TUTUTOR

Tututor es una plataforma en línea donde profesores ofrecen sus clases. Está construida utilizando Node.js con Express y emplea Sequelize para gestionar la base de datos PostgreSQL. La plataforma permite a los usuarios registrarse, seleccionar su rol como profesores o alumnos, y acceder a diferentes paneles según su rol.
Aquí están los aspectos clave del proyecto:

- **Autenticación de usuarios:** Utiliza Passport.js para autenticar a los usuarios y gestionar sesiones. Los usuarios pueden registrarse, iniciar sesión y seleccionar su rol (profesor o alumno).
- **Gestión de clases:** Los profesores pueden crear, editar y eliminar clases, además de subir imágenes y establecer una modalidad (Presencial, Online, Presencial/Online). Los alumnos pueden buscar y filtrar clases por categorías y subcategorías, además de mostrar interés en ellas.
- **Comentarios:** Los usuarios pueden agregar y eliminar comentarios en las clases, con validación para asegurar que solo el creador del comentario o el profesor de la clase pueda eliminarlo.
- **Paneles diferenciados:** Los profesores tienen un panel de administración donde gestionan sus clases, mientras que los alumnos tienen un panel donde pueden ver las clases en las que están interesados.
- **Rutas**: Las rutas están separadas por área pública y área privada, diferenciando los accesos y funciones según el rol del usuario.
- **Funcionalidades adicionales:** Cuenta con funcionalidades como carruseles de imágenes, dropdowns dinámicos, previsualización de archivos, y gestión de modales para confirmación de acciones.

1. Optimización de rendimiento y escalabilidad:

En Tututor, enfoqué la optimización de rendimiento y escalabilidad en varias áreas clave. Utilicé **PostgreSQL** junto con **Sequelize** para gestionar las relaciones entre entidades como las asociaciones **many-to-many** entre clases y subcategorías y **many-to-one** entre clases y usuarios. Estas relaciones permiten manejar consultas complejas sin comprometer el rendimiento, lo que resulta esencial a medida que la plataforma crece.

Para mejorar el rendimiento de las consultas de clases, apliqué un límite en los resultados que se cargan en la página principal, lo que asegura que solo se muestren un número reducido de clases al usuario. Esto evita sobrecargar el servidor y mejora el tiempo de respuesta.

Además, implementé un sistema de subida de imágenes utilizando **Multer**, incluyendo una barra de progreso y previsualización en el frontend . Esto asegura que los usuarios puedan verificar los archivos antes de subirlos, evitando cargas innecesarias en el servidor.

En cuanto a la entrega de archivos estáticos como imágenes y hojas de estilo, utilizo **Express** para servir estos recursos directamente desde la carpeta public, lo que asegura una entrega rápida y eficiente. Aunque no se ha implementado caching en esta fase, el sistema está preparado para crecer y manejar un mayor volumen de tráfico sin comprometer el rendimiento

2. Arquitectura modular

La estructura modular asegura que el código sea mantenible y escalable. Cada componente principal del sistema está dividido en módulos independientes:

- **Controladores:** Gestionan la lógica de negocio de las rutas públicas y privadas. Por ejemplo, los controladores de clases manejan la creación, edición y eliminación de clases, mientras que los controladores de usuarios se encargan de la autenticación y la edición de perfiles.
- **Modelos:** Los modelos están bien organizados con **Sequelize**, lo que facilita la definición de relaciones entre tablas, como la relación entre clases y subcategorías.
- **Rutas:** Separadas por áreas públicas y privadas, donde los usuarios no autenticados pueden navegar por las clases y los usuarios autenticados pueden acceder a funcionalidades avanzadas según su rol (profesor o alumno).

Esta modularidad permite escalar la aplicación agregando nuevas funcionalidades, como un módulo para gestionar comentarios o añadir notificaciones, sin afectar el resto del sistema.

3. Ciclo completo de desarrollo

Estuve involucrado en todo el ciclo de desarrollo del proyecto, desde la planificación hasta el despliegue. Los pasos clave incluyen:

- **Planificación y definición de requisitos:** Identifiqué las funcionalidades principales, como la creación de clases, la gestión de usuarios,
y el filtrado por subcategorías. Durante esta fase, se definió una arquitectura modular que permitiera escalabilidad.
- **Desarrollo iterativo:** Desarrollé el sistema de manera incremental, comenzando por la autenticación de usuarios con **Passport.js** y luego pasando a la gestión de clases y comentarios. Durante esta fase, implementé validaciones tanto en el servidor como en el cliente para asegurar que los datos fueran correctos y seguros.

4. Trabajo con grandes volúmenes de datos

El manejo de grandes volúmenes de datos es clave, sobre todo en la gestión de las clases, los usuarios y los comentarios.
Utilizando **PostgreSQL** como base de datos, pude optimizar las consultas con índices en campos clave como el nombre de las clases y las categorías, lo que mejora el rendimiento en las búsquedas y filtrados.

5. Validación de datos

El sistema de validación está presente en múltiples niveles:

5.1. Validación en el servidor
Utilicé Sequelize para implementar validaciones en los modelos, asegurando que los campos obligatorios, como el nombre de la clase o el correo electrónico de los usuarios, se completen correctamente antes de guardarlos en la base de datos.

5.2. Validación en los controladores
Añadí una capa adicional de validación y sanitización en los controladores, lo que previene el ingreso de datos maliciosos.
Esto es especialmente importante en funcionalidades como la creación de cuentas y la publicación de clases.

5.3. Validación en el lado del cliente
Los formularios en el frontend también tienen validaciones básicas utilizando JavaScript, lo que proporciona feedback inmediato a los usuarios si los datos ingresados no son correctos.