User stories/User requirements. Describe the flows that users will go through and how they will interact with the application.

- Flow of the app
    - (eventually) user login/logout
        - All endpoints will only be accessible through authorization of the user (ie. they must be logged in)
    - Can interact with the application through the url AND also through a very simple UI (for GET calls)
        - Creating queries in the url
        - Or create queries by entering data in input field text boxes
        - The screen will display a visually reasonable format of the return data
    - Can also interact with the application through a very simple UI
        - A field and button to enter data for various endpoints
        - A section that displays a table of results and or information about the given query
- User stories
    - As a user, I want to be able to login/logout so that my info is secure
        - As a user, I want all endpoints to require authorization so that only authorized users can access the application
        - As a user, I want to be able to create an account if I do not already have one
    - As a user, I want to be able to interact with the application through the URL so that I can easily navigate and use the application
        - As a user, I want to be able to create queries in the URL so that I can retrieve specific information
        - As a user, I want to be able to add athletes to the database through the URL so that the database is up to date
    - As a user, I want to be able to interact with the application through a simple UI so that I can easily use the application

        - As a user, I want the screen to display the return data in a visually reasonable format so that I can easily read and understand it
        - As a user, I want a field and button to enter data for various endpoints so that I can input data easily
    - *Why All Sports Warehouse is better than a CRUD*: As a user, I want to be able to compare athletes within a sport

Documentation on what endpoints you will create. This should be at the same level of detail as what I provided in Assignment 1.

get_athlete(id: str) (GET)

- This endpoint returns a single athlete by its identifier. For each athlete it returns:
    - athlete_id: The internal id of the athlete.
    - name: The name of the athlete
    - team_id: The team id the athlete plays for
    - age: The age of the athlete
    - stats: a json returning some of the stats of the athlete
        - games_played, minutes_played, field_goal_percentage, three_point_percentage, free_throw_percentage, total_rebounds, assist, steals, blocks, points

get_game(id: str) (GET)

- This endpoint returns a single game by its identifier
    - Game_id: internal id of game
    - Home_team: name of home team
    - Away_team: name of away team
    - Winner_team: name of winner team
    - Home_team_score: score of home team
    - Away_team_score: score of away team
    - Date: the date the game was held

get_team(id: str) (GET)

- This endpoint returns a single team by its identifier. For each team it returns:
    - team_id: The internal id of the team
    - team_name: The name of the team
    - Wins: Number of games the team won
    - Losses: Number of games the team lost
    - Average Points for: Average number of points the team scored
    - Average Points allowed: Average number of points team allowed

compare_team(team_names: list<str>, compare_by: StatOptions): (GET)

- This endpoint compares any number of teams (> 1) by a single metric
    - Team_names: list of team names (length >1)
    - Compare_by must be one of the following values
        - “Wins”
        - “Number of athletes”
        - (the following will be averaged across all the team’s players)
        - “Average field_goal_percentage”
        - “Average three_point_percentage”
        - “Average free_throw_percentage”
        - “Average points”

compare_athletes(athlete_names: list<str>, stat: StatOptions) (GET)

- This endpoint returns a comparison between the specified athletes (as a table). It allows the user to compare athletes by a stat in `StatOptions`.
    - Could be empty (compares all stats) or could be a list of stats to specifically compare by
- The endpoint will return athletes in ascending or descending order depending on the context of `stat`. i.e. if a higher number is better, it will return athletes in descending order.
- Input validation
    - The list of athletes has length of at least two

-

add_athlete(name: str, team_id: int, age: int, stats: SportStatsObj): (POST)

- This endpoint adds an athlete to the database. The athlete is represented by:
    - name: the name of the athlete
    - team_id: the team id of the athlete's team
    - age: the age of the athlete
    - stats: the stats of the athlete, represented as a <Sport>StatsObj, where <Sport> is the sport of the athlete, and <Sport>StatsObj represents the stats of the athlete specific to the sport
- The endpoint ensures the athlete does not already exist in the database
- The endpoint returns the id of the athlete created

add_game(home_team_id: int, away_team_id: int, winner_id: int, home_team_score: int, away_team_score: int): (POST)

- This endpoint adds a game to the database. The game is represented by:
    - home_team_id: the id of the home team
    - away_team_id: the id of the away team
    - winner_id: the id of the winner's team
    - home_team_score: the score of home team
    - away_team_score: the score of away team
- The endpoint ensures the game does not already exist in the database
- The endpoint returns the id of the game created

(Nice to have (not for initial 5 endpoint implementation))

list_athletes_in_team(team: str, limit: int, offset: int, sort: str): (GET)

- This endpoint will return a list of athletes in the specified team. For each athlete it will return:
    - Athlete_id
    - Athlete name
    - Team_ID
    - Age
    - Stats: stats is represented by a dictionary (see definition under get_athlete)
- It should also be able to sort by:
    - Athlete: sort athletes by name alphabetically
    - Age: sort athletes by age in either ascending or descending order
    - Stats: sort athletes by a specific stat in their sport in either ascending or descending order. This stat will be specific to `sport`.

list_all_athletes(athlete_name: str, limit: int, offset: int, sort: str) (GET)

- Returns all athletes whose name contains 'name.' If no name is given, it will return all athletes. For each athlete it returns:
    - athlete_id: The internal id of the athlete.
    - name: The name of the athlete
    - Team_id: The team id of the athlete's team
    - age : The age of the athlete
    - Stats: stats is represented by a dictionary (see definition under get_athlete)
- You can filter for athletes whose name contains a string by using the `name` query parameter.
- You can sort the results by using the `sort` query parameter:
    - athlete_id, name, sport, gender, age

- The `limit` and `offset` query parameters are used for pagination. The `limit` query parameter specifies the maximum number of results to return. The `offset` query parameter specifies the number of results to skip before returning results.

Detailed descriptions of edge cases and transaction flows. For example, if the app has a credit card checkout, describe what happens if the credit card transaction fails, what happens if the user tries to cancel mid-way through, etc.

- All user-writes will have data integrity checks
    - The data must
        - Not be a duplicate to something already in the DB
        - Match the required format and include all required fields for the given write
    - Failed writes will return an error code and error message to the screen, and prompt the user to follow the format for the given write
- All reads
    - All reads will have a limit to ensure cases where the data is large and may cause performance issues
    - Failed reads will return an error code and a message
    - Inputs will be converted to all uppercase
- User login creation
    - The username must be unique (must not already exist in the DB)
        - If it does already exist, inform the user
    - Require passwords with a minimum number of characters and special characters
    - Require the user to type in the password twice
- User login
    - Reject login after a specified number of failed logins
    - Have case sensitivity on the password