Part 1: State the date on which you met with your IA mentor. This should be in the past, not the future.

October 23, 2021, Saturday at 4PM.

Part 2: Reconsider your answers to the part 2 questions in the preliminary proposal. Elaborate in more detail the main features your service will provide.

Before explaining the main features we want to distinguish between two topics that may be confused with each other: a User and a Client. A User refers to someone who participates in the Fantasy Basketball league and plays the game, users are completely separated from development and are only familiar with playing the game. A client is an entity that will set up a frontend to utilize our Fantasy Basketball backend service to create a UI and playable game for users. (Moreover, the term Player—as distinguished from User and Client—refers to an NBA player.)

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 allow users to run their 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. It is intended that our backend service will have the capability to serve multiple frontend clients, so that each client will have separate data from each other.

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 (or play) in a Fantasy League, they must first have an account registered with our backend 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 user. Once a user is assigned to a Fantasy League, they may create their own Fantasy Team to participate in that instance of a Fantasy League. Each user in a Fantasy League will have their own Fantasy Team. Once Fantasy Players have been assigned to their respective Fantasy Teams through a draft, 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 of a team's structure

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 Fantasy 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, 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.

(Note: Only in the following paragraph, "users" are different from Fantasy Users described above, this addresses a question posed for the project proposal.)

The intended "users" for our service are the clients (or frontend developers) who want to build a Fantasy League application. Our service will provide all of the logic necessary to run the Fantasy League and clients 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, the direct "user" would be the frontend client code that will leverage our backend server via RESTful API calls.

As explained earlier, the service will be implementing four main functionalities:

- User functionality
- League functionality
- Team functionality
- 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
- Team's record (wins and losses)

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
- Player's history of Fantasy points for the current season

Part 3:

Reconsider your answers to the part 3 questions in the preliminary proposal. Elaborate in more detail how you will test the main features of your service from the external client perspective.

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 Fantasy Basketball League Users accumulate points during games over time.This is because Fantasy points are entirely calculated based the real statistics of an NBA player's performance. 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.) We will leverage Postman as a technology when testing our REST APIs.

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 unit tests for much of our service logic. Each modular function that we write in order to oversee User registration and User decisions throughout the League will be tested with a mock Fantasy Basketball League, User, Team, Players, etc. We will leverage Postman as a technology when testing this portion. (An important topic to address during these mock tests is to also consider proper compartmentalization / separation of league data by Client. Each Client that our service supports, trusts that our Fantasy Basketball League backend service will defend the integrity of the league information of their—each Clients'—users.) When writing unit tests 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.

"Dummy" Front-end

As answered in previous questions above, the intended audience for our service are known as Clients—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 Clients 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 for clients to eventually develop a complete front-end for our users' full-fledged Basketball Fantasy League application. We will leverage Postman as a technology when testing our REST APIs. 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.

Part 4: List the libraries, frameworks, tools, etc. you think you might use. Make sure to include (at least) a style checker, static analysis bug finder, test runner and coverage tracker suitable for your language and platform. You will not be held to this list.

- Style Checker: Checkstyle
  - https://checkstyle.sourceforge.io/
- Static analysis Bug Finder: FindBugs
  - http://findbugs.sourceforge.net/
- Test Runner: JUnit4
  - https://junit.org/junit4/
- Coverage Tracker: IntelliJ IDEA Coverage runner

    - https://www.jetbrains.com/help/idea/running-test-with-coverage.html#coverage-run-configurations
- Relational Data Access: Spring Boot JDBC
    - https://www.javatpoint.com/spring-boot-jdbc
- IDE: IntelliJ
    - https://www.jetbrains.com/idea/
- Java based framework: Spring Boot
    - https://spring.io/projects/spring-boot
- AWS resources: RDS or DynamoDB
- Testing: PostmanAPI
- Test Framework: Mockito
    - https://site.mockito.org/