

**UNIVERSIDAD DE LAS FUERZAS  
ARMADAS-ESPE SEDE SANTO DOMINGO**

**DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS**

**CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN**

**PERIODO** : Noviembre 2023 – Marzo 2024

**ASIGNATURA** : Aplicaciones Distribuidas

**TEMA** : Proyecto Unidad 1

**NOMBRES** : Oscar Avellán

Katherine Bravo

Melany Vera

**NIVEL-PARALELO** : Séptimo

**DOCENTE** : Ing. Luis Chica

**FECHA DE ENTREGA** : 06/02/2024

**SANTO DOMINGO - ECUADOR**

**2023**

## **1. Introducción**

Este informe detalla el proceso de desarrollo de una aplicación web orientada a realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) almacenadas en memoria. Para la implementación de esta solución, se han utilizado tecnologías como Spring Boot para el desarrollo del BackEnd, y Angular para la creación del FrontEnd, ofreciendo una experiencia robusta y moderna en la gestión de datos.

El propósito fundamental de esta aplicación es proporcionar una interfaz interactiva que permita a los usuarios realizar operaciones básicas de gestión de datos, como la creación, lectura, actualización y eliminación de información. Este proyecto se enfoca en la implementación de una solución ágil y eficiente que no requiera una base de datos persistente, utilizando la memoria como almacenamiento temporal.

## **2. Sistemas de Objetivos**

### **2.1. Objetivo General:**

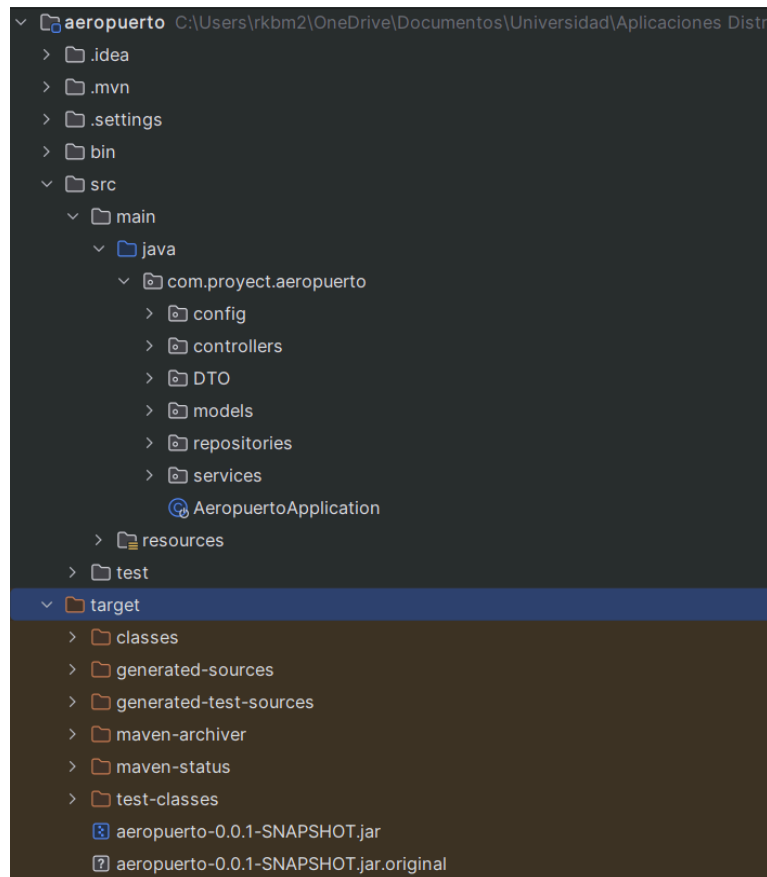
Desarrollar un sistema web que permita realizar CRUD y consumo de API desde un servicio a otro mediante el manejo del sistema distribuido con la tecnología de Spring Boot.

### **2.2. Objetivos Específicos:**

- Desarrollar la programación del BackEnd en Spring Boot
- Desarrollar la programación del FrontEnd en Angular
- Implementar Dockerfile y docker compose para los microservicios de postgres y mongoDB.
- Conectar una aplicación BackEnd de PostgreSQL con una aplicación BackEnd de MongoDB, para manejar nuestro aplicativo en general como un sistema distribuido.

### 3. Desarrollo

#### 3.1. BackEnd en Spring Boot en Postgrest



Cabe destacar que el diagrama de base de datos presentado a continuación, es meramente referencial y solo muestra los datos que manejaremos en nuestra aplicación, ya que en nuestro BackEnd de Spring Boot, no manejaremos base de datos, sino que todas las operaciones CRUD las ejecutaremos en memoria y representaremos los datos de las tablas de la BDD presentada por medio de modelos.

```

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "PilotoModelo")
public class PilotoModelo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String dni;
    private String name;
    private String lastname;
    private String address;
    private String birthday;
    private String fecha_emis_licencia;

    @OneToMany(mappedBy = "pilotoModelo", cascade = CascadeType.PERSIST)
    private List<AvionModelo> avionModelo;
}

```

```

@NoArgsConstructor
@Entity
@Table(name = "AvionModelo")
public class AvionModelo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String nombre_aeronave;
    private String company;
    private int capacidad_pasajeros;
    private int horas_vuelo;

    @ManyToOne()
    @JoinColumn(name = "piloto_id", nullable = false, referencedColumnName = "id")
    @JsonBackReference
    private PilotoModelo pilotoModelo;

    @Override
    public String toString() {
        // Excluir la propiedad que se refiere al PilotoModelo para evitar recursión inf
        return "AvionModelo{" +
            "id=" + id +
            ", nombre_aeronave='" + nombre_aeronave + '\'' +
            ", company='" + company + '\'' +
            ", capacidad_pasajeros=" + capacidad_pasajeros +
            ", horas_vuelo=" + horas_vuelo +
            ", piloto_id=" + pilotoModelo.getId() +
            '}';
    }
}

```

```

@RestController
@RequestMapping("/pilotos")
public class PilotoController {

    @Autowired
    private PilotoService pilotoService;

    @GetMapping()
    public List<PilotoModelo> getAll() { return pilotoService.getAll(); }

    @PostMapping()
    public Map<String, Object> create(@RequestBody PilotoModelo pilotoModelo) {
        try {
            pilotoService.save(pilotoModelo);
            return Map.of( k1: "result", v1: true, k2: "message", v2: "Piloto agregado correctamente");
        } catch (Exception e) {
            return Map.of( k1: "result", v1: false, k2: "message", v2: "Error: No se ha podido agregar");
        }
    }

    @PutMapping("/{pilotoId}")
    public Map<String, Object> update(@PathVariable Long pilotoId, @RequestBody PilotoModelo pilotoModelo) {
        try {
            if (pilotoService.update(pilotoId, pilotoModelo)) {
                return Map.of( k1: "result", v1: true, k2: "message", v2: "Piloto modificado correctamente");
            }
        }
    }
}

```

```

@RestController
@RequestMapping("/aviones")
public class AvionController {

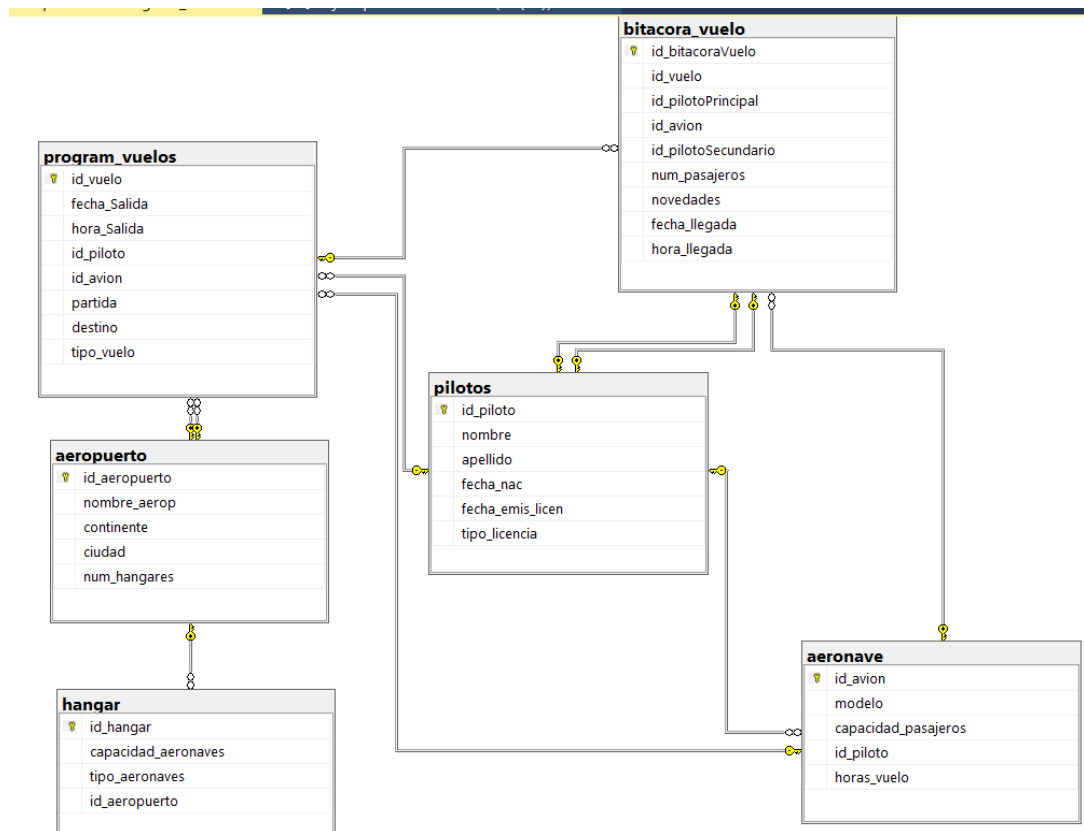
    @Autowired
    private AvionService avionService;

    @PostMapping
    public ResponseEntity<AvionModelo> ingresarAvion(@RequestBody AvionDTO avionDTO) {
        AvionModelo nuevoAvion = avionService.ingresarAvion(avionDTO);
        return new ResponseEntity<>(nuevoAvion, HttpStatus.CREATED);
    }

    @GetMapping("/{id}")
    public ResponseEntity<AvionModelo> obtenerAvionPorId(@PathVariable Long id) {
        Optional<AvionModelo> avion = avionService.obtenerAvionPorId(id);
        return avion.map(value -> new ResponseEntity<>(value, HttpStatus.OK))
            .orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @GetMapping
    public ResponseEntity<List<AvionPilotoDTO>> obtenerTodos() {
        List<AvionPilotoDTO> aviones = avionService.obtenerTodos();
        return new ResponseEntity<>(aviones, HttpStatus.OK);
    }
}

```



Para representar los datos usaremos una clase de Java para modelo donde usaremos anotaciones lombok para manejar métodos setters y getters. Además, también tendremos un constructor.

```

18 usages
@Data
public class PilotoModelo {
    private String id;
    private String dni;
    private String name;
    private String lastname;
    private String address;
    private String birthday;
    no usages
    public PilotoModelo(String dni, String name, String lastname, String address, String birthday) {
        this.id = GenericPilotoID.getNextId();
        this.dni = dni;
        this.name = name;
        this.lastname = lastname;
        this.address = address;
        this.birthday = birthday;
    }
}
  
```

En nuestro repositorio manejaremos las listas para agregar datos.

```

3 usages
public class PilotoRepository {

    1 usage
    private List<PilotoModelo> pilotoModelos = new ArrayList<>();

    9 usages
    public List<PilotoModelo> getAll() { return this.pilotoModelos; }
}

```

En servicio estableceremos las operaciones CRUD que se ejecutarán sobre los datos de un modelo.

```

2 usages
@Service
public class PilotoService {

    9 usages
    private PilotoRepository empRepository = new PilotoRepository();

    1 usage
    public List<PilotoModelo> getAll() { return empRepository.getAll(); }

    1 usage
    public boolean save(PilotoModelo pilotoModelo) {
        for(PilotoModelo emp: empRepository.getAll()) {
            if(emp.getDni().equals(pilotoModelo.getDni())) {
                return false;
            }
        }
        empRepository.getAll().add(pilotoModelo);
        return true;
    }

    1 usage
    public boolean update(String employeeId, PilotoModelo pilotoModelo) {
        for(PilotoModelo emp: empRepository.getAll()) {

```

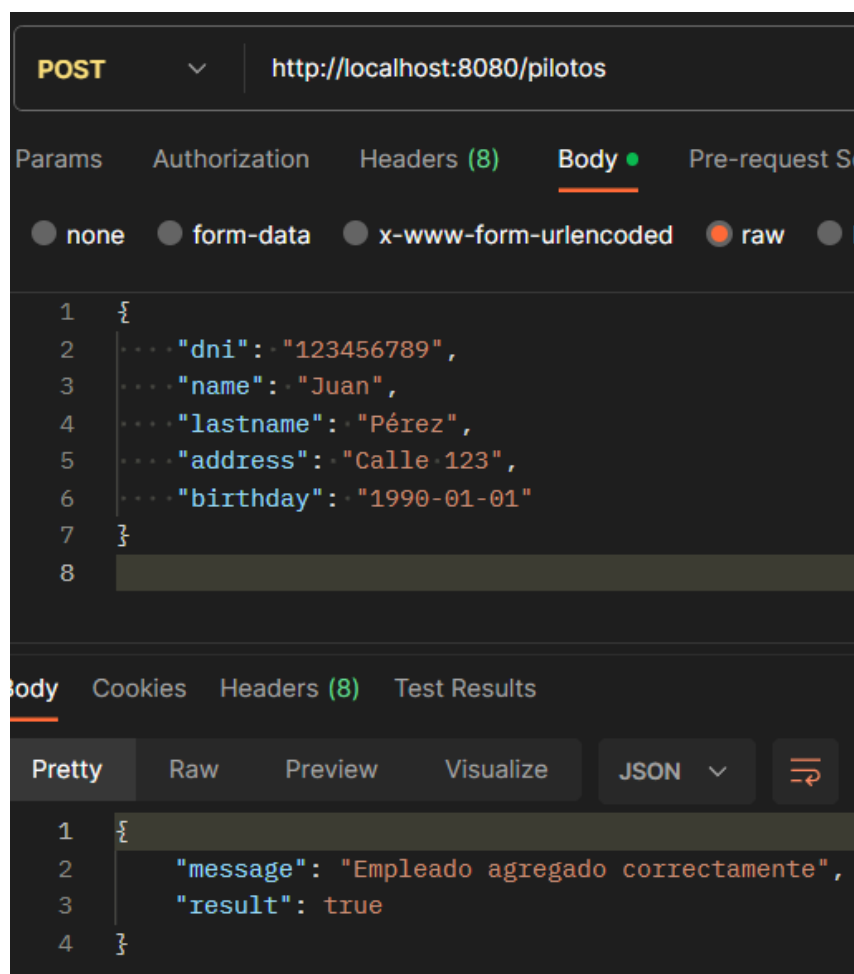
En el controlador manejaremos todos los endpoint que ejecutarán las operaciones CRUD definidas en los métodos del servicio.

```

@PostMapping()
public Map<String, Object> create(@RequestBody PilotoModelo pilotoModelo){
    try {
        Adult adult = new Adult();
        if(!adult.verify(pilotoModelo.getBirthday())) {
            return Map.of( k1: "result", v1: false, k2: "message", v2: "Empleado debe ser mayor de edad");
        }
        if(pilotoService.save(pilotoModelo)) {
            return Map.of( k1: "result", v1: true, k2: "message", v2: "Empleado agregado correctamente");
        } else {
            return Map.of( k1: "result", v1: false, k2: "message", v2: "Ya existe un empleado con esa cédula");
        }
    } catch(Exception e) {
        return Map.of( k1: "result", v1: false, k2: "message", v2: "Error: No se ha podido agregar al empleado");
    }
}

```

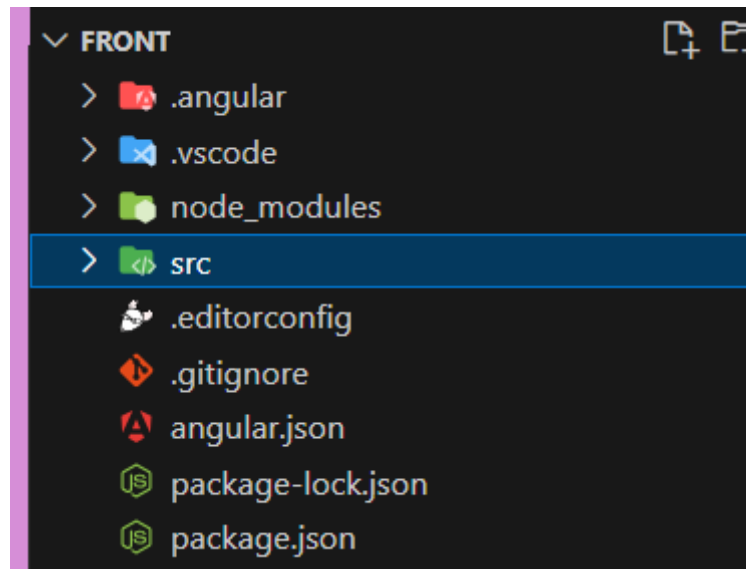
Cómo podemos ver ahora hemos hecho una prueba en Postman donde hemos agregado datos en Pilotos.



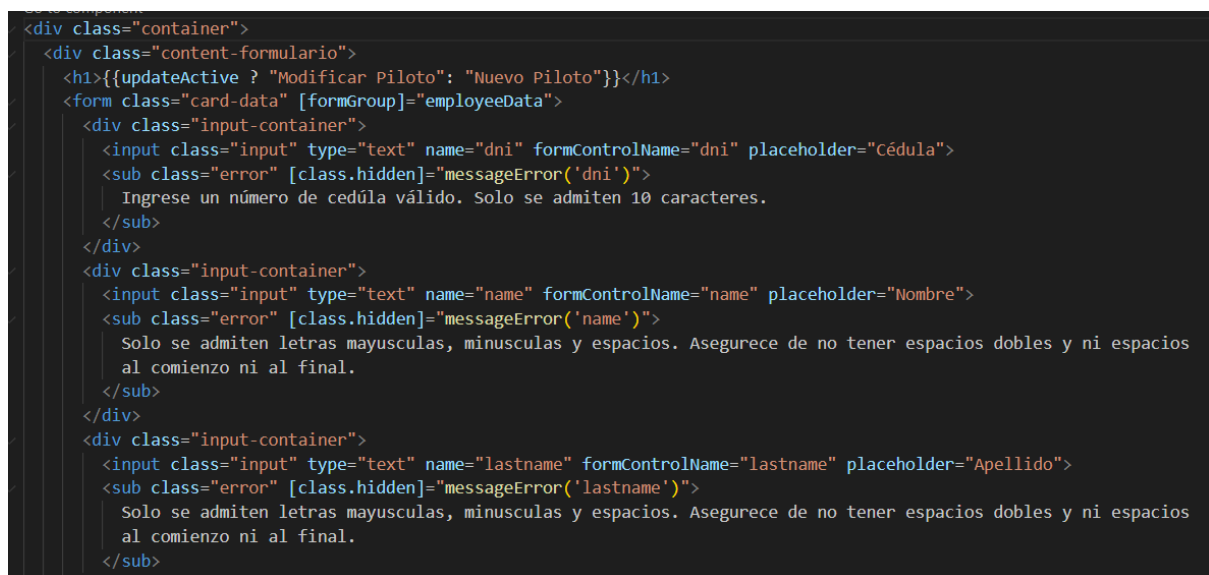


### 3.2. FrontEnd con Angular

Creamos un proyecto en Angular, el cuál nos servirá como FrontEnd para ejecutar las operaciones CRUD de nuestro proyecto de Spring Boot.



En este proyecto crearemos un componente por cada uno de los modelos que tengamos, por ejemplo en el html para el componente pilotos tendremos lo siguiente.



En el Typescript enlazaremos los datos y manejaremos validaciones.

```

uri: string = uri_local + "cards"
urlTipoAvion: string = uri_local + "tipoAvion"
urlPilotos: string = uri_local + "pilotos"
urlN: string = uri + "pilotos"
constructor(private springService: ApiService) {

  ngOnInit(): void {
    this.getPilotos()
  }

  employeeData: FormGroup = new FormGroup({
    dni: new FormControl('', [
      Validators.required,
      Validators.pattern(REGEX_FORM.isValidDNI)
    ]),
    name: new FormControl('', [
      Validators.required,
      Validators.pattern(REGEX_FORM.isValidNAME)
    ]),
    lastname: new FormControl('', [
      Validators.required,
      Validators.pattern(REGEX_FORM.isValidLASTNAME)
    ]),
    address: new FormControl('', [
      Validators.required,
      Validators.pattern(REGEX_FORM.isValidLASTNAME)
    ]),
    birthday: new FormControl('', [
      Validators.required
    ])
  })

```

Además añadiremos estilos en el archivo SCSS para tener una interfaz visualmente atractiva.

```

enviroment.ts x  piloto.component.scss x  app.co
c > app > components > piloto > piloto.component.scss > ..
4  @use ../../../../assets/scss/components/table
5
6  .container {
7    display: grid;
8    grid-template-columns: 1fr 2fr;
9    padding: 50px;
10   gap: 30px;
11
12   .content-formulario {
13     display: grid;
14     padding: 30px;
15     background-color: colors.$white;

```

En el archivo app.module.ts tendremos que asegurarnos que se haya importado de manera correcta todos nuestros componentes.

```
6 import { AvionComponent } from './components/avion/avion.component';
7 import { HttpClientModule } from '@angular/common/http';
8 import { PilotoComponent } from './components/piloto/piloto.component';
9 import { VueloComponent } from './components/vuelo/vuelo.component';
10 import { PresentacionComponent } from './components/presentacion/presentacion.component';
11
12 You, 3 hours ago | 2 authors (Gary and others)
13 @NgModule({
14   declarations: [
15     AppComponent,
16     AvionComponent,
17     PilotoComponent,
18     VueloComponent,
19     PresentacionComponent,
```

Ahora ya podremos ver la interfaz donde podremos agregar pilotos:

The screenshot shows the 'Aviacion Civil' application interface with the 'Pilotos' button selected in the navigation bar. The main content area is divided into two sections. On the left is a form titled 'Nuevo Piloto' with several input fields: a text field containing '2356395729', a text field containing 'Oscar', a text field for 'Apellido', a text field containing 'asdsd', and a date field with the placeholder 'dd/mm/aaaa'. Below the form are two buttons: 'Crear' (blue) and 'Cancelar' (red). On the right is a table titled 'Pilotos Registrados' with the following data:

N°	Cédula	Nombres	Apellido	Dirección	F. Nacimiento	Opciones
1	2300764533	asasdasd	Avellan	Santo Domingo	2000-12-15	<button>Editar</button> <button>Borrar</button>
2	21323	Juan	Pérez	Calle 123	1990-01-01	<button>Editar</button> <button>Borrar</button>

También podremos actualizar los pilotos

### Modificar Piloto

Asegurese de completar todos los campos

### Pilotos Registrados


N°	Cédula	Nombres	Apellido	Dirección	F. Nacimiento	Opciones
1	2300764533	asasdasd	Avellan	Santo Domingo	2000-12-15	<input type="button" value="Editar"/> <input type="button" value="Borrar"/>
2	21323	Juan	Pérez	Calle 123	1990-01-01	<input type="button" value="Editar"/> <input type="button" value="Borrar"/>

Por último también podremos eliminar Pilotos:

### Piloto

los campos

### Pilotos Registrados



#### ¿Deseas eliminar este piloto?

Recuerda que solo podrás eliminar el piloto si no tiene registros vinculados

De esta manera hemos concluido con el CRUD de Pilotos, pero cabe tomar en cuenta que Avión está relacionado con Pilotos por lo que para agregar un avión primero debemos agregar un piloto ya que necesitamos los datos de ahí.

## Nuevo Avión

Seleccionar piloto

Seleccionar piloto

Katherine Bravo

Crear

Cancelar

### Nuevo Avión

Seleccionar piloto

Crear

Cancelar

### Aviones Registrados

	Nombre	Compañía	Capacidad de Pasajeros	Horas de Vuelo	Piloto	Opciones
				15	Katherine Bravo	<div><div>Editar</div><div>Borrar</div></div>

✓

Avión insertado correctamente

OK

### Modificar Avión

Katherine Bravo

Actualizar

Cancelar

### Aviones Registrados

N°	Nombre	Compañía	Capacidad de Pasajeros	Horas de Vuelo	Piloto	Opciones
1	Luxys	Tesla	25	15	Katherine Bravo	<div><div>Editar</div><div>Borrar</div></div>

## Nuevo Aeropuerto

Ingresar el nombre del aeropuerto.

Ingresar el país del aeropuerto.

Ingresar la ciudad del aeropuerto.

Crear

Cancelar

### Aeropuertos Registrados

N°	Nombre	País	Ciudad	N° de Hangares	Opciones
1	Vuelos Seguros	Ecuador	Quito	15	<div>Editar</div> <div>Borrar</div>

Por medio de la siguiente estructura se muestra la configuración del archivo HTML donde contiene todas las rutas de navegación del sitio web de esta manera se podrá comprender cómo se encuentra estructurada la parte del frontend

```

<section class="content-navbar">
  <main class="navbar">
    <div class="identity-page">
      <h1>Aviacion Civil</h1>
    </div>

    <nav class="nav">
      <ul class="nav-list">
        <a routerLink="/inicio" routerLinkActive="active-section"
          [class.active-section]="activeLink.includes('/inicio')">
          <li>Inicio</li>
        </a>

        <a routerLink="/pilotos" routerLinkActive="active-section"
          [class.active-section]="activeLink.includes('/pilotos')">
          <li>Pilotos</li>
        </a>

        <a routerLink="/aviones" routerLinkActive="active-section"
          [class.active-section]="activeLink.includes('/aviones')">
          <li>Aviones</li>
        </a>

        <a routerLink="/aeropuertos" routerLinkActive="active-section"
          [class.active-section]="activeLink.includes('/aeropuertos')">
          <li>Aeropuertos</li>
        </a>

        <a routerLink="/vuelos" routerLinkActive="active-section"
          [class.active-section]="activeLink.includes('/vuelos')">
          <li>Vuelos</li>
        </a>

        <a routerLink="/tipoVuelo" routerLinkActive="active-tipoVuelo"
          [class.active-section]="activeLink.includes('/tipo')">
          <li>Tipos</li>
        </a>
      </ul>
    </nav>
  </main>
</section>

```

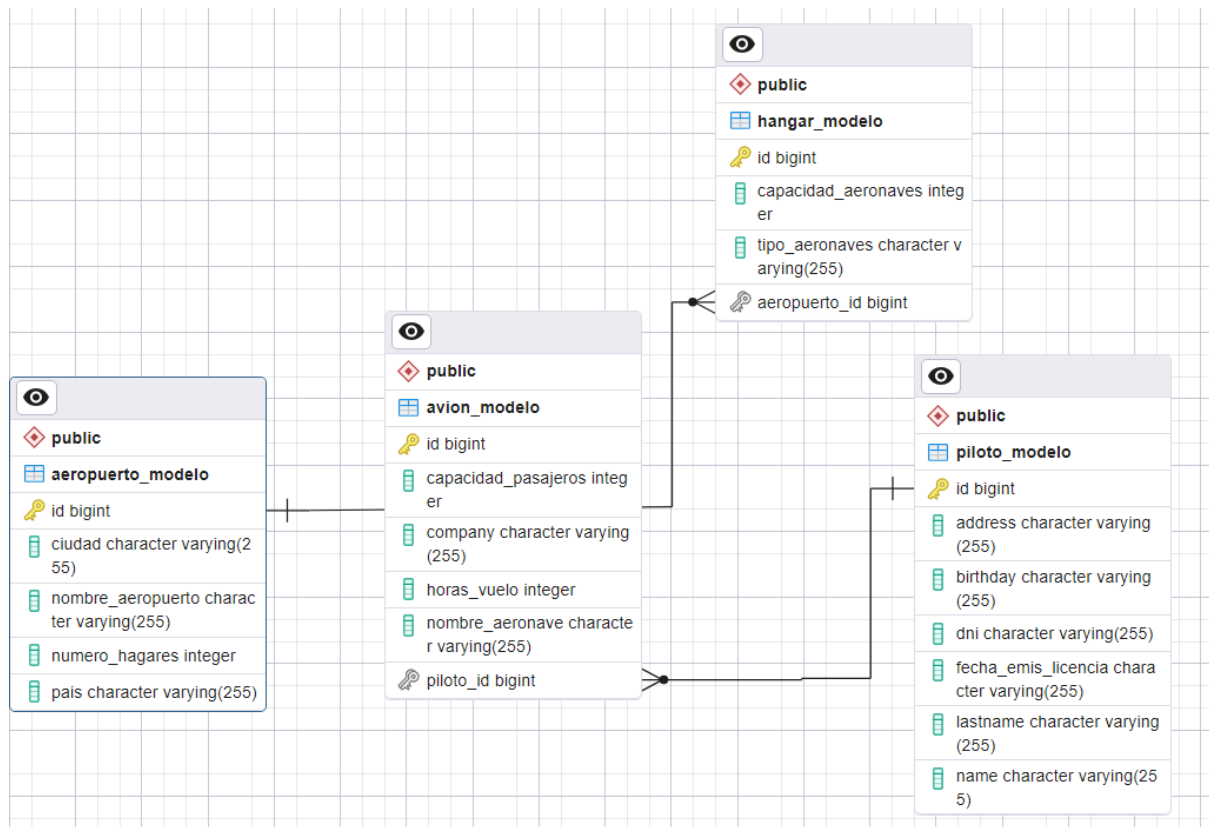
Esta exportación define constantes de entorno para un entorno de desarrollo. url apunta a una dirección IP local y puerto específico. uri\_local y uri\_local\_mongo apuntan a direcciones locales en puertos específicos para servicios web y bases de datos MongoDB, respectivamente. Estas constantes ayudan a configurar y acceder a recursos durante el desarrollo de aplicaciones

```
src > app > environment > ts enviroment.ts > ...
1   export const Enviroment = {
2     url: 'http://192.168.0.106:8080/',
3     uri_local: 'http://localhost:8080/',
4     uri_local_mongo: 'http://localhost:8090/',
5   }
6   |
```

Por medio de la siguiente configuración se puede establecer una conexión con el Frontend del sistema, para esto se deberá especificar el puerto en el cual se estará ejecutando el frontend y las solicitudes HTTP que se permitirá.

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    no usages
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping( pathPattern: "/*")
            .allowedOrigins("http://localhost:4200")
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowedHeaders("*");
    }
}
```



- **Docker del Microservicio de Postgres**

```

docker-compose.yml
1  version: '3'
2
3  services:
4    java_app:
5      container_name: java_app
6      image: pee-java-app:1.0.0
7      build: .
8      ports:
9        - 8080:8080
10
11     environment:
12       - DATABASE_URL=jdbc:postgresql://java_db:5432/postgres
13       - DATABASE_USERNAME=postgres
14       - DATABASE_PASSWORD=postgres
15       - DRIVER_NAME=org.postgresql.Driver
16     depends_on:
17       - java_db
18   java_db:
19     container_name: java_db
20     image: postgres:12
21     ports:
22       - 5432:5432
23     environment:
24       POSTGRES_USER: postgres
25       POSTGRES_PASSWORD: postgres
26       POSTGRES_DB: postgres
  
```



```
docker-compose.yaml Dockerfile x
1 FROM openjdk:17-jdk-alpine
2
3 # Copia todos los archivos del directorio actual al contenedor
4 COPY target/aeropuerto-0.0.1-SNAPSHOT.jar java-app.jar
5
6 # Comando para ejecutar la aplicación al iniciar el contenedor
7 ENTRYPOINT ["java", "-jar", "java-app.jar"]
8
```

### 3.3. Repositorio de Github

En Github crearemos un repositorio donde todos los colaboradores del proyecto realizarán los respectivos commits con las adiciones y modificaciones de código que realicen.

projectDistribuidas

main 1 Branch 0 Tags

Go to file Add file Code

melanyvera2630 Update Piloto 758db1c · 49 minutes ago 18 Commits

.mvn/wrapper	Mensaje descriptivo aquí	8 hours ago
Proyecto_Dsitribuidas	Mensaje descriptivo aquí	8 hours ago
bin	Mensaje descriptivo aquí	8 hours ago
src	Update Piloto	49 minutes ago
.gitignore	Mensaje descriptivo aquí	8 hours ago
mvnw	Mensaje descriptivo aquí	8 hours ago
mvnw.cmd	Mensaje descriptivo aquí	8 hours ago
pom.xml	Mensaje descriptivo aquí	8 hours ago

### 3.4 BackEnd en Spring Boot en Mongo

- Docker del microservicio de Mongo

```
docker-compose.yml x Dockerfile
1 version: '3'
2
3 services:
4   java_app_mongo:
5     container_name: java_app_mongo
6     image: pee-java-app-mongo:1.0.0
7     build: .
8     ports:
9       - 8090:8090
10    depends_on:
11      - mongo_db
12
13   mongo_db:
14     image: mongo:latest
15     ports:
16       - 27017:27017
```

```
docker-compose.yml x Dockerfile
1 FROM openjdk:17-jdk-alpine
2
3 # Copia todos los archivos del directorio actual al contenedor
4 COPY target/Aeropuerto-Mongo-0.0.1-SNAPSHOT.jar java-app-mongo.jar
5
6 # Comando para ejecutar la aplicación al iniciar el contenedor
7 ENTRYPOINT ["java", "-jar", "java-app-mongo.jar"]
```

Containers

[Give feedback](#)

Container CPU usage ⓘ

0.45% / 800% (8 CPUs available)

Container memory usage ⓘ

271.45MB / 3.44GB















[Show charts](#)

🔍

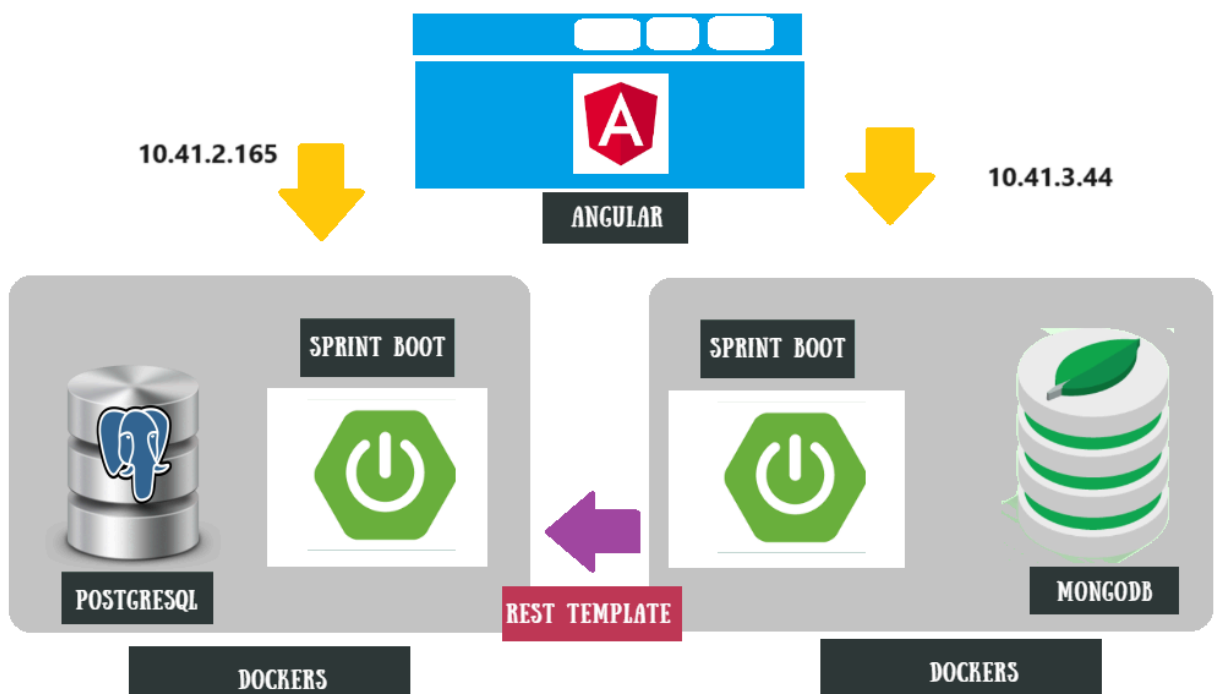
Search

☰

Only show running containers

<input type="checkbox"/>	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input type="checkbox"/>	<div><div>▼</div><div> <b>aeropuerto-mongo</b></div></div>		Exited	0%		47 minutes ago	<div><div>▶</div><div>⋮</div><div>🗑</div></div>
<input type="checkbox"/>	<div><div> <b>java_app_mongo</b></div><div>431e685c2de4 </div></div>	<a href="#">pee-java-app-mongo:1.0.0</a>	Exited (143)	0%	8090:8090 	47 minutes ago	<div><div>▶</div><div>⋮</div><div>🗑</div></div>
<input type="checkbox"/>	<div><div> <b>mongo_db-1</b></div><div>fc8bbce75ac0 </div></div>	<a href="#">mongo:latest</a>	Exited	0%	27017:27017 	47 minutes ago	<div><div>▶</div><div>⋮</div><div>🗑</div></div>
<input type="checkbox"/>	<div><div>▼</div><div> <b>aeropuerto</b></div></div>		Running (2/2)	0.25%		51 minutes ago	<div><div>■</div><div>⋮</div><div>🗑</div></div>
<input type="checkbox"/>	<div><div> <b>java_app</b></div><div>abd7ed47c9bd </div></div>	<a href="#">pee-java-app:1.0.0</a>	Running	0.25%	<a href="#">8080:8080</a> 	51 minutes ago	<div><div>■</div><div>⋮</div><div>🗑</div></div>
<input type="checkbox"/>	<div><div> <b>java_db</b></div><div>c8234fa5eb18 </div></div>	<a href="#">postgres:12</a>	Running	0%	<a href="#">5432:5432</a> 	51 minutes ago	<div><div>■</div><div>⋮</div><div>🗑</div></div>

## ARQUITECTURA DE LA APLICACIÓN



#### 4. Conclusiones

- Implementar el BackEnd con Spring Boot ha permitido una rápida configuración y desarrollo de API RESTful. La flexibilidad y eficiencia de Spring Boot en la gestión de datos y la implementación de la lógica empresarial han optimizado el tiempo de desarrollo y garantizado una base robusta para la aplicación.
- Angular ha posibilitado la creación de un FrontEnd altamente dinámico y escalable. Su arquitectura basada en componentes reutilizables y su sólido manejo de datos han facilitado la presentación interactiva de información, asegurando una experiencia de usuario atractiva y moderna.
- La combinación de Spring Boot en el BackEnd y Angular en el FrontEnd ha permitido operaciones CRUD ágiles y efectivas. A pesar de la ausencia de una base de datos persistente, la capacidad de Spring Boot para gestionar datos en memoria ha garantizado una integración sin inconvenientes con las interfaces dinámicas desarrolladas en Angular, ofreciendo una experiencia coherente al usuario.
- hemos alcanzado con éxito la implementación de una aplicación distribuida en nuestro proyecto. Esta aplicación conecta aplicaciones de backend con bases de datos MongoDB y PostgreSQL, permitiendo operaciones CRUD en cada una de ellas. Para integrar estos sistemas, hemos utilizado RestTemplate, facilitando la conexión de los datos almacenados en nuestra tabla PostgreSQL con una colección en MongoDB. Además, hemos desarrollado un aplicativo en Angular para la parte de FrontEnd, el cual se conecta tanto al backend de

PostgreSQL como al backend de MongoDB, permitiendo realizar operaciones CRUD de manera eficiente desde la interfaz de usuario. Este enfoque integral asegura una experiencia de usuario fluida y una gestión eficaz de los datos en nuestra aplicación distribuida.

## **5. Recomendaciones**

- Incorpora pruebas unitarias y de integración tanto en el BackEnd (Spring Boot) como en el FrontEnd (Angular). Esto asegurará la estabilidad y el correcto funcionamiento del sistema, identificando y solucionando posibles problemas durante el desarrollo, lo que contribuirá a la calidad del proyecto.
- Es recomendable adoptar principios de diseño responsivo en las interfaces de usuario desarrolladas en Angular. Al hacerlo, se garantiza una experiencia uniforme y adecuada en una amplia gama de dispositivos y tamaños de pantalla, mejorando así la accesibilidad y usabilidad del sistema.

## **6. Anexos**

**Link de GitHub:** <https://github.com/ovavellan/proyectoDistribuidas>