Proyecto Node.js con Express y MongoDB (MADS)

Modular
Api
Data
Service

Este proyecto es una aplicación backend construida con Node.js, utilizando Express como framework web y MongoDB para la base de datos. La aplicación está diseñada para ser modular, con una estructura que separa las preocupaciones lógicamente en módulos, controladores, modelos, y rutas. Además, implementa funcionalidades básicas de autenticación de usuarios.

1. Estructura del Proyecto

La aplicación sigue una estructura de directorios modular para facilitar la escalabilidad y el mantenimiento:

- `src/`: Contiene el código fuente de la aplicación.
    - `config/`: Configuraciones del proyecto, incluyendo la conexión a la base de datos.
    - `modules/`: Cada módulo representa una funcionalidad distinta (por ejemplo, login).
        - `login/`: Contiene todo lo relacionado con la autenticación y registro de usuarios.
            - `controllers/`: Lógica de negocio para el manejo de solicitudes.
            - `models/`: Esquemas de Mongoose para los modelos de datos.
            - `routes/`: Definición de rutas de Express específicas del módulo.
    - `app.js`: Punto de entrada principal de la aplicación que configura el servidor Express.
    - `server.js`: Inicializa y escucha en el puerto definido.
- `.env`: Almacena variables de entorno, como la cadena de conexión a MongoDB.
- `package.json`: Define las dependencias y scripts del proyecto.

2. Características Principales

- **Conexión a MongoDB**: Utiliza Mongoose para conectar con MongoDB Atlas, permitiendo operaciones de base de datos.
- **Autenticación de Usuarios**: Implementa registro y CRUD de usuarios con hashing de contraseñas y verificacion MX de correo.

- **Modularidad**: Estructura el proyecto por módulos para mantener una separación clara de responsabilidades.
- **Creación de modulos**: La base de los modulos se puede crear por comando, manteniendo la estructura de archivos y nomenclatura.
- **Logging**: Incorpora Morgan para el registro de solicitudes HTTP, facilitando el debugging y monitoreo.

3. Herramientas y Tecnologías

- Node.js
- Express
- MongoDB & Mongoose
- bcryptjs para hashing de contraseñas.
- jsonwebtoken para manejo de tokens JWT.
- Morgan para logging.

4. Configuración y Uso

1. **Instalación de Dependencias**: Ejecuta `npm install` para instalar las dependencias necesarias.
2. **Variables de Entorno**: Configura las variables de entorno necesarias en el archivo `.env`.
3. **Ejecución del Proyecto**: Utiliza `npm run dev` para iniciar el servidor con `nodemon`, permitiendo el desarrollo con recarga automática.
4. **Estructura de Clases y Snippets**: Implementa clases y snippets para mejorar la estructura y mantenibilidad del código.

4.1. Creación de Módulos Automatizada

Para facilitar la expansión y mantenimiento de nuestro proyecto, hemos implementado un comando personalizado que permite la creación automática de módulos con una estructura de archivos predefinida. Este comando simplifica el proceso de añadir nuevas funcionalidades al proyecto, asegurando la coherencia en la estructura de directorios y la nomenclatura de archivos.

Para crear un nuevo módulo, simplemente ejecuta el siguiente comando desde la raíz del proyecto:

```
npm run create-module <nombreDelModulo>
```

Reemplaza `<nombreDelModulo>` con el nombre que deseas darle a tu nuevo módulo. El comando generará automáticamente una nueva carpeta bajo `src/modules`, conteniendo los archivos básicos del módulo, asi como un CRUD sencillo, cada uno con una plantilla inicial básica.

Este enfoque no solo mejora la eficiencia al evitar la creación manual de archivos y directorios comunes sino que también promueve una estructura de proyecto uniforme, facilitando la colaboración y el mantenimiento a largo plazo.

5. Desarrollo Futuro

El proyecto está configurado para facilitar la adición de nuevas funcionalidades y módulos, siguiendo los principios de diseño y arquitectura ya establecidos.

6. Actualizaciones

6.1. 14/02/2024 - User Crud

Este proyecto ha sido actualizado para incluir mejoras significativas en la estructura del código, manejo de errores, y la implementación de funcionalidades adicionales. A continuación, se detallan los cambios realizados.

- Refactorización de la Función `endpointConfig`
  Se refactorizó la función `endpointConfig` para mejorar la claridad y separar responsabilidades. Esto se logró dividiendo la lógica en funciones más pequeñas y descriptivas, lo que facilita el mantenimiento y la comprensión del código.

- Manejo de Errores de Registro de Usuarios
  Se implementó una verificación adicional para evitar errores de duplicación al registrar nuevos usuarios. Esto incluye comprobaciones para asegurar que los valores únicos, como el nombre de usuario o el correo electrónico, no se repitan en la base de datos.

- Validación y Verificación de Correos Electrónicos
  Se añadieron funciones para validar la estructura de las direcciones de correo electrónico mediante expresiones regulares y para verificar la existencia de registros MX, garantizando así que los correos proporcionados son válidos y capaces de recibir mensajes.

- Organización del Código en Archivos Separados
  Para mejorar la organización del código y separar las responsabilidades, se dividió la lógica del controlador de usuarios en varios archivos según su funcionalidad: creación/registro, obtención, actualización, y eliminación de usuarios. Esto permite una mejor gestión y mantenimiento del código.

- CRUD de Usuarios
  Se implementaron todas las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para los usuarios, permitiendo una gestión completa de estos en la aplicación. Cada operación está claramente definida y accesible a través de endpoints específicos en la API.

- Mejoras en el Manejo de Promesas y Async/Await
  Se ajustaron varias funciones para utilizar correctamente las promesas y la sintaxis async/await, asegurando así un manejo adecuado de las operaciones asíncronas y mejorando la legibilidad del código.

6.2. 15/02/2024 - Autenticación con Códigos de Verificación

- Se ha desarrollado un sistema de autenticación que genera códigos de 4 dígitos aleatorios. Estos códigos son válidos solo por 2 minutos, tras lo cual son automáticamente eliminados de la base de datos.
- El sistema está diseñado para devolver el código generado directamente a la página, en lugar de enviarlo por correo electrónico. Esto actúa como una forma básica de reCAPTCHA, mejorando la

seguridad y la verificación de los usuarios en tiempo real.

6.2.1. Configuración de Resender para Formularios de Contacto

- Se configuró Resender para redirigir los formularios de contacto recibidos al correo personal. Esto facilita la gestión de comunicaciones entrantes, permitiendo una respuesta rápida y eficiente a las consultas de los usuarios.

6.2.2. Detalles Técnicos

- Códigos de Verificación: Se implementó una función en JavaScript para generar códigos aleatorios de 4 dígitos. Posteriormente, se añadió lógica para eliminar estos códigos de la base de datos después de 2 minutos, usando setTimeout junto con operaciones async/await para manejar las acciones de eliminación conforme a las prácticas recomendadas por Mongoose.
- Manejo de Formularios de Contacto: Se estableció una configuración específica para que los formularios de contacto sean enviados automáticamente al correo personal mediante Resender. Esto mejora la interacción con los usuarios al asegurar que sus mensajes sean revisados y atendidos de manera prioritaria.

6.3. 15/02/2024 - Verificacion con JWT