**Leaderboard DB**

Written by:

John Hall
Robert Peterson
Brenden Martensen

**Overview:**

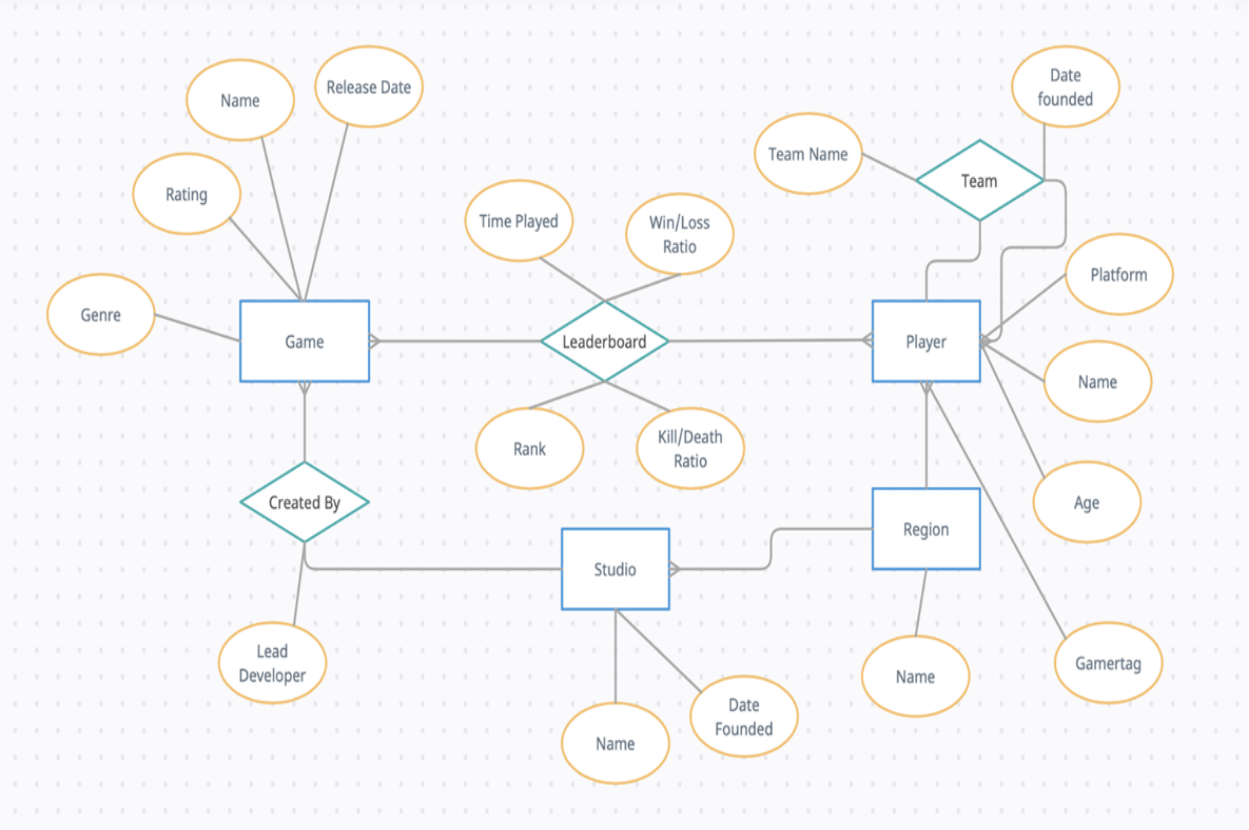
The Leaderboard group John Hall, Robert Peterson, and Brenden Martensen have developed a website for gamers, streamers, and viewers. Our website was created to show various things such as the top players for certain games and the top games based on the number of players and their ratings and much more.

The website will be similar to websites such as Tracker Network https://tracker.gg/ which is the main website that many gamers use to track their stats on video games. Our aim is to provide a similar experience but also show game rankings and leaderboards of a moshpit of games for first person shooter games. We also wanted to make it simple for viewers to learn more about their favorite gamer and games.

**Database Design:**

In this section we will give a visual depiction of our database schema and a high level description of all the tables and their attributes. These are used to know how our different tables are connected throughout our database.

ER Diagram:




Below are the various tables and there attributes

Table Name: createdby

- createdbyID: integer (primary key)
- studioID: integer (foreign key referencing studio.studioID)
- firstName: Var Char
- lastName: Var Char

Table Name: games

- gameID: integer (primary key)

- genre: Var Char
- rating: Decimal
- gameName: Var Char
- releaseDate: Date time
- createdbyID: integer (foreign key referencing studio.studioID)

Table Name: leaderboard

- gameID: integer (foreign key referencing game.gameID)
- playerID: integer (foreign key referencing player.playerID)
  (Use both gameID and playerID as primary key)
- timePlayed: integer
- winLossRatio: numeric
- killDeathRatio: numeric
- playerRank: Var Char

Table Name: players

- playerID: integer (primary key)
- platform: Var Char
- firstName: Var Char
- lastName: Var Char
- age: integer
- gamertag: Var Char
- teamID: integer (foreign key referencing team.teamID)
- regionID: integer (foreign key referencing region.regionID)

Table Name: region

- regionID: integer (primary key)
- regionName: Var Char

Table Name: studio

- studioID: integer (primary key)
- studioName: Var Char
- dateFounded: Date time
- regionID: integer (foreign key referencing region.regionID)

Table Name: teams

- teamID: integer (primary key)
- teamName: Var Char
- dateFounded: Date time

**Database Initialization:**

In this section we will go over how to initialize the database by creating the table and there respective attributes.

```
DROP TABLE IF EXISTS `createdby`;
CREATE TABLE `createdby` (
  `createdbyID` int NOT NULL,
  `studioID` int DEFAULT NULL,
  `firstName` varchar(45) DEFAULT NULL,
  `lastName` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`createdbyID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
INSERT INTO `createdby` VALUES (1,1,'Aaron','Keller'),(2,2,'Joe','Ziegler'),(3,2,'Andrei','van Roon');

DROP TABLE IF EXISTS `games`;
CREATE TABLE `games` (
  `gameID` int NOT NULL,
  `createdbyID` int DEFAULT NULL,
  `gameName` varchar(45) DEFAULT NULL,
  `releaseDate` datetime DEFAULT NULL,
  `genre` varchar(45) DEFAULT NULL,
  `rating` decimal(2,1) DEFAULT NULL,
  PRIMARY KEY (`gameID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
INSERT INTO `games` VALUES (1,1,'Overwatch 2','2022-10-04
00:00:00','FPS',1.9),(2,2,'Valorant','2020-03-02 00:00:00','FPS',4.0),(3,3,'League of Legends','2009-10-27
00:00:00','MOBA',2.9);

DROP TABLE IF EXISTS `leaderboard`;
CREATE TABLE `leaderboard` (
  `playerID` int NOT NULL,
  `gameID` int NOT NULL,
  `timePlayed` int NOT NULL DEFAULT '0',
  `killDeathRatio` decimal(2,1) NOT NULL DEFAULT '0.0',
  `winLossRatio` decimal(2,1) NOT NULL DEFAULT '0.0',
  `playerRank` varchar(25) NOT NULL,
  PRIMARY KEY (`gameID`,`playerID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
INSERT INTO `leaderboard` VALUES
(2,1,2019,1.3,1.8,'Grandmaster'),(3,1,1892,0.9,3.7,'Grandmaster'),(6,1,1,1.0,1.0,'1'),(7,1,782,1.1,2.3,'Plati
num'),(8,1,2017,1.1,2.7,'Silver'),(12,1,178,0.6,0.7,'Gold'),(14,1,1401,1.5,2.4,'Bronze'),(17,1,1,1.0,1.0,'1'),(9
,2,1,1.0,1.0,'Bronze'),(10,2,400,1.2,1.4,'Top
500'),(13,2,1971,1.8,1.8,'Silver'),(14,2,1132,2.6,1.7,'Diamond'),(15,2,163,0.4,2.4,'Bronze'),(16,2,782,1.2,1.
9,'Diamond'),(17,2,1632,1.3,2.8,'Radiant'),(1,3,1,1.0,1.0,'1'),(3,3,800,0.3,1.8,'Diamond'),(4,3,2978,2.0,1.4,'
Master'),(5,3,197,1.3,2.8,'Challenger'),(7,3,1461,0.7,3.4,'Radiant'),(10,3,973,0.8,3.4,'Gold'),(11,3,895,1.0,
2.2,'Masters'),(12,3,178,0.6,0.7,'Master'),(14,3,1132,1.1,3.3,'Diamond'),(17,3,587,1.3,1.2,'Silver');

DROP TABLE IF EXISTS `players`;
```

```
CREATE TABLE `players` (
  `playerID` int NOT NULL,
  `regionID` int NOT NULL,
  `teamID` int DEFAULT NULL,
  `platform` varchar(45) DEFAULT NULL,
  `firstName` varchar(45) NOT NULL,
  `lastName` varchar(45) NOT NULL,
  `age` int NOT NULL,
  `gamertag` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`playerID`),
  UNIQUE KEY `playerID_UNIQUE` (`playerID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
INSERT INTO `players` VALUES
(1,1,1,'Xbox','Michael','Johnson',24,'MikeyJ1000'),(2,1,0,'Xbox','Jess','Smith',28,'JSmithy19'),(3,1,1,'PC','M
atthew','Delisi',23,'Super'),(5,2,2,'PlayStation','Daniel','Garcia',29,'DGarcia2022'),(6,2,2,'PlayStation','Emm
a','Martinez',25,'EMartinez14'),(7,3,3,'PC','Yuta','Nakamura',32,'YNakaGaming'),(8,3,0,'PC','Hana','Kim',26
,'HanaKPC'),(9,4,1,'Nintendo
Switch','Isabella','Fernandez',23,'IFernandez21'),(10,5,2,'Xbox','Liam','Brown',21,'LiamB_Gaming'),(11,5,1,
'Xbox','Charlotte','Wilson',30,'C_Wilson95'),(12,6,3,'PlayStation','Njabulo','Khumalo',26,'NJKhumalo23'),(1
3,6,0,'PlayStation','Thembelihle','Dube',31,'T_Dube_2023'),(14,1,0,'PC','Olivia','Davis',29,'TheBigO'),(15,1,
2,'PC','Ryan','Perez',27,'R_Perez_Gaming'),(16,2,3,'Nintendo
Switch','Marco','Rossi',25,'MRossi_NSwitch'),(17,2,0,'Nintendo
Switch','Sofia','Esposito',30,'SofiaEspositoGamer');

DROP TABLE IF EXISTS `region`;
CREATE TABLE `region` (
  `regionID` int NOT NULL,
  `regionName` varchar(50) NOT NULL,
  PRIMARY KEY (`regionID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
INSERT INTO `region` VALUES (1,'North America'),(2,'Europe'),(3,'Asia'),(4,'South
America'),(5,'Australia'),(6,'Africa');

DROP TABLE IF EXISTS `studio`;
CREATE TABLE `studio` (
  `studioID` int NOT NULL,
  `regionID` int DEFAULT NULL,
  `studioName` varchar(45) DEFAULT NULL,
  `dateFounded` datetime DEFAULT NULL,
  PRIMARY KEY (`studioID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
INSERT INTO `studio` VALUES (1,1,'Blizzard Entertainment','1991-02-08 00:00:00'),(2,1,'Riot
Games','2006-09-01 00:00:00');

DROP TABLE IF EXISTS `teams`;
CREATE TABLE `teams` (
  `teamID` int NOT NULL,
  `teamName` varchar(45) DEFAULT NULL,
  `dateFounded` datetime DEFAULT NULL,
```

```
  PRIMARY KEY (`teamID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
INSERT INTO `teams` VALUES (0,'No One','2000-01-01 00:00:00'),(1,'San Francisco Shock','2017-01-01
00:00:00'),(2,'Luminosity Gaming','2015-02-01 00:00:00'),(3,'Sentinels ','2018-06-01 00:00:00');
```

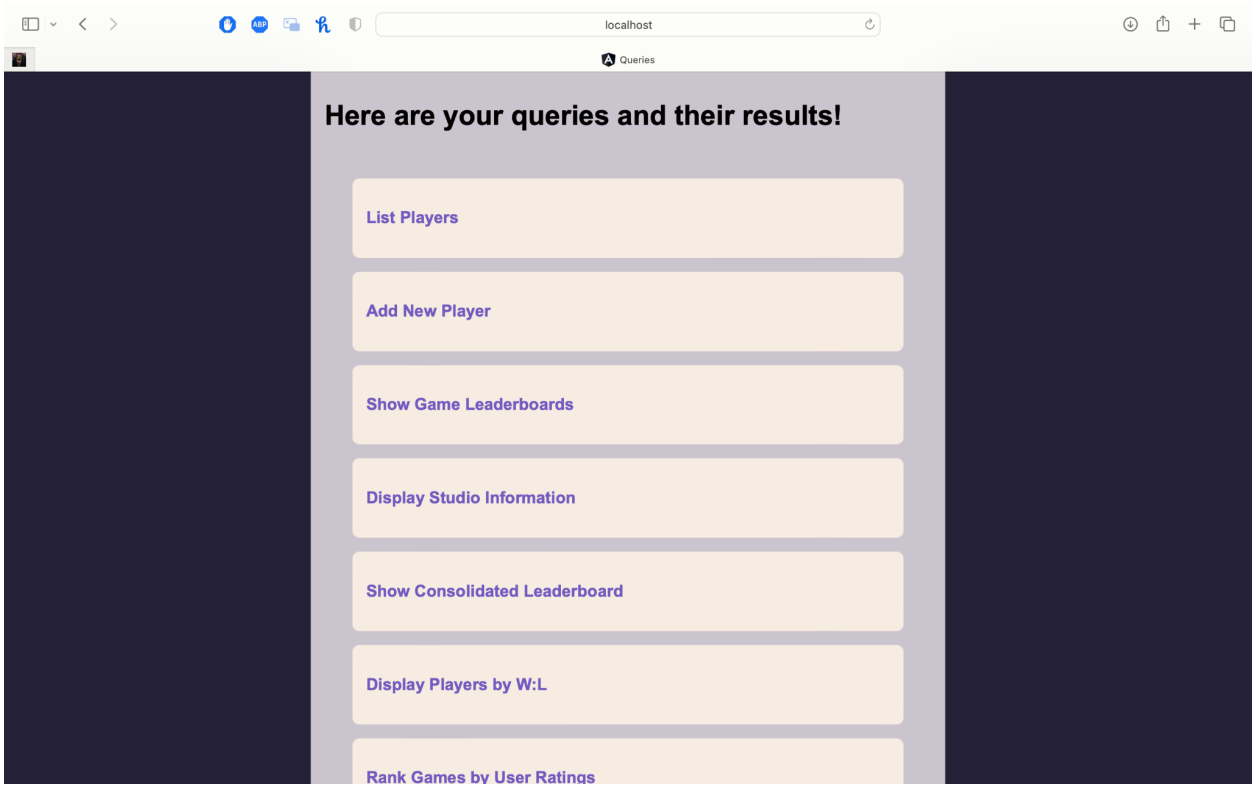
**Technical Layout:**

In this section we will go through the various technologies used to develop our website. The front end of our stack was built using a combination of HTML, CSS, JavaScript, and TypeScript using a library called Angular. Angular is a front end developer kit similar to React that, when used in conjunction with backend servers, can easily output database queries and perform simple route changes, such that page navigation is simple and easy for both the developer and the user. Using Hapi and NodeJS we also built a backend server that receives requests from the Angular front end and performs queries on our database. These requests are processed by the server and outputs a payload that is received by the Angular website, and processed into frontend data. Hapi serves as the middleman between Angular and MySQL and receives and processes website requests, however we used NodeJS to simplify this process and allow our Hapi server to be developed in JavaScript.

**Control Flow:**

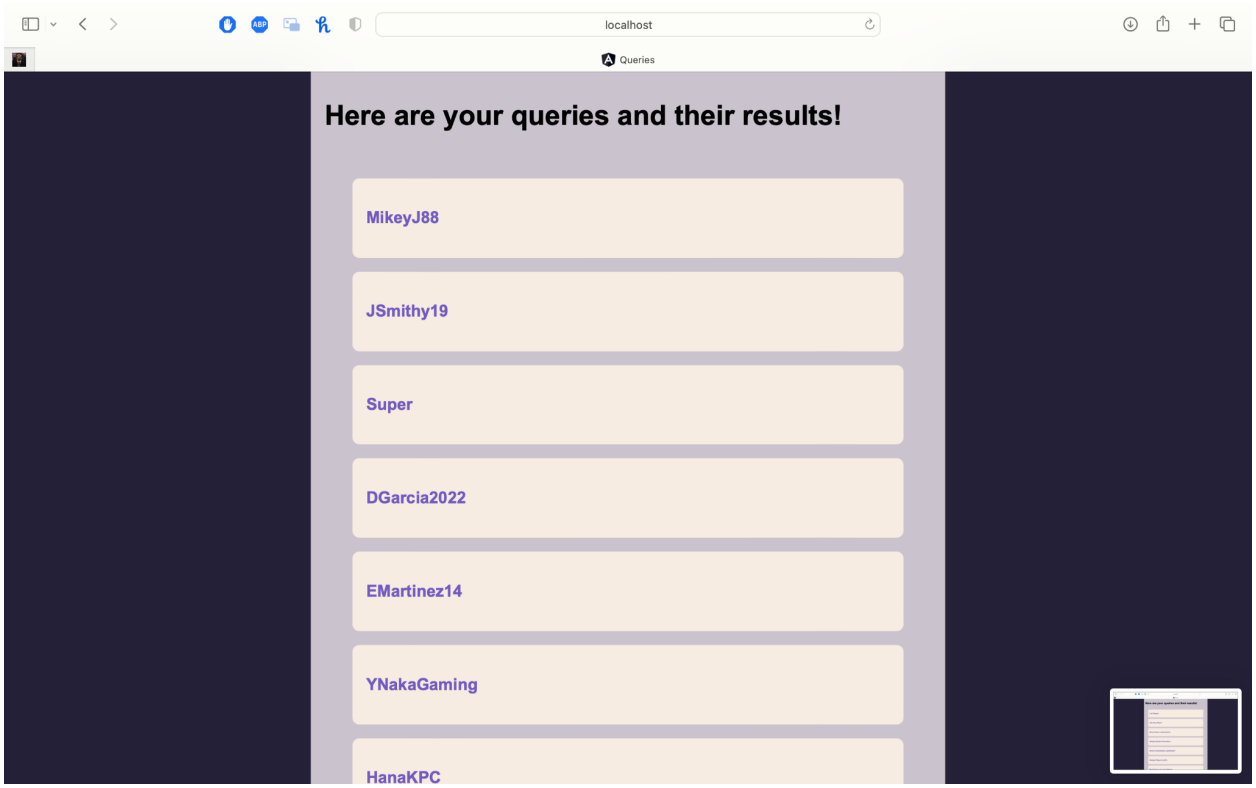
Main Page (1)

- List Players page (2)
    - Player data page (3)
        - Edit Player (4)
        - Add Leaderboard Data (5)
- Add New Players page(6)
- Show Game Leaderboards page(7)
    - Overwatch 2 query (8)
        - Player data page (3)
    - Valorant query (8)
        - Player data page (3)
    - League of Legends query (8)
        - Player data page (3)
- Display Studio Information page (9)
- Show Consolidated Leaderboard page (10)
    - Leaderboard data page (11)
        - Edit Leaderboard (12)
- Display Players by W:L page (13)
- Rank Games by User Ratings page (14)



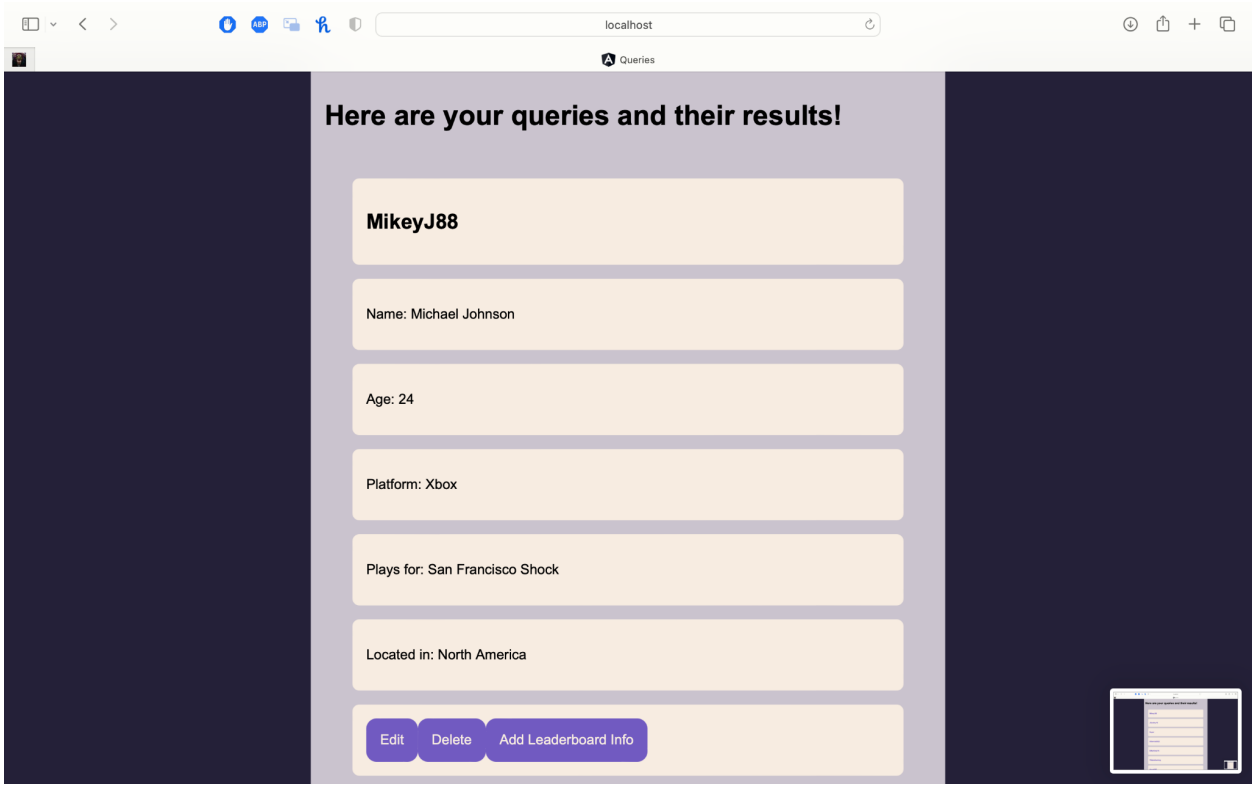

**Page 1:**

This is where the program starts and there are no queries being run at this part of the program. From here is where one can reach all parts of our program. There are 8 buttons you can click on to navigate to any part of our program. It serves as a starting point and provides access to all other pages.



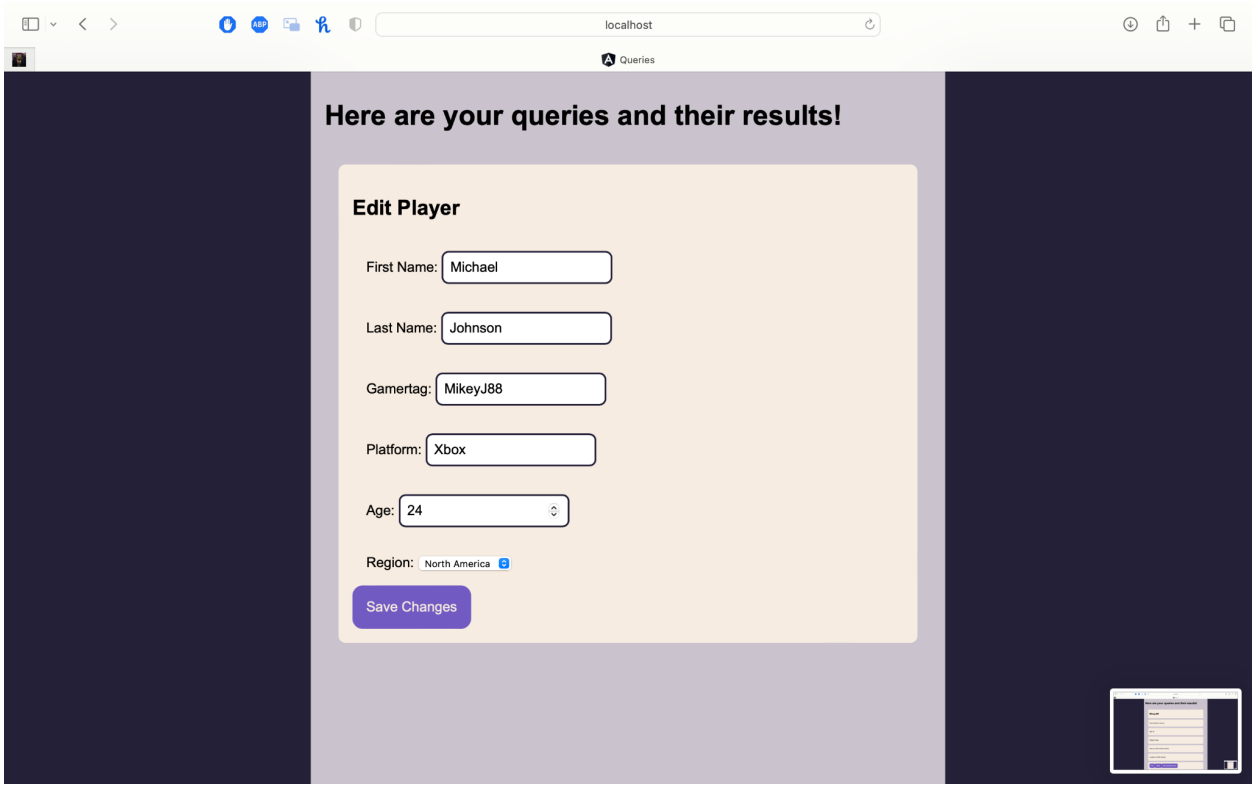

**Page 2:**

The List Players page shows the result of a query that displays all gamertags in the players table. A GET request is made to retrieve all player gamertags from the players table, and the resulting data is displayed on the page. When a player gamertag is clicked, a GET request is made to retrieve the player's data from the players table using the playerID. By clicking on a player gamertag it redirects you to Page 3 and passes the playerID. The player data is queried from the players table in the database using a SELECT statement. Each player is displayed as a button that when clicked redirects to the player data page to edit that player's information (page 3).



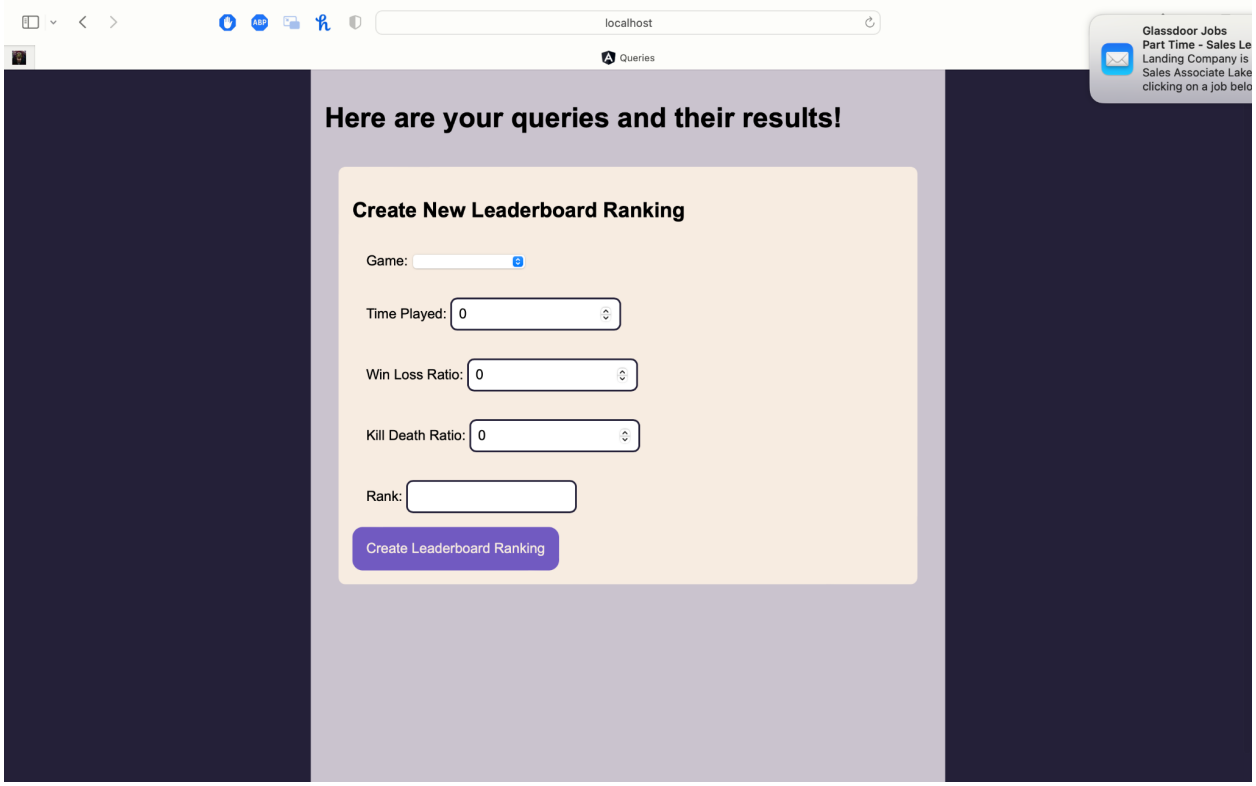

**Page 3:**

Each Player data page is populated with the player data when querying the players table for the passed playerID. Users can then click on Edit player to redirect to Page 4 or Add Leaderboard Data to redirect to Page 5.The data is queried from the players table in the database using a SELECT statement, which filters the results based on the playerID passed to it. A GET request is made to retrieve the player's data from the players table using the playerID. Here you can see the players name, age, platform, team, and regional location.



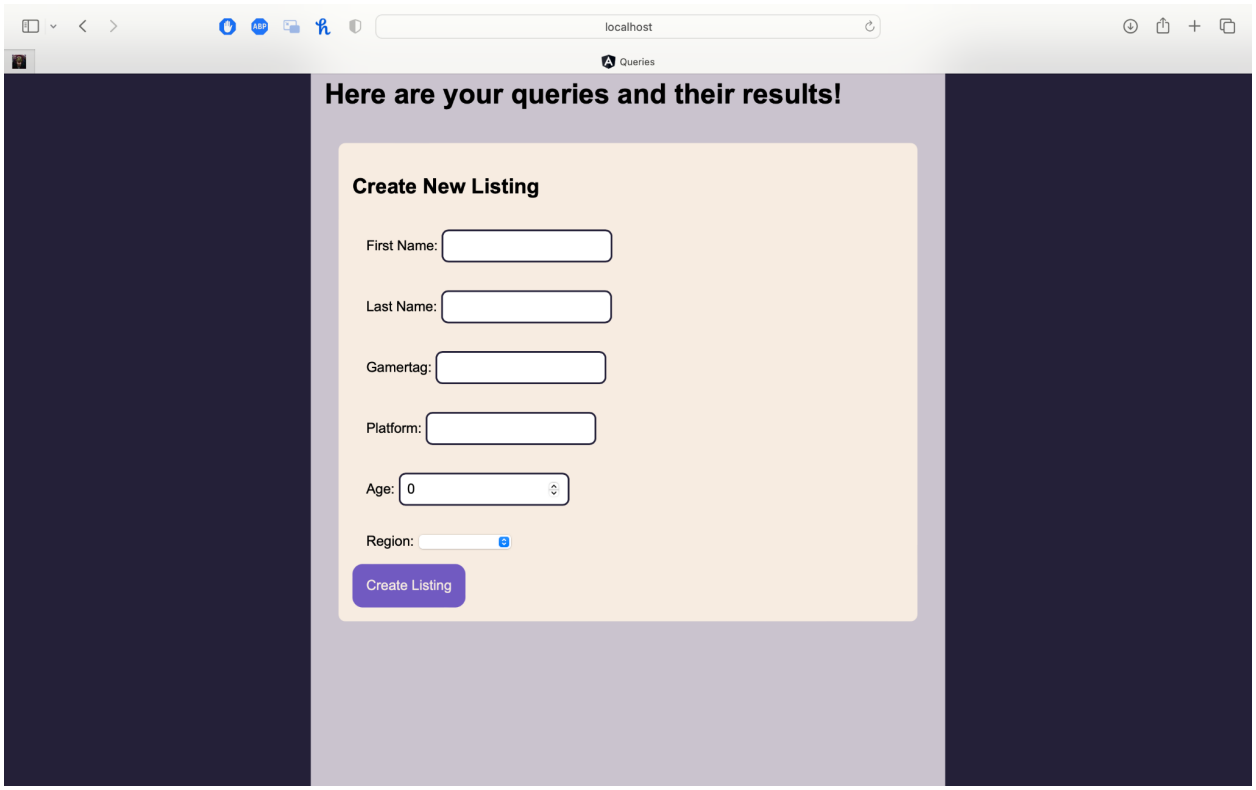

**Page 4:**

In this page we allow users to edit a player's information. For Page 4, a POST request is made to the backend server with the updated player information, and an SQL UPDATE command is used to modify the corresponding player data in the players table. The data is updated in the players table in the database using an UPDATE statement, which is triggered when the user clicks the "Save Changes" button. The data is also updated in the leaderboard table as well so that all instances of that player match.



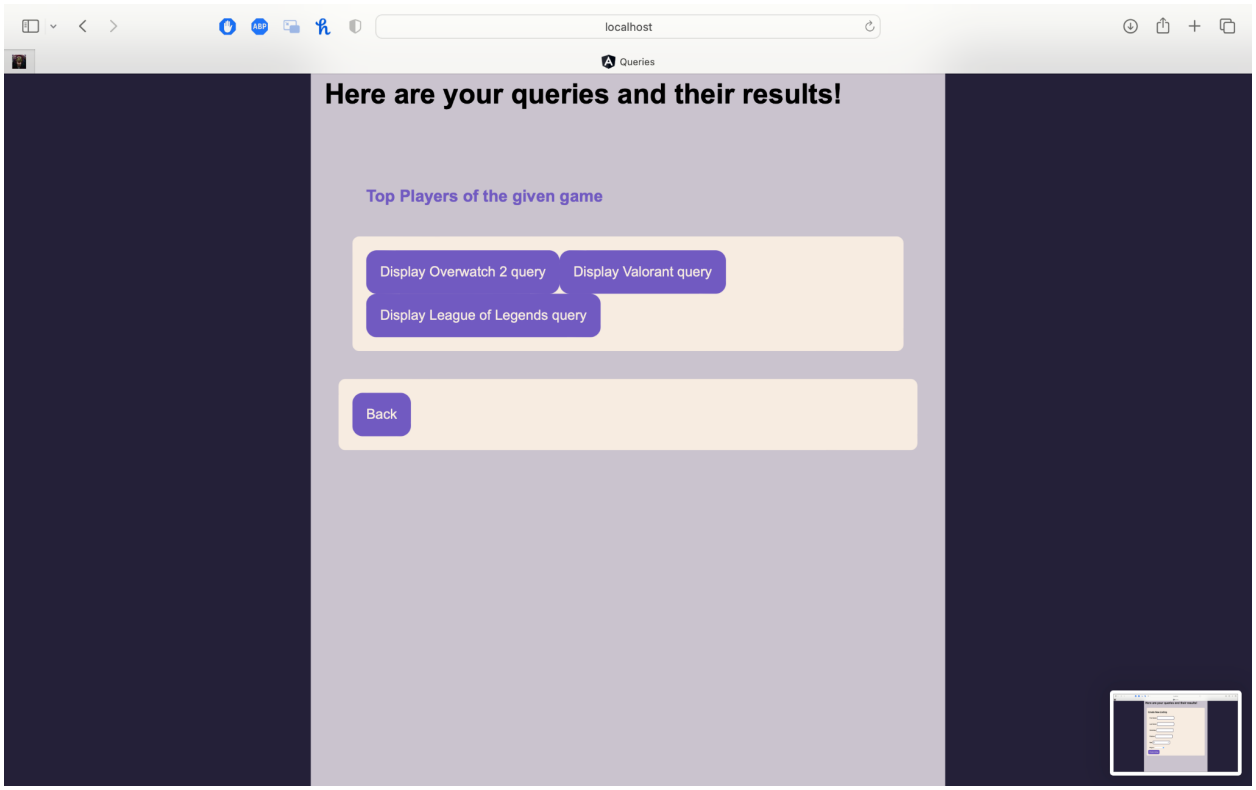

**Page 5:**

Add leaderboard page takes in any relevant information and adds a new leaderboard listing based on the given gameID and the player's playerID to add the relevant information to the leaderboard table. This addition uses a combination of gameID and playerID to prevent any duplicate table entries as games like these do not allow a single player to create multiple saves from the same account.The page makes use of a form that submits a POST request to the server with the provided data. Upon receiving the request, the server will perform a SQL INSERT operation to add the new entry to the Leaderboard table, using the gameID and playerID provided by the user to create a unique entry. The data is inserted into the leaderboard table in the database using an INSERT statement. The insert statement takes in the gameID and playerID for the leaderboard listing, as well as any additional relevant information such as kill death ratio and win loss ratio and rank and time played.



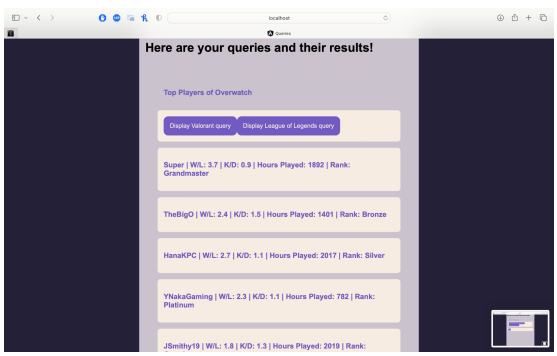

**Page 6:**

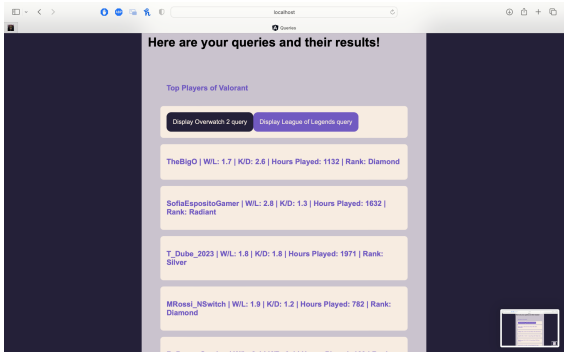
In this page we run an sql query command to call insert to add new players to the database.This page allows users to add new players to the database. When a user navigates to Page 6, they are presented with a form where they can enter the details of a new player. The data is inserted into the players table in the database using an INSERT statement. When the user submits the form, the frontend sends an HTTP POST request to the backend server, which is handled by a specific route defined in the backend code. This route is responsible for inserting the new player data into the database. The backend server uses a SQL INSERT statement to add the new player's information to the players table in the database. The server then sends a response back to the frontend to indicate whether the operation was successful or not.

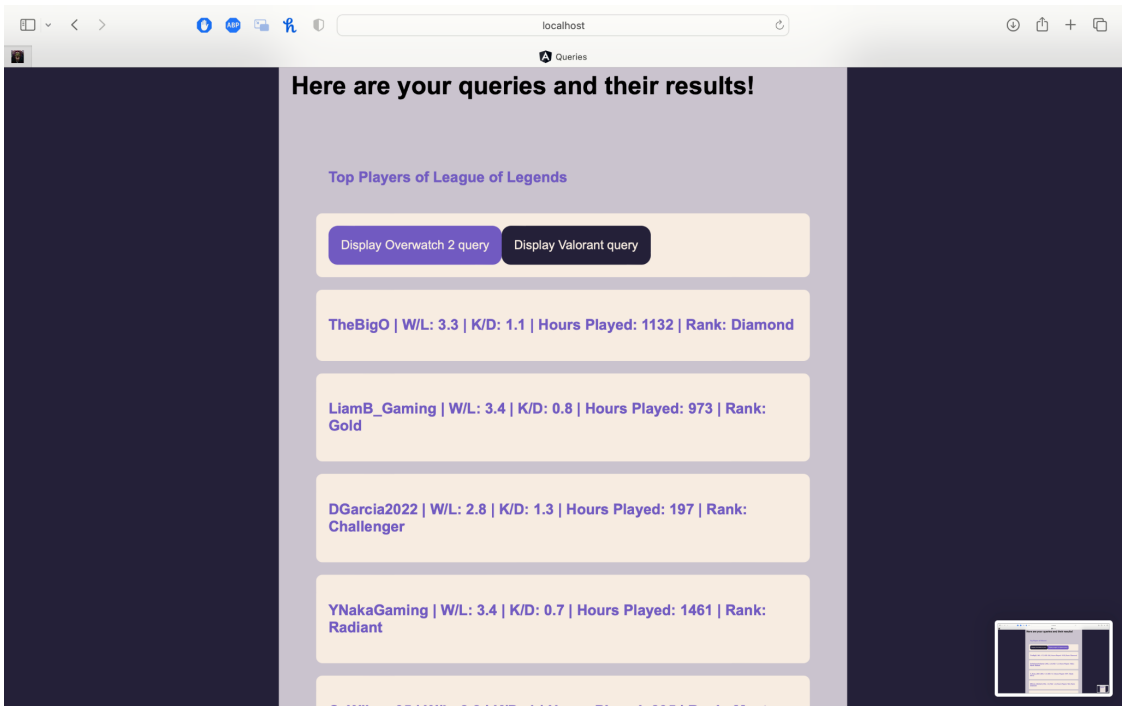



**Page 7:**

Show Game Leaderboards initializes 3 buttons that repopulate the page to redirect to page 8, and passes the gameID on button click. A GET request is made to retrieve all game names and IDs from the games table, and the resulting data is displayed on the page. When a game button is clicked, a GET request is made to retrieve the top leaderboard listings for the corresponding game ID, and the user is redirected to Page 8.

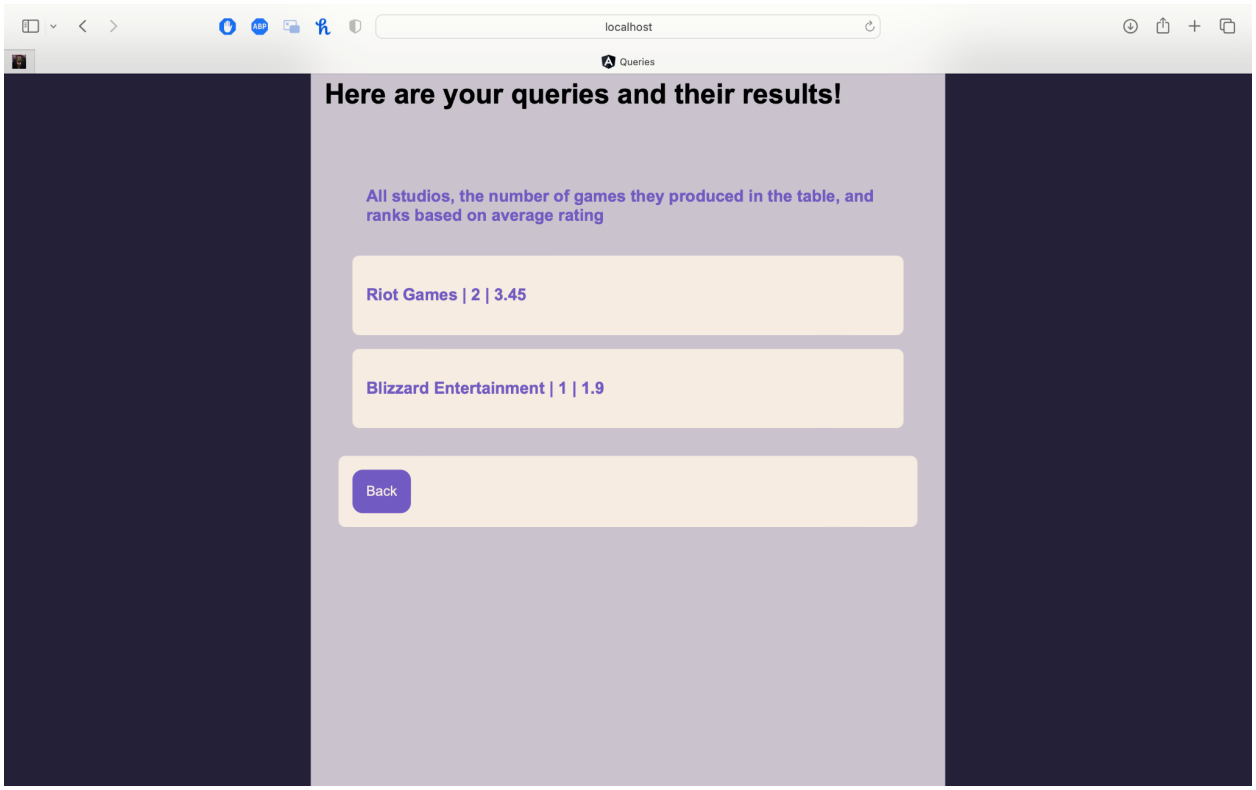


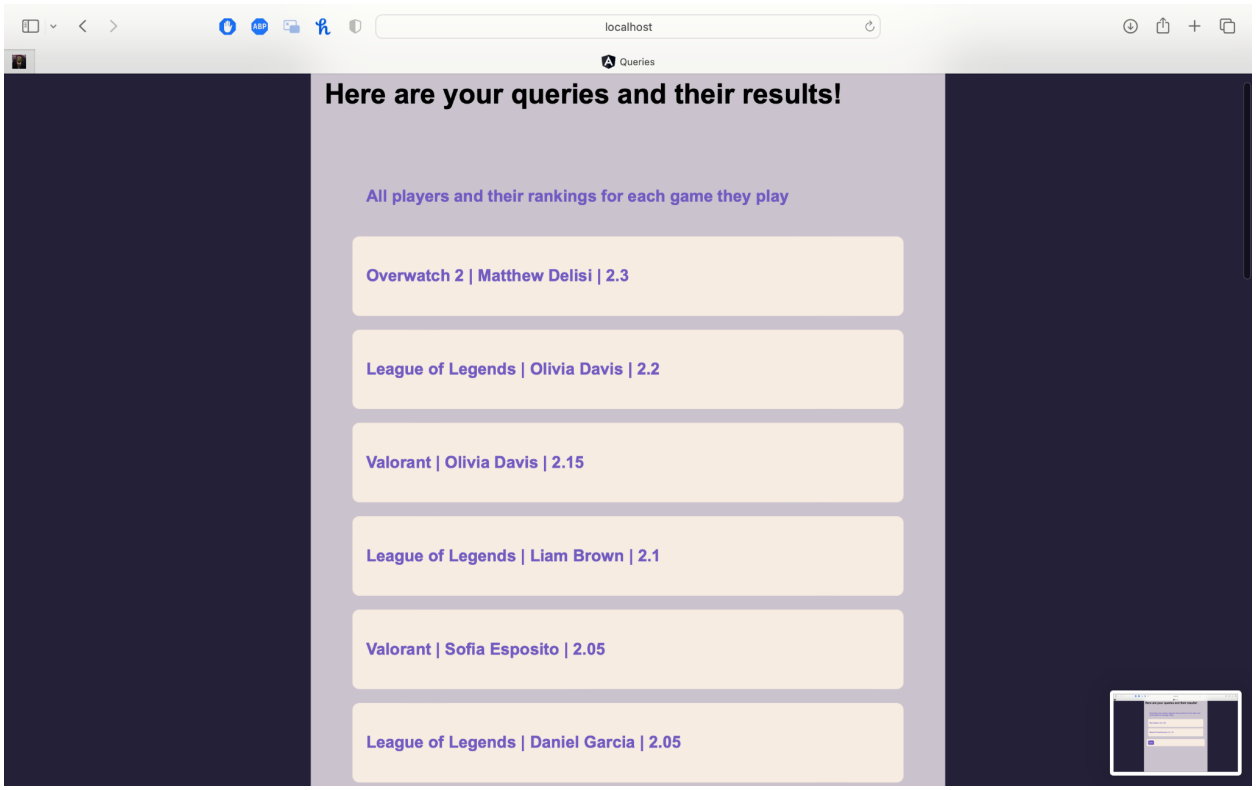

**Page 8:**

From the show games leaderboard page if you click on any of the games it will run a query and return the top players from our query for that specific game. A GET request is made to retrieve the top leaderboard listings for the corresponding game ID. From here you can click on a player and it will direct you to the edit player page if you need to change that player's attributes.



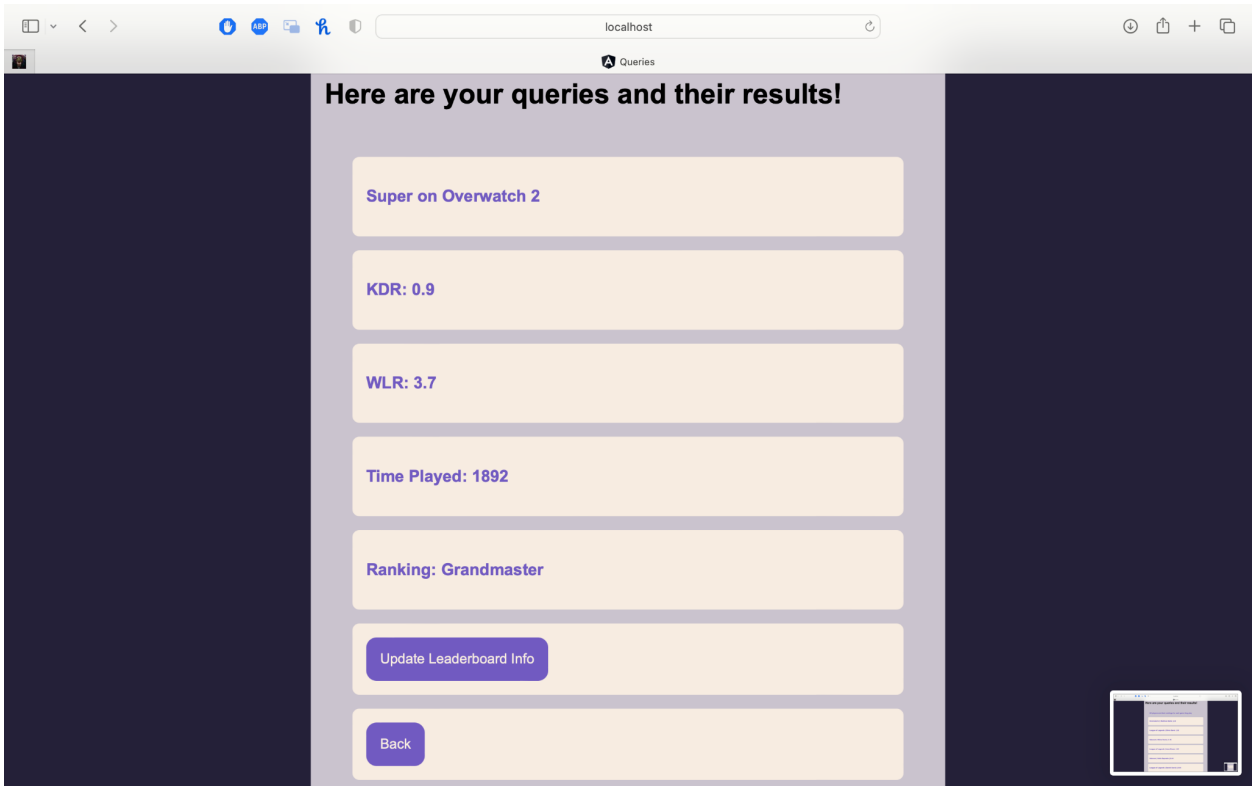

**Page 9:**

Display Studio Information page displays a query that returns the studio information for all studios in the studio table. A GET request is made to retrieve all studio information from the studio table, and the resulting data is displayed on the page.



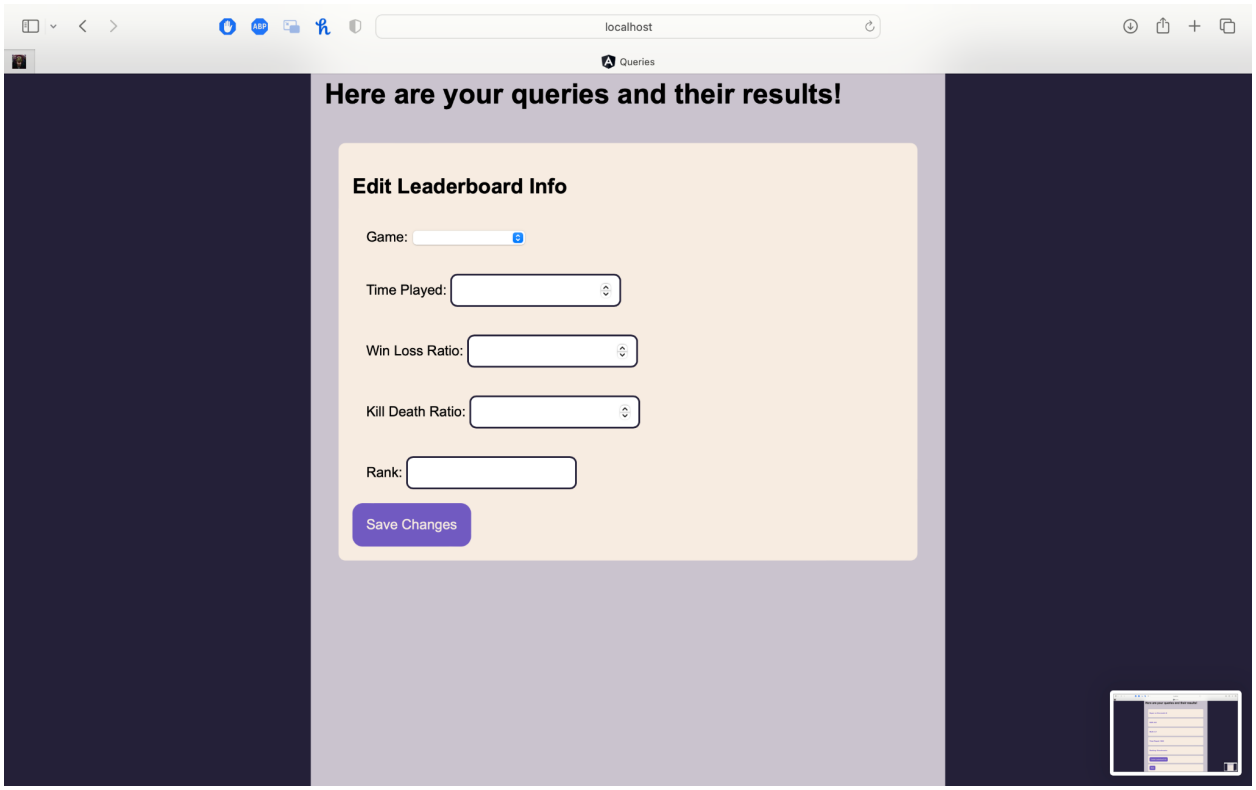

**Page 10:**

Show Consolidated Leaderboard page shows a similar query to page 8 but instead of filtering by game, it shows all players for all games, and upon clicking on a leaderboard listing the user is redirected to that leaderboard listing's Page 11. A GET request is made to retrieve the top leaderboard listings for all games from the leaderboard table, and the resulting data is displayed on the page. When a leaderboard listing is clicked, a GET request is made to retrieve all of the player data from the leaderboard table using the game ID and player ID, and the user is redirected to Page 11. The list is sorted by the formula Top player = (win/loss + kill/death)/2



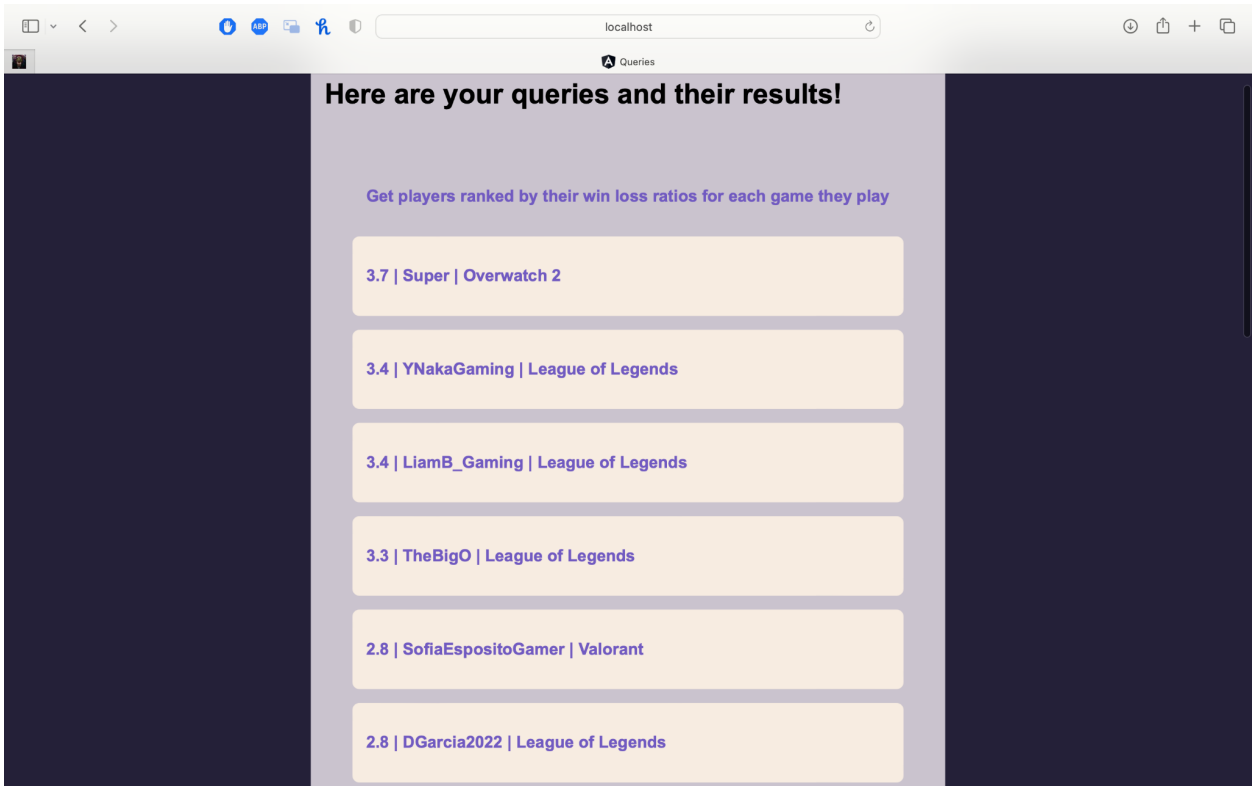

**Page 11:**

Leaderboard data page displays similar data to Page 3, however it queries the data from leaderboard with the given gameID and playerID rather than from the players table. From here a user can reroute to edit a leaderboard listing for that given listing. For Page 11, a GET request is made to retrieve the corresponding player data from the leaderboard table using the game ID and player ID passed from Page 10. The resulting data is displayed on the page, and users can click on the Edit Leaderboard button to redirect to Page 12.



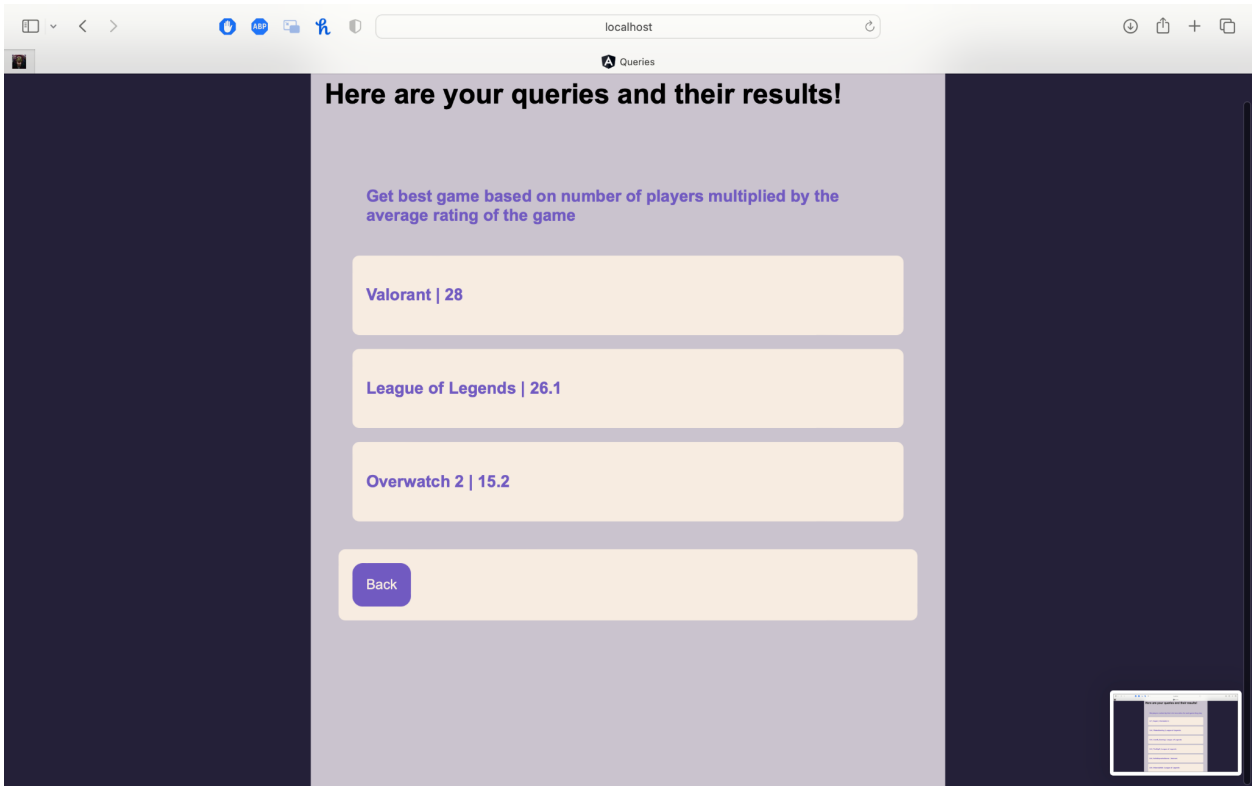

**Page 12:**

Edit Leaderboard. In this page you can update a player's info in the leaderboard which sends an update command to the backend to make sure the leaderboard is correct. For Page 12, a POST request is made to the backend server with the updated leaderboard information, and an SQL UPDATE command is used to modify the corresponding leaderboard data in the leaderboard table.




**Page 13:**

In this page the top players of all games are displayed based on wins and losses. A Query is sent to the backend to select and order the players based on win loss record. A GET request is made to retrieve the top players of all games based on win-loss record from the players table, and the resulting data is displayed on the page. Here game does not matter this is just a general leaderboard showing top players based on win loss ratio.




**Page 14:**

This page displays the top games based on the formula top game = game rating * number of users. We get the number of players for each game by connecting the games table and players table through the leaderboards table. Upon loading the page, a GET request is sent to the server, which queries the leaderboard, games, and players tables to calculate the user rating and number of players for each game. The results are then used to rank the games and display them on the page.

**SQL Queries:**

In this section we will go in depth in each query and provide a high level description and detail of how each query works.

1. Add player: This query adds a new player to the player table with the specified attributes. It uses the INSERT INTO statement to insert a new row into the table with the given values.

   ```
   INSERT INTO players (playerID, regionID, teamID, platform, firstName, lastName, age, gamertag)
   VALUES (playerID, regionID, teamID , platform, firstName, lastName, age, gamertag);
   ```

2. Delete player: This query removes a player from the player table with the specified player ID. It uses the DELETE FROM statement to remove the row with the given player ID from the table.

   ```
   DELETE FROM players WHERE playerID = player_id_to_delete;
   ```

3. Select(show leaderboard for a specific game): This query retrieves all of the leaderboard data for a specific game by joining the leaderboard and player tables on player ID and filtering by game ID. It returns a table with columns for player name, win/loss ratio, kill/death ratio, time played, rank, and player rank.

   ```
   SELECT players.firstName, players.lastName, leaderboard.winLossRatio,
   leaderboard.killDeathRatio, leaderboard.timePlayed, leaderboard.playerRank,
   leaderboard.playerRank
   FROM leaderboard
   JOIN players ON leaderboard.playerID = players.playerID
   WHERE leaderboard.gameID = players.gameID;
   ```

4. Merge(games by studio): This query groups games by studio and returns the studio name and the number of games developed by that studio. It uses the GROUP BY statement to group the games by studio and the COUNT function to count the number of games developed by each studio.

   ```
   SELECT studio.studioName, COUNT(DISTINCT games.gameID) AS num_games,
   AVG(game.rating) AS avg_rating
   FROM games
   JOIN studio ON games.studioID = studio.studioID
   GROUP BY studio.studioName
   ORDER BY num_games DESC;
   ```

5. Update: This query updates the values of a specific row in a table with new values. It uses the UPDATE statement to set new values for the specified columns in the row with the given ID.

   ```
   UPDATE players
   ```

```
SET firstName=?, lastName=?, gamertag=?, platform=?, age=?, regionID=?
WHERE playerID=?;
```

6. Show top player in a specific game where Top player = (win/loss + kill/death)/2: This query retrieves the top player for a specific game by joining the leaderboard and player tables on player ID and filtering by game ID. It calculates the player's score using the specified formula and returns the player's name and score.

```
SELECT players.firstName, players.lastName, leaderboard.winLossRatio,
leaderboard.killDeathRatio, leaderboard.timePlayed, leaderboard.playerRank,
    ((leaderboard.killDeathRatio + leaderboard.winLossRatio) / 2) AS best_player_score
FROM leaderboard
JOIN players ON leaderboard.playerID = players.playerID
WHERE leaderboard.gameID = ?
ORDER BY best_player_score DESC;
```

7. Show all players with a positive win/loss record: This query retrieves all players from the player table with a positive win/loss ratio. It uses the WHERE clause to filter the table for players with a win/loss ratio greater than 1.

```
SELECT players.firstName, players.lastName, leaderboard.winLossRatio
FROM leaderboard
JOIN players ON leaderboard.playerID = players.playerID
WHERE leaderboard.winLossRatio > 0;
```

8. Best game based on player scores: This query calculates the score for each game by multiplying the number of unique players by the game's rating. It then returns the top 5 games with the highest scores by sorting the results in descending order by score and limiting the result set to 5 rows.

```
SELECT games.gameName, games.rating, COUNT(DISTINCT leaderboard.playerID) AS
num_players,
    games.rating * (leaderboard.winLossRatio + leaderboard.killDeathRatio) / 2 AS score
FROM leaderboard
JOIN games ON leaderboard.gameID = games.gameID
GROUP BY games.gameName, games.rating
ORDER BY score DESC;
```