UNIVERSIDAD LUTERANA SALVADOREÑA

FACULTAD DE CIENCIAS DEL HOMBRE Y LA NATURALEZA

LICENCIATURA EN CIENCIAS DE LA COMPUTACIÓN

Proyecto: BookMedic. Sistema de citas médicas desarrollado con node.js

Equipo Desarrollador:

Delgado Mejía, Gabriela Lucía	dm01136416
Mozo García, Josué Ezequiel	mg01135024
Murcia Hernández, José Francisco	mh01134486
Vásquez Domínguez, Kelvin Adonay	vd01136157

Cátedra: Nuevas Tendencias de Programación

Horario: viernes y sábado, 7:00-8:40 am

Facilitador: Irwin Guardado

San Salvador, 03 de diciembre de 2022

INTRODUCCIÓN

Las aplicaciones se han convertido en herramientas que permiten que las personas puedan estar conectadas, actualizadas y que puedan acceder a la información de su preferencia de forma rápida y en cualquier lugar en el que se encuentren. Estas aplicaciones se han convertido una parte importante en el desarrollo e innovación de la nueva era digital, es por esto que poder utilizar estas nuevas aplicaciones en las diferentes ramas de comercio e interconectividad de los usuarios se vuelve importante para las empresas. Áreas como las comunicaciones, turismo, educación, comercio, medicina se van adaptando a estas nuevas tendencias. Esta última ha tomado mucha más fuerza en los últimos años, convirtiéndose en una herramienta que le permite tener mejores controles de los movimientos que se realizan. Es por esto y otras razones que traen múltiples beneficios a los usuarios, se ha desarrollado esta aplicación de citas médicas llamada: BookMedic, que permitirá a los usuarios y centro que hagan uso de ellas tener un mayor control en sus pacientes, citas y calendarización de estas mismas; brindando una innovación y facilidad en el control de citas médicas de las áreas que deseen agregarse, un mejor control de pacientes y médicos.

El presente documento muestra el desarrollo de todas las fases necesarias para la realización de la aplicación de citas médicas brindando a los interesados todas aquellas herramientas utilizadas, lenguajes de programación, bases de datos, diseño, etc. A su vez se realiza el planteamiento de la problemática a resolver, así como sus posibles beneficios con los cuales se pretende poder automatizar e innovar la realización de citas médicas a través de los sitios web del área de medicina. Se presenta el desarrollo de una api para citas médicas utilizando node js, javascript y mongodb como lenguajes de programación y gestor de base de datos respectivamente; diseñando un sistema que permita el control de citas, el registro de paciente y médicos y las acciones que permiten el manejo respectivo de la información registrada por los usuarios.

El presente proyecto presenta las innovaciones que pueden aplicarse en el área de medicina y su automatización para los registros médicos. Así como la aplicación de nuevas formas de programación.

ÍNDICE

ANÁLISIS DEL PROBLEMA

TEMA

BookMedic. API REST de sistema de citas médicas.

PROBLEMÁTICA A RESOLVER

La problemática a resolver es la creación de una API de un sistema de citas médicas con Javascript, Node js adicional a eso se utilizará como gestor de base datos MongoDB el cual tendrá las funciones de gestión de citas, gestión de pacientes y los usuarios con acceso a sistemas, también se podrá ver el estado de las citas.

Una de las principales problemáticas y como reto es utilizar las características en JavaScript async await funciones de tipo flecha conectarse a la base de datos también realizar consultas a la base de datos de MongoDB en la cual se está utilizando un módulo llamado mongoose.

También una de las problemáticas que se resolverán es que con la creación de este sistema se llevará el control de las citas, pacientes activos, se obtendrá un historial de citas por pacientes y por médicos esto también implementando un buscador avanzado así mismo este creará sus respectivos reportes.

OBJETIVOS

OBJETIVO GENERAL

- Realizar una API Rest para un sistema de citas médicas

OBJETIVOS ESPECÍFICOS

- Realizar un módulo para la gestión de citas.
- Realizar un módulo para la gestión de pacientes.
- Realizar un módulo de manejo de estado de citas.
- Realizar un módulo que permita ver el estado de pago.
- Realizar módulos que permitan ver el historial de citas por paciente y médico.

ANÁLISIS DE BENEFICIOS DEL PROYECTO

- El usuario podrá gestionar las citas
- El paciente podrá tener el registro de los pacientes
- El usuario podrá visualizar el estado de pago si estos están pendientes, pagados o anulados.
- El usuario podrá ver el estado de las citas si están pendientes, si el cliente no asistió o está cancelada.
- Se podrá visualizar el historial de citas de pacientes.
- Se podrá visualizar el historial de citas por médico
- Se podrá gestionar el acceso de usuarios al sistema.
- Se podrá generar reportes de las citas

JUSTIFICACIÓN

El presente documento consistirá en la creación de una API de sistema de citas médicas de esta manera el usuario utilizando este sistema podrá tener un control de las citas, con este sistema el usuario podrá gestionar de manera óptima y eficiente lo involucrado con las citas.

Entre los beneficios se pueden mencionar que gracias a este sistema se puede obtener y visualizar de forma efectiva ya que se podrá ver el registro de las citas de los pacientes y de los médicos, también se podrá gestionar la parte de pagos para ver si ya se efectuó el pago o está pendiente. Otra de las características es que se visualiza el estado de las citas como si se realizó o si fue cancelada**.** también es importante recalcar que se está utilizando tecnologías de desarrollo como Nodejs, Rect JS, el gestor de base de datos Mongodb.

Es necesario desarrollar el proyecto porque gracias a este, el usuario podrá tener una mejor gestión de la información y le permitirá evaluar cada uno de los puntos importantes, ya que tendrá la información de forma ordenada y en tiempo real, también cabe mencionar para que el cliente no tenga problema con la asignación de citas el sistema le permitirá ver cuáles son las horas disponibles.

METODOLOGÍA Y CICLO DE VIDA UTILIZADOS

La metodología utilizada para el desarrollo de este proyecto se basa en la recolección de información relacionada al tema a tratar, buscando opciones y respaldos teóricos que puedan servir como bases para el desarrollo de la aplicación web. Se hizo una revisión bibliográfica de sistemas similares que podrían servir como soporte para el desarrollo del mismo, cumpliendo con los requerimientos de las diferentes herramientas de programación.

A través de las técnicas de observación, evaluación y aprobación, se llevó a cabo la toma de decisiones sobre las diferentes ideas de sistemas a desarrollar, evaluando su tiempo de desarrollo, beneficios de realizarlo, si logra brindar una solución a la problemática planteada y otras consideraciones a aplicar. La metodología aplicada permitió la selección del proyecto luego de una lluvia de ideas que se plantearon como grupo. Realizando los puntos de observación, de cada idea presentada, la evaluación de las ideas propuestas para su posterior aprobación.

La revisión bibliográfica permitió identificar los principales beneficios y los puntos que podrían cubrirse para beneficios de terceros. Primeramente, se logró identificar la necesidad que puede existir de aplicaciones que permitan un control de las citas médicas y un acceso fácil a toda la información en esta área.

CICLO DE VIDA DEL SOFTWARE

El ciclo de vida del software y el proceso de desarrollo que este conlleva ha sido utilizado para facilitar el desarrollo de los sistemas de información, ayuda a los gestores de proyecto con la planificación del desarrollo y la puesta en marcha de un sistema de información que reúna los requisitos del usuario, y que sea completado a tiempo y dentro de los límites del presupuesto. Con la aplicación de las fases de análisis, diseño, desarrollo, pruebas, implementación y mantenimiento, se tiene como objetivo alcanzar mejores resultados de funcionalidad e innovación en el sitio desarrollado. Determinar el orden de las fases del proceso de software. entre las funciones que se esperó poder tener resultados fueron:

- Establecer los criterios de transición para pasar de una fase a la siguiente.
- Definir las entradas y salidas de cada fase.
- Describir los estados por los que pasa el producto.
- Describir las actividades a realizar para transformar el producto.

● Definir un esquema que sirve como base para planificar, organizar, coordinar y desarrollar.

Aplicando las fases de desarrollo se pudo identificar de la siguiente manera:

- Planificación: en esta etapa inicial se aplicaron las técnicas de recolección de información, el planteamiento de las ideas y la aprobación de la idea a desarrollar. Se estudiaron las propuestas de acuerdo a los puntos establecidos para solucionar o beneficiar a los usuarios. Una vez aceptada la idea, se prosiguió a la planificación del desarrollo del proyecto, tomando en consideración los tiempos de desarrollo.
- Análisis: en este punto se definieron los beneficios del proyecto, así como la importancia que puede generar y se establecieron las características principales del sistema.
- Diseño: se plantean todas las fases que deberán cubrirse, así como todos los aspectos a cubrir, lenguajes de programación, selección de las herramientas de trabajo, diseño de las bases de datos, vistas, lógica de programación, modelos aplicados, etc. Se trata principalmente de la estructura general del proyecto.
- Implementación: se definieron las herramientas más adecuadas para su desarrollo, lenguaje de programación, gestor de base de datos; definiendo javascript, nodejs y mongodb, vue como herramientas fundamentales para el desarrollo del código.
- Pruebas: se realizan las pruebas de cada una de las partes del proyecto, para observar y corregir posibles errores o acciones que no estén cumpliendo con los requerimientos establecidos.

Se utilizó el modelo en cascada ya que permite la división del proyecto en fases (las cuales se explicaron en los apartados anteriores) con las cuales se busca un desarrollo progresivo y vinculado entre cada una de ellas que permite una revisión detallada, así como la realización y comprobación de cada una de ellas.

Para que todo este proceso pueda cumplirse y el desarrollo del sistema sea factible y con los resultados esperados se utilizaron herramientas que permitieron mantener una constante actualización del sistema que permitió que todos los involucrados tuvieran los accesos necesarios para cumplir con las necesidades descritas con anterioridad.

ANÁLISIS.

API es una interfaz que dos sistemas de computación utilizan para intercambiar información de manera segura a través de Internet. La mayoría de las aplicaciones para empresas deben comunicarse con otras aplicaciones internas o de terceros para llevar a cabo varias tareas. API admite este intercambio de información porque se sigue un estándar de comunicación de software seguros, confiables y eficientes.

Una interfaz de programa de aplicación (API) define las reglas que se deben seguir para comunicarse con otros sistemas de software. Al desarrollar un API es para que otras aplicaciones puedan comunicarse con sus aplicaciones mediante programación. Un ejemplo, es la aplicación de control de citas exponemos una API que solicita el nombre completo de un médico, un paciente y un rango de fechas en las cuales se almacenan las citas programadas. Cuando recibe esta información, procesa internamente la planilla de horarios de citas y devuelve los días en los cuales tendrá que brindar consulta a los pacientes.

Se puede pensar en una API web como una puerta de enlace entre los clientes y los recursos de la Web.

Clientes.

Los clientes son usuarios que desean acceder a información desde la Web. El cliente puede ser una persona o un sistema de software que utiliza la API.

Recursos

Los recursos son la información que diferentes aplicaciones proporcionan a sus clientes. Los recursos pueden ser imágenes, videos, texto, números o cualquier tipo de datos. La máquina encargada de entregar el recurso al cliente también recibe el nombre de servidor. Las organizaciones utilizan las API para compartir recursos y proporcionar servicios web, a la vez que mantienen la seguridad, el control y la autenticación y también ayudan a determinar qué clientes obtienen acceso a recursos internos específicos.

¿Qué es REST?

La transferencia de estado representacional (REST) es una arquitectura de software que impone condiciones sobre cómo debe funcionar una API. En un principio, REST se creó como una guía para administrar la comunicación en una red compleja como Internet. Es posible

utilizar una arquitectura basada en REST para admitir comunicaciones confiables y de alto rendimiento a escala. Puede implementarla y modificarla fácilmente, lo que brinda visibilidad y portabilidad entre plataformas a cualquier sistema de API.

A continuación, se presentan algunos de los principios del estilo arquitectónico de REST:

Interfaz uniforme

La interfaz uniforme es fundamental para el diseño de cualquier servicio web RESTful. Ella indica que el servidor transfiere información en un formato estándar. El recurso formateado se denomina representación en REST. Este formato puede ser diferente de la representación interna del recurso en la aplicación del servidor. Por ejemplo, el servidor puede almacenar los datos como texto, pero enviarlos en un formato de representación HTML.

La interfaz uniforme impone cuatro limitaciones de arquitectura:

Las solicitudes deben identificar los recursos. Lo hacen mediante el uso de un identificador uniforme de recursos.

Los clientes tienen información suficiente en la representación del recurso como para modificarlo o eliminarlo si lo desean. El servidor cumple esta condición por medio del envío de los metadatos que describen el recurso con mayor detalle.

Tecnología sin estado

En la arquitectura de REST, la tecnología sin estado se refiere a un método de comunicación en el cual el servidor completa todas las solicitudes del cliente independientemente de todas las solicitudes anteriores.

Los clientes pueden solicitar recursos en cualquier orden, y todas las solicitudes son sin estado o están aisladas del resto. Esta limitación del diseño de la API REST implica que el servidor puede comprender y cumplir por completo la solicitud todas las veces.

Sistema por capas

En una arquitectura de sistema por capas, el cliente puede conectarse con otros intermediarios autorizados entre el cliente y el servidor y todavía recibirá respuestas del servidor. Los servidores también pueden pasar las solicitudes a otros servidores. Es posible diseñar el servicio web RESTful para que se ejecute en varios servidores con múltiples capas,

como la seguridad, la aplicación y la lógica empresarial, que trabajan juntas para cumplir las solicitudes de los clientes. Estas capas se mantienen invisibles para el cliente.

Almacenamiento en caché

Los servicios web RESTful admiten el almacenamiento en caché, que es el proceso de almacenar algunas respuestas en la memoria caché del cliente o de un intermediario para mejorar el tiempo de respuesta del servidor.

¿Cómo funcionan las API?

La función básica de una API es la misma que navegar por Internet. Cuando requiere un recurso, el cliente se pone en contacto con el servidor mediante la API. A continuación, se indican los pasos generales para cualquier llamada a la API:

El cliente envía una solicitud al servidor. El cliente sigue la documentación de la API para dar formato a la solicitud de una manera que el servidor comprenda.

El servidor autentica al cliente y confirma que este tiene el derecho de hacer dicha solicitud.

El servidor recibe la solicitud y la procesa internamente.

Luego, devuelve una respuesta al cliente. Esta respuesta contiene información que indica al cliente si la solicitud se procesó de manera correcta. La respuesta también incluye cualquier información que el cliente haya solicitado.

Los detalles de la solicitud y la respuesta de la API varían un poco en función de cómo la API se haya diseñado.

Método

Los desarrolladores a menudo implementamos API REST mediante el uso de protocolos de transferencia de hipertexto (HTTP). Un método de HTTP informa al servidor lo que debe hacer con el recurso. A continuación, se indican cuatro métodos de HTTP comunes:

GET

Los clientes utilizan GET para acceder a los recursos que están ubicados en el URL especificado en el servidor. Pueden almacenar en caché las solicitudes GET y enviar

parámetros en la solicitud de la API REST para indicar al servidor que filtre los datos antes de enviarlos.

POST

Los clientes usan POST para enviar datos al servidor. Incluyen la representación de los datos con la solicitud. Enviar la misma solicitud POST varias veces produce el efecto secundario de crear el mismo recurso varias veces.

PUT

Los clientes utilizan PUT para actualizar los recursos existentes en el servidor. A diferencia de POST, el envío de la misma solicitud PUT varias veces en un servicio web REST da el mismo resultado.

DELETE

Los clientes utilizan la solicitud DELETE para eliminar el recurso. Una solicitud DELETE puede cambiar el estado del servidor. Sin embargo, si el usuario no cuenta con la autenticación adecuada, la solicitud fallará.

Datos

Las solicitudes de la API REST pueden incluir datos para que los métodos POST, PUT y otros métodos HTTP funcionen de manera correcta.

Parámetros

Las solicitudes de la API REST pueden incluir parámetros que brindan al servidor más detalles sobre lo que se debe hacer. A continuación, se indican algunos tipos de parámetros diferentes:

Los parámetros de ruta especifican los detalles del URL.

Los parámetros de consultas solicitan más información acerca del recurso.

Cuerpo del mensaje

El cuerpo de la respuesta contiene la representación del recurso. El servidor selecciona un formato de representación adecuado en función de lo que contienen los encabezados de la

solicitud. Los clientes pueden solicitar información en los formatos XML o JSON, lo que define cómo se escriben los datos en texto sin formato.

Diagrama de Caso de uso

En la siguiente imagen se muestran las funciones principales que realizará cada actor en el desarrollo de la API, optando por la implementación del modelo MVC, el cual está orientado a objetos.

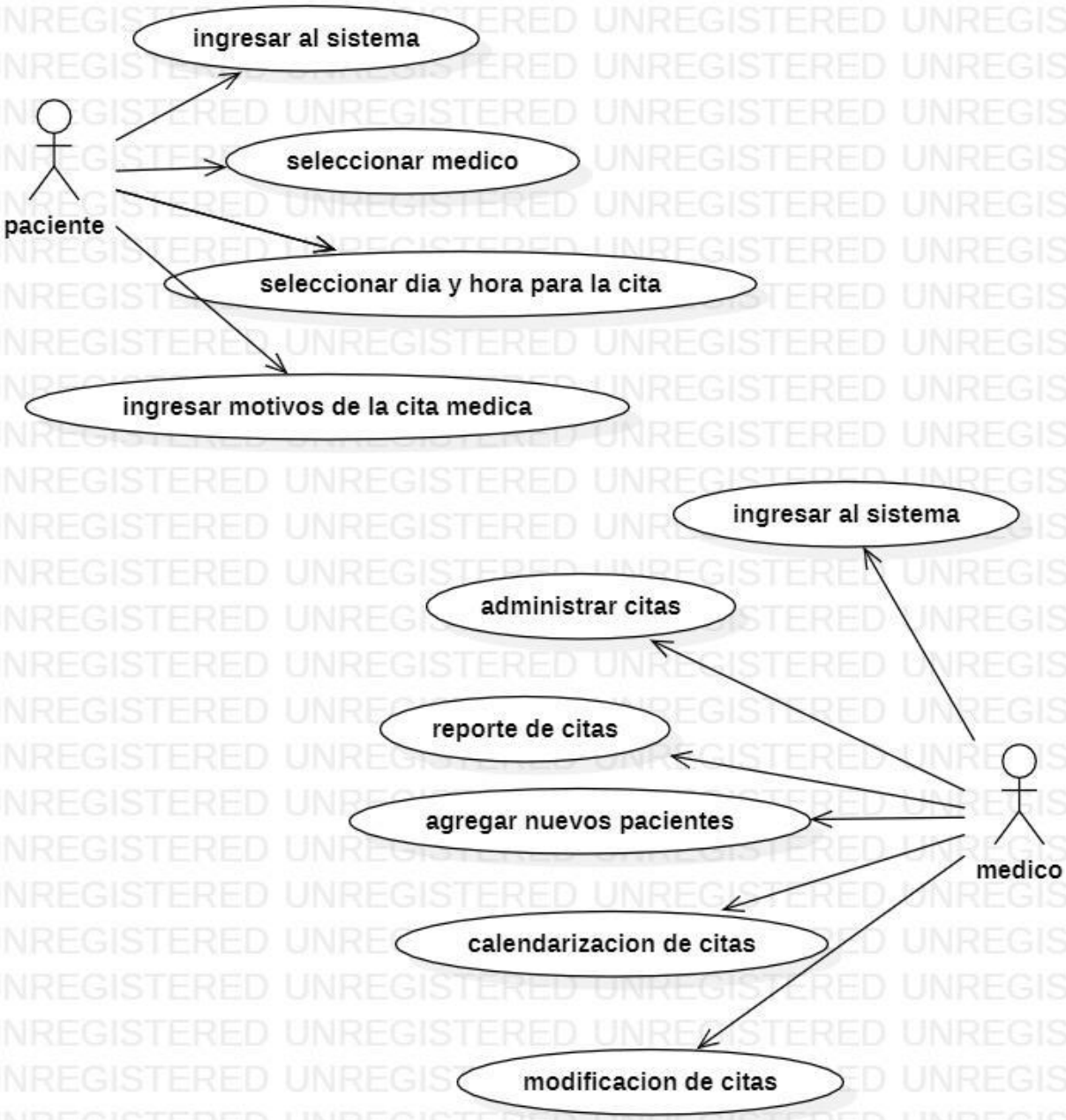
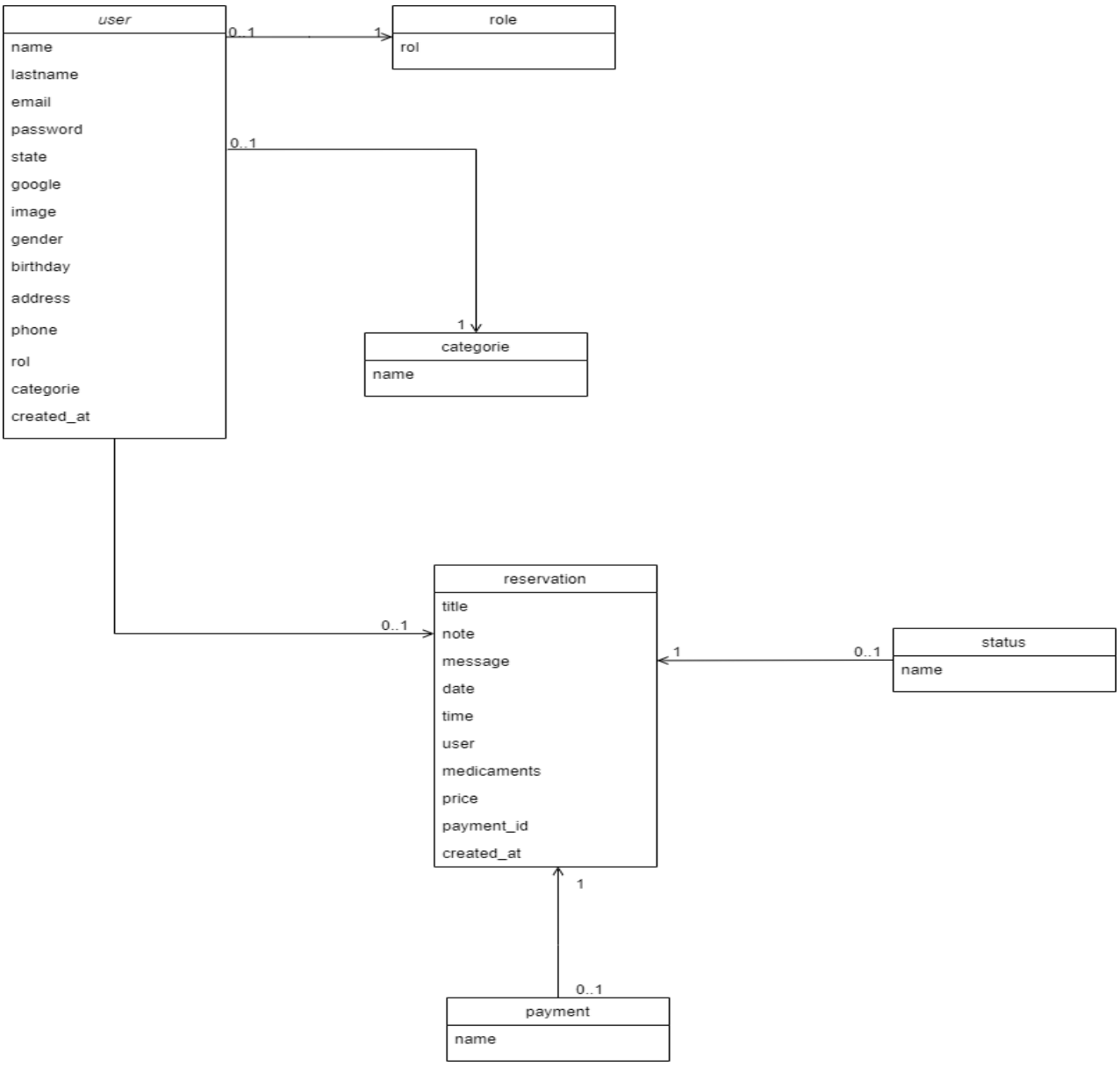


Diagrama de clases.

En este diagrama se visualizan las relaciones de clases, que existen en la api, así como el tipo de roles, categorías, y el registro de información de datos de cada entidad.



DISEÑO

El término diseño de API hace referencia al proceso de desarrollo de interfaces de programación de aplicaciones (API)que expone las funcionalidades de las aplicaciones y los datos para que las utilicen los usuarios y los desarrolladores. Las API son importantes para las empresas modernas, ya que permiten incorporar capacidades nuevas que abarcan desde las operaciones y los productos hasta las estrategias de asociación.

Un programa de API eficaz debe basarse en la estrategia corporativa general de la empresa y contribuir a sus objetivos. Sabrá que tiene los elementos que se necesitan para una buena estrategia cuando pueda responder las siguientes tres preguntas de forma clara:

¿Por qué queremos implementar las API?

¿Cuáles son los resultados concretos a los que queremos llegar con estas API?

¿Cómo pensamos ejecutar el programa de API para lograrlo?

Por lo general, las personas no suelen interpretar este aspecto de la manera correcta. En primer lugar, en vez de centrarse en el valor de la API, es útil considerar el valor de su efecto. No olvidemos que lo valioso es la actividad principal de la empresa, no necesariamente la API. Esta ofrece beneficios cuando se convierte en un medio para acceder de nuevas formas al valor actual que ofrece una empresa.

Otra idea errónea común es creer que una API solo tiene valor si los usuarios están dispuestos a pagar por ella. Esta creencia sólo es correcta en los casos en que la API es un producto en sí, pero no es lo que ocurre en la mayoría de los modelos. Por lo general, las API impulsan alguna otra métrica, como las ventas, la derivación de afiliados, el conocimiento de la marca, etc. El valor de la API para los usuarios es el resultado de la llamada a la API (la solicitud del servicio y su respuesta), en lugar de la llamada en sí.

La segunda pregunta debe ser: ¿cuáles son los resultados concretos a los que queremos llegar con esta API? Es decir, "¿qué hacen las API realmente y cómo influyen en la estrategia comercial general?"

Tanto el concepto de enfoque interno como el de enfoque externo de una empresa permiten definir el objetivo de la API. El primero hace referencia a los recursos específicos y valiosos

de una empresa. Cuanto más valiosos y exclusivos sean los servicios y los recursos que ofrece, más conveniente será la adopción de un programa de API.

La última pregunta, "¿cómo diseñar el programa de API para alcanzar nuestros objetivos?", se trata de la implementación y la ejecución.

Lo que debe preguntarse es:

¿Qué tecnología se utiliza para diseñar las API?

¿Cómo se crean?

¿Cómo se mantienen?

¿Cuáles son los recursos disponibles?

Los desarrolladores de API se relacionan más estrechamente con uno de productos. Ya sea que tenga clientes internos o externos, usted está a cargo de diseñar, implementar, gestionar y optimizar la infraestructura de la que dependen otras personas.

A continuación, se mencionan algunas claves para una buena API:

Prestar un servicio valioso.

Tener un plan y un modelo comercial.

Permitir que sea simple, flexible y fácil de adoptar.

Permitir su gestión y medición.

Garantizar el envío seguro de los archivos y, a su vez, reducir la cantidad de ancho de banda de transferencia.

Gestionar grandes cantidades de datos e intentar relacionarlos de inmediato.

La API es un sistema para llevar el control de citas médicas, pacientes, médicos, historial de citas, áreas médicas y mucho más, pensado para centros médicos, clínicas y médicos independientes.

Es la herramienta ideal para saber los ingresos que se generan en consultorios o clínicas, con la inclusión de reportes y estado de pagos puedes saber las citas o consultas que ya pagaron y así saber el ingreso utilizando un rango de fecha.

Características de la API.

- Vista de Calendario.
- Gestión de citas.
- Gestión de Médicos.
- Gestión de Pacientes.
- Gestión de Usuarios con Acceso al Sistema.
- Manejo del estado de las citas: Pendiente, Aplicada, No asistió, Cancelada.
- Manejo de estado de pago: Pendiente, Pagado, Anulado.
- Historial de citas por Paciente.
- Historial de citas por Médico.
- Buscador avanzado por: Palabra clave, médico, paciente y fecha.
- Reportes de citas.

Vista de Calendario

Puedes ver en un calendario todos tus eventos organizados, usamos el plugin full calendar.

Nueva Cita

Para crear una cita es necesario un asunto, seleccionar un paciente, médico, fecha, hora, una nota opcional, enfermedad del paciente, síntomas presentados, medicamentos o tratamiento recetados.

Ver citas

Podemos ver todas las citas a partir del día de hoy, también existe una página llamada Citas Anteriores donde es posible ver todas las citas a partir del día anterior hacia atrás.

Pacientes

Organiza tu directorio de pacientes, escribe nombre, apellidos, género, fecha de nacimiento, dirección, correo electrónico, teléfono, enfermedad, alergia y/o medicamentos.

Médicos

Organiza tu directorio de pacientes, escribe nombre, apellidos, dirección, correo electrónico y teléfono.

Historiales

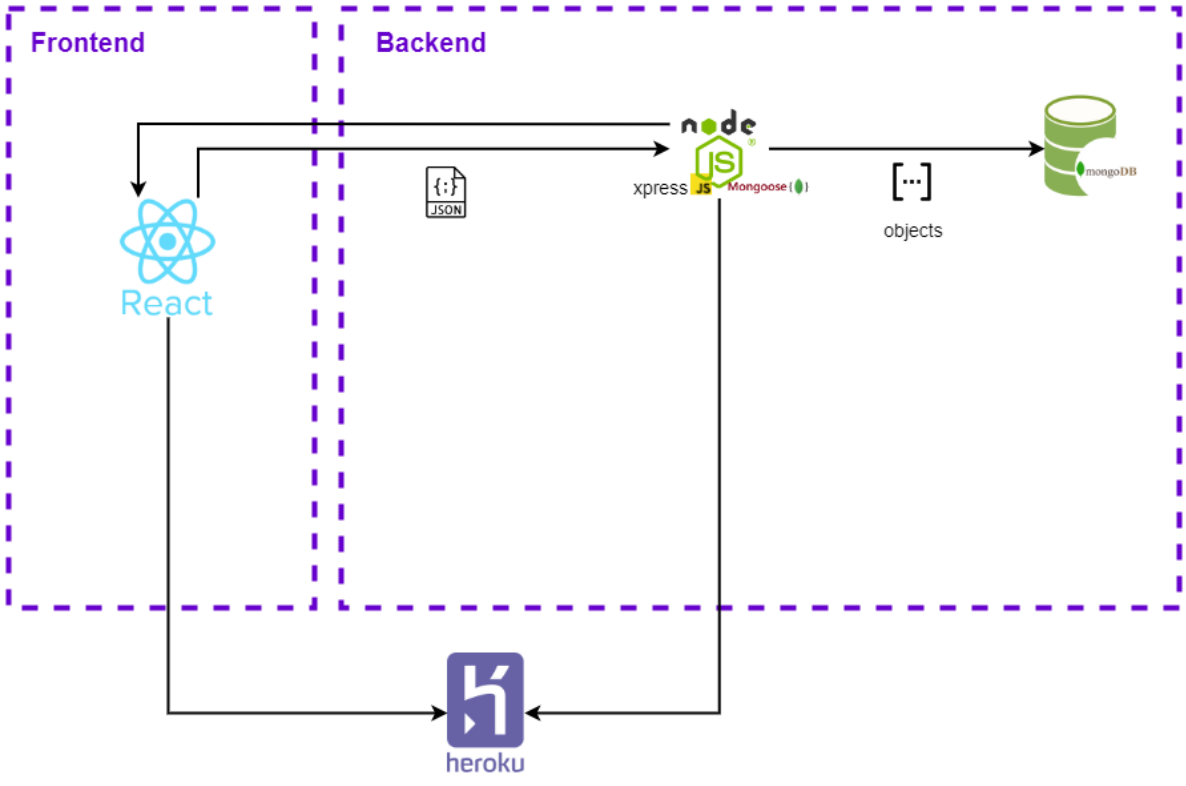
Puedes acceder al historial de citas de un médico o de un paciente.

Reportes

Genera reportes por médicos, pacientes, rango de fechas, estado de la cita, estado del pago, observa costos totales.

Dentro de la API se utilizan los siguientes aplicativos, tanto para diseño visual, como para el desarrollo de programación.

Diagrama de arquitectura de software.



DESARROLLO

HERRAMIENTAS DE DESARROLLO UTILIZADAS

Visual studio code:

Es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.

Características

- Personalización de la barra de herramientas.
- Mejor desplazamiento automático del editor.
- Extensiones ver actualizaciones.
- Resultados de búsqueda en una vista de árbol.
- Compatibilidad con repositorios de Git anidados.
- Terminal Quick Fixes - Sugerencias para corregir errores tipográficos de comandos y configurar un control remoto ascendente.
- Anclar tareas de uso frecuente.
- Validación de enlaces Markdown.
- Autenticación del Servidor de GitHub Enterprise.
- Características de los contenedores de desarrollo.
- Discusiones de la comunidad de VS Code.

LENGUAJE DE PROGRAMACIÓN Y GESTORES DE BASES DE DATOS U OTRAS HERRAMIENTAS UTILIZADAS EN EL DESARROLLO DEL PROYECTO.

Node.js

la versión que se utilizó en el proyecto es node: **v16.17.0**

Es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación JavaScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.

Características

- Interoperabilidad mejorada de la Web Crypto API
- Temporizadores estables promete API

- El motor JavaScript V8 se actualiza a V8 9.0
- Actualizaciones de la cadena de herramientas y el compilador

Javascript

La versión que se utilizó en el proyecto es Javascript **ES6**

Es un lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo (just-in-time) con funciones de primera clase. Si bien es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, y es usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB y Adobe Acrobat JavaScript es un lenguaje de programación basada en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa (por ejemplo, programación funcional). Lee más en acerca de JavaScript.

Características

- Lenguaje del lado del cliente
- Lenguaje orientado a objetos
- De tipado débil o no tipado
- De alto nivel
- Lenguaje interpretado

MongoDB

Es un sistema de base de datos NoSQL, orientado a documentos y de código abierto. En lugar de guardar los datos en tablas, tal y como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Características

- Potente sintaxis de consultas
- Indexación
- Soporte para SQL
- Transacciones
- Base de datos distribuida con gran escalabilidad vertical y horizontal
- Permite ejecutar consultas pasando directamente código JavaScript

Vue.js

Este es un framework de JavaScript que permite construir interfaces de usuarios. Esta es una capa que permite el diseño de las vistas de las aplicaciones que se realicen con ellas. A través de él se trabaja la parte frontend de las aplicaciones y permite utilizar e integrar otras librerías o proyectos. Vue.js permite la creación de aplicaciones de forma rápida, práctica y sencilla, su versión actual es la **3.0 One Piece**.

Características

Entre sus características se pueden destacar las siguientes:

- Es un framework con una curva de aprendizaje muy sencilla, ya que posee la documentación necesaria de todos los componentes que posee.
- Utiliza HTML, CSS y JavaScript.
- Es un framework progresivo. Puede ser utilizado para migrar y adaptar proyectos existentes a Vue.
- Tiene un enfoque tradicional centrado en HTML.
- Utiliza componentes. Estos componentes, que extienden los elementos básicos de HTML son personalizados para que el compilador de Vue pueda agregar un comportamiento.
- Posee reactividad, que le permite utilizar objetos de JavaScript simples a una retransmisión optimizada y personalizada.

CONCLUSIONES

El desarrollo de aplicaciones utilizando JavaScript y otras herramientas permite desarrollar APIS que sean de mucha utilidad y que puedan cumplir con las especificaciones que puedan solicitar los usuarios. El desarrollo de aplicaciones permite brindar a los usuarios aplicaciones que puedan ser utilizadas para optimizar y automatizar sus acciones.

El desarrollo de la aplicación de citas médicas desarrollada con node.js, JavaScript y Vue permite a sus usuarios llevar un control de las citas que realicen dentro de sus organizaciones, ya que le permite llevar un control detallado y visual de las agendas médicas que se manejen. Les permite a los usuarios registrarse de acuerdo al cargo que utilicen, así como el agregar las citas estableciendo fecha y hora. Presenta una interfaz amigable y sencilla que los usuarios pueda utilizar para el manejo de cada uno de los bloques establecidos.

El desarrollo de la api de citas medica permite aplicar las diferentes herramientas de manera que brinden un servicio eficiente para sus usuarios, dejando evidenciado que utilizando node.js se pueden realizar las directrices necesarias para el funcionamiento de la aplicación web, así como la unificación de Vue.js como herramienta para el desarrollo y diseño de las vistas.

RECOMENDACIONES

Para el desarrollo de la aplicación web descrita en el presente documento, se deben tomar a consideración todos los aspectos y características que se han utilizado para que el proyecto pueda funcionar de acuerdo a lo establecido, por esto, es necesario lo siguiente:

- Instalar todas las dependencias utilizadas para el funcionamiento del proyecto.
- Apoyarse de la documentación de node.js y Vue.js para poder desarrollar cada parte del proyecto.
- Utilizar mongo atlas como gestor de base de datos.
- Se pueden incorporar otras secciones que se consideren necesarias y que puedan adaptarse a las necesidades que se deseen cubrir. Pueden agregarse nuevas secciones de acuerdo al área que se aplique.

REFERENCIAS BIBLIOGRÁFICAS

Alvarez, M. A. (20 de Julio de 2020). *Qué es MVC*. Obtenido de desarrolloweb.com: https://desarrolloweb.com/articulos/que-es-mvc.html

developer.mozilla. (24 de Octubre de 2022). *JavaScript*. Obtenido de developer.mozilla.org: https://developer.mozilla.org/es/docs/Web/JavaScript#:~:text=JavaScript%20(JS)%20es%20un%20lenguaje,con%20funciones%20de%20primera%20clase

Intelequia. (28 de Noviembre de 2020). *Ciclo de vida del software*. Obtenido de intelequia.com: https://intelequia.com/blog/post/2083/ciclo-de-vida-del-software-todo-lo-que-necesitas-saber

ITDO. (27 de Julio de 2021). *¿Qué es Node.js, y para qué sirve?* Obtenido de itdo.com: https://www.itdo.com/blog/que-es-node-js-y-para-que-sirve/

Manz.dev. (s.f.). *¿Qué es Vue?* Obtenido de lenguajejs.com: https://lenguajejs.com/vuejs/introduccion/que-es-vue/

Node.js. Descarga de node.js. Obtenido de: https://nodejs.org/es/download/

Mongo Atlas. Cuenta. Obtenido de: https://www.mongodb.com/cloud/atlas/register

ANEXOS

MANUAL DEL DESARROLLADOR

Introducción

El presente manual tiene la finalidad de presentar todas las acciones que son requeridas para poder iniciar el desarrollo del proyecto de la api de citas médicas, permitiendo el desarrollo y función de cada una de sus acciones establecidas. Se presentan los procesos de instalación y configuración de todas las herramientas que se utilizaron, node.js, mongo atlas, mongo compass y de igual manera, la estructura principal del proyecto, el modelo MVC aplicado, mostrando la división de carpetas y todos aquellos archivos esenciales para la puesta en marcha de la api, tal como la conexión a la base de datos, las configuraciones, dependencia utilizadas, entre otros.

Objetivos

Objetivo General

Presentar al usuario una serie de pasos que muestran las configuraciones y acciones a realizar para utilizar la API de citas médicas, permitiendo realizarlo de una forma sencilla.

Objetivos Específicos

Mostrar todas las herramientas utilizadas.

Explicar la conexión realizada al gestor de base de datos.

Mostrar la estructura general del proyecto.

Desarrollo

1. Configuración de entorno de trabajo.

A continuación, se detalla todas las instalaciones necesarias de las herramientas de trabajo para poder realizar el proyecto y ejecutarlo correctamente.

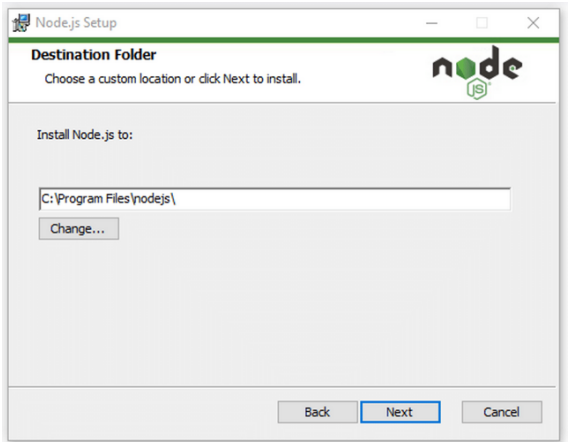
1.1 Instalación y configuración de node js.

Es necesario descargar el instalador de node.js, el cual puede encontrarse en la página oficial de Node. Link de Descarga: https://nodejs.org/es/download/ y se debe seleccionar la versión de acuerdo a la plataforma a utilizar.

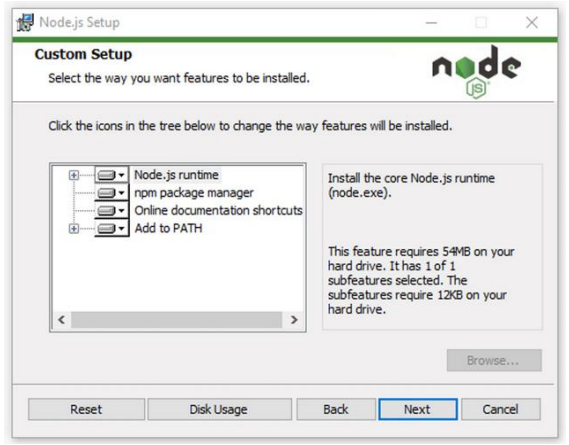
Una vez descargado el instalador, procedemos a ejecutarlo e iniciar la instalación de node. Para el caso de Windows es importante descargar el instalador MSI ya que lleva una colección de archivos de instalación esenciales para instalar, actualizar o modificar la versión existente. Una vez descargado en el ordenador, se hace clic para continuar con el proceso.

Se inicia el proceso de instalación dando clic en el archivo y se abrirá la primera pantalla aceptando el acuerdo de licencia y presionando next.

Luego debe seleccionarse la ruta de destino de node.js; de no cambiarse la ruta se deja la que trae predeterminada y se da en **next.**

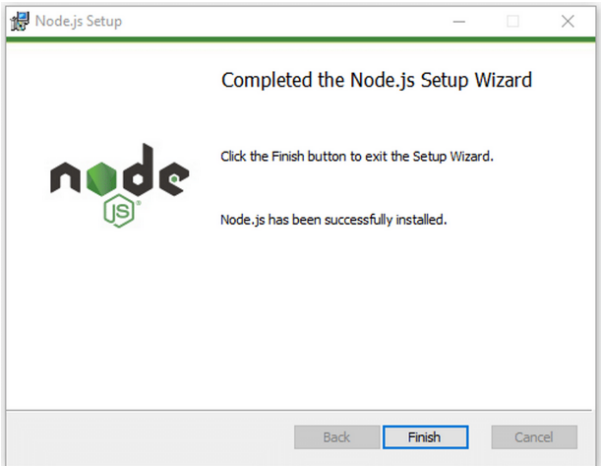


Luego se muestran todas las opciones de configuración. Para obtener la configuración estándar se da clic en el botón **next.**



Como acción opcional se pueden descargar herramientas para módulos nativos. Se puede seleccionar la casilla y se pulsa en el botón **next** y se espera a que termine toda la descarga.

Por último, se presiona el botón install. El sistema completará la instalación en unos segundos o minutos y mostrará un mensaje de éxito. Haz clic en el botón **Finalizar** para cerrar el instalador de Node.js.



Para verificar que la instalación se realizó correctamente se puede comprobar la versión instalada con el comando **node –version.** y para la versión de npm se utiliza el comando **npm –version**.

De esta forma se realiza la instalación de node.js y npm en Windows.

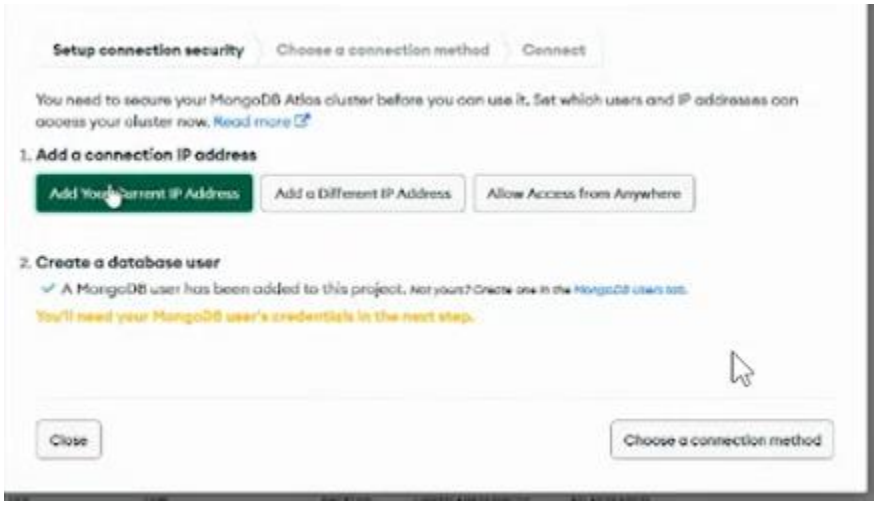
1.2 Instalación y configuración de mongo atlas.

Para trabajar con mongo atlas es necesario la creación de una cuenta en su página oficial. Para esto, se debe ingresar a la página oficial. Link de la página: https://account.mongodb.com/account/login?_ga=2.235234369.1742037960.1666841855-1488743830.1666841855 . Seleccionamos la opción de entrar con una cuenta de google y posteriormente se abrirá una pantalla para verificar la cuenta que se desee utilizar.

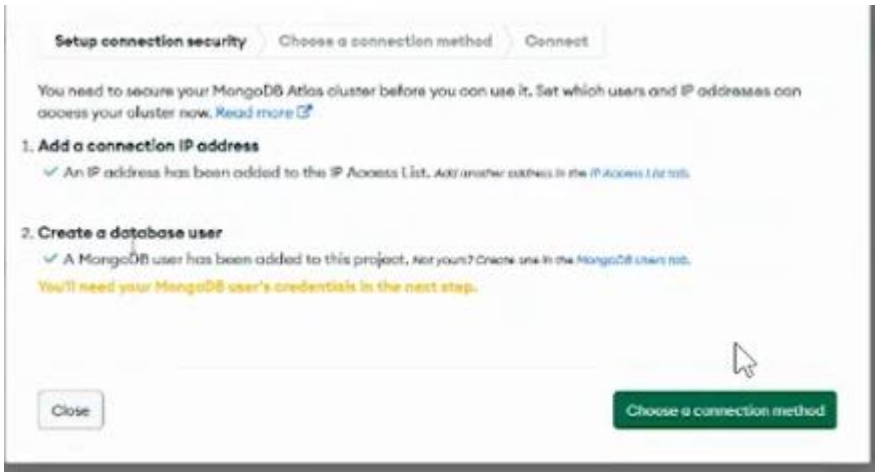
Aceptamos los términos y condiciones seleccionando la casilla y damos clic en **submit**. Nos muestra una pantalla de bienvenida y posteriormente configuramos las opciones de mongo atlas. Se deben responder las preguntas y el lenguaje de preferencia, en este caso javascript y continuamos dando clic en finish. Luego seleccionamos la opción gratis para continuar.

Luego configuramos y creamos un cluster compartido, dejando las opciones predeterminadas realizando cambios únicamente en el Cluster name, donde se puede agregar el nombre de su preferencia. Y seleccionamos el botón **CREATE CLUSTER.**

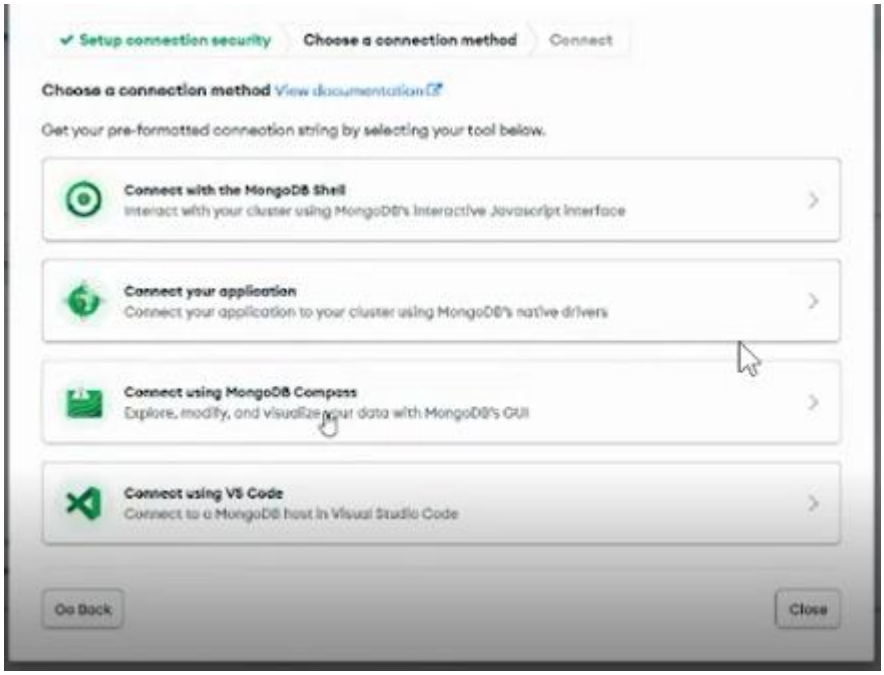
Se debe continuar con la seguridad, agregando un usuario y contraseña (estos nos permitirán conectarnos a mongo atlas) seleccionando el botón **CREATE USER**. Luego, en la barra lateral izquierda nos vamos a la parte de **DATABASE** y presionamos el botón **CONNECT**. Aquí agregamos la opción para conectarnos por una dirección ip, seleccionando el primer botón.



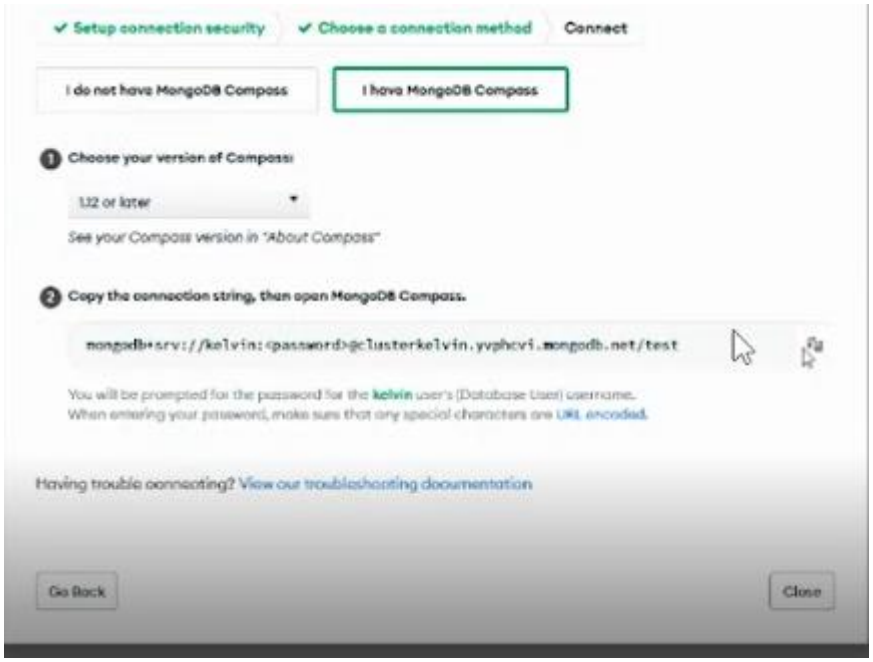
Agregamos la ip pública del ordenador u otra de su preferencia.



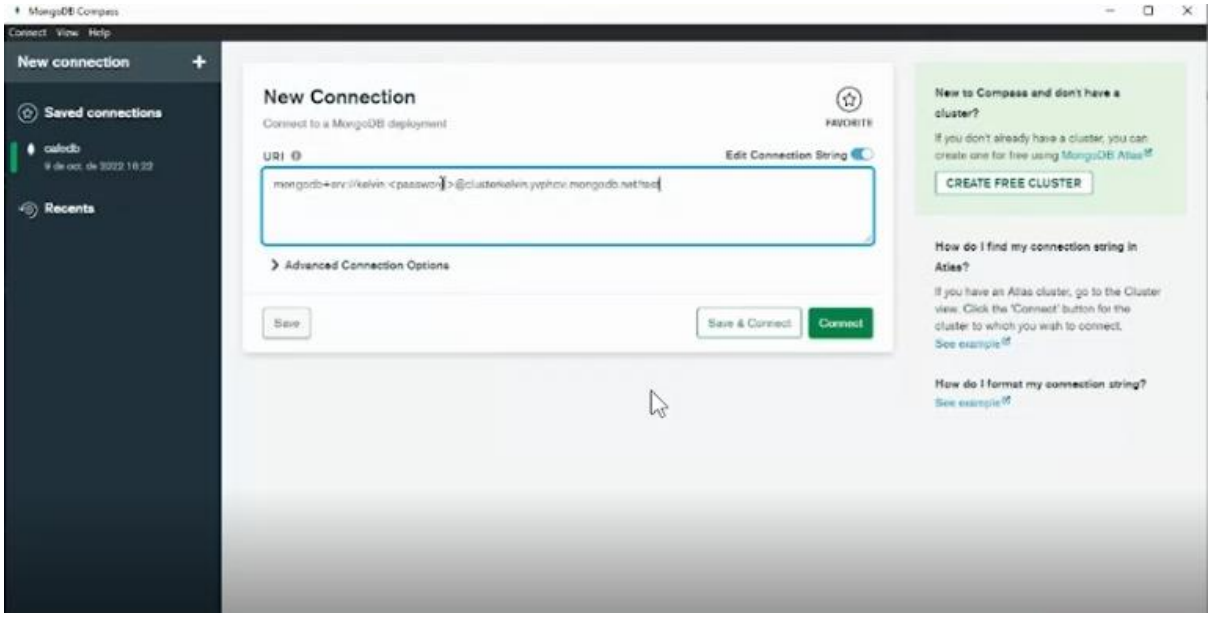
Luego de esto podemos conectarnos a través de mongodb compass, seleccionando esa opción.



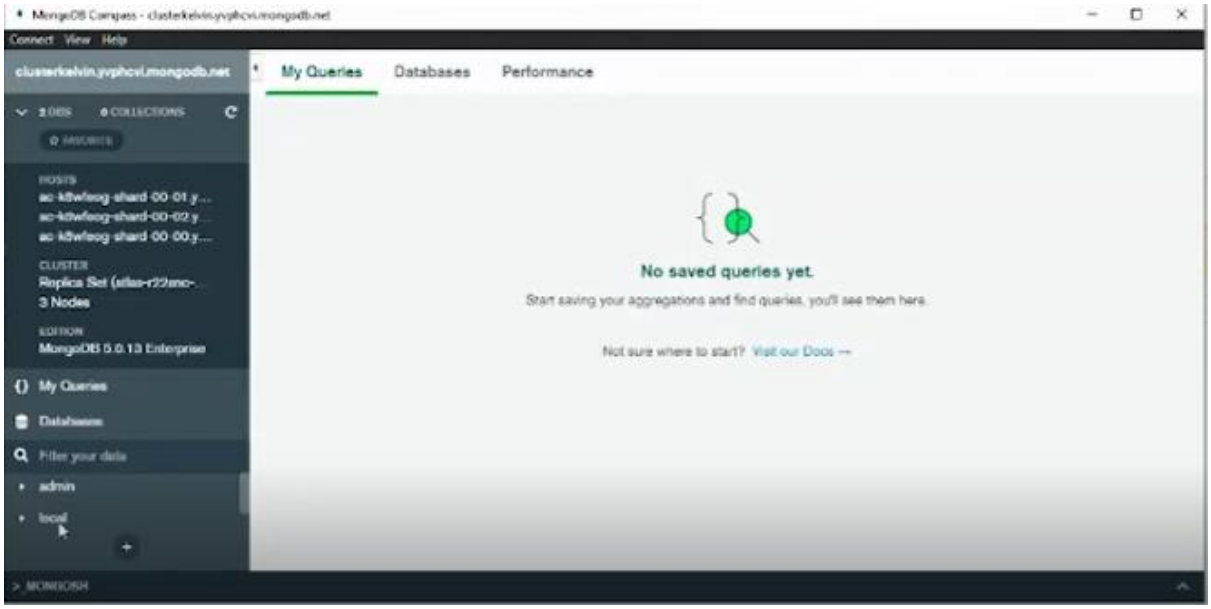
La siguiente pantalla nos muestra la opción para descargar e instalar mongo compass, lo cual se explica en el punto 1.3 Instalación y configuración de mongo compass. En la siguiente pantalla copiamos el link de conexión que aparece ya que servirá para vincularse en compass.



Nos dirigimos a mongo compass, seleccionamos la opción de nueva conexión y agregamos la URL copiada, realizando los cambios en la parte de la contraseña, que debe reemplazarse por la que se creó en los pasos anteriores.



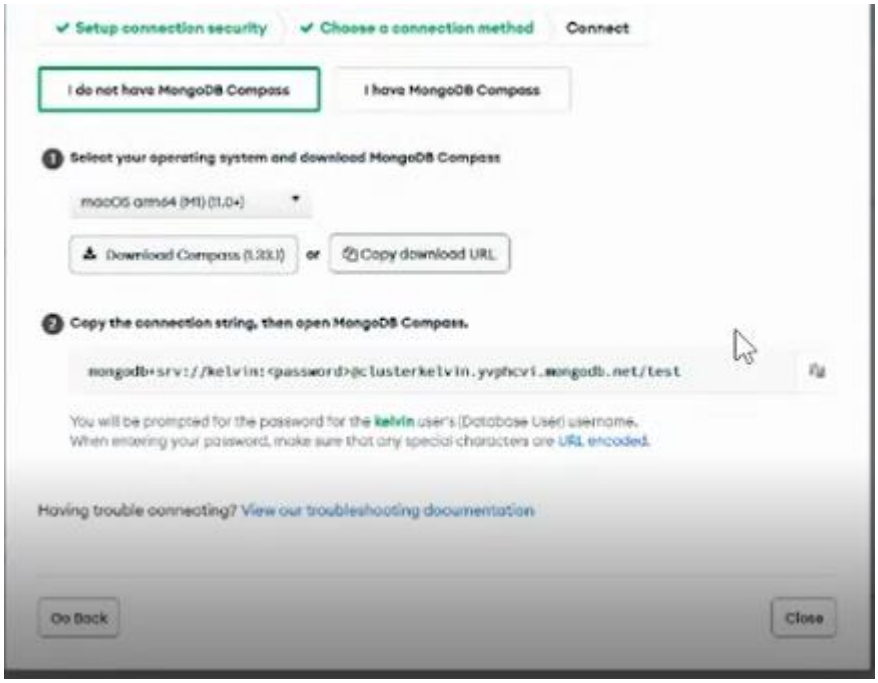
Seleccionamos el botón **CONNECT** y se conectará a mongo atlas automáticamente.



De esta forma, se ha vinculado el uso de mongo atlas y mongo compass como gestores para la base de datos.

1.3 Instalación y configuración de mongo compass.

Ya que se encuentra vinculado con mongo atlas, la pantalla nos muestra la opción para descargar compass y poder conectar mongodb atlas.



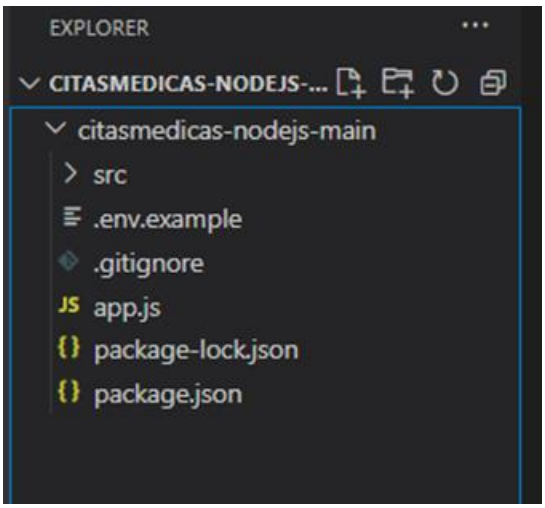
Descargamos la versión que aparece, iniciamos el instalador y seguimos los pasos que se muestran en la pantalla. Aceptar los términos de la licencia y pulsar en next, realizando los procesos que se pidan.

2. Configuración de estructura del proyecto.

Ya que se utilizó el modelo MVC (modelo - vista - controlador) se crearon las respectivas carpetas, tal como se muestran en la imagen. Las cuales definen la estructura del proyecto.

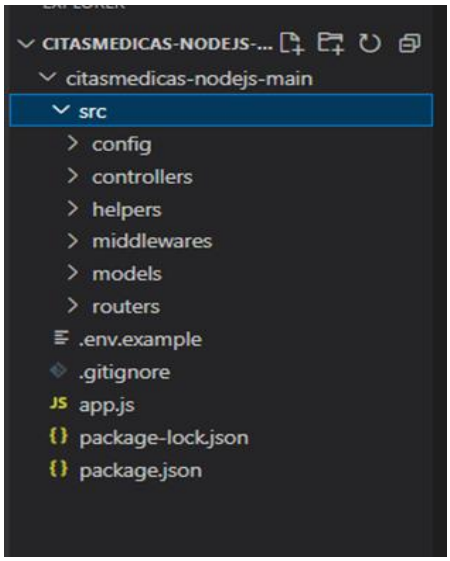
De igual forma, se encuentra la estructura general del sistema API, mostrando las carpetas principales, subcarpetas y archivos de importancia.

Estructura de archivos.



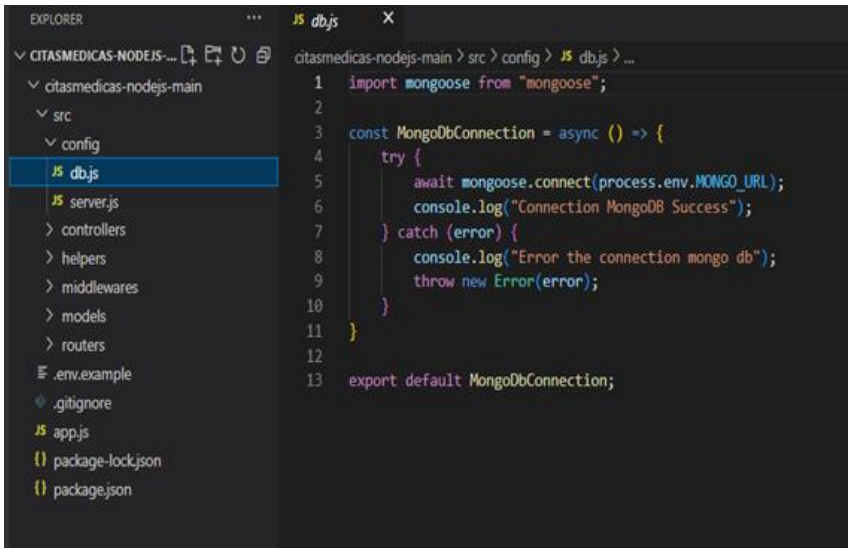
Distribución de carpetas de citas médicas, tenemos la carpeta src archivos. env, archivo app,js, archivos package-lock.json, archivo package.json.

Además también contiene los complementos necesarios de node JS, para que la aplicación funcione correctamente.



Dentro de la carpeta SRC encontrará otras subcarpetas, las cuales contienen los archivos de programador, en los cuales se le dará detalle a detalle cómo funciona cada archivo, y cuál es la lógica de cada sentencia.

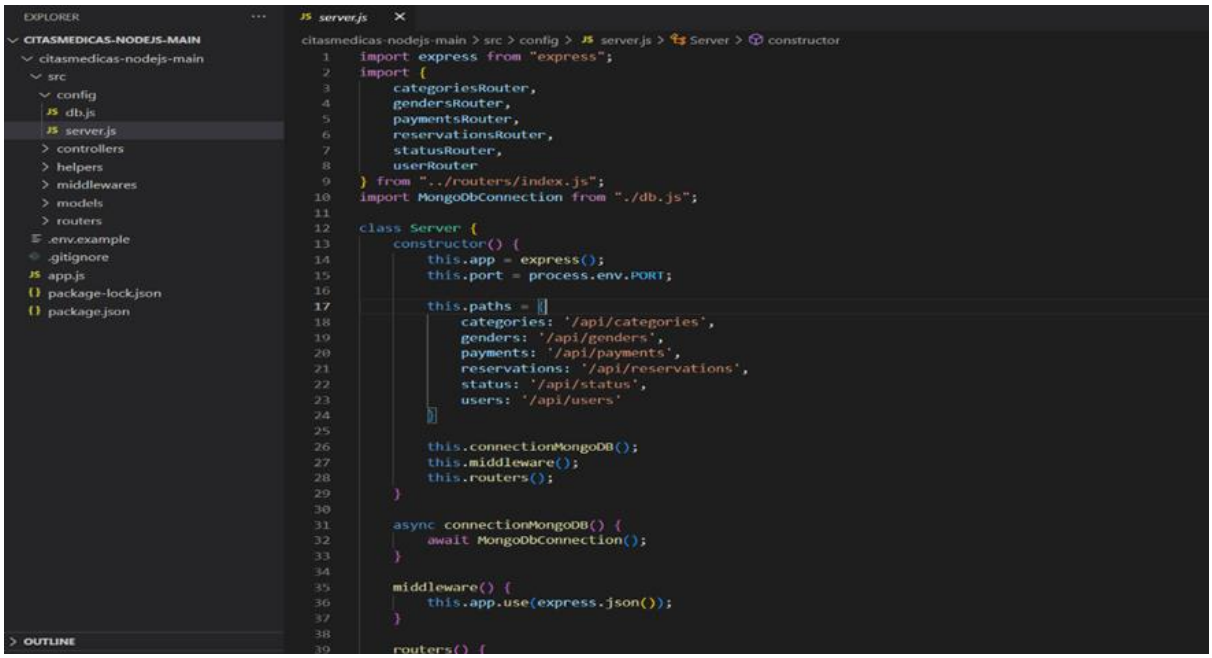
Carpeta config.



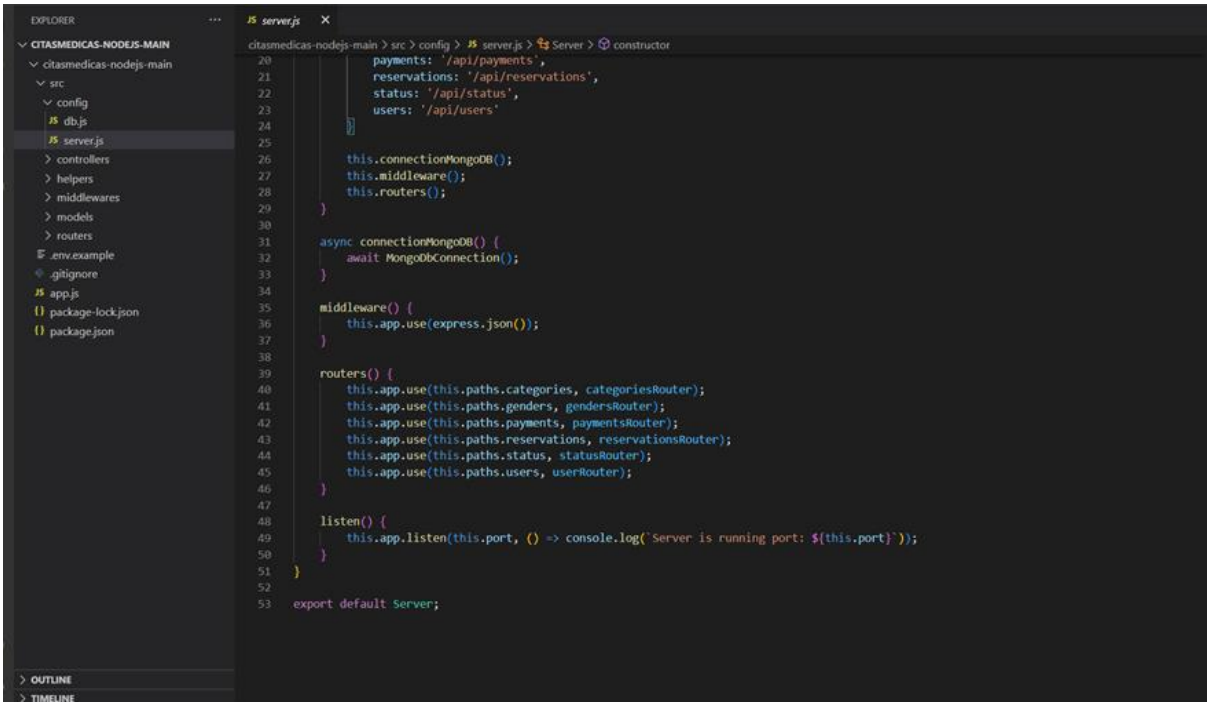
Dentro de la carpeta config, se encuentran los archivos db.js en el cual está la configuración de la conexión a la base de datos.

Como se mencionó el archivo db.js permite la conexión a mongodb.

También se encuentra el archivo server.js, el cual contiene las listas de las rutas de los diferentes archivos de configuración, de las estructuras del crud MVC. Así como los middlewares y las importaciones necesarias.



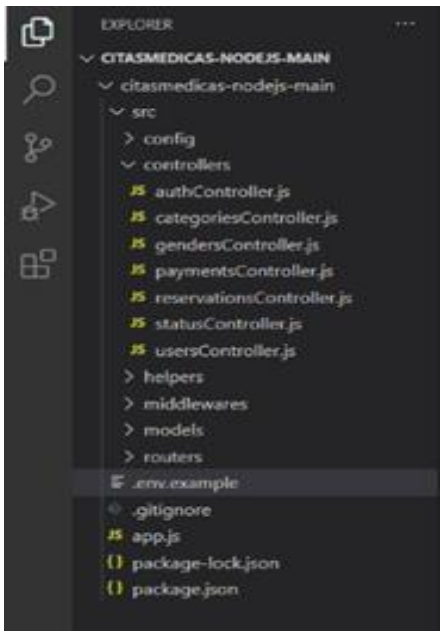
Vista del código del archivo server.js. 1.0.



Vista del código del archivo server.js. 1.1.

Carpeta controllers

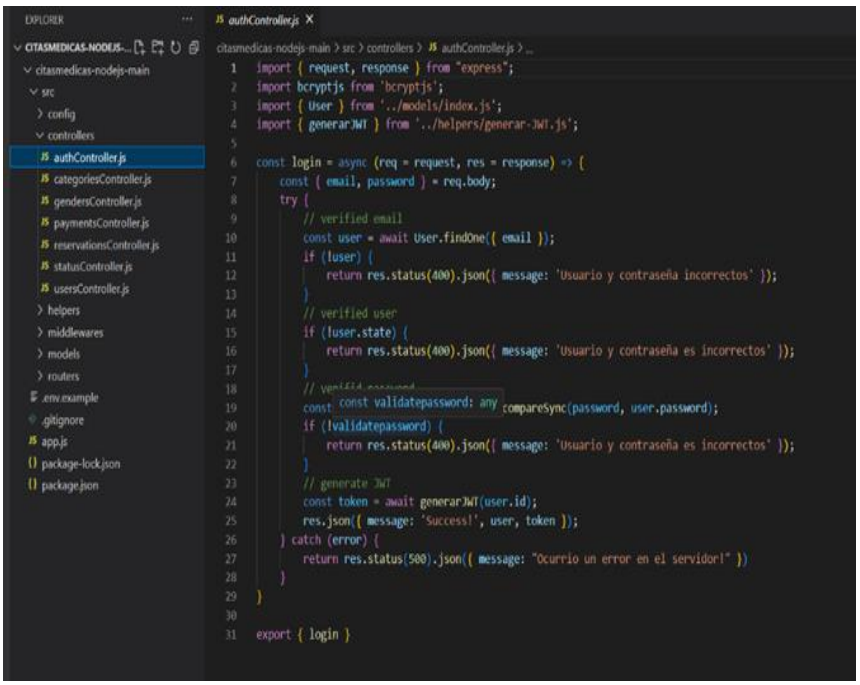
Los archivos contenidos en esta carpeta son los que se muestran en la siguiente imagen.



Archivos contenidos en la carpeta Controller.

Se presenta una breve explicación de algunos archivos dentro de esta carpeta.

Archivo authController.js

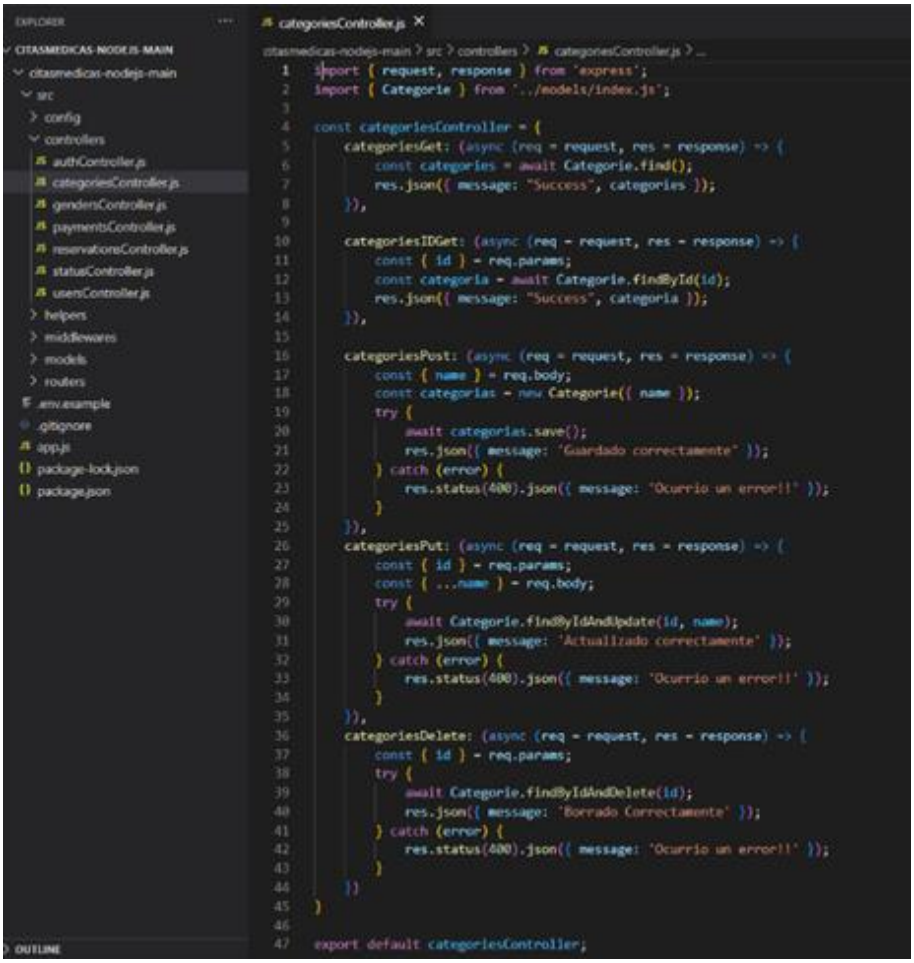


Contiene los parámetros de validación de seguridad de logueo, en el cual se compara los datos ingresados por el usuario, con los datos almacenados en el Sistema, y se le devuelve un mensaje de confirmación o de notificación de que algo ingresado está mal, o existen problemas con el servidor.

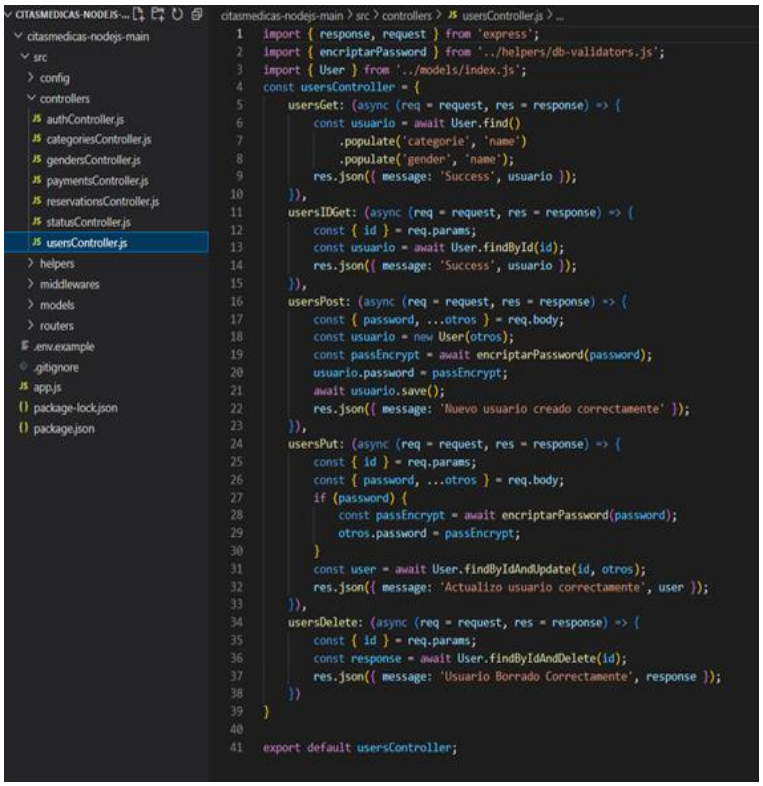
Archivo categoriesController.js

En este archivo se definen los métodos de insertar, actualizar, buscar por id, borrar, de la función categorías.

Estos comandos son repetitivos hasta el archive statusController.js.



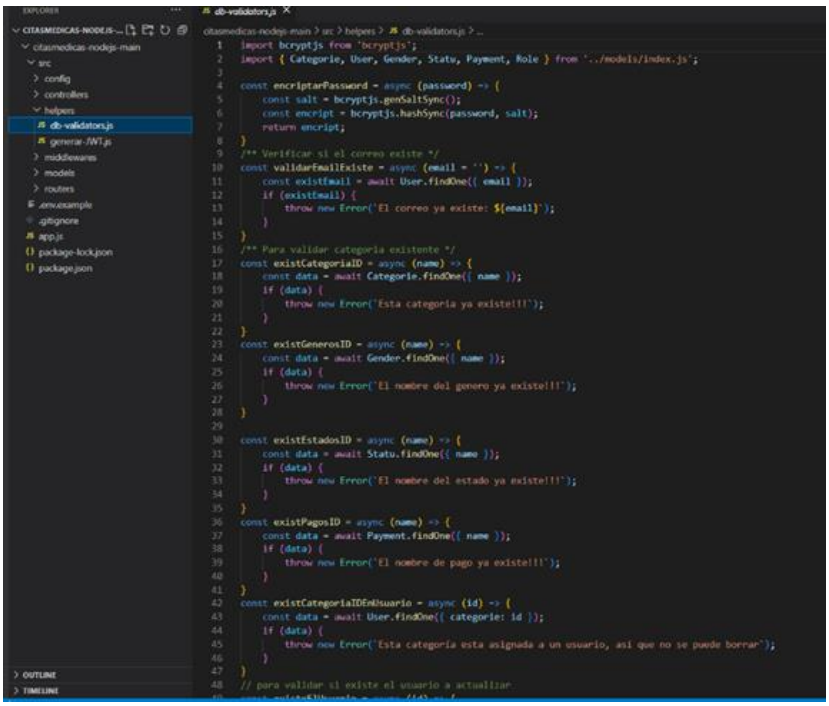
Contenido de archivo usersController.js



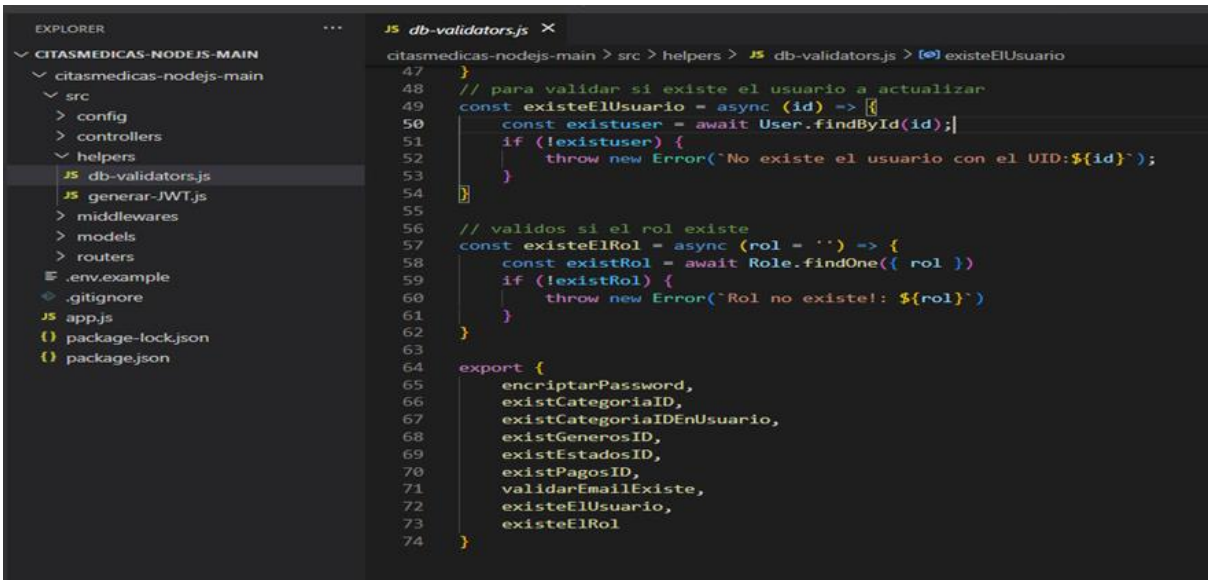
Aquí la única diferencia del resto de archivos es que se aplica una encriptación a la contraseña, que el usuario ingrese, para evitar que ésta se obtenga de manera fácil, y de igual manera se repiten los métodos de consulta.

Carpeta helpers.

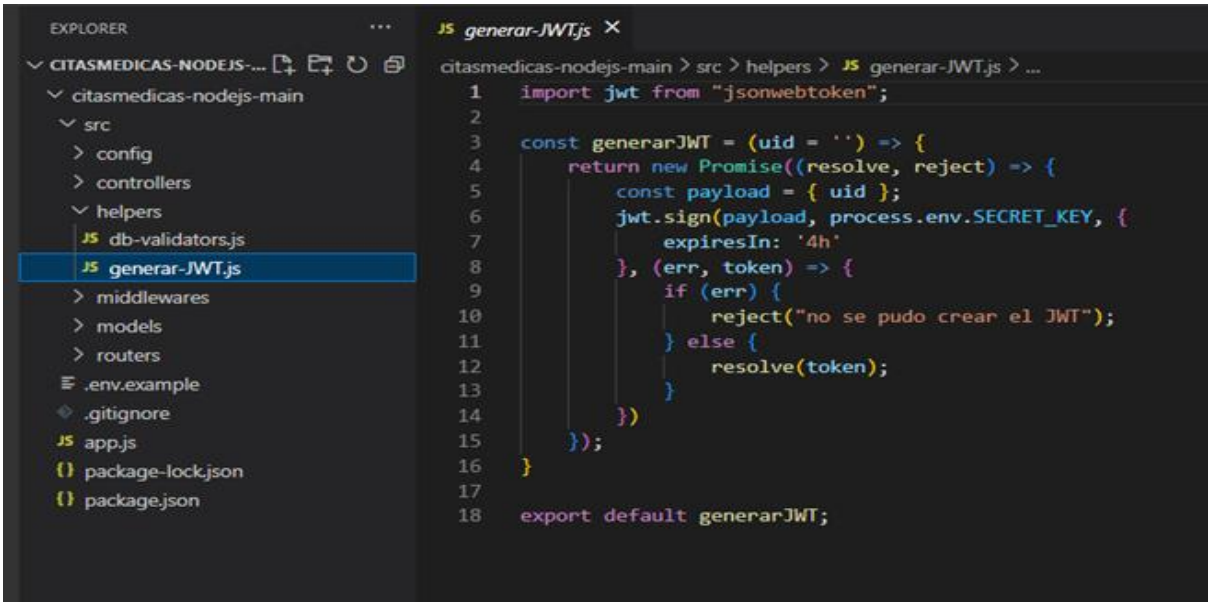
Dentro de la carpeta helper, se encuentran 2 archivos; el primero con nombre **db-validators.js**, el cual contiene los métodos de verificación de datos introducidos por el usuario, y verifica si coinciden con los datos almacenados y que estos no se repitan.



Así mismo en la siguiente imagen, se muestra la exportación de los validadores.

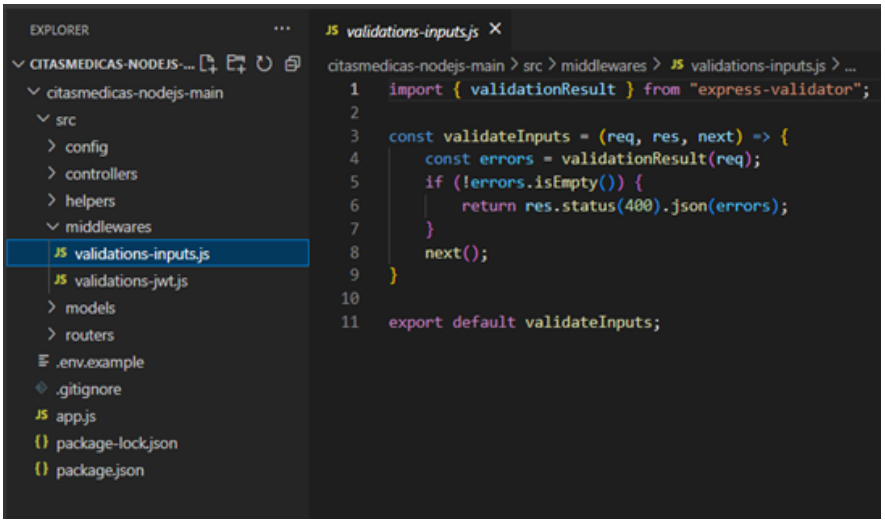


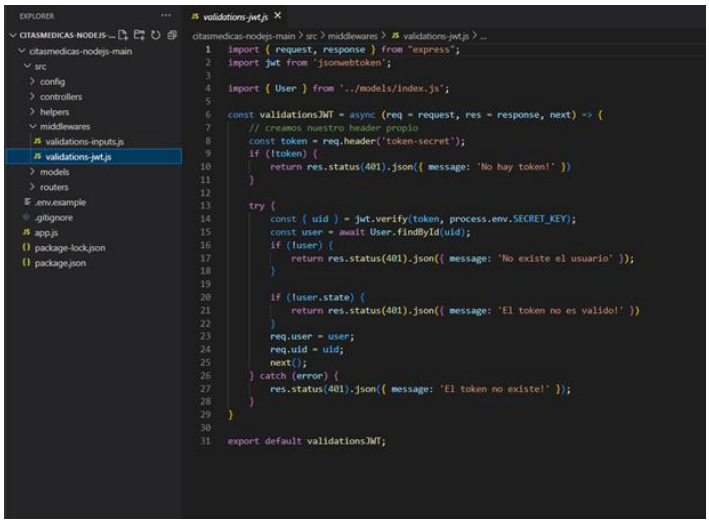
El Segundo archivo contenido la carpeta helper, se llama **generar-JWT.js**, este archivo nos permite generar un token de paga, por un periodo de 4 horas, o de lo contrario, nos mostrará que no se puede crear, cuando este logra crear, el genera un token de pago.



Carpeta middlewares.

Dentro de la carpeta middlewares, se encontrarán 2 archivos, el primer archivo con nombre **validations-inputs.js**, sirve para verificar que no queden campos vacíos al momento de la inserción de datos por parte del usuario.



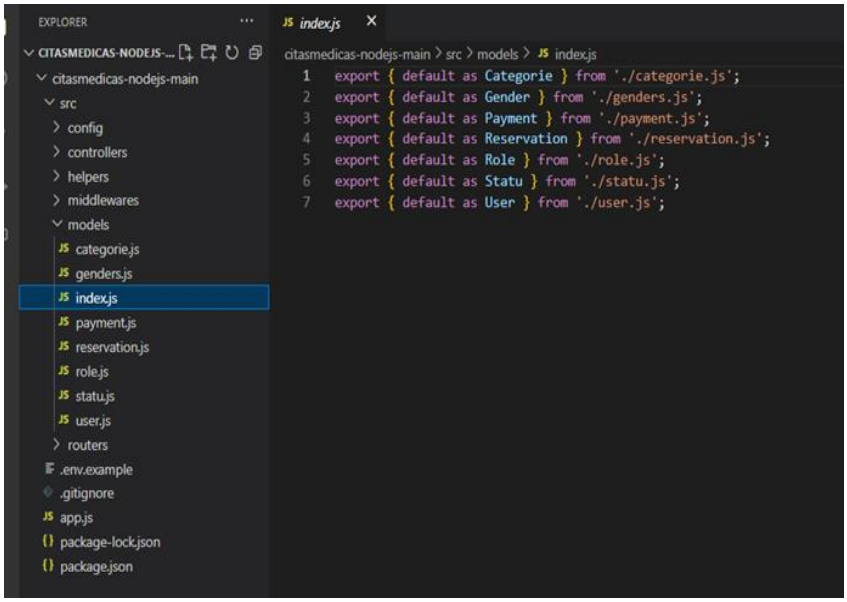


El Segundo archivo con nombre **validations-jwt.js**, este contiene la validación del token que se genera, y verifica si este coincide con el antes creado y que corresponda al usuario que lo creó, si el usuario existe o no, o si el token está vacío.

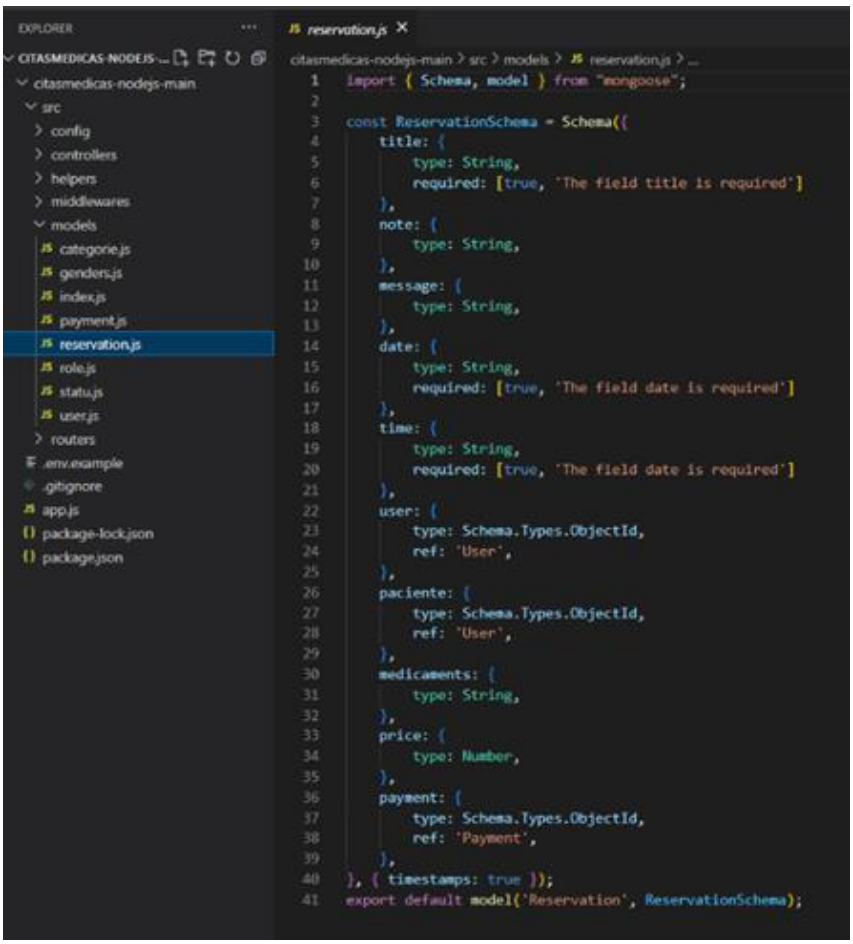
Carpeta models

Dentro de la carpeta modelo, encontraremos 8 archivos.

En todos los archivos exceptuando el archivo index.js, se encuentra el esquema general de cómo se capturan los datos ingresados por el usuario, así como el tipo de dato requerido ya sea, string, int, number, etc. En el archivo index.js, se encuentran las exportaciones de todos los modelos de esquemas, así como la ruta de ubicación, del archivo.

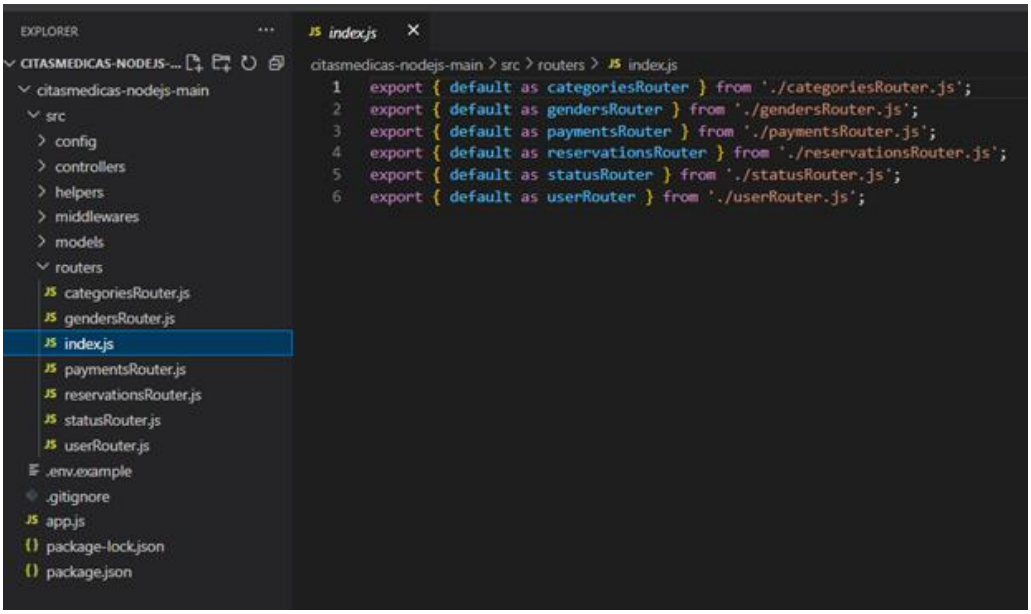


A continuación, se muestra la estructura de esquema del archivo **reservation.js**.

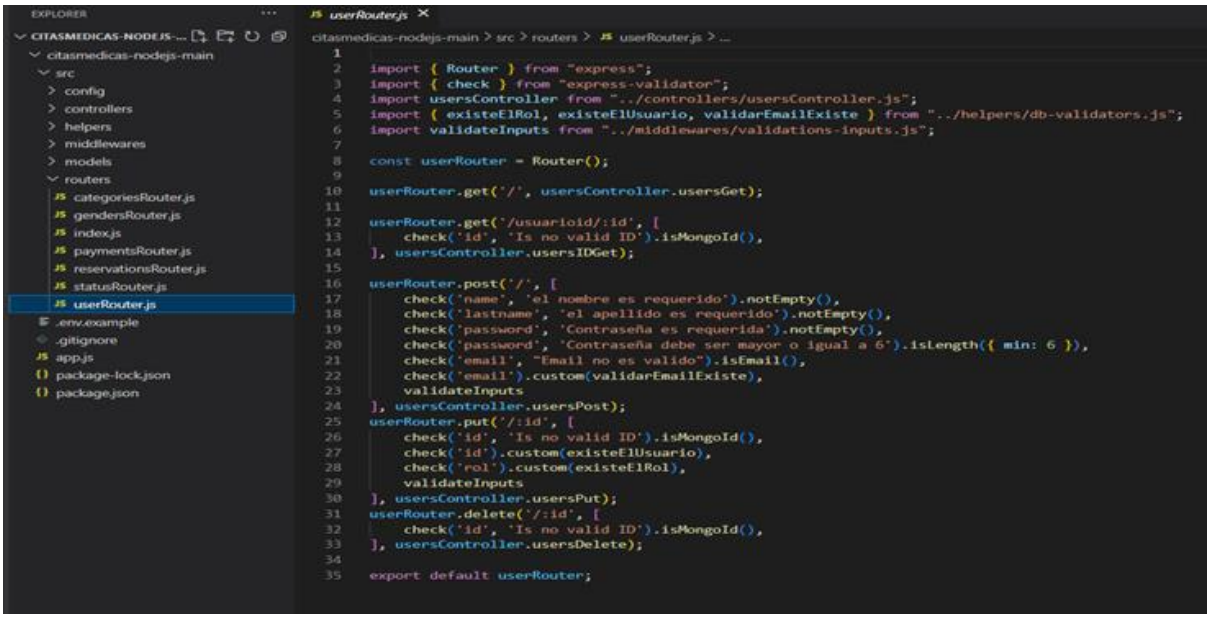


Carpeta routers.

Dentro de esta carpeta se encuentran 7 archivos, en los cuales se encuentra la programación de validación de datos al momento de ser capturados por el Sistema; en esta sección nos aseguramos de que el usuario, rellene la información requerida por el Sistema, y que no falte ningún dato por parte del usuario.



Así mismo, también incluyen la exportación de los archivos complementos para efectuar la validación de información capturada, así como también que esta información se almacene de manera correcta en la base de datos, tal como se muestra en la siguiente imagen:



3. Configuración de estructura de Vue.JS

Para la continuidad del proyecto, se debe configurar y realizar el desarrollo de las vistas de la aplicación, el cual se podrá conectar con la API configurada y poder realizar todas las peticiones establecidas. Para esto, es necesario utilizar las herramientas de Vue.JS y sus respectivos componentes.

3.1 Instalación de VueJS.

Para instalar Vue en su versión 3 debe realizarse los siguientes pasos: debe ubicarse en una terminal.

```
npm install -g @vue/cli
# OR
yarn global add @vue/cli
```

Utilizar el comando escrito solo una vez para realizar la instalación.

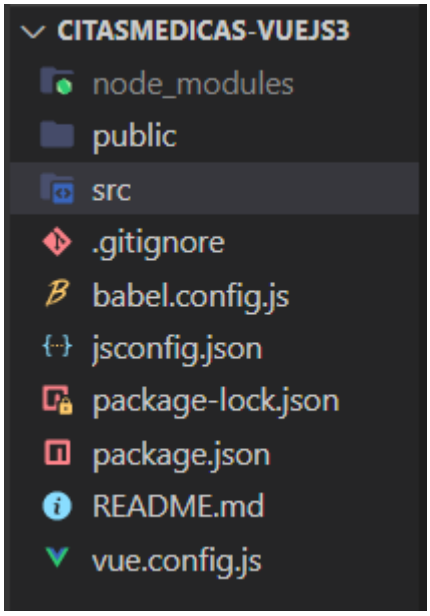
Para verificar la versión instalada utilizamos el siguiente comando.

```
vue --version
```

Para crear un proyecto nuevo utilizamos el comando ***vue create nombre_proyecto*** y seleccionamos la opción de vue 3.

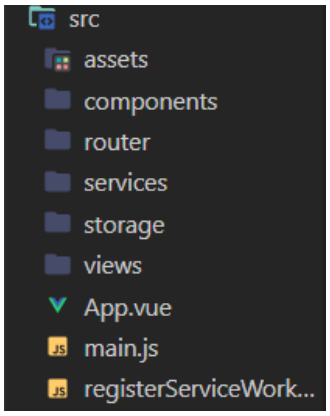
3.2 Estructura del proyecto

La distribución de carpetas está definida de la siguiente manera:



Cada una de las carpetas presenta los archivos de configuración necesarios. Se pueden visualizar la carpeta public, carpeta SRC (contiene los archivos a configurar), la carpeta node_modules y los archivos json necesarios. La carpeta con la que se trabaja es la SRC, ya que dentro de ellas se deberán configurar los archivos que permitirán desarrollar las vistas del proyecto, así como su funcionamiento y código JavaScript.

La carpeta SRC contiene los siguientes archivos



La carpeta assets contiene las configuraciones de bootstrap.

La carpeta **components** contiene los archivos que sirvan de apoyo y como agregados en el proyecto, se incluye el archivo **navbar.vue** y el archivo **HelloWorld.vue**.

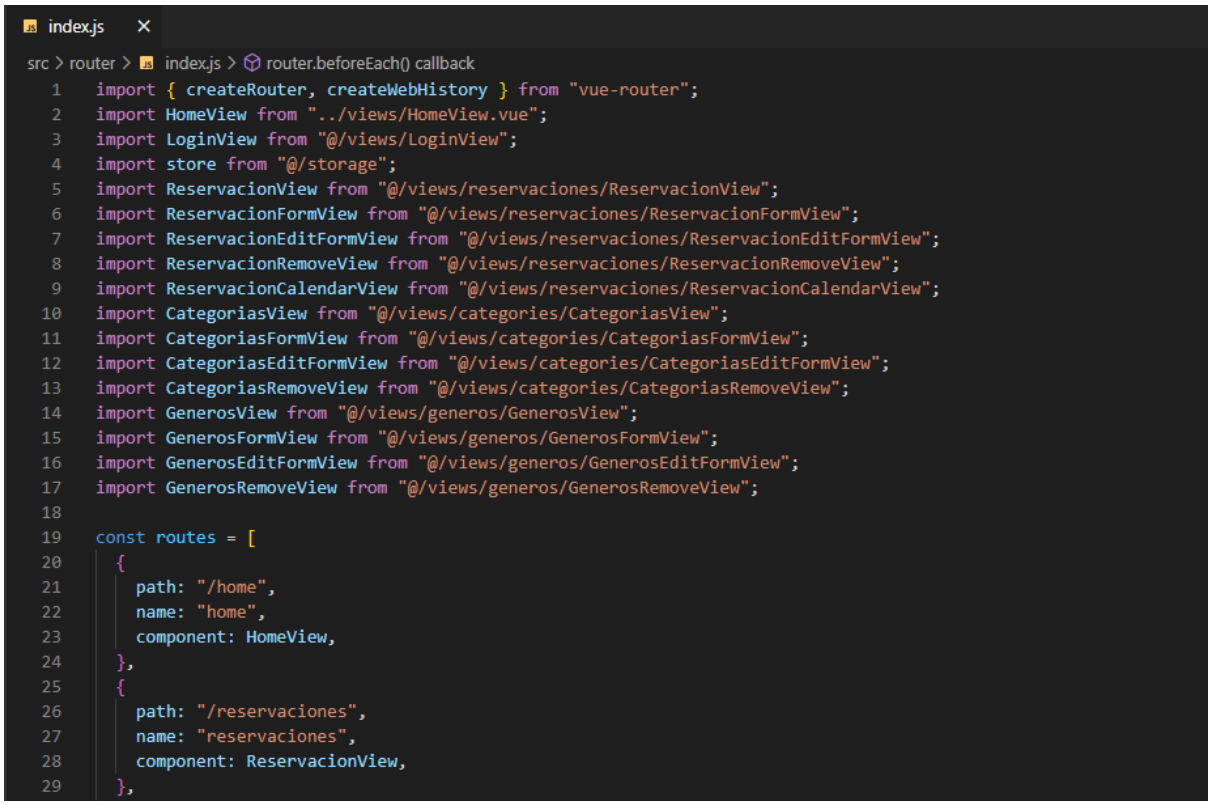
NavComponent.vue

src > components > NavComponent.vue > Vetur > { } "NavComponent.vue" > template

```
<template>
  <nav class="navbar navbar-expand-lg bg-light mb-3">
    <div class="container-fluid">
      <router-link to="/home" class="navbar-brand" >{{$store.state.isUser.name}} {{$store.state.isUser.lastname}}</router-link>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
      aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav ms-auto mb-2 mb-lg-0">
          <li class="nav-item">
            <router-link to="/home" class="nav-link active" aria-current="page" href="#">Home</router-link>
          </li>
          <!-- <li class="nav-item">
            <router-link to="/about" class="nav-link">Reservacion</router-link>
          </li> -->
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">
              Reservaciones
            </a>
            <ul class="dropdown-menu">
              <li><router-link class="dropdown-item" to="/crearreservacion">Crear</router-link></li>
              <li><router-link class="dropdown-item" to="/reservaciones">Ver reservaciones</router-link></li>
              <li><router-link class="dropdown-item" to="/calendarreservations">Ver Calendario</router-link></li>
            </ul>
          </li>
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">
              Perfil
            </a>
            <ul class="dropdown-menu">
              <li><a class="dropdown-item" href="#">Ver</a></li>
```

Código de archivo que contiene el navbar utilizado.

La carpeta router contiene el archivo index.js



Sección del código de archivo que contiene las rutas y las views utilizadas.

Este archivo contiene la importación de todas las views utilizadas y se configuran las rutas utilizadas para cada módulo, las cuales son configuradas en el path.

La carpeta **services** contiene el archivo index.js que contiene la configuración para la conexión con axios. En él se establece la ruta que fue definida en el node.js y su posterior conexión con el backend realizado.

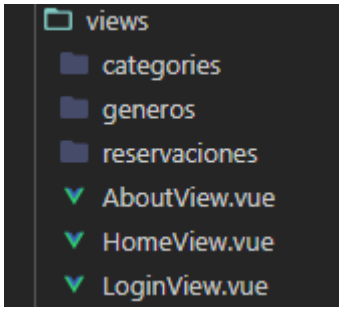
index.js M

src > services > index.js > ...

```
import axios from "axios";
/*** Configuraciones para conexion axios * */

axios.defaults.baseURL = "http://localhost:9000/api/";

axios.defaults.headers = {
  "Content-Type": "application/json",
  Accept: "application/json",
};

axios.interceptors.request.use(
  (config) => {
    let token = localStorage.getItem("token");
    if (token) {
      config.headers.tokenauthentication = token;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);
axios.interceptors.response.use(
  (response) => {
    return response;
  },
  (error) => {
    return Promise.reject(error);
  }
);

export default axios;
```

Código de archivo de la conexión con axios.

La carpeta views contiene los archivos de las vistas de la API y está estructurado de la siguiente manera:



La carpeta categorías, géneros y reservaciones contiene los cuatro archivos de vistas, actualizar, agregar y eliminar respectivamente. Fuera de estas carpetas se encuentran los archivos AboutView, HomeView y LoginView.

LoginView.vue

src > views > LoginView.vue > Vetur > {} "LoginView.vue" > template

```
<template>
<div class="row mt-3 d-flex justify-content-center">
  <div class="col-3 col-md-3 col-lg-3 col-xl-3">
    <div class="card">
      <div class="card-header bg-primary text-white">
        Inicio de Sesión
      </div>
      <div class="card-body">
        <form type="post"  class="row g-3" @submit.prevent="loginUser">
          <div class="col-12">
            <label>Correo electronico</label>
            <input type="text" class="form-control" placeholder="email" v-model="email" required />
          </div>
          <div class="col-12">
            <label>Contraseña:</label>
            <input type="password" class="form-control" placeholder="contraseña" v-model="password" required />
          </div>

          <div class="col text-center">
            <div  v-show="loading" class="spinner-border text-primary" role="status">
              <span class="visually-hidden">Loading...</span>
            </div>
          </div>

          <button v-show="!loading" type="submit" class="btn btn-primary">Iniciar Sesión</button>
        </form>
      </div>
    </div>
  </div>
</div>
</template>
```

Código de sección template que contiene la configuración de la vista.

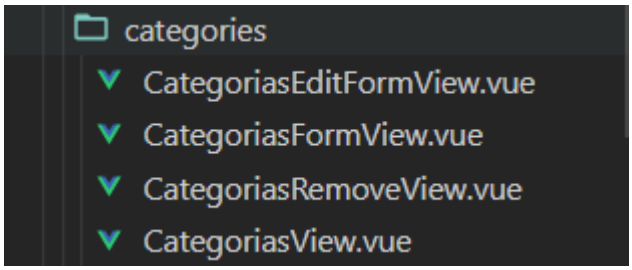
LoginView.vue

src > views > LoginView.vue > Vetur > { } "LoginView.vue" > template

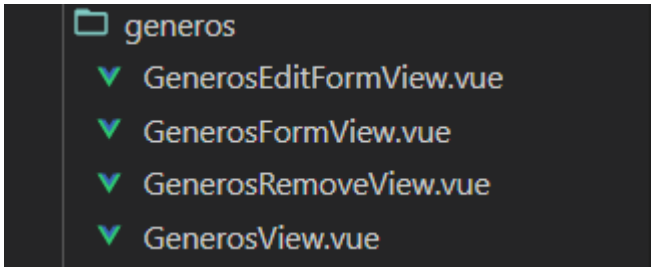
```
<script>

import store from "@/storage";

export default {
  name: "LoginView",
  data(){
    return {
      email:'',
      password: '',
      loading: false
    }
  },
  methods: {
    async loginUser () {
      let params = {
        email: this.email,
        password: this.password
      }
      this.loading = true;
      await this.axios.post('/auth/login', params)
        .then((res) => {
          const {token, user, message} = res.data
          localStorage.setItem('token', token);
          this.$store.commit('setIsAuthenticated', true);
          this.$store.commit('setUserData', user);

          this.$router.push("/home");
          this.loading = false;
        })
        .catch((err) => {
          //const {message} = err.response.data;
          console.log(err)
          this.loading = false;
        });

    }
  }
}
</script>

<style scoped>

</style>
```

Sección de código script.

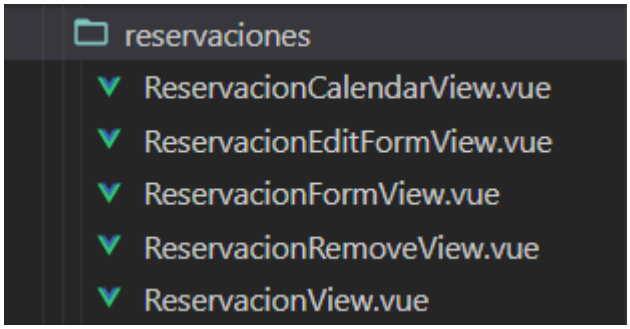
Carpeta categorías



Carpeta géneros



Carpeta reservaciones



Archivo App.vue

App.vue

src > App.vue > Vue Language Features (Volar) > {} script

```
<template>
  <NavComponent v-show="$store.state.isAuthenticated" />
  <router-view/>
</template>

<style>
</style>
<script>
// @ is an alias to /src
import NavComponent from '@/components/NavComponent.vue'

export default {
  name: 'App',
  components: {
    NavComponent
  },
  computed: {
  }
}
</script>

```

Archivo main.js

main.js

src > main.js

```
import { createApp } from "vue";
import App from "./App.vue";
import "./registerServiceWorker";
import router from "./router";

import VueAxios from "vue-axios";
import store from "@/storage";

import "@/assets/bootstrap.min.css";
import "@/assets/bootstrap.bundle.min.js";
import axios from "@/services";

createApp(App).use(router).use(store).use(VueAxios, axios).mount("#app");

```

MANUAL DEL USUARIO

Introducción

La utilidad de las APIS desarrolladas con javascript y node. js se vuelven herramientas útiles para páginas web, por esta razón es necesario desarrollar y poder entender los procesos que pueden realizar las diferentes acciones programadas. En el presente manual, se presentan y describen las diferentes acciones que realiza cada módulo y como es la forma de su funcionamiento. Se presentan las 4 acciones de un CRUD (insertar, mostrar, actualizar y eliminar) aplicadas a cada uno de los módulos que conforman la base de datos y el funcionamiento de la misma, así como acciones adicionales para una mejor funcionalidad del proyecto.

Objetivos

Objetivo General

Presentar como utilizar cada una de las partes de la API rest y que el usuario conozca cada una de las partes de esta.

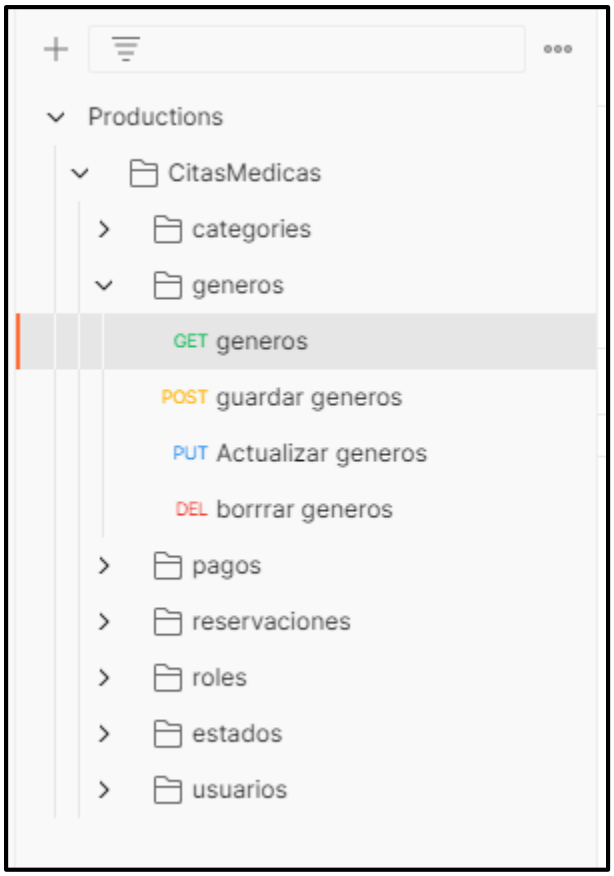
Objetivo Específico

- Mostrar al usuario la información resumida de nuestra API rest
- Dar a conocer al usuario los pasos para el uso correcto de la API REST.
- Dar a conocer al usuario cada una de las partes de la API REST.

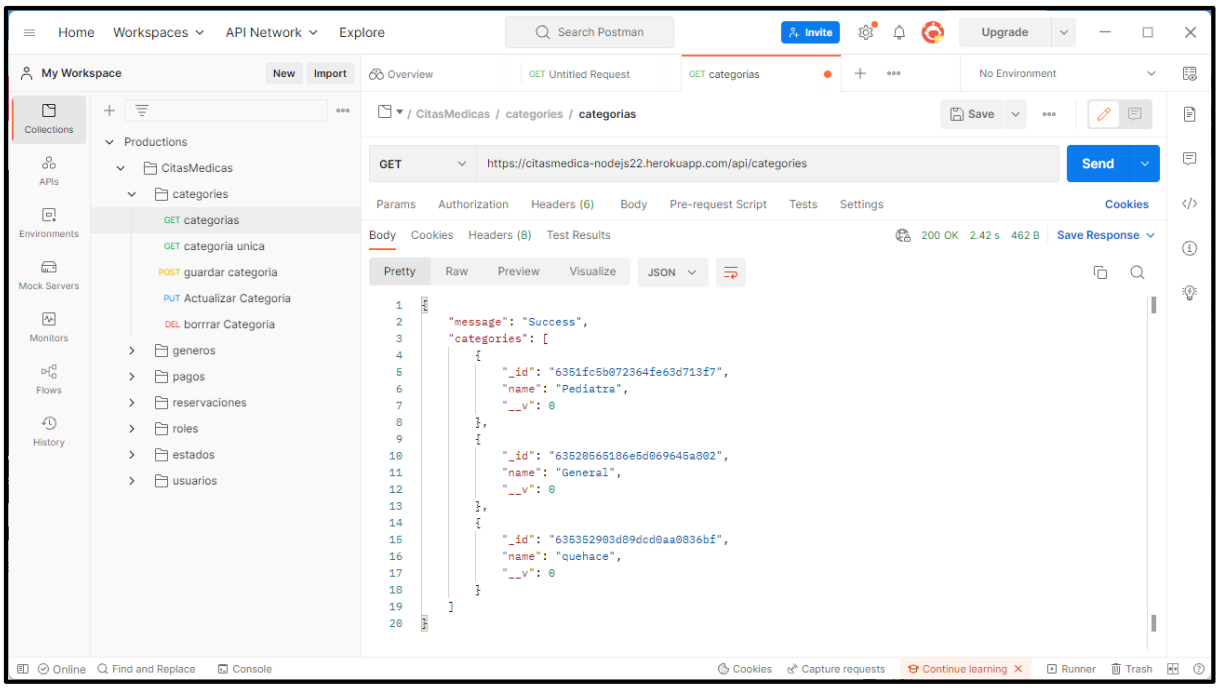
Desarrollo

1. **Funcionalidad de los endpoint que forman la API.**

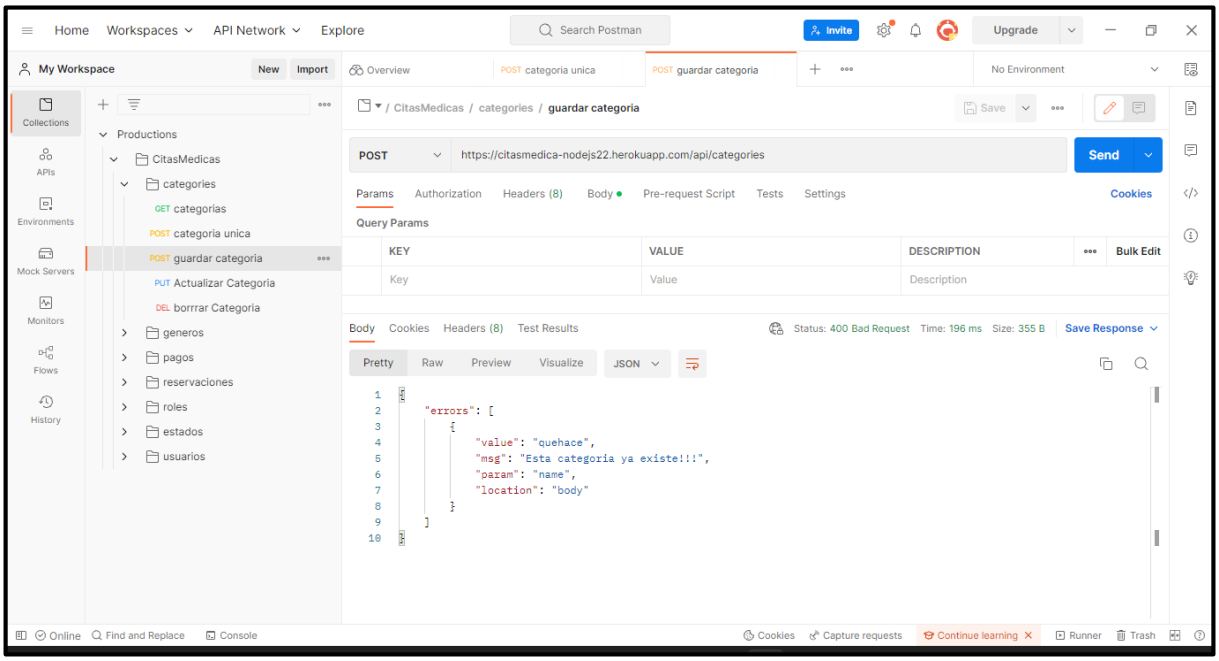
La API está conformada de las siguientes partes: categories, géneros, pagos, reservaciones, roles, estados, usuarios



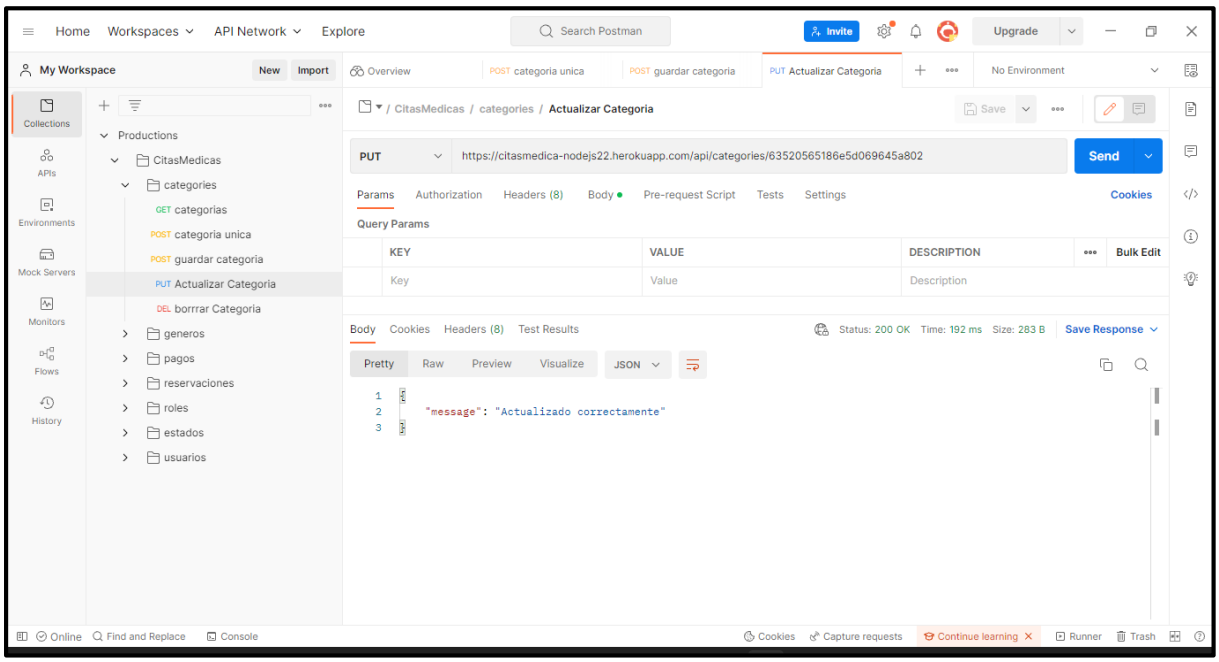
1. Seleccionamos el método GET al darle click en el botón “send”, mostrará el resultado, podrá observar que se puede observar la parte el ID, el nombre de la categoría.



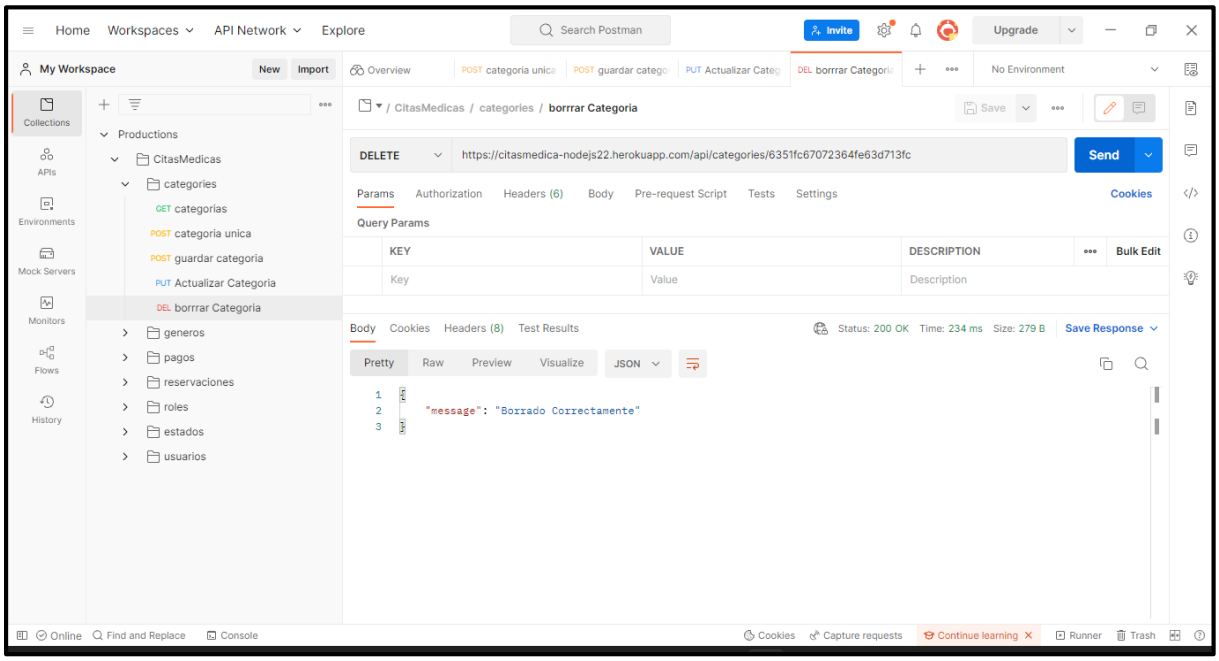
2. Seleccionamos el método POST igualmente se selecciona el botón “Send” para ver el resultado de los datos guardados.



3. Seleccionamos el método PUT y podremos ver la actualización de la información, así mismo se mostrará un mensaje que confirma la actualización.

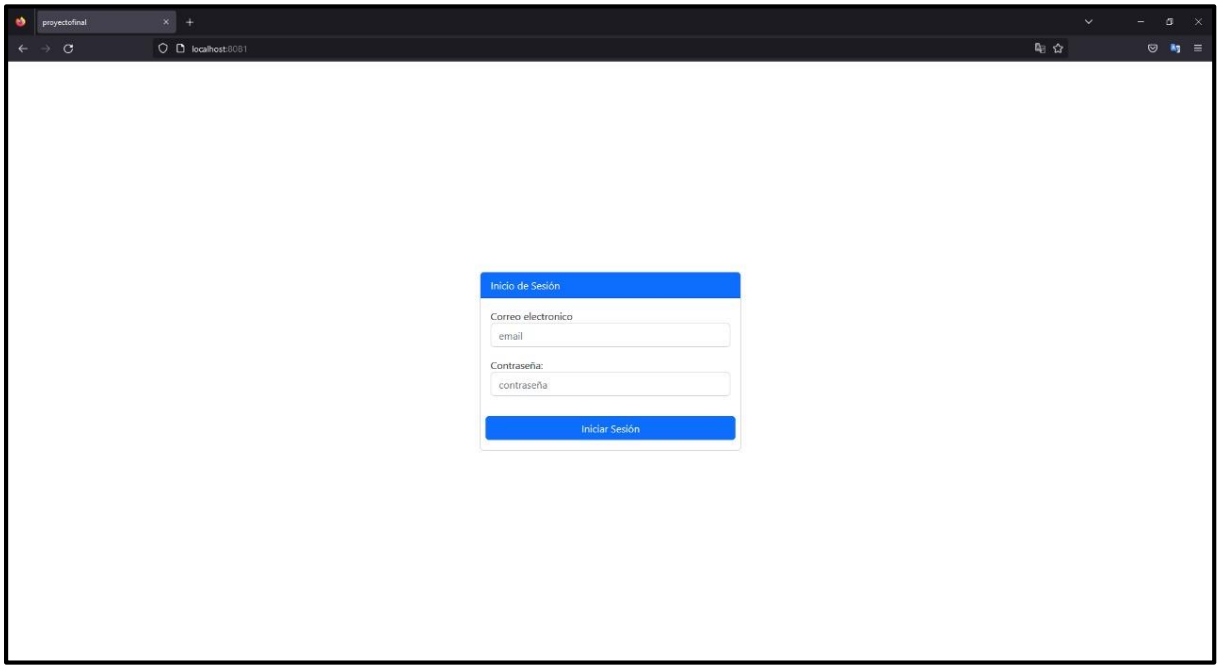


4. Seleccionamos el método Delete, el cual tiene la función de eliminar la función de eliminar, una vez realizada nos mostrará un mensaje que nos indique que se ha eliminado la información de manera correcta.



2. Funcionalidad de la API desde la web.

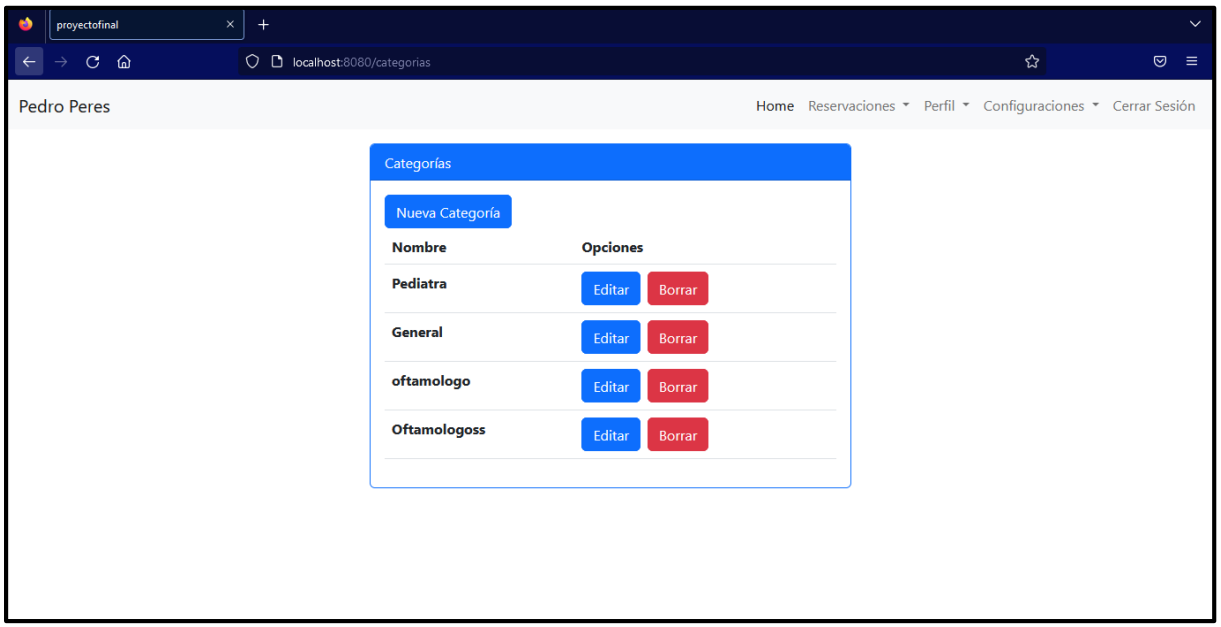
Para poder ingresar a la API, es necesario tener un navegador de preferencia y escribir la siguiente URL: localhost:8080, la cual nos mostrará la pantalla de login del proyecto.



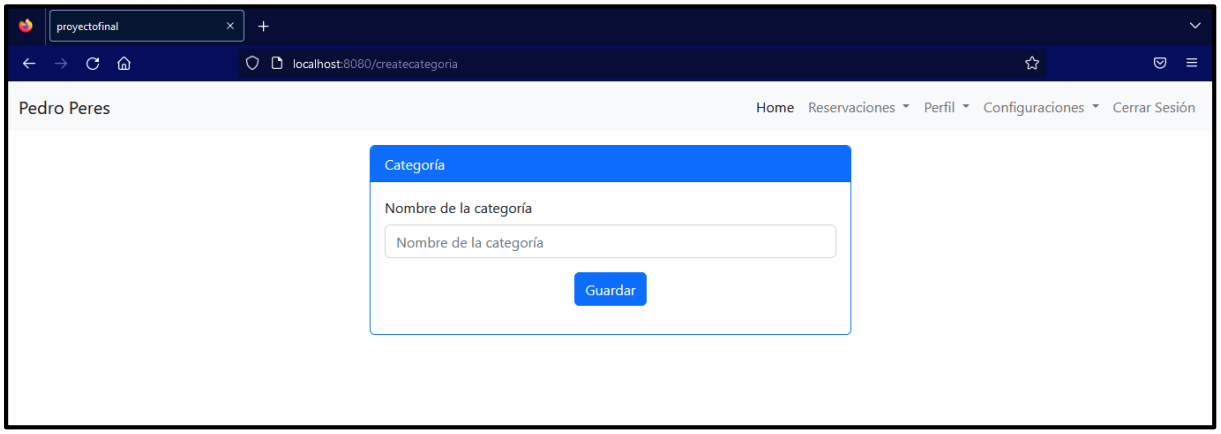
Luego debe ingresarse con las credenciales correspondientes y se abrirá la pantalla principal del proyecto.

Se muestra el navbar que nos permite navegar en diferentes módulos: Home, reservaciones, configuraciones: categorías, géneros.

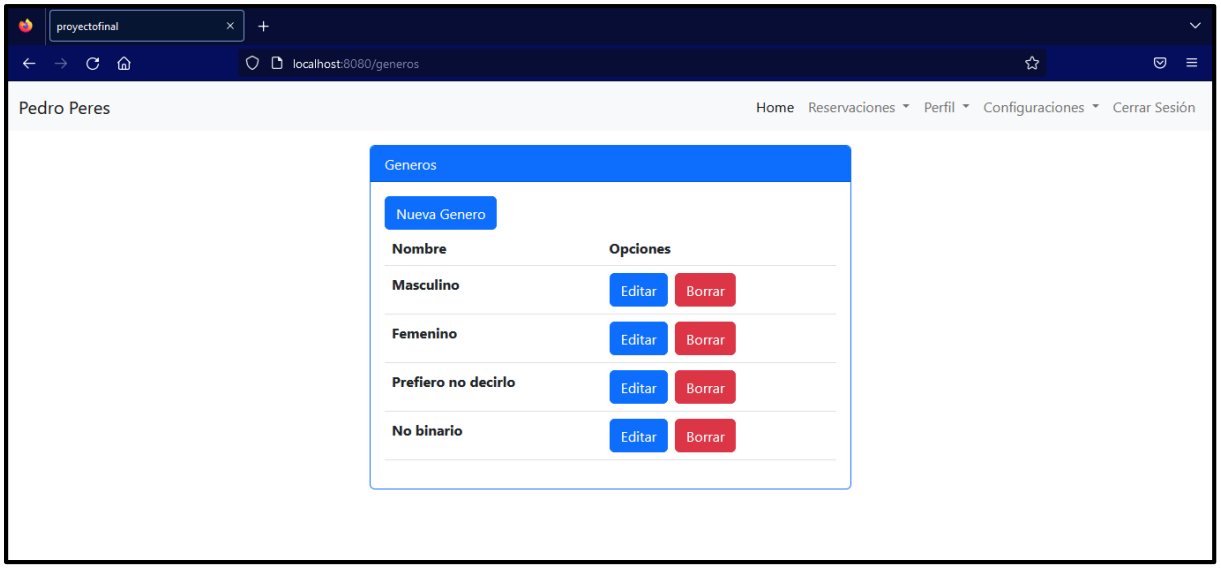
Para ingresar al módulo de categorías, se debe hacer clic en la parte de configuraciones y se desplegara la lista de todas las categorías, así como la opción de agregar una nueva y la opción de editar y eliminar.



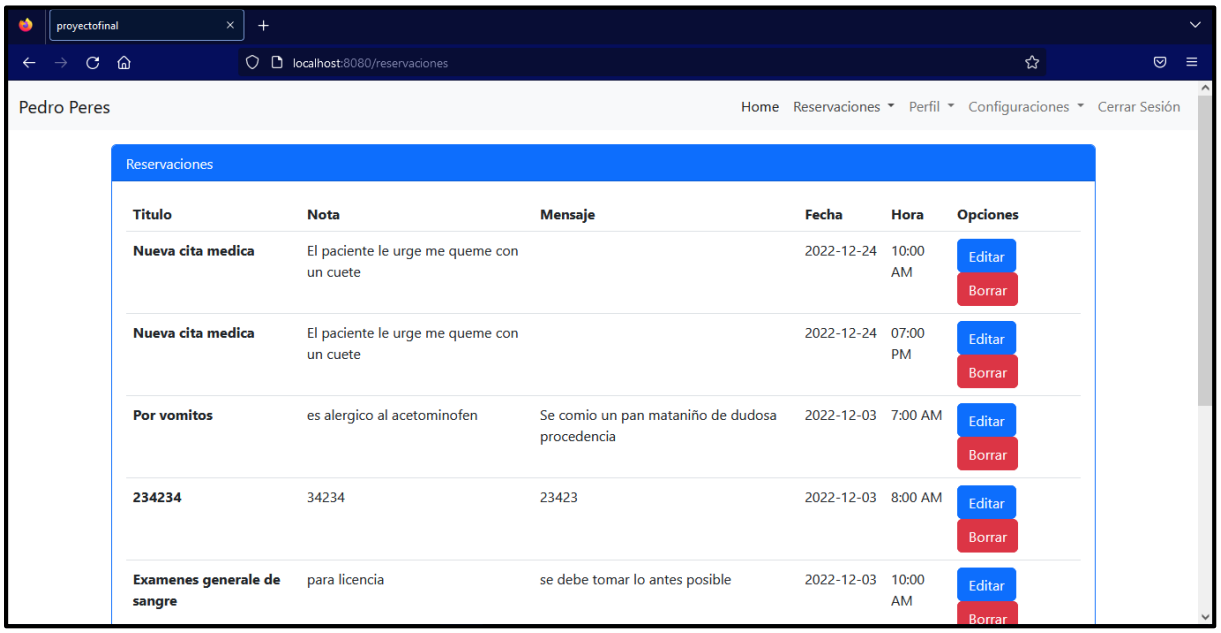
Al dar clic en el botón de nueva categoría, se presenta un formulario que podrá registrarlas.



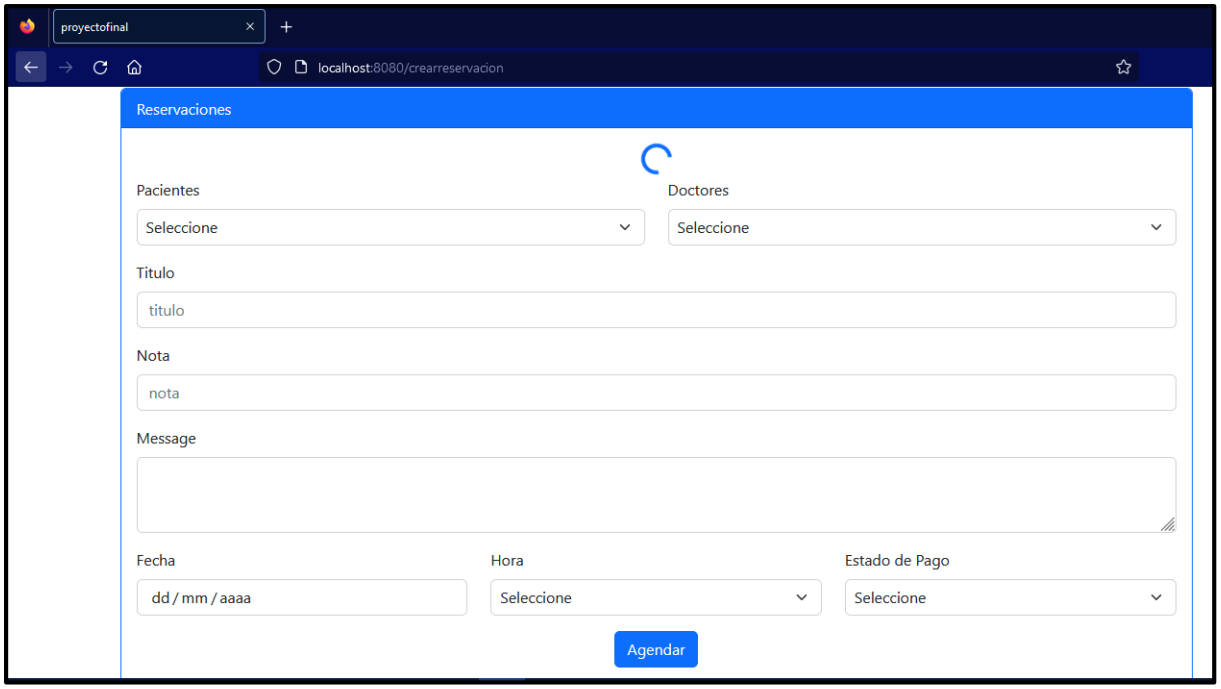
Para ingresar al módulo de géneros, se ubica en configuraciones para poder ingresar; de igual manera se presenta la lista de los géneros almacenados, así como la opción de agregar, actualizar y eliminar.



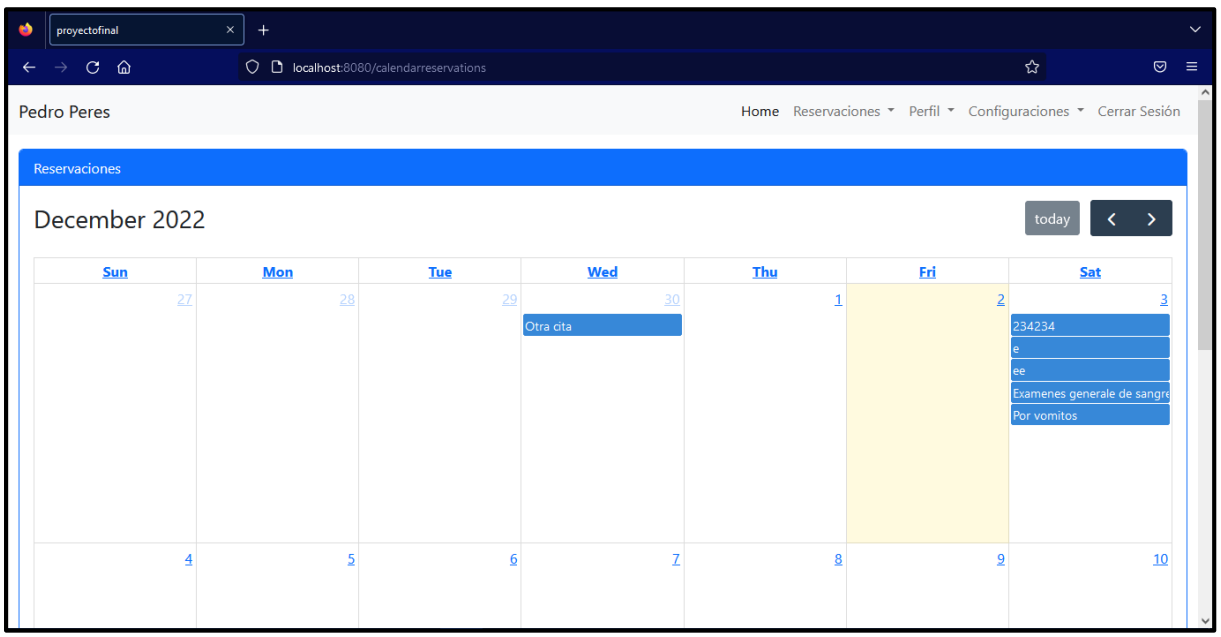
Para poder revisar las reservaciones realizadas, se debe ubicar en la sección de reservaciones, donde, se presentan las opciones de agregar, editar y eliminar.



Para agregar una nueva reservación, se debe completar el siguiente formulario.



Para visualizar la agenda semanal o mensual de todas las reservaciones realizadas, se presenta en formato calendario.



OTROS

Librerías utilizadas

A continuación, se presentan todas las dependencias que se utilizaron para el desarrollo del proyecto.

- **Mongoose**

Link: https://www.npmjs.com/package/mongoose

Mongoose es una herramienta de modelado de objetos de MongoDB diseñada para trabajar en un entorno asíncrono. Mongoose soporta tanto promesas como callbacks.

Mongoose proporciona una solución directa, basada en esquemas, para modelar los datos de la aplicación. Incluye la función de conversión de tipos, la validación, la construcción de consultas, los ganchos de lógica de negocio y más, desde el principio.

- **Nodemon**

Link: https://www.npmjs.com/package/nodemon

Nodemon es una herramienta que ayuda a desarrollar aplicaciones basadas en Node.js reiniciando automáticamente la aplicación node cuando se detectan cambios de archivos en el directorio.

Nodemon no requiere ningún cambio adicional en el código o método de desarrollo. Es una envoltura de reemplazo para node. Para utilizar nodemon, hay que sustituir la palabra node en la línea de comandos cuando ejecute su script.

- **Express**

Link: https://www.npmjs.com/package/express

Es un módulo de Node.js disponible a través del registro npm, proporciona herramientas pequeñas y robustas para servidores HTTP. Está diseñado para construir aplicaciones web de una sola página, multipágina e híbridas, también se ha convertido en el estándar para desarrollar aplicaciones backend con Node.js.

- **Express-validator**

Link: https://www.npmjs.com/package/express-validator

Express-validator es un conjunto de middlewares express.js que envuelve las funciones de validación y sanitización de validator.js.

- **Cors**

Link: https://www.npmjs.com/package/cors

CORS es un paquete de node.js para proporcionar un middleware Connect/Express que puede utilizarse para habilitar CORS con varias opciones.

- **Bcryptjs**

Link: https://www.npmjs.com/package/bcryptjs

Optimizado bcrypt en JavaScript con cero dependencias. Compatible con la vinculación de bcrypt de C++ en node.js y también funcionando en el navegador. Bcrypt es una función adaptativa: con el tiempo, el recuento de iteraciones se puede aumentar para hacerlo más lento, por lo que sigue siendo resistente a los ataques de búsqueda de fuerza bruta, incluso con el aumento de la potencia de cálculo.

- **Dotenv**

Link: https://www.npmjs.com/package/dotenv

Dotenv es un módulo de dependencia cero que carga las variables de entorno de un archivo .env en process.env. El almacenamiento de la configuración en el entorno separado del código se basa en la metodología de The Twelve-Factor App.

- **Jsonwebtoken**

Link: https://www.npmjs.com/package/jsonwebtoken

En el mismo se define un mecanismo para poder propagar entre dos partes, y de forma segura, la identidad de un determinado usuario, además con una serie de claims o privilegios. Estos privilegios están codificados en objetos de tipo JSON, que se incrustan dentro de del payload o cuerpo de un mensaje que va firmado digitalmente.

- **Validaciones**

Se utilizó el módulo express-validator para las validaciones de los datos. Siguiendo una serie de pasos para poder realizar las configuraciones y aplicar estos módulos en la API.

Link: https://codigoencasa.com/validacion-de-entradas-de-usuario-en-su-aplicacion-express-js-con-express-validator/

- **Full Calendar.**

Utilizado para crear y mostrar la vista de calendario.

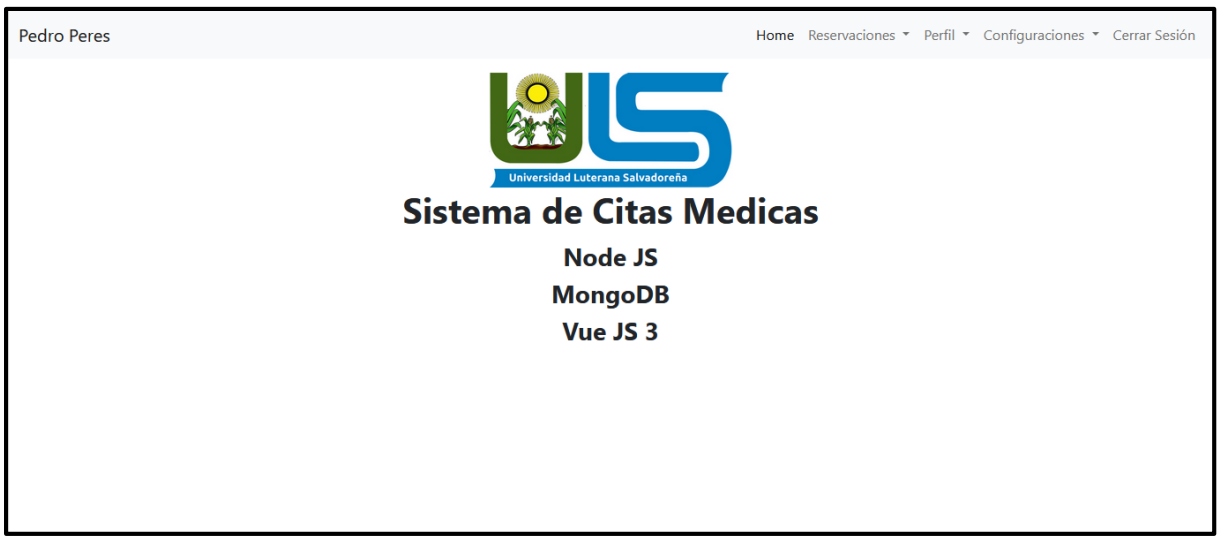
- **Axios.**

Es una librería de JavaScript que ayuda a realizar las peticiones HTTP en el navegador. Está basado en promesas para ser utilizado con node.js

Vistas de la aplicación.

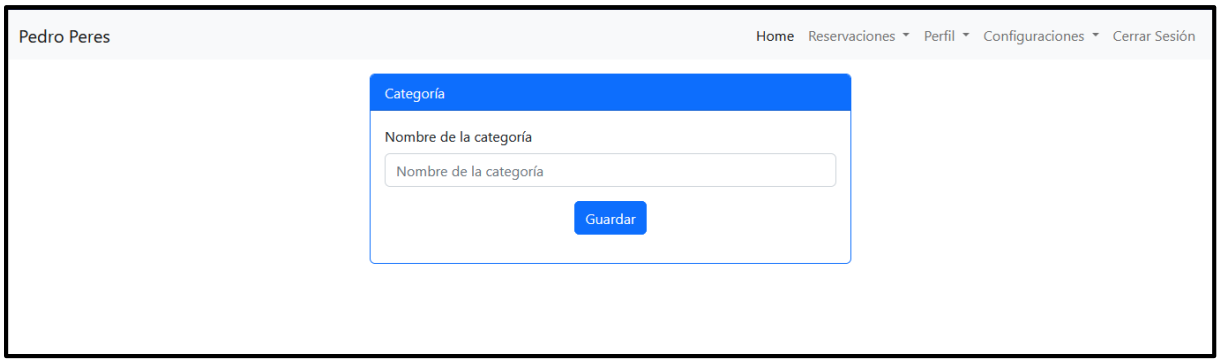
- **Pantalla Principal/Inicio de sesión**

- **Pantalla principal**

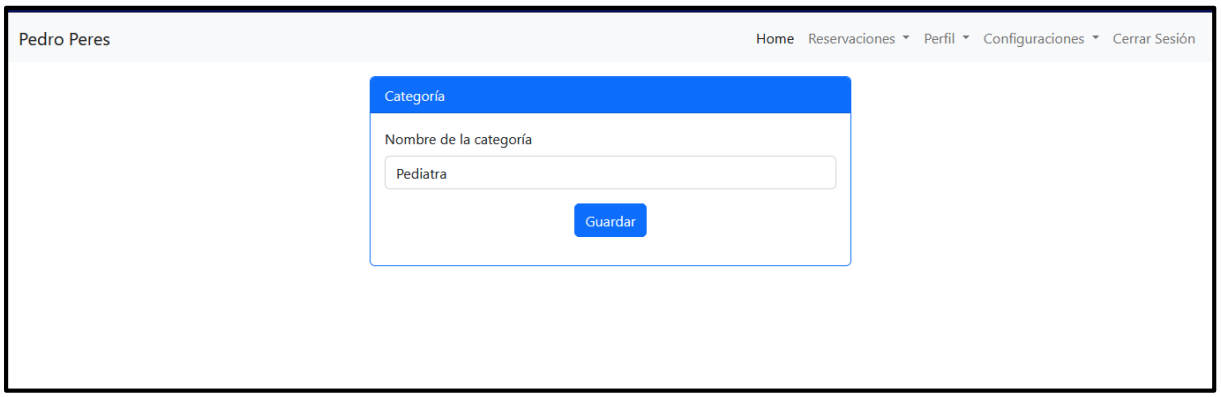


❖ **Módulo Categorías.**

❖ **Nueva categoría.**



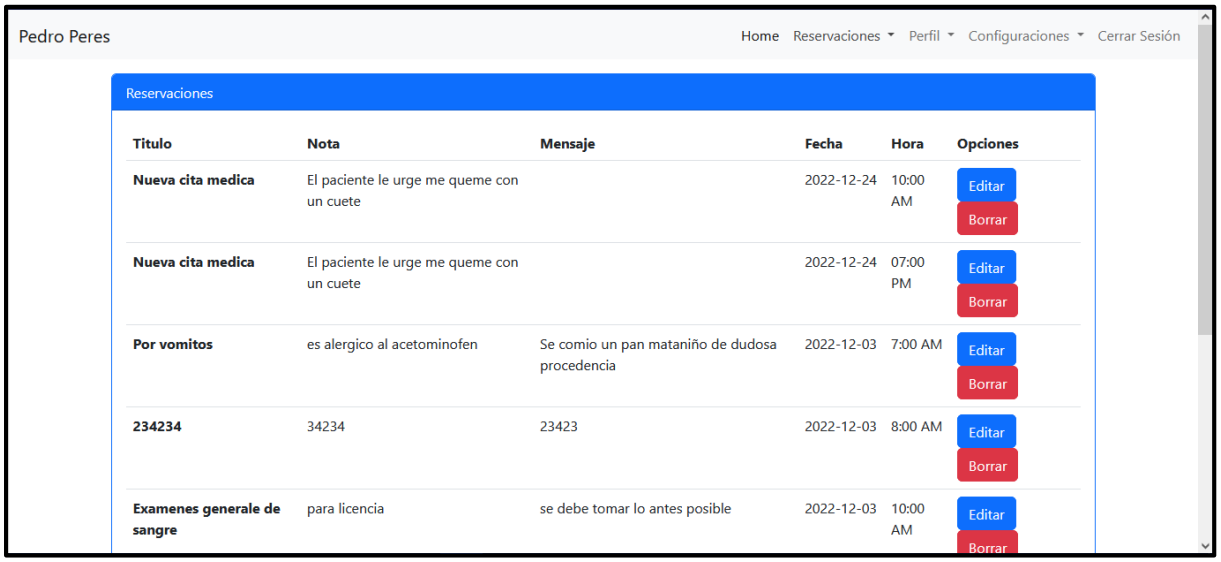
❖ **Editar categoría.**



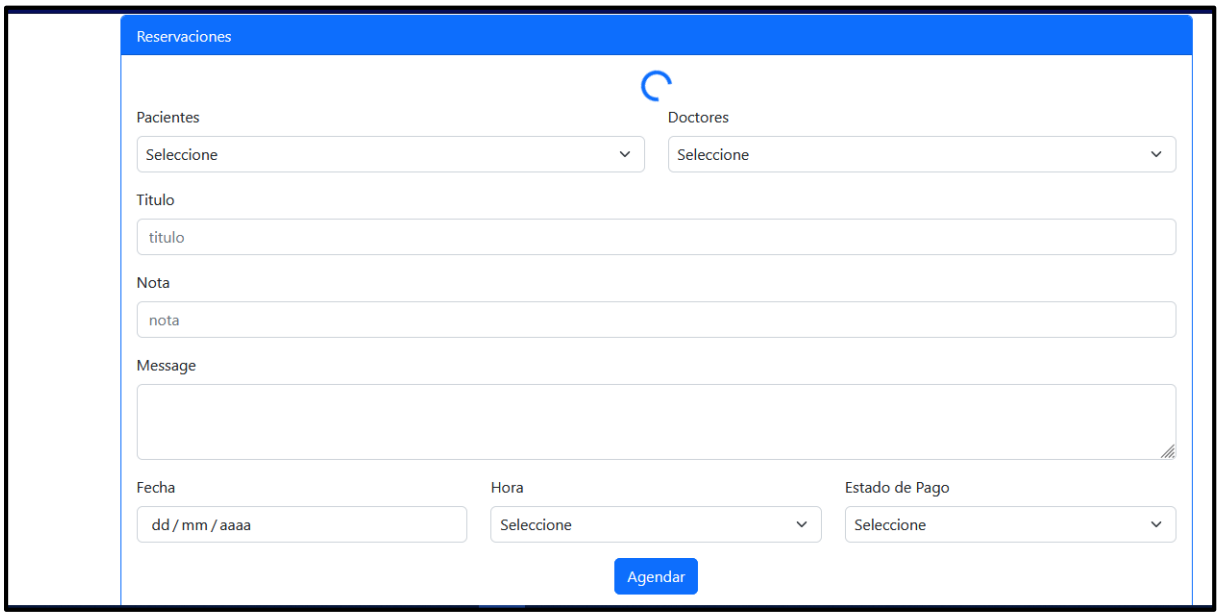
❖ **Módulo géneros**

❖ **Nuevo género.**

❖ **Modulo reservaciones.**



❖ **Nueva reservación**



❖ **Vista de calendario de citas realizadas.**