

# APUNTES JAVA REST API

## API REST API\_FLIGHTS

### 1.1\_CREAR EL PROYECTO

Proyecto Java Springboot,

Tipo Maven,

Añadir Dependencias:

Springboot DevTools

SpringWeb

REST Repository

SpringData JPA

Postgresql

Se crean las siguientes dependencias en POM.xml:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-rest</artifactId>
  </dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  </plugin>
</plugins>
</build>
```

## 2.1\_CONECTAR CON LA BASE DE DATOS

Con la funcionalidad de *JAVA DATASOURCE*

```
port: 5432
usuario: postgres
password: postgres
database: {name_database}
url: jdbc:postgresql://localhost:5432/{name_database}
```

## 2.2\_AÑADIR POSTGRESQL AL PROYECTO

**Añadir** las dependencias Maven desde el sitio web MAVEN REPOSITORY.

**Añadir** la funcionalidad IntelliJIDEA.

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>42.7.3</version>
</dependency>
```

### 3.\_CREAR EL FICHERO CONFIGURACION SPRINGBOOT

**Añadir** manualmente el fichero 'application.properties' a la carpeta */resources*:

```
spring.application.name={name application}
spring.datasource.username= postgres
spring.datasource.password= postgres
spring.datasource.driver-class-name= org.postgresql.Driver
spring.datasource.url= jdbc:postgresql://localhost:5432/{name_database}
spring.jca.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
server.servlet.context-path=/flights_api
```

### 4\*[Opcional]AÑADIR SPRINGBOOT A UN PROYECTO YA EXISTENTE

**Añadir** las <dependencias> al fichero POM desde el sitio web MAVEN REPOSITORY.

**Crear** la faceta y el modulo springboot.

```
<!-- Spring Boot -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <version>3.4.2</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
  <version>3.4.2</version>
</dependency>
```

## 5.1\_AÑADIR HIBERNATE AL PROYECTO

**Añadir** las <dependencias> al fichero POM desde el sitio web MAVEN REPOSITORY.

**Añadir** la faceta Hibernate desde el Project Structure del entorno INTELLIJIDEA y también el fichero descriptor de configuración.

**Configurar** la sesion Factory del fichero de configuracion hibernate, 'hibernate.cfg.xml'.

```
<session-factory>
<property name="connection.url">jdbc:postgresql://localhost:5432/{database}</property>
<property name="connection.driver_class">org.postgresql.Driver</property>
<property name="connection.username">postgres</property>
<property name="connection.password">postgres</property>
</session-factory>
```

### 6.1\_ENTITY [CLASE]

Clase resultado del mapeo HIBERNATE. Contiene la informacion almacenada en la base de datos Postgres.

### 6.2\_REPOSITORY-DAO [INTERFAZ]

Interfaz que extiende de CRUDREPOSITORY. Podemos usar las funciones ya implementadas para realizar tareas CRUD.

### 6.3\_SERVICE [CLASE]

Clase que contiene la lógica para devolver la información que necesitamos.

### 6.4\_DTO [CLASE]

Clase que permite mandar objetos complejos entre la capa de Controlador y la capa Servicio. Util para llamadas POST.

### 6.5\_CONTROLADOR [CLASE]

Clase que controla las interacciones del usuario. Puede ser de distintos tipos, según el tipo de aplicación que queramos conseguir.

BASE DE DATOS

ENTITY

DAO - REPOSITORY

SERVICE

DTO

CONTROLLER

USER

## 6.1\_ "ENTITY"

La clase entity es el resultado de mapear las clases de la base de datos con Hibernate.

Contiene las **validaciones** y **las consultas** sobre los tipos de datos.

```
@Entity
@Table(name = "airports")
public class AirportEntity {
    @Id
    @NotBlank
    @Size(max = 4)
    @Column(name = "code", nullable = false, length = 4)
    private String code;

    @NotBlank
    @Size(max = 100)
    @Column(name = "name", nullable = false, length = 100)
    private String name;
    getters / setters
    @NamedQueries({
        @NamedQuery(
            name = "FlightEntity.findAllOrigins",
            query = "SELECT DISTINCT f.source FROM FlightEntity f"
        ),
        @NamedQuery(
            name = "FlightEntity.findDestinationsByOrigin",
            query = "SELECT f.destination FROM FlightEntity f WHERE f.source.code = :originId"
        ),
        @NamedQuery(
            name = "FlightEntity.findFlightCodeByRoute",
            query = "SELECT f.flightCode FROM FlightEntity f WHERE f.source.code = :originId AND
f.destination.code = :destinationId"
        )
    })
    @NamedNativeQueries({
        @NamedNativeQuery(
            name = "FlightEntity.countSeatNumber",
            query = "SELECT yveeli_01_available_seats_int(:flightCode, :dateOfTravel)",
            resultClass = Integer.class
        ),
        @NamedNativeQuery(
            name = "FlightEntity.checkSeatAvailability",
            query = "SELECT yveeli_02_available_seats_bool(:flightCode, :dateOfTravel)",
            resultClass = Boolean.class
        )
    })
}
```

## 6.2\_ "DAO - REPOSITORY"

Se trata de una interfaz. Es el repositorio que contiene herramientas CRUD. Son llamadas desde la capa de Servicios. Las funciones se nombran pero no se definen, ya que son llamadas desde la capa servicio.

**@Repository**

```
public interface AirportRepository extends CrudRepository<AirportEntity, String> {  
}
```

**@Repository**

```
public interface IFlightsEntityDAO extends CrudRepository<FlightEntity, String> {
```

```
    List<AirportEntity> findAllOrigins();
```

```
    List<AirportEntity> findDestinationsByOrigin(
```

```
        @Param("originId") String originId);
```

```
    List<String> findFlightCodeByRoute(
```

```
        @Param("originId") String originId,
```

```
        @Param("destinationId") String destinationId);
```

```
// PostgreSQL Functions -
```

```
    Integer countSeatNumber(
```

```
        @Param("flightCode") String flightCode);
```

```
    Boolean checkSeatAvailability(
```

```
        @Param("flightCode") String flightCode,
```

```
        @Param("dateOfTravel") LocalDate dateOfTravel);
```

```
    }
```

### 6.3\_ "SERVICE"

Clase que hace las llamadas al repositorio de la base de datos. Contiene las funciones que usa el controlador para interactuar con el usuario.

**@Service**

**public class FlightService {**

    @Autowired

    private IFlightsEntityDAO flightDAO;

    public List<AirportEntity> findAllOrigins(){

        return flightDAO.findAllOrigins();

    }

    public List<AirportEntity> findDestinationsByOrigin(String originCode){

        return flightDAO.findDestinationsByOrigin(originCode);

    }

    public List<String> findFlightCodeByRoute(String origin, String destination){

        return flightDAO.findFlightCodeByRoute(origin, destination);

    }



## 6.4\_”CONTROLLER” - TIPO REST

Clase que controla la interaccion con el usuario. Puede ser de varios tipos:  
De tipo **REST** devuelven un JSON, de tipo **CONTROLLER** devuelven datos.  
Los parámetros pueden ser de tipo:

### Definicion

#### @RestController

#### @RequestMapping("/laLiga")

```
public class TicketController {  
    @Autowired  
    private TicketService ticketService;
```

### Endpoint

#### @GetMapping("/equipos")

```
public List<EquipoEntity> getEquipos(){  
    return equipo.getEquiposList();  
}
```

### PathVariable

#### @GetMapping("/Destinations/{originId}")

```
public List<AirportEntity> findDestinationsByOrigin(  
    @PathVariable String originId)  
{  
    // Cuerpo de la funcion  
    // recoge datos de ticketService  
}
```

#### ejemplo:

/Destinations/Alicante

### QueryParameters

#### @GetMapping("/Available")

```
public Boolean checkSeatAvailability(  
    @RequestParam String flight_code,  
    @RequestParam LocalDate flight_date)  
{  
    // Cuerpo de la funcion  
    // recoge datos de ticketService  
}
```

#### ejemplo:

/Available?flight\_code=AB123&flight\_date=2025-05-10

```

@RestController
@RequestMapping("/Flights")
public class FlightsController {
    @Autowired
    private FlightService flightService;

    @GetMapping("/Origins")
    public List<AirportEntity> findAllFlights(){
        return flightService.findAllOrigins();
    }

    @GetMapping("/Destinations/{originId}")
    public List<AirportEntity> findDestinationsByOrigin(
        @PathVariable String originId){
        return flightService.findDestinationsByOrigin(originId);
    }

    @GetMapping("/{originId}/{destinationId}")
    public List<String> findFlightCodeByRoute(
        @PathVariable String originId,
        @PathVariable String destinationId){
        return flightService.findFlightCodeByRoute(originId, destinationId);
    }

    @GetMapping("/Available/{flight_code}/{flight_date}")
    public Boolean checkSeatAvailability(
        @PathVariable String flight_code,
        @PathVariable LocalDate flight_date){
        return flightService.areSeatsAvailable(flight_code, flight_date);
    }
}

```

**GET /Flights/Origins**

**GET /Flights/Destinations/LIA**

**GET /Flights/LIA/SFIA**

**GET /Flights/Available/XC120/2025-12-25**

### 7.1.\_DEPENDENCIAS THYMELEAF

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

### 7.2.\_CLASE CONTROLLER HTML

Clase que controla las llamadas a los endpoints desde el cliente y los envia a Thymeleaf para que muestre los datos en un HTML.

```
@Controller
@RequestMapping("/airports")
public class AirportController {
    @Autowired
    private AirportService airportService;

    @GetMapping("/all")
    public String getAirports(Model model) {
        List<Airports> allAirports = airportService.getAllAirports();
        model.addAttribute("airports",allAirports);
        return "airport-view";
    }
}
```

### 7.3.\_HTML QUE MUESTRA THYMELEAF

Página web HTML que utiliza Thymeleaf para mostrar los datos que devuelve el controlador REST.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Lista de Aeropuertos</title>
    <link type="stylesheet" th:href="@{/css/testing.css}"
</head>
<body>
    <h1>Lista de Aeropuertos</h1>
    <ul>
        <li th:each="airport : ${airports}"
            th:text="${airport.name}">
        </li>
    </ul>
</body>
</html>
```



## 7.A.\_EJEMPLO THYMELEAF 'A'

### 7.A.1\_Clase CityEntity

```
@NamedQueries({
    @NamedQuery(
        name = "CityEntity.getAllCodes",
        query = "SELECT city.code FROM CityEntity city"
    ),
    @NamedQuery(
        name = "CityEntity.getParcialCityWithCode",
        query = "SELECT city.name FROM CityEntity city WHERE city.code = :ciudadCodigo"
    )
})
@Entity
@Table(name = "cities")
public class CityEntity
```

### 7.A.2\_Clase ICityEntityDAO

#### @Repository

```
public interface ICitiesEntityDAO extends CrudRepository<CityEntity, String> {
    List<String> getAllCodes();
    List<String> getParcialCityWithCode(@Param("ciudadCodigo") String ciudadCodigo);
}
```

### 7.A.3\_Clase CityService

#### @Service

```
public class CityService {
    @Autowired
    ICitiesEntityDAO cityDAO;
    public List<String> getAllCodes(){
        return cityDAO.getAllCodes();
    }
    public List<String> getParcialCityWithCode(String ciudadCodigo){
        return cityDAO.getParcialCityWithCode(ciudadCodigo);
    }
}
```

### 7.A.3\_Clase CityControllerWeb

#### @Controller

```
@RequestMapping("/web/cities")
public class CityControllerWEB {
    @Autowired
    CityService cityService;
    @GetMapping("/allcodes")
    public String getAllCodes(Model model){
        List<String> codesCity = cityService.getAllCodes();
        model.addAttribute("codes", codesCity);
        return "city";
    }
}
```

### 7.A.4\_Recurso HTML - city.html

```
<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <title>City</title>
    <link rel="stylesheet" th:href="@{/css/estilos.css}">
    /src/main/resources/static/css/estilos.css'
</head>
<body>
<h1>Cities</h1>
<select id="combo">
    <option>Ciudad</option>
    <option
        th:each="codigo : ${codes}"
        th:value="${codigo}"
        th:text="${codigo}">
    </option>
</select>
<p id="city"></p>
<script>
    var infoCity = document.getElementById("combo");
    var infodetail = document.getElementById("city");
    infoCity.addEventListener('change', function () {
        const code = this.value;
        loadCityWithCode(code);
    });
    function loadCityWithCode(code) {
        fetch('http://localhost:8080/flights_api/cities/partial/${code}')
            .then(response => response.json())
            .then(data => {
                infodetail.innerText = data[0];
            });
    }
</script> </body> </html>
```

## 8.1\_APLICACION JAVA FX

Crear un proyecto nuevo de tipo JavaFX, lenguaje JAVA, sistema MAVEN.

Dependencias del proyecto:

```
<dependencies>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>17.0.6</version>
  </dependency>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-fxml</artifactId>
    <version>17.0.6</version>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
  </dependency>
  <!-- JSON -->
  <dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20210307</version>
  </dependency>
</dependencies>
```

## 8.2\_Clase Application

Es el punto de entrada de la aplicacion. Carga una escena de un tamaño especificado.

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;
import java.io.IOException;

public class HTTPApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(HTTPApplication.class.getResource("view-ticket-form.fxml"));
        Scene scene = new Scene(fxmlLoader.load());
        stage.setTitle("Flights APP");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args){
        launch();
    }
}
```



### 8.3\_Clase Controller

Gestiona las ordenes que aporta el usuario a la interfaz grafica.

```
import andres.flights_jfxtemplate.Bridges.Ticket_in_Passenger;
import andres.flights_jfxtemplate.DTO.TicketDTO;
import andres.flights_jfxtemplate.Entity.AirportEntity;
import andres.flights_jfxtemplate.Entity.FlightEntity;
import andres.flights_jfxtemplate.Service.FlightService;
import andres.flights_jfxtemplate.Service.TicketService;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.stage.Stage;
import java.io.IOException;
import java.time.LocalDate;
import java.util.List;

public class TicketsController {
    @FXML
    private DatePicker datePicker;
    @FXML
    private ComboBox<AirportEntity> originCombo;
    @FXML
    private ComboBox<AirportEntity> destinationCombo;
    @FXML
    private ComboBox<FlightEntity> flightCombo;
    @FXML
    private ComboBox<String> typeFlightCombo;
    @FXML
    private TextField passportField;
    @FXML
    private Button logButton;
    @FXML
    private Label logLabel;

    private String errorMessage;
    private boolean formHasErrors;

    private FlightService flightService = new FlightService();
    private TicketService ticketService = new TicketService();
```

@FXML

```
public void initialize(){
    // check server_status
    checkServerStatus();

    // load combo_box
    loadDate();
    loadOrigins();
    loadTypeFlight();

    // event_listeners
    listenerOrigins();
    listenerDestination();
    listenerTypeFlights();
}

private void loadDate(){
    datePicker.setDayCellFactory(picker -> new DateCell(){
        @Override
        public void updateItem(LocalDate date, boolean empty){
            super.updateItem(date, empty);
            setDisable(empty || date.isBefore(LocalDate.now()));
        }
    });
    datePicker.setValue(LocalDate.now());
}

private void loadOrigins(){
    originCombo.getItems().addAll(flightService.getAllOrigins());
}

// EVENT LISTENERS
public void listenerOrigins(){
    originCombo.setOnAction(event -> {
        AirportEntity selectedOrigin = originCombo.getSelectionModel().getSelectedItem();
        if (selectedOrigin != null){
            destinationCombo.getItems().clear();

            List<AirportEntity> destinations =
flightService.getDestinationsByOrigin(selectedOrigin.getCode());
            destinationCombo.getItems().addAll(destinations);
        }
    });
}

public void listenerDestination(){
    destinationCombo.setOnAction(event -> {
```

```

        AirportEntity selectedOrigin = originCombo.getSelectionModel().getSelectedItem();
        AirportEntity selectedDestination = destinationCombo.getSelectionModel().getSelectedItem();
        if (selectedOrigin != null || selectedDestination != null) {
            flightCombo.getItems().clear();
            List<FlightEntity> flights = flightService.getAllFlights(selectedOrigin.getCode(),
selectedDestination.getCode());
            flightCombo.getItems().addAll(flights);
        }
    });
}

```

```

private void loadTypeFlight() {
    typeFlightCombo.getItems().add("J");
    typeFlightCombo.getItems().add("👤");
    typeFlightCombo.getItems().add("✈️");
}

```

```

public void listenerTypeFlights() {
    typeFlightCombo.setOnAction(event -> {
        int selectedIndex = typeFlightCombo.getSelectionModel().getSelectedIndex();
        if (selectedIndex >= 0 && selectedIndex < flightCombo.getItems().size()) {
            flightCombo.getSelectionModel().select(selectedIndex);
        }
    });
}

```

// FUNCIONALIDAD DE LOS BOTONES 'CREATE\_USER' Y 'CREATE\_TICKET'

```

public void createNewUser(ActionEvent event) {
    try {
        String passport = passportField.getText();
        if (passport.isEmpty()) {
            showWarningMessage("Error", "New user can't be created. \nPassport must be 8 characters long.");
            return;
        }
    }
}

```

```

FXMLLoader loader =
new FXMLLoader(getClass().getResource("/andres/flights_jfxtemplate/nvg-passenger-creation.fxml"));
Parent nuevaVista = loader.load();

```

```

Ticket_in_Passenger controller = loader.getController();
controller.setPassportno(passport);

```

```

Scene nuevaEscena = new Scene(nuevaVista);
Stage stage = (Stage)((Node) event.getSource()).getScene().getWindow();
stage.setScene(nuevaEscena);
stage.show();

```

```

    } catch (IOException e){
        throw new RuntimeException(e);
    }
}

public void createNewTicket(ActionEvent event){
    FlightEntity flight = flightCombo.getValue();
    String type = typeFlightCombo.getValue();
    String passport = passportField.getText();

    formHasErrors = false;
    validateFields(flight, passport, type);

    if (formHasErrors){
        showWarningMessage("Warning", "The form present several errors.\nCheck the error log panel.");
        return;
    }

    TicketDTO ticket = new TicketDTO();
    ticket.setDateOfTravel(datePicker.getValue());
    ticket.setFlightCode(flight.getFlightCode());
    ticket.setPassportno(passport);

    ticketService.createNewTicket(event, ticket, passport);
}

private void validateFields(FlightEntity flight, String passport, String type){
    checkServerStatus();
    if(formHasErrors)
        return;

    if (flight == null){
        logLabel.setStyle("-fx-background-color: red;");
        errorMessage = "Flight field must be completed.";
        formHasErrors = true;
        return;
    }

    if (type == null){
        logLabel.setStyle("-fx-background-color: red;");
        errorMessage = "Flight-VIP field must be completed.";
        formHasErrors = true;
        return;
    }

    if (passport == null || passport.isEmpty()){

```

```

        logLabel.setStyle("-fx-background-color: red;");
        errorMessage = "Passport field must be completed.";
        formHasErrors = true;
        return;
    }

    if (!passport.matches("^[A-Z][0-9]{7}$")) {
        logLabel.setStyle("-fx-background-color: red;");
        errorMessage = "Passport format is not valid.";
        formHasErrors = true;
        return;
    }

    boolean availableSeats = isAvailableSeats();
    if (!availableSeats) {
        errorMessage = "There are no available seats.";
        formHasErrors = true;
        return;
    }

    boolean duplicatedTicket = isDuplicatedTicket();
    if (duplicatedTicket) {
        errorMessage = "There is a ticket for that day already.";
        formHasErrors = true;
        return;
    }
}

private boolean isAvailableSeats() {
    String flightCode = flightCombo.getValue().getFlightCode();
    LocalDate flightDate = datePicker.getValue();
    return ticketService.getAvailableSeats(flightCode, flightDate);
}

private boolean isDuplicatedTicket() {
    String passportno = passportField.getText();
    LocalDate flightDate = datePicker.getValue();
    return ticketService.getDuplicatedFlight(passportno, flightDate);
}

private void checkServerStatus() {
    boolean status = ticketService.checkServerStatus();
    if (!status) {
        logLabel.setStyle("-fx-background-color: red;");
        errorMessage = "The application is not connected to the server.";
        formHasErrors = true;
    }
}

```

```
    } else {  
        formHasErrors = false;  
    }  
}  
  
public void showErrorMessage(){  
    Alert alert = new Alert(Alert.AlertType.ERROR);  
    alert.setTitle("Error");  
    alert.setHeaderText("Please, check these form errors");  
    alert.setContentText(errorMessage);  
    alert.showAndWait();  
}  
  
private void showWarningMessage(String title, String message){  
    Alert alert = new Alert(Alert.AlertType.WARNING);  
    alert.setTitle(title);  
    alert.setHeaderText(null);  
    alert.setContentText(message);  
    alert.showAndWait();  
}  
}
```

## 8.4\_Fichero FXML

Contiene el diseño de la interfaz grafica, que interactua con el controlador.

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane maxHeight="500.0" maxWidth="400.0" minHeight="500.0" minWidth="400.0"
xmlns="http://javafx.com/javafx/8"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="andres.flights_jfxtemplate.Controller.TicketsController">
    <children>

        <VBox alignment="TOP_CENTER" prefHeight="500.0" prefWidth="400.0" spacing="20.0">
            <children>
                <HBox alignment="TOP_CENTER" prefHeight="30.0" prefWidth="200.0" spacing="10.0">
                    <children>
                        <DatePicker fx:id="datePicker" minWidth="240.0"/>
                    </children>
                    <VBox.margin>
                        <Insets top="20.0" />
                    </VBox.margin>
                </HBox>
                <ComboBox fx:id="originCombo" minWidth="240.0" prefWidth="200.0" promptText="Select an origin">
                    <VBox.margin>
                        <Insets />
                    </VBox.margin>
                </ComboBox>
                <ComboBox fx:id="destinationCombo" prefWidth="240.0" promptText="Select a destination" />

                <HBox alignment="TOP_CENTER" prefWidth="260.0" spacing="10.0">
                    <children>
                        <ComboBox fx:id="flightCombo" prefWidth="170.0" promptText="Select a flight" />
                        <ComboBox fx:id="typeFlightCombo" minWidth="50.0" />
                    </children>
                </HBox>

                <HBox alignment="TOP_CENTER" prefWidth="260.0" spacing="10.0">
                    <children>
                        <TextField fx:id="passportField" prefWidth="170.0" promptText="Insert your passport" />
                        <Button fx:id="newUserButton" text="Create" minWidth="60.0" onAction="#createNewUser"/>
                    </children>
                </HBox>

            </children>
        </VBox>
    </children>
</AnchorPane>
```

```
        <children>
            <Button fx:id="logButton" prefWidth="170.0" text="Error log panel" onAction="#showErrorMessage"/>
            <Label fx:id="logLabel" prefWidth="60.0" minHeight="24" style="-fx-background-color: green;"/>
        </children>
    </HBox>
    <Button fx:id="newTicketButton" minWidth="240.0" text="Buy Ticket" onAction="#createNewTicket"/>
</children>
    <padding>
        <Insets top="50.0" />
    </padding>
</VBox>
</children>
</AnchorPane>
```



## 9.\_DOCUMENTACION

Manual de instrucciones de nuestras aplicaciones.

### 9.1\_DOCUMENTACION DE APIREST CON SWAGGER

Añadir las dependencias al fichero .pom.

```
<dependency>  
    <groupId>org.springdoc</groupId>  
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>  
    <version>2.7.0</version>  
</dependency>
```

Acceder a la interfaz SWAGGER.

localhost:8080/{name api}/swagger-ui/index.html

## 9.2\_DOCUMENTACION DE JAVA CON JAVADOCS

**\_Opcion A:** 'javadoc' desde linea de comandos.

+ Es imprescindible instalar el sdk de javaFX. Se descarga de "<https://gluonhq.com/products/javafx/>" y debemos guardar la carpeta en algún lugar de nuestro disco duro.

+ Se debe ejecutar el siguiente comando:

```
javadoc -d docs/  
-sourcepath src/main/java  
-module-path "C:\Program Files\javafx-sdk-21.0.7\lib"  
-add-modules javafx.controls,javafx.fxml {package_name}
```

donde:

docs\ : carpeta contenedora de documentación.

src\main\java: ruta del código fuente.

C:\Program Files\javafx-sdk-21.0.7\lib: ruta del sdk de javaFx.

-add-modules javafx.controls,javafx.fxml {package\_name}: con el nombre del package.

**\_Opcion B:** 'mvn' desde linea de comandos.

+ Es imprescindible instalar maven en nuestro dispositivo. Se descarga de "<https://maven.apache.org/download.cgi>". Se puede instalar con Chocolatey con el comando "choco install maven" y se puede comprobar la version con el comando "mvn -version".

+ Necesitamos modificar el fichero .pom

```
<plugin>  
  <groupId>org.apache.maven.plugins</groupId>  
  <artifactId>maven-compiler-plugin</artifactId>  
  <version>3.13.0</version>  
  <configuration>  
    <source>24</source>  
    <target>24</target>  
  </configuration>  
</plugin>
```

+ El comando para crear la documentacion es el siguiente:

```
mvn javadoc:javadoc
```



## Etiquetas estándar de Javadoc.

Etiqueta	Descripción	Contexto de uso
<code>@author</code>	Indica el autor del código.	A nivel de clase
<code>@version</code>	Indica la versión del archivo/clase.	A nivel de clase
<code>@since</code>	Indica desde qué versión está disponible el código.	Clase, método, campo
<code>@deprecated</code>	Marca un elemento como obsoleto (no recomendado para su uso).	Clase, método, campo
<code>@see</code>	Referencia cruzada a otra clase, método o recurso relacionado.	Cualquier lugar
<code>@link</code>	Crea un enlace en línea a otro elemento (clase, método, etc.).	En el texto del comentario
<code>@param</code>	Describe un parámetro de un método o constructor.	En métodos y constructores
<code>@return</code>	Describe el valor de retorno de un método.	En métodos que no son <code>void</code>
<code>@throws</code>	Describe una excepción que puede lanzar el método.	Métodos y constructores
<code>{@code }</code>	Muestra el texto en monoespaciado como código (escapa HTML).	Dentro del texto
<code>{@literal}</code>	Muestra texto literalmente (no interpreta HTML o Javadoc).	Dentro del texto
<code>{@value}</code>	Inserta el valor constante de un campo.	En campos <code>static final</code>
<code>@hidden</code>	Oculto el elemento de la documentación. (desde Java 9+)	En cualquier elemento del código
<code>@index</code>	Añade una entrada al índice generado. (desde Java 18+)	Clase, método, campo, paquete

Usa `@param`, `@return` y `@throws` en **todos los métodos públicos**.

`@deprecated` puede ir acompañado de una explicación del reemplazo.

Dentro de los comentarios puedes usar HTML: `<p>`, `<ul>`, `<b>`, etc.

Las etiquetas `@hidden` y `@index` son útiles para controlar o mejorar la documentación pública.

## A.\_VALIDACIONES CON NOTACIONES @VALID

Estas son algunas de las anotaciones para validar en la clase ENTITY

```
public class PassengerEntity {  
    @NotNull(message = "El nombre no debe ser nulo")  
    @NotBlank(message = "El nombre es obligatorio")  
    @NotEmpty(message="El nombre no debe estar vacio")  
    @Size(min = 2, max = 50, message = "El nombre debe tener entre 2 y 50 caracteres")  
    private String name;  
  
    @Email(message = "Email no válido")  
    private String email;  
  
    @Min(value = 0, message = "La edad no puede ser negativa")  
    @Max(value = 120, message = "La edad no puede superar los 120 años")  
    private int age;  
  
    @Pattern(regexp = "\\+34\\d{9}", message = "El teléfono debe comenzar con +34 y tener 9 dígitos")  
    private String phone;  
  
    @Pattern(regexp = "[A-Za-z][0-9]{7}", message = "El pasaporte debe empezar con letra y contener 7 digitos")  
    private String passport;  
  
    @Past(message = "La fecha de nacimiento debe ser en el pasado")  
    private LocalDate birthDate;  
  
    @Future(message = "La fecha de reserva debe ser en el futuro")  
    private LocalDate reservationDate;
```

## **B. @NAMEDQUERY Y @NAMEDQUERIES**

```
@NamedQuery(  
    name=""  
    query=""  
)
```

```
@NamedQueries(  
    {  
        @NamedQuery(name="", query=""),  
        @NamedQuery(name="", query=""),  
        @NamedQuery(name="", query=""),  
    }  
)
```

```
@NamedQueries({  
    @NamedQuery(  
        name = "FlightEntity.findAllOrigins",  
        query = "SELECT flight.source FROM FlightEntity flight"  
    ),  
    @NamedQuery(  
        name = "FlightEntity.findDestinationsByOrigin",  
        query = "SELECT f.destination FROM FlightEntity f WHERE f.source.code = :originId"  
    ),  
    @NamedQuery(  
        name = "FlightEntity.findFlightCodeByRoute",  
        query =  
            "SELECT f.flightCode  
            FROM FlightEntity f  
            WHERE f.source.code = :originId AND f.destination.code = :destinationId"  
    )  
})
```

```
@NamedNativeQueries({  
    @NamedNativeQuery(  
        name = "FlightEntity.countSeatNumber",  
        query = "SELECT yveeli_01_available_seats_int(:flightCode, :dateOfTravel)",  
        resultClass = Integer.class  
    ),  
    @NamedNativeQuery(  
        name = "FlightEntity.checkSeatAvailability",  
        query = "SELECT yveeli_02_available_seats_bool(:flightCode, :dateOfTravel)",  
        resultClass = Boolean.class  
    )  
})
```

## @QUERIES IN DAO

### @Repository

```
public interface IFlightsEntityDAO extends CrudRepository<FlightEntity, String> {  
    @Query("SELECT DISTINCT f.source FROM FlightEntity f")  
    List<AirportEntity> findAllOrigins();  
    @Query("SELECT f.destination FROM FlightEntity f WHERE f.source.code = :originId")  
    List<AirportEntity> findDestinationsByOrigin(@Param("originId") String originId);  
    @Query("SELECT f.flightCode FROM FlightEntity f  
    WHERE f.source.code = :originId AND f.destination.code = :destinationId")  
    Optional<List<String>> findFlightCodeByRoute(  
    @Param("originId") String originId, @Param("destinationId") String destinationId);  
  
    @Query(value = "SELECT yveeli_01_available_seats_int(:flightCode)",  
    nativeQuery = true)  
    Integer countSeatNumber(@Param("flightCode") String flightCode);  
  
    @Query(value = "SELECT yveeli_02_available_seats_bool(:flightCode)",  
    nativeQuery = true)  
    Boolean checkSeatAvailability(@Param("flightCode") String flightCode);  
}
```

## COMPROBAR LAS ANOTACIONES @RESPONSEBODY


```
@GetMapping("/hello")
public String sayHello() {
    return "hello";
}
```

 Esto buscará una vista llamada hello.html (por ejemplo, una plantilla Thymeleaf).

```
@GetMapping("/hello")
@ResponseBody
public String sayHello() {
    return "hello";
}
```

 Esto devolverá literalmente el texto "hello" en el cuerpo de la respuesta HTTP.

```
@GetMapping("/airport")
@ResponseBody
public AirportEntity getAirport() {
    return new AirportEntity("JFKI", "John F. Kennedy");
}
```

 Esto se convierte automáticamente en:

json

Copiar

Editar

```
{
  "code": "JFKI",
  "name": "John F. Kennedy"
}
```

 ¿Y si usamos **@RestController**?

Un **@RestController** es equivalente a poner **@Controller** + **@ResponseBody** en todos los métodos.

Entonces, en tu clase:

java

CopiarEditar

```
@RestController
@RequestMapping("/Flights")
public class FlightsController {
    ...
}
```

✓ Todos los métodos ya devuelven JSON sin que tú tengas que poner **@ResponseBody** cada vez.



**@Service**

**public class PassengerService {**

    @Autowired

    private IPassengerEntityDAO passengerDAO;

    public boolean **createPassenger**(PassengerDTO passengerDTO) {

        // validatePassenger(passengerDTO);

        // Convertir el DTO a una entidad

        PassengerEntity passengerEntity = new PassengerEntity();

        passengerEntity.setPassportno(passengerDTO.getPassportno());

        passengerEntity.setFirstname(passengerDTO.getFirstname());

        passengerEntity.setLastname(passengerDTO.getLastname());

        passengerEntity.setAddress(passengerDTO.getAddress());

        passengerEntity.setPhone(passengerDTO.getPhone());

        passengerEntity.setSex(passengerDTO.getSex());

        // Guardar la entidad en la base de datos usando el DAO

        try {

            passengerDAO.save(passengerEntity);

            return true;

        } catch (Exception e) {

            return false;

        }

    }

    public PassengerEntity **findPassengerByPassport**(String passportno) {

        return passengerDAO.findByPassportno(passportno);

    }

**// VALIDATING FUNCTIONS - SERVICE LAYER -**

    private void validatePassport(String passport) {

        if (passport == null || passport.trim().isEmpty()) {

            throw new IllegalArgumentException("Passport identity is mandatory.");

        }

        if (passport.length() != 8) {

            throw new IllegalArgumentException("Passport identity number must be 8

characters long.");

        }

    }

```
private void validatePassenger(PassengerEntity passenger) {  
    if (passenger.getFirstname() == null || passenger.getFirstname().trim().isEmpty()) {  
        throw new IllegalArgumentException("First name is mandatory.");  
    }  
  
    if (passenger.getLastname() == null || passenger.getLastname().trim().isEmpty()) {  
        throw new IllegalArgumentException("Last name is mandatory.");  
    }  
}  
}
```

**@Service**

**public class TicketService {**

    @Autowired

    private ITicketEntityDAO ticketDAO;

    public boolean **createTicket**(TicketDTO ticket) {

        TicketEntity newTicket = new TicketEntity();

        newTicket.setId(ticket.getId());

        newTicket.setDateOfBooking(ticket.getDateOfBooking());

        newTicket.setDateOfTravel(ticket.getDateOfTravel());

        newTicket.setDateOfCancellation(ticket.getDateOfCancellation());

        newTicket.setPassportno(ticket.getPassportno());

        newTicket.setFlightCode(ticket.getFlightCode());

        newTicket.setPrice(ticket.getPrice());

        try{

            ticketDAO.save(newTicket);

            return true;

        } catch (Exception e){

            System.out.println(e.getMessage());

        }

        return false;

    }

    public TicketEntity **findTicketByld**(Integer ticketNumber) {

        return ticketDAO.getTicketEntityByld(ticketNumber);

    }

**// STORED PROCEDURES - SERVICE LAYER -**

**// Function that checks if passenger has already bought a ticket**

    public boolean **checkTicketExists**(LocalDate travelDate, String passportNo) {

        Boolean exists = ticketDAO.checkTicketExists(travelDate, passportNo);

        return exists != null && exists;

    }

}