Manual de Procesos – Agenda de Contactos (Ionic + Angular + Backend)

Introduccion

El manual documenta los procesos técnicos necesarios para instalar, configurar y mantener la aplicación Agenda de Contactos, la cual está compuesta por dos componentes principales: un frontend desarrollado con Ionic + Angular y un backend personalizado, construido en este caso con Spring Boot.

Requisitos previos para el frontend

- Node.js
- npm instalados

Ionic CLI instalado globalmente:

abrimos la terminal dentro

verificamos si tenemos instalado ionic, sino, lo agregamos con el siguiente comando en la terminal.

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

Crear proyecto Ionic Angular

abrimos la terminal dentro de una carpeta nueva y escribimos el siguiente comando:

```
ionic start agenda-contactos blank --type=angular
cd agenda-contactos
```

Estructura del Proyecto

Frontend (Ionic + Angular)

```
src/
└── app/
    ├── home/                    # Módulo o página principal (puede ser landing o
dashboard)
    ├── interfaces/              # Interfaces TypeScript para tipado de datos
    │   ├── api-response-dto.ts  # Modelo para respuestas genéricas del backend
    │   └── contact.ts           # Modelo de datos para Contacto
    ├── pages/
    │   └── contact/             # Página principal de contactos
    │       ├── contact.page.html
    │       ├── contact.page.scss
```

```
    │       └── contact.page.ts
    ├── services/                # Servicios que se comunican con el backend
    │   ├── base.service.ts      # Servicio base con configuración común de
peticiones
    │   ├── contact.service.ts   # Lógica de negocio para contactos (CRUD)
    │   └── app.config.ts        # Configuración global (URL API, etc.)
    ├── app.component.*          # Componente raíz de la aplicación
    ├── app.routes.ts            # Definición de rutas y navegación
└── environments/
    ├── environment.ts           # Configuración para desarrollo
    └── environment.prod.ts      # Configuración para producción
```

Contenido de carpeta de eviroments

environment.prod.ts

La constante environment que muestras es parte de los archivos de entorno en Angular/Ionic, normalmente definidos en environment.ts y environment.prod.ts, y sirve para configurar valores según el entorno de ejecución (desarrollo o producción).

```
export const environment = {
  production: true,                        // Indica si es el entorno de
producción
  apiHost: 'https://fn2pmc21-9000.use.devtunnels.ms',  // Dirección del servidor
backend
  apiPort: '443',                          // Puerto usado por la API (opcional
si ya está en la URL)
  apiPrefix: '/api'                        // Prefijo común para las rutas del
backend
};
```

environment.ts

Define variables globales para el entorno de desarrollo, y es reemplazado automáticamente por environment.prod.ts al compilar con ng build --prod.

```
export const environment = {
  production: false,             // Marca que este es el entorno de desarrollo
  apiHost: 'http://localhost',   // Dirección base de la API (backend local)
  apiPort: '8082',               // Puerto donde corre el backend
  apiPrefix: '/api'              // Prefijo común para las rutas del backend
};
```

Cómo se usa en el código En un servicio como base.service.ts, podrías construir la URL completa así:

```
import { environment } from 'src/environments/environment';

const API_URL =
`${environment.apiHost}:${environment.apiPort}${environment.apiPrefix}`;
// Resultado: http://localhost:8082/api

this.http.get(`${API_URL}/contactos`);
```

De este modo, si luego compilas en producción, automáticamente se usará el archivo environment.prod.ts con la URL del backend desplegado (por ejemplo, en Azure, AWS o DevTunnels).

Uso del environment en la configuración global

En el archivo src/app/api.config.ts, se importa el archivo de entorno para construir dinámicamente la URL base (API_BASE_URL), usada por los servicios del frontend para comunicarse con el backend.

Código de api.config.ts

```
import { environment } from '../environments/environment';

const { apiHost, apiPort, apiPrefix } = environment;

export const API_BASE_URL =
  apiPort === '443'
    ? `${apiHost}${apiPrefix}`
    : `${apiHost}:${apiPort}${apiPrefix}`;
```

¿Qué hace este código?

- Desestructura los valores configurados desde el environment.
- Evita agregar el puerto cuando este es 443 (HTTPS por defecto), para no generar errores de conexión.
- Genera una URL base limpia y reutilizable que se adapta automáticamente al entorno actual.

Comando para arrancar el ionic

```
ionic serve
```

Flujo de Funcionamiento

1. contact.page.ts consume contact.service.ts para cargar o modificar contactos.
2. contact.service.ts usa base.service.ts para hacer peticiones HTTP genéricas (GET, POST, PUT, DELETE).

3. Las interfaces como contact.ts y api-response-dto.ts definen la forma de los datos intercambiados con el backend.
4. app.config.ts puede centralizar la URL base de la API y otras constantes.
5. environment.ts define el apiUrl usado por los servicios, facilitando el cambio entre desarrollo y producción.

Configuración del Backend (Spring Boot)

Requisitos previos

- Java 11 o superior.
- MySQL (o el gestor de base de datos que se esté utilizando).
- Maven o Gradle para gestionar dependencias.

Configuración de la Base de Datos

En el archivo src/main/resources/application.properties, debes definir la configuración de conexión a la base de datos MySQL y otros parámetros de la aplicación.

```
spring.application.name=parcial

# Puerto del servidor
server.port=8082

# Configuración de la base de datos MySQL
spring.datasource.url=jdbc:mysql://localhost:3308/parcial
spring.datasource.username=root
spring.datasource.password=abcd1234

# Driver de MySQL
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Activación de persistencia en la base de datos
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

# Configuración para la creación y eliminación automática de tablas
spring.jpa.hibernate.ddl-auto=create-drop

# Activación de la consola de logs para depurar errores
logging.level.org.hibernate.SQL=debug

# Configuración de Swagger con Springdoc OpenAPI para documentación de la API
springdoc.swagger-ui.path=/swagger-ui.html
```

Dependencias en pom.xml

Las dependencias necesarias para el proyecto se especifican en el archivo pom.xml. Asegúrate de incluir las siguientes dependencias para que el proyecto funcione correctamente.

```
<dependencies>
    <!-- Dependencia para acceso a datos JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- Dependencia para el desarrollo web en Spring Boot -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Dependencia para herramientas de desarrollo en tiempo de ejecución -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>

    <!-- Conector JDBC para MySQL -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>

    <!-- Dependencia para realizar pruebas en el proyecto -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>

    <!-- Dependencia para la integración de Swagger con Springdoc OpenAPI -->
    <dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
        <version>2.8.5</version>
    </dependency>
</dependencies>
```

Swagger para Documentación de la API

Si deseas exponer la documentación de la API de manera interactiva, puedes usar Swagger mediante Springdoc OpenAPI.

Con la configuración anterior en el archivo application.properties, puedes acceder a la documentación Swagger en la siguiente URL:

```
http://localhost:8082/swagger-ui.html
```

Estructura del Código

```
src/
└── main/
    └── java/
        └── com/
            └── ejemplo/
                └── agenda/
                    ├── common/
                    │   ├── base/
                    │   │   ├── ABaseController.java
                    │   │   ├── ABaseEntity.java
                    │   │   ├── ABaseService.java
                    │   │   └── IBaseService.java
                    │   └── dto/
                    │       └── ApiResponseDto.java
                    └── modules/
                        └── contacto/
                            ├── controller/
                            │   └── ContactoController.java
                            ├── service/
                            │   └── ContactoService.java
                            ├── repository/
                            │   └── ContactoRepository.java
                            └── entity/
                                └── Contacto.java
```

Clase Base ABaseController

La clase ABaseController es responsable de definir las operaciones básicas para las entidades: obtener, guardar, actualizar y eliminar registros. Aquí está la implementación general de esta clase:

```
public class ABaseController<T extends ABaseEntity, S extends IBaseService<T>> {
    // Métodos para operaciones CRUD
    @GetMapping
    public ResponseEntity<ApiResponseDto<List<T>>> findByStateTrue() {
        try {
            return ResponseEntity.ok(new ApiResponseDto<>("Datos obtenidos",
service.findByStateTrue(), true));
        } catch (Exception e) {
            return ResponseEntity.internalServerError().body(new ApiResponseDto<>
```

```
(e.getMessage(), null, false));
        }
    }
    // Métodos similares para show, save, update, delete
}
```

Clase Base ABaseEntity

La clase ABaseEntity define los campos comunes para todas las entidades, como id, status, createdAt, updatedAt, etc., y se utiliza como base para cada entidad específica.

```
@MappedSuperclass
public abstract class ABaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Boolean status = true;
    private LocalDateTime createdAt;
    private Long createdBy;
    // Otros campos comunes
}
```

Servicios Generales con ABaseService

La clase ABaseService maneja la lógica de negocio básica, como la validación de estado, creación, actualización y eliminación de registros. Se extiende para cada servicio específico de entidad.

```
public abstract class ABaseService<T extends ABaseEntity> implements
IBaseService<T> {
    protected abstract IBaseRepository<T, Long> getRepository();

    // Implementaciones de métodos CRUD
    public void update(Long id, T entity) throws Exception {
        // Lógica de actualización con auditoría
    }
}
```

Auditoría y Gestión de Estados

En el campo status de las entidades, se gestiona si un registro está activo o inactivo. Esta información se actualiza automáticamente cuando se elimina un registro. Además, se registran los cambios en los campos de auditoría, como createdBy, updatedBy, y deletedBy.

Método findByStateTrue

Este método se utiliza para obtener solo los registros activos, filtrando aquellos cuyo campo status esté marcado como true.

```
@Override
public List<T> findByStateTrue() {
    return getRepository().findAllByStatusTrue();  // Método para obtener solo
registros activos
}
```

Pruebas y Ejecución

Para iniciar el backend, se utiliza Maven o Gradle desde la terminal. El comando para ejecutar la aplicación con Maven es:

```
mvn spring-boot:run
```

Y para Gradle:

```
gradle bootRun
```

Conclusión

La aplicación Agenda de Contactos demuestra una integración efectiva entre tecnologías modernas de frontend y backend. El uso de Ionic + Angular permite una experiencia de usuario multiplataforma, mientras que Spring Boot garantiza una base sólida y escalable para la gestión de datos y servicios.

Gracias a la arquitectura modular, el uso de buenas prácticas como la reutilización de componentes y servicios, y la incorporación de herramientas como Swagger para la documentación de la API, este sistema es fácilmente mantenible y ampliable.