**Group 6 Project 2 Report (11/29/2022)**

**Hayden Rieder, Jacob Green, Nathan Haileyesus, Ali Ibrahim**

For this project, our team decided to create a web-app that can quickly and efficiently compare and contrast the sports statistics of two players from the National Basketball Association (NBA) to see who is the better player in different categories. The idea for this project came to us after a friendly debate among the group following one lecture. One member believed Trae Young from the Atlanta Hawks to be a better player than Ja Morant from the Memphis Grizzlies, and another member of the group did not agree. They both went into research on the seasonal statistics of the two players and came up with ambiguous and biased data to show the narrative they wanted, and the argument was never settled. As a group of sports fans, we wanted to find a quick and easy way to compare the players’ stats without heaps of data being shown. There have been other studies attempting to do the same thing. A 2016 article from the Journal of Sports Analytics attempted to look into more optimal ways to track NBA player data and comparisons and stated that nowadays, the amount of information is beginning to pose challenges for coaches and analysts (1). They came up with a method of creating principal component scores and statistical diversity indices (SDI) to create multiple use cases for showing a player’s strengths and weaknesses for team management and negotiating salaries and trades of players. Our team is not attempting to create some groundbreaking researching in this field, but rather we want to create the simplest platform possible to correctly display the direct comparison of two players.

Once we settled on this topic, we began planning our design. We were given few strict requirements, so a lot of time went into brainstorming and planning. The goal was to create a web-app in which a user can easily see the statistics of two players and who leads the other player in those categories and overall. Functional requirements for the web-app would include features such as correct player retrieval and visible and correct statistics. Non-functional requirements would include features such as fast data retrieval and easy to use UI. We decided to model our system following an Agile architecture with an emphasis on test-driven development at each cycle of development. Hayden was the project manager and took lead on the backend. Jacob took the lead on the frontend and also served as a QA lead. Nathan headed up the design and requirements lead and also took lead on the presentation/report aspect of the project. Ali was the primary tester and assisted with styling.

Our team decided to use JavaScript for both the frontend and backend. We built out our frontend using the React framework, with styling in vanilla CSS, since our team was familiar with React following our previous project, and we wanted to maintain a good baseline to create a simple and easy user interface. On top of these features, we used the Axios library for easier interaction with the backend REST API. As for the backend, we used Express.js in JavaScript in a “Node.js” runtime environment. We used Express to set up a proper REST API that interacts with external APIs. Along with Express for the backend, we used libraries such as “Nodemon” and “Concurrently” to improve the development experience by providing fast-reloading and the ability to concurrently run the backend and frontend using just one command. Most of the testing on the backend was done using Jest with additional testing on the frontend using Cypress. Finally, we used Git/GitHub for version control among the team. Below are a couple diagrams, both high-level and low-level, demonstrating the design of our application.

Letter

Description automatically generated

Figure : High-level Flow Diagram

Diagram

Description automatically generated

Figure : Class Diagram

Chart, box and whisker chart

Description automatically generated

Figure : Sequence Diagram

After finishing our design, we jumped right into our test-driven development approach to the implementation, where test cases were written at each step of the process rather than at the end. This ensured us that we were maintaining and operatable platform during the entire process. We made sure to incorporate both blackbox testing and whitebox testing. Some of our test cases included correct splitting of first and last name, cases where there is no player data (e.g. out for injury), correct calculations for seasonal aggregation, etc. Some faults appeared when dividing total points of a player by the number of games they played if the player had not played in that season, so we had to account for that with a check to see if it was non-zero. Once we had a fully fleshed out backend with a compatible frontend, we ensured we had a branch and statement coverage of 90% to finish testing.

Text

Description automatically generated

Text

Description automatically generated

Below is a screenshot of what the current UI looks like, as well as an alternate more visual UI we designed.

Graphical user interface

Description automatically generated

Figure : Current UI

Graphical user interface

Description automatically generated

Figure : Alternate UI