Project Specification for Group TUT0301-06

Team Name: NBA Wizz

Domain:

Our project domain is Sports Analytics, focusing on NBA player and team insights. We will create an interactive platform that allows users to look up, compare, and analyze player and team data. Features include player lookup and filters, team comparisons, performance graphs over time, and AI-generated insights or stat predictions. We will use NBA datasets from Kaggle and integrate a sports data API along with a free LLM API (e.g., LangChain) for insights. We chose this domain because it combines our interest in sports and data analytics, providing opportunities for visualization, comparison, and AI-driven predictions.

User Stories:

User Story 1: As a user, I want to search for a specific NBA player in a search bar so that I can quickly view their career statistics and performance graphs.

User Story 2: As a user, I want to filter NBA players by team, position, and season range so that I can focus on specific groups of players using dropdown menus for team and position and a slider for selecting the season range.

User Story 3: As a user, I want to compare two or more players or teams so that I can analyze their strengths and weaknesses side by side.

User Story 4: As a user, I want to favourite or bookmark players/teams so that I can revisit their stats quickly.

User Story 8: As a user, I want to unfavourite players/team in case I no longer want the player/team on my favourites.

User Story 5: As a user, I want to sort player stats (e.g., by points per game) so that I can identify top performers easily.

User Story 6: As a casual user, I want the AI system to generate insights or summaries about player or team performance so that I can understand complex stats more easily.

User Story 7: As a sports enthusiast, I want to see AI-based predictions for player stats or team performance so that I can get an idea of future outcomes based on past data.

Use Cases:

Use Case 1: Searching for players

Main flow:

1. The user enters the name of an NBA player in the search bar which also offers autofilling capabilities.
2. The user enters the seasons in years (e.g., 2022-2025) and chooses the specific statistics they wish to view from a drop-down menu (e.g., points, assists, rebounds).
3. The system retrieves and aggregates the player's data from the external dataset.
4. The system displays the results in a table format, organized by year, and generates performance graphs for each selected statistic using the JavaFX Framework.

Alternative flow:

1. If the user enters a player name that does not exist, the system displays an error message prompting the user to re-enter a valid player name.
2. If the user enters a season before 1980 or after 2025, the system prompts the user to input a valid range.
3. If data for some or all of the selected statistics or seasons is unavailable, the system displays whatever information is available and notifies the user of the missing data with an error message appearing on screen.

Use Case 2: Filtering players by information

Main flow:

1. The user opens the Players page with the full player list displayed.
2. The user opens the filter panel.
3. The user selects one or more filter criteria (e.g., team(s), position(s), season(s):
4. The system validates the selections and creates badges for them to display above player list (e.g., valid season format, recognized team/position codes).
5. The user clicks Apply Filters.
6. The system queries the dataset using the selected filters.
7. The system returns the filtered results and updates the player list.
8. The system displays the active filters as badges above the results.
9. The user may refine filters (edit, add, remove) and re-apply to update the list.
10. The user can click Clear Filters to reset to the full player list.

Alternative flow:

11. IIf the user selects a filter option that is unavailable (e.g., no players exist for that team, position, or season), the system displays a message such as “No players match your filters. Adjust filters and try again.” and provides a Clear Filters button to reset.
12. If the system detects that a selected filter combination would produce no results (for example, choosing a team that didn’t exist in the selected season), the system immediately shows a non-blocking warning message like “No data available for this combination.”
13. If the system cannot retrieve data from the external source (e.g., API timeout or partial data), the system displays the available player information and a notice: “Some data could not be loaded.”
14. If the data request fails completely, the system shows an error message such as “Unable to load results. Please try again.” and retains the current filter selections so the user can retry.
15. If the selected filters return a very large dataset, the system paginates or limits the displayed players, showing a summary like “Displaying first 100 of 1,200 players.”

Use Case 3: Compare players/teams side-by-side

Main flow:

1. The user opens the “Compare” page in the app.
2. The user selects whether they want to compare teams or players.
3. The user selects at least two and up to five players or teams from the list.
4. The user selects a time range(e.g., a specific season).
5. The system collects the chosen data from the dataset.
6. The system organizes the statistics for comparison.
7. The app shows the result in a side-by-side table, a scroll bar is used to fit the size.
8. The user can change the players or teams to update the comparison.

Alternative flow:

1. If the user selects players or teams with a number fewer than 2, the system shows a warning and asks the users to select more.
2. If some stats are missing, the app still shows the chart with available data and marks the missing part for that specific team with an “\” mark.
3. If data from different seasons can’t be compared (e.g., no data for one team for that season), the system suggests comparing overlapping years.

Use Case 4 + 8: Favourite Toggle Players or Teams

Main flow:

9. The user navigates to a player or team profile page or views the search results list.
10. The user clicks the “favourite” or bookmark icon next to a player or team, where the icon is then toggled on.
11. The system saves the selected player or team to the user’s favourites list.
12. The user navigates to the “Filter” section in the app and filters by favourites, or non-favourites

Alternative flow:

4. If the user favourites a person twice, the person is unfavourited. This is indicated to the user because the favourited button is toggleable.
5. If the user has not favourited anyone, the system displays a message informing the user that no favourites exist and may suggest browsing or searching for players/teams to add.

Use Case 5: Sorting player stats

Main flow:

---

13. The user clicks the “Filter and Sort” option from the main menu and the page shows a list/table of players.
14. The user clicks a column header in the list (e.g., PPG, APG, RPG, FG%) to sort by that field (descending by default).
15. The system sorts the list according to the selection and shows an arrow ↑/↓ to indicate the current order.
16. The user can toggle the order by clicking again.

Alternative flow:

---

6. If some players are missing the selected stat, the system still completes sorting and places missing values at the end (for descending) or at the beginning (for ascending).
7. If the selected field is non-numeric (e.g., position), the system uses lexicographic ordering.

Use Case 6 + 7: LLM-Powered Insight Backend System:

To address user stories 6 and 7, we will build an LLM-powered backend pipeline that uses our collected data to generate AI insights about player or team performances, dynamics, and predictions on future outcomes.

Main Flow:

---

1. The user selects a player or team from the UI and clicks “Generate Insights.”
2. UI Controller receives the request and sends it to the GenerateInsightsInteractor.
3. Interactor requests relevant historical performance data (stats, trends) from the Data Access Layer.
4. The Interactor formats the data into a prompt template for the LLM (e.g., “Summarize these stats in simple terms…”).
5. LLM API receives the prompt and generates a summary if the prompt is relevant and sensical. If the prompt is deemed as nonsensical or out of context, then the LLM will give a pre-defined response asking the user to ask a question that makes more sense.
6. Interactor receives the summary, wraps it in an output data structure, and passes it to the Presenter.
7. Presenter formats the summary for the View.
8. View displays the insight to the user.

Alternative Flows

Not Enough Data to Predict:
1. The player has too few games or missing data.
2. System uses simplified prediction (or none).
3. Presenter informs user (“Not enough data for a reliable prediction.”)

Prediction Computation Error

1. Data unexpectedly inconsistent (e.g., NaN values).
2. Interactor returns an error report.

LLM Response Failure

1. Prediction is generated, but LLM explanation fails.
2. Presenter displays numeric predictions with no natural-language explanation.

MVP:

Lead	Use Case	User Story
Mark Wang	Use Case 6 + 7: LLM-Powered Insight Backend System	User Story 6 + User Story 7
Ibraheem Hussain	Use case 1: Searching for players	User Story 1
Wilson Liang	Use Case 5: Sorting player stats	User Story 5
Yaohui Huang	Use case 3: Comparing players/teams side-by-side	User Story 3
Parsa Hemmati	Use case 4: Favourite toggle Players or Teams	User story 4 + User story 8
Andrej Prekajski	Use case 2: Filtering players by information	User story 2
Name of the Lead for Use Case #n	Use Case #n	User Story #m

Proposed Entities for the Domain:

Core entities include Player, Team, Game/Season Stats, and AI Insights. The main actions involve searching, filtering, comparing, visualizing, and predicting, which together create an engaging data-driven experience for NBA fans and analysts.

Entities:

Player

- **playerID:** int - unique identifier for each player
- **name:** String - full name of the player
- **team:** Team - current team (association)
- **position:** String - e.g., “Guard”, “Forward”, “Center”
- **age:** int - player’s age
- **height:** double - height in meters or inches
- **weight:** double - weight in kilograms or pounds
- **careerStats:** List<SeasonStats> - all season statistics
- **favouritesCount:** int - number of users who have favourited this player

---

Team

- **teamID:** int - unique identifier for each team
- **name:** String - team name
- **city:** String - city or location of the team
- **players:** List<Player> - roster of players
- **wins:** int - total wins in current season
- **losses:** int - total losses in current season
- **conference:** String - “Eastern” or “Western” conference
- **teamStats:** Map<String, Double> - aggregated team performance metrics

SeasonStats

- **seasonYear:** int - year of the season
- **pointsPerGame:** double - average points per game
- **assistsPerGame:** double - average assists per game
- **reboundsPerGame:** double - average rebounds per game
- **fieldGoalPercentage:** double - shooting efficiency
- **games Played:** int - total games played in the season
- **minutesPerGame:** double - average minutes played per game
- **player:** Player-associated player
- **threePointPercentage:** double - efficiency beyond the 3-point line

AIInsight

- **id:** int - unique identifier for each generated insight
- **entityType:** String - “Player” or “Team”
- **entityName:** String - name of the player or team the insight is about
- **summaryText:** String - AI-generated summary of performance
- **prediction:** Map<String, Double> - predicted metrics (e.g., future PPG, rebounds)
- **timestamp:** LocalDateTime - time when the insight was generated
- **confidenceScore:** double - reliability score for the AI prediction

Filter

- **filterID:** int - unique identifier for each filter instance
- **teamName:** String - team filter (optional)
- **position:** String - position filter (e.g., “Guard”, “Forward”)

- **minPoints:** double - minimum points per game threshold
- **maxPoints:** double - maximum points per game threshold
- **seasonYear:** int - filter by season
- **sortBy:** String - which stat to sort by (e.g., "PPG", "APG")
- **sortOrder:** String - "ascending" or "descending"

Proposed API for the project:

We are going to use the Ollama4j (https://ollama4j.github.io/ollama4j/) API, which allows for the interaction with Ollama in Java. Ollama is a tool that enables developers to run large language models (LLMs) locally for free without cost. For our project, we will be running open-source LLMs using Ollama4j as part of the AI backend of our project to provide insights and predictions for NBA players and teams. More specifically, we are currently looking to use lightweight Mistral models for faster inference.

Datasets:

1. https://www.kaggle.com/datasets/eoinamoore/historical-nba-data-and-player-box-scores/
2. https://www.kaggle.com/datasets/justinas/nba-players-data
3. https://www.kaggle.com/datasets/wyattowalsh/basketball

Notes:

- Domain is fine
- User stories
    - "As a sports fan", "As a basketball analyst" etc can be replaced with "As a a user" Or implement separate roles
    - User story 3: Need scroll bar to fit everything
    - User story 4: Add unfavourite as another user story
    - User story 5: Where can you sort? (Same place as filtering ?) List the attributes we want to filter/sort by.
    - User story 6: Control what the user can enter.
    - User story 7: Predict what?
- What's the timeframe? Make sure to process the data.
- How to handle null data? How to display errors? (Popups)
- Use cases

    - Case 1: Main flow. The view has buttons each representing the features. Think about UX. What do we mean graphs for each statistic? Do we need a lib? Implement out of field in the search bar. Alternative flow.
    - Case 2: Main flow. Make sure to paginate, or cut the number of pages if too much. Be specific what you can filter by.
    - Case 5: Main flow. In memory? Alternative flow. Tied?
    - Case 6+7: Main flow. Remove freedom from the user to ask anything.
- Errors can also be banners, or a text that just pops up.
- Add unfavourite
- Proposed entities
    - Add a filter entity
    - Playerlist entity rather than just an ArrayList
- Make sure to test API and other libraries.