HOCKEY STATS

A Python and SQLite Web Application

INFO 606

Winter Term 2021

Kevin Fitzpatrick

Matthew Merenich

Kim-Cuong Nguyen

Kurt Shadle

Contents

Summary

Our web application is a simple and straightforward way to users to look up statistics about hockey teams. It utilizes a clean user interface built with HTML, CSS, and Javascript that connects to a SQLite database. The data is updated daily though an API so there is no need for manual insertion of data once the connections were set up. The API connections as well as the connections to the front end are all built using the Python framework, Flask. The data that we are pulling in goes back from 2018 up to the current season. As new seasons happen, our database will automatically update which in turn updates the user interface. This means that even if no more coding happens to this application, it can run and stay updated as long as the API remains stable.

Introduction

Sports statistics have always received significant attention from sports lovers. There has been an undeniably growing demand for quick access to sports information including news, events, season statistics like time and scores, teams, and players, etc. However, those above-mentioned types of information may not be easily accessed at just one place; instead, the information is usually scattered at different news portals. This inconvenience motivated the team to create a one-stop information hub that provides all relevant information for sport stats. As a full solution involving all sports stats is more than can be done in a term, our team focused on starting with hockey team stats.

Goals

The goal of this application is to provide a simple interface for users to look up hockey team stats. The application will pull three seasons worth of data and will stay updated as more games are played.

Scope

In-Scope

The scope of this project will be limited to the following functionalities:

- Data: The following data sets are stored in the database and can be quickly accessed.
    - Game Results
    - Team Standings (within their divisions)
    - Team Stats
- Reporting:
    - All seasons starting from 2018 to current

- Summary of season
- Team performance throughout seasons
- Game results by chosen team

Out-Scope

The API we chose, API-HOCKEY by api-sports (api-sports, n.d.), to use did not have player data available, so all player specific stats are not part of this application. A secondary API, hockey-LIVE.sk data (plasoft, 2020), was reviewed to pull in player data but that idea had to be dropped due to time constraints. The application does not include such information as player transfers, events, news. All the data provided would be historical data; there will be no predictions.

Despite a lack of player data, the information made available to us was vast. Accessibility was key to utilizing this API: the code for submitting the request was provided with multiple methods and languages. Our application was running on Python, so to stay consistent we processed the data using this language as well.

Design

Database

Creating Csv Files

The overall strategy was to create the data in lists that act as a row within a table, and then combine a list of lists into a dataframe for easy organization and extraction to CSV files and simple upload into SQL. When we first accessed the data, it came through in the form of a massive string of JSON data. It took a bit of manipulation, but we eventually transformed the raw data into a data dictionary that was able to be accessed for the information we needed. The data we needed from the API was able to be accessed through two endpoints: Games and Standings. From this data, we needed to create 3 tables: *Teams*, *Games*, and *Game Results*.

The first table we created was “team info”. Using the Standings endpoint, we were able to utilize the 2019 final standings to find all teams for our historical data from 2018-2020. (the number of teams is static within this timeframe – 2017 did not have the Vegas Golden Knights, and 2021 will see a 32nd team be introduced to the league, the Seattle Kraken). From here, we created the following attributes: *teamID, team name, conference*, and *division*. *“teamID”, “team name”*, and *“conference”* were given to us by the API. The last datapoint was the teams’ *“division”*, which was a bit more difficult to pull. This year, the NHL created new divisions based on location in order to adhere to COVID-19 protocols by limiting the travel each team required between games. Therefore, we could not use 2020 divisional data and retrofit the current divisions. It was simpler to create a data dictionary based on 2019 divisions – this information was readily available and

easy to create. Once all attributes were accessed, the dataframe was created and downloaded to a csv file.

The next two tables were related to each team's games and their results. The first of these dataframes consisted of attributes to create a simple schedule: *GameID, Date, Home Team, Away Team*, and *the Season (year) of the game*. All these attributes were readily available in the original data from the API. Once each year's lists were created, we combined them all to create the dataframe.

The second dataframe was a bit more complicated. We needed to create two copies of each game – one for the home team and their results (goals, wins, losses, points) and the same for away. The teams' information was not difficult to extract, but the goals for each period and the final score needed a split to capture the integers we required. For example, each game's final score is listed as "2-1". We needed to get the "2" for the home team's score and "1" for the away team. The next step was implementing the logic to create the points of each game for each team. If a team won without needing OT, the winning team receives 2 points while the losing team receives 0. If the game went to overtime though, the winning team still gets 2 points, but the losing team receives 1. There were future games that had "null" values to account for, so this was the longest while loop in the code for all 3 years. All of this was to ensure the ability to create standings in our application. The list of list logic held, and a dataframe with all games and their results was created to be downloaded as a csv file. In the end, we created three CSV files after transforming the API JSON data.

Creating Database

After pulling required data from the relevant API, we proceeded to building the database. The hockey database has three tables: *teams, games*, and *gameresults*, structured following the three dataframes built in the previous step.

We created an SQLite database from Python program.

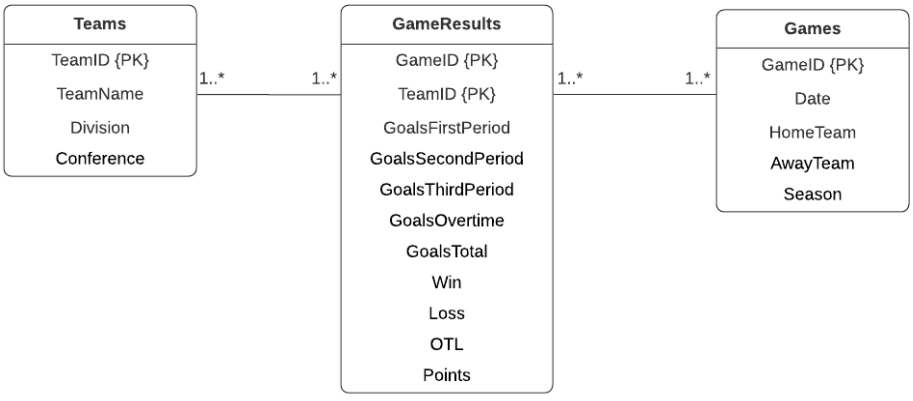

Figure 1. ERD of the hockey database

Table *GameResults* has foreign key *GameID* references *Games* table, and foreign key *TeamID* references *Teams* table.

User Interface

Framework

The user interface was designed to be clutter free and allow the user to get right to the point, the stats. To create a clean, modern looking interface, Bulma (Bulma, n.d.) was chosen as the CSS framework to work with. Several CSS frameworks were looked at and evaluated but Bulma was chosen due to the aesthetic and the light footprint of the framework. Another consideration in the CSS choice was Bulma was a framework not used before by the team and this application was all about trying new things.

On the Javascript side, jQuery (jQuery, n.d.) is used in combination with DataTables by CloudTables (CloudTables, n.d.) to create tables that display the data. By using DataTables, users are given options to sort the columns, search the displayed data, and given pagination controls. A few different table options were reviewed but DataTables was chosen for its rich number of features and helpful documentation. Aside from that, some custom Javascript was used to build out the custom functionality we required.

Visual Design

The overall design was kept minimalistic to not distract from the main purpose of the application, the data. The application was designed with a single HTML page in mind to minimize load times for the user. This led to using tabs for the navigation to control what set of data and controls were being displayed to the user.

The final version of the application has all user inputs as dropdown selections, fed from the database, for users to pick from. There initially were other search criteria options such as allowing custom date ranges, but they had to be cut due to time constraints. The application was meant to be responsive, display correctly for mobile devices, but that effort was only partially completed.

Methods

Database.py

The database.py Python file serves to transform data in csv files pulled from API into a database. In order to build an SQLite database with Python program, we utilize the necessary Python package - sqlite3 module. We first created a new SQLite database, then create relevant tables in SQLite database, and finally insert database into the SQLite database.

The first function, *create_connection(db_file),* is to create a new SQLite database from a Python program. This function creates connection to a database on local drive; in case the database does not exist, it will be automatically created by SQLite. *db_file* represents a path to the drive where the database is/should be located.

The second function*, create_table(conn, create_table_sql),* takes a connection object and an SQL statement as inputs. The connection object represents an SQL database specified by the database file parameter *db_file*, returned by calling the function *create_connection(db_file).* The SQL statement is a statement to create tables.

The third function, *create_tablenames(conn, values)* inserts each data row into the relevant table in the SQLite database. Inside the function, we define a SQL statement to insert rows into the tables. The values are extracted from dataframes imported from CSV files created previously by interacting with the sports API.

Run.py

In the first part of our *"run.py"* script, we import the necessary Python packages, create a database variable path, then initiate our app. The next two functions, *get_db()* and *close_db()* are used to connect to our SQLite database and automatically disconnect when our request is finished so as not to leave the connection open.

The next two functions, *create_standings()* and *stats()* are helper functions that take the data retrieved from our database, manipulate it, and create some HTML tables to be passed to our app. After that we get to the routing portion of the script, which are essentially the different URL paths the user can take.

Our app's home page can be found at the *@app.route('/')* section which ultimately renders our index.html file and actually passes some variables to our html page, which is an extremely useful perk of using Flask. We are passing 2 variables to index.html, "teams" and "seasons", which are database queries that query all the teams and seasons from our database. These variables are then used to populate our dynamic drop downs in our web app.

Figure 2 shows a quick snapshot of our *"index.html"* page which shows you how you can use Flask to dynamically populate your html files. Flask uses a package called Jinja, which uses the *"{{ }}"* and *"{% %}"*syntax that you'll see in the image below. Using this syntax, you can actually perform "Python-like" loops to construct HTML elements. Below, we are creating a drop-down that contains each team in the NHL from our database. As stated earlier, we passed the *teams* variable into our page, and below it is looped over to create drop-down options for every team.

```
<label for="teamDropDown">Choose a team:</label>
<select name="teamDropDown" id="teamDropDown">
    {% for o in teams %}
    <option value="{{ o.id }}" id="{{ o.id }}">{{ o.name }}</option>
    {% endfor %}
</select>
```

Figure 2. Flask variable in HTML

The last four routes are handlers for AJAX requests coming in from our JavaScript files. For example, when a user browses to the Team Stats page of our app, an AJAX request is made to the URL */api/get_stats* which is directed to the *@app.route('/api/get_stats')* section of our code below. This route will grab the season and split that are selected in the dropdown inputs and pass them to our database query to pull the appropriate data. It will then return the data in json format to our app and get rendered on the users screen.

Db.py

Our *"db.py"* Python file is a Python class that is used to execute SQL queries to pull data for our app. When this class is initialized, it makes a connection to our SQLite hockey.db database. The next function is a *select()* function that is used throughout the rest of the script. It initiates a cursor, then executes a SQL select statement and also has the ability to add different parameters. The parameters for our app change according to the selections chosen on each of the drop-down menus within each page.

The *"Db.py"* file is mainly used as a helper class to efficiently pull our data from the database. The input for the class comes from the user by way of the drop-downs and gets passed into each function. For example, the drop-downs throughout our app are populated by the *get_teams()* and *get_seasons()* functions in our *"Db.py"* file, as seen in Figure 3.

```
def get_teams(self):
    data = self.select(
        'SELECT * FROM teams')
    return [{
        'id': d[0],
        'name': d[1],
        'division': d[2],
        'conference': d[3]
    } for d in data]

def get_seasons(self):
    data = self.select(
        'SELECT distinct season from games')
    return [{
        'season': d[0]
    } for d in data]
```

Figure 3. db.py functions to retrieve data for drop-downs from the database

In each of these functions, we are using a SELECT statement to pull either the teams or seasons from their respective database tables, then returning the data as a Python dictionary. We return the data as a Python dictionary because that is easily transformed into json, which can be readily interpreted by JavaScript, which populates our pages.

Hockey.js

"Hockey.js" is one of our JavaScript files that populates our app. Within this file is a JS object called Hockey that performs various tasks like populating HTML tables and sending AJAX requests to the back-end server (Python). When our app is first populated, each page gets event handlers attached to each of the drop-down menus. These handlers sit and wait for the user to make a change to the drop-down menu, then spring into action. For example, when the user selects a different team from the menu, the handler will send an AJAX request to the server to get data for that team. To make this workflow clearer, below is an example of how our application functions:

1. A user launches our web app, HTML pages are rendered with the help of JavaScript, which also builds in event handlers on each of the drop-downs

2. When any drop-down menu is changed, the handler gets invoked and sends an AJAX request for fresh data based on the selection to the server back-end, which is Python

3. Python receives the request, and queries our database based on the parameters given

4. Python sends the data back to JavaScript in either json form or an already constructed HTML table

5. JavaScript clears the current page and appends the new data table in its place

Using the App

When creating our web app using Python, we chose the Flask framework to implement it. The Flask framework is simply a library of code that makes app development a lot easier. It is installed just like any Python package and contains many functions and reusable code needed for app development. It also allows you to keep all your HTML, JavaScript, CSS, and data files organized and in the appropriate folders and handles a lot of the logistics of rendering the pages of your app. After firing up a Flask app, it launches an HTTP server and your app is then live and viewable on your favorite web browser by going to your local host.

Installation

Required installations to run our web app:

- Python 3.x+
- Python package requirements:
    - Flask (pip install flask)
    - Sqlite3 (pip install sqlite3)
    - Pandas (pip install pandas)
    - Numpy (pip install numpy)

Installation guide

1. First download and install Python 3.x+
2. Once Python is installed, open up your terminal / command prompt
3. From there, type in “pip install flask”
4. Once Flask is installed, type in “pip install sqlite3”
5. Once SQLite is installed, type in “pip install pandas”
6. Once Pandas is installed, type in “pip install numpy”

Running the App

Our Flask app is launched by opening a terminal / command prompt at the location of our app folder. From there, you can type “python run.py” which will launch the app at the following url: http://localhost:8080. Type that url into your web browser of choice and you will see our apps homepage.

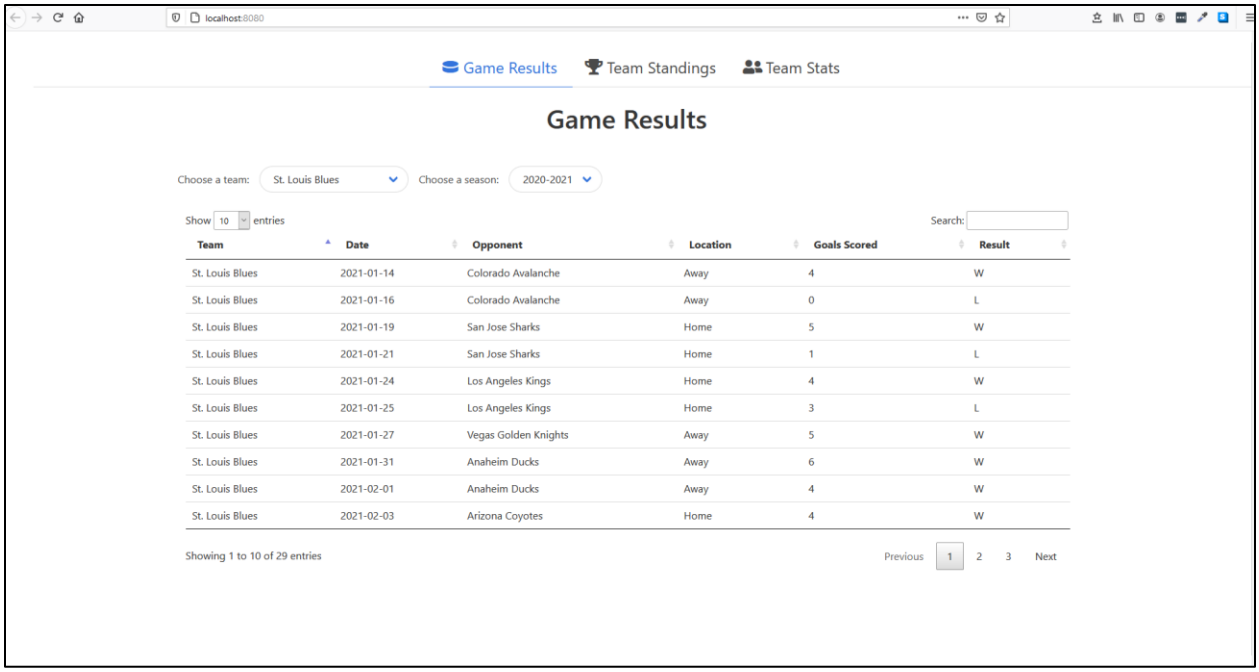

Figure 4. App homepage

Appendix

References

api-sports. (n.d.). *API-Hockey*. RapidAPI. Retrieved January 21, 2021, from https://rapidapi.com/api-sports/api/api-hockey/endpoints

Bulma. (n.d.). *Bulma*. Bulma. Retrieved February 3, 2021, from https://bulma.io/

Cloud Tables. (n.d.). *Examples index*. DataTables. Retrieved February 8, 2021, from https://datatables.net/examples/index

jQuery. (n.d.). *JQuery*. JQuery. Retrieved February 8, 2021, from https://jquery.com/

plasoft. (2020, May). *Hockey-LIVE.sk data*. Rapid API. https://rapidapi.com/palsoft/api/hockey-live-sk-data?endpoint=apiendpoint_877c439d-6545-4366-ba90-0bc9c2ed4ee4

SQLite Python. *SQLiteTutorial.* Retrieved March 5, 2021, from https://www.sqlitetutorial.net/sqlite-python/

Code

GetData.py

```
# Python Imports
import http.client
import pandas as pd
from io import StringIO
import requests

# API Request 1
url = "https://api-hockey.p.rapidapi.com/games/"

querystring = {"league":"57","season":"2020"}

headers = {
   'x-rapidapi-key': "23d7885293msh1f8b541140e738fp1c1e7fjsn226817e609e9",
   'x-rapidapi-host': "api-hockey.p.rapidapi.com"
   }

response = requests.request("GET", url, headers=headers, params=querystring)

print(response.text)

NHLdict=response.json()
NHLdict2 = NHLdict['response']
NHLdict2[0]
```

```
# API Request 2
url = "https://api-hockey.p.rapidapi.com/standings/"

querystring = {"league":"57","season":"2019"}

headers = {
   'x-rapidapi-key': "5bd247d204msh38487128c08f705p1bd68bjsn98976f65eec8",
   'x-rapidapi-host': "api-hockey.p.rapidapi.com"
   }

response = requests.request("GET", url, headers=headers, params=querystring)

NHL2019dict=response.json()
NHL2019dict2 = NHL2019dict['response'][0]
NHL2019dict2[0]

# Setting up teams
i = 0

teams = []

while i <31:
   team = NHL2019dict2[i]['team']['name']
   teams.append(team)
   i+=1

teams

# Create Divisional data
division_dict = {
   'St. Louis Blues': 'Central',
'Colorado Avalanche': 'Central',
'Vegas Golden Knights': 'Pacific',
'Edmonton Oilers': 'Pacific',
'Dallas Stars': 'Central',
'Winnipeg Jets': 'Central',
'Calgary Flames': 'Pacific',
'Vancouver Canucks': 'Pacific',
```

```
'Nashville Predators': 'Central',
'Minnesota Wild': 'Central',
'Arizona Coyotes': 'Pacific',
'Chicago Blackhawks': 'Central',
'Anaheim Ducks': 'Pacific',
'Los Angeles Kings': 'Pacific',
'San Jose Sharks': 'Pacific',
'Boston Bruins': 'Atlantic',
'Tampa Bay Lightning': 'Atlantic',
'Washington Capitals': 'Metropolitan',
'Philadelphia Flyers': 'Metropolitan',
'Pittsburgh Penguins': 'Metropolitan',
'Carolina Hurricanes': 'Metropolitan',
'Toronto Maple Leafs': 'Atlantic',
'Columbus Blue Jackets': 'Metropolitan',
'New York Islanders': 'Metropolitan',
'New York Rangers': 'Metropolitan',
'Florida Panthers': 'Atlantic',
'Montreal Canadiens': 'Atlantic',
'Buffalo Sabres': 'Atlantic',
'New Jersey Devils': 'Metropolitan',
'Ottawa Senators': 'Atlantic',
'Detroit Red Wings': 'Atlantic'
}

# Create team info dataframe
i = 0

teams_lists = []
teams = []

while i <31:
   teamID = NHL2019dict2[i]['team']['id']
   team = NHL2019dict2[i]['team']['name']
   conf = NHL2019dict2[i]['group']['name']
   div = division_dict[team]

   teams.append(teamID)
```

```
    teams.append(team)
    teams.append(conf)
    teams.append(div)

    teams_lists.append(teams)

    teams = []
    i+=1

teams_lists

NHLTeams_df = pd.DataFrame(teams_lists, columns = ["TeamID", "Team Name", "Conference",
"Division"])

NHLTeams_df

NHLTeams_df.to_csv("C:\\Users\\mmere\\Documents\\Drexel\\Masters\\INFO
606\\Project\\nhl_teams.csv", index = False)
```

 # API Request 3

```
url = "https://api-hockey.p.rapidapi.com/games/"

querystring = {"league":"57","season":"2020"}

headers = {
    'x-rapidapi-key': "23d7885293msh1f8b541140e738fp1c1e7fjsn226817e609e9",
    'x-rapidapi-host': "api-hockey.p.rapidapi.com"
    }

response = requests.request("GET", url, headers=headers, params=querystring)

print(response.text)

NHL2020dict=response.json()
NHL2020dict2 = NHL2020dict['response']
NHL2020dict2[0]
```

```
# Create 2020 games list
games_lists_2020 = []

games = []

i = 0

while i < 868:
   game_id = NHL2020dict2[i]['id']
   game_date = NHL2020dict2[i]['date']
   home_team = NHL2020dict2[i]['teams']['home']['name']
   away_team = NHL2020dict2[i]['teams']['away']['name']
   season = '2020-2021'

   games.append(game_id)
   games.append(game_date)
   games.append(home_team)
   games.append(away_team)
   games.append(season)

   games_lists_2020.append(games)

   games = []
   i+=1

games_lists_2020

# API Request 4
url = "https://api-hockey.p.rapidapi.com/games/"

querystring = {"league":"57","season":"2019"}

headers = {
   'x-rapidapi-key': "23d7885293msh1f8b541140e738fp1c1e7fjsn226817e609e9",
   'x-rapidapi-host': "api-hockey.p.rapidapi.com"
   }

response = requests.request("GET", url, headers=headers, params=querystring)
```

```
print(response.text)

NHL2019dict=response.json()
NHL2019dict2 = NHL2019dict['response']
NHL2019dict2[0]

# Create 2019 games list
games2019 = len(NHL2019dict2)
print(games2019)

games_lists_2019 = []

games = []

i = 0

while i < 1331:
    game_id = NHL2019dict2[i]['id']
    game_date = NHL2019dict2[i]['date']
    home_team = NHL2019dict2[i]['teams']['home']['name']
    away_team = NHL2019dict2[i]['teams']['away']['name']
    season = '2019-2020'

    games.append(game_id)
    games.append(game_date)
    games.append(home_team)
    games.append(away_team)
    games.append(season)

    games_lists_2019.append(games)

    games = []
    i+=1

games_lists_2019
games_lists_2019[-1]
```

```
# API Request 5
url = "https://api-hockey.p.rapidapi.com/games/"

querystring = {"league":"57","season":"2018"}

headers = {
   'x-rapidapi-key': "23d7885293msh1f8b541140e738fp1c1e7fjsn226817e609e9",
   'x-rapidapi-host': "api-hockey.p.rapidapi.com"
   }

response = requests.request("GET", url, headers=headers, params=querystring)

print(response.text)

NHL2018dict=response.json()
NHL2018dict2 = NHL2018dict['response']
NHL2018dict2[0]

# Create 2018 games list
games2018 = len(NHL2018dict2)
print(games2018)

games_lists_2018 = []

games = []

i = 0

while i < 1469:
   game_id = NHL2018dict2[i]['id']
   game_date = NHL2018dict2[i]['date']
   home_team = NHL2018dict2[i]['teams']['home']['name']
   away_team = NHL2018dict2[i]['teams']['away']['name']
   season = '2018-2019'

   games.append(game_id)
   games.append(game_date)
   games.append(home_team)
```

```
    games.append(away_team)
    games.append(season)

    games_lists_2018.append(games)

    games = []
    i+=1

games_lists_2018
games_lists_2018[-1]

# Create game schedule dataframe
total_games = []

for i in games_lists_2020:
    total_games.append(i)

for i in games_lists_2019:
    total_games.append(i)

for i in games_lists_2018:
    total_games.append(i)

total_games

NHLGames_df = pd.DataFrame(total_games, columns = ["GameID", "Date", "Home Team",
"Away Team", "Season"])

NHLGames_df['Date'] = pd.to_datetime(NHLGames_df['Date']).dt.date

NHLGames_df

NHLGames_df.to_csv("C:\\Users\\mmere\\Documents\\Drexel\\Masters\\INFO
606\\Project\\nhl_games.csv", index = False)

# Create 2020 game results list
games_stats_2020 = []
```

```
home_games = []
away_games = []
i = 0

while i < 868:
    game_id = NHL2020dict2[i]['id']

    home_team_id = NHL2020dict2[i]['teams']['home']['id']
    away_team_id = NHL2020dict2[i]['teams']['away']['id']

    first_score = NHL2020dict2[i]['periods']['first']
    if first_score == None:
        first_home = 0
        first_away = 0
    else:
        first_home, first_away = [int(i) for i in first_score.split("-")]

    second_score = NHL2020dict2[i]['periods']['second']
    if second_score == None:
        second_home = 0
        second_away = 0
    else:
        second_home, second_away = [int(i) for i in second_score.split("-")]

    third_score = NHL2020dict2[i]['periods']['third']
    if third_score == None:
        third_home = 0
        third_away = 0
    else:
        third_home, third_away = [int(i) for i in third_score.split("-")]

    if NHL2020dict2[i]['periods']['overtime'] == None:
        OT_home = 0
        OT_away = 0
    else:
        OT_home, OT_away = [int(i) for i in NHL2020dict2[i]['periods']['overtime'].split("-")]

    if NHL2020dict2[i]['scores']['home'] == None:
```

```
    home_final = 0
    away_final = 0
else:
    home_final = NHL2020dict2[i]['scores']['home']
    away_final = NHL2020dict2[i]['scores']['away']

if NHL2020dict2[i]['scores']['home'] == None:
        home_wins = 0
        home_loss = 0
        home_OTL = 0
        home_points = 0

        away_wins = 0
        away_loss = 0
        away_OTL = 0
        away_points = 0

elif NHL2020dict2[i]['status']['long'] == "After Over Time":
    if NHL2020dict2[i]['scores']['home'] > NHL2020dict2[i]['scores']['away']:
        home_wins = 1
        home_loss = 0
        home_OTL = 0
        home_points = 2

        away_wins = 0
        away_loss = 0
        away_OTL = 1
        away_points = 1
    else:
        home_wins = 0
        home_loss = 0
        home_OTL = 1
        home_points = 1

        away_wins = 1
        away_loss = 0
        away_OTL = 0
        away_points = 2
```

```
else:
    if NHL2020dict2[i]['scores']['home'] > NHL2020dict2[i]['scores']['away']:
        home_wins = 1
        home_loss = 0
        home_OTL = 0
        home_points = 2

        away_wins = 0
        away_loss = 1
        away_OTL = 0
        away_points = 0
    else:
        home_wins = 0
        home_loss = 1
        home_OTL = 0
        home_points = 0

        away_wins = 1
        away_loss = 0
        away_OTL = 0
        away_points = 2

home_games.append(game_id)
home_games.append(home_team_id)
home_games.append(first_home)
home_games.append(second_home)
home_games.append(third_home)
home_games.append(OT_home)
home_games.append(home_final)
home_games.append(home_wins)
home_games.append(home_loss)
home_games.append(home_OTL)
home_games.append(home_points)

away_games.append(game_id)
away_games.append(away_team_id)
away_games.append(first_away)
away_games.append(second_away)
```

```
    away_games.append(third_away)
    away_games.append(OT_away)
    away_games.append(away_final)
    away_games.append(away_wins)
    away_games.append(away_loss)
    away_games.append(away_OTL)
    away_games.append(away_points)

    games_stats_2020.append(home_games)
    games_stats_2020.append(away_games)

    home_games = []
    away_games = []
    i+=1

games_stats_2020[:10]

# Create 2019 game results list
games_stats_2019 = []

home_games = []
away_games = []
i = 0

while i < 1331:
    game_id = NHL2019dict2[i]['id']

    home_team_id = NHL2019dict2[i]['teams']['home']['id']
    away_team_id = NHL2019dict2[i]['teams']['away']['id']

    first_score = NHL2019dict2[i]['periods']['first']
    if first_score == None:
        first_home = 0
        first_away = 0
    else:
        first_home, first_away = [int(i) for i in first_score.split("-")]

    second_score = NHL2019dict2[i]['periods']['second']
```

```
if second_score == None:
   second_home = 0
   second_away = 0
else:
   second_home, second_away = [int(i) for i in second_score.split("-")]

third_score = NHL2019dict2[i]['periods']['third']
if third_score == None:
   third_home = 0
   third_away = 0
else:
   third_home, third_away = [int(i) for i in third_score.split("-")]

if NHL2019dict2[i]['periods']['overtime'] == None:
   OT_home = 0
   OT_away = 0
else:
   OT_home, OT_away = [int(i) for i in NHL2019dict2[i]['periods']['overtime'].split("-")]

if NHL2019dict2[i]['scores']['home'] == None:
   home_final = 0
   away_final = 0
else:
   home_final = NHL2019dict2[i]['scores']['home']
   away_final = NHL2019dict2[i]['scores']['away']

if NHL2019dict2[i]['scores']['home'] == None:
      home_wins = 0
      home_loss = 0
      home_OTL = 0
      home_points = 0

      away_wins = 0
      away_loss = 0
      away_OTL = 0
      away_points = 0

elif NHL2019dict2[i]['status']['long'] == "After Over Time":
```

```
    if NHL2019dict2[i]['scores']['home'] > NHL2019dict2[i]['scores']['away']:
        home_wins = 1
        home_loss = 0
        home_OTL = 0
        home_points = 2

        away_wins = 0
        away_loss = 0
        away_OTL = 1
        away_points = 1
    else:
        home_wins = 0
        home_loss = 0
        home_OTL = 1
        home_points = 1

        away_wins = 1
        away_loss = 0
        away_OTL = 0
        away_points = 2
else:
    if NHL2019dict2[i]['scores']['home'] > NHL2019dict2[i]['scores']['away']:
        home_wins = 1
        home_loss = 0
        home_OTL = 0
        home_points = 2

        away_wins = 0
        away_loss = 1
        away_OTL = 0
        away_points = 0
    else:
        home_wins = 0
        home_loss = 1
        home_OTL = 0
        home_points = 0

        away_wins = 1
```

```
            away_loss = 0
            away_OTL = 0
            away_points = 2

        home_games.append(game_id)
        home_games.append(home_team_id)
        home_games.append(first_home)
        home_games.append(second_home)
        home_games.append(third_home)
        home_games.append(OT_home)
        home_games.append(home_final)
        home_games.append(home_wins)
        home_games.append(home_loss)
        home_games.append(home_OTL)
        home_games.append(home_points)

        away_games.append(game_id)
        away_games.append(away_team_id)
        away_games.append(first_away)
        away_games.append(second_away)
        away_games.append(third_away)
        away_games.append(OT_away)
        away_games.append(away_final)
        away_games.append(away_wins)
        away_games.append(away_loss)
        away_games.append(away_OTL)
        away_games.append(away_points)

        games_stats_2019.append(home_games)
        games_stats_2019.append(away_games)

        home_games = []
        away_games = []
        i+=1

games_stats_2019[:10]
```

```
# Create 2018 game results list
games_stats_2018 = []

home_games = []
away_games = []
i = 0

while i < 1469:
    game_id = NHL2018dict2[i]['id']

    home_team_id = NHL2018dict2[i]['teams']['home']['id']
    away_team_id = NHL2018dict2[i]['teams']['away']['id']

    first_score = NHL2018dict2[i]['periods']['first']
    if first_score == None:
        first_home = 0
        first_away = 0
    else:
        first_home, first_away = [int(i) for i in first_score.split("-")]

    second_score = NHL2018dict2[i]['periods']['second']
    if second_score == None:
        second_home = 0
        second_away = 0
    else:
        second_home, second_away = [int(i) for i in second_score.split("-")]

    third_score = NHL2018dict2[i]['periods']['third']
    if third_score == None:
        third_home = 0
        third_away = 0
    else:
        third_home, third_away = [int(i) for i in third_score.split("-")]

    if NHL2018dict2[i]['periods']['overtime'] == None:
        OT_home = 0
        OT_away = 0
    else:
```

```
    OT_home, OT_away = [int(i) for i in NHL2018dict2[i]['periods']['overtime'].split("-")]

if NHL2018dict2[i]['scores']['home'] == None:
    home_final = 0
    away_final = 0
else:
    home_final = NHL2018dict2[i]['scores']['home']
    away_final = NHL2018dict2[i]['scores']['away']

if NHL2018dict2[i]['scores']['home'] == None:
        home_wins = 0
        home_loss = 0
        home_OTL = 0
        home_points = 0

        away_wins = 0
        away_loss = 0
        away_OTL = 0
        away_points = 0

elif NHL2018dict2[i]['status']['long'] == "After Over Time":
    if NHL2018dict2[i]['scores']['home'] > NHL2018dict2[i]['scores']['away']:
        home_wins = 1
        home_loss = 0
        home_OTL = 0
        home_points = 2

        away_wins = 0
        away_loss = 0
        away_OTL = 1
        away_points = 1
    else:
        home_wins = 0
        home_loss = 0
        home_OTL = 1
        home_points = 1

        away_wins = 1
```

```
        away_loss = 0
        away_OTL = 0
        away_points = 2
else:
    if NHL2018dict2[i]['scores']['home'] > NHL2018dict2[i]['scores']['away']:
        home_wins = 1
        home_loss = 0
        home_OTL = 0
        home_points = 2

        away_wins = 0
        away_loss = 1
        away_OTL = 0
        away_points = 0
    else:
        home_wins = 0
        home_loss = 1
        home_OTL = 0
        home_points = 0

        away_wins = 1
        away_loss = 0
        away_OTL = 0
        away_points = 2

home_games.append(game_id)
home_games.append(home_team_id)
home_games.append(first_home)
home_games.append(second_home)
home_games.append(third_home)
home_games.append(OT_home)
home_games.append(home_final)
home_games.append(home_wins)
home_games.append(home_loss)
home_games.append(home_OTL)
home_games.append(home_points)

away_games.append(game_id)
```

```
    away_games.append(away_team_id)
    away_games.append(first_away)
    away_games.append(second_away)
    away_games.append(third_away)
    away_games.append(OT_away)
    away_games.append(away_final)
    away_games.append(away_wins)
    away_games.append(away_loss)
    away_games.append(away_OTL)
    away_games.append(away_points)

    games_stats_2018.append(home_games)
    games_stats_2018.append(away_games)

    home_games = []
    away_games = []
    i+=1

games_stats_2018[:10]

# Combine 2020, 2019, 2018 game data to one results dataframe
total_games_stats = []

for i in games_stats_2020:
    total_games_stats.append(i)

for i in games_stats_2019:
    total_games_stats.append(i)

for i in games_stats_2018:
    total_games_stats.append(i)

total_games_stats[:25]

NHLGameStats_df = pd.DataFrame(total_games_stats, columns = ['Game ID', 'Team ID', "First
Period Goals", "Sceond Period Goals", "Third Period Goals", "Overtime Goals", "Final Score",
"Win", "Loss", "OT Loss", "Points"])
```

```
NHLGameStats_df

NHLGameStats_df.to_csv("C:\\Users\\mmere\\Documents\\Drexel\\Masters\\INFO
606\\Project\\nhl_gamestats.csv", index = False)

# API calls we didn't end up utilizing
url = "https://api-hockey.p.rapidapi.com/teams/"

querystring = {"league":"57","season":"2020"}

headers = {
   'x-rapidapi-key': "5bd247d204msh38487128c08f705p1bd68bjsn98976f65eec8",
   'x-rapidapi-host': "api-hockey.p.rapidapi.com"
   }

response = requests.request("GET", url, headers=headers, params=querystring)

print(response.text)

url = "https://api-hockey.p.rapidapi.com/teams/statistics/"

querystring = {"league":"57","season":"2020","team":"670"}

headers = {
   'x-rapidapi-key': "5bd247d204msh38487128c08f705p1bd68bjsn98976f65eec8",
```

Database.py

```
# Python Imports
import sqlite3
from sqlite3 import Error
import pandas as pd

 # Create connection to an existing SQLite database, or create a new SQLite database if not
existing yet
def create_connection(db_file):
   conn = None
   try:
```

```
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return conn

 # Create tables
def create_table(conn, create_table_sql):
    try:
        c = conn.cursor()
        c.execute(create_table_sql)
    except Error as e:
        print(e)

# Insert data into 3 tables: teams, games, and game_results
def create_teams(conn, team):
    sql = ''' INSERT INTO teams(TeamID,TeamName,Conference,Division)
              VALUES(?,?,?,?) '''
    cur = conn.cursor()
    cur.execute(sql, team)
    conn.commit()
    return cur.lastrowid

def create_games(conn, game):
    sql = ''' INSERT INTO games(GameID,GameDate,HomeTeam,AwayTeam,Season)
              VALUES(?,?,?,?,?) '''
    cur = conn.cursor()
    cur.execute(sql, game)
    conn.commit()
    return cur.lastrowid

def create_results(conn, result):
    sql = ''' INSERT INTO game_results(GameID,TeamID,goalsFirstPeriod,goalsSecondPeriod,
    goalsThirdPeriod,goalsOvertime,goalsTotal,win,loss,otl,points)
              VALUES(?,?,?,?,?,?,?,?,?,?,?) '''
    cur = conn.cursor()
    cur.execute(sql, result)
```

```
    conn.commit()
    return cur.lastrowid

# Organize all function to create the complete database
def main():
    database = "DB PATH"
    sql_create_teams_table = """ CREATE TABLE IF NOT EXISTS teams (
                                    TeamID text PRIMARY KEY,
                                    TeamName text NOT NULL,
                                    Conference text NOT NULL,
                                    Division text NOT NULL
                                ); """

    sql_create_games_table = """CREATE TABLE IF NOT EXISTS games (
                                    GameID text PRIMARY KEY,
                                    GameDate date NOT NULL,
                                    HomeTeam text NOT NULL,
                                    AwayTeam text NOT NULL,
                                    Season text NOT NULL
                                );"""

    sql_create_results_table = """CREATE TABLE IF NOT EXISTS game_results (
                                    GameID text NOT NULL,
                                    TeamID text NOT NULL,
                                    goalsFirstPeriod integer,
                                    goalsSecondPeriod integer,
                                    goalsThirdPeriod integer,
                                    goalsOvertime integer,
                                    goalsTotal integer,
                                    win integer,
                                    loss integer,
                                    otl integer,
                                    points integer,
                                    PRIMARY KEY (GameID,TeamID)
                                    FOREIGN KEY (GameID) REFERENCES games (GameID)
                                    FOREIGN KEY (TeamID) REFERENCES teams (TeamID)
                                );"""
```

```
    # create a database connection
    conn = create_connection(database)

    # create tables
    if conn is not None:
        # create teams table
        create_table(conn, sql_create_teams_table)

        # create games table
        create_table(conn, sql_create_games_table)

        # create games resulst table
        create_table(conn, sql_create_results_table)

    else:
        print("Error! cannot create the database connection.")

    with conn:
        #Insert teams csv
        team = pd.read_csv('nhl_teams.csv')
        for row in team.values.tolist():
            create_teams(conn, row)

        # insert games csv
        game = pd.read_csv('nhl_games.csv')
        for gm in game.values.tolist():
            create_games(conn, gm)

        #create game stats
        stats = pd.read_csv('nhl_gamestats.csv')
        for stat in stats.values.tolist():
            create_results(conn, stat)
```

 # Execute the *main()* function

```
if __name__ == '__main__':
    main()
```

Run.py

```
import os
from flask import Flask, g, json, jsonify, render_template, request
from db import Database
import pandas as pd
import numpy as np
import time

DATABASE_PATH = 'hockey.db'

app = Flask(__name__)

def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db = Database(DATABASE_PATH)
    return db

@app.teardown_appcontext
def close_db(exception):
    db = getattr(g, '_database', None)
    if db is not None:
        db.close()



def create_standings(data):
    combos = []
    for i in [[d['conf'], d['div']] for d in data]:
        if i not in combos:
            combos.append(i)
    cur_conf = combos[0][0]
    html = f"""<h1>{cur_conf}</h1>"""
    for c in combos:
        this_data = [d for d in data if d['conf'] == c[0] and d['div'] == c[1]]
        if this_data[0]['conf'] != cur_conf:
            cur_conf = this_data[0]['conf']
            html = html + f"""<h1>{cur_conf}</h1>"""
        html = html + f"""<h2>{c[1]}</h2><table class="table">
```

```
            <thead>
            <tr>
            <th>Team</th>
            <th>W</th>
            <th>L</th>
            <th>OTL</th>
            <th>Pts</th>
            </tr>
            </thead>"""
    for td in this_data:
        html = html + f"""
            <tr>
            <td>{td['team_name']}</td>
              <td>{td['w']}</td>
              <td>{td['l']}</td>
              <td>{td['otl']}</td>
              <td>{td['pts']}</td>
            </tr>
          """
    html = html + "</table>"
  return html


def stats(df, split):
  df = pd.DataFrame(df, columns=['gameid', 'teamname', 'Opponent', 'Location',
'goalsFirstPeriod', 'goalsSecondPeriod',
   'goalsThirdPeriod', 'goalsOvertime', 'goalsTotal', 'win', 'loss', 'otl', 'points', 'conference',
'division', 'season',
    'gamedate', 'oppConference', 'oppDivision', 'oppGoalsTotal'])
  df['gamesPlayed'] = 1
  if split == 'vs Division':
    df['Split'] = np.where(df['division'] == df['oppDivision'], 'Division', 'Non-Division')
    splits = ['Division', 'Non-Division']
  elif split == 'vs Conference':
    df['Split'] = np.where(df['conference'] == df['oppConference'], 'Conference', 'Non-
Conference')
    splits = ['Conference', 'Non-Conference']
  elif split == 'Home vs Away':
```

```
        df['Split'] = np.where(df['Location'] == 'H', ' Home', 'Away')
        splits = ['Home', 'Away']


    res = df.groupby(['teamname', 'Split']).sum().reset_index()
    res['GoalsFor/Game'] = round(res['goalsTotal'] / res['gamesPlayed'],2)
    res['GoalsAgainst/Game'] = round(res['oppGoalsTotal'] / res['gamesPlayed'],2)
    res['Record'] = res['win'].astype(str) + '-' + res['loss'].astype(str) + '-' + res['otl'].astype(str)
    res = res[['teamname', 'Split', 'gamesPlayed', 'Record', 'points', 'GoalsFor/Game',
'GoalsAgainst/Game']]
    res.columns = ['Team', 'Split', 'Games', 'Record', 'Pts', 'GoalsFor/Game', 'GoalsAgainst/Game']
    res = res.set_index(['Team','Split']).unstack(level=-1).reset_index()
    res.columns = res.columns.swaplevel(0, 1)
    res.sort_index(axis=1, level=0, inplace=True)
    res = res.fillna("")

    html = f"""<table class="table">
          <thead>
          <tr>
          <th> </th>
          <th colspan="5" style="text-align:center;border-right: 2px solid black;">{splits[0]}</th>
          <th colspan="5" style="text-align:center;">{splits[1]}</th>
          </tr>
          <tr>
          <th>Team</th>
          <th>Games</th>
          <th>GoalsAgainst/Game</th>
          <th>GoalsFor/Game</th>
          <th>Pts</th>
          <th style="border-right: 2px solid black;">Record</th>
          <th>Games</th>
          <th>GoalsAgainst/Game</th>
          <th>GoalsFor/Game</th>
          <th>Pts</th>
          <th>Record</th>
          </thead>
    """
    for d in res.itertuples():
```

```
        html = html + f"""
                <tr>
                <td>{d[1]}</td>
                <td>{d[2]}</td>
                <td>{d[3]}</td>
                <td>{d[4]}</td>
                <td>{d[5]}</td>
                <td style="border-right: 2px solid black;">{d[6]}</td>
                <td>{d[7]}</td>
                <td>{d[8]}</td>
                <td>{d[9]}</td>
                <td>{d[10]}</td>
                <td>{d[11]}</td>
                </tr>"""
    html = html + "</table>"
    return html


@app.route('/')
def home():
    return render_template('index.html', teams=get_db().get_teams(),
seasons=get_db().get_seasons())


@app.route('/api/get_games', methods=['GET', 'POST'])
def api_get_games():
    team_id = request.form.get('team_id')
    season = request.form.get('season')
    return jsonify({
        'games': get_db().get_games(team_id, season)
    })


@app.route('/api/get_standings', methods=['GET', 'POST'])
def api_get_standings():
    season = request.form.get('season')
    return jsonify({
        'standings': create_standings(get_db().get_standings(season))
    })
```

```
@app.route('/api/get_stats', methods=['GET', 'POST'])
def api_get_stats():
    season = request.form.get('season')
    split = request.form.get('split')
    return jsonify({
        'stats': stats(get_db().get_stats(season), split)
    })

@app.route('/api/get_teams', methods=['GET'])
def api_get_teams():
    return jsonify({
        'teams': get_db().get_teams()
    })

if __name__ == '__main__':
    app.run(host='localhost', port=8080, debug=True)
```

DB.py

```
import sqlite3

class Database:
    def __init__(self, path):
        self.conn = sqlite3.connect(path)

    def select(self, sql, parameters=[]):
        c = self.conn.cursor()
        c.execute(sql, parameters)
        return c.fetchall()


    def get_games(self, team, season):
        data = self.select("""
        select teams.teamname, games.gamedate, case when teams.teamname =
games.hometeam then games.awayteam else games.hometeam end as opponent,
        case when teams.teamname = games.hometeam then 'Home' else 'Away' end as location,
gr.goalsTotal as GoalsScored,
        case when gr.win = 1 then 'W' else 'L' end as result
```

```
        from game_results gr
        join games on games.gameid = gr.gameid
        join teams on teams.teamid = gr.teamid
        where teams.teamid = ? and games.gamedate < date() and games.season = ?
        order by games.gamedate asc""", [team, season])
        return [{
            'team': d[0],
            'date': d[1],
            'opp': d[2],
            'loc': d[3],
            'goals': d[4],
            'result': d[5]
        } for d in data]


    def get_standings(self, season):
        data = self.select("""
        select gr.teamid, teams.teamname, sum(gr.win) as W, sum(gr.loss) as L, sum(gr.otl) as OTL,
sum(gr.points) as Pts, teams.division, teams.conference  from game_results gr
        join teams on teams.teamid = gr.teamid
        join games on games.gameid = gr.gameid
        where games.season = ?
        group by gr.teamid, teams.teamname, teams.division, teams.conference
        order by teams.conference asc, teams.division asc, sum(gr.points) desc, sum(gr.loss) asc
        """, [season])
        return [{
            'team_id': d[0],
            'team_name': d[1],
            'w': d[2],
            'l': d[3],
            'otl': d[4],
            'pts': d[5],
            'div': d[6],
            'conf': d[7]
        } for d in data]

    def get_teams(self):
        data = self.select(
```

```
        'SELECT * FROM teams')
    return [{
        'id': d[0],
        'name': d[1],
        'division': d[2],
        'conference': d[3]
    } for d in data]

def get_seasons(self):
    data = self.select(
        'SELECT distinct season from games')
    return [{
        'season': d[0]
    } for d in data]

def get_stats(self, season):
    data = self.select("""
    with base as (
    select games.gameid,
        gr.teamid,
        teams.teamname,
        CASE
        when teams.teamname = games.hometeam then games.AwayTeam
        else games.HomeTeam
        end as Opponent,
        CASE
        when teams.teamname = games.hometeam then 'H' else 'A' end as Location,
        gr.goalsFirstPeriod,
        gr.goalsSecondPeriod,
        gr.goalsThirdPeriod,
        gr.goalsOvertime,
        gr.goalsTotal,
        gr.win,
        gr.loss,
        gr.otl,
        gr.points,
        teams.conference,
        teams.division,
```

```
        games.season,
        games.gamedate
    from game_results gr
    join games on games.gameid = gr.GameID
    join teams on teams.teamid = gr.TeamID)
    select base.gameid,
        base.teamname,
        base.opponent,
        base.location,
        base.goalsFirstPeriod,
        base.goalsSecondPeriod,
        base.goalsThirdPeriod,
        base.goalsOvertime,
        base.goalsTotal,
        base.win,
        base.loss,
        base.otl,
        base.points,
        base.conference,
        base.division,
        base.season,
        base.gamedate,
        teams.conference as oppConference,
        teams.division as oppDivision,
        gr.goalsTotal as oppGoalsTotal
    from base
    join teams on teams.teamname = base.opponent
    join game_results gr on gr.gameid = base.gameid and gr.teamid = teams.teamid
    where base.season = ? and base.gamedate <= date()
    """, [season])
    return data


def close(self):
    self.conn.close()
```

Hockey.js

```
function Hockey() {
  const that = this;

  this.updateGameResults = function (games) {
    $('#gameResults').empty();

    const table = $('<table class="table"></table>');
    const header = $(`
            <thead>
            <tr>
            <th>Team</th>
            <th>Date</th>
            <th>Opponent</th>
            <th>Location</th>
            <th>Goals Scored</th>
            <th>Result</th>
            </tr>
            </thead>
        `);
    $(table).append(header);
    for (var row = 0; row < games.length; row++) {
      const game = games[row];

      const tr = $(`
        <tr>
          <td>${game.team}</td>
          <td>${game.date}</td>
          <td>${game.opp}</td>
          <td>${game.loc}</td>
          <td>${game.goals}</td>
          <td>${game.result}</td>
        </tr>
      `);
      $(table).append(tr);

    }

    $('#gameResults').append(table);
```

```
    }

    this.load = function () {
        this.loadGameResults();
        this.loadStandings();
        this.loadStats();
        $('#teamDropDown').change(function () {
            that.loadGameResults();
        });

        $('#seasonDropDown').change(function () {
            that.loadGameResults();
        });
        $('#standingsSeasonDropDown').change(function () {
            that.loadStandings();
        });
        $('#statsSeasonDropDown').change(function () {
            that.loadStats();
        });

        $('#statsSplits').change(function () {
            that.loadStats();
        });
    }

    this.updateStandings = function (standings) {
        $('#standings').empty();

        $('#standings').append(standings);

    }

    this.loadStandings = function () {
        $.post('/api/get_standings', {season: $("#standingsSeasonDropDown").val()}, function
(data) {
            that.updateStandings(data.standings);
        });
    }
```

```
    this.loadGameResults = function () {
        $.post('/api/get_games', {team_id: $("#teamDropDown").val(), season:
$("#seasonDropDown").val()}, function (data) {
            that.updateGameResults(data.games);
        });
    }

    this.updateStats = function (stats) {
        $("#stats").empty();

        $("#stats").append(stats);
    }

    this.loadStats = function () {
        $.post('/api/get_stats', {season: $("#statsSeasonDropDown").val(), split:
$("#statsSplits").val()}, function (data) {
            that.updateStats(data.stats);
        });
    }
}
```

index.html

```
<!DOCTYPE html>
<html>
<head>
<title>Hockey Stats</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<!--jquery-->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha256-
4+XzXVhsDmqanXGHaHvgh1gMQKX40OUvDEBTu8JcmNs=" crossorigin="anonymous"></script>
<!--Bulma styles-->
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.1/css/bulma.min.css">
<!--font awesome for icons-->
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.2/css/all.min.css" integrity="sha512-
```

```
HK5fgLBL+xu6dm/Ii3z4xhlSUyZgTT9tuc/hSrtw6uzJOvgRr2a9jyxxT1ely+B+xFAmJKVSTbpM/CuL7
qxO8w==" crossorigin="anonymous" />

<!--link js to handle data transfer between back end and front end-->
<script type="text/javascript" src="static/js/hockey.js" ></script>
<script src="/static/lib/jquery/jquery.min.js"></script>
<script src="/static/lib/popper/popper.min.js"></script>


<!--data tables styles-->
<link rel="stylesheet" type="text/css" href="https://cdn.datatables.net/v/dt/dt-1.10.23/cr-
1.5.3/fh-3.1.8/kt-2.6.1/r-2.2.7/rg-1.1.2/datatables.min.css"/>

<!--data table scripts-->
<script type="text/javascript" src="https://cdn.datatables.net/v/dt/dt-1.10.23/cr-1.5.3/fh-
3.1.8/kt-2.6.1/r-2.2.7/rg-1.1.2/datatables.min.js"></script>

<!--custom javascript-->
<script type="text/javascript" src="static/js/custom.js"></script>

<!--custom css-->
<link rel="stylesheet" href="static/css/custom.css">

</head>


<body>
<div class="content">
<!--start tabs section-->
<div id="tabs-with-content">
  <!--start tab head-->
<div  class="tabs is-centered is-large ">
  <!--list out the tabs-->
  <ul>
    <li class="tab is-active" onclick="tabbing(event, 'Game Results')">
      <a >
        <span class="icon is-small"><i class="fas fa-hockey-puck" aria-hidden="true"></i></span>
        <span>Game Results</span>
```

```
    </a>
   </li>
   <li class="tab" onclick="tabbing(event, 'Team Standings')">
    <a >
     <span class="icon is-small"><i class="fas fa-trophy" aria-hidden="true"></i></span>
     <span>Team Standings</span>
    </a>
   </li>
   <li class="tab" onclick="tabbing(event, 'Team Stats')">
     <a >
     <span class="icon is-small"><i class="fas fa-user-friends" aria-hidden="true"></i></span>
     <span>Team Stats</span>
    </a>
   </li>
  </ul>
</div>
  <!--end tab header-->

  {% block content %}

  <!--begin game results section-->
<div class="tabContent container" id="Game Results"><h2 class="has-text-centered is-size-
2">Game Results</h2><br>
<!--Vertically center select boxes and labels-->
<div class="columns is-desktop is-vcentered">

<label for="teamDropDown">Choose a team:</label>

<div class="select is-rounded">
<select name="teamDropDown" id="teamDropDown">
   {% for o in teams %}
   <option value="{{ o.id }}" id="{{ o.id }}">{{ o.name }}</option>
   {% endfor %}
</select>
</div>

<!--end select is-rounded-->
```

```
<label for="seasonDropDown">Choose a season:</label>
<div class="select is-rounded">
<select name="seasonDropDown" id="seasonDropDown">
   {% for i in seasons %}
   <option value="{{ i.season }}" id="{{ i.season }}">{{ i.season }}</option>
   {% endfor %}
</select>
</div>
</div>
<!--End columns is-desktop is-vcentered-->
<div id="gameResults"></div>


</div>
 <!--end game results section-->

 <!--Start Team Standings section-->
 <div class="tabContent container" id="Team Standings" style="display:none">
  <h2 class="has-text-centered is-size-2">Team Standings</h2>
  <div class="columns is-vcentered">
  <label for="standingsSeasonDropDown">Choose a season:</label>
  <div class="select is-rounded">
  <select name="standingsSeasonDropDown" id="standingsSeasonDropDown">
     {% for i in seasons %}
     <option value="{{ i.season }}" id="{{ i.season }}">{{ i.season }}</option>
     {% endfor %}
  </select>
  </div>
  <!--end select is rounded-->
  </div>
  <!--end column is-vcentered-->

  <div id="standings"></div>

 </div>
 <!--End Team Standings section-->
 <!--Start Team Stats section-->
```

```
 <div class="tabContent container" id="Team Stats" style="display:none"><h2 class="has-text-
centered is-size-2">Team Stats</h2>
   <div class="columns is-vcentered">
   <label for="statsSeasonDropDown">Choose a season:</label>
   <div class="select is-rounded">
   <select name="statsSeasonDropDown" id="statsSeasonDropDown">
      {% for i in seasons %}
      <option value="{{ i.season }}" id="{{ i.season }}">{{ i.season }}</option>
      {% endfor %}
   </select>
   </div>
   <!--end select is-rounded-->

   <label for="statsSplits">Choose a split:</label>
   <div class="select is-rounded">
   <select name="statsSplits" id="statsSplits">
     <option value="Home vs Away" id="Home vs Away">Home vs Away</option>
     <option value="vs Division" id="vs Division">vs Division</option>
     <option value="vs Conference" id="Home vs Away">vs Conference</option>
   </select>
 </div>
 <!--end select is-rounded-->
 </div>
 <!--end colum is-vcentered-->

   <div id="stats"></div>


 <!--End team stats section-->

</div>
<!--end tabs-->
</div>
<!--end content-->
<script>
   $(function () {
      const hockey= new Hockey();
      hockey.load();
```

```
   });
</script>


{% endblock %}

</body>

</html>
```

Custom.css

```
#tabs-with-content {
  overflow: hidden;
}
.container {
  width: 90%;
}

.hidden {
  display: none;
}

.is-primary {
  margin: .5em !important;
}
input[type="text"] {
  width:20em;
}

#teamDateSubmission {
  display: block;
}

.error {
  color:red;
}

th[colspan="5"] {
```

```
  text-align: center;
}

.content {
  padding-bottom: 2em;
}

.is-rounded {
  margin:1em;
}
```

Custom.js

```
//control the tab elements of the navigation
function tabbing(evt, tabName) {
  var i, x, tablinks;
  x = document.getElementsByClassName("tabContent");
  for (i = 0; i < x.length; i++) {
     x[i].style.display = "none";
  }
  tablinks = document.getElementsByClassName("tab");
  for (i = 0; i < x.length; i++) {
     tablinks[i].className = tablinks[i].className.replace(" is-active", "");
  }
  document.getElementById(tabName).style.display = "block";
  evt.currentTarget.className += " is-active";
}
//end tab code


/*All of this code is either for items that did not make it into the final version of the app or was only for testing purposes

//display the correct fields for Game Results form based on the radio selection
function teamDate() {
  if (document.getElementById('team-choice').checked) {
     document.getElementById('teamSection').style.display = 'block';
     document.getElementById('dateSection').style.display = 'none';
```

```
    // document.getElementById('teamDateSubmission').style.display = 'block';
  } else {
    document.getElementById('teamSection').style.display = 'none';
   // document.getElementById('resultsTeamName').value = '';
    document.getElementById('dateSection').style.display = 'block';
    //document.getElementById('teamDateSubmission').style.display = 'block';
  }
}

//Create a constant of all NHL team names for error checking
const teamNames = ['tampa bay lightning','lightning','florida panthers','panthers','carolina
hurricanes',
'hurricanes','chicago blackhawks','blackhawks','columbus blue jackets','blue jackets','nashville
predators',
'predators','dallas stars','stars','detroit red wings','red wings','new york islanders','islanders',
'washington capitals','capitals','boston bruins','bruins','philadelphia flyers','flyers','pittsburgh
penguins',
'penguins','new york rangers','rangers','new jersey devils','devils','buffalo
sabres','sabres','toronto maple leafs',
'maple leafs','winnipeg jets','jets','montreal canadiens','canadiens','edmonton
oilers','oilers','calgary flames',
'flames','vancouver canucks','canucks','ottawa senators','senators','vegas golden
knights','golden knights',
'st. louis blues','st louis blues','blues','colorado avalanche','avalanche','minnesota
wild','wild','arizona coyotes',
'coyotes','los angeles kings','kings','anaheim ducks','ducks','san jose sharks','sharks']

//get the input values from the Game Results by team
function teamGameResults() {
  //turn input values into variables
  var team = document.getElementById('resultsTeamName').value;
  var resultsSeason = document.getElementById('teamGameResultsSeason').value;
  //make user input letter case not matter for matching data
  var teamLowerCase = team.toLowerCase();

//check if the entered team name matches an actual team
//input transformed to lowercase to make case not matter for the user
  if (teamNames.includes(teamLowerCase)) {
```

```
   console.log(team);
   console.log(resultsSeason)
   //clear error message since there's a valid result
   document.getElementById("teamError").innerHTML = '';
  } else {
   //display error message
   document.getElementById("teamError").innerHTML = 'Please enter a valid team name';
  }
}
//end teamGameResults

//get user input for game results by date
function dateGameResults() {
  //turn input values into variables
  //var startDate = new Date(document.getElementById('resultsStartDate').value);
  //var endDate = new Date(document.getElementById('resultsEndDate').value);
  var startDate = document.getElementById('resultsStartDate').value;
  var endDate = document.getElementById('resultsEndDate').value;

  //determine if the user date input is a valid format
  var dateRegex = /^(0?[1-9]|1[0-2])\/(0?[1-9]|1\d|2\d|3[01])\/(20)\d{2}$/;


  if (!dateRegex.test(startDate) ||!dateRegex.test(endDate) ) {
   document.getElementById("dateError").innerHTML = 'Please enter a valid date in the
DD/MM/YYYY format';
 } else {
   document.getElementById("dateError").innerHTML = '';
   console.log('Start date: ' + startDate);
   console.log('End date: ' + endDate);
   //console.log(startDate.getTime());
 }

}

//get input valuees from the Team Standings question
function teamStandings() {
  console.log(document.getElementById('teamStandingsSeason').value);
```

```
}

//get input valuees from the Team stats question
function teamStats() {
  console.log(document.getElementById('teamStats').value);
}

//get input valuees from the Team stats question
function playerStats() {
  console.log(document.getElementById('playerStats').value);
}
*/
```