Project 4 - Mobile to Cloud application

By Han Bao (AndrewID: hanb)

This project is a distributed application I designed to provide NBA information to users through a native Android application that connects to a web service. The application enables users to search for NBA team details and game information using a simple and user-friendly interface.
Here's how the project is structured:
Android Application

- The Android app allows users to input a team name or select a specific date to fetch NBA-related data.
- Results, such as team details and game results, are displayed in a responsive and visually clear layout.
- Errors are handled gracefully, and users receive immediate feedback through Toast messages if something goes wrong, like invalid inputs or network issues.

Web Service

- I implemented a servlet-based web service using TomEE, which is also connected to a MongoDB database for logging and storing relevant data.
- The web service processes HTTP requests from the Android app, fetches data from the NBA API, and returns filtered JSON results that are easy for the app to display.
- The service logs critical information such as user interactions, API response times, and request details, making it easier to analyze usage patterns and debug issues.

Third-Party Integration

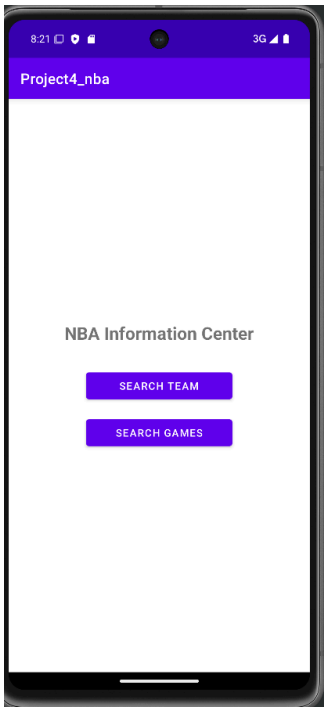
- The app integrates with the RapidAPI NBA API to fetch real-time NBA team and game data.
- API responses are processed and formatted on the server side before being sent to the Android app, ensuring the mobile app only processes the data it needs.

Logging and Analytics Dashboard

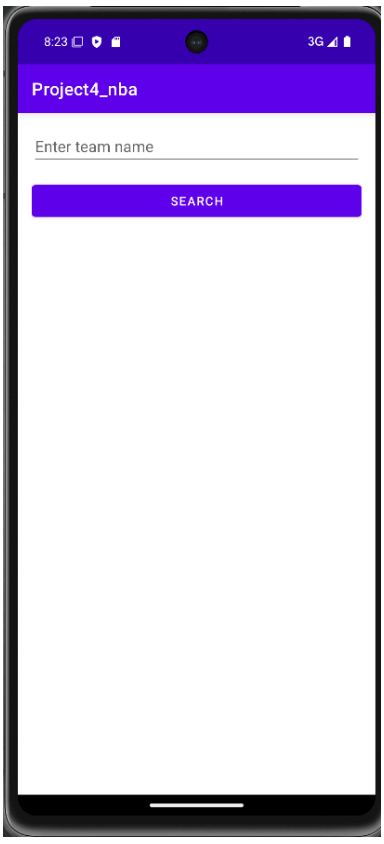
- I created a web-based dashboard to provide detailed insights into how the web service is used.
- The dashboard displays formatted logs of user interactions and analytics, such as the most frequently searched teams, average API latency, and the volume of game searches on specific dates.

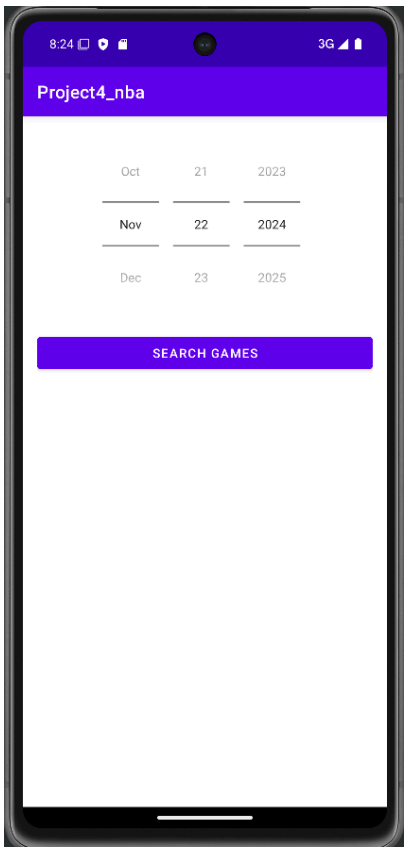
Requirement 1: Native Android Application

1. Three Different Views in Layout
    - The Android application uses the following views:
        - TextView: Displays titles and informational text (e.g., "NBA Information Center" on the homepage).
        - EditText: Allows users to input team names for searching.
        - Button: Allows users to submit search requests (e.g., "Search Team" and "Search Games").
        - RecyclerView (Bonus): Displays search results for teams and games in a scrollable list.



2. User Input
    - Users provide input through:
        - An EditText field to enter a team name for the "Search Team" feature.
        - A date picker for the "Search Games" feature.

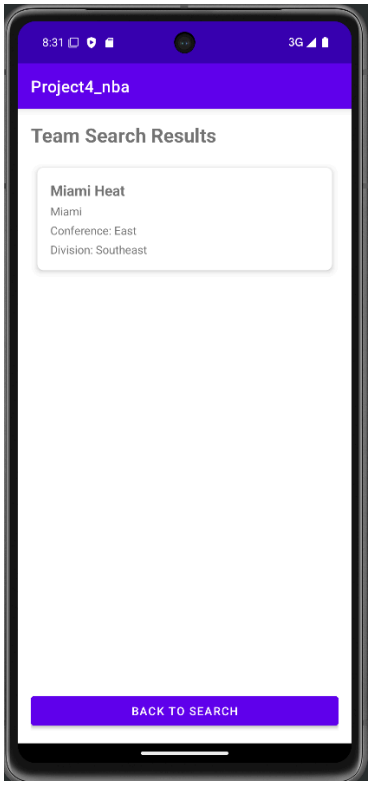




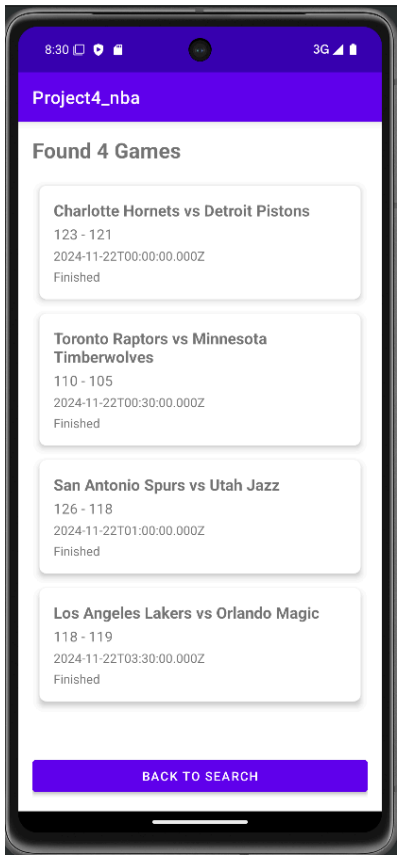
3. HTTP Request to Web Service
    - The app makes HTTP GET requests to the deployed web service:
        - Team Search: Sends the team name to the /nbaData endpoint with type=team.
        - Game Search: Sends the selected date to the /nbaData endpoint with type=games.
4. JSON Parsing
    - The Android app parses the JSON response from the web service using the GSON library.
    - For Team Search, the app extracts team details such as name, nickname, city, and league information.
    - For Game Search, the app extracts game details such as home/away teams, scores, and game status.
5. Displaying New Information
    - Search results are displayed in a RecyclerView:
        - Team search results show the team name, nickname, city, and league information.
        - Game search results show the home/away team names, scores, and game status.
6. Repeatable
    - The app allows users to perform multiple searches without restarting the application. Users can return to the homepage and switch between "Search Team" and "Search Games" features.

Requirement 2: Web Service

1. API Implementation
    - A servlet-based web service is implemented using TomEE and deployed to GitHub Codespaces.
    - Endpoints:
        - /nbaData?type=team&name=<teamName>: Fetches team details from the API.
        - /nbaData?type=games&date=<date>: Fetches game details for the specified date.
2. Receives HTTP Requests
    - The web service receives requests from the Android app via HTTP GET. Query parameters are validated before being processed.
3. Business Logic



- For Team Search:
  - Queries the third-party NBA API for team details using the provided team name.
  - Filters and formats the JSON response to include only necessary fields (e.g., team name, nickname, city, league).
- For Game Search:
  - Queries the NBA API for game details using the provided date.
  - Filters and formats the JSON response to include only relevant fields (e.g., home/away team names, scores, and game status).
  -



4. Replies with JSON
   - The web service responds with a JSON object containing the formatted results for either teams or games.

Here is several link for API:
https://rapidapi.com/blog/nba-basketball-stats-api/
https://rapidapi.com/api-sports/api/api-nba

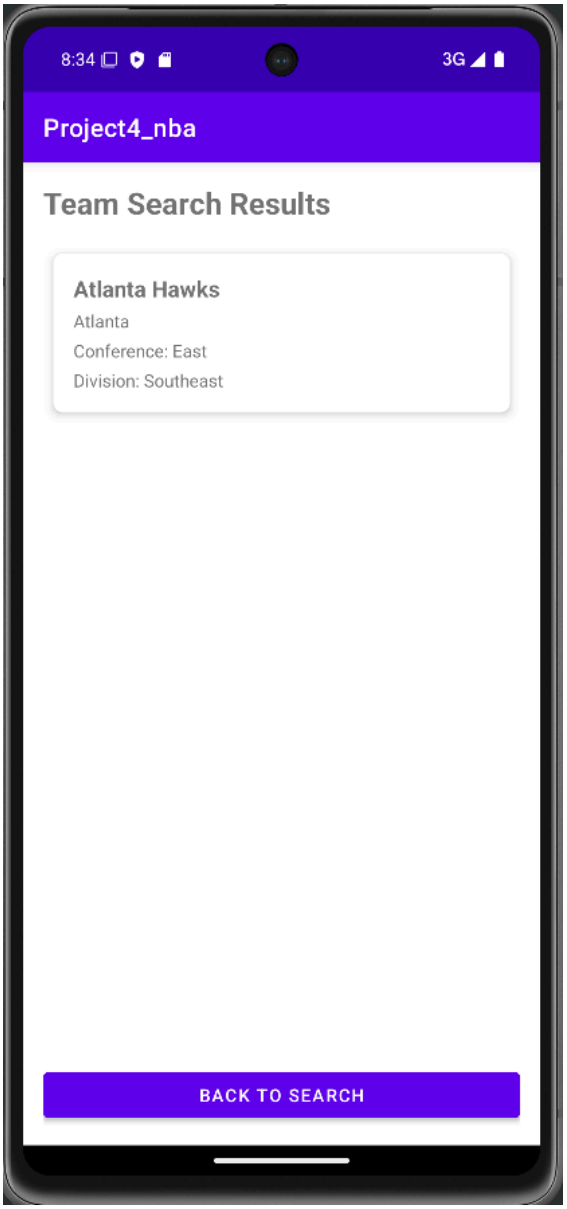
Requirement 3: Handling Error Conditions

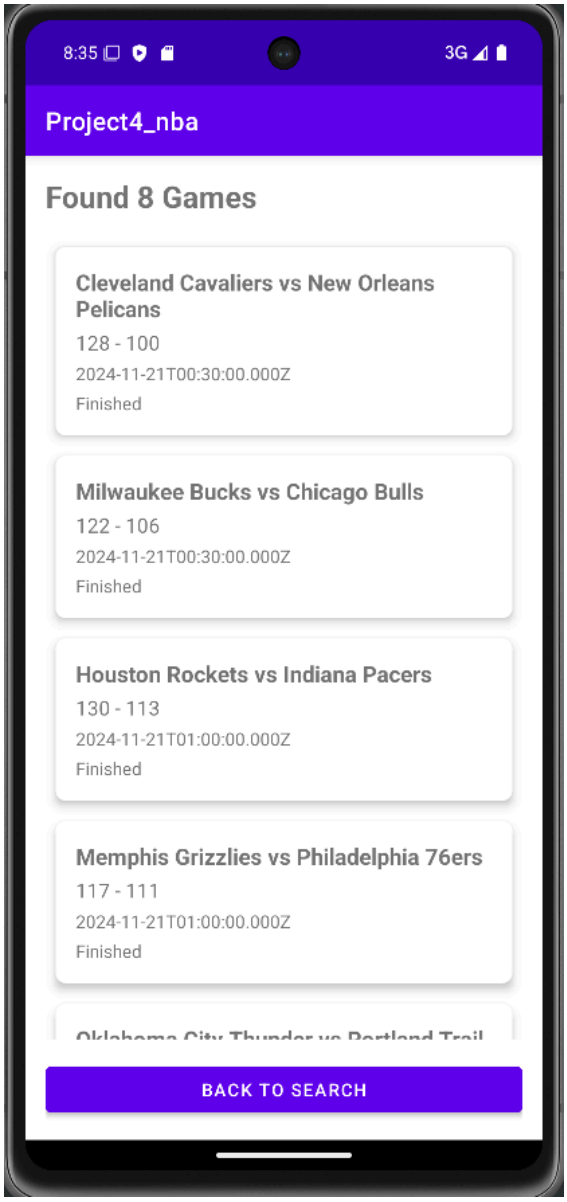
1. Mobile App Errors
   - Displays appropriate Toast messages for:
     - Empty input fields.
     - Network errors or server unavailability.
   - Logs errors in the Logcat for debugging.
2. Server-Side Errors
   - Validates input parameters and returns appropriate error messages (e.g., "Invalid type parameter").
   - Handles third-party API errors (e.g., 404, 500) gracefully and logs them.
3. Third-Party API Errors
   - Catches and logs exceptions for invalid or unavailable data from the NBA API.

Requirement 4: Logging Useful Information

The web service logs the following six pieces of information in MongoDB for each request:

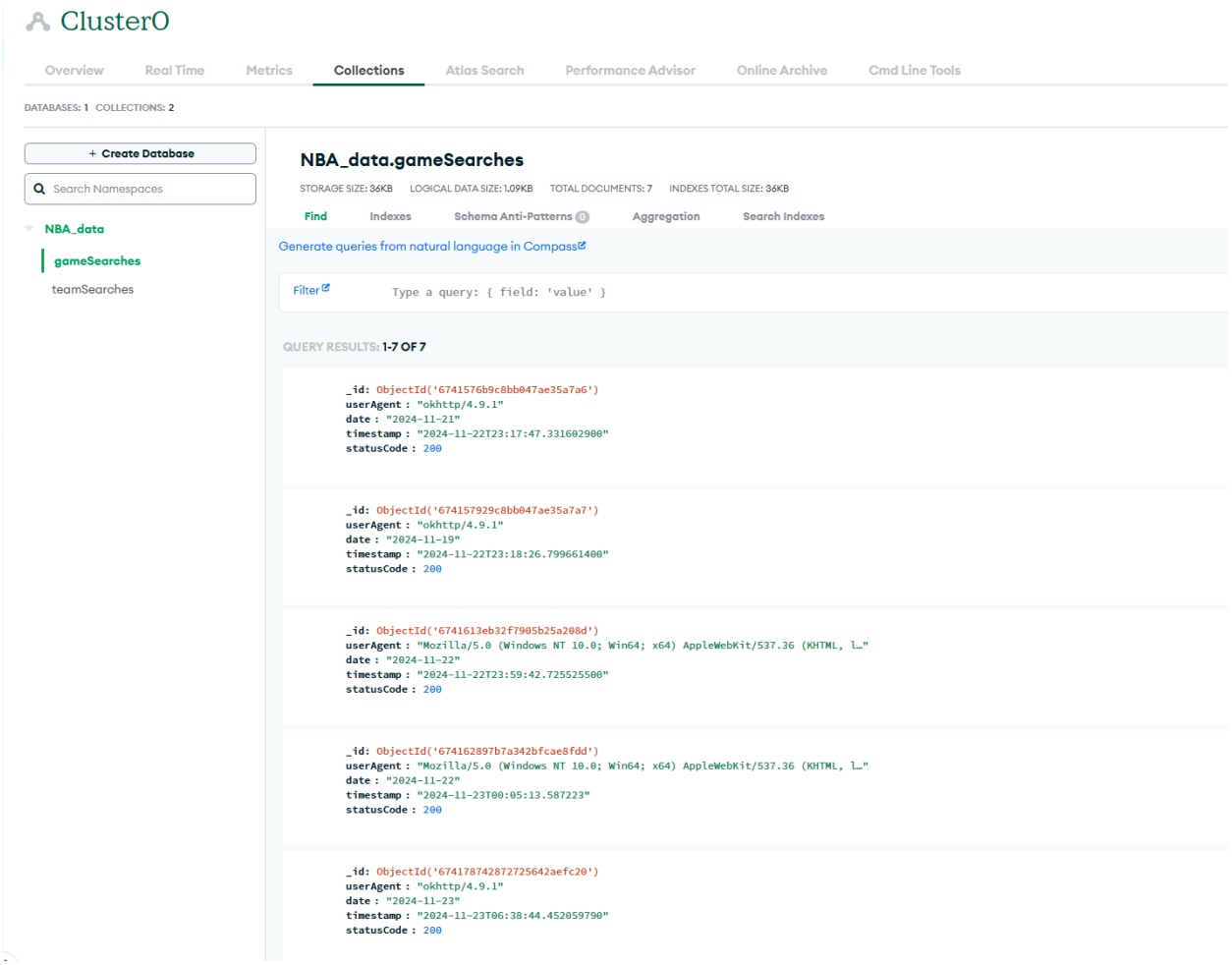
1. User-Agent of the mobile app making the request.
2. Request parameters (e.g., teamName or date).
3. Timestamps for when the request was received and processed.
4. Response status codes from the third-party NBA API.
5. Filtered data returned to the mobile app.
6. Response latency for the third-party API.





Requirement 5: Persistent Log Storage

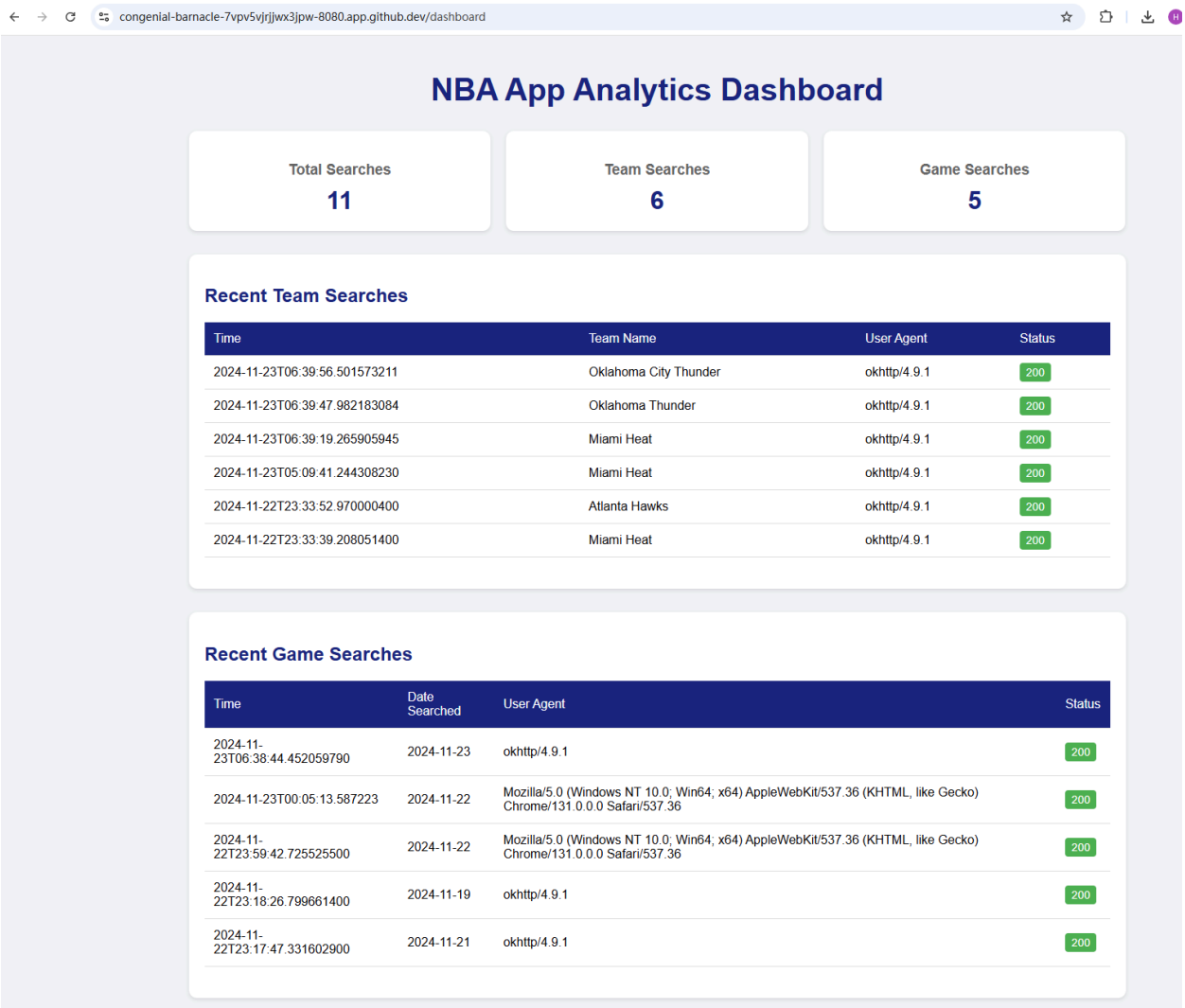
- Logs are stored persistently in a MongoDB database hosted on MongoDB Atlas.
- CRUD operations are performed programmatically using the MongoDB Java driver.



Requirement 6: Dashboard

1. **URL**
    - A unique URL (e.g., `https://congenial-barnacle-7vpv5vjrjjwx3jpw-8080.app.github.dev/dashboard`) addresses the operations dashboard.
2. **Analytics**
    - Displays three interesting analytics:
        - Top 6 most searched team names.

        - Top 6 most searched Game Days.
        - Number of game searches (separately or collectively) per day.
3. **Formatted Logs**
    - The dashboard displays all logs in a formatted HTML table using JSP. The table includes:
        - User-Agent, request parameters, timestamps, response status codes, and response times.



Requirement 7: Deployment to GitHub Codespaces

1. **Deployment**
    - The web service is deployed to GitHub Codespaces following the provided instructions.

(https://github.com/CMU-Heinz-95702/distributed-systems-project-04-hanb1252)
    - The application is packaged as a `ROOT.war` file and runs on TomEE.
2. **Testing**
    - The service URL is accessible from both the Android app and a browser.
    - The port visibility is set to "Public" to allow unauthenticated access.

How to Use the Project

1. **Setup**
    - Clone the repository from GitHub and open the Android app in Android Studio.
    - Open the web service in IntelliJ IDEA or another Java IDE.
2. **Run the Web Service**
    - Deploy the web service to TomEE on GitHub Codespaces.
    - Ensure the MongoDB Atlas connection is properly configured.
3. **Run the Android App**
    - Test the app using an Android emulator or a physical device.
    - Perform searches for teams and games using the app.
4. **View the Dashboard**
    - Access the dashboard at the provided URL to view logs and analytics.