

A thick dark blue vertical bar runs down the left side of the page. To its right, several thin, curved lines in dark blue and light grey sweep upwards and outwards from the bottom left corner.

09/05/2021

Soccer Player System

Distributed Systems Assignment 2020

Sean Gavin
A00251388

Table of Contents

Introduction.....	2
Project Overview	2
Main Window	4
Menu Options.....	5
Setting Up the Project	5
Main Function.....	6
Requirements.....	6
Players Table	14
Video	15

Introduction

I was given the task of developing a Jax-Rs RESTful web application that includes both a client and a server for sending and receiving HTTP requests. The client application is expected to parse the responses and output them to a java swing GUI using an XMLPullParser. The details will be saved in an HSQLDB database.

This project uses an Apache Tomcat 9.0 server to host a database that stores and manipulates information about Players. The data is stored in a single HSQLDB table, which includes an Id (primary key), Player_name, Overall, Club, League, Nationality, Position and Age. Both the Interface and Postman can be used to retrieve, add to, edit and delete data.

Project Overview

The System is implemented into multiple different packages:

Server Side:

com.sean.server.dao - Data Access Object for players (CRUD).

com.sean.server.resource - package which is server paths running through servlet in the web.XML file.

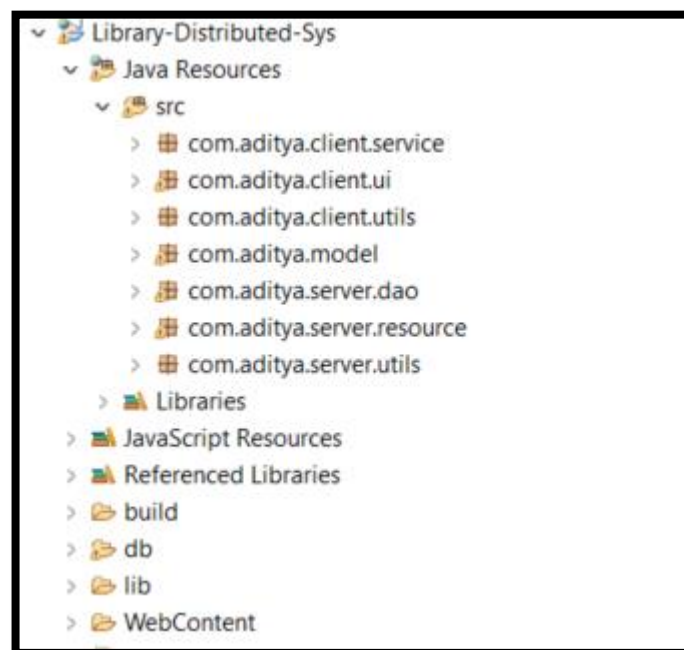
com.sean.server.utils – Database connection and Startup Listener to create Database.

Client side

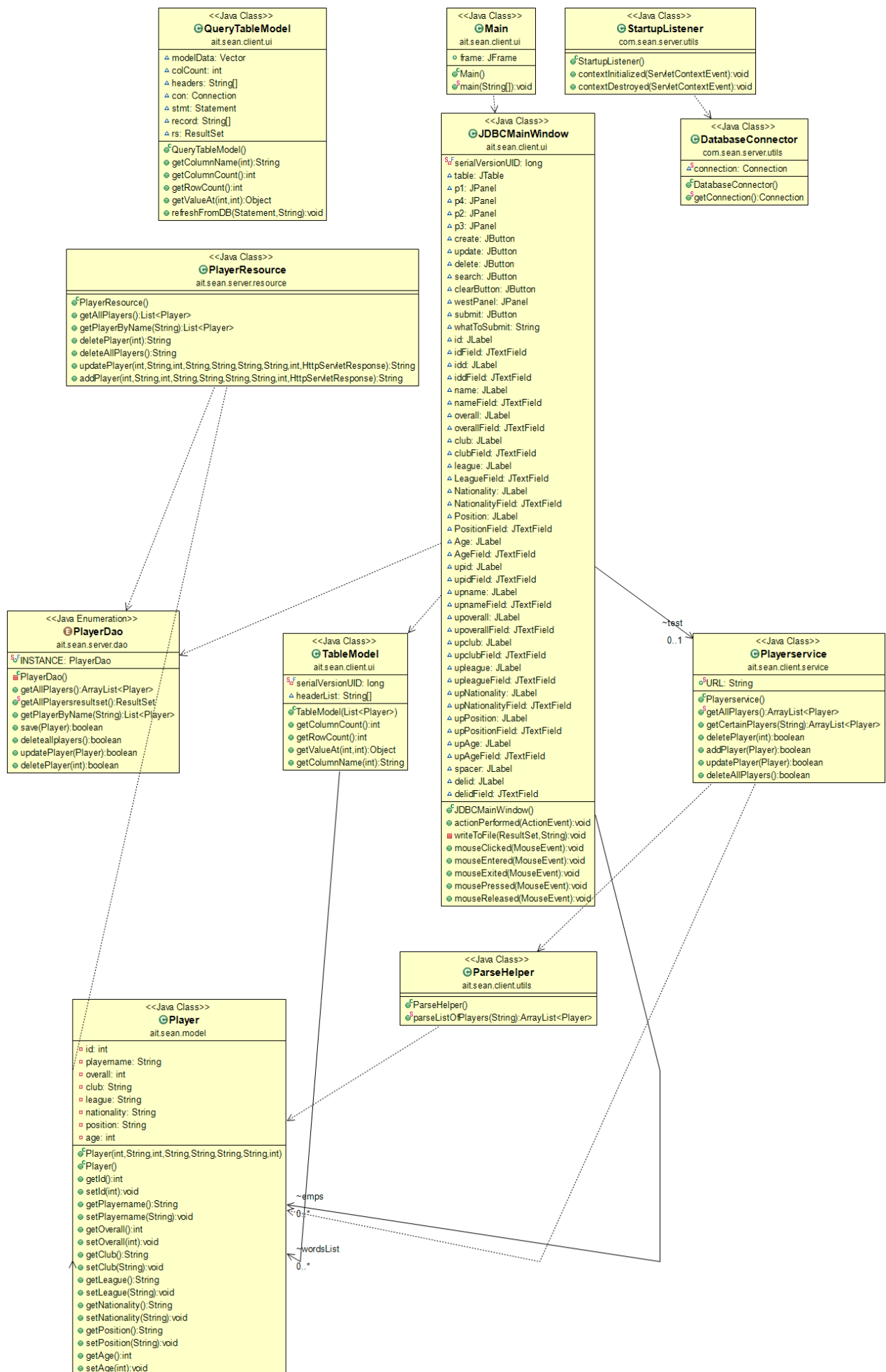
com.sean.client.service – this is the class at the client side that processes all requests to the server and accepts the response.

com.sean.client.ui - GUI that sends all the requests to the Servers and then displays all the received responses to the GUI.

com.sean.client.utils – contains XMLPullParser for displaying the information in the GUI.



Soccer Player project class diagram below:



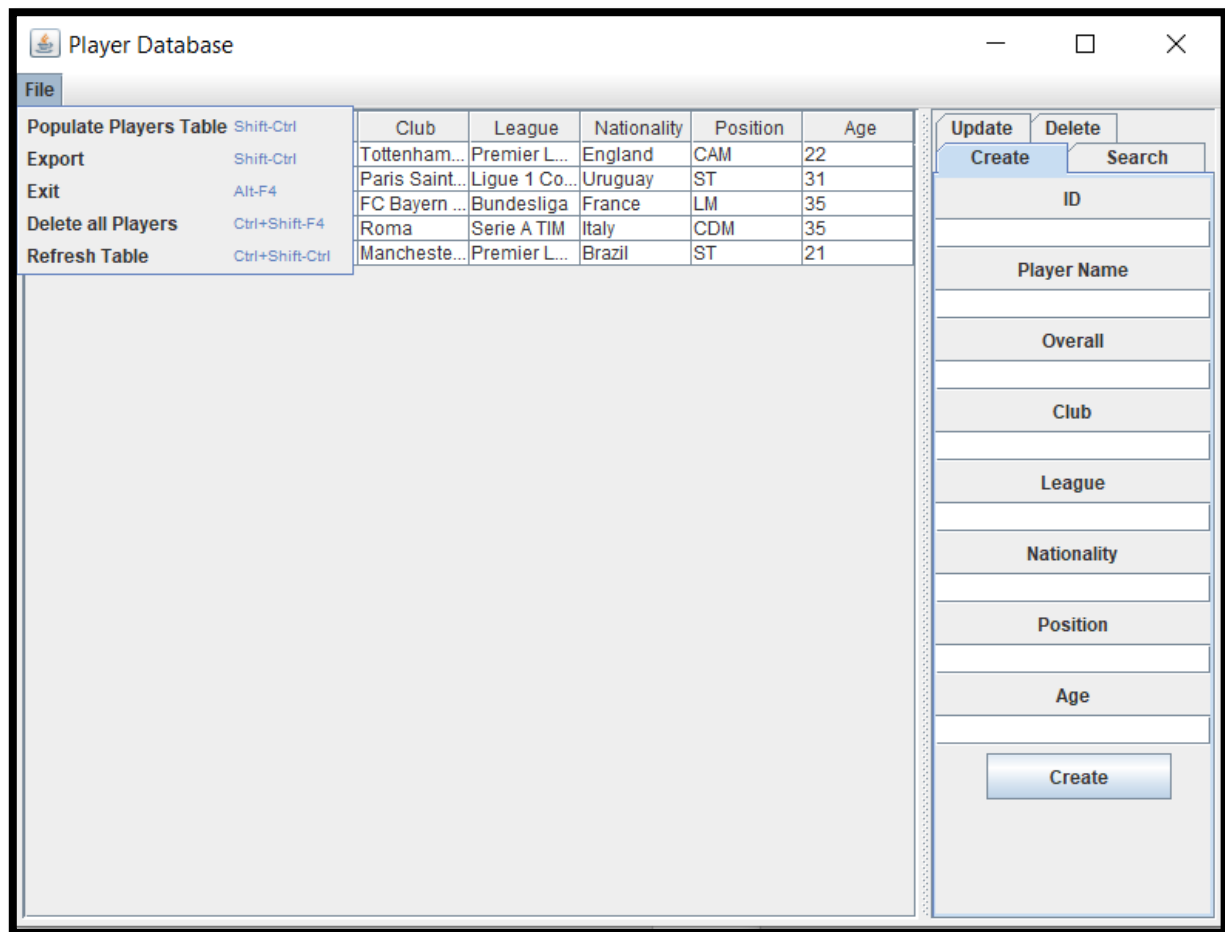
Main Window

id	playername	Overall	Club	League	Nationality	Position	Age
1	Dele Alli	85	Tottenham...	Premier L...	England	CAM	22
2	Edinson C...	90	Paris Saint...	Ligue 1 Co...	Uruguay	ST	31
3	Franck Rib...	86	FC Bayern ...	Bundesliga	France	LM	35
4	Daniele D...	84	Roma	Serie A TIM	Italy	CDM	35
5	Gabriel Fe...	84	Mancheste...	Premier L...	Brazil	ST	21

The frame of my client application is shown above. I tried to keep the user interface design as basic as possible so that the user could easily access the application. A table on the left-hand panel contains all of the information for the Players database. The panel on the right contains a selection panel which allows user to choose different operation such as update, delete, add and search. Each section panel contains buttons which carry out different operations for that section.

The table is created when the application is started, and the data is retrieved from the database using the Jax-RS server, then parsed with the XMLPullParser and added to a List. The numerous CRUD operations that I have designed into the application are shown on the right-hand side.

Menu Options



The Menu section of my client application is shown above. The user has the ability to use shortcuts to carry out the functionality in the menu. I have implemented a number of different functionalities into the menu section which makes the application helpful for users.

Setting Up the Project

Steps for First Time Use:

1. Run Tomcat Server (VERSION USED 9.0).
2. RUN managedDB script in ANT.
3. Ant -> Manage. (Check out the Players table)
4. Run Client-side GUI (MainClass) as a Java Application. If database table Players doesn't exist, the table will be created when main class is ran(StartupListener).
5. Select Populate Players Table in menu bar option to insert data if data doesn't exists in the database already.

Main Function

Data table – Returns the data from players database in a JTable format.

Delete - Enter an Id number of player and it will be removed from the database.

Update – Click on a player in the JTable and Update box's will be populated with that players details. The user can then manipulate the players data.

Create – Enter a new players data in order to be added to the database.

Search Player – Enter part of a Player's name and a new JTable will be generated on the user search.

Clear – Removes data from textboxes in the GUI and lets the user fill in their own data.

Menu bar option

Populate Players Table - Creates Player entities for the table.

Export – Exports all the data from the player tables to an excel sheet (comma separated values).

Exit- Allows the user to exit the application.

Delete All Players – Deletes all entries from the players database.

Refresh Player – Refresh's the data from the database for the JTable.

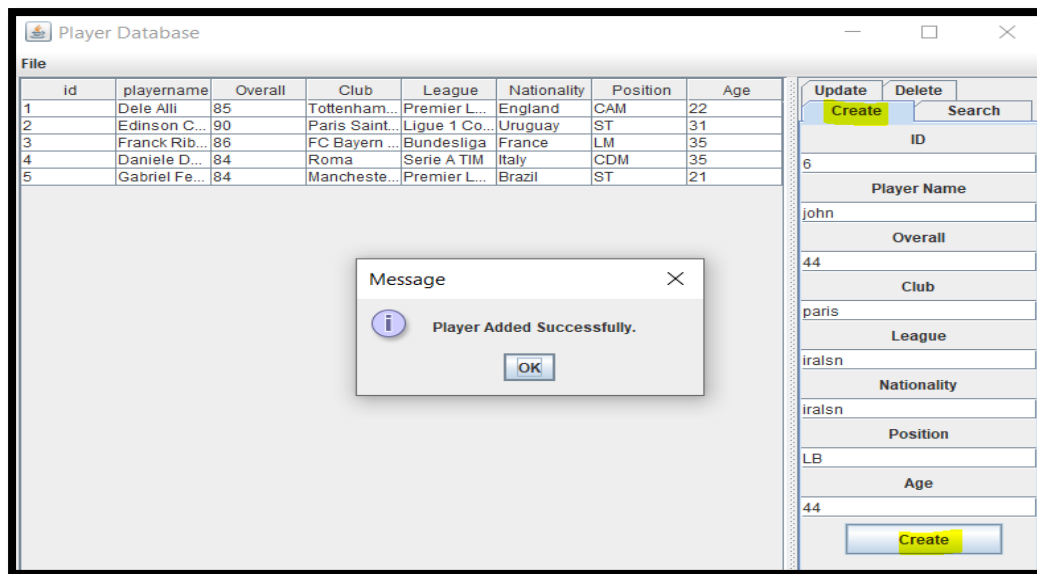
Requirements

1. "Build a client application that sends all of the HTTP requests GET/PUT/POST/DELETE."

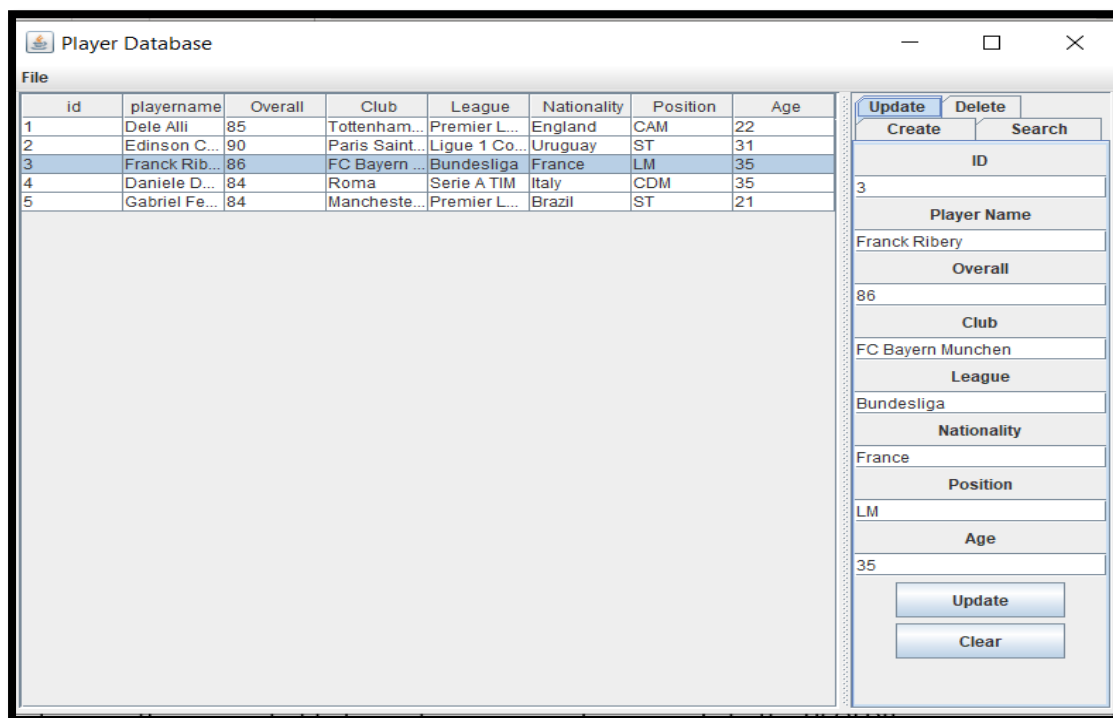
The Screenshots below contain the GUI which the user interacts with to send http request and receive output from the database. Any information received to the client application is constantly updated to reflect the information in the database.

I have given a summary of each of these commands in the main function section, I will now go into more detail about each GET/PUT/POST/DELETE client application HTTP request.

Post – A screen appears when the user clicks the "Add Player" button on the bottom right corner of the create player panel, allowing them to enter information for the new player they wish to add. The GUI sends a request to the server and is automatically displayed in the database after all fields are filled in and the user presses create.



PUT – Select a row from the table as shown below, edit the text fields in the hand side of the GUI and then click the "UpdatePlayer" button once done to update the details of a single player. Since the ID field is the primary key for each row, it cannot be changed. When the update button is pressed, the GUI refreshes, so there is no need to restart it; the updates are immediately visible.



Delete – We select a player from the table by clicking the row, just as we did with the put command. Rather than changing the data in the text fields, we simply press the delete button in the bottom right corner of the screen this time.

This eliminates the player from the database automatically by submitting a request to the server and the GUI updates instantly, as a result there is no need to refresh the GUI to see the

most recent database changes.



The screenshot shows the 'Player Database' application window. It features a table with 8 columns: id, playername, Overall, Club, League, Nationality, Position, and Age. The table contains 5 rows of player data. To the right of the table is a search panel with buttons for 'Update', 'Delete', 'Create', and 'Search'. Below these buttons is a text input field labeled 'ID' containing the value '5', and a 'Delete' button.

id	playername	Overall	Club	League	Nationality	Position	Age
1	Dele Alli	85	Tottenham...	Premier L...	England	CAM	22
2	Edinson C...	90	Paris Saint...	Ligue 1 Co...	Uruguay	ST	31
3	Franck Rib...	86	FC Bayern ...	Bundesliga	France	LM	55
4	Daniele D...	84	Roma	Serie A TIM	Italy	CDM	35
5	Gabriel Fe...	84	Mancheste...	Premier L...	Brazil	ST	21

GET - Return the information of all the players in the database , when you launch the application as first this process is carried out as shown in the Table above. The user can also send a request to retrieve a certain player by searching for their name in the search panel.



Delete All – The user can delete all users in the table by clicking the “Delete All players” in the menu bar options. Once the client makes this http request the sever executes the command and the data is deleted from the database.

If the user enters input into the GUI, they will receive an output to ensure them if the operation they carried out is successful or not.



2. “Build a server application using tomcat server, that responds to all of the HTTP requests GET/PUT/POST/DELETE”

GET

GET: GET is used to return values from the database to the GUI; it has a variety of features in the project. Many of the values in the database table are fetched with a single GET. The second GET is used to retrieve the values associated with a given name; the name is transferred as a PathParam in the URI. The client receives the response in XML format from GET.

```
@Path("/player")
public class PlayerResource {
    @GET
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON, MediaType.TEXT_XML })
    public List<Player> getAllPlayers() {
        return PlayerDao.INSTANCE.getAllPlayers();
    }
    @GET
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON, MediaType.TEXT_XML })
    @Path("/{name}")
    public List<Player> getPlayerByName(@PathParam("name") String Player_Name) {
        return PlayerDao.INSTANCE.getPlayerByName(Player_Name);
    }
}
```

GET Request to the server and Response:

The screenshot shows a REST client interface with a GET request to `localhost:8080/Footballproject/player`. The response status is 200 OK, with a time of 20 ms and a size of 1.28 KB. The response body is displayed in XML format, showing a list of players.

KEY	VALUE	DESCRIPTION
Key	Value	Description

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<players>
  <player>
    <id>1</id>
    <playername>Dele Alli</playername>
    <overall>85</overall>
    <club>Tottenham Hotspur</club>
    <league>Premier League</league>
    <nationality>England</nationality>
    <position>CAM</position>
    <age>22</age>
  </player>
  <player>
    <id>2</id>
    <playername>Edinson Cavani</playername>
    <overall>90</overall>
    <club>Paris Saint-Germain</club>
  </player>

```

PUT

PUT is used to update a single row in a database table that has a particular ID. PUT accepts parameters in the form as FormParams, which are forwarded to the server from the client.

```
//Update certain attributes for a Player using their Specific ID
@PUT
@Produces(MediaType.APPLICATION_XML)
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Path("/{id}")
public String updatePlayer(
    @PathParam("id") int id,
    @FormParam("playerName") String Player_Name,@FormParam("overall") int Overall,
    @FormParam("club") String Club,@FormParam("league") String League,
    @FormParam("nationality") String Nationality,@FormParam("position") String Position,
    @FormParam("age") int Age,@Context HttpServletResponse servletResponse) {
    Player emp = new Player(id,Player_Name,Overall,Club,League,Nationality,Position,Age);

    return PlayerDao.INSTANCE.updatePlayer(emp) ? "Updated Player Successfully" : "Error Updating Player";
}
```

Put Request to the server and Response:

The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** localhost:8080/Footballproject/player/6
- Body Type:** x-www-form-urlencoded
- Parameters:**

KEY	VALUE	DESCRIPTION
id	6	
playerName	John Smith	
overall	44	
club	sdf	
league	sdf	
- Response:** Updated Player Successfully (Status: 200 OK, Time: 63 ms, Size: 190 B)

POST

POST is used for inserting data in the database tables. POST at the server side accepts all the parameters as FormParams and then we cast it to Model class object by passing them to the Model class constructor and then this Model class object is passed to the DAO class which uses the getters of the Model class to extract the parameters and insert the data in the database table.

```
//Add new Player to the Database
@POST
@Produces(MediaType.TEXT_HTML)
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public String addPlayer(
    @FormParam("id") int id,
    @FormParam("playerName") String Player_Name,
    @FormParam("overall") int Overall,
    @FormParam("club") String Club, @FormParam("league") String League,
    @FormParam("nationality") String Nationality, @FormParam("position") String Position,
    @FormParam("age") int Age, @Context HttpServletResponse servletResponse) throws ClassNotFoundException, SQLException {
    Player emp = new Player(id, Player_Name, Overall, Club, League, Nationality, Position, Age);
    return PlayerDao.INSTANCE.save(emp) ? "Added Player Successfully" : "Error Adding Player";
}
```

POST Request to the server and Response:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:8080/Footballproject/player/
- Params:** Authorization, Headers (9), Body (selected), Pre-request Script, Tests, Settings
- Body Type:** x-www-form-urlencoded
- Body Data:**

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> id	6	
<input checked="" type="checkbox"/> playerName	sfd	
<input checked="" type="checkbox"/> overall	44	
<input checked="" type="checkbox"/> club	sdf	
<input checked="" type="checkbox"/> league	sdf	

Response: Status: 200 OK, Time: 9 ms, Size: 182 B

Body: Added Player Successfully

DELETE

DELETE is used to delete data from a single row in a database table that is marked by an ID passed by the client side. DELETE accepts the ID as a PathParam, which means the ID is transferred to the server along with the URI from the client.

```
//Delete player by specific ID
@DELETE
@Produces(MediaType.TEXT_HTML)
@Path("/{id}")
public String deletePlayer(@PathParam("id") int id) {
    return PlayerDao.INSTANCE.deletePlayer(id) ? "Player Deleted Successfully" : "Error Deleting Players";
}
//Delete all Players
@DELETE
@Produces(MediaType.TEXT_HTML)
public String deleteAllPlayers() {
    return PlayerDao.INSTANCE.deleteallplayers() ? "All Players Deleted Successfully" : "Error Deleting Players";
}
```

Delete Request to the server and Response:

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** localhost:8080/Footballproject/player/6
- Status:** 200 OK
- Time:** 9 ms
- Size:** 184 B
- Response Body:** 1 Player Deleted Successfully

KEY	VALUE	DESCRIPTION
Key	Value	Description

3. "The client application will parse the response using XMLPullParser and outputs to the GUI" + "A tomcat server that responds to all of the HTTP requests GET/PUT/POST/DELETE"

I will discuss how the Get Response works in relation to the XMLPullParser class in the project.

When we execute a client-side program, the client sends an HTTP GET Request to the server, which accepts it. The GET method on the server then calls the getAllPlayers() method on the DAO class, which returns all the values as an ArrayList, which is then translated to XML and sent back to the client.

HTTP Client Service GET Example:

```
public static String URL = "/Footballproject/player/"; // url server
//Http request to get all players
public static ArrayList<Player> getAllPlayers(){
    try {
        //setting the configuration to make the request
        URI uri = new URIBuilder()
            .setScheme("http")
            .setHost("localhost")// url server
            .setPort(8080)
            .setPath(URL)
            .build();

        HttpGet httpReq = new HttpGet(uri);
        httpReq.setHeader("Accept", "application/xml");

        CloseableHttpResponse httpResponse = HttpClient.createDefault().execute(httpReq);
        // saving Response xmlpullparser
        return ParseHelper.parseListOfPlayers(EntityUtils.toString(httpResponse.getEntity()));
    } catch (URISyntaxException | IOException ex) {
        ex.printStackTrace();
    }
    return null;
}
```

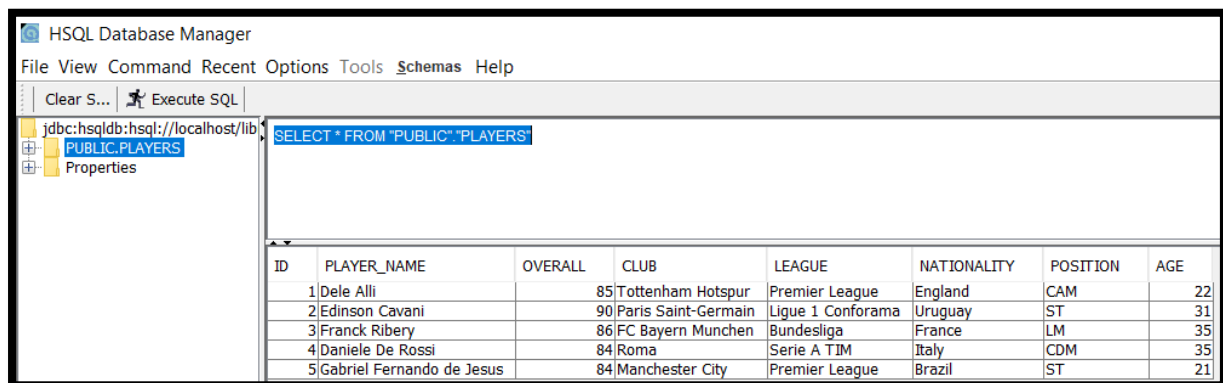
When the Client receives this response, it transfers it to the XML Parser class, which decodes the data to normal text and then casts it to TableModel, which is then sent to the Products Report class, which sets the returned TableModel to the Textarea in the JFrame.

id	playername	Overall	Club	League	Nationality	Position	Age
1	Dele Alli	85	Tottenham...	Premier L...	England	CAM	22
2	Edinson C...	90	Paris Saint...	Ligue 1 Co...	Uruguay	ST	31
3	Franck Rib...	86	FC Bayern ...	Bundesliga	France	LM	35
4	Daniele D...	84	Roma	Serie A TIM	Italy	CDM	35
5	Gabriel Fe...	84	Mancheste...	Premier L...	Brazil	ST	21

I carried out HTTP client requests for PUT, POST, DELETE functionality which can be seen in the Playerservice class.

4. "The data in the response will be taken from an HSQLDB database."

For storing and accessing the data, I used HSQLDB as the data model. The HSQLDB is linked directly to the Server-Side application, which performs all database operations and only sends the response to the client.



The screenshot shows the HSQL Database Manager interface. The left pane displays the database structure with 'jdbc:hsqldb:hsqldb://localhost/lib' selected, and 'PUBLIC.PLAYERS' is highlighted. The main pane shows the SQL query 'SELECT * FROM "PUBLIC"."PLAYERS"' and the resulting data table.

ID	PLAYER_NAME	OVERALL	CLUB	LEAGUE	NATIONALITY	POSITION	AGE
1	Dele Alli	85	Tottenham Hotspur	Premier League	England	CAM	22
2	Edinson Cavani	90	Paris Saint-Germain	Ligue 1 Conforama	Uruguay	ST	31
3	Franck Ribery	86	FC Bayern Munchen	Bundesliga	France	LM	35
4	Daniele De Rossi	84	Roma	Serie A TIM	Italy	CDM	35
5	Gabriel Fernando de Jesus	84	Manchester City	Premier League	Brazil	ST	21

I built a Players table for the project, which was identical to the Book and User tables that had previously been used in labs during the semester. The rows in the table were as follows:

- ID – Key to identify each player.
- Player_Name – Name of the footballer, example would be lionel messi.
- Overall – (integer) The rating of the soccer player, example 55.
- Club – the professional team the player is part of, example Arsenal.
- League – The professional league the player plays for, example Premier league.
- Nationality – Country of origin for the player, example Ireland.
- Position – Length (3) Position on the football pitch, example RWB
- Age – (integer) the age of the football players, example 27

Players Table

```
CREATE TABLE PLAYERS(  
ID INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH 1,  
INCREMENT BY 1) NOT NULL PRIMARY KEY,  
PLAYER_NAME VARCHAR(35) DEFAULT NULL,  
OVERALL INTEGER DEFAULT NULL,  
CLUB VARCHAR(27) DEFAULT NULL,  
LEAGUE VARCHAR(25) DEFAULT NULL,  
NATIONALITY VARCHAR(22) DEFAULT NULL,  
POSITION VARCHAR(3) DEFAULT NULL,  
AGE INTEGER DEFAULT NULL)
```

Video

Click on link to view the video. If any issues accessing the video, please contact me.

<https://youtu.be/MYwc1p4hCLQ>