Link to project: https://github.com/koitu/cs348-project

Black text = Milestone 1
Blue text = Additions done for Milestone 2

Description

The proposed application for this project is a comprehensive National Hockey League (NHL) database for game and player statistics from the 2008-2009 season until the current season. The application itself will be a web app focused on getting simple to advanced statistics on players and teams, allowing for an easy user experience when retrieving the data. The aim is to provide a simple way for hockey enthusiasts to look up whatever data they need. The database management system will be designed such that there are administrators who can modify any type of data, team accounts that can modify the data of their team, and users that can simply query the database.

The functionalities that will be supported include searching for individual players and their statistics. This includes stats accumulated in an arbitrary time period, specific games where certain stats were met, and so on. Similar queries will be supported for team searches just with team-specific stats, such as wins between time periods, wins against certain opponents, and so on. An account system with certain saved queries and personalized information will also be supported for the users of the application. This allows certain important and potentially expensive queries to be cached to improve the usability of the app.

The functionality remains largely the same. An account system was implemented as described. New features are admins and user classes of accounts. The admins will have the privilege to update the database, while the users have the ability to save favorite player and favorite teams.

System Support

Currently, the application is a web-based app that runs on Windows, MacOS, and Linux currently using local hosts for the frontend and backend sides. This allows for simple launching, testing, and demonstrating of the web app. It would be possible to use a remote server and have the app operate as a more typical website, however, this is an additional step and could realistically only be implemented in later steps, if at all. The programming frameworks chosen are Flask (Python-based web framework) for the backend and React (Javascript-based runtime environment) for the frontend.

The database used in this project will be locally hosted. Again this is for ease of use, testing, and demonstration. It would be possible to host the database on a remote server, and this could be implemented in later milestones. The database will use MySQL as the DBMS to query the database, allow different users to log in, and other database features. Python has MySQL packages which makes the implementation of queries simple on the backend.

The system support has remained the same.

Data Gathering

The current plan for getting both toy and sample data is to use and reformat the data held in this NHL, database https://moneypuck.com/data.htm. This database holds detailed statistics from games played

from 2009 to the present day. The database also has detailed player information. This database has some redundant information, and its current schema must be modified to match our current schema. The database also uses .csv files which will need to be converted to an appropriate SQL file. However these are minor changes, and the sheer volume of data is exactly what is needed for this project, and this will be the primary source of data.

We will use 3 or 4 players and 3-4 sample teams from the same season for the sample data. This will provide enough information to run the sample queries.

The source used also updates its data regularly, so there must be methods in place to update the database to obtain new data. A solution to this problem is to have the DBMS fetch and update the tables from the source. This could be done manually or scripted to automatically update if the data is too old. This approach could help with the final database, however, it will not be implemented for the sample data as there is plenty of data to use already.

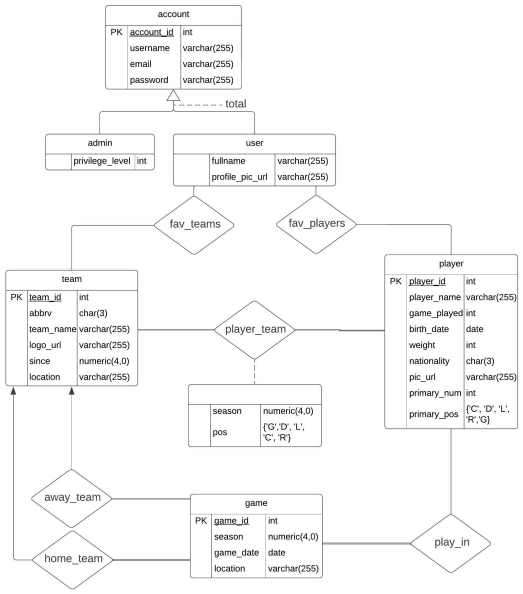
The production data was taken https://moneypuck.com/data.htm, and ran through a script to convert the .csv files to .sql files usable in the database. Another database was used to get the player profile pictures was https://www.nhl.com/. The picture links were merged into the player table from the first database. Accounts for production data were both added by hand and also generated with a script. Data for relations that use account ids (like Fav_team) were generated with a script as well. Additionally, using the app to sign up will add user accounts. In the future it will be possible to add admin accounts this way as well.

Database Design

Assumptions made about the data being modeled:

1. Players ids are all unique
2. Team ids are all unique
3. Game ids are all unique
4. Accounts will either be users or admins, not both or neither.
5. A player in a game is playing on either the home or away team, but not both or neither.
6. A players position must be be on of the following, 'G', 'D', 'L', 'C', 'R'
7. A player added to the plays_in table must be a member of one of the two teams playing in that game

The ER diagram was changed to add team and game locations



Admin(account_id, username, email, password, privilege_level)

User(account_id, username, email, password, fullname, profile_pic_url)

Fav_Teams(uid, tid)
. **Foreign Key:** uid references account_id in User
. **Foreign Key:** tid references team_id in Team

Fav_players(uid, pid)
. **Foreign Key:** uid references account_id in User
. **Foreign Key:** pid references player_id in Player

Player(player_id, player_name, games_played, birth_date, weight, nationality, pic_url, primary_num, primary_pos)

Team(team_id, abbrv, team_name, logo_url, since, location)

Player_team(pid, tid, season, pos)
. **Foreign Key:** pid references player_id in Player
. **Foreign Key:** tid references team_id in Team

Play_in(pid, gid)
. **Foreign Key:** pid references player_id in Player
. **Foreign Key:** gid references game_id in Game

Game(game_id, season, game_date, homeid, awayid, location)
. **Foreign Key:** homeid references team_id in Team
. **Foreign Key:** awayid references team_id in Team

Feature Descriptions

1. Searching for players is one feature of the application. In the search bar you can type in a string and any players whose names have a matching substring will appear in the results. The results will be links to the player's page which will then have detailed stats about them. This allows the application to work almost as a search engine. In the future, an algorithm on the backend can detect typos and attempt to return the closest matches. A simple example is case-insensitive searching. Extra clauses can be added to the where section of the query for filtering.

   The template query goes as follows. Note that currently, the sample query returns all the attributes associated with the player. This can be modified to only return the necessary attributes, or fancy features can make use of these attributes. Also, note that seasons will be an optional join that will only join on the specified seasons in the future.

   The feature remains the same, although the filters are yet to be implemented. The query format has also changed. The player_name attribute has been indexed to allow for faster search queries with large production data.

   ```
   SELECT DISTINCT player_id, player_name, pic_url
   FROM (SELECT *
         FROM Player
         WHERE player_name LIKE '%<substring>%') AS pab
   JOIN PG USING (player_id)
   JOIN Game USING (game_id)
   ORDER BY game_date DESC
   LIMIT 10;
   ```

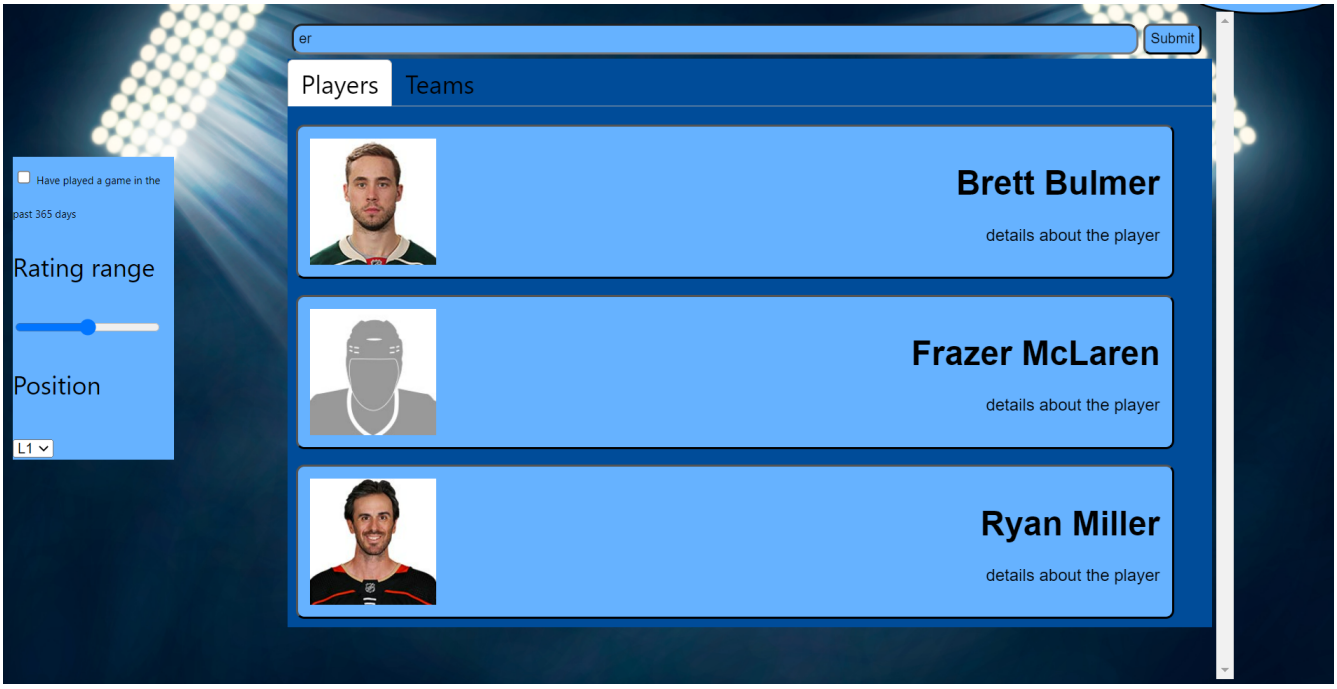
   As a sample query:

   ```
   SELECT DISTINCT player_id, player_name, pic_url
   FROM (SELECT *
         FROM Player
         WHERE player_name LIKE '%ab%') AS pab
   JOIN PG USING (player_id)
   JOIN Game USING (game_id)
   ORDER BY game_date DESC
   LIMIT 10;
   ```

   A useful feature of this is that with an empty substring, all players will be returned. This allows for

finding players based on these optional filters.

A screenshot of the query



2. Searching for teams is another feature of the application similar to searching for players. The search result will return teams instead. Filters can be used to narrow down teams, however, the filters will now be different. Teams can be viewed as one team, or if a year filter is selected the teams from just those seasons will appear. This feature also remains the same and still needs the filter feature to be added. The query has slightly changed to limit the teams to just the top 10. The team_name attribute has been indexed to allow for faster search queries with large production data.

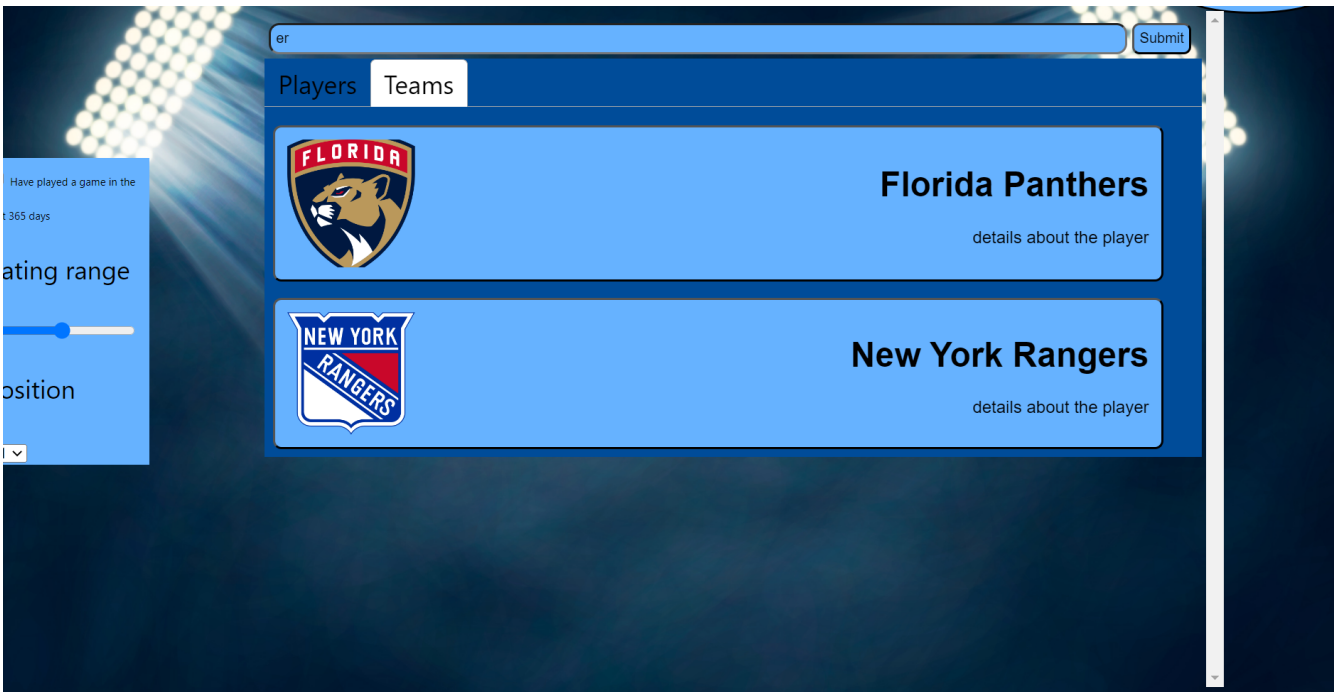
   The template query goes as follows, again the same nuances apply to the returned attributes.

```
SELECT team_id, abbrv, team_name, logo_url
FROM Team
WHERE team_name LIKE '%<substring>%'
ORDER BY since
LIMIT 10;
```

   As a sample query:

```
SELECT team_id, abbrv, team_name, logo_url
FROM Team
WHERE team_name LIKE '%m%'
ORDER BY since
LIMIT 10;
```

   A screenshot of the query



3. On a player page, there will be a section for recent games played. These games will be retrieved from the database. Initially only the 5 most recent games will be shown, subject to the constraints given by the filter from the search. This can be changed later on to add an option to show more games, however, 5 is enough for now. This feature's query was changed

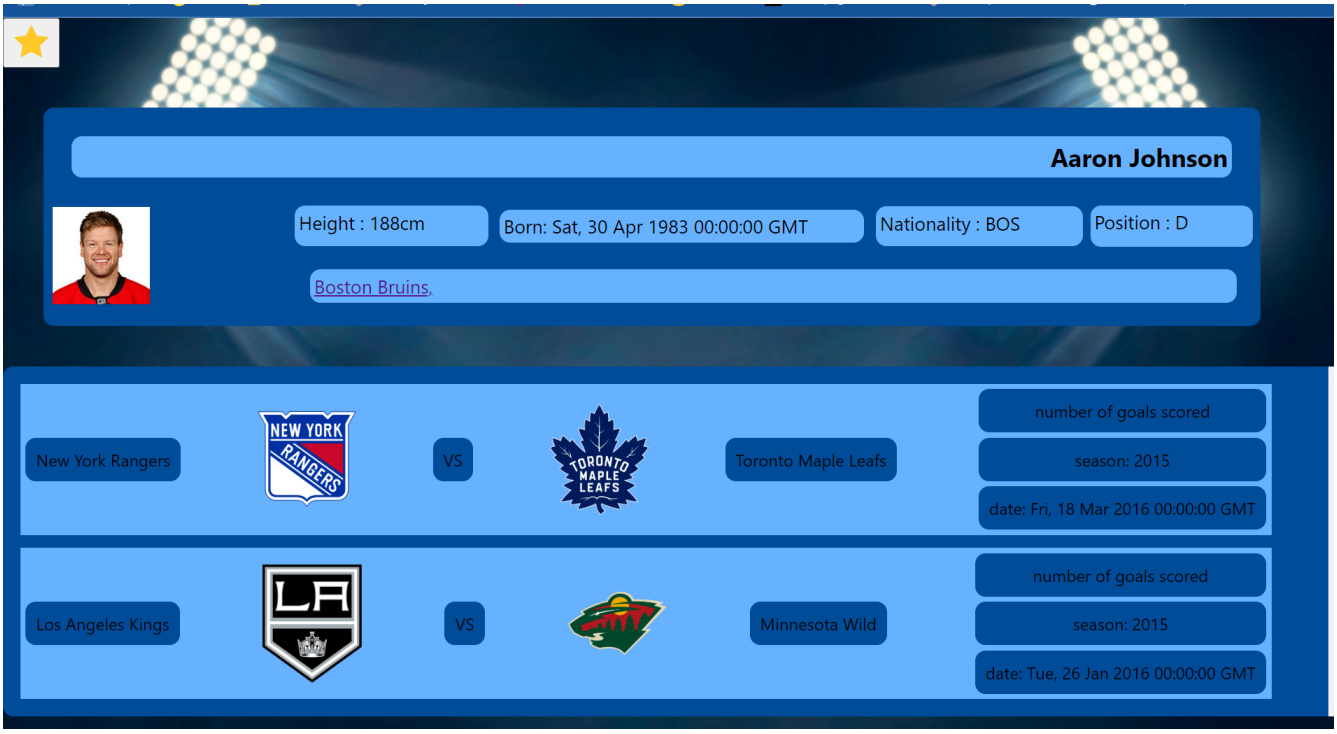
   The template query goes as follows:

```
select *
from (select game_id, points
      from PG
      where player_id = <player>) as gop
join Game using (game_id)
order by game_date desc
limit 5
```

   As a sample query:

```
select *
from (select game_id, points
      from PG
      where player_id = <8469534>) as gop
join Game using (game_id)
order by game_date desc
limit 5
```

   A screenshot of the query



4. A similar feature to the one above is that on a team page, the 5 most recent games of that team will be shown. In the future, an option can be added to show a user selected amount of games, but 5 is enough for now. A note is that game_date is indexed to allow for faster queries on production data.

   The template query

```
select *
from Game
where team_home_id = <team> or team_away_id = <team>
order by game_date desc
limit 5;
```

   As a sample query:

```
select *
from Game
where team_home_id = 101 or team_away_id = 101
order by game_date desc
limit 5;
```

5. The next feature is for users to add favorites to their favorites player page. This will allow users to quickly access the players they want most. This feature can also be quickly changed to allow for favorite teams pages to be saved. This feature also works for favorite teams by changing the table. The template queries go as follows:

```
insert into fav_players
values (<current_user_id>, <player_id>)
```

   As a sample query:

```
insert into fav_players
```

```
values (200002, 8469534)
```

6. As well as adding favorite players, a user can then view their favorite players on their homepage. This feature will be extended for favorite teams as well, but the query remains largely similar. The player_name atrribute has been indexed to allow for faster search queries with large production data.

   The template query

```
select player_id, player_name, pic_url
from (select player_id
      from Fav_Players
      where account_id = <current_account>) as fp
join Player using (player_id)
```

   As a sample query:

```
select player_id, player_name, pic_url
from (select player_id
      from Fav_Players
      where account_id = <200001>) as fp
join Player using (player_id)
```

7. A signup feature for new accounts was created. This allows users to make an account to access the app. This feature also ensures that no duplicate users will signup (either via username or email).

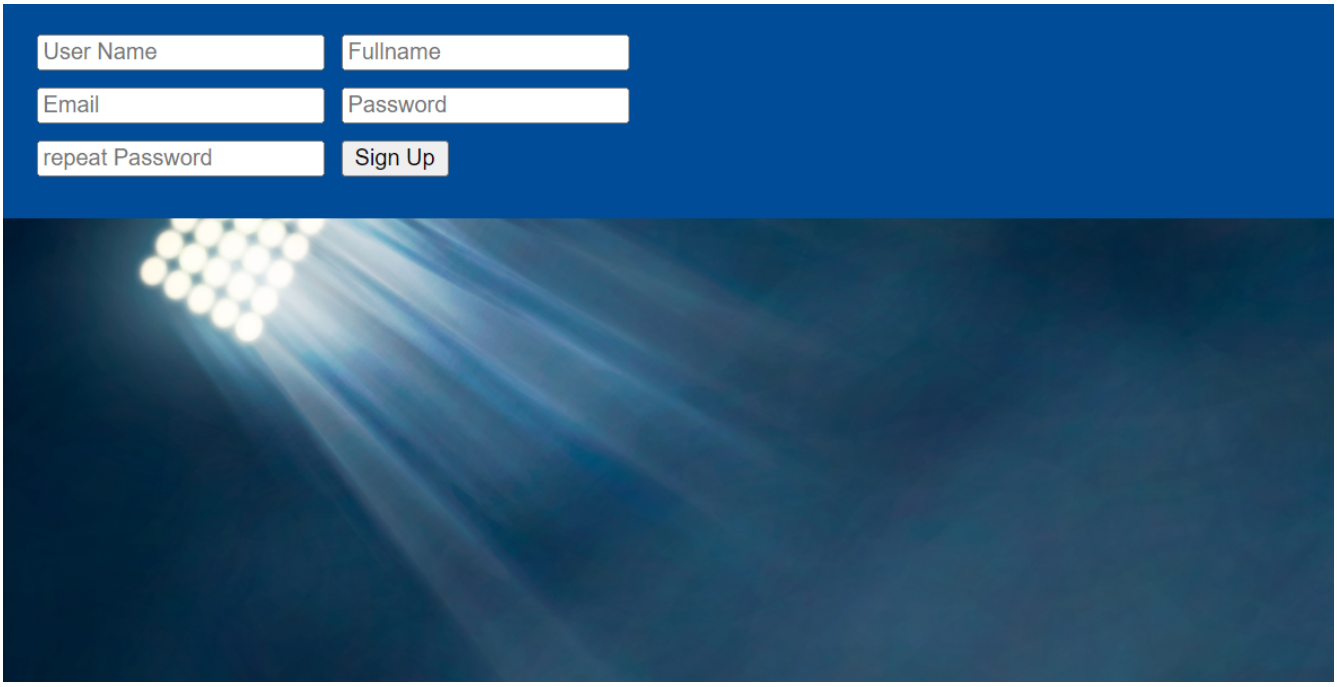
   The template query

```
start transaction;
insert into Account(username, email, acc_pass)
values(<username>, <email>, <password>);

insert into User(account_id, full_name, pic_url)
values(LAST_INSERT_ID(), <name>, <pic_url>)
commit;
```

   As a sample query:

```
START TRANSACTION;
INSERT INTO Account(username, email, acc_pass)
VALUES ('shayanmk', 'sample@test.con', 'Hello123');

INSERT INTO User(account_id, full_name, pic_url)
VALUES(LAST_INSERT_ID(),'Shayan Mohammadi','i.redd.it/5pf0gzyzmxf41.jpg');
COMMIT;
```

   A screenshot of the sign up page



8. A sign-in feature allows for users to log into their accounts to access their profile-specific data. Note that the username attribute is auto-indexed, as it has been declared to be unique when its table is created. No index must be explicitly created to speed up searches.

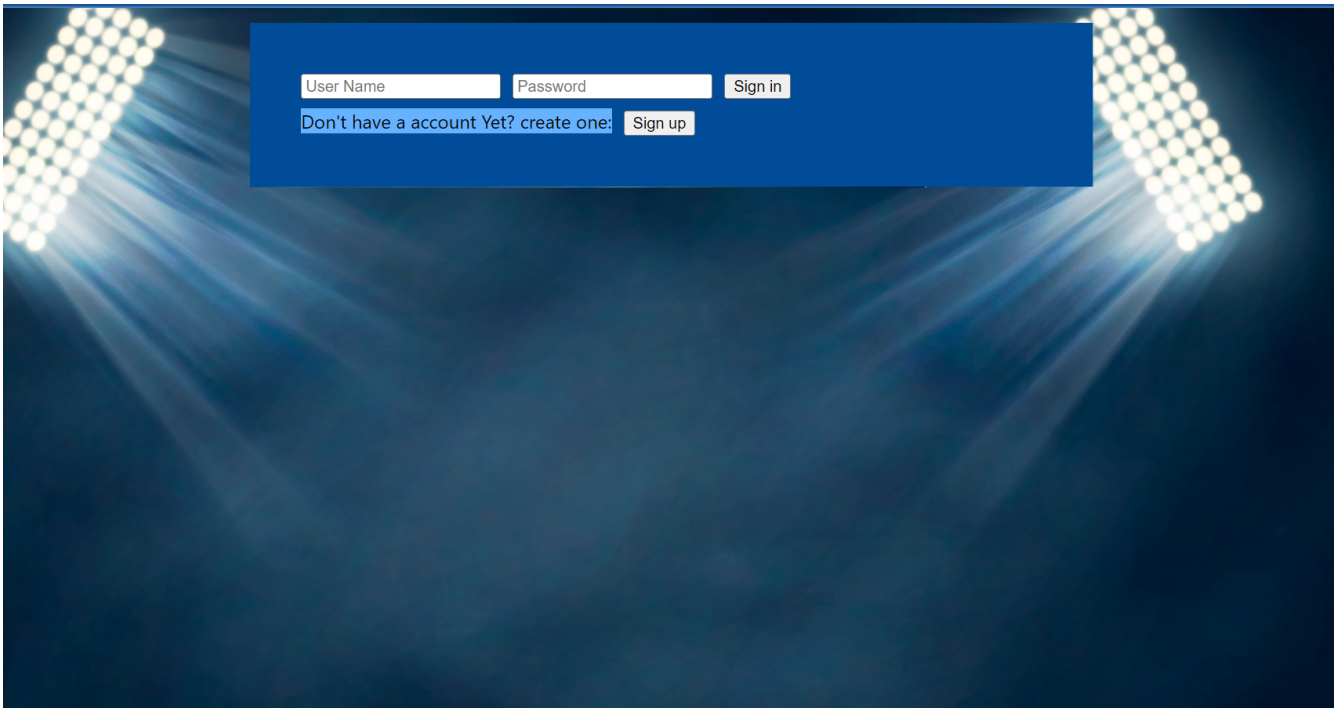
   The template query

   ```
   select account_id
   from Account
   where username = <login name> and acc_pass = <login password>
   ```

   As a sample query:

   ```
   select account_id
   from Account
   where username = 'shayanmk' and acc_pass = 'Hello123'
   ```

   A screenshot of the sign in page



9. A player's personal page can be viewed. From here stats of the player as well as the player's height, weight, birthday, etc. Can be viewed

   As a template query:

   ```
   select *
   from Player
   where player_id = <player>
   ```

   As a sample:

   ```
   select *
   from Player
   where player_id = 8469534
   ```

   A screenshot of the player's personal page:

Members

Milestone 1:

- Andrew Wang (20882272). Responsible for the front-end and back-end implementations and implementing queries.
- Shayan Mohammadi Kubijari (20967990). Responsible for schema design, query implementations, and converting the database to SQL, along with modifying the database to fit the schema.
- Arvin Asgharian Rezaee (20972431). Responsible for the front-end and back-end implementations, setting up the scripts to launch the project and query implementation.
- Victor Astinov (20851407). Responsible for the template queries, documentation, and sourcing of the database.
- Shang Jin (21051963). Responsible for the schema design, ER diagrams, and relational data model.

All team members did their part as expected. The progress made for all team members is the expected progress for milestone 1.

Milestone 2: Along with the above remaining the same, additional tasks for milestone 2 were:

- Andrew Wang (20882272): Feature implementation on front and back end, generating some production data and scripts
- Shayan Mohammadi Kubijari (20967990). Transferring data from the source database to the production database, adding new queries and creating samples for them. Making the SQL statements that create the database
- Arvin Asgharian Rezaee (20972431). Added new features on the frontend and backend
- Victor Astinov (20851407). Created some scripts to generate data for relationship tables, report updating, and sample queries created
- Shang Jin (21051963). Helped transfer data from the source database to the production database. Making the SQL statements that create the database. Finding a database for player images.

All team members did their parts and help other teammates in completing milestone 2. The progress made by all team members is what was expected for milestone 2.