Part 1: If your team name, team membership, planned programming language, platform and/or team github repository has changed since your team formation assignment, please explain. If not, simply state that nothing has changed. You need permission from the instructor in advance (before you submit this assignment) to change to any language other than C++, Java, Python and to change to any platform other than Linux, Mac, Windows.

Part 1: State the full names and UNI's of all team members.

- Isabella Cho (isc2120)
- Emanuel Daka (ed2918)
- Stephane Meunier (sjm2264)
- Tanay Murdia (tm3149)
- Patrycja Przewoznik (pap2154)

Part 2: Choose a name for your team consisting only of alphanumeric characters.

- STIPE

Part 3: State which programming language and platform your team plans to use.

- Java
- Mac

Part 4: Submit the link to your team github repository.

- https://github.com/Daktor99/ASWE_TeamProject

Part 2: Write a few paragraphs that provide an overview of the service that your team would like to develop and answers these three sets of questions:

1. What will your service do? What kind of functionality or features will it provide?

Our service will implement the logic for a Fantasy Basketball League. All communications to and from the Fantasy Basketball service will be over RESTful API calls. These API calls will be used to view, create and modify the state of a given Fantasy League instance being hosted by our service. Client code will be able to utilize our service to perform the core logic necessary to be able to run fantasy teams. All communications with our service will be through API calls and there is no GUI that will be developed as a part of our service.

The service will provide the core functionality of Fantasy Leagues that can be organized into the following categories:

- *User functionality*
- *Team functionality*
- *League functionality*
- *Player functionality.*

User Functionality:
In order for a user to participate in a Fantasy League, they must first register with the service. Once registered, a user will be able to either create their own Fantasy League or join an existing Fantasy League that has been created by another player. Once a player is assigned to a Fantasy League, they may create their own Fantasy Team to participate in that instance of a Fantasy League. Once Fantasy Players have been assigned to their respective Fantasy Teams, the user that is the "owner" of the Fantasy Team will have the ability to manage their starting lineups, make trades, drop players from their team or pick up "free agents" (players not currently drafted to a team) in preparation of their games against other users in the League.

Team Functionality:
Every Fantasy League consists of multiple Fantasy Teams. Each Fantasy Team is "owned" by only one user who has the rights to change the structure of the team. Possible changes include, proposing and accepting trades between teams, changing the starting lineup for each weekly matchup against a player, dropping players from their team, adding undrafted players onto their team. At the end of each week, each player's "points" will be summed up to get a total score for a team. Whichever team has the most points in a head-to-head matchup that week, will win the matchup.

League Functionality:
The purpose of each Fantasy League instance is to congregate all users that have registered for the league and their corresponding teams. The main responsibilities of the Fantasy League include: scheduling games between Fantasy Teams over the season, facilitation of trades, hosting the Fantasy draft (where each user will choose their players), maintaining team records and rankings in the League and creation and management of the playoffs

Player Functionality:
The service will also be able to pull live data from APIs like the "balldontlie" API, which provides up-to-date statistics on players to determine how many fantasy points they score in a given week. The Fantasy points will be scored through the following rubric (as outlined on https://fantasydata.com/api/fantasy-scoring-system/nba):

- *Three Point Field Goals: 3 points*
- *Two Point Field Goals: 2 points*
- *Free Throws Made: 1 point*
- *Rebounds: 1.2 points*
- *Assists: 1.5 points*
- *Blocked Shots: 2 points*
- *Steals: 2 points*
- *Turnovers: -1 points*

The “balldontlie” API will also allow our service to provide historical statistical averages for players to that users may predict how many points might be expected by a given player.

2. Who or what will be its users? What might they use the functionality for?

The intended users for our service are developers who want to build a Fantasy League application. Our service will provide all of the logic necessary to run the Fantasy League and our users will be able to focus on developing the frontend capability of the web/mobile application that will be making the API calls to the backend. Ultimately, basketball fans would be the users of this service, but the direct user would be the frontend client code that calls the API.

3. What kind of data will your service create or accumulate? What will the data be used for?

As explained earlier, the service will be implementing four main functionalities: User functionality, League functionality, Team functionality, and Player functionality. The data that will be created or accumulated for our service will be the data that is necessary to run all of the service logic necessary to run a Fantasy League. The following are examples of what will be stored for each corresponding functionality:

User Functionality:
- User’s email and Fantasy League username
- User’s first name and last name

League Functionality:
- Users and their corresponding teams registered for the League.
- Schedule of when teams are playing
- Rankings
- Proposed trades
- Playoff schedule

Team Functionality:
- The user that “owns” the team
- Players that have been drafted to this team
- Starting lineup for upcoming matchup
- Points generated each week

Player Functionality:
- Points that the player has scored
- Running list of active players (once teams are drafted, must also keep track of “free agents”)
- Whether the current player is active or not

Part 3: Write a few paragraphs that describe, at a high level, how you plan to test the functionality of your service without any clients, GUI or otherwise. (It's ok to use testing tools that have GUIs.) You will expand testing in the second iteration to include sample clients, but need to test stand-alone during the first iteration. Think in terms of testing your service as a whole, in addition to unit testing of individual subroutines, and answer these three questions:

1. How will you test that your service does what it is supposed to do and provides the intended functionality?

Testing the functionality of our service brings a few important topics to light. Each is expanded below:

- *Season data*
- *Mock Fantasy Basketball League*
- *"Dummy" front-end*

NBA Season Data

A critical dependency for our Fantasy Basketball League service is temporal data of a "live" basketball season, during which real players accumulate points during games over time. This is critical because corresponding player performance is the only way players / teams / users in the Fantasy Basketball League may accumulate points as well. It would be nonsensical to wait for a live NBA season in order to test our game / reenact point accumulation and proper database interfacing. To this end, our group will utilize data from past basketball seasons. (This is available on "balldontlie".) This will be used to ensure proper data transmission and querying from the "live" season data, to test our database-to-service endpoints. (In phase two, we may devise a script to update the mock / past season database over time with player performance statistics from "new" games, to ensure proper handling of "live" updates in season data.)

Mock Fantasy Basketball League

The Fantasy Basketball League is trivial without active users, teams, etc. Thus, we will create mock users, mock teams, and a mock drafting schedule when writing unittests for much of our service logic. Each modular function that we write in order to oversee user registration / user decisions throughout the League will be tested with a mock Fantasy Basketball League, team, players, etc.

"Dummy" Front-end

As answered in previous questions above, the intended users for our service are developers who want to build a Fantasy League application. Our service will provide all of the logic necessary to run the Fantasy League, and our users will be able to focus on developing the frontend capability of the web/mobile application that will be making the API calls to the backend. Thus, during the final phases of testing, we will develop our own "dummy" front-end in order to ensure that the proper API calls send useful data in order to eventually develop a complete front-end for our users' full-fledged Basketball Fantasy League application.

2. How will you check that your service does not behave badly if its clients use it in unintended ways or provide invalid inputs? How will you test that your service handles its data the way it's supposed to?

For each of the three aforementioned entities required to test the functionality of service in full, we will be sure to create at least one "healthy" version, as well as one "unhealthy" version of each of the dependencies / interactions as well.

<u>NBA Season Data</u>
We will test our service with a "healthy" NBA statistics database, as well as an unhealthy one.

<u>Mock Fantasy Basketball League</u>
When writing unittests for our user registration / user decision functions, we will create fail tests as well, to handle errors safely when users behave unpredictably while interfacing with our service.

<u>"Dummy" Front-end</u>
Upon creating our "dummy" front-end, we will be sure to test the functionality of our service upon ill-formed / problematic front-end requests as well during testing.

Note: Our Fantasy Basketball League service meets the following requirements:

1. Maintains relevant data persistently: User data, team data, player data, and league data will be persistently maintained throughout seasons, and maintained consistently throughout matches. Moreover, season data regarding player performance will also be maintained by our service.
2. Supports multiple clients: A Fantasy Basketball League consists of multiple users managing multiple teams. Thus, the nature of the service inherently is intended for multiple clients.
3. No GUI: The intended users for our service are developers who want to build a Fantasy League application. Our service will provide all of the logic necessary to run the Fantasy League, and our users will be able to focus on developing the frontend capability of the web/mobile application that will be making the API calls to the backend. We will not develop a GUI at all.