**FootballClub Spring Boot Application**

Simple Spring Boot app to manage football clubs with CRUD operations, connected to either a local H2 database or an AWS RDS MySQL database.

**Features**

* CRUD operations for Football Club entities (id, city, name, year founded)
* Uses Spring Data JPA and Hibernate ORM
* Supports local testing with in-memory H2 database
* Supports production with AWS RDS MySQL database
* Tested with Postman for API requests
* Started locally via IntelliJ IDEA

**Prerequisites**

* Java 21+
* Maven
* IntelliJ IDEA (recommended for development)
* AWS account with RDS MySQL instance (optional for production)
* Postman (for API testing)

**Note:** The MySQL database on AWS RDS must be created manually via the AWS Console before connecting the app. The application will create and update tables automatically via Hibernate when running.

**Running the Application**

* The app is started locally from IntelliJ IDEA by running the main Spring Boot application class.
* The REST API will be available at http://localhost:8080.

**Testing**

* Use Postman (or any REST client) to test the API endpoints for CRUD operations.
* When running locally with H2, data will reset on each restart.
* When connected to AWS RDS MySQL, data persists between runs.

**Project Structure**

* entity — contains JPA entities (each entity corresponds to a table)
* repository — Spring Data JPA repositories
* controller — REST controllers exposing API endpoints
* service — business logic (optional, if used)

**Dependencies**

* Spring Boot Starter Data JPA
* Spring Boot Starter Web
* MySQL Connector/J
* H2 Database (runtime for local tests)
* Spring Boot Starter Test

Here's a simple Spring Boot app example for managing football clubs with CRUD (Create, Read, Delete) operations, connected to an Amazon RDS MySQL database.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**folder/package structure** for FootballClub Spring Boot project

A screenshot of a computer program

AI-generated content may be incorrect.

After importing downlodaded .zip from springinitializer to InteliJ:

A screenshot of a computer program

AI-generated content may be incorrect.

Pom.XML:



<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.5.3</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.example</groupId>  
 <artifactId>FootballClub</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>FootballClub</name>  
 <description>FootballClub project for Spring Boot</description>  
 <url/>  
 <licenses>  
 <license/>  
 </licenses>  
 <developers>  
 <developer/>  
 </developers>  
 <scm>  
 <connection/>  
 <developerConnection/>  
 <tag/>  
 <url/>  
 </scm>  
 <properties>  
 <java.version>21</java.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>com.mysql</groupId>  
 <artifactId>mysql-connector-j</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

APPLICATION.PROPERTIES;



A screenshot of a computer program

AI-generated content may be incorrect.

spring.application.name=FootballClub  
  
#H2 Config Default  
spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driver-class-name=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format\_sql=true  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
spring.h2.console.enabled=true  
spring.h2.console.path=/h2-console  
  
  
# AWS RDS MySQL DB  
#spring.datasource.url=jdbc:mysql://<your-rds-endpoint>:3306/<your-database-name>?useSSL=false&serverTimezone=UTC  
#spring.datasource.username=<your-username>  
#spring.datasource.password=<your-password>  
#spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
#spring.jpa.hibernate.ddl-auto=update  
#spring.jpa.show-sql=true  
#spring.jpa.properties.hibernate.format\_sql=true  
#spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect

Check DB In browser to: http://localhost:8080/h2-console

JDBC URL: jdbc:h2:mem:testdb (same as in your properties)

A screenshot of a computer

AI-generated content may be incorrect.

Currently empty since we don’t have any tables yet;

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

FootballClub.java: (entity) table in DB



package com.example.FootballClub;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
  
@Entity  
public class FootballClub {  
 @jakarta.persistence.Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private long id;  
 private String name;  
 private String city;  
 private int yearFounded;  
  
 public FootballClub(){  
  
 }  
  
 public FootballClub(Long id,String name, String city,int yearFounded){  
 this.id=id;  
 this.name=name;  
 this.city=city;  
 this.yearFounded=yearFounded;  
 }  
  
 public long getId() {  
 return id;  
 }  
  
 public void setId(long id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getCity() {  
 return city;  
 }  
  
 public void setCity(String city) {  
 this.city = city;  
 }  
  
 public int getYearFounded() {  
 return yearFounded;  
 }  
  
 public void setYearFounded(int yearFounded) {  
 this.yearFounded = yearFounded;  
 }  
}

Now DB is updated with new table FOOTBALL\_CLUB after creating this class and restart app;

A screenshot of a computer

AI-generated content may be incorrect.

Create the FootballClubRepository



JpaRepository<FootballClub, Long>:

* Tells Spring that this is a JPA repository for the FootballClub entity
* With Long being the type of the primary key (id)

You automatically get:

* save(), findById(), findAll(), deleteById() and more — no code needed!

package com.example.FootballClub;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface FootballClubRepository extends JpaRepository<FootballClub,Long> {  
}

FootballClubService.java



@Service marks this as a Spring service component.

The repository is injected via the constructor.

Common CRUD methods:

* getAllClubs() — get all clubs
* getClubById(id) — get a single club by ID
* saveClub(club) — create or update a club
* deleteClub(id) — remove a club by ID

It’s a **service layer** class annotated with @Service.

This means it holds the **business logic** of your application and acts as an intermediary between the controller (web layer) and the repository (data access layer).

package com.example.FootballClub;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class FootballClubService {  
  
 @Autowired // <-- field injection here  
 private FootballClubRepository footballClubRepository;  
  
 //Constructor injection (commented out)  
 //public FootballClubService(FootballClubRepository footballClubRepository){  
 // this.footballClubRepository=footballClubRepository;}  
  
 public List<FootballClub>getAllClubs(){  
 return footballClubRepository.findAll();  
 }  
  
 public Optional<FootballClub>getClubById(Long id){  
 return footballClubRepository.findById(id);  
 }  
  
 public FootballClub addClub(FootballClub club){  
 return footballClubRepository.save(club);  
 }  
  
 public void deleteClub(Long id){  
 footballClubRepository.deleteById(id);  
 }  
  
  
}

FootballClubConroller.java



Controller will handle HTTP requests (like GET, POST, DELETE), call the Service layer to perform operations, and then return responses to clients.

package com.example.FootballClub;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/clubs")  
public class FootballClubController {  
 @Autowired  
 private FootballClubService footballClubService;  
  
 @GetMapping  
 public List<FootballClub> getAllClubs() {  
 return footballClubService.getAllClubs();  
 }  
  
 @GetMapping("/{id}")  
 public ResponseEntity<FootballClub> getClubById(@PathVariable Long id) {  
 Optional<FootballClub> club = footballClubService.getClubById(id);  
  
 if (club.isPresent()) {  
 FootballClub club1 = club.get();  
 return new ResponseEntity<>(club1, HttpStatus.*OK*); // 200 OK with club data  
 } else {  
 return new ResponseEntity<>(HttpStatus.*NOT\_FOUND*); // 404 Not Found, no body  
 }  
 }  
  
 @PostMapping  
 public FootballClub addClub(@RequestBody FootballClub club){  
 return footballClubService.addClub(club);  
 }  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity<Void>deleteClub(@PathVariable Long id) {  
 footballClubService.deleteClub(id);  
 return new ResponseEntity<>(HttpStatus.*NO\_CONTENT*);  
 }  
}

Using postman we can test Requests:

Get is empty;

A screenshot of a computer

AI-generated content may be incorrect.

Adding more clubs with POST:

A screenshot of a computer

AI-generated content may be incorrect.

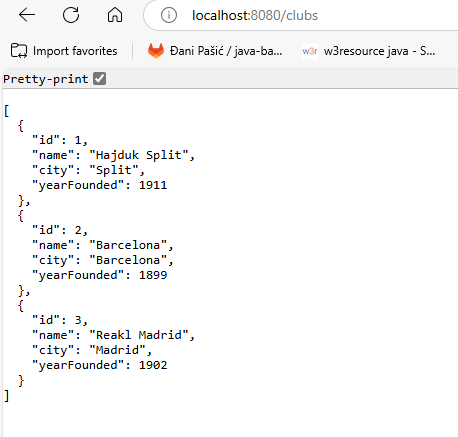
A screenshot of a computer

AI-generated content may be incorrect.

In DB and in GET request in browser we can see now entries;

A screenshot of a computer

AI-generated content may be incorrect.



If we delete one club with DELETE .e.g. with ID 3 it will be gone from DB also;

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

CREATING AWS RDS MySQL instance as DB:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

UN/PASS: admin/Administrator123A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

DB CREATED:

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

~ $ mysql -h databasetest.czqg6iykc8s2.us-east-1.rds.amazonaws.com -u admin -P 3306 -p

Enter password:

Welcome to the MariaDB monitor. Commands end with ; or \g.

Your MySQL connection id is 31

Server version: 8.0.41 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

NOW WE START APP IN INTELIJ;

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

ADDING CLUBS TO DB;

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

AFTER DELETING ID2 CLUB IS DELETED FROM DB ALSO;

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.