Universidad Internacional San Isidro Labrador

Tema: Página web para citas médicas

Estudiante:

Aaron Elizondo Vargas

Profesor:

Estefanía Boza Villalobos

Cuatrimestre II 2025

Contenido

Tabla de Figuras

Justificación

El desarrollo de esta página web para citas médicas va dirigida para consultorios privados, responde a la necesidad de mejorar la eficiencia en la gestión de consultas. Esta herramienta tecnológica permitirá a los doctores administrar sus horarios de manera más efectiva ofreciendo una experiencia fácil de usar a los clientes. Se podrá agendar, modificar o cancelar citas. Asimismo, el administrador de la página podrá crear, editar y eliminar citas.

Asimismo, el sistema permitirá a los usuarios autenticarse de forma segura, lo que garantiza la protección de los datos sensibles. También, se implementará un diseño responsivo en HTML y CSS para asegurar la compatibilidad con múltiples dispositivos. Se hará uso de una base de datos como SQLServer para almacenar los datos de los clientes y las citas. Se integrará un sistema de pago y se podrá llevar control de los cobros por las consultas.

Objetivo General

Crear una página web que permita a doctores privados agendar y habilitar citas a los clientes de manera más eficiente, optimizando tiempo y disponibilidad.

Objetivos Específicos

- Incorporar un rol de administrador para que los doctores puedan habilitar, crear, eliminar y editar citas con su fecha y hora.
- Permitir que desde un rol de cliente se puedan seleccionar, cancelar o cambiar una cita.
- Conectar la aplicación con una base de datos SQL para el almacenamiento de citas.

Alcances

- Seguridad y Protección de Datos: Implementación de autenticación segura para proteger la información de los pacientes. Cifrado de datos y cumplimiento de normativas de privacidad.
- Administración de Horarios y Disponibilidad: Configuración de horarios personalizados por parte de los doctores. Bloqueo automático de horarios ocupados para evitar conflictos.
- Integración con Sistemas de Facturación: Conexión con plataformas de pago para gestionar cobros de consultas. Registro de pagos y emisión de facturas electrónicas.
- Accesibilidad Multiplataforma: Compatible con dispositivos móviles y computadoras para facilitar el acceso desde cualquier lugar. Diseño responsivo para una mejor experiencia de usuario.

Requerimientos

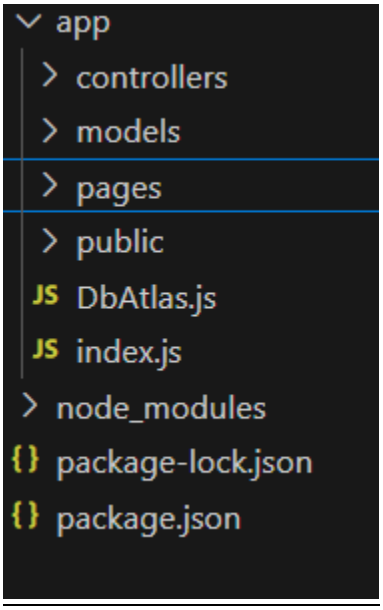
Funcionales

- Envío de confirmaciones y notificaciones.
- Registro y autenticación de usuarios.
- Creación, modificación y eliminación de citas por parte del administrador.

No Funcionales

- Diseño responsivo con HTML y CSS.
- Seguridad en la manipulación de datos con SQL.
- Respuesta rápida en la carga de páginas y procesamiento de datos.
- Adaptabilidad a diferentes dispositivos (PC, tablet, móvil).

Figura no. 1: Estructura del sistema

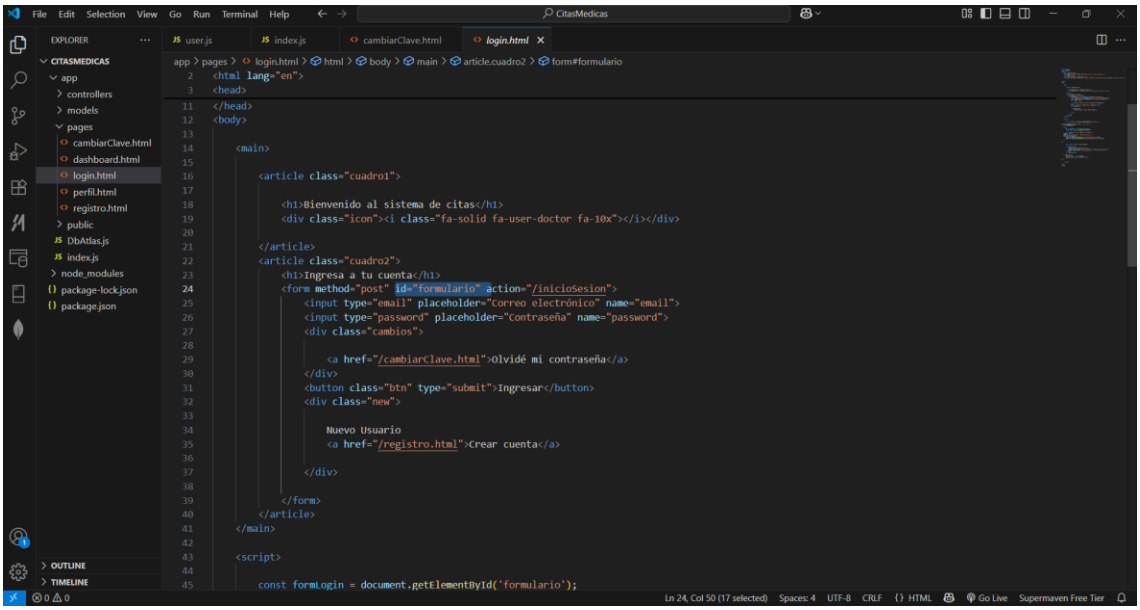


Un archivo principal que envuelve las demás carpetas, un archivo js para la base de datos y el index que es la configuración principal del proyecto.

Vistas

Login.html

Figura no. 2: Vista Login.html



En este inicio de sesión se solicita solo el correo electrónico y la contraseña, tiene enlaces hacia la vista para cambiar clave y para crear usuario, además envia el fetch a la backend mediante el id del form.

Login.css

- main tiene un ancho del 80% y 600px de alto, con la misma imagen de fondo, borde redondeado y sombra para destacar el bloque sobre el fondo.
- Formulario Los input tienen fondo oscuro translúcido, texto blanco y bordes redondeados.
- Los placeholders se animan al enfocar el campo (:focus::placeholder), moviéndose hacia arriba y desapareciendo con una transición suave.
- Inputs de tipo email y password tienen un color de fondo específico (rgba(66, 112, 206, 0.5)).
- El botón .btn tiene fondo azul oscuro, texto blanco y está diseñado para parecer un botón moderno y llamativo.

Figura no. 3: Estilos del Login

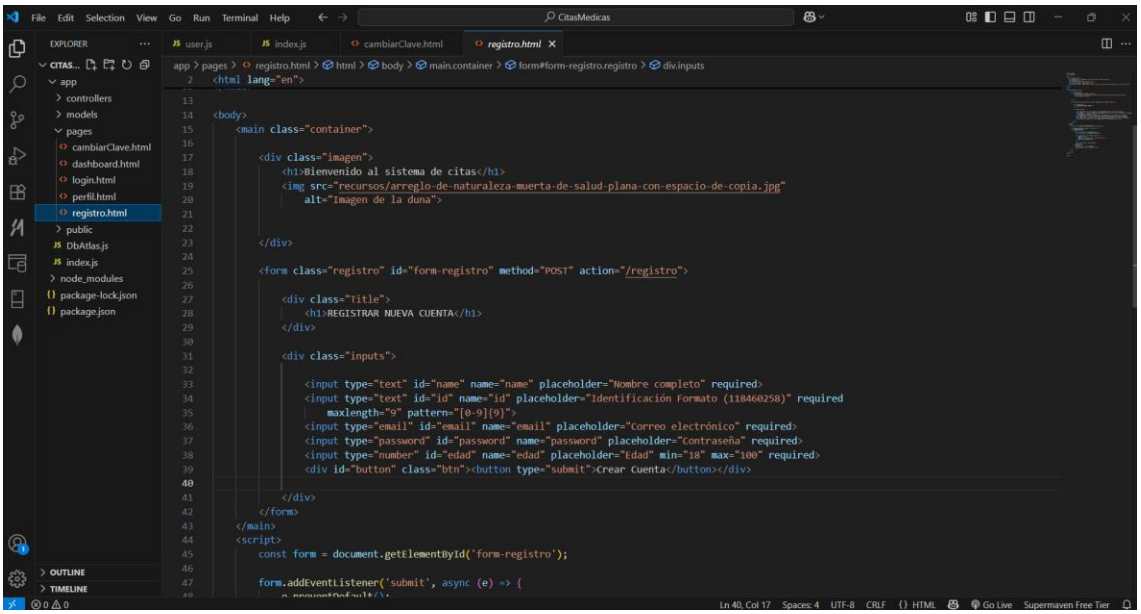
app > public > # login.css > form input[type="text"]:focus::placeholder

```css
*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
body{
    background-image: url(recursos/login.jpg);
    background-size: cover;
    font-family: Arial, Helvetica, sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    width: 100%;
}
main{
    width: 80%;
    height: 600px;
     background-image: url(recursos/login.jpg);
    background-size: cover;
    box-shadow: 0px 0px 20px 10px rgba(0, 150, 255, 0.8);
    border-radius: 20px;
    padding: 30px;
    display: flex;
    justify-content: space-around;
    align-items: center;
    color: white;
}
main .cuadro2{
    background-color: rgba(0, 0, 0, 0.5);
    padding: 20px;
    border-radius: 10px;
    width: 40%;
    margin: 40px;
    display: flex;
    flex-direction: column;
    justify-content: center;
```

Figura no. 4: Vista de inicio de sesión

Registro.html

Figura no. 5: Vista Registro.html



Este form solicita varios campos marcados como requeridos con su id único y el name que servirá para el backend.

Registro.css

Figura no. 6: Estilos de la vista Registro

app > public > # registro.css > .opcion::before

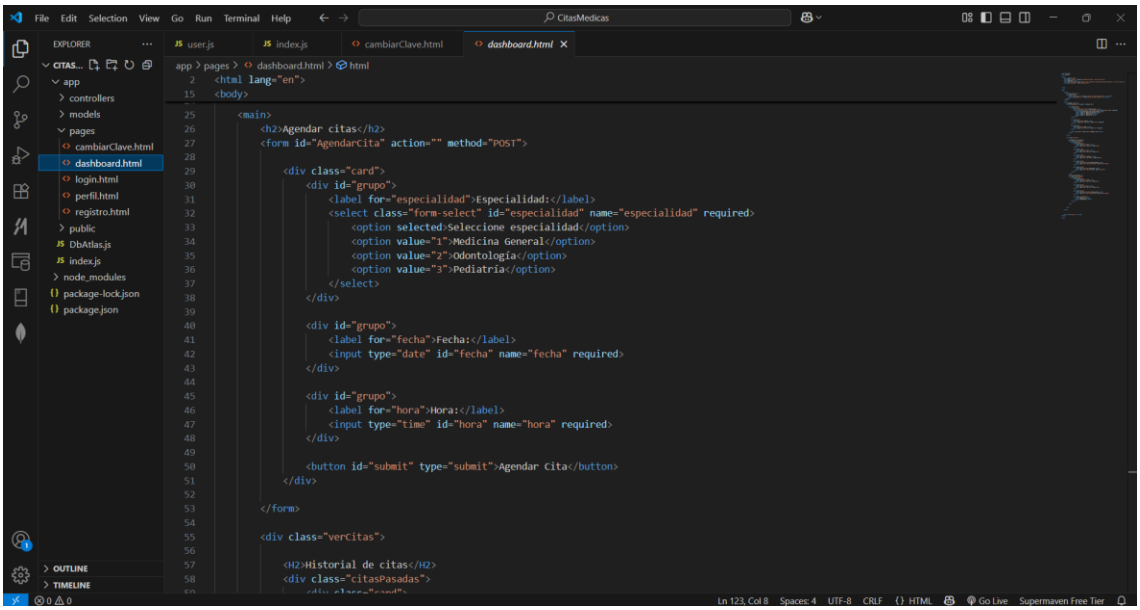
```css
*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
body{
    background-color: rgb(43, 45, 46);
    width: 100%;
    height: 100vh;
    display: flex;
    padding: 10px;
}
.container{
    width: 100%;
    display: flex;
    height: 100%;
    justify-content: center;
    align-items: center;
    padding: 10px;
    gap: 10px;
}
.imagen{
    width: 50%;
    height: 100%;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    color: white;

}
.imagen img{
    width: 100%;
    height: 100%;
    object-fit: cover;
    border-radius: 10px;
}
```

Se establece un diseño flexible (flexbox) en el body y .container para alinear elementos horizontalmente al centro de la pantalla. La clase .imagen ocupa el 50% del ancho y muestra una imagen de fondo ajustada (object-fit: cover). La clase .registro también ocupa el otro 50% e incluye el formulario de registro estilizado. Los inputs tienen fondo oscuro, texto blanco, esquinas redondeadas y márgenes para espaciado.

Figura no. 7: Vista de registro de usuarios

Dashboard.html

Figura no. 8: Vista de Dashboard



Este es la vista principal del sistema y está dividido en 3: para agendar citas, ver las citas ya hechas y las futuras citas

Dashboard.css

- El header contiene un título y una sección .icon alineada a la derecha para íconos
- El main contiene dos partes principales:

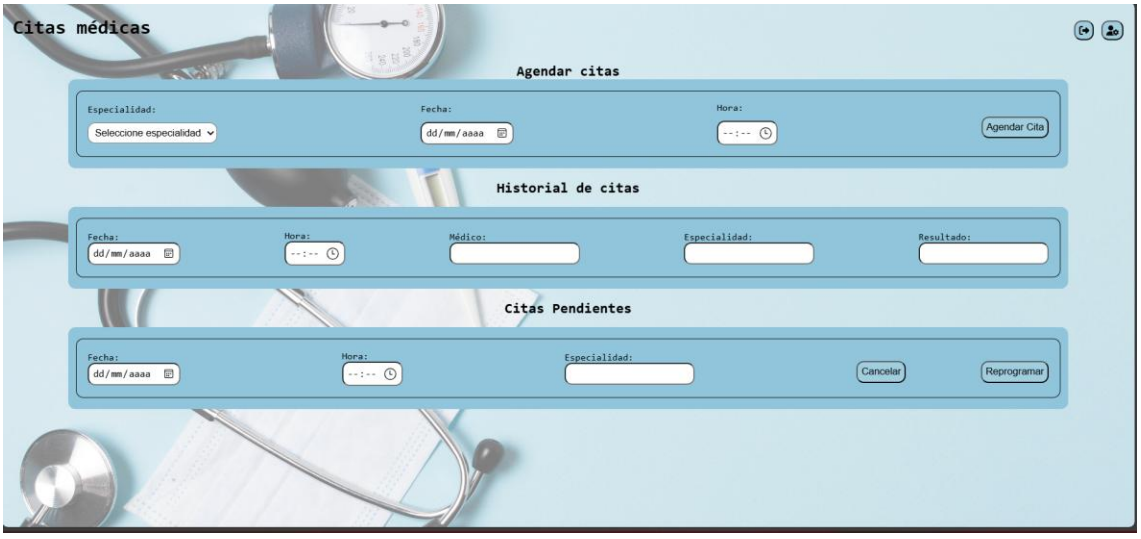
- Formulario de agendar cita (#AgendarCita): ocupa el 90% del ancho con redondeados y diseño en columna. Cada grupo de campos dentro tiene su propia tarjeta (.card) con diseño flexible, espaciado, y bordes.
- Sección de citas (.verCitas): contiene títulos y dos secciones: .citasPasadas y .futurasCitas, ambas con diseño similar al formulario, incluyendo tarjetas para cada cita con inputs y botones bien espaciados.

Figura no. 9: Estilos del Dashboard

app > public > # dashboard.css > body::before

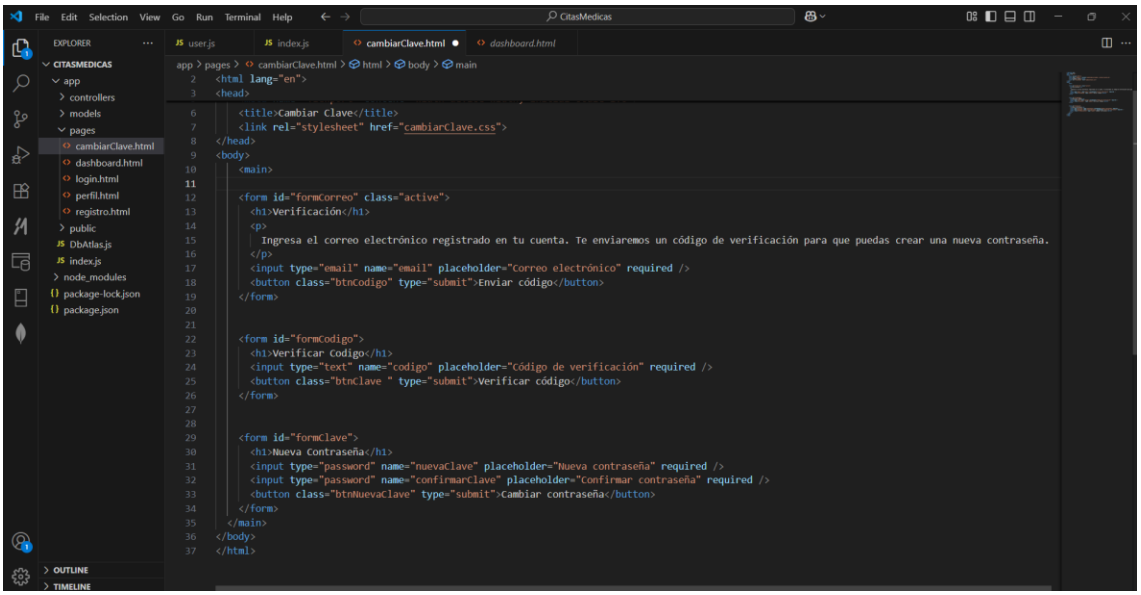
```css
*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
body {
    position: relative;
    font-family: monospace;
    min-height: 100vh;
    width: 100vw;
    margin: 0;
}

body::before {
    content: "";
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-image: url(recursos/arreglo-de-naturaleza-muerta-de-salud-plana-con-espacio-de-copia.jpg);
    background-size: cover;
    background-position: center;
    opacity: 0.5;
    z-index: -1;
}

header{
    display: flex;
    align-items: center;
    padding: 15px;
    color: rgb(0, 0, 0);
}
.icon{
    display: flex;
    margin-left: auto;
    gap: 10px;
```

Figura no. 10: Registrar citas medicas



CambiarClave.html

Figura no. 11: Vista CambiarClave.html



Esta vista esta divida en 3 partes, la primera servirá para verificar el usuario mediante correo electrónico con un código, la segunda verifica el código y si es correcta dejara cambiar la clave.

CambiarClave.css

El body usa una imagen de fondo ajustada a pantalla completa y centra el contenido con flexbox. El contenedor principal (main) tiene un fondo blanco semi-transparente, bordes redondeados y una sombra azul brillante que le da enfoque visual.

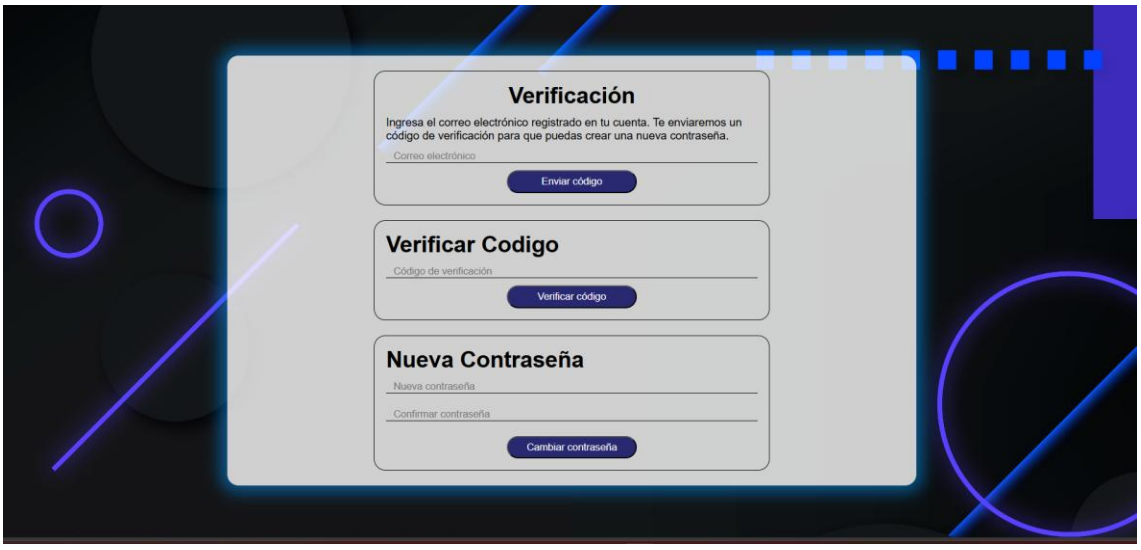
Cada formulario comparte el mismo estilo: diseño en columna, bordes redondeados, borde negro, espacio interno (padding) y separación entre elementos (gap). Los input son minimalistas, con fondo transparente y solo una línea inferior como borde, y sin outline al enfocar. Los botones tienen fondo azul oscuro, texto blanco y cambio de tono al pasar el cursor (hover).

Figura no. 12: Estilos de la vista CambiarClave

app > public > # cambiarClave.css > main

```css
*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body{
    background-image: url(recursos/login.jpg);
    background-size: cover;
    font-family: Arial, Helvetica, sans-serif;
    width: 100%;
    height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
}
main{
    width: 60%;
    height: auto;
    background-color: rgba(255, 255, 255, 0.8);
    box-shadow: 0px 0px 20px 10px rgba(0, 150, 255, 0.5);
    border-radius: 15px;
    padding: 20px;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;

}
#formCorreo{
    display: flex;
    flex-direction: column;
    font-size: 15px;
    gap: 10px;
    width: 60%;
    border: 1px solid black;
    padding: 15px;
```

Figura no. 13: Vista de Cambio de contraseña



Tecnologías utilizadas

- Para la gestión de base de datos utilicé MongoDB como motor de base de datos, usando los documentos como tipo JSON, Utilizo Mongo Atlas para gestionar la base de datos desde la nube también.
- Para el backend y frontend he utilizado Node.js y javascript utilizando fetch para enviar y recibir datos.
- Justificación técnica:
- MongoDB permite almacenar de manera flexible y fácilmente escalable sin la necesidad de crear un esquema escalable.
- Node.js por la facilidad al combinarse con MongoDB y la capacidad para manejar múltiples solicitudes simultaneas.
- Express.js: Lo utilice para organizar todas las rutas, middleware y otras cosas del proyecto, por ejemplo, al iniciar el servidor busco todas las citas guardadas disponibles y las guarda en una memoria.
- Para seguridad utilice bcrypt para usar hash de contraseñas antes de guardarlas, también uso de JWT para identificar validaciones de usuarios mediante un token creado.

Diagrama de la base de datos

Usé colecciones para usuarios y citas. Las citas se ordenan por profesión, fecha y hora, y una fecha puede tener varios usuarios, pero un reloj solo puede tener un usuario por día. En el modelo de datos que está utilizando actualmente, cada colección o tabla almacena sus datos de forma independiente y sin relaciones explícitas. Esto significa que los documentos contienen los datos completos requeridos para cada entidad, sin hacer referencia a otros documentos de otras colecciones. Por ejemplo, cada documento de la colección de citas contiene los campos necesarios para describir la cita. Este modelo se conoce como modelo desnormalizado, que prioriza la simplicidad y la velocidad de lectura, evitando la necesidad de realizar consultas complejas con múltiples uniones o referencias

Lista de funcionalidades ya completadas:

Requerimientos Funcionales

- **Envío de confirmaciones y notificaciones:** De momento solo tengo envio de verificación por correo al intentar cambiar la contraseña por si se ha olvidado usando un código generado al azar de 5 digitos.
- **Registro y autenticación de usuarios:** El registro y autenticación de usuarios ya está totalmente terminado.
- **Creación, modificación y eliminación de citas por parte del administrador:** El administrador ya puede liberar citas marcando unas casillas(checkBox) con las horas que

quiera liberar, escogiendo la fecha que quiera liberar y la especialidad a la que la quiera asignar, también puede eliminar las citas que ha liberado.

Requerimientos No Funcionales

- **Diseño responsivo con HTML y CSS:** He ido diseñando las vistas lo mas responsivo posible pero no lo he adaptado a cada tamaño de pantalla todavía.
- **Seguridad en la manipulación de datos con SQL:** He utilizado MongoDB en lugar de SQL, igualmente, he implementado seguridades en los registros de usuarios como hash para las claves, token para validar usuarios y correos generados para cambios de claves.
- **Respuesta rápida en la carga de páginas y procesamiento de datos:** Las páginas se cargan rápido

Conexión a la base de datos.

Figura no. 14: Fragmento de código-Conexión con la BD

app > JS DbAtlas.js > default

```
1   // Conexión a MongoDB Atlas
2   import mongoose from "mongoose";
3
4   mongoose.connect("mongodb+srv://aaronev2002:A08e11V2002@agendadb.zg8s0et.mongodb.net/")
5
6   const db = mongoose.connection;
7   db.on("error", (err) => console.error("Error de conexión:", err));
8   db.once("open", () => console.log("Conectado a MongoDB Atlas"));
9
10
11
12  export default db;
```

Utilizo moongose y le paso la cadena de conexión de mi BD que se puede encontrar en la nube o ya sea en MongoDB Compass que es la app de escritorio. Si la cadena no es valida o no encuentra la ruta entonces tira un mensaje de error, pero si conecta correctamente entonces se vera un mensaje de confirmación en la consola o terminal. Lo exporto como db y lo importo en mi Index.js

Conclusiones del Sistema

Desarrollo e implementación: El sistema Citas Médicas fue implementado como una solución web que permite la gestión de usuarios, programación y control de citas en línea. Se desarrolló bajo un esquema cliente-servidor, utilizando Node.js y Express en el backend, junto con MongoDB Atlas como base de datos en la nube.

Parte gráfica: La interfaz gráfica es intuitiva y modular, organizada en páginas específicas (login, registro, dashboard, perfil, gestión de roles). Cada sección cuenta con su propio archivo CSS y JavaScript, lo que facilita la personalización y mantenimiento. Además, se integró un calendario interactivo con FullCalendar, que mejora la usabilidad en la gestión de citas.

Conexión con la base de datos: El sistema utiliza MongoDB Atlas para almacenar de forma segura la información de usuarios, roles, citas y cache de citas. La conexión se maneja mediante Mongoose, lo que asegura integridad y escalabilidad en los datos.

Diseño y arquitectura: La aplicación sigue una estructura basada en el patrón MVC (Modelos, Controladores y Vistas). Esto permitió mantener separadas las responsabilidades:

- Los modelos definen la estructura de los datos.
- Los controladores gestionan la lógica de negocio (registro, login, actualización de perfil, etc.).
- Las vistas en HTML/CSS/JS representan la capa visual para los usuarios.

Backend y seguridad: El backend implementa validaciones y autenticación de usuarios. Se usaron librerías como dotenv para el manejo de credenciales y bcryptjs para cifrar contraseñas, garantizando seguridad en el manejo de datos sensibles.

Logros del sistema:

- Se logró implementar un sistema web funcional de citas médicas con múltiples roles de usuario.
- Se integró la gestión de imágenes de perfil y la administración de citas en tiempo real.
- El sistema es escalable y adaptable, con un diseño modular que facilita la incorporación de nuevas funciones.
- Se desplegó sobre una base de datos en la nube, lo que asegura disponibilidad y acceso remoto.

Vídeo del sistema

Desde el celular.mp4