Servicio Nacional de Aprendizaje

Centro Industrial del Diseño y la Manufactura.

2879871 programación de software

Aprendiz: Jair Coral

Docente: Guillermo Bejerano

MANUAL TÉCNICO APLICACIÓN GESTION DE EMPLEADOS

Floridablanca
Agosto 2024

Manual Técnico del Proyecto: Módulo de Gestión de Empleados

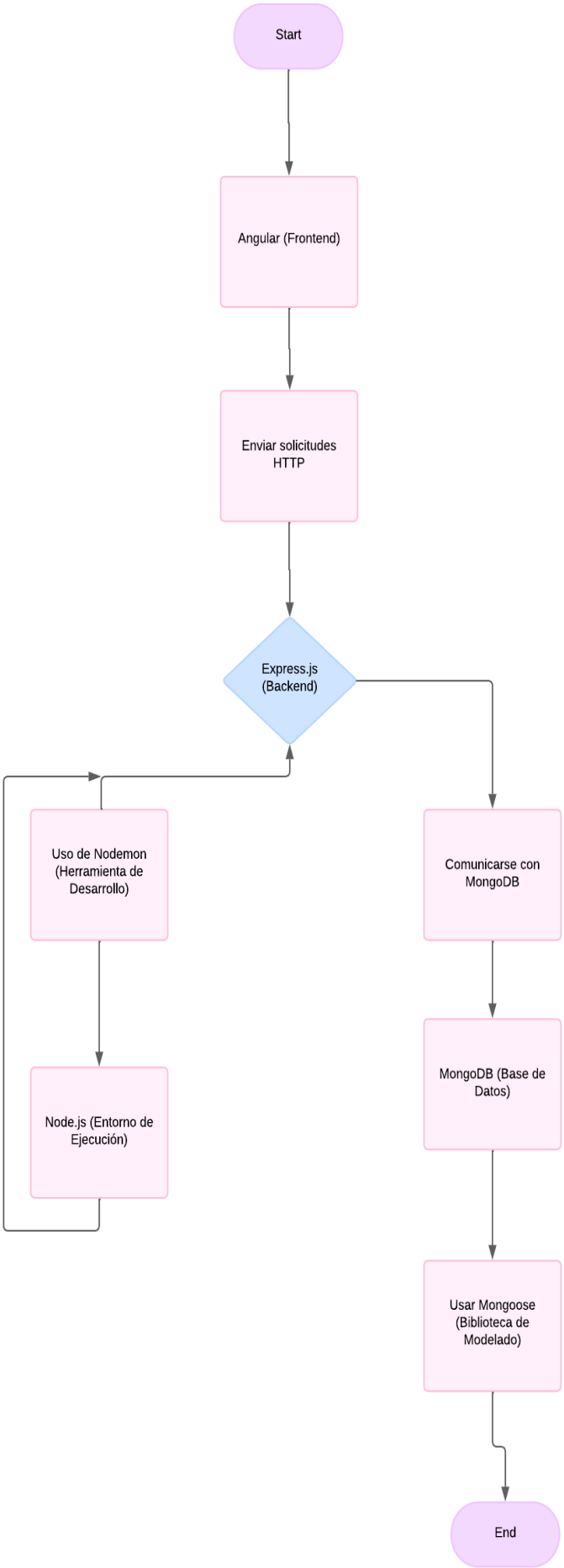
1. Introducción:

Este proyecto consiste en la construcción de una API para el módulo de Gestión de Empleados, utilizando el stack tecnológico MEAN (MongoDB, Express, Angular, Node.js). La aplicación permite realizar todas las operaciones de un CRUD (Create, Read, Update, Delete) con la información almacenada en una base de datos NoSQL (MongoDB). La aplicación se diseñó en base a los requisitos proporcionados por el instructor.

2. Arquitectura del Sistema

2.1 Diagrama de Arquitectura

Descripción: La aplicación está dividida en dos partes principales: el frontend, desarrollado con Angular, y el backend, desarrollado con Node.js y Express, que interactúa con una base de datos MongoDB:


Start
Angular (Frontend)
Enviar solicitudes HTTP
Express.js (Backend)
Uso de Nodemon (Herramienta de Desarrollo)
Node.js (Entorno de Ejecución)
Comunicarse con MongoDB
MongoDB (Base de Datos)
Usar Mongoose (Biblioteca de Modelado)
End

3. Backend

3.1 Descripción General

- **Lenguaje de programación:** JavaScript.
- **Framework utilizado:** Express.js.
- **Base de datos:** MongoDB.

3.2 Estructura del Proyecto

```
gestion-empleados/
 ├── backend/
|   ├── controllers/
|   |   └── empleado.controller.js
|   ├── models/
|   |   └── empleado.js
|   ├── routes/
|   |   └── empleado.routes.js
|   ├── apiTests.md
|   ├── database.js
|   ├── index.js
|   └── package.json
 ├── frontend/
```

- **Controllers:**
  - **empleado.controller.js:** Define la lógica de control para manejar las solicitudes relacionadas con los empleados. Aquí procesas las operaciones CRUD (crear, leer, actualizar, eliminar) y te comunicas con los modelos para gestionar los datos.
- **Models:**
  - **empleado.js:** Define el esquema del modelo de empleado, que representa cómo se almacenan los datos de empleados en la base de datos MongoDB. Incluye campos como nombre, salario, y otros atributos de un empleado.
- **Routes:**
  - **empleado.routes.js:** Define las rutas de la API para las operaciones relacionadas con los empleados. Es el punto de entrada que conecta las solicitudes HTTP (GET, POST, PUT, DELETE) con las funciones del controlador.
- **Archivos adicionales:**
  - **database.js:** Configuración de la conexión con la base de datos MongoDB. Nota: El URI de la base de datos se almacena directamente en este archivo por razones educativas. En futuras actualizaciones se planea mover esta configuración a un archivo .env.
- **index.js:** Punto de entrada del backend donde se inicializa el servidor Express y se configuran las rutas de la aplicación.

- **apiTests.md:** Contiene información o pruebas sobre cómo interactuar con la API o sobre el proceso de testing de las rutas.

3.3 Instalación y Configuración

Requisitos previos: Node.js, MongoDB.

Instrucciones de instalación:

1. Clonar el repositorio:

   git clone https://github.com/MrDemon97/Gestion-Empleados-SENA.git

2. Instalar las dependencias:

```
npm install express morgan cors mongoose

npm install --save-dev nodemon
```

3. Configurar las variables de entorno (por ejemplo, la URI de MongoDB) en un archivo .env (a implementar en futuras actualizaciones, actualmente se almacenan en database.js).
4. Ejecutar el servidor:

```
npm run dev
```

3.4 Endpoints de la API

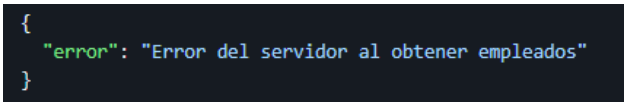
Endpoint principal: http://localhost:3000/api/empleados

3.4.1 PRUEBAS PARA GET

- **GET /empleados**
  - **Petición**: GET http://localhost:3000/api/empleados
  - **Descripción**: Devuelve todos los empleados de la base de datos. En caso de error, se muestra un mensaje correspondiente.

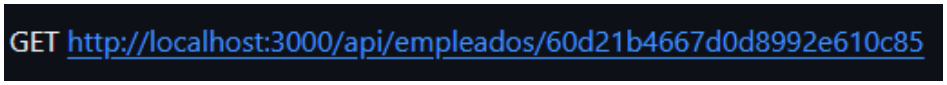
- **Ejemplo de respuesta exitosa**:

- **Ejemplo de respuesta de error**:



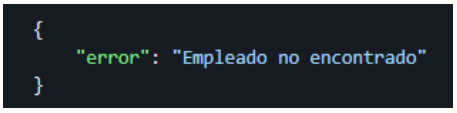
- **GET /empleados/:id**

  - **Petición**: GET http://localhost:3000/api/empleados/:id
  - **Descripción**: Devuelve los detalles de un único empleado, identificado por su ID.
  - **Ejemplo de petición con id**



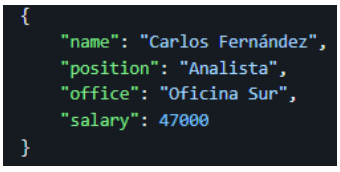
  - **Ejemplo de respuesta exitosa**:

  - **Ejemplo de respuesta de error**:



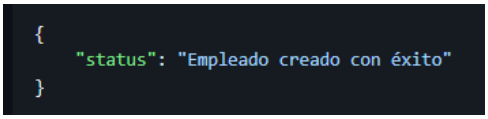
3.4.2 PRUEBAS PARA POST

- **POST /empleados**
  - **Petición**: POST http://localhost:3000/api/empleados
  - **Cuerpo de la solicitud**:

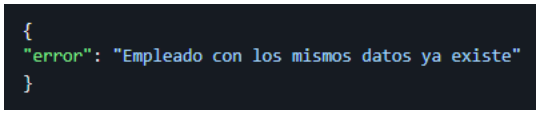


  - **Descripción**: Crea un nuevo empleado. Si ya existe un empleado con los mismos datos, se devuelve un mensaje de error.

- **Ejemplo de respuesta exitosa**:



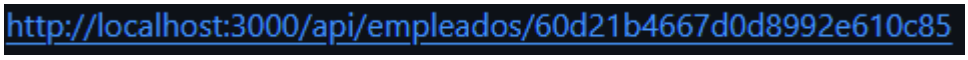
- **Ejemplo de respuesta de error**:



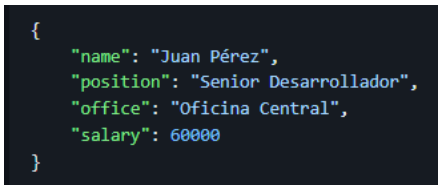
3.4.3 PRUEBAS PARA PUT

- **PUT /empleados/:id**

  - **Petición**: PUT http://localhost:3000/api/empleados/:id
  - **Ejemplo de petición PUT**:

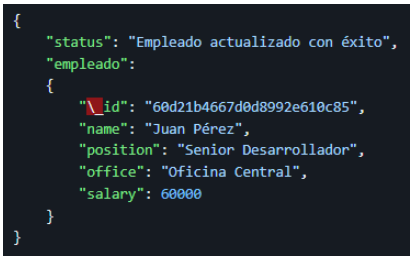


  - **Cuerpo de la solicitud**:

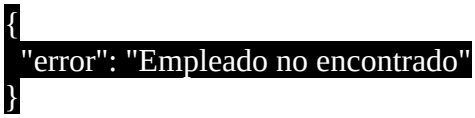


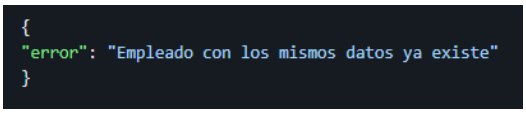
  - **Descripción**: Actualiza un empleado existente. Si el empleado no existe, se devuelve un mensaje de error.

  - **Ejemplo de respuesta exitosa**:



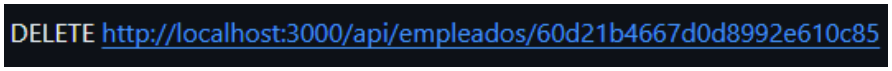
  - **Ejemplo de respuesta de error**:



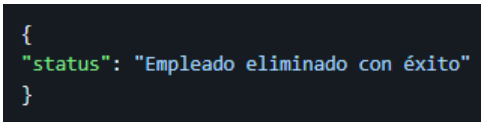


3.4.4 PRUEBAS PARA DELETE

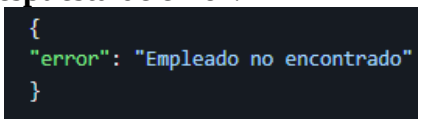
- **DELETE /empleados/:id**
    - **Petición**: DELETE http://localhost:3000/api/empleados/:id
    - **Ejemplo petición DELETE**:



    - **Descripción**: Elimina un empleado por su ID.
    - **Ejemplo de respuesta exitosa**:



    - **Ejemplo de respuesta de error**:



3.5 Conexión a la Base de Datos

- **Configuración**: El archivo database.js maneja la conexión a MongoDB. Se espera mover la configuración al archivo .env en futuras actualizaciones.

4. Frontend

4.1 Descripción General

- **Lenguaje de programación**: TypeScript.
- **Framework/Biblioteca**: Angular.

4.2 Estructura del Proyecto

```
gestion-empleados/
├── backend/
├── frontend/
│   ├── public/
│   └── src/
│       ├── app/
│       │   ├── components/
│       │   │   └── empleados/
│       │   │       ├── empleados.component.css
│       │   │       ├── empleados.component.html
│       │   │       ├── empleados.component.spec.ts
│       │   │       └── empleados.component.ts
│       │   ├── models/
│       │   │   ├── empleado.spec.ts
│       │   │   └── empleado.ts
│       │   ├── services/
│       │   │   ├── empleado.service.spec.ts
│       │   │   └── empleado.service.ts
│       │   ├── app-routing.module.ts
│       │   ├── app.component.css
│       │   ├── app.component.html
│       │   ├── app.component.spec.ts
│       │   ├── app.component.ts
│       │   ├── app.config.ts
│       │   ├── app.module.ts
│       │   ├── app.routes.ts
│       │   └── index.html
│       ├── main.ts
│       ├── styles.css
│       ├── angular.json
│       ├── package-lock.json
│       ├── package.json
│       ├── tsconfig.app.json
│       ├── tsconfig.json
│       └── tsconfig.spec.json
├── .editorconfig
├── .gitignore
├── README.md
└── node_modules/
```

- **src/app**:
  - **components/empleados**:
    - empleados.component.css: Estilos específicos para el componente de empleados.
    - empleados.component.html: Define la vista del componente de empleados.
    - empleados.component.ts: Lógica del componente Angular, donde defines las propiedades y métodos que interactúan con la vista.
    - empleados.component.spec.ts: Pruebas unitarias para el componente de empleados.
  - **models**:
    - empleado.ts: Define la estructura del modelo de datos en el frontend, asegurando consistencia con el backend.
    - empleado.spec.ts: Pruebas unitarias para el modelo de empleado.
  - **services**:
    - empleado.service.ts: Métodos que interactúan con el backend para gestionar empleados.
    - empleado.service.spec.ts: Pruebas unitarias para el servicio de empleados.
  - **app-routing.module.ts**: Define las rutas de la aplicación.
  - **app.component.ts**: Componente principal que controla la estructura general de la aplicación.
  - **styles.css**: Hoja de estilos global.
- **Archivos de Configuración**:
  - angular.json: Configuración para la construcción y despliegue de la aplicación Angular.
  - tsconfig.json: Configuración del compilador TypeScript.

4.3 Instalación y Configuración

Requisitos previos: Node.js, Angular CLI.

Instrucciones de instalación:

1. Ir a la carpeta Frontend

```
cd frontend
```

2. Instalar las dependencias

```
npm install -g @angular/cli
```

3. Iniciar el Servidor.

```
ng serve
```

4.4. Interacción de Componentes con el Backend a través de Servicios HTTP

La interacción entre los componentes y el backend se realiza mediante el servicio EmpleadoService. Este servicio gestiona las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) a través de solicitudes HTTP. A continuación, se detalla cómo los componentes envían y reciben datos desde el backend utilizando el servicio:

4.4.1 EmpleadoService

El servicio EmpleadoService es responsable de realizar las solicitudes HTTP al backend, específicamente a la API de empleados ubicada en la URL http://localhost:3000/api/empleados. Utiliza el módulo HttpClient de Angular para gestionar estas solicitudes, y maneja errores a través de catchError de RxJS, lo que permite capturar y propagar errores de manera adecuada.

Métodos del Servicio

- **getEmpleados**: Envía una solicitud GET para obtener la lista de empleados desde el backend.

```
getEmpleados() {
  return this.http.get(this.URL_API).pipe(
    catchError(err => {
      return throwError (() => new Error('Error al cargar empleados'));
    })
  );
}
```

- **postEmpleado**: Envía una solicitud POST para crear un nuevo empleado. Si se detecta un error, como un empleado duplicado, se gestiona con un mensaje adecuado.

```
postEmpleado(empleado: Empleado) {
  return this.http.post(this.URL_API, empleado). pipe(
    catchError(err => {
      if(err.status === 400){
        return throwError(() => new Error('Empleado con los mismos datos ya existe'));
      }
      return throwError (() => new Error('Error al crear empleado'));
    })
  );
}
```

- **putEmpleado**: Envía una solicitud PUT para actualizar los datos de un empleado existente.

```
putEmpleado(empleado: Empleado): Observable<any> {
  return this.http.put<any>(`${this.URL_API}/${empleado._id}`, empleado).pipe(
    catchError(err => {
      if(err.status === 400){
        return throwError(() => new Error('Empleado con los mismos datos ya existe'));
      }
      return throwError (() => new Error('Error al actualizar empleado'));
    })
  );
}
```

- **deleteEmpleado**: Envía una solicitud DELETE para eliminar un empleado basado en su ID.

5.2 EmpleadosComponent

El componente EmpleadosComponent es responsable de interactuar con el servicio EmpleadoService y renderizar los datos en la interfaz de usuario

Este manual técnico proporciona una guía completa para la configuración, instalación y uso del proyecto de Gestión de Empleados. Se actualizará con mejoras como la implementación de archivos .env para el almacenamiento seguro de configuraciones sensibles en futuras versiones.