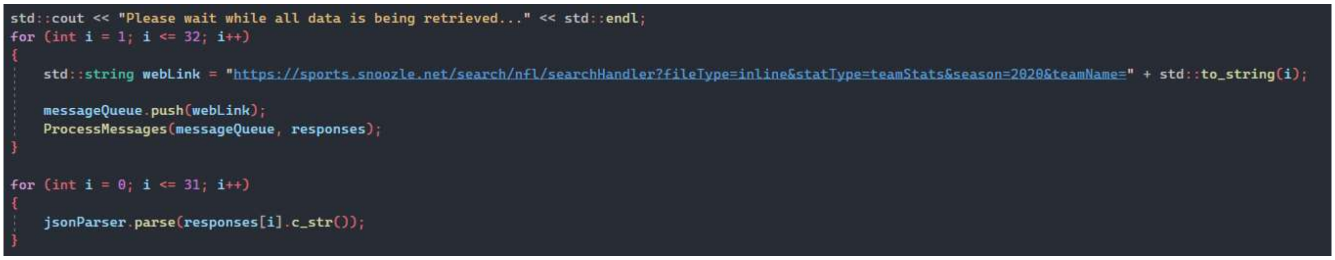
Jasmine Parmar, Kevin Osadiaye
CSE 681 Software Modeling
Project #2

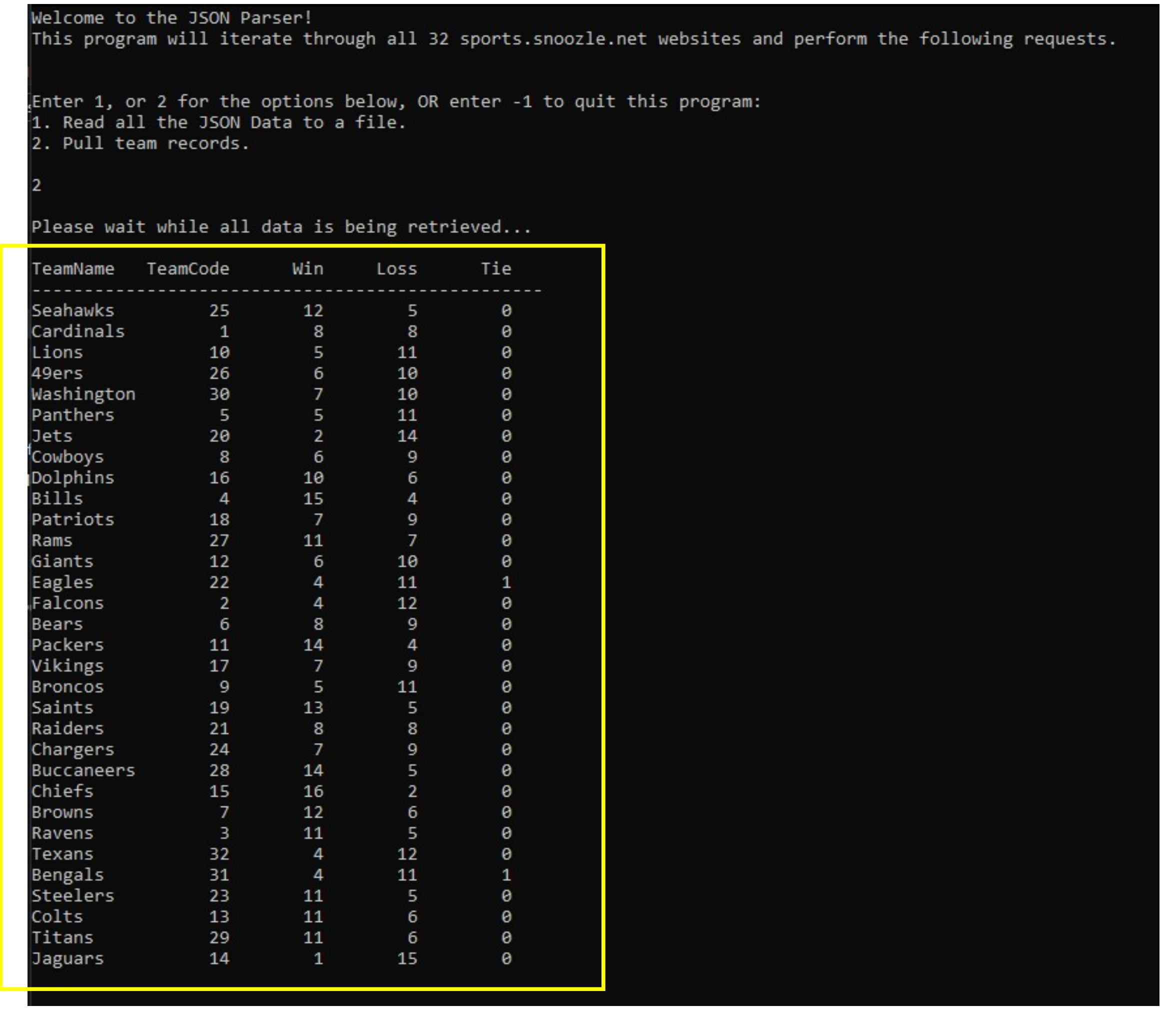
1. Requirement: *For this simulation, pull all the team's data for the 2020 season by modifying the get request value teamName (numbers 1-32). Printout all the teams' names, associated team number, and their season records using the score attribute from each game.*

In order to iterate through all 32 websites, the teamName request value was modified in the code below. The for loop below is included in the *promptAllOptions* function within the Main.cpp file. Each web link is pushed into a message queue, and all messages are processed through the ProcessMessages function.



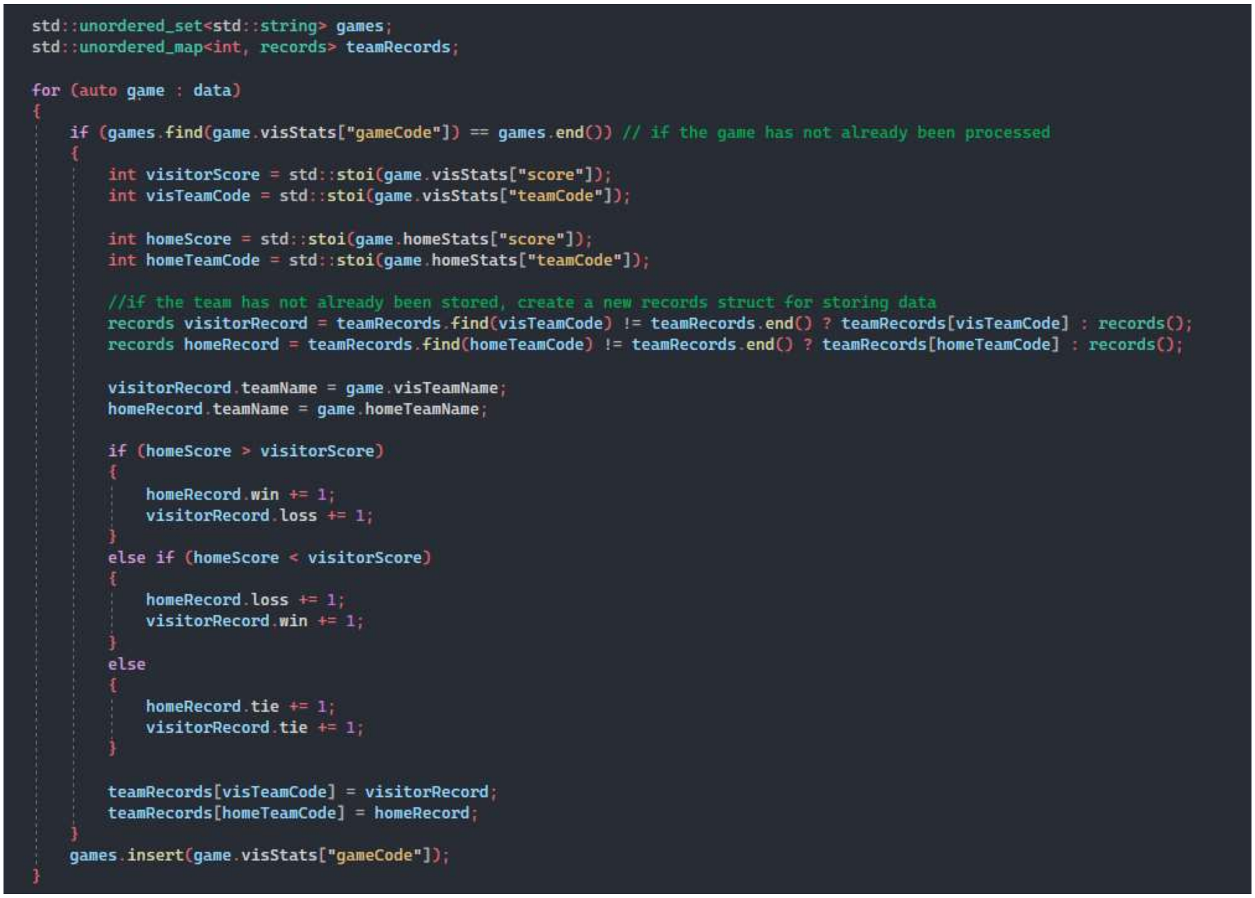
```
std::cout << "Please wait while all data is being retrieved..." << std::endl;
for (int i = 1; i <= 32; i++)
{
    std::string webLink = "https://sports.snoozle.net/search/nfl/searchHandler?fileType=inline&statType=teamStats&season=2020&teamName=" + std::to_string(i);

    messageQueue.push(webLink);
    ProcessMessages(messageQueue, responses);
}

for (int i = 0; i <= 31; i++)
{
    jsonParser.parse(responses[i].c_str());
}
```

The output for option #2 (Pull the team records and show wins/losses/ties is shown below.



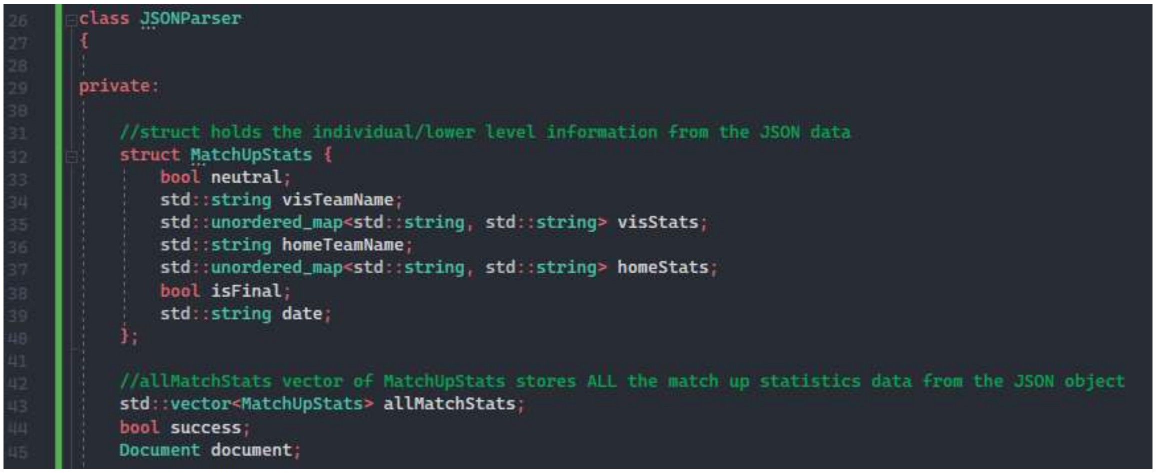
```
Welcome to the JSON Parser!
This program will iterate through all 32 sports.snoozle.net websites and perform the following requests.


Enter 1, or 2 for the options below, OR enter -1 to quit this program:
1. Read all the JSON Data to a file.
2. Pull team records.

2

Please wait while all data is being retrieved...

TeamName   TeamCode       Win       Loss       Tie
-------------------------------------------------
Seahawks         25        12          5         0
Cardinals         1         8          8         0
Lions            10         5         11         0
49ers            26         6         10         0
Washington       30         7         10         0
Panthers          5         5         11         0
Jets             20         2         14         0
Cowboys           8         6          9         0
Dolphins         16        10          6         0
Bills             4        15          4         0
Patriots         18         7          9         0
Rams             27        11          7         0
Giants           12         6         10         0
Eagles           22         4         11         1
Falcons           2         4         12         0
Bears             6         8          9         0
Packers          11        14          4         0
Vikings          17         7          9         0
Broncos           9         5         11         0
Saints           19        13          5         0
Raiders          21         8          8         0
Chargers         24         7          9         0
Buccaneers       28        14          5         0
Chiefs           15        16          2         0
Browns            7        12          6         0
Ravens            3        11          5         0
Texans           32         4         12         0
Bengals          31         4         11         1
Steelers         23        11          5         0
Colts            13        11          6         0
Titans           29        11          6         0
Jaguars          14         1         15         0
```

The calculations for the season records are performed in the *getTeamRecords* function within the Main.cpp file:



```cpp
std::unordered_set<std::string> games;
std::unordered_map<int, records> teamRecords;

for (auto game : data)
{
    if (games.find(game.visStats["gameCode"]) == games.end()) // if the game has not already been processed
    {
        int visitorScore = std::stoi(game.visStats["score"]);
        int visTeamCode = std::stoi(game.visStats["teamCode"]);

        int homeScore = std::stoi(game.homeStats["score"]);
        int homeTeamCode = std::stoi(game.homeStats["teamCode"]);

        //if the team has not already been stored, create a new records struct for storing data
        records visitorRecord = teamRecords.find(visTeamCode) != teamRecords.end() ? teamRecords[visTeamCode] : records();
        records homeRecord = teamRecords.find(homeTeamCode) != teamRecords.end() ? teamRecords[homeTeamCode] : records();

        visitorRecord.teamName = game.visTeamName;
        homeRecord.teamName = game.homeTeamName;

        if (homeScore > visitorScore)
        {
            homeRecord.win += 1;
            visitorRecord.loss += 1;
        }
        else if (homeScore < visitorScore)
        {
            homeRecord.loss += 1;
            visitorRecord.win += 1;
        }
        else
        {
            homeRecord.tie += 1;
            visitorRecord.tie += 1;
        }

        teamRecords[visTeamCode] = visitorRecord;
        teamRecords[homeTeamCode] = homeRecord;
    }
    games.insert(game.visStats["gameCode"]);
}
```

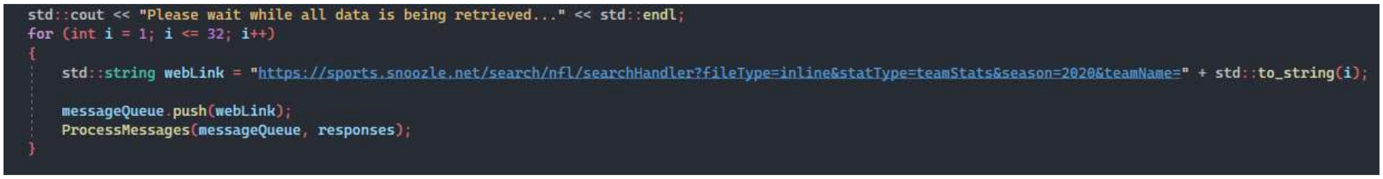
The main data structure used was not modified since the previous Project #1. The data structure used to hold all Match Up Statistics information is the MatchUpStats struct. Once information for one game is passed to the struct, all struct information is passed to a vector of MatchUpStats (shown on line 43 below).



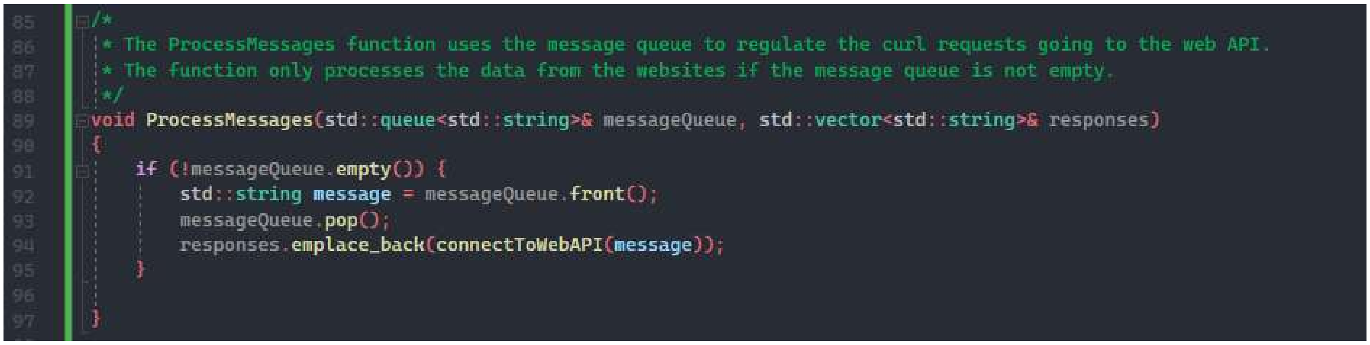
```cpp
26  class JSONParser
27  {
28
29  private:
30
31      //struct holds the individual/lower level information from the JSON data
32      struct MatchUpStats {
33          bool neutral;
34          std::string visTeamName;
35          std::unordered_map<std::string, std::string> visStats;
36          std::string homeTeamName;
37          std::unordered_map<std::string, std::string> homeStats;
38          bool isFinal;
39          std::string date;
40      };
41
42      //allMatchStats vector of MatchUpStats stores ALL the match up statistics data from the JSON object
43      std::vector<MatchUpStats> allMatchStats;
44      bool success;
45      Document document;
```

2. Message Queue Extra Credit: Message queues are a useful middle for controlling responses, add a message queue implementation to regulating the requests going to your server. You can use any implementation you want (binaries or build from source). Include a model of the message queue in your documentation based on your implementation.

As mentioned previously, a message queue was implemented in this project in order to regulate requests being sent to the Web API. The messages are sent to a queue that contains the messages. The *ProcessMessages* function checks whether the message queue is empty, and if it is not empty, then the message is popped from the queue and the data is obtained from the Web API.

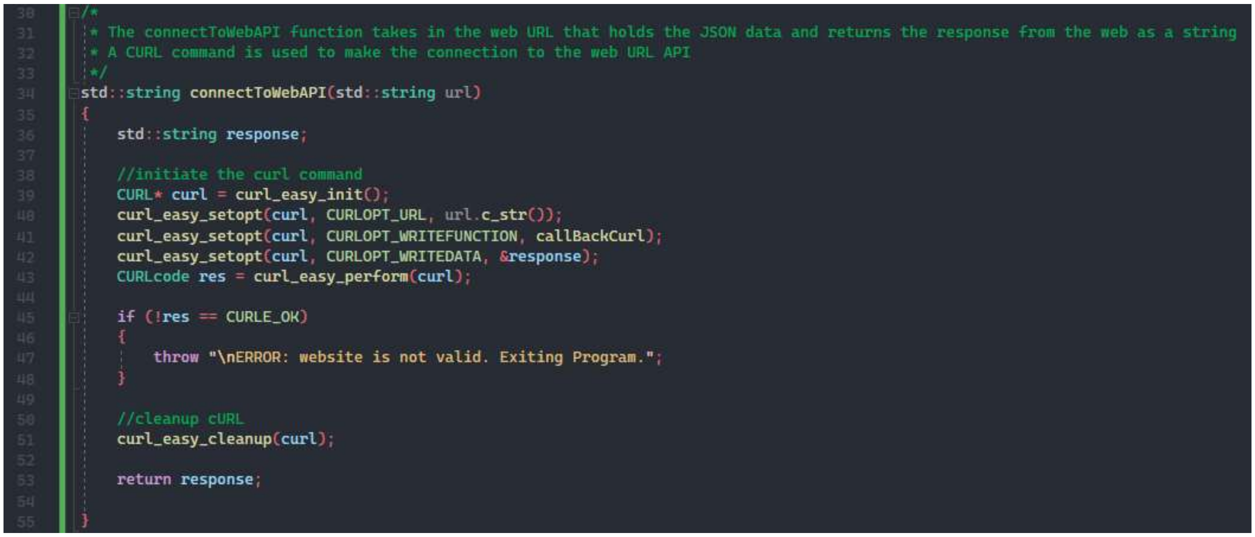


```
std::cout << "Please wait while all data is being retrieved..." << std::endl;
for (int i = 1; i <= 32; i++)
{
    std::string webLink = "https://sports.snoozle.net/search/nfl/searchHandler?fileType=inline&statType=teamStats&season=2020&teamName=" + std::to_string(i);

    messageQueue.push(webLink);
    ProcessMessages(messageQueue, responses);
}
```



```
/*
* The ProcessMessages function uses the message queue to regulate the curl requests going to the web API.
* The function only processes the data from the websites if the message queue is not empty.
*/
void ProcessMessages(std::queue<std::string>& messageQueue, std::vector<std::string>& responses)
{
    if (!messageQueue.empty()) {
        std::string message = messageQueue.front();
        messageQueue.pop();
        responses.emplace_back(connectToWebAPI(message));
    }


}
```

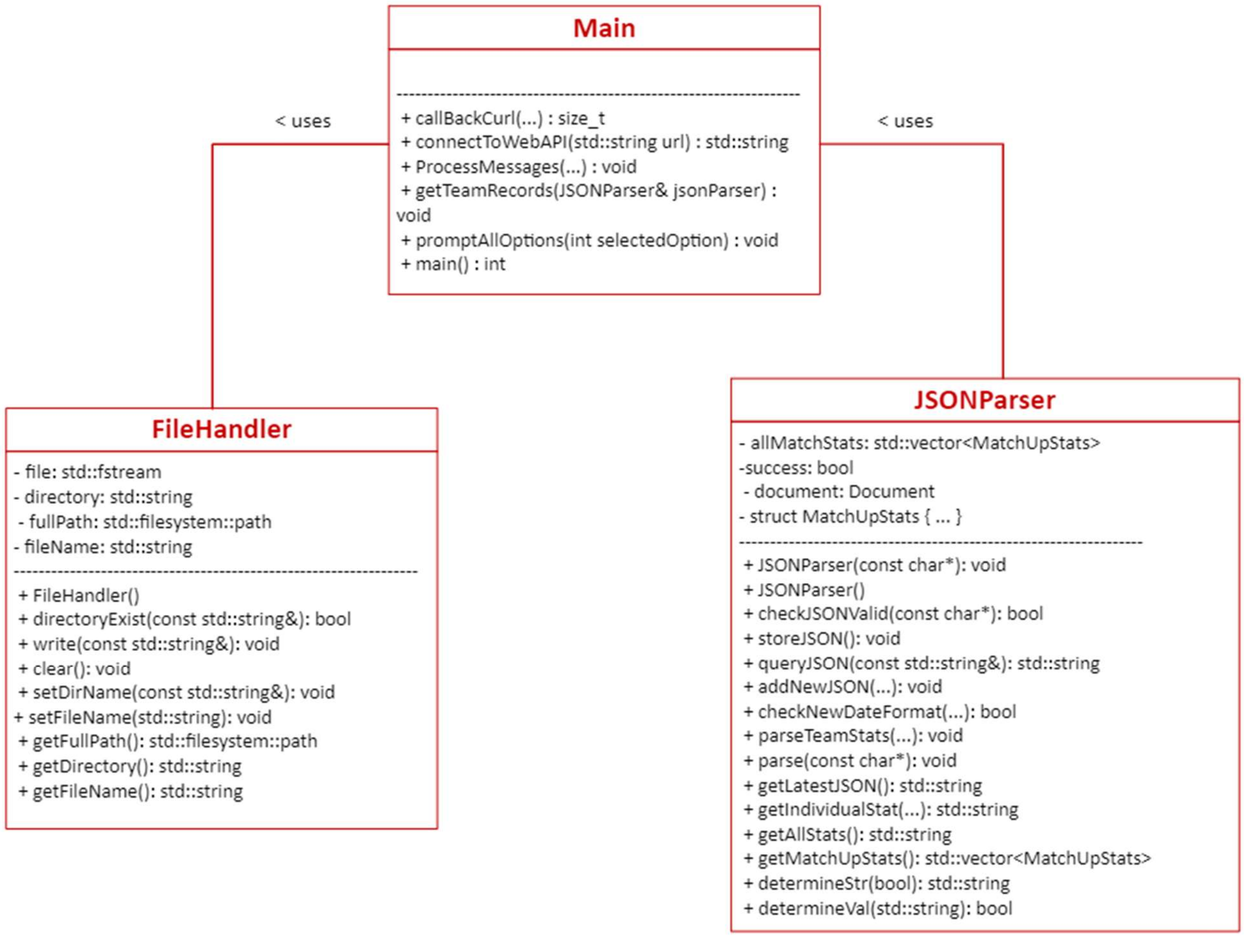
The connectToWebAPI function was also reused from the previous Project #1 to make the Web API connection. This function utilizes lib curl to make the connection.



```
/*
* The connectToWebAPI function takes in the web URL that holds the JSON data and returns the response from the web as a string
* A CURL command is used to make the connection to the web URL API
*/
std::string connectToWebAPI(std::string url)
{
    std::string response;

    //initiate the curl command
    CURL* curl = curl_easy_init();
    curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, callBackCurl);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, &response);
    CURLcode res = curl_easy_perform(curl);

    if (!res == CURLE_OK)
    {
        throw "\nERROR: website is not valid. Exiting Program.";
    }

    //cleanup cURL
    curl_easy_cleanup(curl);

    return response;

}
```

3. System Modeling 1: Show the model of your classes using a design paradigm (UML, flow chart, etc.) using a design paradigm of your choosing.

UML Diagram



Main

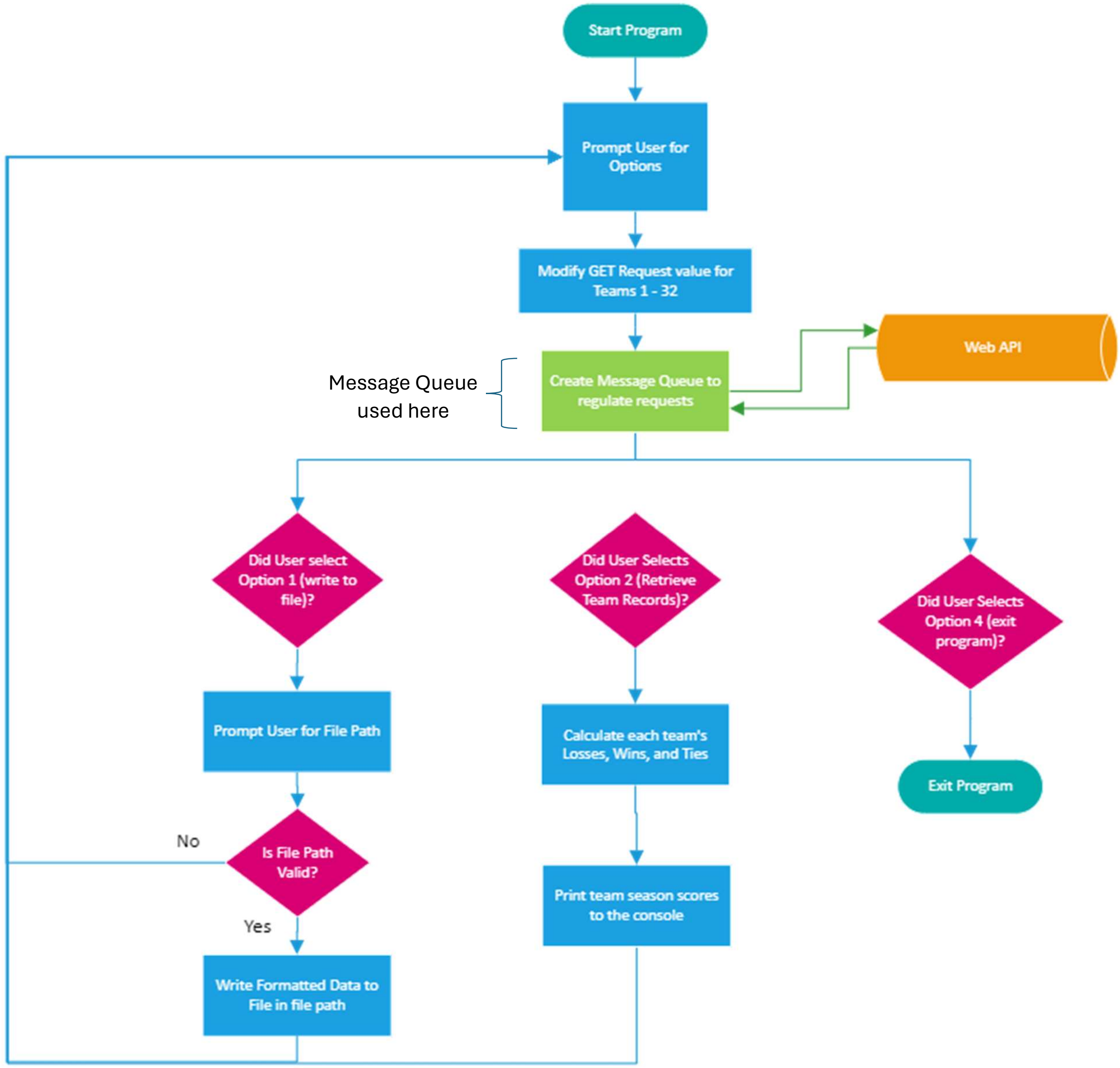
```
---------------------------------------------------------------
+ callBackCurl(...) : size_t
+ connectToWebAPI(std::string url) : std::string
+ ProcessMessages(...) : void
+ getTeamRecords(JSONParser& jsonParser) : void
+ promptAllOptions(int selectedOption) : void
+ main() : int
```

FileHandler

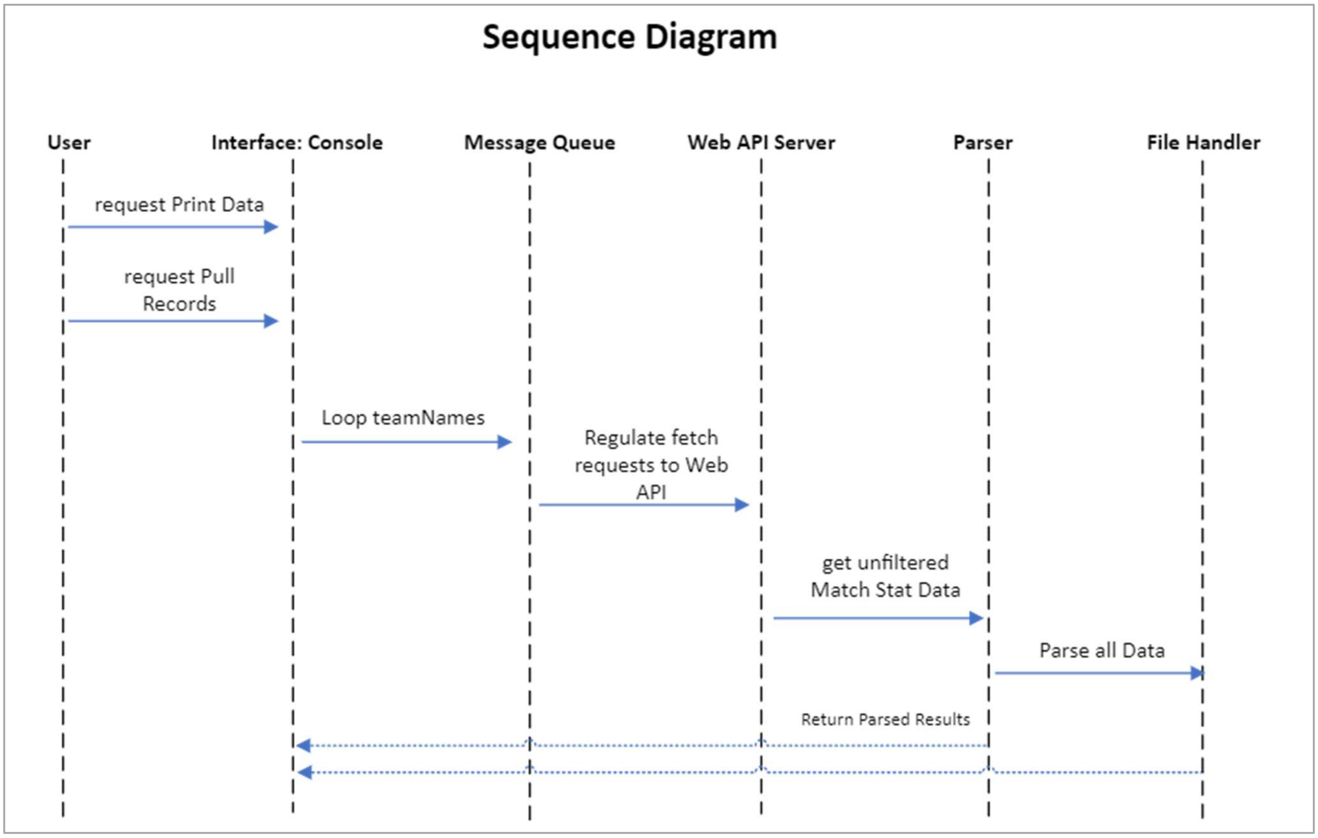
```
- file: std::fstream
- directory: std::string
- fullPath: std::filesystem::path
- fileName: std::string
---------------------------------------------------------------
+ FileHandler()
+ directoryExist(const std::string&): bool
+ write(const std::string&): void
+ clear(): void
+ setDirName(const std::string&): void
+ setFileName(std::string): void
+ getFullPath(): std::filesystem::path
+ getDirectory(): std::string
+ getFileName(): std::string
```

JSONParser

```
- allMatchStats: std::vector<MatchUpStats>
-success: bool
- document: Document
- struct MatchUpStats { ... }
---------------------------------------------------------------
+ JSONParser(const char*): void
+ JSONParser()
+ checkJSONValid(const char*): bool
+ storeJSON(): void
+ queryJSON(const std::string&): std::string
+ addNewJSON(...): void
+ checkNewDateFormat(...): bool
+ parseTeamStats(...): void
+ parse(const char*): void
+ getLatestJSON(): std::string
+ getIndividualStat(...): std::string
+ getAllStats(): std::string
+ getMatchUpStats(): std::vector<MatchUpStats>
+ determineStr(bool): std::string
+ determineVal(std::string): bool
```


Start Program
Prompt User for Options
Modify GET Request value for Teams 1 - 32
Create Message Queue to regulate requests
Web API
Message Queue used here
Did User select Option 1 (write to file)?
Did User Selects Option 2 (Retrieve Team Records)?
Did User Selects Option 4 (exit program)?
Prompt User for File Path
Calculate each team's Losses, Wins, and Ties
Exit Program
No
Is File Path Valid?
Yes
Print team season scores to the console
Write Formatted Data to File in file path

4. System Modeling 2: Show the model of your system including the software and API server using a design paradigm of your choosing including message queue if implemented.


Sequence Diagram
User
Interface: Console
Message Queue
Web API Server
Parser
File Handler
request Print Data
request Pull Records
Loop teamNames
Regulate fetch requests to Web API
get unfiltered Match Stat Data
Parse all Data
Return Parsed Results