HW10 – SQLITE

goteam database

The GoTeam application is a sophisticated application that leverages Spring Boot for its backend services, SQLite as its database, and is linked to a frontend component ensuring real-time data synchronization. Below is a detailed walkthrough of the project, including database modifications, backend configuration, and how Maven is utilized to manage the project’s lifecycle.

**Steps done to achieve what is needed:**

1. **Backend Configuration:**
2. **application.properties:**
3. **Server Port:**

* **server.port=8089:** This property sets the port number for the application to run on. In this case, it’s set to 8089.

1. **Data Source Configuration:**

* **spring.datasource.url=jdbc:sqlite:goteam.db:** specifies the URL of the database. Here, SQLite is being used, and the database file is named ‘**goteam.db**’.
* **spring.datasource.driver-class-name=org.sqlite.JDBC:** sets the JDBC driver class name.
* **spring.datasource.initialization-mode=always:** determines when the database should be initialized. It’s set to ‘**always**’, meaning it will initialize the database every time the application starts.

1. **JPA/Hibernate Configuration:**

* **spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLiteDialect:** The Hibernate dialect is set to a custom SQLite dialect. This is crucial Hibernate to generate the correct SQL statements for SQLite.
* **spring.jpa.hibernate.ddl-auto=update:** The ‘**ddl-auto**’ property is set to ‘**update**’, meaning Hibernate will update the schema to match the entities in your application upon startup.

1. **Logging Configuration:**

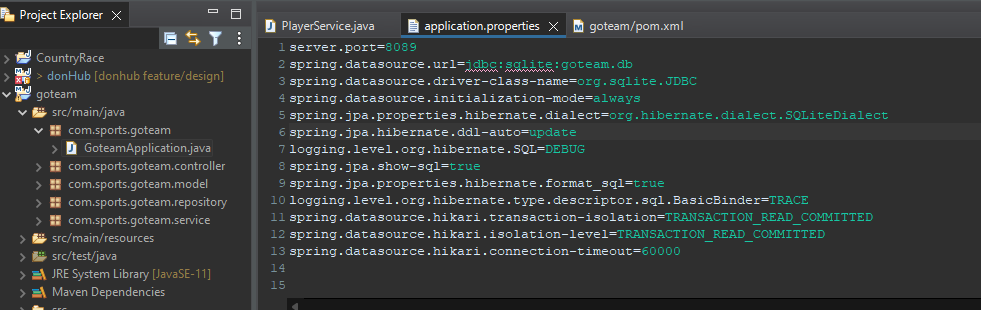
* **logging.level.org.hibernate.SQL=DEBUG**
* **spring.jpa.show-sql=true**
* **spring.jpa.properties.hibernate.format\_sql=true**
* **logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE**

These properties are set to log SQL statements and bind parameters at a detailed level. This is helpful for debugging.

1. **Transaction Isolation Level:**

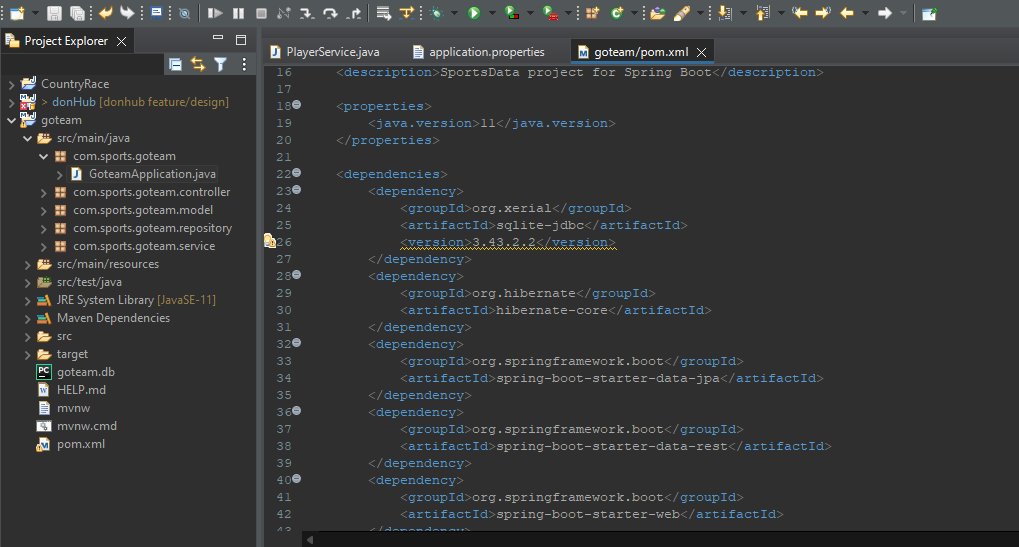
* **spring.datasource.hikari.transaction-isolation=TRANSACTION\_READ\_COMMITTED**
* **spring.datasource.hikari.isolation-level=TRANSACTION\_READ\_COMMITTED**
* **spring.datasource.hikari.connection-timeout=60000**

These properties configure the HikariCP connection pool with a specific transaction isolation level and connection timeout. SQLite supports a different set of isolation levels than some other databases, so it’s important to set this correctly.



1. **pom.xml:**

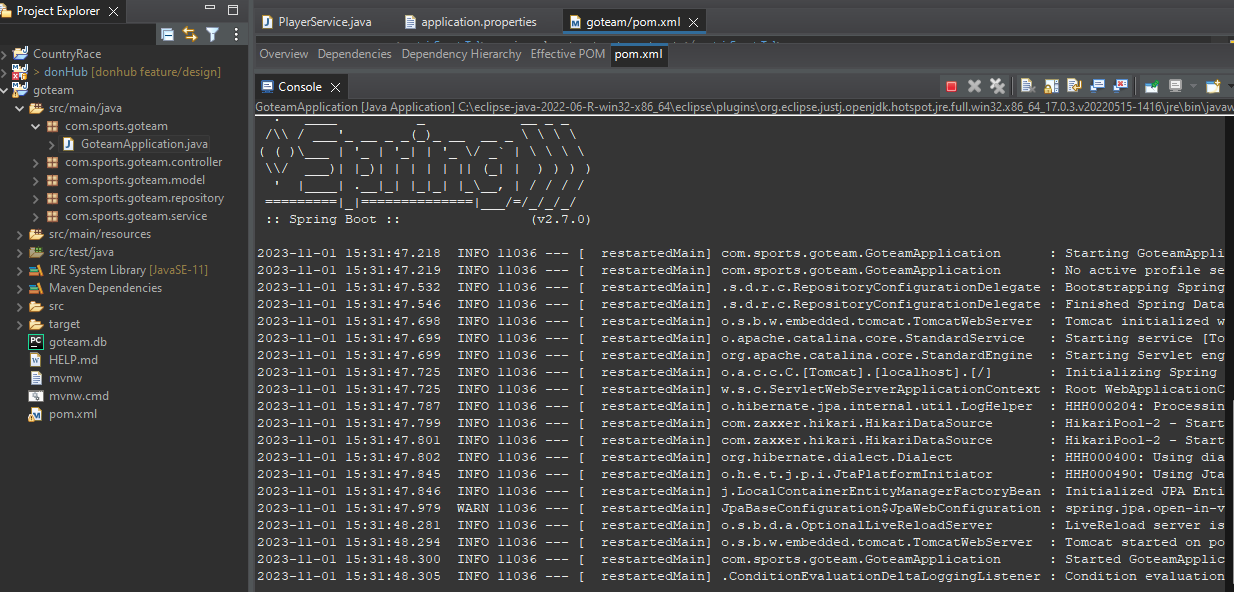
* **org.xerial:sqlite-jdbc:3.43.2.2** is the SQLite JDBC driver.
* **com.github.gwenn:sqlite-dialect:0.1.0** provides the Hibernate SQLite dialect.
* Various dependencies are included for **Spring Boot, Hibernate**, and other libraries.



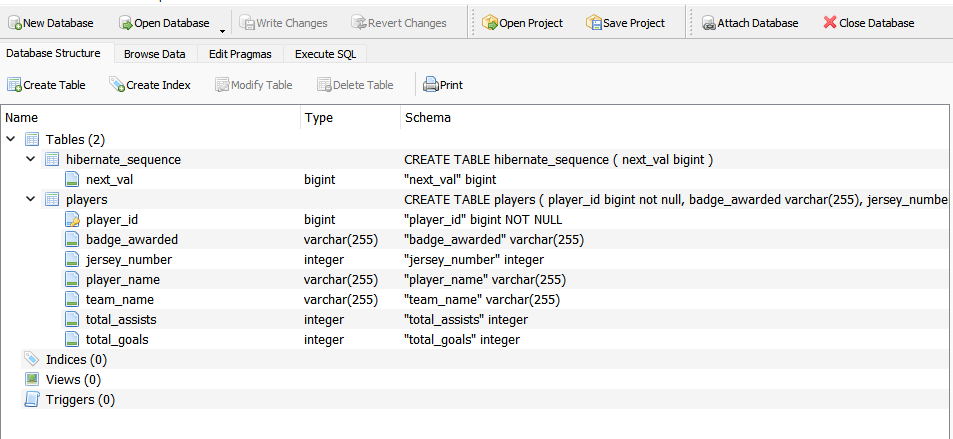
1. **Maven Lifecycle Management:** Throughout the development process, the project was cleaned to remove all files generated by the previous build, installed .jar files in the local repository, and updated to fetch any new dependencies.
2. **Transactional operations:** In the service layer of the application, **@Transactional** annotation is utilized to ensure the consistency and integrity of our database during update and delete operations.
3. **Database Modifications:**
4. **DB Browser for SQLite:**

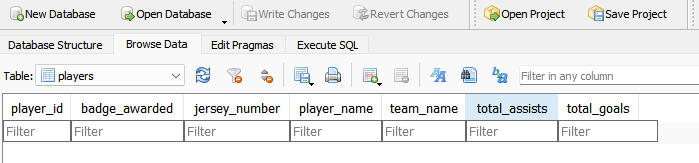
* The project was initiated by opening DB Browser for SQLite.
* Clicked on **Open Database** -> select **goteam.db** file to access and modify the GoTeam database.

1. **Application Up and Running:**

****

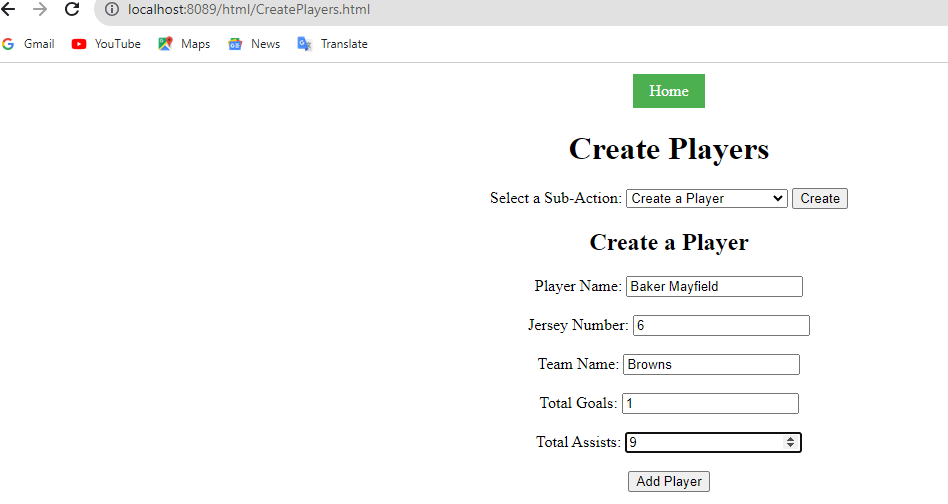
1. **Creating players table in sqlite:**

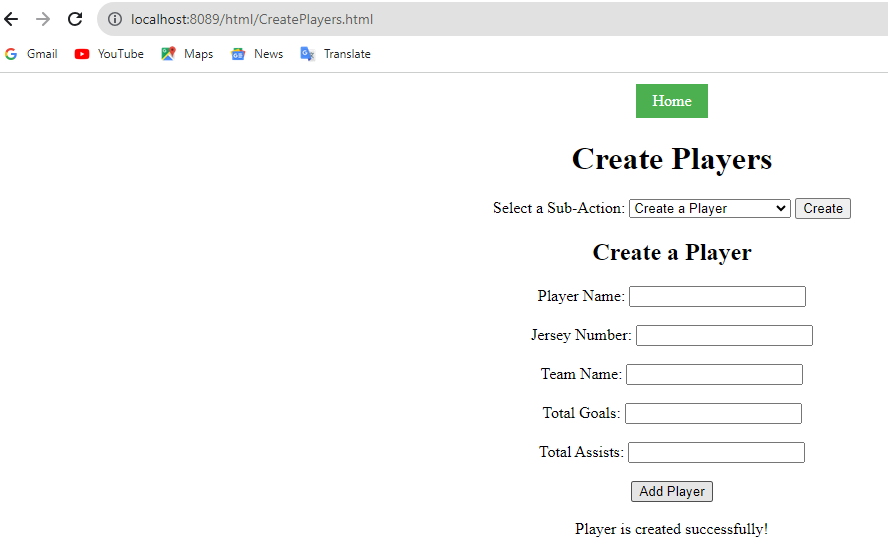
****

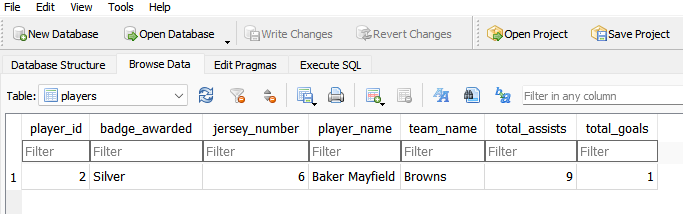
****

1. **Inserting players to the database via UI:**

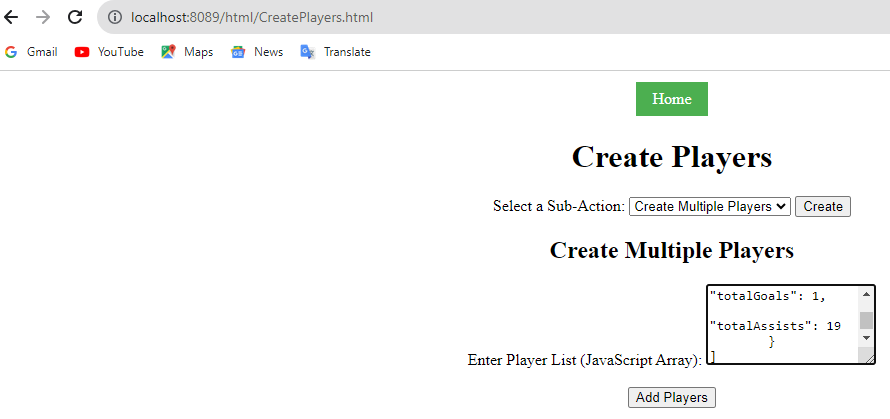
**Inserting single player:**

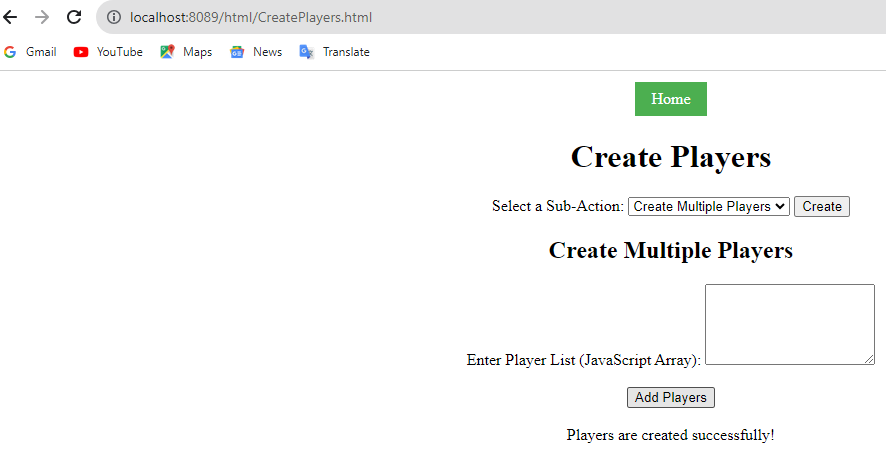
****

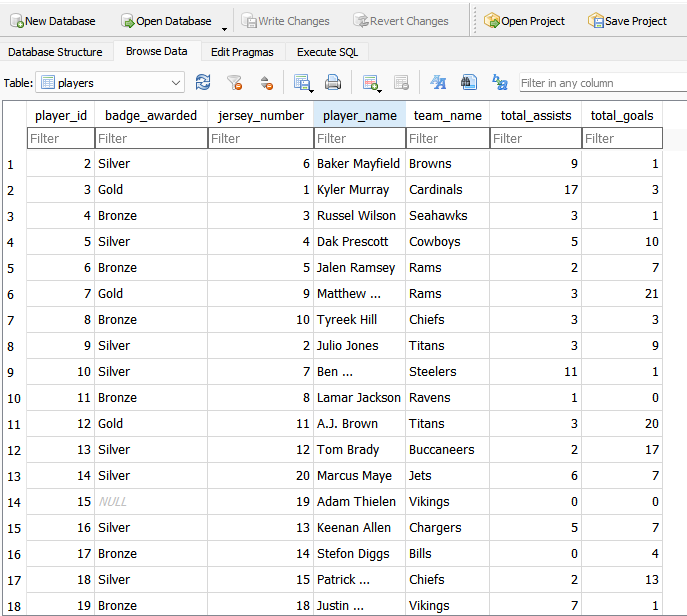
****

****

**Creating multiple players:**

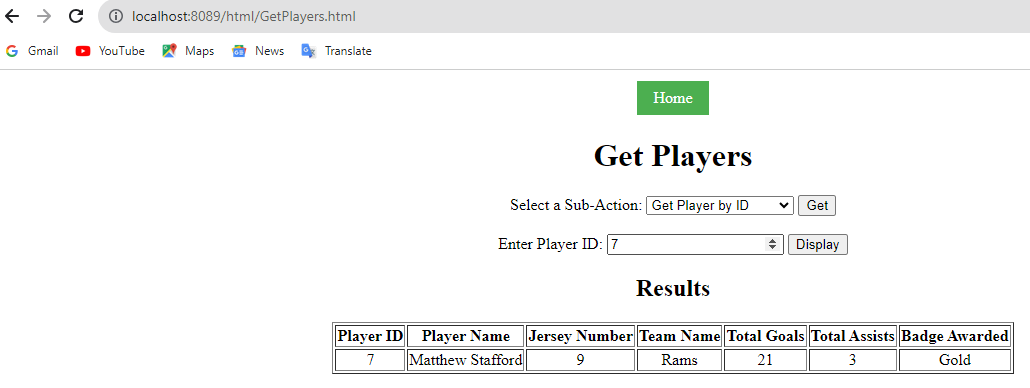
****

****

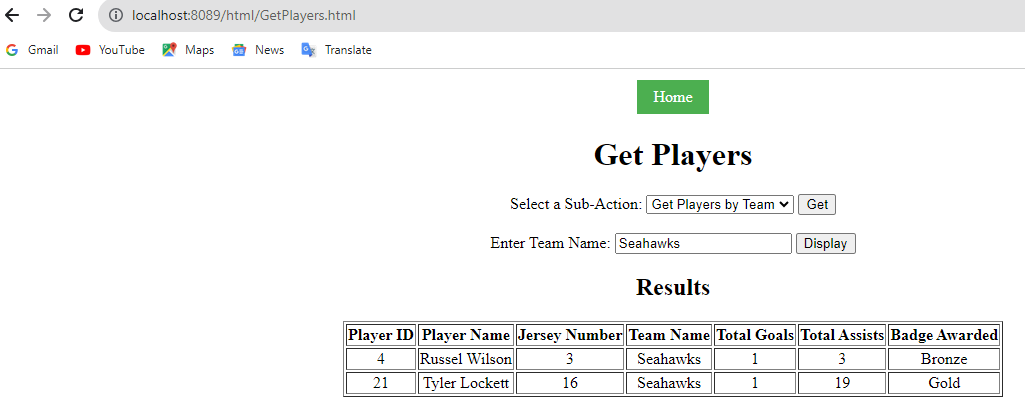
****

1. **Reading players from Sqlite:**

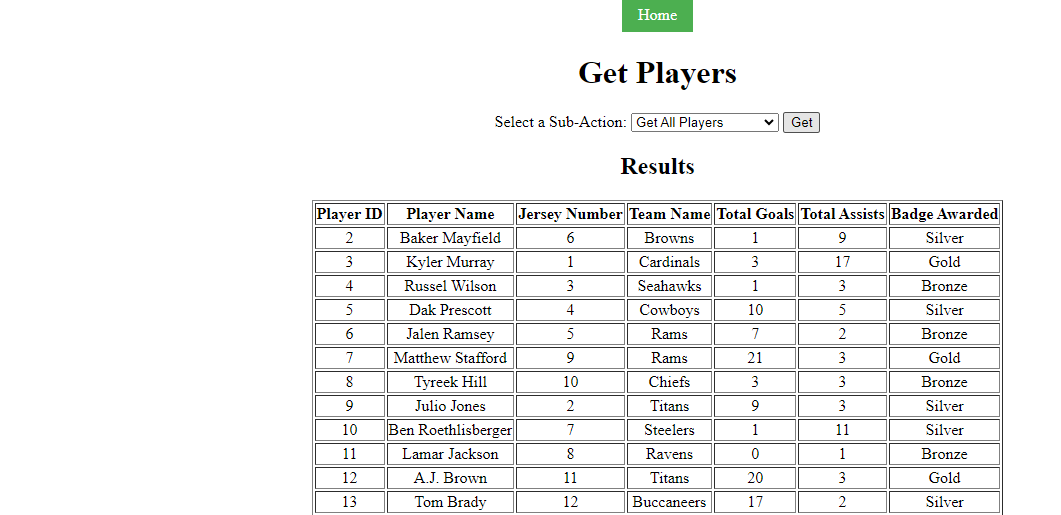
**Reading player by id:**

****

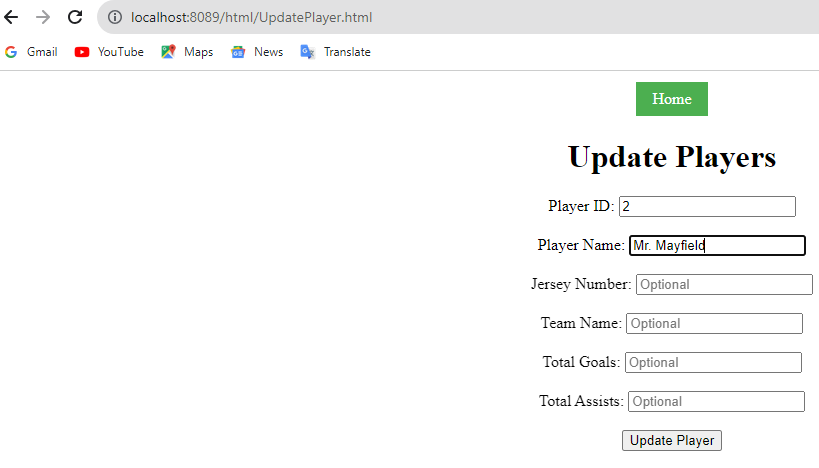
**Reading players by teamname:**

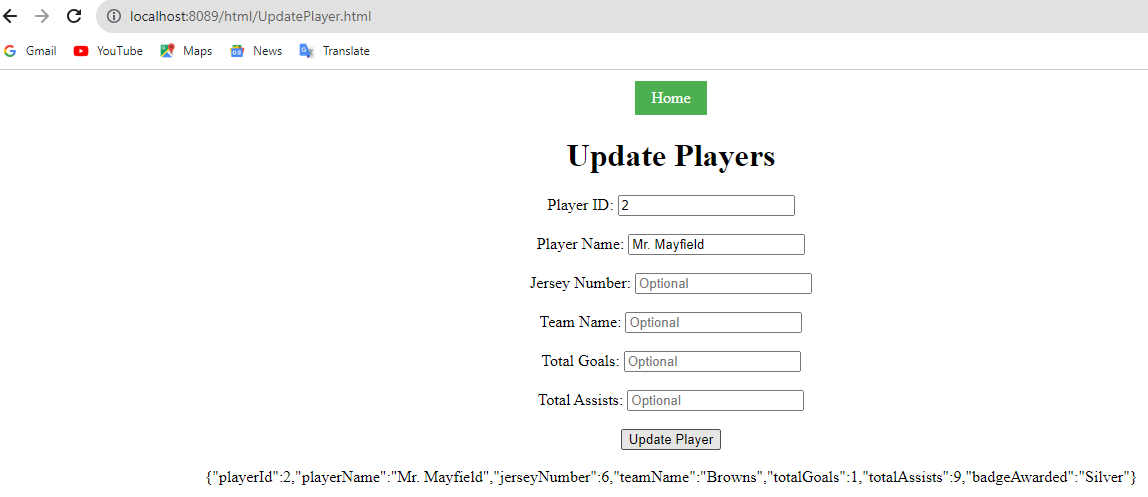
****

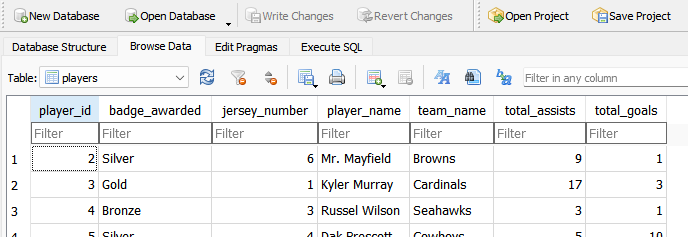
**Reading all players:**

****

1. **Updating Players in the database via UI:**

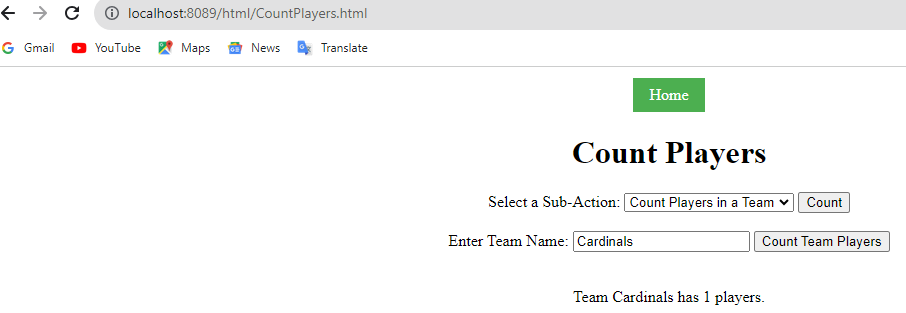
****

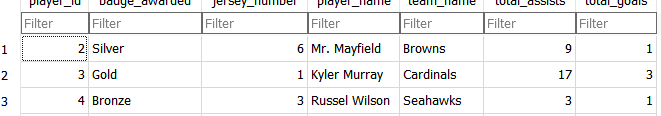
****

****

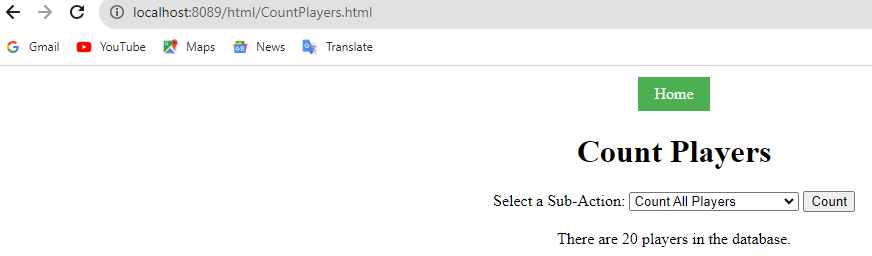
1. **Counting players:**

**Counting players in a team:**

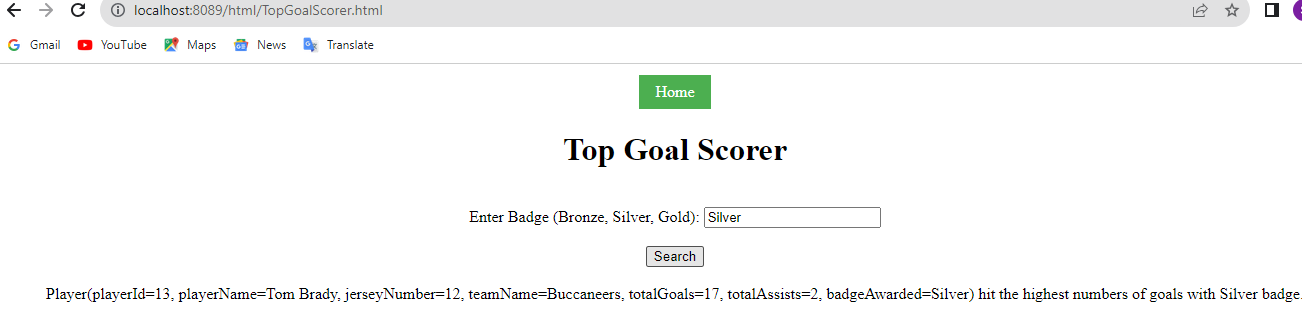
****

****

**Counting all players:**

****

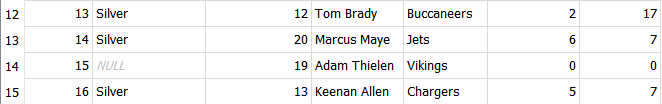
1. **Top Goal Scorer:**

****

****

****

****

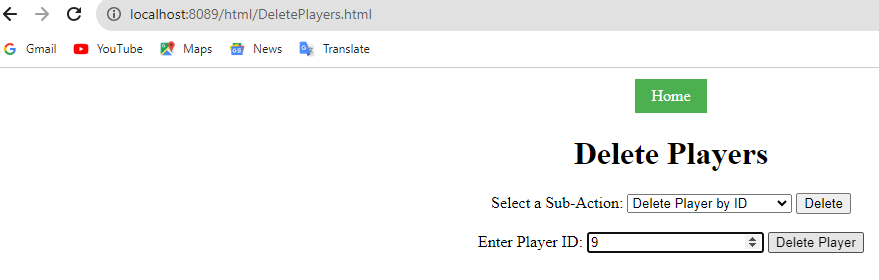
****

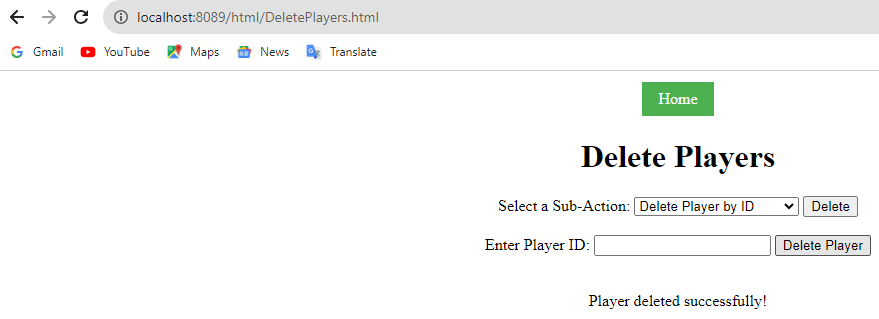
****

****

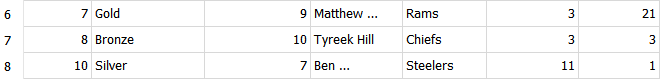
1. **Deleting players from Sqlite using UI:**

**Deleting player by id:**

****

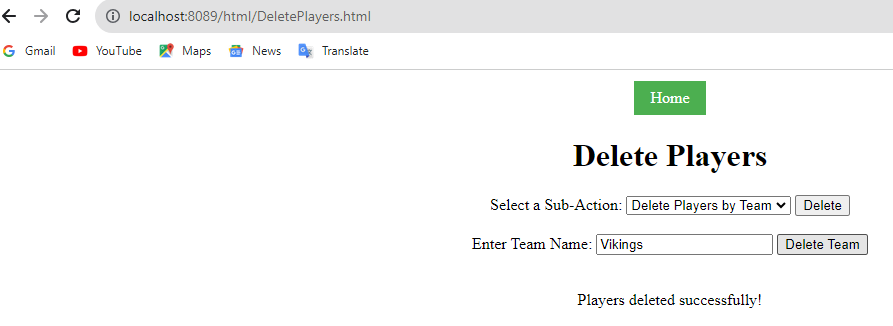
****

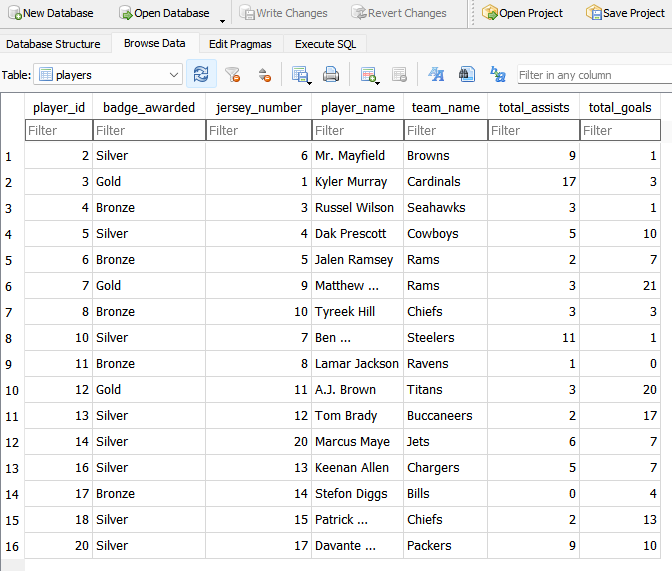
****

****

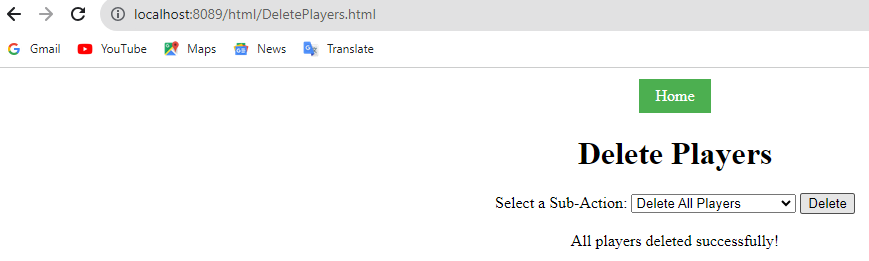
**Deleting players by team:**

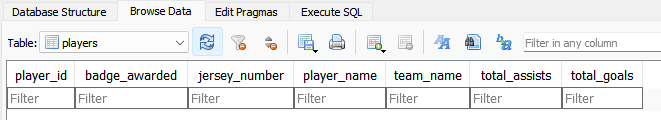
****

****

****

**Deleting all players:**

****

****

1. **Application of DRY Principles:** It is all about reducing repetition in the code, and while this is often associated with the actual source code of the application, it can also apply to configuration and setup, especially in the context of a Spring Boot application with a front-end.
2. **Centralized Configuration:**

* **application.properties:** This file centralizes the configuration of the application, including database connection settings, JPA/Hibernate settings, and logging settings. By placing all of this configuration in one place, I avoid repeating these settings in different parts of the application, adhering to the DRY principle.

1. **Maven Dependencies:** Using the Spring Boot POMs and specifying dependencies means that we don’t have to manually manage the version of each dependency or their transitive dependencies. This is a form of DRY, as you are not repeating version numbers and dependency information across different projects. Similarly, by configuring your build and plugins in **pom.xml**, you ensure a consistent build and packaging process, which adheres to the DRY principle.
2. **Consistent Database Interaction:** Spring Data JPA is used as suggested by the presence of JPA properties in **application.properties**, it’s like using repositories to interact with the database. These repositories provide a consistent way to perform CRUD operations, queries, etc., without having to write repetitive SQL queries or JDBC code.
3. **Front-End Interaction:** By configuring the back-end properly and exposing it to the front-end can consistently interact with the back-end without needing to know the details of the database or other internals of the server. This ensures any changes in the database or logic only need to be updated in one place, adhering to the DRY principle.