

- Note: Changes after the peer-review and Professor's comments are highlighted

**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
  - sport: The sport the athlete plays
  - team: The team the athlete plays for
  - gender: The gender of the athlete
  - age: The age of the athlete
  - years: the years they played
  - stats: a json returning some of the stats of the athlete
    - stats is represented by a dictionary in which the keys are dependent on the sport the athlete plays. For example, if the athlete plays baseball, stats will include batting average.

#### **compare\_athletes(athlete\_names: list<str>, stat: str) (GET)**

- This endpoint returns a comparison between the specified athletes (as a table). It allows the user to compare athletes by `stat`.

- 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
  - All athletes are of the same sport
  - The list of athletes has length of at least two

**list\_athletes\_in\_sport(athlete\_name: str, sport: str, year: int, limit: int, offset: int, sort: str): (GET)**

- This endpoint will return a list of athletes in the specified sport that played in `year`. For each athlete it will return:
  - Athlete\_id
  - Athlete name
  - Sport
  - Gender
  - Age
  - Year (the years they played)
  - Stats
    - stats is represented by a dictionary in which the keys are dependent on the sport the athlete plays. For example, if the athlete plays baseball, stats will include batting average.
- 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\_athletes\_all\_sports(athlete\_name: str, years: int, 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
  - sport: The sport the athlete plays
  - gender: The gender of the athlete
  - Year: the years they played
  - age : The age of the 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
  - Note: you CANNOT sort by stats in this endpoint, since athletes from different sports may not share the same stats
- 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.

**list\_athletes\_by\_team(team: str, year: int, limit: int, offset: int, sort: str) -> roster (GET)**

- This endpoint returns:
  - team\_name: the name of the team
  - team\_id: the id of the team
  - sport: the sport of the team
  - athletes: for each athlete on the team it returns:
    - athlete\_id: The internal id of the athlete.
    - name: The name of the athlete
    - sport: The sport the athlete plays
    - gender: The gender of the athlete
    - age : The age of the athlete
    - stats: a json returning some of the stats of the athlete

- stats is represented by a dictionary in which the keys are dependent on the sport the athlete plays. For example, if the athlete plays baseball, stats will include batting average.
- You can sort the list of athletes by using the `sort` query parameter by:
  - athlete\_id, name, sport, gender, age, stat
  - Note: stat is specific to the sport.
- The `limit` and `offset` query parameters are used for pagination. The `limit` query parameter specifies the maximum number of `athletes` to return. The `offset` query parameter specifies the number of results to skip before returning results.

**add\_athlete(name: str, sport: str, gender: str, team: str, age: int, stats: SportStatsObj):**

**(POST)**

- This endpoint adds an athlete to the database. The athlete is represented by:
  - name: the name of the athlete
  - sport: the sport the athlete plays
  - gender: the gender of the athlete
  - team: the team of the athlete
  - 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

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