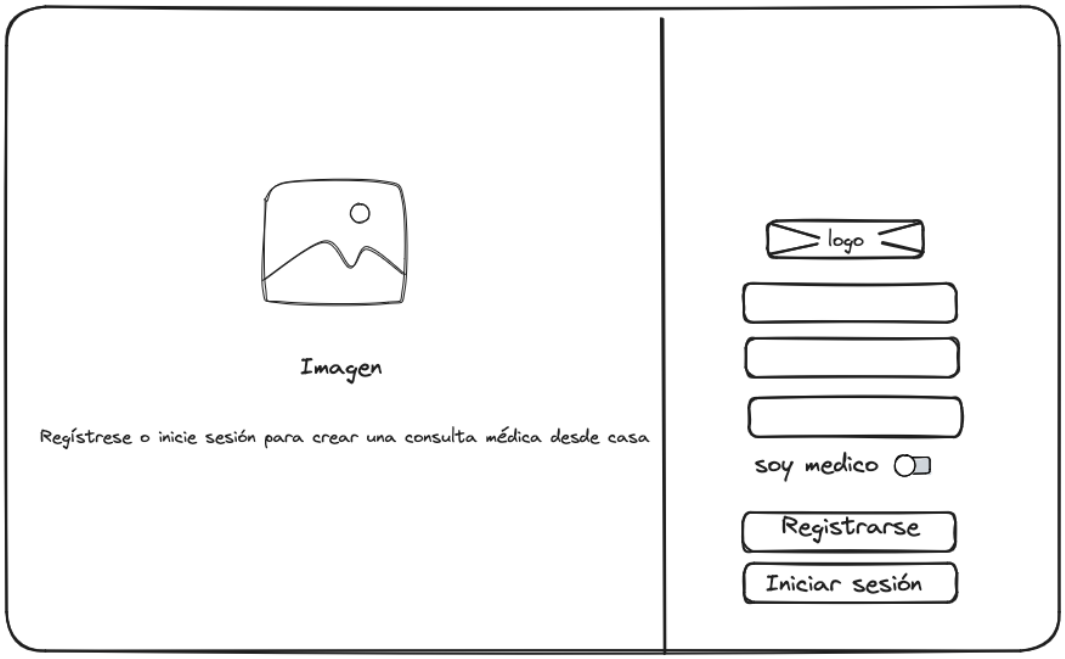


Figura 20. Prototipo Lo-Fi Escritorio – Registro

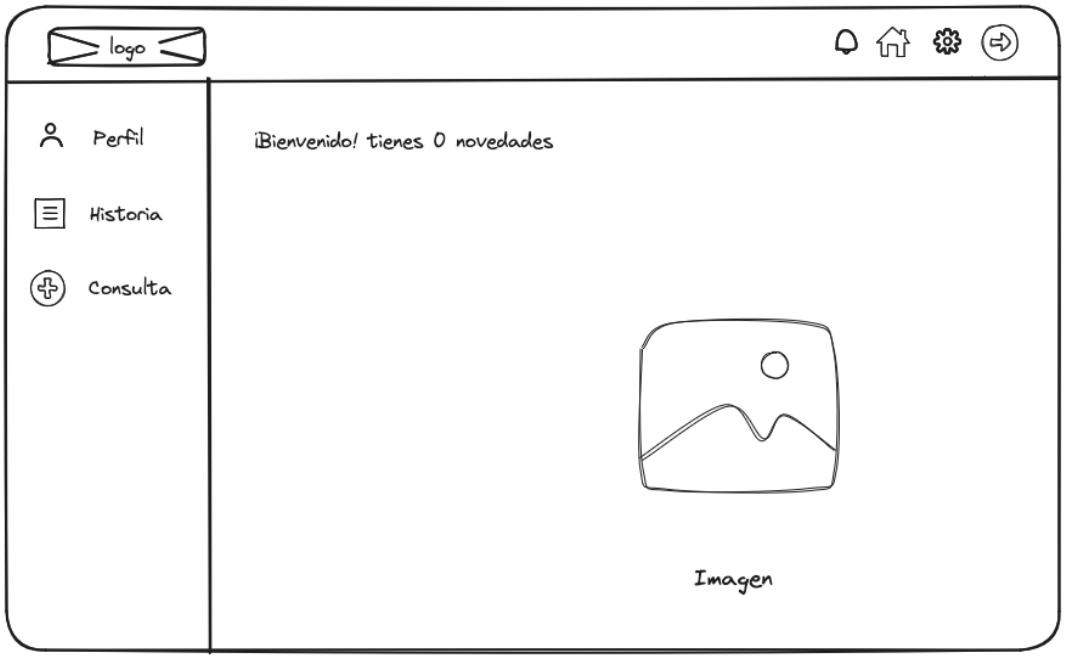


Figura 21. Prototipo Lo-Fi Escritorio – Pantalla Inicio

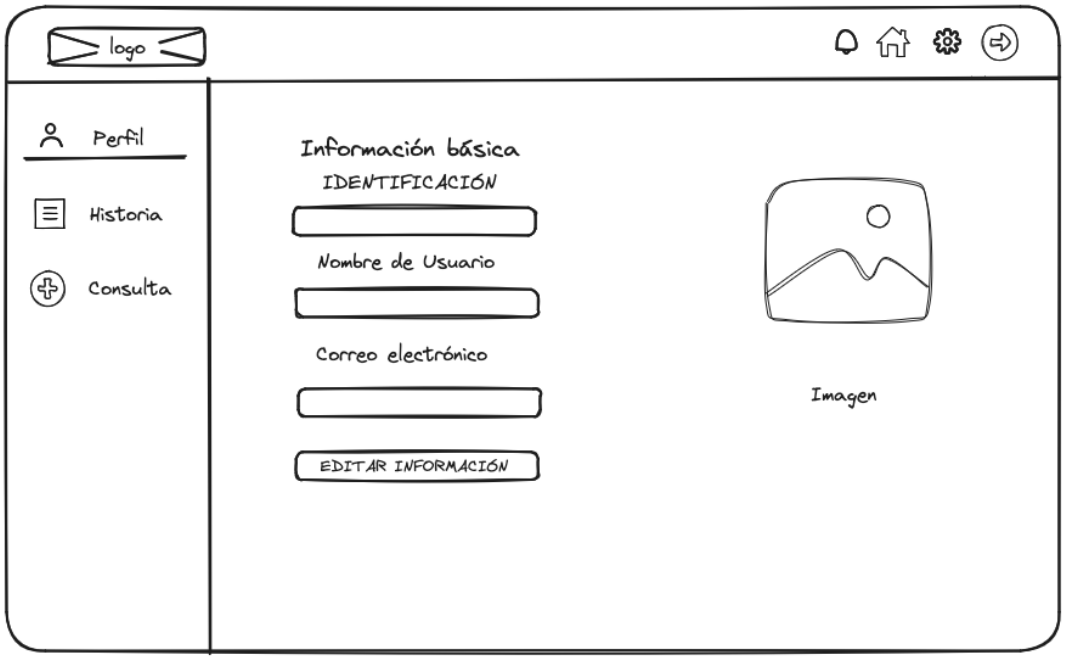


Figura 22. Prototipo Lo-Fi Escritorio – Pantalla Perfil

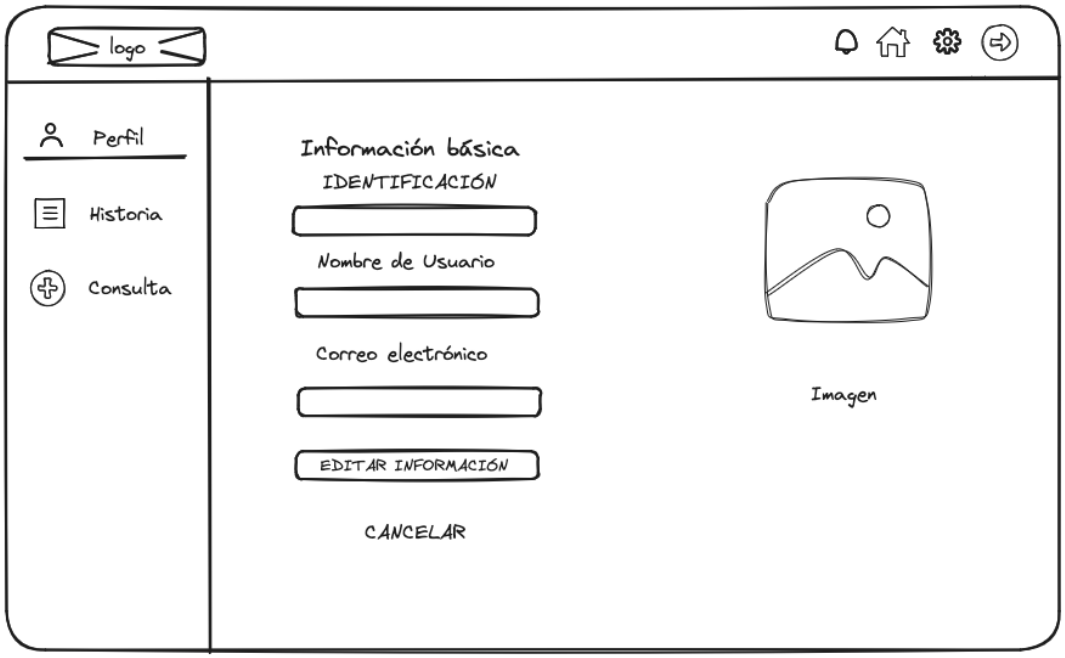


Figura 23. Prototipo Lo-Fi Escritorio – Pantalla Perfil Editar

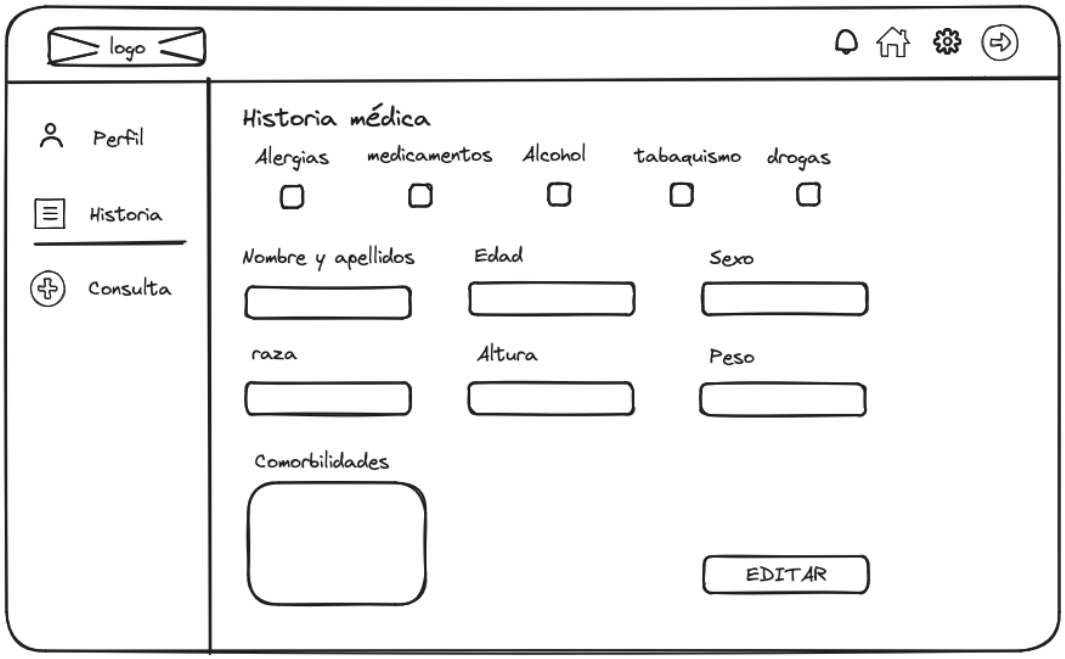


Figura 24. Prototipo Lo-Fi Escritorio – Pantalla Historia

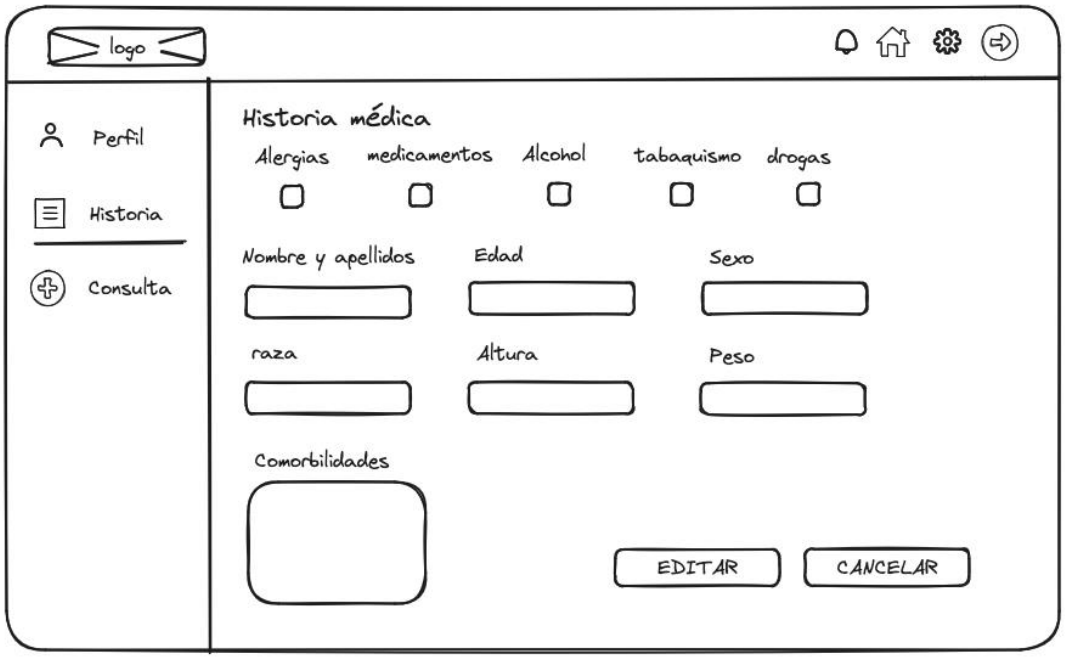


Figura 25. Prototipo Lo-Fi Escritorio – Pantalla Historia Editar

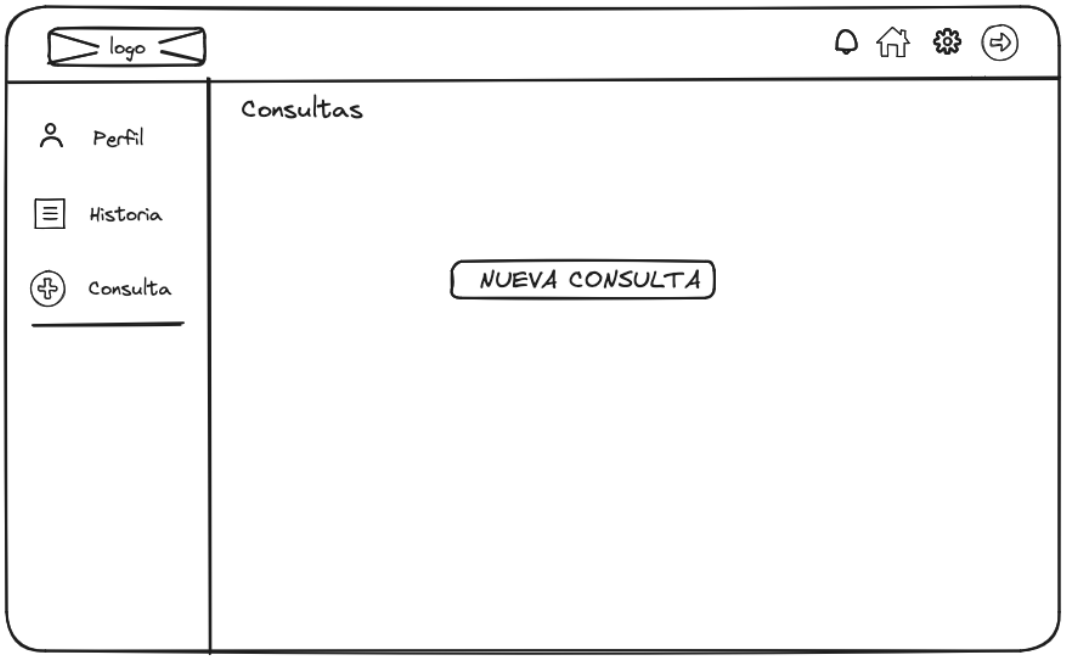


Figura 26. Prototipo Lo-Fi Escritorio – Pantalla Nueva Consulta

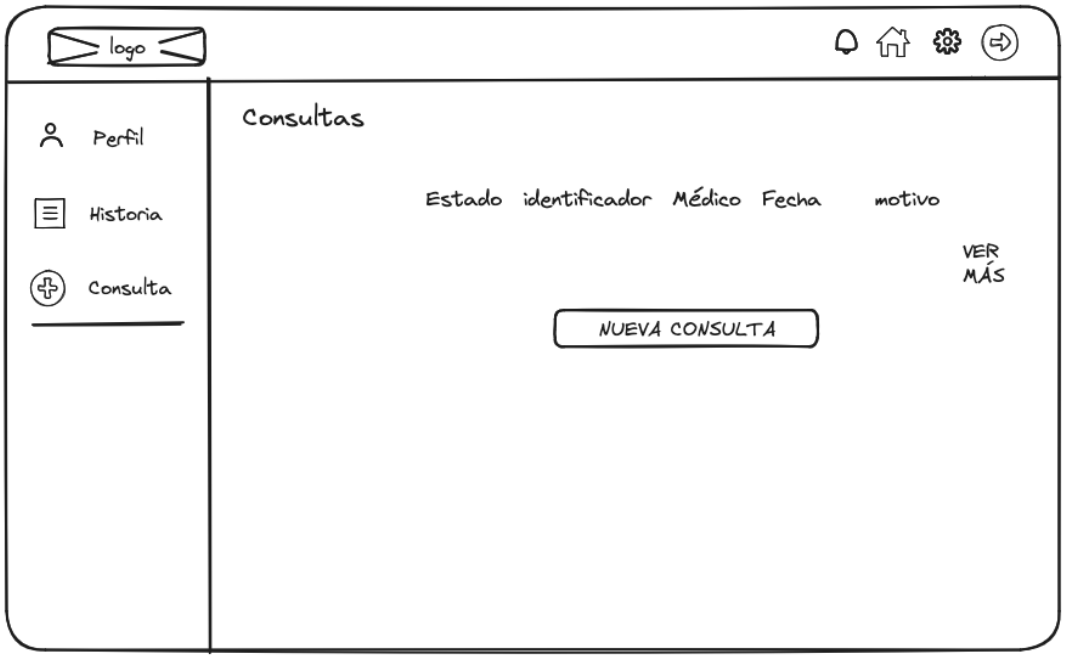


Figura 27. Prototipo Lo-Fi Escritorio – Pantalla Consulta

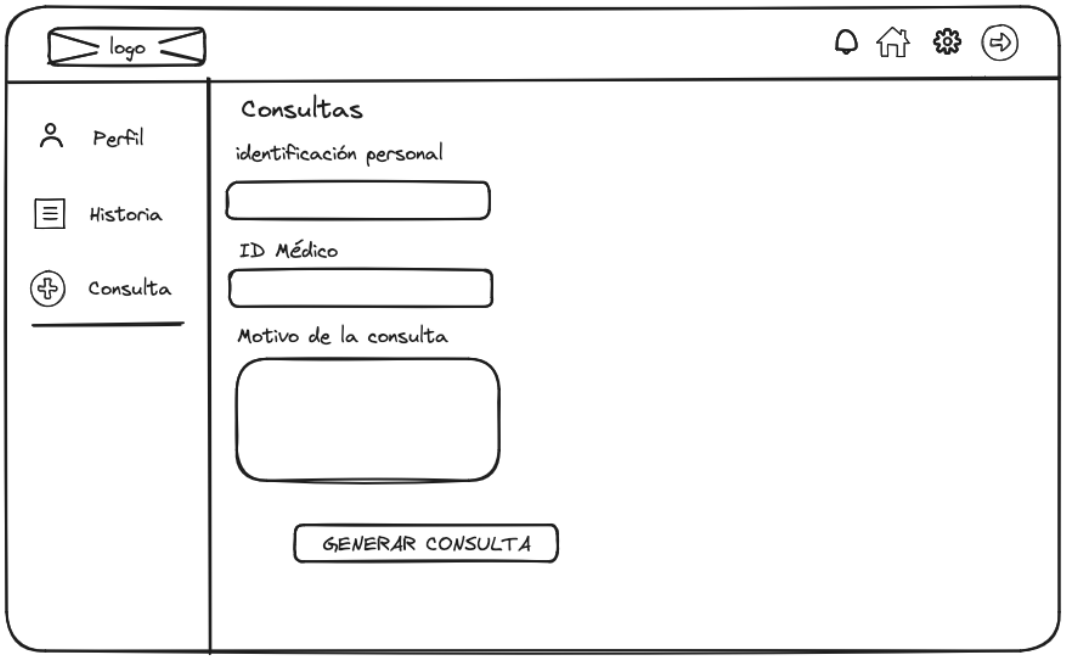


Figura 28. Prototipo Lo-Fi Escritorio – Pantalla Consulta Editar

6.2. Hi-Fi

Los prototipos Hi-Fi son representaciones más detalladas y de alta fidelidad de la aplicación web, que se asemejan visualmente a la versión final. Estos prototipos suelen ser creados utilizando herramientas de diseño gráfico y desarrollo web, lo que permite simular con mayor precisión la apariencia y funcionalidad de la aplicación real.

La elaboración de prototipos Hi-Fi permite realizar pruebas más avanzadas y detalladas con los usuarios y las partes interesadas. Estos prototipos son útiles para evaluar la experiencia de usuario, el diseño visual, la interacción y la usabilidad antes de pasar a la etapa de desarrollo completa. Los prototipos Hi-Fi también son útiles para presentar y comunicar el concepto de la aplicación a los equipos de desarrollo, diseñadores y otras partes interesadas, para asegurarse de que todos tengan una comprensión clara de las características y funcionalidades esperadas.

A continuación, se muestran algunos de los prototipos más importantes de la aplicación.

Figura 29. Prototipo Hi-Fi Escritorio – Home

Figura 30. Prototipo Hi-Fi Escritorio – Registrarse

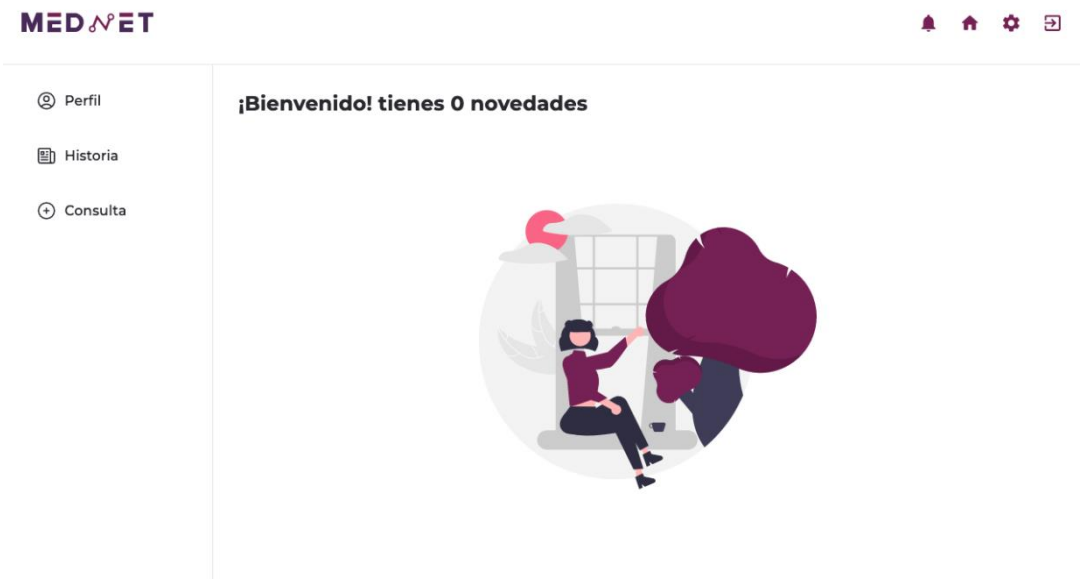


Figura 31. Prototipo Hi-Fi Escritorio – Pantalla Inicio

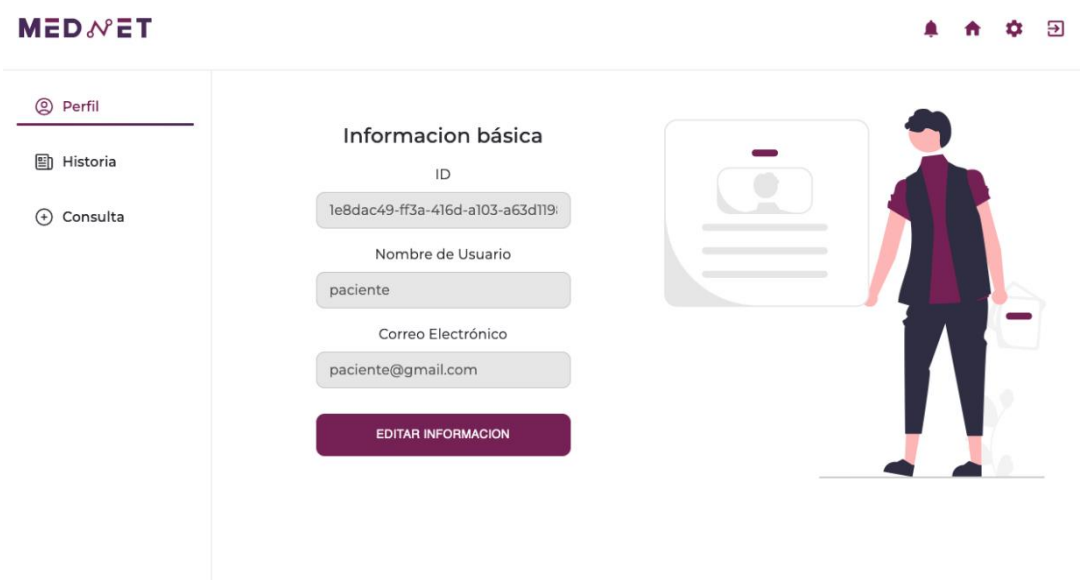


Figura 32. Prototipo Hi-Fi Escritorio – Pantalla Perfil

Figura 33. Prototipo Hi-Fi Escritorio – Pantalla Perfil Editar

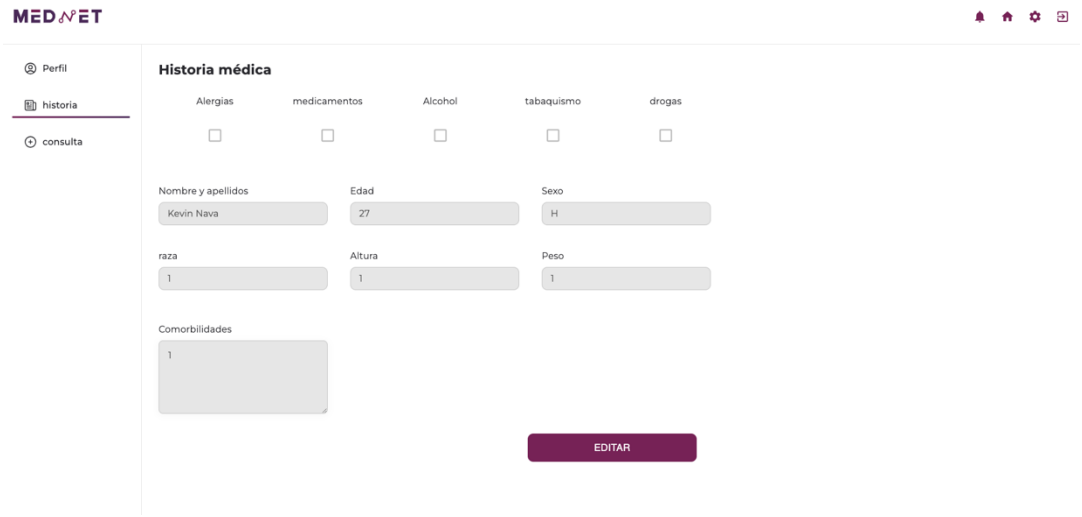


Figura 34. Prototipo Hi-Fi Escritorio – Pantalla Historia

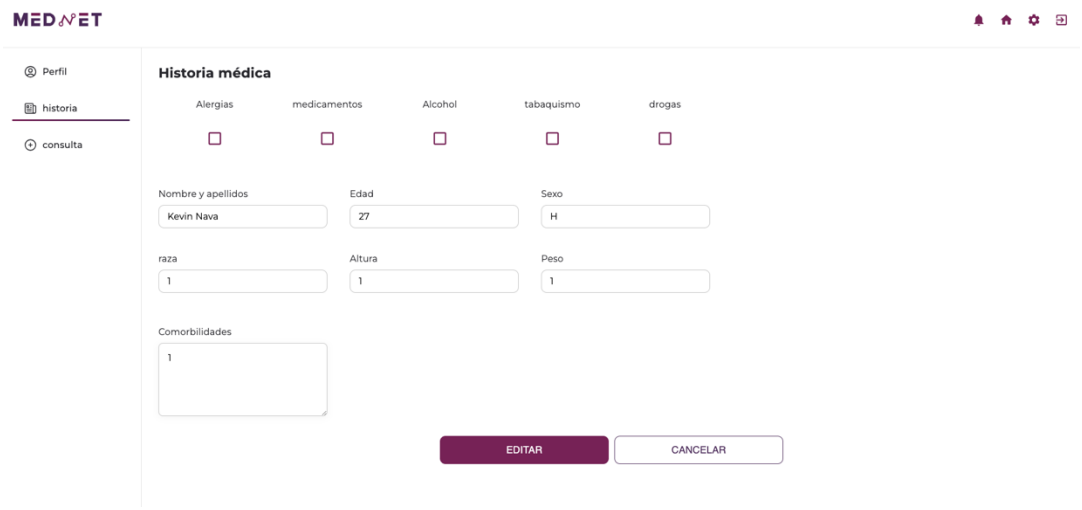


Figura 35. Prototipo Hi-Fi Escritorio – Pantalla Historia Editar

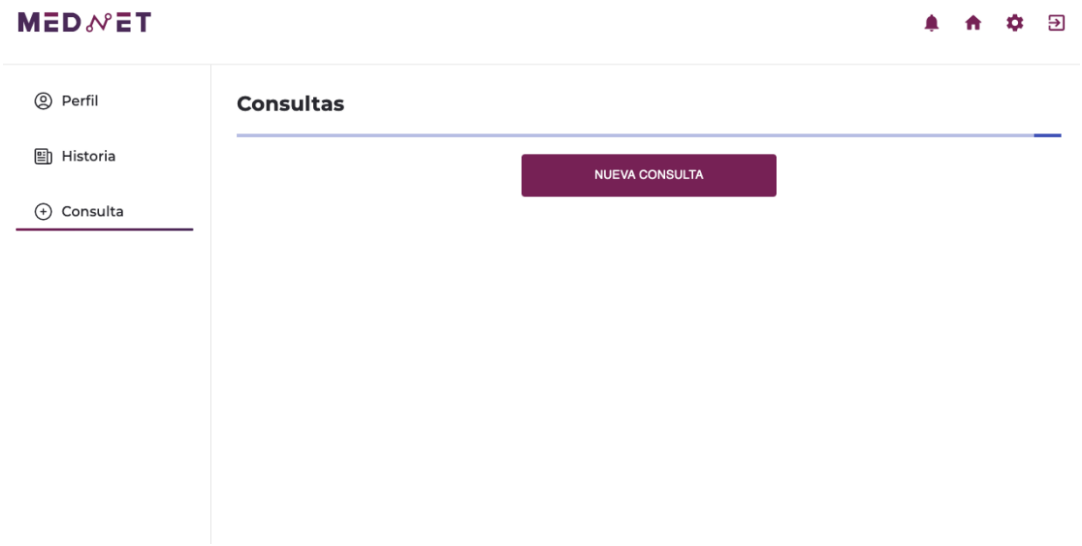


Figura 36. Prototipo Hi-Fi Escritorio – Pantalla Nueva Consulta

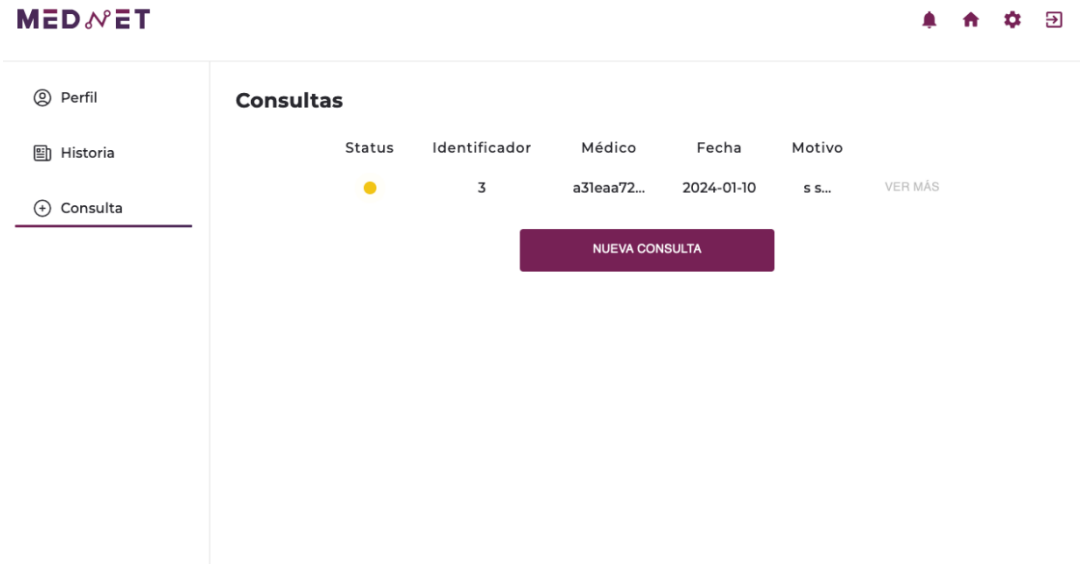


Figura 37. Prototipo Hi-Fi Escritorio – Pantalla Consulta

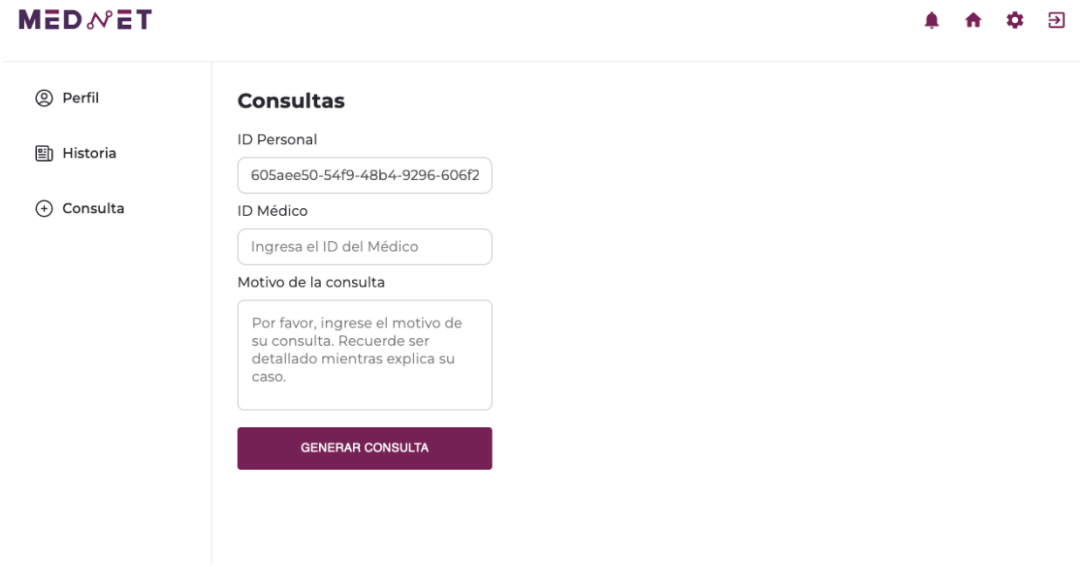


Figura 38. Prototipo Hi-Fi Escritorio – Pantalla Generar Consulta

7. Implantación

7.1. Instrucciones de Implantación

7.1.1. Back-End

Para instalar el Back-End de la aplicación se deberá seguir los siguientes pasos:

1. **Crear bases de datos:**
   Se deberá crear dos bases de datos, una en MySQL y otra en MongoDB para almacenar los datos de la aplicación.
2. **Crear archivo ".env":**
   En la raíz del proyecto, crea un archivo llamado ".env" con los siguientes parámetros:

```
DATABASE_URL="mysql://Usuario:MYSQLPassword@localhost:3306/NombreDeLaBD"
SECRET_KEY="ClavePrivadaJWT"
MONGO_DB="NombreDeLaBDMongoDB"
URI="mongodb://localhost:27017/"
```

Código 1. Parámetros de configuración para el Back-End

   - *"DATABASE_URL"*: Ruta de conexión a la base de datos MySQL.
   - *"SECRET_KEY"*: Clave privada para JWT.
   - *"MONGO_DB"*: Nombre de la base de datos de MongoDB.
   - *"URI"*: Ruta de conexión a la base de datos MongoDB.
3. **Instalar las dependencias:**
   En el Terminal, se debe ejecutar el comando "npm install" para instalar las dependencias del proyecto.
4. **Compilar el proyecto:**
   En el Terminal, se debe ejecutar el comando "npm run build" para así compilar el proyecto a una versión ejecutable.
5. **Ejecutar el programa:**
   Finalmente, para iniciar el servidor se deberá ejecutar el comando "node dist/server.js" en el Terminal.

7.1.2. Front-End

Para instalar el Front-End de la aplicación de manera exitosa se deberá seguir los siguientes pasos:

1. **Instalar las dependencias:**
   En el Terminal, se deberá ejecutar el comando "npm install" para instalar las dependencias del proyecto.
2. **Compilar el proyecto:**

Una vez instaladas las dependencias del proyecto se deberá ejecutar el comando “react-scripts build” para compilar el proyecto y generar una versión ejecutable. Este comando compilará y empaquetará el código del Front-End, generando archivos estáticos listos para ser desplegados en un servidor web.

3. **Ejecutar el programa:**
   Después de compilar el proyecto, se puede iniciar el servidor de desarrollo ejecutando el comando “react-scripts start”. Este comando iniciará el servidor de desarrollo y abrirá la aplicación en el navegador web, siendo accesible desde “**http://localhost:3000**” (o en otro puerto si el 3000 está en uso).

8. Estructura del Proyecto

8.1. Listado de Archivos

A continuación, se presenta el listado de archivos para el Back-End y Front-End de la aplicación, detallando la estructura organizativa de cada componente.

8.1.1. Back-End

```
├── prisma
│   ├── migrations
│   │   ├── 20211015230243_create_tables
│   │   │   └── migration.sql
│   │   └── migration_lock.toml
│   └── schema.prisma
├── src
│   ├── common
│   │   ├── controllers
│   │   │   ├── authenticateUserController.ts
│   │   │   ├── createUserController.ts
│   │   │   ├── generateConsultController.ts
│   │   │   ├── getChatHistoryController.ts
│   │   │   ├── getMedicConsultController.ts
│   │   │   ├── getPacientConsultsController.ts
│   │   │   ├── getPacientDataController.ts
│   │   │   ├── getUserInfoController.ts
│   │   │   ├── insertUseInfoController.ts
│   │   │   ├── updateUserController.ts
│   │   │   └── updateUserInfoController.ts
│   │   ├── setup.ts
│   │   ├── types.ts
│   │   └── worker.ts
│   ├── middlewares
│   │   ├── rateLimit.ts
│   │   └── validateToken.ts
```

```
│   ├── model
│   │   ├── chat.ts
│   │   ├── cloudinary.ts
│   │   └── connection.ts
│   ├── prisma
│   │   └── client.ts
│   ├── router
│   │   ├── consult.ts
│   │   └── user.ts
│   ├── services
│   │   ├── authenticateUser.ts
│   │   ├── createUser.ts
│   │   ├── generateConsult.ts
│   │   ├── getChatHistory.ts
│   │   ├── getMedicConsult.ts
│   │   ├── getPacientConsult.ts
│   │   ├── getPacientData.ts
│   │   ├── getUserInfo.ts
│   │   ├── insertUserInfo.ts
│   │   ├── updateConsultStatus.ts
│   │   ├── updateUser.ts
│   │   └── updateUserInfo.ts
│   ├── sockets
│   │   └── socket.ts
│   ├── utils
│   │   └── FindUser.ts
│   ├── app.ts
│   └── server.ts
├── .env
├── package.json
└── tsconfig.json
```

8.1.2. Front-End

```
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
├── src
│   ├── assets
│   │   ├── animations
│   │   │   ├── 22449-medic-calendar.json
│   │   │   ├── 40996-medical-try.json
│   │   │   ├── 53911-online-medical-assistance-animation.json
│   │   │   └── 75781-consultation.json
│   │   └── images
│   │       ├── answer.svg
│   │       ├── asd2.svg
```

```
│   │   ├── asd3.png
│   │   ├── check.svg
│   │   ├── copy.svg
│   │   ├── dashboard.svg
│   │   ├── delete.svg
│   │   ├── empty-questions.svg
│   │   ├── empty.svg
│   │   ├── fill.svg
│   │   ├── filldata.svg
│   │   ├── google-icon.svg
│   │   ├── illustration.svg
│   │   ├── kit.svg
│   │   ├── like.svg
│   │   ├── login.svg
│   │   ├── logo.png
│   │   ├── logo.svg
│   │   ├── logo2.svg
│   │   ├── MedNet.svg
│   │   ├── newlogo.png
│   │   ├── profileData.svg
│   │   ├── profilePage.svg
│   │   ├── register.png
│   │   ├── register.svg
│   │   ├── signup.png
│   │   ├── Untitled.png
│   │   └── welcome.svg
│   ├── components
│   │   ├── Aside
│   │   │   ├── Aside.jsx
│   │   │   └── aside.scss
│   │   ├── AsideOption
│   │   │   ├── Asideoption.jsx
│   │   │   └── asideoption.scss
│   │   ├── Button
│   │   │   ├── Button.jsx
│   │   │   └── button.scss
│   │   ├── Chat
│   │   │   ├── chat.jsx
│   │   │   └── chat.scss
│   │   ├── ConsultCards
│   │   │   ├── ConsultCards.jsx
│   │   │   └── consultCards.scss
│   │   ├── HamburguerButton
│   │   │   ├── HamburguerButton.jsx
│   │   │   └── hamburguerButton.scss
│   │   ├── Header
│   │   │   ├── Header.jsx
│   │   │   └── header.scss
│   │   ├── HistoryTable
│   │   │   ├── HistoryTable.jsx
│   │   │   └── historyTable.scss
│   │   ├── Modal
```

```
│   │   │   ├── Alert.jsx
│   │   │   └── alert.scss
│   ├── context
│   │   └── AuthContext.js
│   ├── hooks
│   │   ├── useLocalStorage.js
│   │   └── useLogout.js
│   ├── pages
│   │   ├── auth
│   │   │   ├── Auth.jsx
│   │   │   └── auth.scss
│   │   ├── consult
│   │   │   ├── Consult.jsx
│   │   │   └── consult.scss
│   │   ├── dashboard
│   │   │   ├── AdminDashboard.jsx
│   │   │   └── adminDashboard.scss
│   │   ├── history
│   │   │   ├── History.jsx
│   │   │   └── history.scss
│   │   ├── mainData
│   │   │   ├── MainData.jsx
│   │   │   └── mainData.scss
│   │   ├── medicConsult
│   │   │   ├── MedicConsult.jsx
│   │   │   └── medicConsult.scss
│   │   ├── MedicDashboard
│   │   │   ├── MedicDashboard.jsx
│   │   │   └── medicDashboard.scss
│   │   ├── pacientConsult
│   │   │   ├── pacientConsult.jsx
│   │   │   └── pacientConsult.scss
│   │   ├── profile
│   │   │   ├── Profile.jsx
│   │   │   └── profile.scss
│   │   ├── signup
│   │   │   ├── Signup.jsx
│   │   │   └── signup.scss
│   ├── redux
│   │   ├── slices
│   │   │   ├── appSlice.js
│   │   │   ├── authSlice.js
│   │   │   ├── consultSlice.js
│   │   │   └── updateDataSlice.js
│   │   ├── defaultStates.js
│   │   └── store.js
│   ├── services
│   │   ├── api.js
│   │   └── socket.js
│   ├── utils
│   │   └── index.js
│   ├── App.js
```

```
│   ├── App.module.scss
│   ├── App.test.js
│   ├── global.scss
│   ├── index.js
│   ├── logo.svg
│   ├── reportWebVitals.js
│   └── setupTests.js
├── .firebaserc
├── firebase.json
├── package-lock.json
├── package.json
└── yarn.lock
```

8.2. Funciones más Representativas

8.2.1. Back-End

1. **Controladores (controllers):**
   - authenticateUserController.ts: Función para autenticar usuarios.
   - createUserController.ts: Función para crear nuevos usuarios.
   - generateConsultController.ts: Función para generar consultas.
   - getChatHistoryController.ts: Función para obtener el historial de chat.
   - getMedicConsultController.ts: Función para obtener consultas médicas.
   - getPacientConsultsController.ts: Función para obtener consultas de pacientes.
   - getPacientDataController.ts: Función para obtener datos de pacientes.
   - getUserInfoController.ts: Función para obtener información de usuarios.
   - insertUseInfoController.ts: Función para insertar información de usuarios.
   - updateUserController.ts: Función para actualizar usuarios.
   - updateUserInfoController.ts: Función para actualizar información de usuarios.
2. **Middlewares:**
   - rateLimit.ts: Middleware para limitar la frecuencia de solicitudes.
   - validateToken.ts: Middleware para validar tokens de autenticación.
3. **Modelo (model):**
   - chat.ts: Definición del modelo de chat.
   - cloudinary.ts: Definición del modelo de Cloudinary.
   - connection.ts: Definición del modelo de conexión.

4. **Prisma:**
    - client.ts: Configuración del cliente Prisma para interactuar con la base de datos.
5. **Enrutadores (router):**
    - consult.ts: Enrutador para las consultas.
    - user.ts: Enrutador para los usuarios.
6. **Servicios (services):**
    - authenticateUser.ts: Servicio para autenticar usuarios.
    - createUser.ts: Servicio para crear usuarios.
    - generateConsult.ts: Servicio para generar consultas.
    - getChatHistory.ts: Servicio para obtener el historial de chat.
    - getMedicConsult.ts: Servicio para obtener consultas médicas.
    - getPacientConsult.ts: Servicio para obtener consultas de pacientes.
    - getPacientData.ts: Servicio para obtener datos de pacientes.
    - getUserInfo.ts: Servicio para obtener información de usuarios.
    - insertUserInfo.ts: Servicio para insertar información de usuarios.
    - updateConsultStatus.ts: Servicio para actualizar el estado de consultas.
    - updateUser.ts: Servicio para actualizar usuarios.
    - updateUserInfo.ts: Servicio para actualizar información de usuarios.
7. **Sockets:**
    - socket.ts: Configuración del socket para la comunicación en tiempo real.
8. **Utilidades (utils):**
    - FindUser.ts: Función para encontrar usuarios en la base de datos.

8.2.2. Front-End

1. **Componentes (components):**
    - Aside.jsx y aside.scss: Componente y estilos del menú lateral.
    - AsideOption.jsx y asideoption.scss: Componente y estilos de las opciones del menú lateral.
    - Button.jsx y button.scss: Componente y estilos de botones.
    - Chat.jsx y chat.scss: Componente y estilos del chat.
    - ConsultCards.jsx y consultCards.scss: Componente y estilos de las tarjetas de consulta.
    - HamburguerButton.jsx y hamburguerButton.scss: Componente y estilos del botón de menú hamburguesa.
    - Header.jsx y header.scss: Componente y estilos del encabezado.
    - HistoryTable.jsx y historyTable.scss: Componente y estilos de la tabla de historial.
    - Alert.jsx y alert.scss: Componente y estilos de las alertas modales.

2. **Contexto (context):**
   - AuthContext.js: Configuración del contexto de autenticación.
3. **Hooks:**
   - useLocalStorage.js: Hook para manejar el almacenamiento local.
   - useLogout.js: Hook para gestionar el cierre de sesión.
4. **Páginas (pages):**
   - Auth.jsx y auth.scss: Página de autenticación.
   - Consult.jsx y consult.scss: Página de consultas.
   - AdminDashboard.jsx y adminDashboard.scss: Página del panel de administrador.
   - History.jsx y history.scss: Página de historial.
   - MainData.jsx y mainData.scss: Página de datos principales.
   - MedicConsult.jsx y medicConsult.scss: Página de consultas médicas.
   - MedicDashboard.jsx y medicDashboard.scss: Página del panel médico.
   - PacientConsult.jsx y pacientConsult.scss: Página de consultas de pacientes.
   - Profile.jsx y profile.scss: Página de perfil de usuario.
   - Signup.jsx y signup.scss: Página de registro.
5. **Redux:**
   - appSlice.js, authSlice.js, consultSlice.js y updateDataSlice.js: Definición de acciones y reducers del estado de la aplicación.
6. **Servicios (services):**
   - api.js: Configuración del servicio API para comunicarse con el backend.
   - socket.js: Configuración del servicio de socket para la comunicación en tiempo real.
7. **Utilidades (utils):**
   - index.js: Funciones y utilidades diversas.

8.3. Explicación de Funciones en Back-End

8.3.1. AuthenticateUser.ts

1. Controller

```
import { Request, Response } from 'express';
import { AuthenticateUser } from '../services/authenticateUser';
class AuthenticateUserController {
  async handle(req: Request, res: Response) {
    const { body } = req;
```

```
    const authenticateUser = new AuthenticateUser(body);

    const user = await authenticateUser.execute();


    return res.status(200).json(user);
  }
}
export {AuthenticateUserController};
```

Código 2. BackEnd – AuthenticateUserController

Este código está escrito en TypeScript y utiliza el framework Express para construir un controlador (*AuthenticateUserController*) que maneja una solicitud de autenticación de usuario. Aquí hay una explicación línea por línea:

1) **import { Request, Response } from 'express';:** Importa las interfaces *Request* y *Response* de Express. Estas interfaces se utilizan para asignar los objetos *req* y *res* que se pasan a los manejadores de rutas.
2) **import { AuthenticateUser } from '../services/authenticateUser';:** Importa la clase *AuthenticateUser* desde el módulo *authenticateUser* en el directorio services.
3) **class AuthenticateUserController {:** Define una clase llamada *AuthenticateUserController*.
4) **async handle(req: Request, res: Response) {:** Declara un método asincrónico llamado *handle*, que toma un objeto de solicitud (*req* de tipo *Request*) y un objeto de respuesta (*res* de tipo *Response*). Este método manejará la lógica de autenticación del usuario.
5) **const { body } = req;:** Extrae la propiedad *body* del objeto de solicitud (*req*) utilizando desestructuración. Esta línea asume que la solicitud contiene un cuerpo JSON.
6) **const authenticateUser = new AuthenticateUser(body);:** Crea una instancia de la clase *AuthenticateUser*, pasando el cuerpo de la solicitud como argumento al constructor. La clase *AuthenticateUser* se encarga de la lógica de autenticación del usuario.
7) **const user = await authenticateUser.execute();:** Invoca el método *execute* en la instancia de *AuthenticateUser* de manera asíncrona. Este método realiza la autenticación y devuelve un objeto que representa al usuario autenticado.
8) **return res.status(200).json(user);:** Envía una respuesta JSON al cliente con el estado 200 (OK) y el objeto *user*. Esta respuesta es la respuesta exitosa de la autenticación.
9) **}:** Cierra el método handle.
10) **export { AuthenticateUserController };:** Exporta la clase *AuthenticateUserController* para que pueda ser utilizada en otros archivos.

2. Service

```
import { FindUser } from '../utils/FindUser';
import { IRequest } from './../common/types';
import { compare } from "bcryptjs";
import { sign } from 'jsonwebtoken';
import 'dotenv/config';

class AuthenticateUser {
  username: string;
  password: string;

  constructor(credentials : IRequest) {
    Object.assign(this, credentials);
  }

  async execute() {

    const findUser = new FindUser(this.username);

    const user = await findUser.byUsername();

    if(!user) throw new Error('Invalid user or email');

    const passwordMatch = await compare(this.password, user.password);

    if(!passwordMatch) throw new Error('Invalid user or email');

    const { password:_, ...userData } = user;
    const token =  sign(userData, process.env.SECRET_KEY, {
      subject: userData.id,
      expiresIn: '30m',
    });

    return { token, userInfo: {...userData } };
  }
}

export { AuthenticateUser }
```

Código 3. BackEnd – Service AuthenticateUser

Este código implementa la lógica de autenticación de usuario utilizando Node.js y algunas bibliotecas como *bcryptjs* para comparar contraseñas y *jsonwebtoken* para generar tokens de autenticación. Aquí está una explicación línea por línea:

1) **import { FindUser } from '../utils/FindUser';:** Importa la clase FindUser desde el módulo *FindUser* ubicado en el directorio utils. Esta clase se utiliza para buscar información de usuario en la base de datos.
2) **import { IRequest } from './../common/types';:** Importa la interfaz *IRequest* desde el módulo *types* ubicado en el directorio *common*. Esta interfaz *define* la estructura de la solicitud, y se utiliza para asignar el parámetro *credentials* en el constructor de la clase *AuthenticateUser*.
3) **import { compare } from "bcryptjs";:** Importa la función *compare* de la biblioteca *bcryptjs*, que se utiliza para comparar la contraseña proporcionada por el usuario con la contraseña almacenada de manera segura en la base de datos.
4) **import { sign } from 'jsonwebtoken';:** Importa la función *sign* de la biblioteca *jsonwebtoken*, que se utiliza para generar un token de autenticación.
5) **import 'dotenv/config';:** Importa la configuración del archivo *.env* para acceder a las variables de entorno, que se utiliza más adelante para firmar el token de manera segura.
6) **class AuthenticateUser {:** Define una clase llamada *AuthenticateUser*.
7) **username: string; password: string;:** Declara propiedades *username* y *password* en la clase.
8) **constructor(credentials: IRequest) { Object.assign(this, credentials); }:** El constructor de la clase toma un objeto *credentials* que debe seguir la estructura definida en la interfaz *IRequest* y asigna sus propiedades al objeto actual (*this*). Esto permite inicializar las propiedades *username* y *password* de la instancia de la clase con los valores proporcionados.
9) **async execute() {:** Define un método asincrónico llamado *execute*, que lleva a cabo la lógica de autenticación.
10) **const findUser = new FindUser(this.username);:** Crea una instancia de la clase *FindUser* pasando el nombre de usuario (*this.username*) como argumento.
11) **const user = await findUser.byUsername();:** Utiliza la instancia de *FindUser* para buscar un usuario en la base de datos por nombre de usuario, y espera la respuesta de forma asíncrona.
12) **if (!user) throw new Error('Invalid user or email');:** Si no se encuentra un usuario, lanza un error indicando que el usuario o el correo electrónico son inválidos.
13) **const passwordMatch = await compare(this.password, user.password);:** Compara la contraseña proporcionada por el usuario con la contraseña almacenada en la base de datos utilizando la función *compare* de *bcryptjs*.
14) **if (!passwordMatch) throw new Error('Invalid user or email');:** Si las contraseñas no coinciden, lanza un error indicando que el usuario o el correo electrónico son inválidos.

15) **const { password: _, ...userData } = user;:** Desestructura el objeto *user*, eliminando la propiedad *password* y asignando el resto a *userData*. Esto se hace para evitar incluir la contraseña en el token.
16) **const token = sign(userData, process.env.SECRET_KEY, { subject: userData.id, expiresIn: '30m', });:** Genera un token de autenticación utilizando la función *sign* de *jsonwebtoken*. Se firma con la información del usuario (*userData*), la clave secreta almacenada en las variables de entorno (*process.env.SECRET_KEY*), se establece el sujeto como el *ID* del usuario y se define un tiempo de expiración de 30 minutos.
17) **return { token, userInfo: { ...userData } };:** Devuelve un objeto que contiene el token de autenticación y la información del usuario, excluyendo la contraseña.
18) **}:** Cierra el método *execute*.
19) **export { AuthenticateUser };** Exporta la clase *AuthenticateUser* para que pueda ser utilizada en otros archivos.

8.3.2. createUser

1. Controller

```
import { Request, Response } from "express";
import { CreateUser } from "../services/createUser";

class CreateUserController {
  async handle(request: Request, response: Response) {
    const { body } = request;

    const createUser = new CreateUser(body);

    const user = await createUser.execute();

    return response.status(200).json(user);
  }
}
export { CreateUserController };
```

Código 4. BackEnd – CreateUserController

Este código implementa un controlador (*CreateUserController*) en TypeScript utilizando el framework Express para manejar la creación de usuarios. A continuación, se proporciona una explicación línea por línea:

1) **import { Request, Response } from "express";:** Importa las interfaces *Request* y *Response* de Express, que se utilizan para asignar los objetos *request* y *response* que se pasan a los manejadores de rutas.

2) **import { CreateUser } from "../services/createUser";:** Importa la clase *CreateUser* desde el módulo *createUser* ubicado en el directorio services. Esta clase se encarga de la lógica para crear un nuevo usuario.
3) **class CreateUserController {:** Define una clase llamada *CreateUserController*.
4) **async handle(request: Request, response: Response) {:** Declara un método asincrónico llamado *handle* que toma un objeto de solicitud (*request* de tipo *Request*) y un objeto de respuesta (*response* de tipo *Response*). Este método manejará la lógica de creación de usuario.
5) **const { body } = request;:** Extrae la propiedad *body* del objeto de solicitud (*request*) utilizando desestructuración. Se asume que la solicitud contiene un cuerpo en formato JSON con la información necesaria para crear un usuario.
6) **const createUser = new CreateUser(body);:** Crea una instancia de la clase *CreateUser*, pasando el cuerpo de la solicitud como argumento al constructor. La clase *CreateUser* contiene la lógica para crear un nuevo usuario en la base de datos.
7) **const user = await createUser.execute();:** Invoca el método *execute* en la instancia de *CreateUser* de manera asíncrona. Este método realiza la creación del usuario y devuelve el objeto del usuario recién creado.
8) **return response.status(200).json(user);:** Devuelve una respuesta JSON al cliente con el estado 200 (OK) y el objeto del usuario recién creado. Esto indica que la operación de creación de usuario se ha realizado con éxito.
9) **}:** Cierra el método *handle*.
10) **export { CreateUserController };:** Exporta la clase *CreateUserController* para que pueda ser utilizada en otros archivos.

2. Service

```
import { FindUser } from '../utils/FindUser';
import { client } from '../prisma/client';
import { hash } from 'bcryptjs';
import Joi from 'joi';
import { IUserRequest, IUser } from '../common/types';

class CreateUser {
  fullname: string;
  username: string;
  password: string;
  email: string;
  medicRole: boolean;

  constructor(data: IUserRequest) {
    Object.assign(this, data);
```

```
  }

  async execute() {
    const userValues: IUser = {
      username: this.username,
      password: this.password,
      email: this.email,
      role: this.medicRole ? 'MEDIC' : 'USER',
    }
    const findUser = new FindUser(this.username);
    const user = await findUser.byUsername();
    const { error } = this.validateUserForm(userValues);

    const emailAlreadyExist = await client.user.findFirst({
      where: {
        email: this.email
      }
    });

    if(error) {
      throw new Error(error.details[0].message);
    }

    if(user || emailAlreadyExist) {
      throw new Error('User or Email already exist');
    }

    const passwordHash = await hash(this.password, 8);
    userValues.password = passwordHash;

    await client.user.create({
      data: {
        ...userValues
      },
    });
    return { status: 'success', message: 'You have been successfully registered'
};
  }

  validateUserForm = (data) =>
  Joi.object({
    username: Joi.string().min(5).required().messages({
      'any.required': '"username" is required',
      'string.min': '"username" length must be at least 5 characters long',
    }),
```

```
    email: Joi.string().email().required().messages({
      'any.required': '"email" is required',
      'string.email': '"email" must be a valid email',
    }),
    password: Joi.string().min(6).required().messages({
      'string.min': '"password" length must be 6 characters long',
      'any.required': '"password" is required',
    }),
    role: Joi.required().messages({
    }),
  }).validate(data);
}

export { CreateUser };
```

Código 5. BackEnd – Service CreateUser

Este código implementa una clase llamada *CreateUser* en TypeScript que se encarga de la lógica para crear un nuevo usuario. A continuación, se proporciona una explicación línea por línea:

1) **import { FindUser } from '../utils/FindUser';:** Importa la clase *FindUser* desde el módulo *FindUser* ubicado en el directorio utils. Esta clase probablemente se utiliza para buscar información de usuario en la base de datos.
2) **import { client } from '../prisma/client';:** Importa el objeto *client* desde el módulo *client* ubicado en el directorio prisma. Este objeto proporciona acceso a las operaciones de base de datos utilizando Prisma.
3) **import { hash } from 'bcryptjs';:** Importa la función *hash* de la biblioteca *bcryptjs*, que se utiliza para generar un hash seguro de la contraseña del usuario.
4) **import Joi from 'joi';:** Importa la biblioteca *joi* para realizar la validación de los datos del usuario.
5) **import { IUserRequest, IUser } from '../common/types';:** Importa las interfaces *IUserRequest* e *IUser* desde el módulo *types* ubicado en el directorio common. Estas interfaces definen la estructura de los datos del usuario.
6) **class CreateUser {:** Define una clase llamada *CreateUser*.
7) **fullname: string; username: string; password: string; email: string; medicRole: boolean;:** Declara propiedades para almacenar la información del usuario, como nombre completo, nombre de usuario, contraseña, correo electrónico y un indicador de si el usuario tiene un rol médico.
8) **constructor(data: IUserRequest) { Object.assign(this, data); }:** El constructor de la clase toma un objeto data que debe seguir la estructura definida en la interfaz *IUserRequest* y asigna sus propiedades al objeto

actual (*this*). Esto permite inicializar las propiedades de la instancia de la clase con los valores proporcionados.

9) **async execute() {:** Define un método asincrónico llamado *execute* que lleva a cabo la lógica de creación de usuario.
10) **const userValues: IUser = { /* ... */ };** Crea un objeto *userValues* que contiene las propiedades necesarias para crear un nuevo usuario. El valor del campo role se determina según el valor de *medicRole*.
11) **const findUser = new FindUser(this.username);:** Crea una instancia de la clase *FindUser* pasando el nombre de usuario (*this.username*) como argumento.
12) **const user = await findUser.byUsername();:** Utiliza la instancia de *FindUser* para buscar un usuario en la base de datos por nombre de usuario, y espera la respuesta de forma asíncrona.
13) **const { error } = this.validateUserForm(userValues);:** Realiza la validación de los datos del usuario utilizando el método *validateUserForm* y obtiene el posible error.
14) **const emailAlreadyExist = await client.user.findFirst({ /* ... */ });:** Verifica si ya existe un usuario con el correo electrónico proporcionado en la base de datos.
15) **if (error) { throw new Error(error.details[0].message); }:** Si hay un error de validación, lanza una excepción con el mensaje de error detallado.
16) **if (user || emailAlreadyExist) { throw new Error('User or Email already exist'); }:** Si ya existe un usuario con el mismo nombre de usuario o correo electrónico, lanza una excepción indicando que el usuario o el correo electrónico ya existen.
17) **const passwordHash = await hash(this.password, 8);:** Genera un hash seguro de la contraseña utilizando la función *hash* de *bcryptjs*.
18) **userValues.password = passwordHash;:** Asigna el hash de la contraseña al campo *password* en el objeto *userValues*.
19) **await client.user.create({ /* ... */ });:** Crea un nuevo usuario en la base de datos utilizando Prisma y los valores proporcionados en *userValues*.
20) **return { status: 'success', message: 'You have been successfully registered' };:** Si la creación del usuario es exitosa, devuelve un objeto indicando el estado y un mensaje.
21) **validateUserForm = (data) => Joi.object({ /* ... */ }).validate(data);:** Define un método llamado *validateUserForm* que utiliza la biblioteca *joi* para validar los datos del usuario según un esquema predefinido.
22) **export { CreateUser };:** Exporta la clase *CreateUser* para que pueda ser utilizada en otros archivos.

8.3.3. Middlewares – ValidateToken

```
import { Response, Request, NextFunction } from "express";
import { verify } from "jsonwebtoken";
```

```
import 'dotenv/config';

export function validateToken(req: Request, res: Response, next: NextFunction) {
  const token = req.headers.authorization;

  if(!token) throw new Error("Missing auth token");
  try {
    const user = verify(token, process.env.SECRET_KEY);
    res.locals.user = user;

    return next();
  } catch(err) {
    throw new Error("Token invalid")
  }
}
```

Código 6. BackEnd – MiddleWares ValidateToken

Este código define una función middleware llamada *validateToken* en TypeScript para validar un token de autenticación en las solicitudes entrantes a través de Express. A continuación, se proporciona una explicación línea por línea:

1) **import { Response, Request, NextFunction } from "express";:** Importa las interfaces *Response*, *Request* y *NextFunction* de Express, que se utilizan para asignar los objetos *req*, *res* y *next* que se pasan a la función *middleware*.
2) **import { verify } from "jsonwebtoken";:** Importa la función *verify* de la biblioteca *jsonwebtoken*, que se utiliza para verificar la autenticidad de un token.
3) **import 'dotenv/config';:** Importa la configuración del archivo *.env* para acceder a las variables de entorno, que se utiliza para obtener la clave secreta necesaria para verificar el token.
4) **export function validateToken(req: Request, res: Response, next: NextFunction) {:** Exporta la función middleware *validateToken*, que toma los objetos de solicitud (*req*), respuesta (*res*) y la función *next*.
5) **const token = req.headers.authorization;:** Extrae el token de autenticación del encabezado *Authorization* de la solicitud.
6) **if (!token) throw new Error("Missing auth token");:** Si no se proporciona un token en el encabezado *Authorization*, lanza un error indicando que falta el token de autenticación.
7) **try { /* ... */ } catch (err) { /* ... };:** Utiliza un bloque *try-catch* para manejar posibles errores durante la verificación del token.
8) **const user = verify(token, process.env.SECRET_KEY);:** Verifica la autenticidad del token utilizando la función *verify* de *jsonwebtoken* y la clave secreta almacenada en las variables de entorno.

9) **res.locals.user = user;:** Almacena la información del usuario verificado en el objeto *res.locals* para que esté disponible para los manejadores de rutas subsiguientes.
10) **return next();:** Llama a la función *next()* para pasar la ejecución al siguiente middleware o manejador de ruta en la cadena.
11) **throw new Error("Token invalid"):** Si hay un error durante la verificación del token, lanza un error indicando que el token es inválido.
12) **}:** Cierra el bloque *try-catch*.
13) **}:** Cierra la función *validateToken*.

8.3.4. Router – Users

```
import { UpdateUserInfoController } from
      './../controllers/updateUserInfoController';
import { Router } from 'express';
import { AuthenticateUserController } from
      '../controllers/authenticateUserController';
import { CreateUserController } from '../controllers/createUserController';
import { UpdateUserController } from '../controllers/updateUserController';
import { validateToken } from '../middlewares/validateToken';
import { limiter } from '../middlewares/rateLimit';
import { InsertUserDataController } from '../controllers/insertUseInfoController';
import { GetUserInfoController } from '../controllers/getUserInfoController';
import { GetPacientDataController } from
      '../controllers/getPacientDataController';

const userRouter = Router();

const createUser = new CreateUserController();
const authenticateUser = new AuthenticateUserController();
const updateUser = new UpdateUserController();
const inserData = new InsertUserDataController();
const findUserInfo = new GetUserInfoController();
const updateUserInfo = new UpdateUserInfoController();


userRouter.post('/sign-up', createUser.handle);
userRouter.post('/login', limiter, authenticateUser.handle);
userRouter.put('/edit', validateToken, updateUser.handle);
userRouter.post('/data/update', validateToken, updateUserInfo.handle)
userRouter.post('/data', validateToken, inserData.handle);
userRouter.get('/data/health', validateToken, findUserInfo.handle);

export { userRouter };
```

Código 7. BackEnd – Router Users

Este código configura las rutas para las operaciones relacionadas con usuarios en una aplicación Express. A continuación, se proporciona una explicación línea por línea:

1) **import { UpdateUserInfoController } from './../controllers/updateUserInfoController';:** Importa el controlador *UpdateUserInfoController* desde el archivo *updateUserInfoController* ubicado en el directorio controllers.
2) **import { Router } from 'express';:** Importa el objeto *Router* de Express para configurar las rutas.
3) **import { AuthenticateUserController } from '../controllers/authenticateUserController';:** Importa el controlador *AuthenticateUserController* desde el archivo *authenticateUserController*.
4) **import { CreateUserController } from '../controllers/createUserController';:** Importa el controlador *CreateUserController* desde el archivo *createUserController*.
5) **import {UpdateUserController } from '../controllers/updateUserController';:** Importa el controlador *UpdateUserController* desde el archivo *updateUserController*.
6) **import { validateToken } from '../middlewares/validateToken';:** Importa la función middleware *validateToken* desde el archivo *validateToken*.
7) **import { limiter } from '../middlewares/rateLimit';:** Importa la función middleware *limiter* desde el archivo *rateLimit*.
8) **import { InsertUserDataController } from '../controllers/insertUseInfoController';:** Importa el controlador *InsertUserDataController* desde el archivo *insertUseInfoController*.
9) **import { GetUserInfoController } from '../controllers/getUserInfoController';:** Importa el controlador *GetUserInfoController* desde el archivo *getUserInfoController*.
10) **import { GetPacientDataController } from '../controllers/getPacientDataController';:** Importa el controlador *GetPacientDataController* desde el archivo *getPacientDataController*.
11) **const userRouter = Router();:** Crea una instancia de Router llamada *userRouter* para configurar las rutas relacionadas con usuarios.
12) **const createUser = new CreateUserController();:** Crea una instancia del controlador *CreateUserController*.
13) **const authenticateUser = new AuthenticateUserController();:** Crea una instancia del controlador *AuthenticateUserController*.
14) **const updateUser = new UpdateUserController();:** Crea una instancia del controlador *UpdateUserController*.
15) **const inserData = new InsertUserDataController();:** Crea una instancia del controlador *InsertUserDataController*.
16) **const findUserInfo = new GetUserInfoController();:** Crea una instancia del controlador *GetUserInfoController*.

17) **const updateUserInfo = new UpdateUserInfoController();:** Crea una instancia del controlador *UpdateUserInfoController*.
18) **userRouter.post('/sign-up', createUser.handle);:** Configura la ruta POST *'/sign-up'* para llamar al método *handle* del controlador *CreateUserController* cuando se recibe una solicitud en esa ruta.
19) **userRouter.post('/login', limiter, authenticateUser.handle);:** Configura la ruta POST *'/login'* para aplicar el middleware *limiter* y luego llamar al método *handle* del controlador *AuthenticateUserController* cuando se recibe una solicitud en esa ruta.
20) **userRouter.put('/edit', validateToken, updateUser.handle);:** Configura la ruta PUT *'/edit'* para aplicar el middleware *validateToken* y luego llamar al método *handle* del controlador *UpdateUserController* cuando se recibe una solicitud en esa ruta.
21) **userRouter.post('/data/update', validateToken, updateUserInfo.handle):** Configura la ruta POST *'/data/update'* para aplicar el middleware *validateToken* y luego llamar al método *handle* del controlador *UpdateUserInfoController* cuando se recibe una solicitud en esa ruta.
22) **userRouter.post('/data', validateToken, inserData.handle);:** Configura la ruta POST *'/data'* para aplicar el middleware *validateToken* y luego llamar al método *handle* del controlador *InsertUserDataController* cuando se recibe una solicitud en esa ruta.
23) **userRouter.get('/data/health', validateToken, findUserInfo.handle);:** Configura la ruta GET *'/data/health'* para aplicar el middleware *validateToken* y luego llamar al método *handle* del controlador *GetUserInfoController* cuando se recibe una solicitud en esa ruta.
24) **export { userRouter };:** Exporta el router *userRouter* para que pueda ser utilizado en otros archivos de la aplicación.

8.4. Explicación de Funciones en Front-End

8.4.1. Services – Api.js

```
import axios from 'axios';

export const signup = async (signupValues) => {
  const { username, email, password, medicRole } = signupValues;
  const request = await axios({
    method: 'post',
    url: 'http://localhost:3000/api/user/sign-up',
    data: {
      username,
      email,
```

```
      password,
      medicRole
    },
  }).then(response => {
    return response
  })
  .catch(error => {
      return error.response
  });
  return request;
}

export const login = async (loginFormValues) => {
  const { username, password } = loginFormValues;
  const request = await axios({
    method: 'post',
    url: 'http://localhost:3000/api/user/login',
    data: {
      username,
      password,
    },
  }).catch(err => err.response);
  return request;
}

export const editUserData = async (userData, token) => {
  const request = await axios({
    method: 'put',
    url: 'http://localhost:3000/api/user/edit',
    headers: {
      Authorization: token,
    },
    data: {
      username: userData.username,
      email: userData.email,
      fullname: userData.fullname,
      id: userData.id,
    },
  });
  return await request;
}

export const setUserDataRequest = async (data, token) => {
  const { userData, healthData } = data;
  const request = await axios({
```

```
    method: 'post',
    url: 'http://localhost:3000/api/user/data',
    headers: {
      Authorization: token,
    },
    data: {
      userData,
      healthData
    },
  }).catch(err => err.response);
  return request;
}

export const updateUserData = async (data, token) => {
  const { userData, healthData } = data;
  const { id, userDataId, ...userHealthData } = healthData;
  const request = await axios({
    method: 'post',
    url: 'http://localhost:3000/api/user/data/update',
    headers: {
      Authorization: token,
    },
    data: {
      userData,
      userHealthData
    },
  }).catch(err => err.response);
  return request;
}


export const getUserDataRequest = async (token) => {
  const request = await axios({
    method: 'get',
    url: 'http://localhost:3000/api/user/data/health',
    headers: {
      Authorization: token,
    },
  }).catch(err => err.response);
  return request;
}

export const generateConsult = async (consult, token, image) => {
  const request = await axios({
    method: 'post',
```

```
    url: 'http://localhost:3000/api/consult/new',
    headers: {
      Authorization: token,
    },
    data: {
      consult
    },
  }).catch(err => err.response);;
  return request;
}

export const getUserConsults = async (token) => {
  const request = await axios({
    method: 'get',
    url: 'http://localhost:3000/api/consult/pacient',
    headers: {
      Authorization: token,
    },
  });
  return request;
}

export const getMedicConsults = async (token) => {
  const request = await axios({
    method: 'get',
    url: 'http://localhost:3000/api/consult/medic',
    headers: {
      Authorization: token,
    },
  });
  return request;
}

export const getPacientData = async (pacientId, token) => {
  const request = await axios({
    method: 'get',
    url: `http://localhost:3000/api/consult/medic/pacientData/${pacientId}`,
    headers: {
      Authorization: token,
    },
  }).catch(err => err.response);
  return request;
}

export const getChatHistory = async (roomId, token) => {
```

```
  const request = await axios({
    method: 'get',
    url: `http://localhost:3000/api/consult/chat/history/${roomId}`,
    headers: {
      Authorization: token,
    },
  }).catch(err => err.response);
  return request.data;
}
```

Código 8. FrontEnd – Services Api

Este código proporciona funciones que utilizan la biblioteca Axios para realizar solicitudes HTTP a diferentes rutas de una API. Cada función está diseñada para interactuar con la API de un servidor Express que maneja operaciones relacionadas con usuarios y consultas médicas. A continuación, se proporciona una explicación para cada función:

1) **signup:** Realiza una solicitud HTTP POST para registrarse en la aplicación. Toma los valores del formulario de registro (*signupValues*) y los envía al endpoint *'/api/user/sign-up'* del servidor.
2) **login:** Realiza una solicitud HTTP *POST* para iniciar sesión en la aplicación. Toma los valores del formulario de inicio de sesión (*loginFormValues*) y los envía al endpoint *'/api/user/login'* del servidor.
3) **editUserData:** Realiza una solicitud HTTP *PUT* para actualizar la información del usuario. Toma los datos del usuario (*userData*) y el token de autenticación (*token*) y los envía al endpoint *'/api/user/edit'* del servidor.
4) **setUserDataRequest:** Realiza una solicitud HTTP *POST* para insertar información del usuario y datos de salud. Toma los datos (*data*) que incluyen la información del usuario y los datos de salud, así como el token de autenticación (*token*), y los envía al endpoint *'/api/user/data'* del servidor.
5) **updateUserData:** Realiza una solicitud HTTP *POST* para actualizar la información del usuario y los datos de salud. Toma los datos (*data*) que incluyen la información del usuario y los datos de salud, así como el token de autenticación (*token*), y los envía al endpoint *'/api/user/data/update'* del servidor.
6) **getUserDataRequest:** Realiza una solicitud HTTP *GET* para obtener la información de salud del usuario. Toma el token de autenticación (*token*) y lo envía al endpoint *'/api/user/data/health'* del servidor.
7) **generateConsult:** Realiza una solicitud HTTP *POST* para generar una nueva consulta médica. Toma la información de la consulta (*consult*) y el token de autenticación (*token*) y los envía al endpoint *'/api/consult/new'* del servidor.

8) **getUserConsults:** Realiza una solicitud HTTP *GET* para obtener las consultas de un paciente. Toma el token de autenticación (*token*) y lo envía al endpoint *'/api/consult/pacient'* del servidor.
9) **getMedicConsults:** Realiza una solicitud HTTP *GET* para obtener las consultas de un médico. Toma el token de autenticación (*token*) y lo envía al endpoint *'/api/consult/medic'* del servidor.
10) **getPacientData:** Realiza una solicitud HTTP *GET* para obtener datos de un paciente específico. Toma el ID del paciente (*pacientId*) y el token de autenticación (*token*), y los envía al endpoint *'/api/consult/medic/pacientData/{pacientId}'* del servidor.
11) **getChatHistory:** Realiza una solicitud HTTP *GET* para obtener el historial de chat de una sala específica. Toma el ID de la sala de chat (*roomId*) y el token de autenticación (*token*), y los envía al endpoint *'/api/consult/chat/history/{roomId}'* del servidor.

8.4.2. AuthContext

```
import React, { createContext } from "react";
import { useHistory } from "react-router-dom";
import { signup, login } from "../services/api";
import { useDispatch, useSelector } from "react-redux";
import { resetSignup, setLoginForm, setSignState, setSignup } from
      "../redux/slices/authSlice";
export const AuthContext = createContext({});

export default function AuthContextProvider(props) {
  const globalState = useSelector(state => state.authentication);
  const { signupValues, loginForm } = globalState;
  const history = useHistory();
  const dispatch = useDispatch();
  function fillFormFields({ target }, isLogin) {
    const value = target.type === "checkbox" ? target.checked : target.value;
    const name = target.name;
    if (isLogin) {
      dispatch(setLoginForm({name, value}));
    }
    dispatch(setSignup({name, value }));
  }
  async function handleSignup(e) {
    e.preventDefault();
    dispatch(setSignState(
      {signUp: true, signIn: false, serverResponse:
        { message: '', status: ''},
    }));
    const { data } = await signup(signupValues);
```

```
    if(data.status === 'success') {
      dispatch(resetSignup());
      return dispatch(setSignState({signUp: false, signIn: false, serverResponse:
{ ...data } }));
    }
    setInterval(()=> {
      dispatch(setSignState(
        {signUp: false, signIn: false, serverResponse:
          { message: data.message, status: data.status },
      }));
    }, 4000)
  }
 async function handleLogin(e) {
    e.preventDefault();
    const { data } = await login(loginForm);
    if(data.token) {
      window.localStorage.setItem("user", JSON.stringify(data));
      setSignState((prevState) => ({ ...prevState, signIn: false }));
      if(data.userInfo.role === 'USER') return history.push("/dashboard");
      return history.push("/dashboard/medic");
    }
    setSignState((prevState) => ({ ...prevState, signIn: true }));
    return data;
  }
  return (
    <AuthContext.Provider
      value={{
        fillFormFields,
        handleLogin,
        handleSignup,
        setSignState,
      }}
    >
      {props.children}
    </AuthContext.Provider>
  );
}
```

Código 9. Context – AuthContext

Este código implementa un contexto de autenticación en React utilizando *createContext* y proporciona un proveedor de contexto llamado *AuthContextProvider*. A continuación, se proporciona una explicación línea por línea:

1) **import React, { createContext } from "react";:** Importa las bibliotecas React y *createContext* de React.

2) **import { useHistory } from "react-router-dom";:** Importa el hook *useHistory* de "*react-router-dom*" para manejar el historial de navegación.
3) **import { signup, login } from "../services/api";:** Importa las funciones *signup* y *login* desde el módulo *"../services/api*", que contiene llamadas a la API para las operaciones de registro e inicio de sesión.
4) **import { useDispatch, useSelector } from "react-redux";:** Importa las funciones *useDispatch* y *useSelector* de "*react-redux*" para interactuar con el *store* de Redux.
5) **import { resetSignup, setLoginForm, setSignState, setSignup } from "../redux/slices/authSlice";:** Importa acciones del *slice* de Redux llamado *authSlice* para actualizar el estado de autenticación en el *store* de Redux.
6) **export const AuthContext = createContext({});:** Crea un contexto de autenticación utilizando *createContext* y exporta el objeto *AuthContext*.
7) **export default function AuthContextProvider(props) {:** Define un componente de función llamado *AuthContextProvider* que actúa como un proveedor de contexto.
8) **const globalState = useSelector(state => state.authentication);:** Utiliza *useSelector* para obtener el estado global del *store* de Redux, específicamente el *slice* llamado *authentication*.
9) **const { signupValues, loginForm } = globalState;:** Destructura el estado global para obtener las propiedades *signupValues* y *loginForm*.
10) **const history = useHistory();:** Inicializa el hook *useHistory* para acceder al historial de navegación.
11) **const dispatch = useDispatch();:** Inicializa la función *dispatch* para despachar acciones a Redux.
12) **function fillFormFields({ target }, isLogin) { /* ... */ };** Define una función *fillFormFields* que actualiza el estado de Redux con los valores de los formularios de inicio de sesión o registro.
13) **async function handleSignup(e) { /* ... */ };** Define una función *handleSignup* que maneja el proceso de registro, hace una llamada a la API y actualiza el estado según la respuesta del servidor.
14) **async function handleLogin(e) { /* ... */ };** Define una función *handleLogin* que maneja el proceso de inicio de sesión, hace una llamada a la API y actualiza el estado según la respuesta del servidor.
15) **return (/* ... */);** Devuelve el componente *AuthContext.Provider*, que envuelve a los componentes hijos y proporciona el contexto de autenticación a través del valor del proveedor.
16) **{props.children};** Renderiza los componentes hijos dentro del proveedor de contexto.
17) **); };** Cierra la función *AuthContextProvider*.

9. Seguridad

La seguridad en una aplicación web médica es crucial para proteger la privacidad y confidencialidad de los datos sensibles de los usuarios. Una forma efectiva de lograrlo es mediante el uso de JSON Web Tokens (JWT) en conjunto con una API.

JWT es un estándar abierto (RFC 7519) que permite autenticar y autorizar a los usuarios de manera segura. En el contexto de una aplicación web médica, JWT se utiliza para generar un token de acceso después de que un usuario se autentica correctamente. Este token contiene información sobre la identidad del usuario y los permisos específicos.

Cuando el usuario realiza una solicitud a través de la API de la aplicación web médica, el token JWT se envía junto con la solicitud. La API valida la autenticidad del token utilizando una clave secreta compartida, lo que garantiza que el token no haya sido alterado o falsificado.

Una vez validado, el token JWT permite que la API autorice al usuario y les otorgue acceso a los recursos solicitados. Además, la API puede verificar los permisos específicos en el token para garantizar que el usuario tenga los privilegios necesarios para realizar la acción solicitada.

La seguridad de JWT en una aplicación web médica también se puede fortalecer mediante la implementación de prácticas adicionales, como el uso de HTTPS para cifrar la comunicación entre el cliente y el servidor, y la gestión adecuada de las claves secretas utilizadas para firmar y verificar los tokens.

En resumen, la combinación de JSON Web Tokens (JWT) y una API es una forma efectiva de garantizar la seguridad en una aplicación web médica. Al utilizar JWT, se proporciona una capa adicional de autenticación y autorización, asegurando que solo los usuarios autorizados puedan acceder a los recursos protegidos y mantener la confidencialidad de la información médica confidencial.

10. Conclusiones

Al llegar al final de este arduo viaje académico, concluyo con la realización de mi Trabajo de Fin de Grado, un proceso que ha representado un desafío significativo pero gratificante. A lo largo de esta travesía, me he sumergido en las complejidades del diseño y desarrollo de una aplicación web médica, enfrentando obstáculos y superando retos que me han permitido crecer tanto profesional como personalmente.

La investigación y aplicación de metodologías específicas, el análisis funcional, el diseño técnico y la implementación de tecnologías adecuadas han sido pasos esenciales para llevar a cabo este proyecto. La arquitectura del sistema, la elección de tecnologías, el diseño de la base de datos y el desarrollo del backend y frontend han conformado un conjunto integral que ahora se materializa en una aplicación funcional.

El proceso de pruebas y ajustes ha sido crucial para garantizar el correcto funcionamiento de la aplicación, identificando y corrigiendo posibles errores. La documentación técnica generada, incluyendo manuales de usuario, sirve como guía detallada para comprender la estructura, implementación y mantenimiento del sistema.

Este proyecto no solo marca el fin de una etapa académica, sino también la culminación de un esfuerzo y el comienzo de nuevas oportunidades y aprendizajes.

11. Bibliografía

Duvander, J., y Romhagen, O. (2019). *What affects the choice of a JavaScript framework*. https://www.diva-portal.org/smash/get/diva2:1352822/FULLTEXT01.pdf

Eryılmaz, M. E. (2019). Entrepreneurship. En M. Khosrow-Pour (Ed.), *Encyclopedia of Information Science and Technology* (4ta Ed., pp. 433-444). IGI Global. https://doi.org/10.4018/978-1-5225-7766-9.ch034

Escudero Cuevas, J. (2019). *Cómo hacer el plan de empresa de una app: ¿Quieres saber cuáles son las claves del plan de negocio de una aplicación móvil?* Emprendedores portad web: Planes de negocio- Crea tu empresa. https://www.emprendedores.es/crear-una-empresa/a77608/plan-negocio-app-aplicacion-movil/

Fernández Silano, M. (2014). La Salud 2.0 y la atención de la salud en la era digital. *Revista Médica de Risaralda*, *20*(1), 41-46. https://doi.org/10.22517/25395203.8483

Geneves, P., Layaida, N., y Quint, V. (2012). On the analysis of cascading style sheets. *Proceedings of the 21st international conference on World Wide Web*, 809-818. https://doi.org/10.1145/2187836.2187946

Google. (2023). *¿Qué es MySQL?* Google Cloud. https://cloud.google.com/mysql?hl=es-419

Hernández-Rodríguez, J. I. (2014). *Análisis y Desarrollo Web*. Universidad de Cienfuegos.

IBM. (2021). *¿Qué es MongoDB?* IBM Cloud. https://www.ibm.com/es-es/topics/mongodb

Kantor, I. (2022). An Introduction to JavaScript. En *The Modern JavaScript Tutorial*. Kindle Edition.

Kolade, C. (2021). *What is HTML – Definition and Meaning of Hypertext Markup Language*. freeCodeCamp. https://www.freecodecamp.org/news/what-is-html-definition-and-meaning/

Meta Platforms Inc. (2017). *React: A JavaScript library for building user interfaces*. Reactjs.org. https://legacy.reactjs.org

Microsoft. (2013). *TypeScript: JavaScript With Syntax For Types*. TypeScript. https://www.typescriptlang.org

Mozilla Foundation. (2023a). *HTML basics*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics

Mozilla Foundation. (2023b). *Primeros pasos en React*. MDN Web Docs. https://developer.mozilla.org/es/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started

Mozilla Foundation. (2023c). *What is CSS?* MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/CSS

Mozilla Foundation. (2023d). *What is JavaScript?* MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript

OpenJS Foundation. (2010). *Express 4.18.1*. Express. https://www.npmjs.com/package/express/v/4.18.1

OpenJS Foundation. (2011). *About Node.js®*. Nodejs. https://nodejs.org/en/about

Organización Internacional del Trabajo. (2021). Observatorio de la OIT: La COVID‑19 y el mundo del trabajo. Estimaciones actualizadas y análisis. *Observatorio de la OIT: La COVID ‑ 19 y el mundo del trabajo*, *1*, 1-28. https://www.ilo.org/wcmsp5/groups/public/---dgreports/---dcomm/documents/briefingnote/wcms_755917.pdf

Osorio, F., Gálvez, E., y Murillo, G. (2010). La estrategia y el emprendedor: diversas perspectivas para el análisis. *Cuadernos de Administración*, *Vol.1*(No.43), 65-80. http://www.scielo.org.co/pdf/cuadm/n43/n43a6.pdf

Redacción Codecademy. (2020). *What is Express.js?* Codecademy. https://www.codecademy.com/article/what-is-express-js#:~:text=js.-,Express.,js.

Redacción GeeksforGeeks. (2023). *TypeScript*. GeeksforGeeks. https://www.geeksforgeeks.org/typescript/

Redacción Guide IONOS. (2020). *Desarrollo web: fundamentos y herramientas*. Guide IONOS. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/fundamentos-del-desarrollo-web/

Semah, B. (2022). *What Exactly is Node.js? Explained for Beginners*. freeCodeCamp. https://www.freecodecamp.org/news/what-is-node-js/

Terrón, J., Peñafiel, C., Catalán, D., y Ramos, M. (2021). *Comunicación y promoción de la Salud en la era digital*. Dykinson, S.L. https://doi.org/10.2307/j.ctv1s7cc6j

Willard, W. (2009). *HTML: A Beginner's Guide*. McGraw Hill Interamericana.

12. Anexos

Anexo 1 – Instrucciones de Uso de la Aplicación

A continuación, se proporcionan las instrucciones de uso para la aplicación web médica:

Bienvenido/a a nuestra aplicación web médica, diseñada para brindarle acceso rápido y seguro a una variedad de servicios de salud y funciones personalizadas. A continuación, encontrará una guía detallada sobre cómo aprovechar al máximo todas nuestras características.

1. **Registro e inicio de sesión:**

Para comenzar, le invitamos a visitar nuestro sitio web https://mednet-4330e95d057a.herokuapp.com/, donde encontrará el botón "Registrarse". Al hacer clic en él, se le pedirá que proporcione los datos requeridos para crear su cuenta. Asegúrese de ingresar su nombre de usuario preferido, dirección de correo electrónico y contraseña. Además, en caso de ser médico, tendrá la opción de indicarlo durante el proceso de registro.

2. **Explorar servicios y funciones:**

Una vez que haya iniciado sesión, llegará al panel de control principal. Aquí, encontrará una variedad de secciones y servicios a su disposición.

Perfil – Para comenzar, le recomendamos explorar la sección de "Perfil". En esta sección, podrá acceder a su perfil, donde podrá ver y cambiar toda su información básica. Si necesita actualizar alguno de sus datos, simplemente seleccione la opción de "Editar información" y realice los cambios necesarios de manera fácil y rápida.

Historial Médico – en este apartado tendrá acceso a su historial médico, donde podrá indicar y consultar todos sus datos relacionados con su salud. Esto incluye información como alergias, medicamentos que toma, consumo de alcohol, hábito de fumar, consumo de drogas, así como su altura, peso y cualquier enfermedad que padezca.

Lo mejor de todo es que tendrá la opción de editar todos estos datos con solo hacer clic en el botón "Editar". De esta manera, podrá mantener su historial médico actualizado en todo momento, garantizando una precisión y veracidad de la información que se encuentra registrada en el sistema. Esto es especialmente útil cuando hay cambios en su estado de salud o necesita proporcionar detalles actualizados para el cuidado y seguimiento médico adecuado.

Consultas – Nuestra aplicación web médica ofrece diversas funcionalidades para mejorar la gestión de su atención médica. En la sección de "Consultas", podrá acceder a toda la información relevante sobre cada consulta médica específica, incluyendo la fecha y motivo de la visita. Esto le permitirá tener una visión completa de su historial médico y facilitará el seguimiento de su atención.

Además, desde esta sección, podrá comunicarse de manera rápida y accesible con un médico, lo que le brindará la oportunidad de realizar consultas adicionales o consultar cualquier duda que tenga. Esto agiliza la comunicación y le permite obtener respuestas a sus preguntas de forma oportuna.

Le invitamos a explorar todas las secciones y funciones disponibles en nuestra aplicación web médica, para que pueda beneficiarse al máximo en su cuidado de salud. Si en algún momento tiene alguna pregunta o necesita asistencia, nuestro equipo de soporte técnico estará disponible para ayudarle. No dude en ponerse en contacto con ellos.

Estas son algunas de las principales características de nuestra aplicación web médica. Explore todas las secciones y funciones para descubrir cómo puede beneficiarse de ellas en su cuidado de salud. Si tiene alguna pregunta o necesita asistencia, no dude en ponerse en contacto con nuestro equipo de soporte técnico.

¡Esperamos que disfrute utilizando nuestra aplicación y que les facilite el acceso a servicios médicos de calidad!

Anexo 2 – Repositorio de GitHub

El repositorio asociado a este TFG se encuentra disponible en GitHub, una plataforma líder en alojamiento y colaboración de proyectos de desarrollo de software. El acceso al repositorio puede obtenerse a través del siguiente enlace:

https://github.com/KevinNavaArgos/MedNet

Anexo 3 – Manual de Instalación

Manual de Instalación (Ver PDF adjunto)