CSC 431

Canes Focus

System Architecture Specification (SAS)

Group 12

Bradley Harmer	System Overview, System Diagram, Actor ID
Jeremiah Moise	<Role>
Kaphael Philogene	<Role>

Version History

Version	Date	Author(s)	Change Comments

Table of Contents

Table of Tables

<Generate table here>

Table of Figures

<Generate table here>

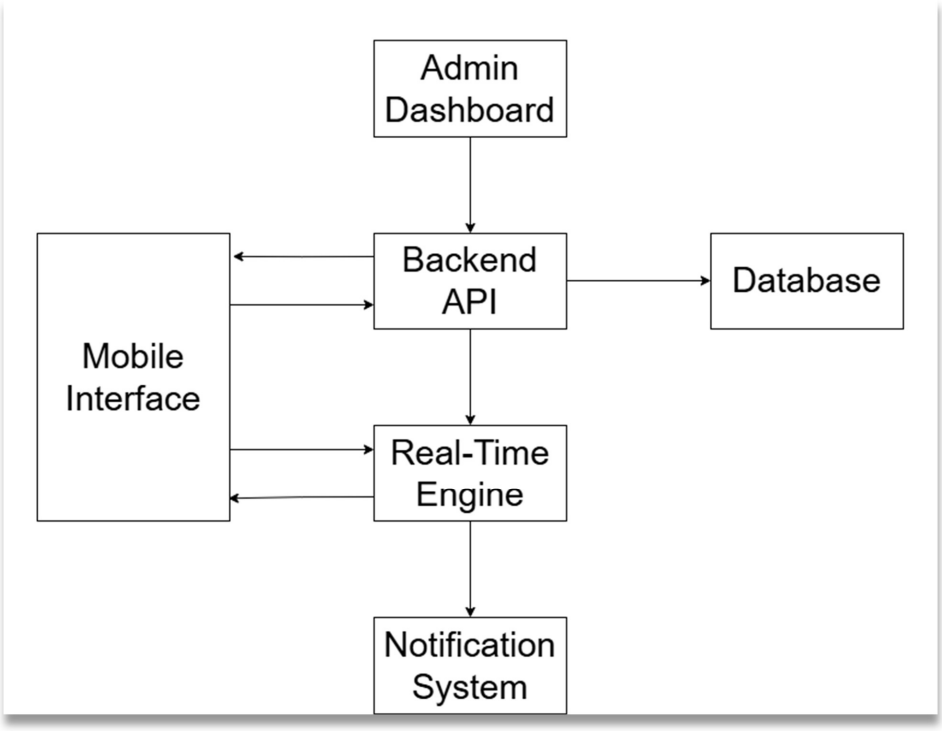
1. System Analysis

System Overview

Our System is a mobile app built around the idea of improving the experience for sports fans and student athletes at the University of Miami. Its core functions will include a calendar to see all upcoming sporting events and a large historical and current archive of sports data including scores, player statistics, and wins and losses. It will also include functionality for personalized push notifications and a way to purchase tickets within the app.

Our System will use a layered architecture with 3 tiers: frontend, backend, and database.

System Diagram



Actor Identification

1. **Students** – Regular users of the app. Will sign in using university credentials.
2. **Student-Athletes** – Same interactions as Students plus special privileges such as creating a biography for themselves, attaching their social medias, and interacting with fans
3. **System Administrators** – Official users who will be working to maintain the app and ensuring accurate data.
4. **Database** – Entering and updating all relevant scores and statistics as they become available.
5. **Notification System** – The notification software must interact with the backend to provide the notice to send the notification.

Design Rationale

a. Architectural Style

Architectural Style: 3-Tier Architecture
This project illustrates a 3-tier architecture, which separates the system into three independent layers:

1. **Presentation Tier (Frontend)**:
    a. Technology: **Flutter**
    b. Responsible for the user interface and user experience.
2. **Application Tier (Backend)**:
    a. Technology: **Node.js**
    b. Contains the business logic and handles client requests.
3. **Data Tier (Database)**:
    a. Technology: **Firebase**
    b. Manages data storage, retrieval, and persistence.

This architecture promotes **modularity**, **scalability**, and **ease of maintenance** by decoupling each responsibility into its own layer.

b. Design Pattern(s)

Design Pattern: MVC (Model-View-Controller)

This follows a 3-tier architecture pattern, used in frontend development to separate concerns and improve maintainability:

- **Model (Data Layer)**: Manages data like player stats, game results, and user profiles.
- **View (Presentation Layer)**: Built with Flutter, it displays dynamic content to users.
- **Controller (Logic Layer)**: Handles user input, business logic, and communicates with backend APIs.

By dividing responsibilities across these three components, the MVC pattern enhances scalability, modularity, and ease of testing.

Design Pattern: Client-Server

This follows a 2-tier architecture, separating responsibilities between the client and the server:

- **Client (Frontend)**: The mobile app acts as a client, sending API requests over HTTP(S) to the server.
- **Server (Backend)**: Processes logic, handles data validation, and securely stores data.

This model ensures scalability, secure communication, and a clear separation of concerns, making it ideal for mobile and web applications.

c. Framework

Framework: Flutter

Framework Used:

Flutter – An open-source UI toolkit by Google.

- Mobile UI Development: Flutter is used for building the mobile application's user interface with a focus on visual appeal and performance.
- Cross-Platform: Allows development for both Android and iOS using a single codebase, reducing development time and cost.
- Expressive & Customizable Widgets: Enables highly customizable and responsive UI designs, which enhance user experience.
- Performance: Delivers smooth animations and fast rendering, ideal for building sleek and engaging apps.
- Community & Support: Maintained by Google, with a strong community and frequent updates, ensuring long-term reliability.

Why Flutter?

Chosen for its productivity, modern UI capabilities, and cross-platform efficiency, making it ideal for rapid development of visually rich mobile apps.

Framework: Express.js

Framework Used:

Express.js – A minimalist web framework for Node.js.

- Backend Foundation: Acts as the core backend framework, built on top of Node.js to handle server-side logic.
- RESTful APIs: Ideal for building efficient and lightweight REST APIs to support frontend applications.
- Routing & Middleware: Simplifies the handling of HTTP requests, routing, and middleware integration.
- Real-Time Support: Leverages Node.js's event-driven nature, making it suitable for real-time applications.
- Language Compatibility: Easily integrates with JavaScript and TypeScript, making development faster and more flexible.

Why Express.js?

Chosen for its simplicity, speed, and scalability in building robust server-side applications. It's widely adopted, well-documented, and perfectly fits the needs of modern web and mobile backends.

Identified Database Technology: Firebase

Database Used:

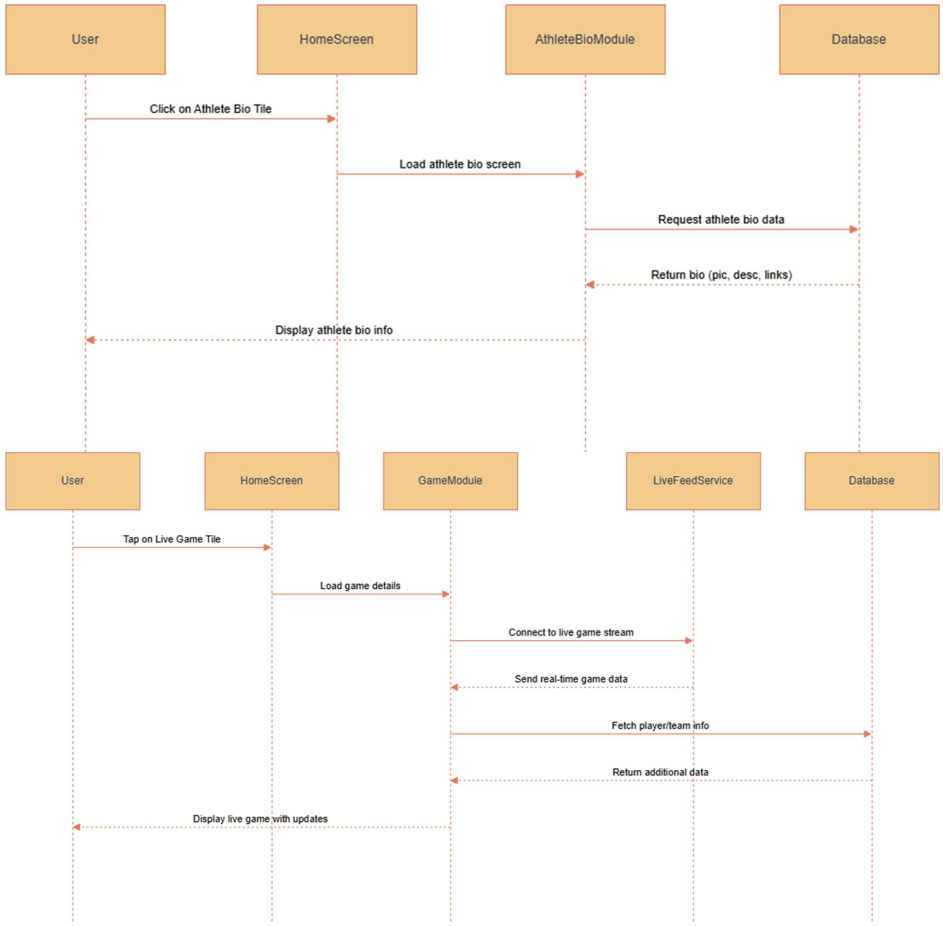
Firebase – A cloud-based Backend-as-a-Service (BaaS) platform by Google.

- Real-Time Database: Firebase provides real-time data synchronization, making it ideal for apps that need live updates like chats, leaderboards, or notifications.
- Integrated Services: Offers built-in support for authentication, cloud functions, and push notifications, reducing the need for separate services.
- Cross-Platform Integration: Seamlessly integrates with Flutter and Node.js, ensuring smooth data flow across the entire application stack.

- Scalability: Firebase offers scalable infrastructure, allowing the application to grow without heavy server management.
  Why Firebase?
  Chosen for its speed of development, real-time capabilities, and tight integration with frontend and backend technologies. It simplifies backend management while providing robust features for modern mobile and web apps.

2. Functional Design

<Identify all significant workflows as sequence diagrams using the following format>


User
HomeScreen
AthleteBioModule
Database
Click on Athlete Bio Tile
Load athlete bio screen
Request athlete bio data
Return bio (pic, desc, links)
Display athlete bio info
User
HomeScreen
GameModule
LiveFeedService
Database
Tap on Live Game Tile
Load game details
Connect to live game stream
Send real-time game data
Fetch player/team info
Return additional data
Display live game with updates

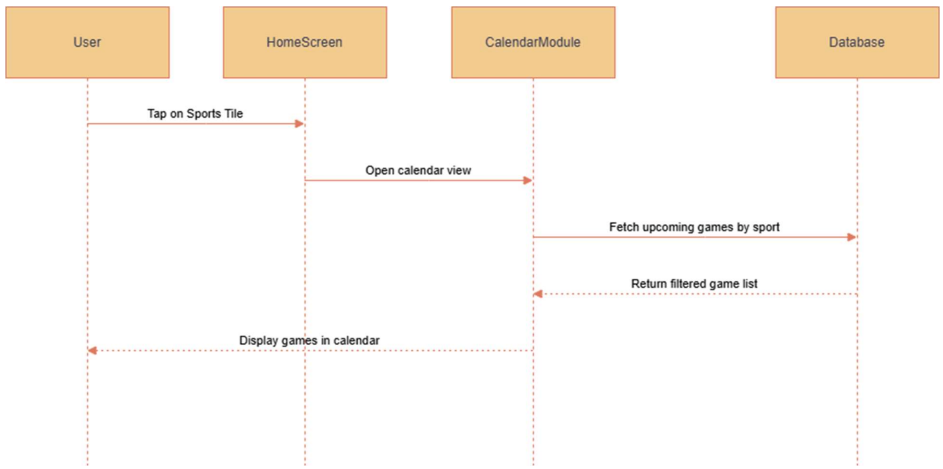
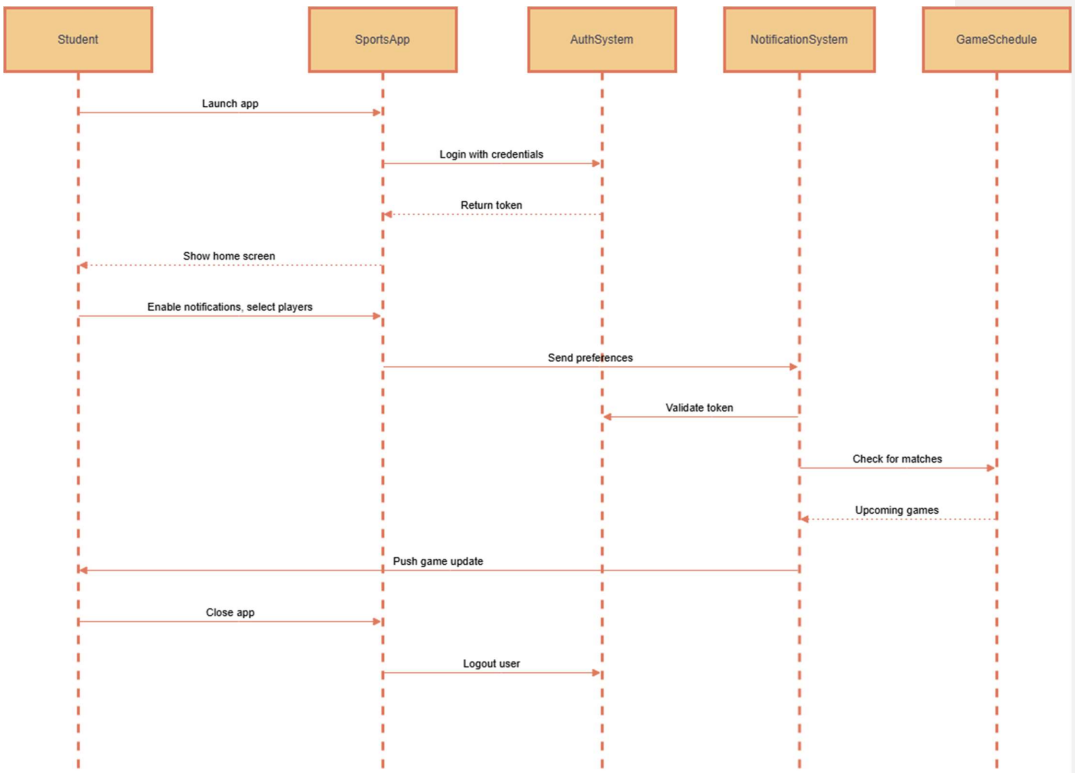
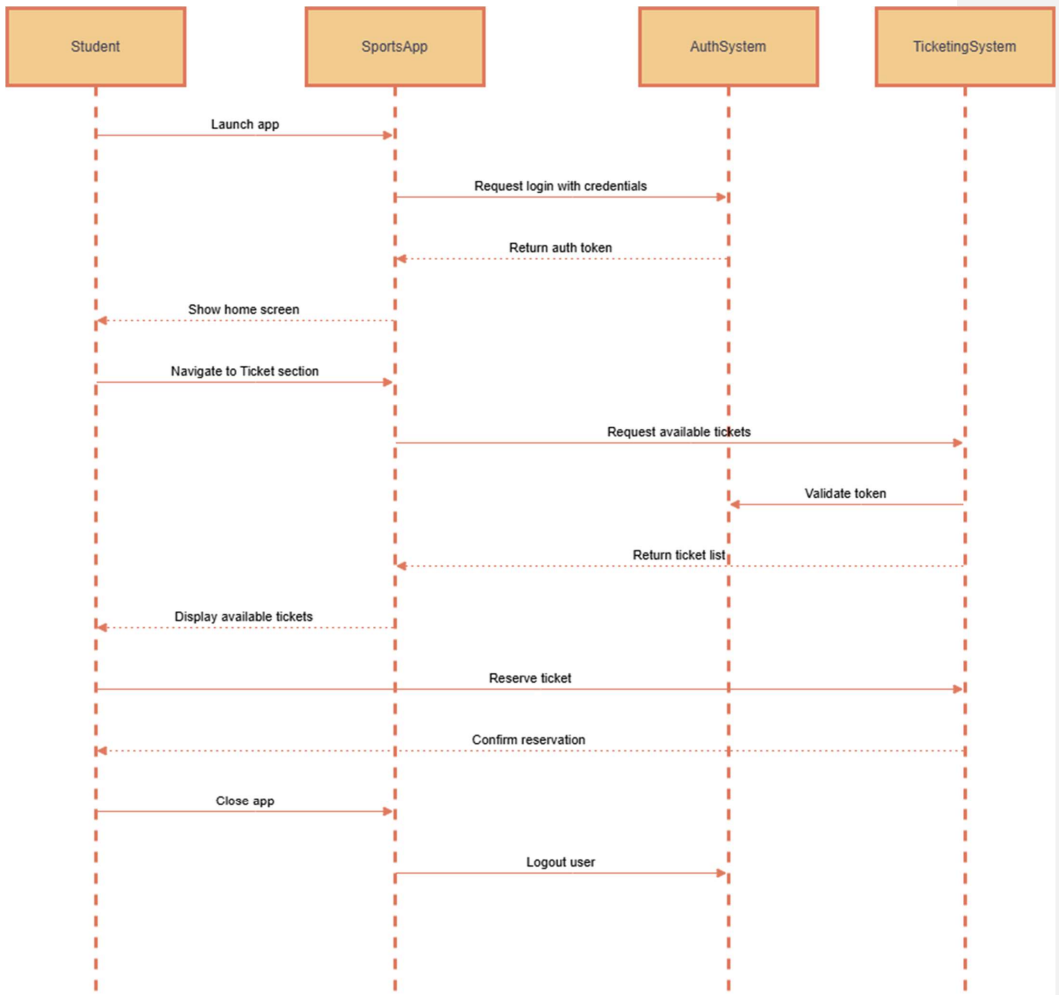
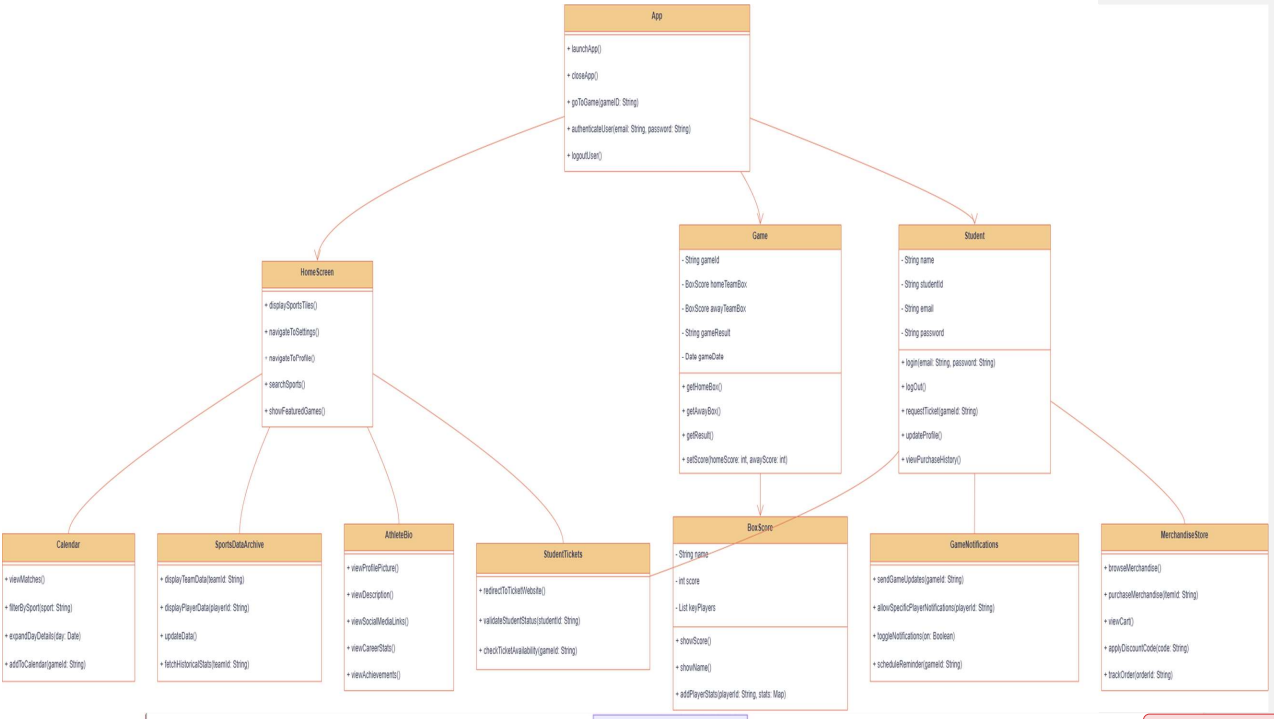

User
HomeScreen
CalendarModule
Database
Tap on Sports Tile
Open calendar view
Fetch upcoming games by sport
Return filtered game list
Display games in calendar

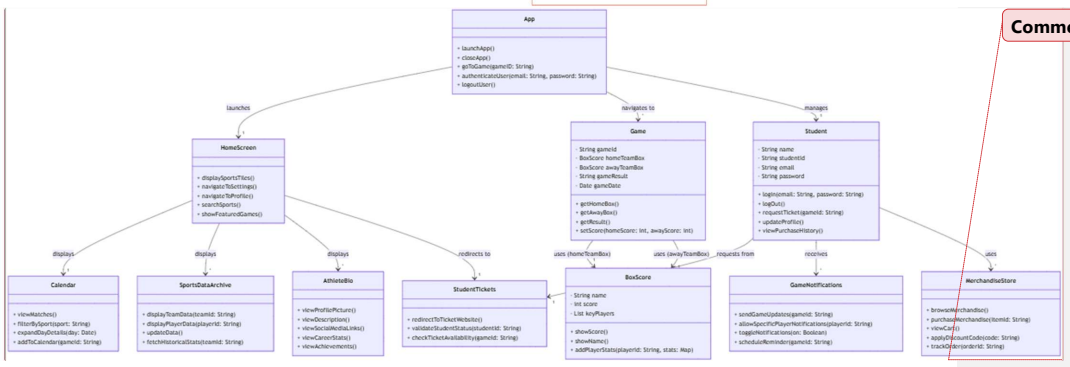
Diagram Title


Student
SportsApp
AuthSystem
NotificationSystem
GameSchedule
Launch app
Login with credentials
Return token
Show home screen
Enable notifications, select players
Send prefefences
Validate token
Check for matches
Upcoming games
Push game update
Close app
Logout user


Student
SportsApp
AuthSystem
TicketingSystem
Launch app
Request login with credentials
Return auth token
Show home screen
Navigate to Ticket section
Request available tickets
Validate token
Return ticket list
Display available tickets
Reserve ticket
Confirm reservation
Close app
Logout user

3. Structural Design





Commented [MJ1]: w/ * and 1's associations