

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

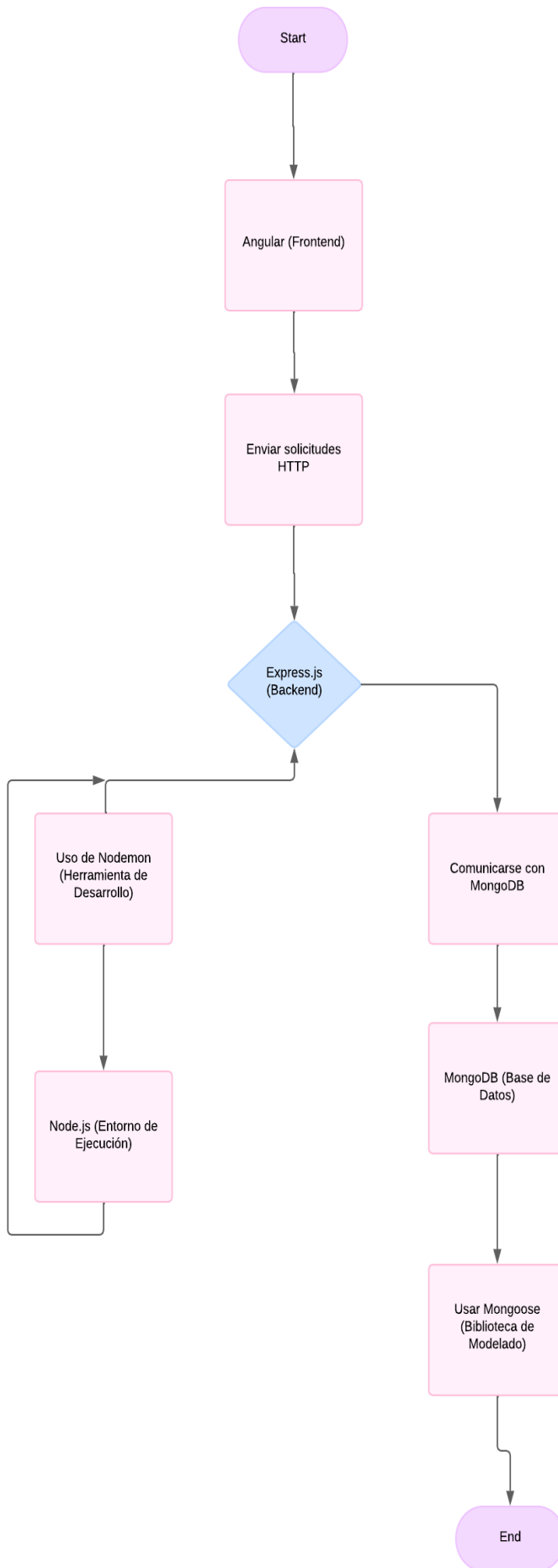
1. Introducción:

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:

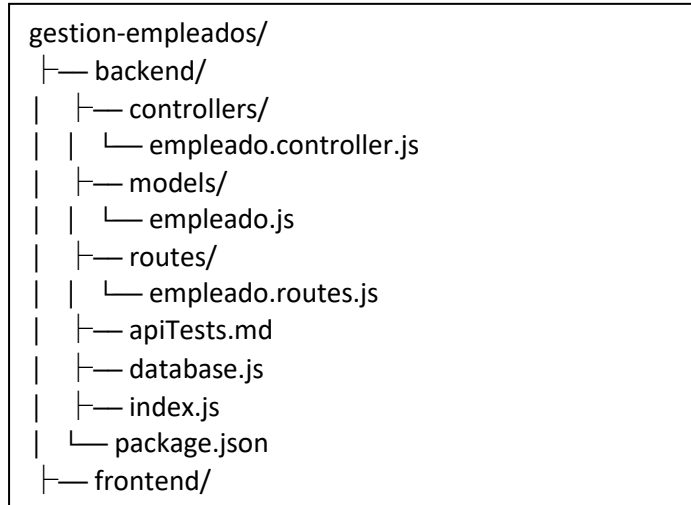


3. Backend

3.1 Descripción General

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

3.2 Estructura del Proyecto



- **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:**

```
[
  {
    "_id": "60d21b4667d0d8992e610c85",
    "name": "Juan Pérez",
    "position": "Desarrollador",
    "office": "Oficina Central",
    "salary": 50000
  },
  {
    "_id": "60d21b4667d0d8992e610c86",
    "name": "Ana Gómez",
    "position": "Diseñadora",
    "office": "Oficina Norte",
    "salary": 45000
  }
]
```

- **Ejemplo de respuesta de error:**

```
{
  "error": "Error del servidor al obtener empleados"
}
```

- **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**

GET <http://localhost:3000/api/empleados/60d21b4667d0d8992e610c85>

- **Ejemplo de respuesta exitosa:**

```
{
  "id": "60d21b4667d0d8992e610c85",
  "name": "Juan Pérez",
  "position": "Desarrollador",
  "office": "Oficina Central",
  "salary": 50000
}
```

- **Ejemplo de respuesta de error:**

```
{
  "error": "Empleado no encontrado"
}
```

3.4.2 PRUEBAS PARA POST

- **POST /empleados**

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

```
{
  "name": "Carlos Fernández",
  "position": "Analista",
  "office": "Oficina Sur",
  "salary": 47000
}
```

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

- Ejemplo de respuesta exitosa:

```
{
  "status": "Empleado creado con éxito"
}
```

- Ejemplo de respuesta de error:

```
{
  "error": "Empleado con los mismos datos ya existe"
}
```

3.4.3 PRUEBAS PARA PUT

- PUT /empleados/:id

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

<http://localhost:3000/api/empleados/60d21b4667d0d8992e610c85>

- Cuerpo de la solicitud:

```
{
  "name": "Juan Pérez",
  "position": "Senior Desarrollador",
  "office": "Oficina Central",
  "salary": 60000
}
```

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

```
{
  "status": "Empleado actualizado con éxito",
  "empleado": {
    "id": "60d21b4667d0d8992e610c85",
    "name": "Juan Pérez",
    "position": "Senior Desarrollador",
    "office": "Oficina Central",
    "salary": 60000
  }
}
```

- Ejemplo de respuesta de error:

```
{
  "error": "Empleado no encontrado"
}
```

```
{
  "error": "Empleado con los mismos datos ya existe"
}
```

3.4.4 PRUEBAS PARA DELETE

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

```
DELETE http://localhost:3000/api/empleados/60d21b4667d0d8992e610c85
```

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

```
{
  "status": "Empleado eliminado con éxito"
}
```

- **Ejemplo de respuesta de error:**

```
{
  "error": "Empleado no encontrado"
}
```

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.

4.1.2 EmpleadosComponent

El componente EmpleadosComponent es responsable de interactuar con el servicio EmpleadoService y renderizar los datos en la interfaz de usuario. A continuación, se detalla cómo los métodos del servicio son utilizados en este componente.

- **Obtener Empleados:** Al inicializar el componente, se llama al método getEmpleados del servicio para obtener la lista de empleados desde el backend y mostrarla en la tabla.
- **Crear o Actualizar Empleados:** El formulario puede estar en modo de creación o edición de empleados. Dependiendo del estado (isEditing), el formulario llamará a postEmpleado o putEmpleado:
- **Eliminar Empleados:** Cuando se selecciona un empleado para eliminar, se llama al método deleteEmpleado del servicio:

De esta manera, EmpleadoService y EmpleadosComponent trabajan conjuntamente para permitir la interacción entre el frontend y el backend, facilitando la gestión de empleados en la aplicación.

5. Instrucciones página:

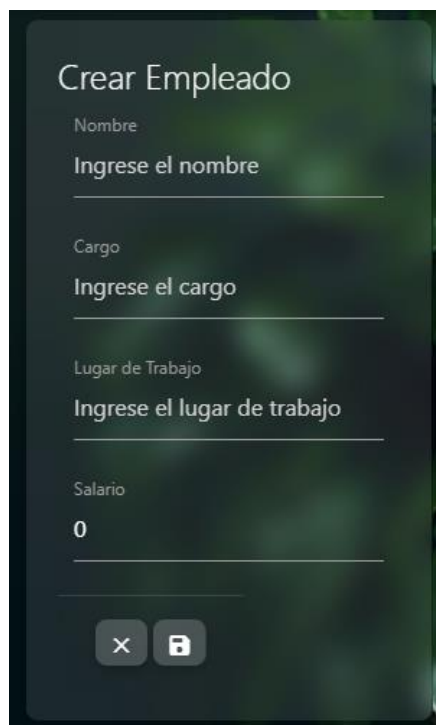
5.5.1 Uso de la Página

Esta sección proporciona una guía paso a paso sobre cómo utilizar la aplicación de gestión de empleados. A continuación, se describen las funcionalidades principales de la aplicación junto con capturas de pantalla para ilustrar cada paso.

5.5.2 Página de Inicio

En la página de inicio, encontrarás dos secciones principales: el formulario para crear o editar empleados y la lista de empleados.

5.5.3 Formulario para Crear o Editar Empleado

Una captura de pantalla de un formulario web con el título "Crear Empleado". El formulario tiene un fondo oscuro con un patrón de hojas verdes. Contiene cuatro campos de entrada: "Nombre" con el texto "Ingrese el nombre", "Cargo" con "Ingrese el cargo", "Lugar de Trabajo" con "Ingrese el lugar de trabajo", y "Salario" con el valor "0". En la parte inferior hay dos botones: uno con una 'X' para cerrar y otro con un ícono de candado para guardar.

5.5.3.1 Descripción: Este formulario permite ingresar los datos de un nuevo empleado o actualizar la información de un empleado existente.

5.5.3.2 Campos:

- **Nombre:** Campo para ingresar el nombre del empleado.
- **Cargo:** Campo para especificar el cargo del empleado.
- **Lugar de Trabajo:** Campo para indicar el lugar donde el empleado trabaja.
- **Salario:** Campo para ingresar el salario del empleado.

5.5.3.3 Botones:

- **-Limpiar:** Borra los datos ingresados en el formulario.
- **-Guardar:** Envía los datos al servidor para crear o actualizar el empleado.

5.5.4 Lista de Empleados



Nombre	Cargo	Oficina	Salario	Acciones
Jair Coral	Tecnico Asesor	Ipiales	300000	 
Guillermo Bejerano	Docente	Floridablanca	5350000	 
Fernando Perez	Contador	Gualmatan	2530000	 
Ernesto Perez	Guardabosques	Paramo de Sumapaz	1000000	 
Betty Pinzon	Contador	Bogotá	300000	 
Armando Mendoza	Presidente	Bogotá	9000000	 

5.5.4.1 Descripción: Muestra una tabla con la lista de todos los empleados registrados. Puedes ordenar la tabla por nombre, cargo, oficina y salario.

5.5.4.2 Acciones:

- **Editar:** Permite seleccionar un empleado para actualizar su información.
- **Eliminar:** Permite eliminar un empleado de la lista.

5.5.5 Interacción con la Aplicación

5.5.5.1 Agregar un Empleado

- Rellena los campos del formulario con la información del nuevo empleado.
- Haz clic en el botón "Guardar" para añadir el empleado a la lista.
- El nuevo empleado aparecerá en la tabla de la lista de empleados.

5.5.5.2 Editar un Empleado

- Haz clic en el icono de "editar" al lado del empleado que deseas modificar.
- El formulario se llenará automáticamente con los datos del empleado seleccionado.
- Modifica la información según sea necesario.
- Haz clic en el botón "Guardar" para actualizar la información del empleado.

- El empleado con la información actualizada aparecerá en la tabla de la lista de empleados

5.5.5.3 Eliminar un Empleado

- Haz clic en el icono de "eliminar" al lado del empleado que deseas borrar.
- Confirma la acción si es necesario.
- El empleado se eliminará de la lista.

5.5.6 Ordenar la Tabla de Empleados

5.5.6.1 **Descripción:** La tabla de empleados permite ordenar los datos por columnas específicas: Nombre, Cargo, Oficina y Salario.

5.5.6.2 Cómo Funciona:

- **Ordenar por Nombre:** Haz clic en el encabezado de la columna "Nombre" para ordenar la tabla alfabéticamente por el nombre de los empleados.
- **Ordenar por Cargo:** Haz clic en el encabezado de la columna "Cargo" para ordenar la tabla por el cargo de los empleados.
- **Ordenar por Oficina:** Haz clic en el encabezado de la columna "Oficina" para ordenar la tabla por la oficina en la que trabaja el empleado.
- **Ordenar por Salario:** Haz clic en el encabezado de la columna "Salario" para ordenar la tabla por el salario de los empleados.
- **Indicador de Ordenación:** Haz clic nuevamente en el encabezado para ordenar la orientación

6. Posibles mejoras a futuro

- **Despliegue en la nube:** Una de las mejoras más importantes sería desplegar la aplicación en la nube utilizando servicios como AWS, Heroku o Firebase. Esto permitiría que la aplicación sea accesible desde cualquier lugar y facilitaría la colaboración en tiempo real.
- **Seguridad y autenticación:** Implementar medidas de seguridad adicionales como autenticación y autorización para garantizar que solo los usuarios autorizados puedan acceder y modificar los datos de empleados.
- **Mejora de la interfaz de usuario:** Refinar la interfaz de usuario (UI) para hacerla más intuitiva y atractiva, incorporando animaciones y mejoras en la accesibilidad, como soporte para temas oscuros y claros.

- **Manejo avanzado de errores:** Mejorar el manejo de errores tanto en el frontend como en el backend, incluyendo la implementación de notificaciones para los administradores en caso de fallos críticos.

7. Conclusiones

El proyecto de Gestión de Empleados desarrollado durante la etapa de aprendizaje del programa Técnico en Programación del Servicio Nacional de Aprendizaje (SENA) ha permitido consolidar conocimientos en tecnologías MEAN (MongoDB, Express, Angular, Node.js), ofreciendo una aplicación funcional que facilita la gestión de empleados en una organización. Aunque el proyecto cumple con los requisitos básicos establecidos, existen varias áreas de mejora que pueden ser exploradas en el futuro para optimizar su desempeño, seguridad y usabilidad.

El aprendizaje obtenido a través de este proyecto no solo refuerza las habilidades técnicas, sino que también prepara para enfrentar desafíos más complejos en el desarrollo de software, fortaleciendo la capacidad de análisis, resolución de problemas y adaptación a nuevas tecnologías.