CIS 628 Software Engineering
Jacob Wansor
Robert Nugent
Vimal Ramnarain

Tennis Application Final Document

1. Introduction

This application will be designed for Android phones and will provide users with easily accessible, up to date, and relevant statistics regarding professional tennis players and tournaments.

The application main screen will include a feed regarding well known tennis players and tournaments. A persistent search bar will allow users to look for specific players and tournaments. Core player information will include full name, date of birth/age, nationality, dominant hand, height and weight, and profile picture—along with career and performance data. Career and performance data will include win-loss records, current form (last 5 matches played or streaks), performance by surface (hard, clay, grass, indoor), head-to-head record against other players (such as rivals or current opponent in live tournaments), best of 3 and best of 5 performances, current ranking (ATP/ITF), highest career ranking, lowest career ranking, grand-slam titles won, Olympic medals won, years active, career prize money, prize money earned in the current season, sponsorships, and social media links. Tennis tournament information will encompass match scores, match dates, tournament rounds (for live tournaments, this will update as the tournament progresses), court surface, tournament location and category (Grand Slam, Masters 1000, or ATP 500). A persistent bottom navigation bar will allow users to return to the home screen, adjust profile settings, and log out of their account and return to the authentication page with just a single tap.

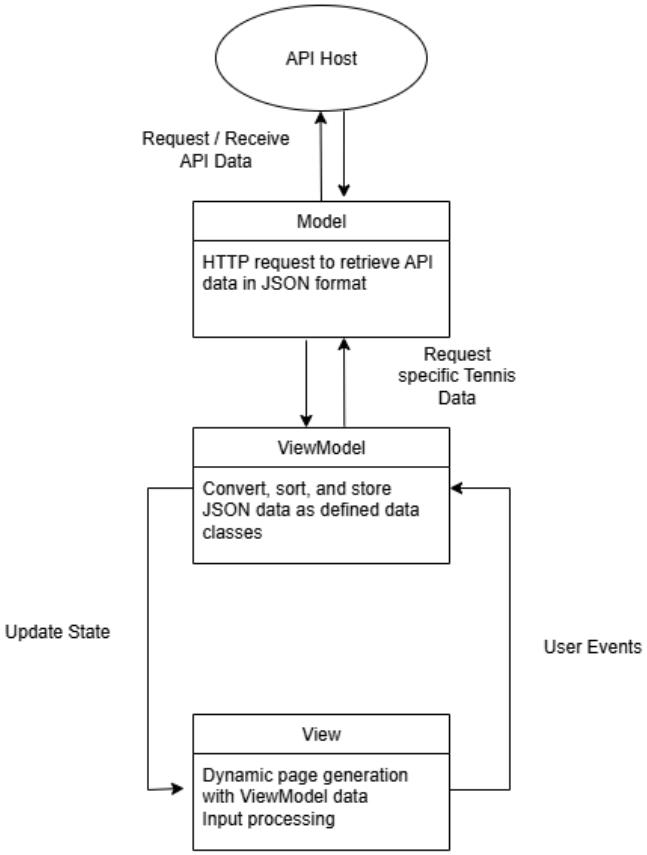
The application will include an optional and free account creation feature that enables users to follow and receive notifications on specific players and tournaments. Users who choose not to create an account may still use the app as a guest, but will not be able to receive notifications from the application.

2. System Architecture

The system architecture for this Android application relies on the Model-View-ViewModel (MVVM) pattern. This architecture was chosen because it provides a clear separation of concerns, making the codebase easier to maintain and extend over time. The model layer handles the data and business logic, the view is responsible only for the user interface, and the ViewModel acts as the mediator that binds data to the UI. This separation ensures that changes in one layer, such as updating the UI design or modifying data sources, can be done independently without heavily impacting the other layers. Additionally, using MVVM enables reactive UI updates through LiveData or StateFlow, which reduces boilerplate code and minimizes the risk of bugs when synchronizing UI with underlying data.

The system was primarily designed with the qualities of maintainability, testability, and scalability in mind. MVVM directly supports these qualities by isolating logic in the ViewModel, which makes it easier to write unit tests without relying on UI components. It also supports scalability because new features can be added by extending the model or adding new ViewModels without restructuring the entire application. Furthermore, this architecture enhances readability and collaboration, as different developers can focus on different layers of the system in parallel. Overall, MVVM is a strong fit for modern Android applications, such as this Tennis App, because it aligns well with lifecycle-aware components and supports long-term system growth while keeping the codebase organized and efficient.

Tennis App Model-View-ViewModel (MVVM) Architecture Overview



3. Class Diagram: See attached enclosure.

4.Requirements Table

Requirement Number	Description	Priority	Page
UR 4.1	Navigation via Bottom Bar	3	4
UR 4.2	Player Search Functionality	2	5
UR 4.3	View Player and Match Details	1 (Highest)	6
UR 4.4	Authentication	6	7
UR 4.5	User Privacy and Data Protection	7	8
UR 4.6	Track Specific Players and Tournaments	9 (Lowest)	9
UR 4.7	Home Feed and Live Highlights	4	10
UR 4.8	Tournament and Match Browsing	5	11
UR 4.9	Profile Management and Account Actions	8	12

UR 4.1 — Navigation via Bottom Bar

Description:
The system shall allow users to navigate between the Home Page, Profile Page, and Logout using a persistent bottom navigation bar.

Priority: 3

Functional System Requirements:

- **4.1.1F** The system shall display a bottom navigation bar on all primary user-facing pages (Home, Profile, Player, Match, Tournament).
- **4.1.2F** The system shall route the user to the appropriate page when a navigation item (Home, Profile, or Logout) is tapped.
- **4.1.3F** The system shall maintain the user session across navigations and only terminate the session when Logout is tapped.
- **4.1.4F** The system shall visually highlight the currently active navigation item.

Non-Functional System Requirements:

- **4.1.5NF** The bottom navigation bar shall be rendered within 200 milliseconds after any page load. *(Performance)*
- **4.1.6NF** If the target page fails to load due to a network issue, the system shall display a retry prompt and allow the user to attempt reloading.
- **4.1.7NF** The navigation bar shall be accessible and usable with one hand on standard smartphone screen sizes (≥ 5"). *(Usability)*
- **4.1.8NF** The navigation bar shall remain visually consistent and fixed at the bottom across all pages. *(Design Consistency)*
- **4.1.9NF** The system shall ignore rapid repeated taps (<300 ms apart) on the same navigation item to prevent accidental double navigation.
- **4.1.10NF** If the user selects Logout, the system shall clear all cached user data before redirecting to the Login screen.

UR 4.2 — Player Search Functionality

Description:
The system shall allow users to search for tennis players using a top search bar accessible across all relevant pages.

Priority: 2

Functional System Requirements:

- **4.2.1F** The system shall display a top search bar on the Home, Search, Player, Match, and Tournament pages.
- **4.2.2F** The system shall return a list of player cards (name, short info) based on the user's input query.
- **4.2.3F** The system shall allow users to click on any player in the search results to open the Player Info Page.
- **4.2.4F** The system shall retain the last search query and results until the user clears the search bar or exits the Search Page.
- **4.2.5F** The system shall display a "No Results Found" message if no player matches the search query.
- **4.2.6F** The system shall suggest autocomplete search terms as the user types.

Non-Functional System Requirements:

- **4.2.7NF** The search function shall return relevant results within 1 second of input completion. *(Performance)*
- **4.2.8NF** The system shall support partial and fuzzy search queries (e.g., typo tolerance or substring matches). *(Functionality/Robustness)*
- **4.2.9NF** The search bar shall have a consistent appearance (font size, padding, border) across all pages. *(Design Consistency)*
- **4.2.10NF** The system shall handle and recover gracefully if the search API request fails, displaying a "Retry" option.
- **4.2.11NF** Search suggestions shall not display more than 10 results at a time to maintain UI clarity.

UR 4.3 — View Player and Match Details

Description:
The system shall allow users to view a selected player's profile, including recent match and tournament information.

Priority: 1

Functional System Requirements:

- **4.3.1F** The system shall display a Player Info Page with player name, image (optional), stats, and basic info.
- **4.3.2F** The system shall provide links or interactive elements to navigate to recent Match Pages for that player.
  4.3.3F The system shall provide links or interactive elements to navigate to the player's associated Tournament Pages.
- **4.3.4F** The system shall display a "No Data Available" message if player stats or match history cannot be retrieved.
- **4.3.5F** The system shall display placeholders for player image and stats while data is being fetched.
- **4.3.6F** The system shall sort matches chronologically, with the most recent matches appearing

Non-Functional System Requirements:

- **4.3.7F** The Player Info Page shall load within 2 seconds on a 4G network. *(Performance)*
- **4.3.8F** All match and tournament links shall be accessible with a minimum 44x44dp touch target size. *(Accessibility)*
- **4.3.9F** The layout of the Player Info Page shall be responsive to various screen sizes and orientations. *(Usability/Responsive Design)*
- **4.3.10F** The system shall allow retrying data retrieval if the initial load fails due to a network error.
- **4.3.11F** Player images shall be cached locally to speed up subsequent loads.

UR 4.4 — Authentication

Description:
The system shall allow users to register an account on the app to personalize their feed. Users may skip authentication and proceed as a guest. Guest accounts will have a default feed.

Priority: 6

Functional System Requirements:

- **4.4.1F** When opening the app, the first screen displayed will be a login page where the user can enter their username and password to login.
- **4.4.2F** The login page will include a “Register”,“Forgot Password” and “Contact Us” button for account registration and recovery.
- **4.4.3F** The system shall have a register screen where the user provides an e-mail, username, password, confirmation of password, and security question to create an account.
- **4.4.4F** When a user registers an account, the system will ensure an account associated with that e-mail does not already exist. If one does exist, the system will notify the user and not create the account.
- **4.4.5F** The system will enable account recovery strictly within the app when “Forgot Password” is selected. Users must correctly answer their security question to reset their password.
- **4.4.6F** The “Contact Us” display allows users to send an e-mail to a support team for further account issues.
- **4.4.7F** The Log In screen will include a “Guest” button to skip the login process.
- **4.4.8F** Users can log out of their account by clicking the “Logout” icon on the bottom navigation tool bar after logging in.

Non-Functional System Requirements:

- **4.4.9NF** After a user provides the correct username and password, navigation to the home page will take less than 1 second with a 5G mobile network . *(Performance)*
- **4.4.10NF** The system will be linked to and communicate with a Firebase authentication server to store registered user emails and passwords. *(Design Constraint)*
- **4.4.11NF** The system will be linked to and communicate with Firebase Firestore to store additional account information to include security questions, hashed answers to security questions, whether the account is locked, and most recent login timestamp. *(Design Constraint)*.

UR 4.5 — User privacy and data protection

Description:
The system shall ensure user information retained by the system uses best security practices and protects the user's privacy and data.

Priority: 7

Functional System Requirements:

- **4.5.1F** After user registration, the app displays a security policy pop up reminding users to not share their password and that the app will not initiate unsolicited contact with user or distribute/sell user data to 3rd parties.
- **4.5.2F** System will notify user and lock their account for 30 minutes after 3 unsuccessful login attempts
- **4.5.3F** System will notify user and lock their account for 30 minutes after 3 unsuccessful attempts to answer their security question
- **4.5.4F** When a user attempts to register an account with a password that does not meet security requirements, the user will be notified that account creation was unsuccessful due to the password not meeting requirements. The system then persistently displays password length and character requirements.
- **4.5.5F** System will automatically delete accounts that have been inactive for 365 consecutive days.

Non-Functional System Requirements:

- **4.5.6NF** User passwords must be at least 8 characters and contain at least 1 special character. *(Organizational Security Policy)*
- **4.5.7NF** User account passwords will be hashed with the "bcrypt" algorithm prior to storage in Firebase Authentication Server. *(Organizational Security Policy)*
- **4.5.8NF** User account security questions will be hashed with the "bcrypt" algorithm prior to storage in Firestore database. *(Organizational Security Policy)*
- **4.5.9NF** System will ensure that each account is associated with a unique e-mail. *(Organizational Security Policy)*

UR 4.6 — Track specific players and tournaments

Description:
The system will enable users to subscribe to specific tennis players and tournaments. The home screen feed will automatically include information about players and tournaments that a user has subscribed to. The system will also send notifications if there are any updates about subscribed items.

Priority: 9

Functional System Requirements:

- **4.6.1F** All players and tournaments cards will include a toggleable bell icon in the top right corner indicating that a user wishes to follow and track updates on those items.
- **4.6.2F** Notifications will be sent to the user's phone any time a game, set, or match is won or lost involving a subscribed player.
- **4.6.3F** Notifications will be sent to the user's phone any time a match is won or lost involving a subscribed tournament.
- **4.6.4F** Users may choose to turn off notifications within the profile screen.
- **4.6.4F** After a user has logged in, the home page tennis feed is updated to include the latest information regarding subscribed players and tournaments.

Non-Functional System Requirements:

- **4.6.4NF** Database will retain player and tournament subscriptions for each account. (*Design)*
- **4.6.5NF** Firebase cloud messaging will allow push notifications to the user phone even when the application is not running. *(Design)*

UR 4.7 — Home Feed & Live Highlights

Description:
The system shall present a Home feed with a featured live tournament card and quick modules for player stats and tournaments, each navigable to their detailed pages.

Priority: 4

Functional System Requirements:

- **4.7.1F** The Home page shall display a “Live Tournament” highlight card when live data is available; tapping it routes to the Tournament Page.
- **4.7.2F** The Home page shall show “Player Stats” and “Tournaments” modules that deep-link to Player and Tournament pages respectively.
- **4.7.3F** The system shall auto-refresh the Home feed content at most every 60 seconds while the app is foregrounded.
- **4.7.4F** If live data is unavailable, the system shall gracefully degrade to the most recent cached feed.

Non-Functional System Requirements:

- **4.7.5NF** Live highlight and modules shall render above the fold on devices ≥ 5" screens. (Usability)
- **4.7.6NF** Home feed refresh shall not block user interaction and shall complete within 1.5 seconds on a 4G network. (Performance)

UR 4.8 — Tournament & Match Browsing

Description:
The system shall provide a Tournament Page listing matches with core details (players, scores, date/time, location) and enable navigation to Match details.

Priority: 5

Functional System Requirements:

- **4.8.1F** The Tournament Page shall list matches with opponent names, latest score, date/time, location, and a brief stats snippet.
- **4.8.2F** Tapping a match row shall open the Match Page with expanded details.
- **4.8.3F** Users shall be able to filter matches by status (Upcoming, Live, Completed) and sort by date/time.
- **4.8.4F** The Tournament Page search bar shall filter visible matches by player name or substring.

Non-Functional System Requirements:

- **4.8.5NF** Match list scrolling shall maintain 60 FPS on mid-tier Android devices (2022+). (Performance)
- **4.8.6NF** Touch targets for each match row shall be ≥ 44×44dp. (Accessibility)

UR 4.9 — Profile Management & Account Actions

Description:
The system shall allow users to view and edit profile information (username, email, profile picture) and perform account actions (reset password, delete account) from the Profile page.

Priority: 8

Functional System Requirements:

- **4.9.1F** The Profile Page shall display username, email, and profile picture with an option to update each field.
- **4.9.2F** The system shall allow changing the profile picture by selecting a local image or taking a photo; images are stored in the user's profile record.
- **4.9.3F** The Profile Page shall expose "Reset Password" and "Delete Account" actions; delete requires explicit confirmation and re-authentication.
- **4.9.5F** After a successful update, the system shall show an in-app confirmation and persist changes to Firestore/Auth.

Non-Functional System Requirements:

- **1.9.5NF** Profile updates shall complete within 2 seconds on a 4G network. (Performance)

- **1.9.6NF** Images uploaded for profile pictures shall be validated (≤ 5 MB, JPEG/PNG/WebP) and downscaled to ≤ 512×512 px server-side. (Reliability/Security)

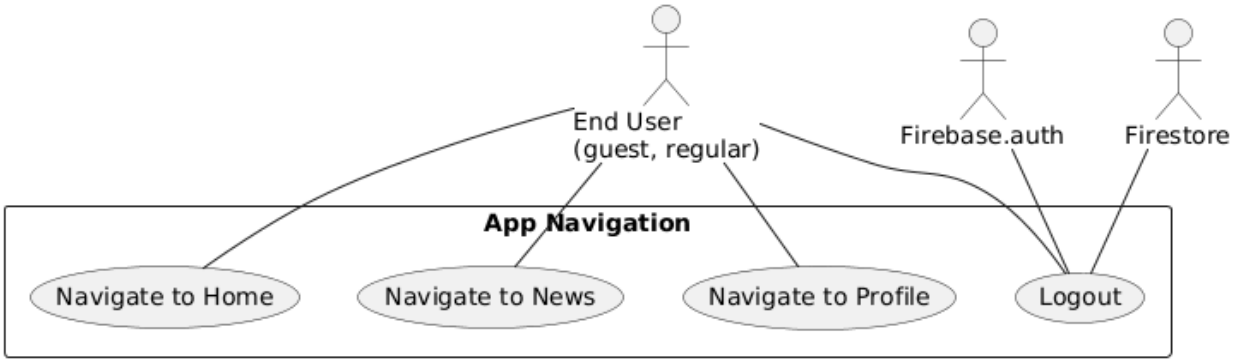
5. Use Case Table

Use Case	Description	Priority	Page
UC 5.1	Navigate with Bottom Navigation Bar	High	14
UC 5.2	View Live Match Scores	High	16
UC 5.3	View Player Profile	High	19
UC 5.4	View Details on Past Tournament	High	22
UC 5.5	Register Account	Low	24
UC 5.6	Login with Account	Low	27
UC 5.7	Live News	Medium	30
UC 5.8	Search for Players	Low	33
UC 5.9	Manage User Profile	Low	36

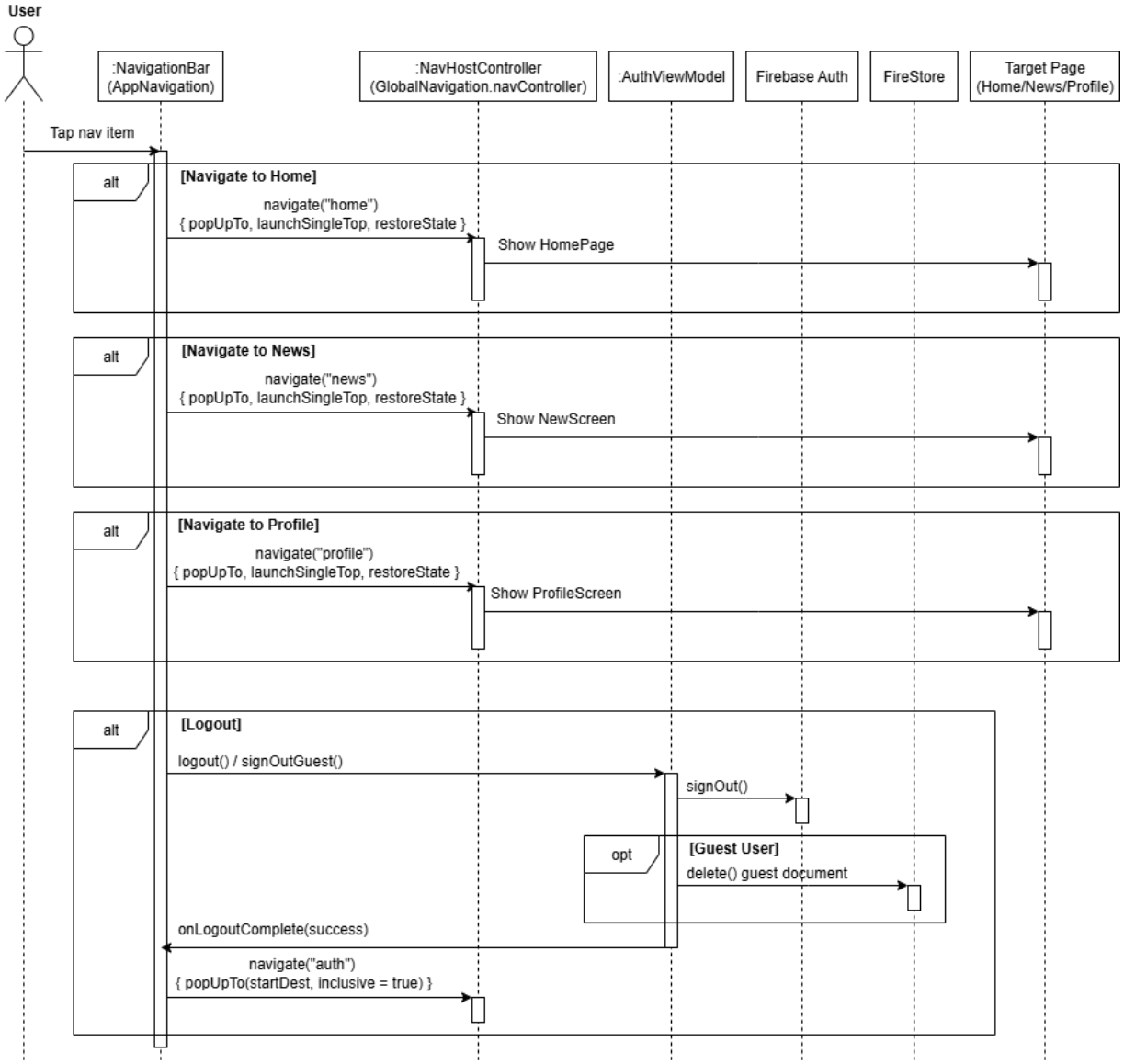
5.1.1 USE CASE

Title:	Navigate with Bottom Navigation Bar
Description:	The user navigates between Home, News, Profile and Logout using the persistent bottom navigation bar inside Scaffold (AppNavigation). The bar is hidden on auth pages. Normal route taps call NavHostController.navigate(...) with popUpTo/ launchSingleTop/ restoreState. Logout uses AuthViewModel to sign out (guest or regular), deletes guest user doc when needed, and only then navigates to the auth screen (back stack cleared).
Actors:	End User (guest, regular) Firebase.auth (auth state) Firestore (guest user document deletion)
Stimulus:	The user taps a bottom navigation item (Home, News, Profile, Logout).
Preconditions:	1. navController is created and assigned to GlobalNavigation.navController. 2. firstScreen chosen based on Firebase.auth.currentUser. 3. AuthViewModel is available and has logout() and signOutGuest() operations. 4. The bottom bar is visible (not on auth routes).
Postconditions:	1. For normal navigation: target page displayed, active nav item visually highlighted, session remains intact. 2. For Logout: guest user record deleted (if guest), Firebase auth state cleared, then navigation to auth route with backstack cleared. Navigation occurs after sign-out is complete.
Main Success Scenario:	1. The user taps a non-logout item. a. AppNavigation calls navController.navigate(item.route) with popUpTo(item.route) { inclusive = true }, launchSingleTop = true, restoreState = true. UI highlights the selected item. 2. The user taps Logout: a. AppNavigation reads Firebase.auth.currentUser (or delegates to AuthViewModel) and decides guest vs normal. b. AppNavigation calls AuthViewModel.signOutGuest(onComplete) or AuthViewModel.logout(onComplete). These methods perform all sign-out operations (delete guest doc, delete guest user, sign out) and invoke the callback when finished (success or failure). c. On successful sign-out, AppNavigation navigates to "auth" with popUpTo(navController.graph.startDestinationId) { inclusive = true }. d. If sign-out fails, show an error (snackbar/dialog) and do not navigate. 3. HomePage AppLifecycleObserver background sign-out must also call AuthViewModel.signOutGuest(onComplete) and wait for completion before navigating to auth.
Extensions:	1. If rapid repeated taps: add a timestamp guard to ignore taps < 300 ms. 2. If AuthViewModel.logout() is currently synchronous (auth.signOut()), augment it to clear caches and expose a completion callback. 3. Guest deletion is asynchronous—add success/failure handlers and only redirect after successful deletion.
Priority:	3

5.1.2 USE CASE DIAGRAM



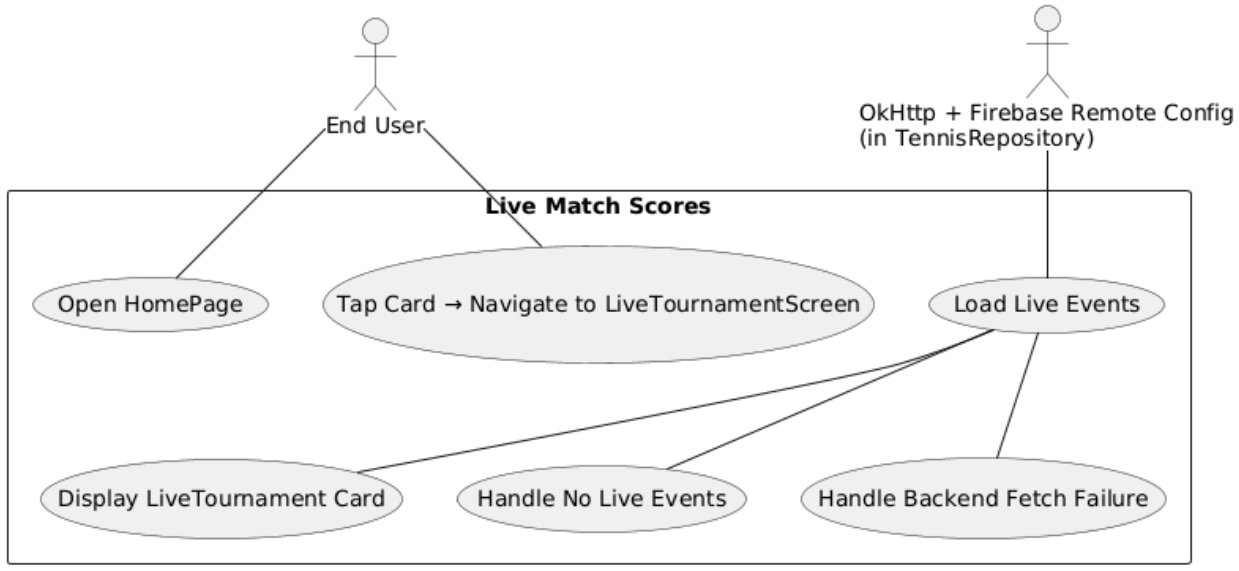
5.1.3 USE CASE SEQUENCE DIAGRAM



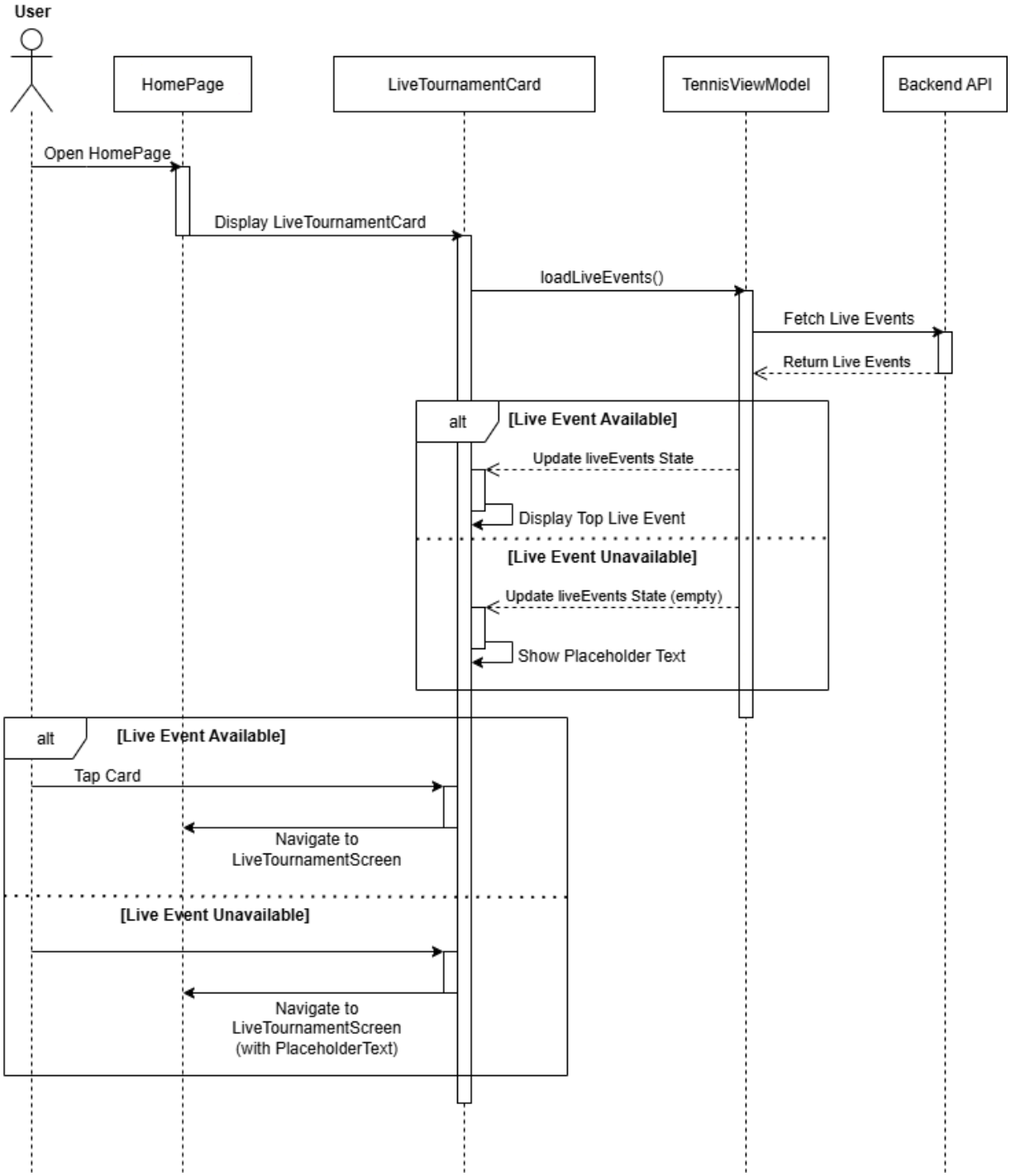
5.2.1 USE CASE

Title:	View Live Match Scores
Description:	The user views ongoing tennis match scores via a LiveTournament card on the HomePage. The card displays the first available live event with tournament name, season, location, match date, player names with country flags, and current set scores. Tapping the card navigates the user to the full LiveTournamentScreen for additional matches. The system fetches live data from the backend through TennisViewModel when the card is displayed. If no live events are available, the card shows a “No live events currently” message.
Actors:	End User OkHttp + Firebase Remote Config (in TennisRepository)
Stimulus:	The user opens the HomePage or taps on the LiveTournament card
Preconditions:	1. The app is running and the TennisViewModel is initialized. 2. Live match data is available in the backend API. 3. LiveTournamentCard is visible on the HomePage.
Postconditions:	1. The top live event is displayed on the card. 2. If the user taps the card, the app navigates to the LiveTournamentScreen. 3. If no live events are available, the “No live events currently” message is displayed.
Main Success Scenario:	1. The user opens the Home feed. 2. LiveTournamentCard triggers TennisViewModel.loadLiveEvents() via LaunchedEffect. 3. The system fetches live events from the backend and updates liveEvents state. 4. If liveEvents is empty, the card displays “No live events currently.” 5. If liveEvents is not empty, the first live event is displayed using LiveTournamentView. 6. The card shows: 7. Tournament name (shortened if needed), season, location, match date 8. Home and away player/team names with country flags 9. Current set scores for up to 3 sets 10. The user taps the card. 11. The system navigates to the full Live Tournament screen ("live" route) where additional live events are displayed.
Extensions:	1. No Live Events: If liveEvents is empty, display the placeholder text and prevent navigation to an empty screen. 2. Partial Data: If player flags, scores, or tournament info are missing, default placeholders (e.g., “unknown” or 0) are displayed. 3. Rapid Taps: Ignore rapid repeated taps (<300 ms) on the card to prevent redundant navigation calls. 4. Backend Fetch Failure: If fetching live events fails, display the placeholder message and allow the user to retry by refreshing the feed.
Priority:	2

5.2.2 USE CASE DIAGRAM



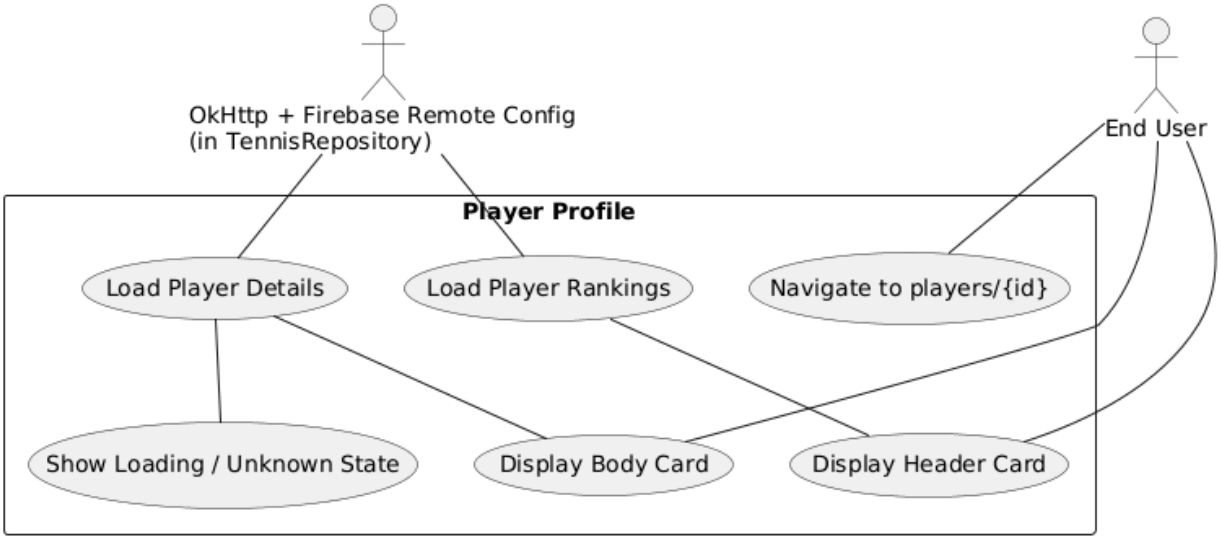
5.2.3 USE CASE SEQUENCE DIAGRAM



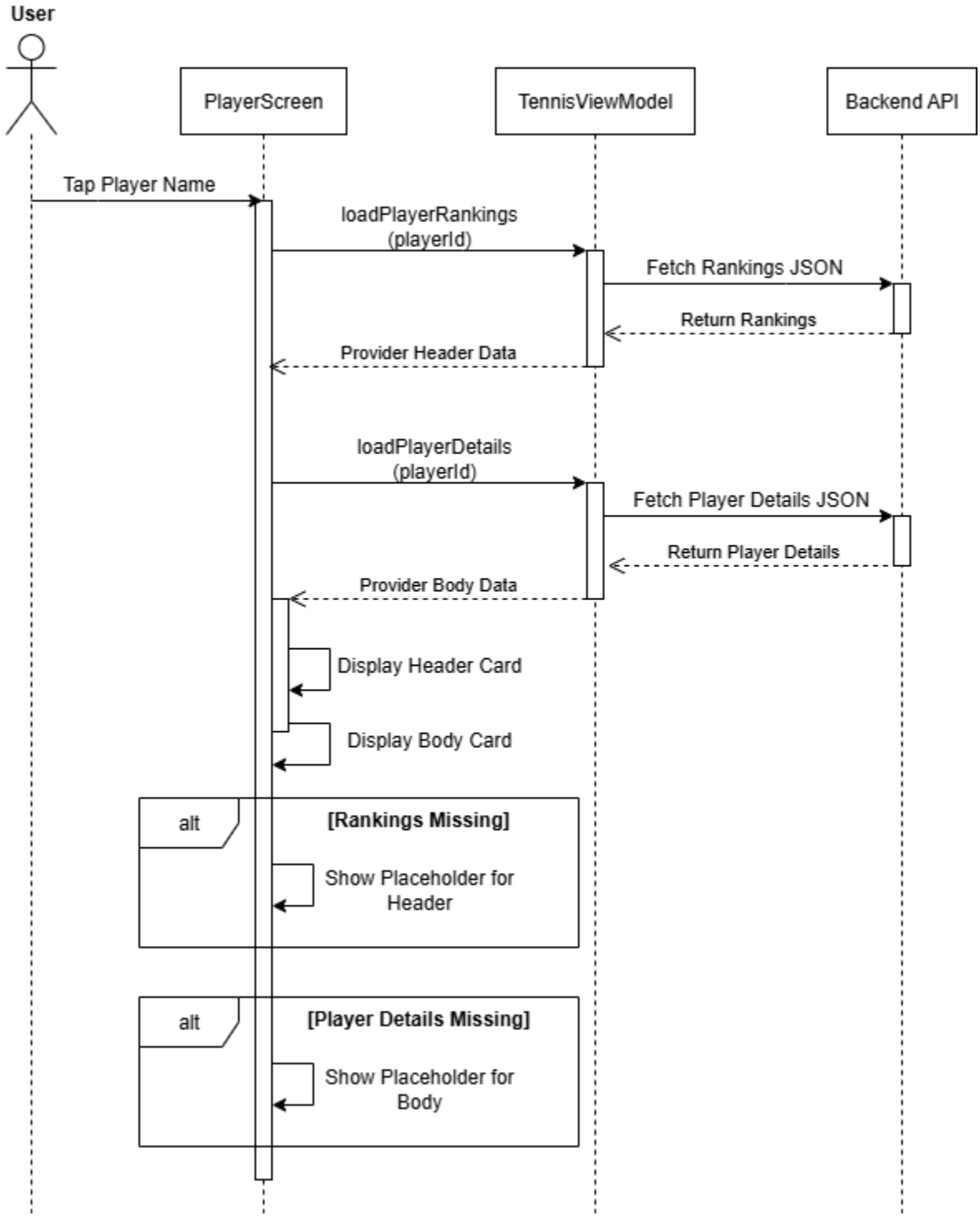
5.3.1 USE CASE

Title:	View Player Profile
Description:	When a user navigates to a specific player route (players/{id}), the app displays a PlayerScreen showing a header card with the player's name, country flag (mapped from assets), rank, and points (from rankings). The body card asynchronously loads extended PlayerDetails (status, turned pro year, physical attributes, birthplace, residence, handedness, and coach). PlayerScreen requests tennisViewModel.loadPlayerDetails(playerId) in a LaunchedEffect when the ID changes. Rankings are pulled from tennisViewModel.rankings and detailed player info from tennisViewModel.playerDetails. When data is unavailable, the screen shows a CircularProgressIndicator or a fallback "Unknown" value.
Actors:	End User OkHttp + Firebase Remote Config (in TennisRepository)
Stimulus:	The user taps on a player name in PlayerStatsCard on HomeScreen or in search results causing navigation to route players/{id}.
Preconditions:	1. "players/{id}" route is registered in NavHost. 2. TennisViewModel exposes rankings and playerDetails, with loadPlayerDetails(id: Int) implemented. 3. TennisRepository is configured and able to fetch player details JSON (nested structure parsed into PlayerDetails → PlayerInformation). 4. Flag SVGs exist in assets/flags/{countryAcr}.svg, with fallback to placeholder.
Postconditions:	1. The PlayerScreen header card displays the player's name, rank, points, and flag. 2. Body card displays extended player details from API. 3. UI remains responsive; loading state shows progress indicator. 4. Invalid/missing details fall back to "N/A" or "Unknown" gracefully.
Main Success Scenario:	1. User taps a player from PlayerRankListView → NavHost navigates to players/{id}. 2. PlayerScreen loads with LaunchedEffect(playerId) → tennisViewModel.loadPlayerDetails(id). 3. Rankings state already available in tennisViewModel.rankings provides player's name, rank, points, and country info. 4. AsyncImage loads country flag SVG (file:///android_asset/flags/{acronym}.svg) with fallback to placeholder. 5. While tennisViewModel.playerDetails is null, show a centered CircularProgressIndicator. 6. Once details load, show fields (status, turned pro, height, weight, birthplace, residence, plays, coach) using lexendLight typography. 7. The user can scroll vertically through details (verticalScroll).
Extensions:	1. If playerId is null → screen logs error and shows "Unknown". 2. If API fails to load player details → progress indicator persists, and user sees no details (consider adding Retry button). 3. Missing flag asset (unknown.svg) → show placeholder text or emoji instead of broken image. 4. If a ranking entry is not found for playerId, fallback rank = "-" and points = "-". 5. Navigation to future "Match History" (matches/{playerId}) can be added via clickable element.
Priority:	1

5.3.2 USE CASE DIAGRAM



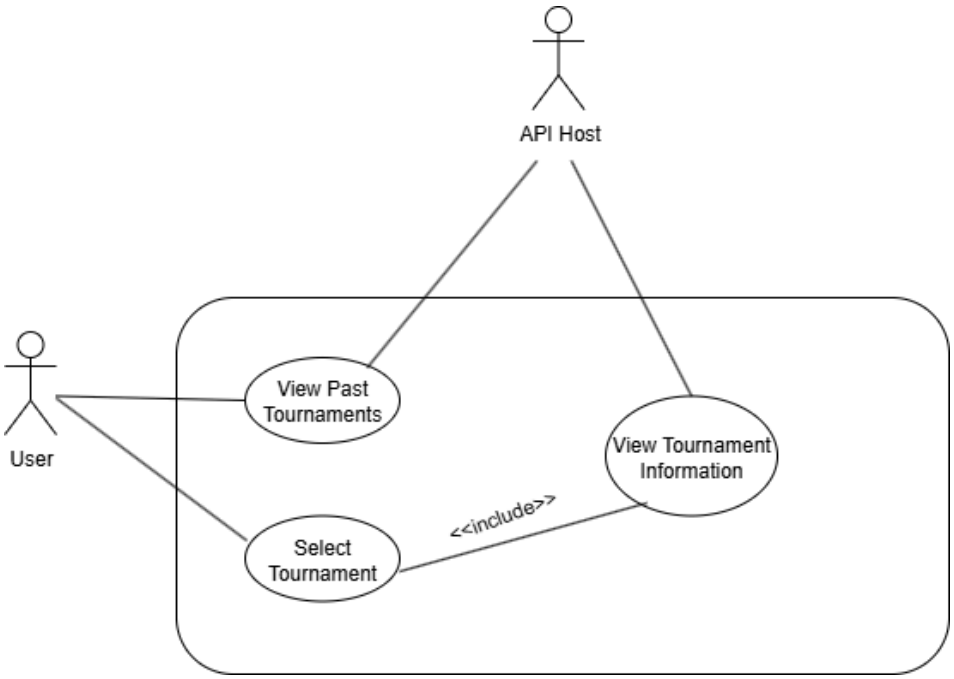
5.3.3 USE CASE SEQUENCE DIAGRAM



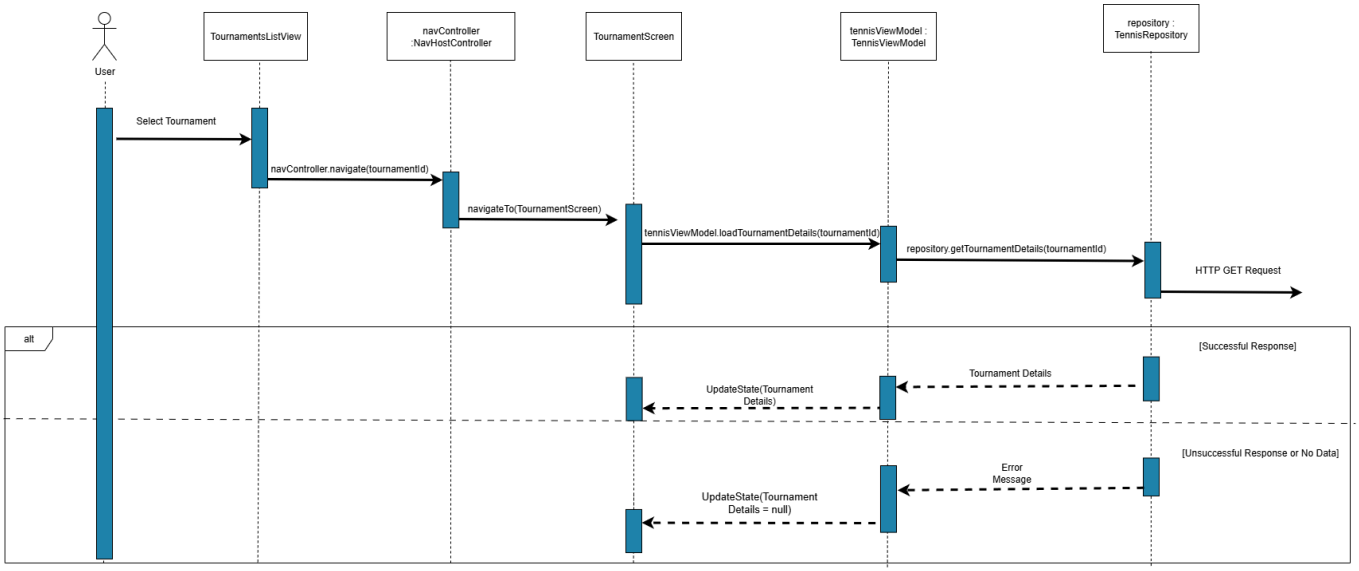
5.4.1 USE CASE

Title:	View Details on Past Tournament
Description:	When a user navigates to a specific tournament route (tournament/{tournamentId}), the app displays a Tournament screen showing a header card with the tournament name and date. Separate expandable rows are generated below the header card that provide details on specific rounds of the tournament. Each expandable row lists the results of every game played in that round. TournamentScreen requests tennisViewModel.loadTournamentDetails(tournamentId) in a LaunchedEffect when the ID changes. TournamentScreen retrieves the tournament with the corresponding tournamentID from tennisViewModel.tournaments and retrieves information from that tournament from tennisViewModel.tournamentDetail. When data is unavailable, the screen gracefully shows a CircularProgressIndicator or a fallback “Unknown” value.
Actors:	Guest User, Registered User
Stimulus:	The user selects a tournament from the list of tournaments on the HomeScreen.
Preconditions:	1. Home screen successfully retrieved API data 2. Tournament identifier corresponds to pulled API data 3. Device online (or cached data exists)
Postconditions:	1. Navigation to Tournament Screen 2. Tournament Header Card displayed 3. Expandable rows with tournament round information displayed.
Main Success Scenario:	1. The user clicks a specific tournament from the list of tournaments on the home screen. 2. System navigates to TournamentScreen, fetches tournament data from .TennisViewModel 3. TournamentScreen generates a header card and dynamically generates expandable rows based on the number of unique roundIDs. 4. When an expandable row is expanded, a scrollable view displays all match results corresponding to the tennis round.
Extensions:	1. Data source missing sections → System shows placeholders and “No data available.” 2. Network error → System shows cached data and a refresh option. 3. Round IDs not mapped → Header card displayed with no expandable rows.
Priority:	5

5.4.2 USE CASE DIAGRAM



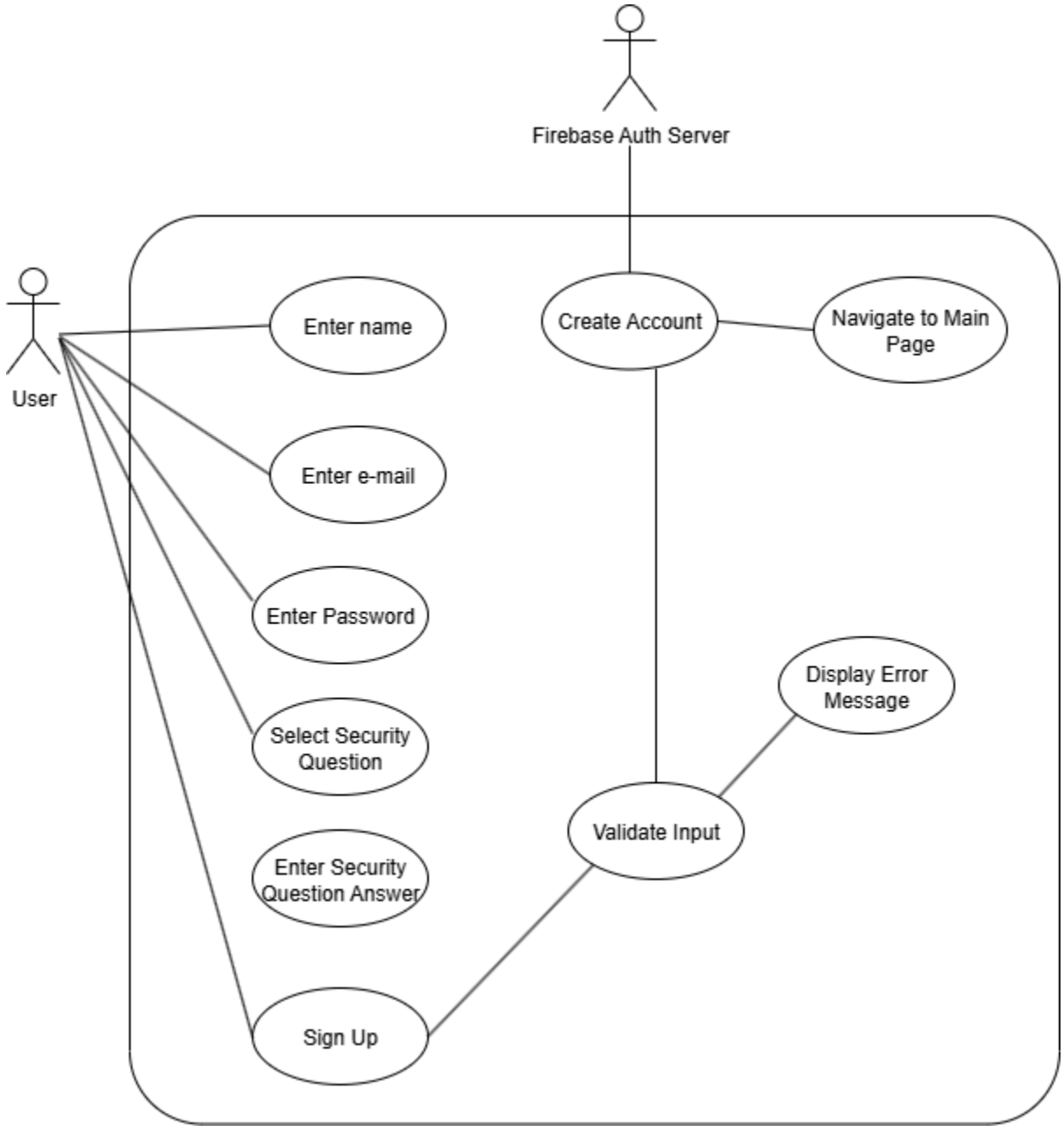
5.4.3 USE CASE SEQUENCE DIAGRAM



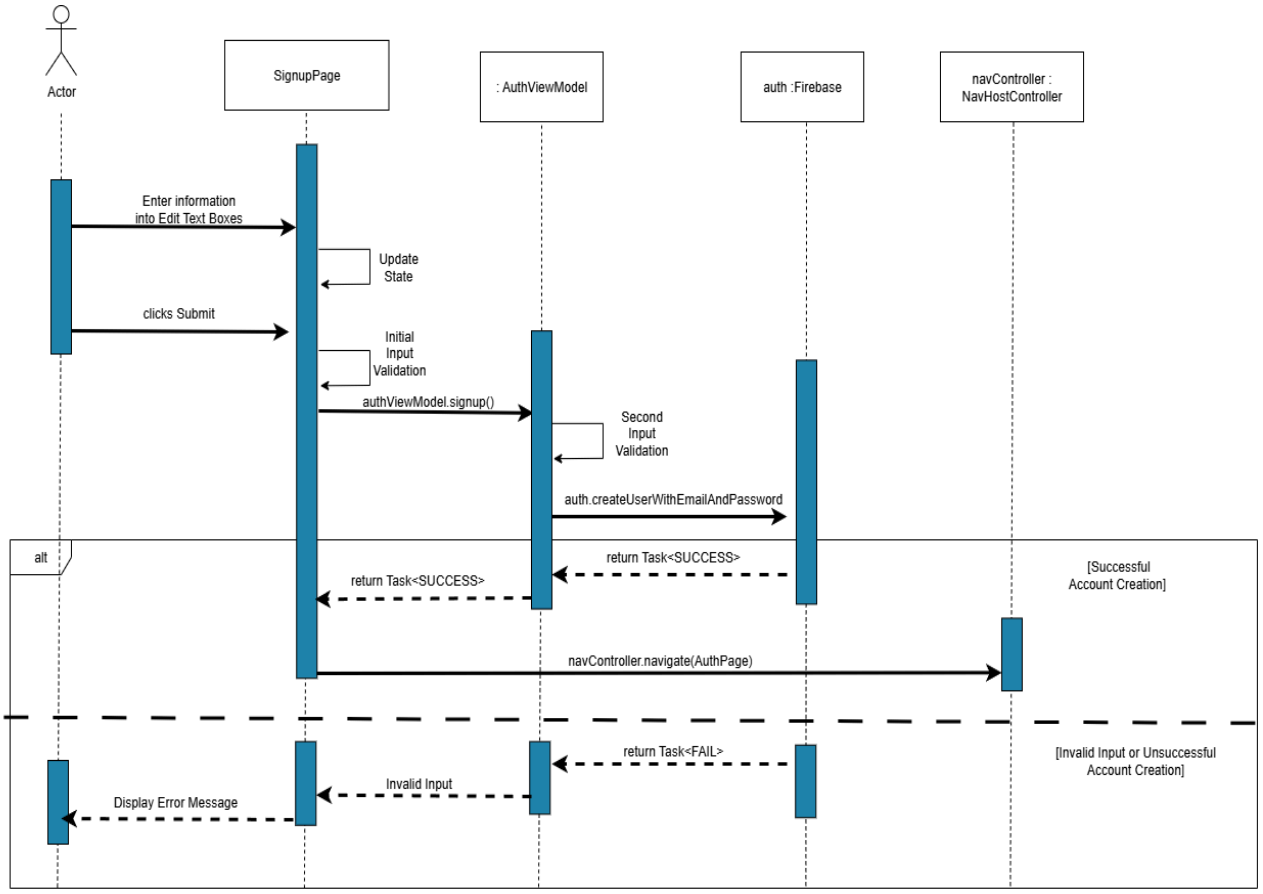
5.5.1 USE CASE

Title:	Register Account
Description:	The user registers an account to access personalized features (e.g., profile management, notifications). On the Sign Up Page user enters their name, email, desired password, selects a security question from the drop down menu, and provides an answer to the security question. Once all information has been provided the user selects the Submit button to register the account.
Actors:	Unregistered user
Stimulus:	User selects the Sign Up button
Preconditions:	1. Guest is not authenticated, 2. The auth service is available, 3. The user has provided valid input to the name, email, password, security question, and security question answer fields.
Postconditions:	1. A user account is created 2. The account is stored in the Firebase Authentication server.
Main Success Scenario:	1. User selects “Sign-Up” 2. System prompts for name, email, password, security question, and answer to security question 3. The user submits valid inputs. (No empty fields, minimum password length, etc.) 4. The system creates an initial user profile record and stores information in Firebase Authentication Server. 5. System navigates user to Log In screen
Extensions:	1. Invalid email/password strength → System notifies user of invalid input. No account created. 2. Fields left empty → System shows Toast error. No account created. 3. Email already in use → System notifies the user of a non-unique email. No account Created. 4. Profile creation fails after auth success → System notifies user of an unknown error. No account created. 5. No internet connection → System notifies user of a network error
Priority:	7

5.5.2 USE CASE DIAGRAM



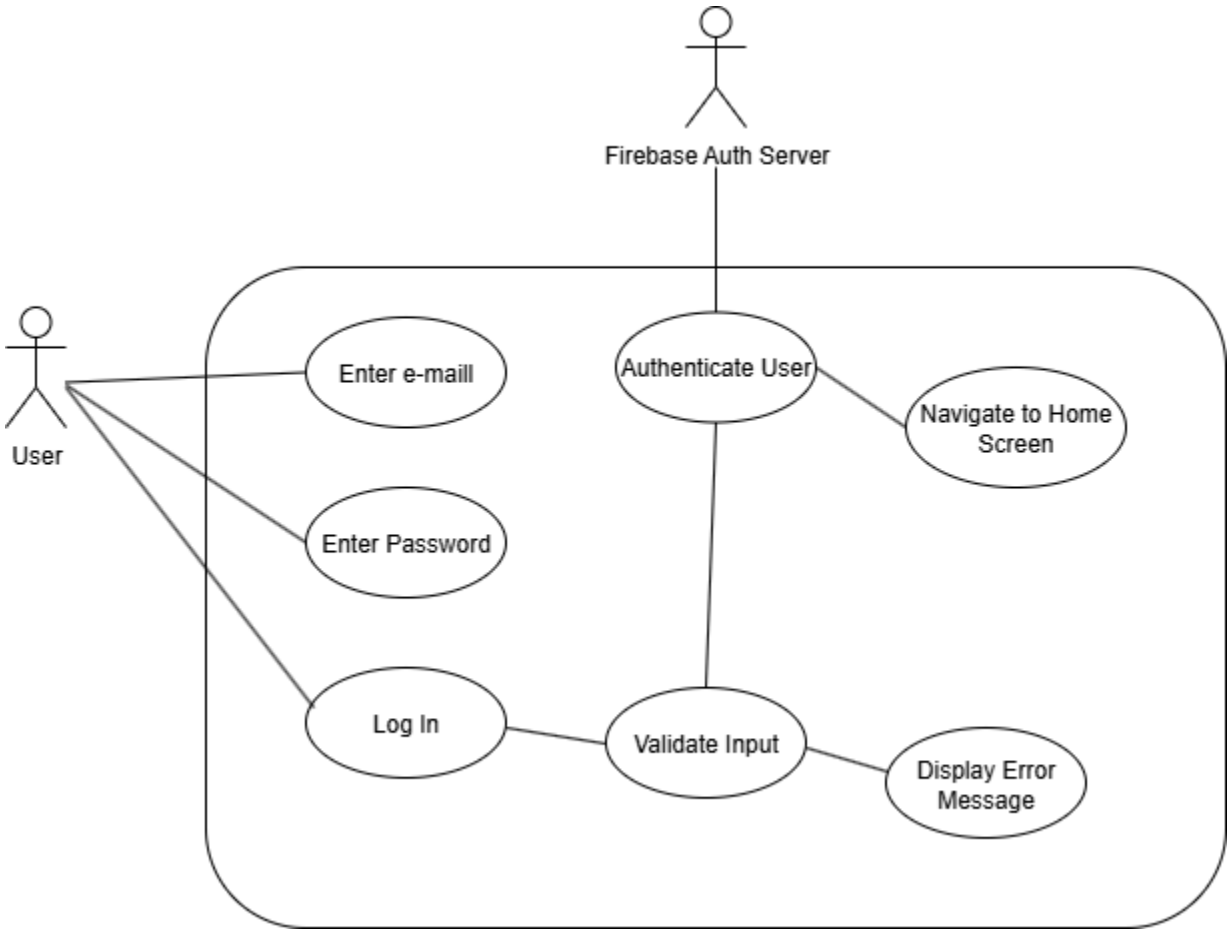
5.5.3 USE CASE SEQUENCE DIAGRAM



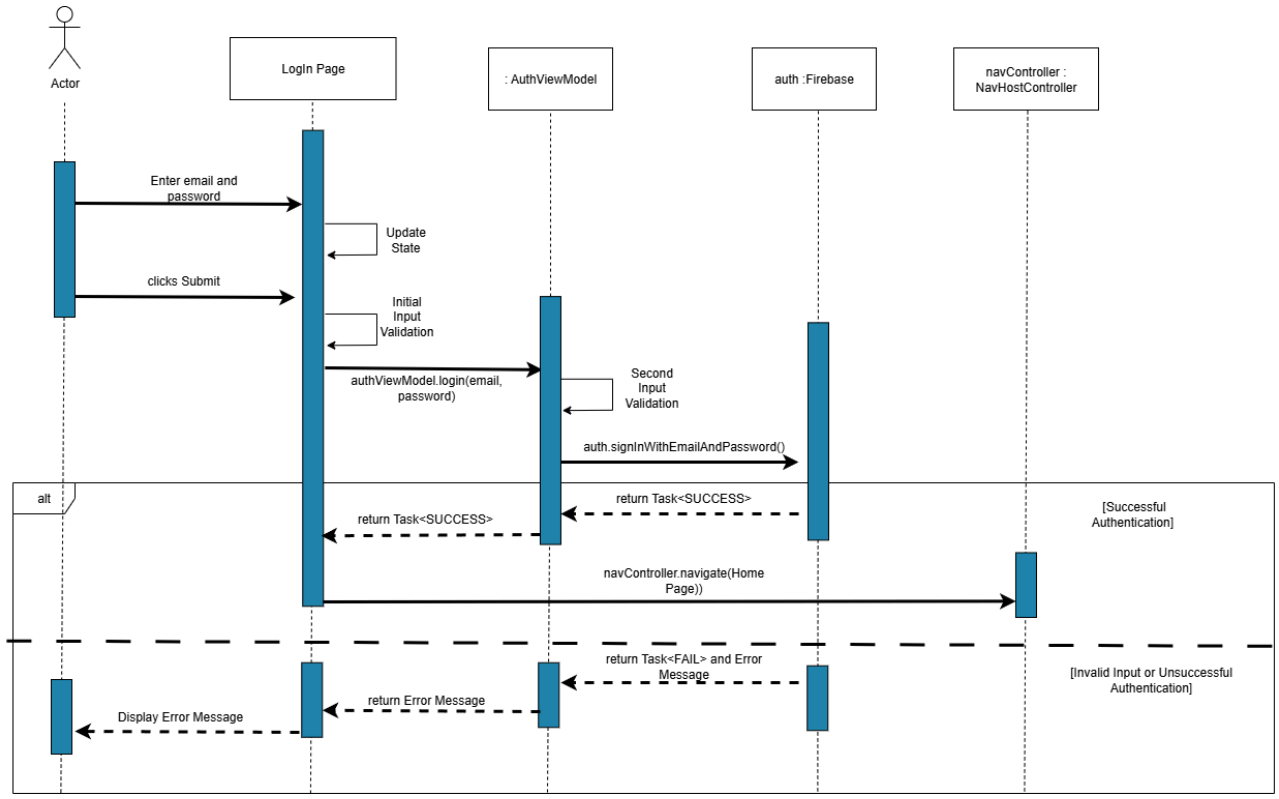
5.6.1 USE CASE

Title:	Login with Account
Description:	A user with an existing account provides their registered e-mail and password on the Log In page and clicks submit. The e-mail and password are passed to the Authentication View Model which calls Firebase function signInWithEmailAndPassword. The e-mail and password are compared with accounts stored in Firebase Authentication Server.
Actors:	Registered User, Unregistered User
Stimulus:	User taps Login
Preconditions:	1. The User is not authenticated 2. auth service available.
Postconditions:	1. On login, the user is authenticated 2. Session tokens are stored
Main Success Scenario:	1. The user enters email and password and submits. 2. The system verifies credentials with the auth provider. 3. The system fetches user profiles and preferences. 4. The system routes the user to the home/profile screen.
Extensions:	1. Invalid credentials → System shows Toast error. 2. Fields left empty → System shows Toast error. 3. Network not required → Perform local sign-out and defer server revocation if needed.
Priority:	8

5.6.2 USE CASE DIAGRAM



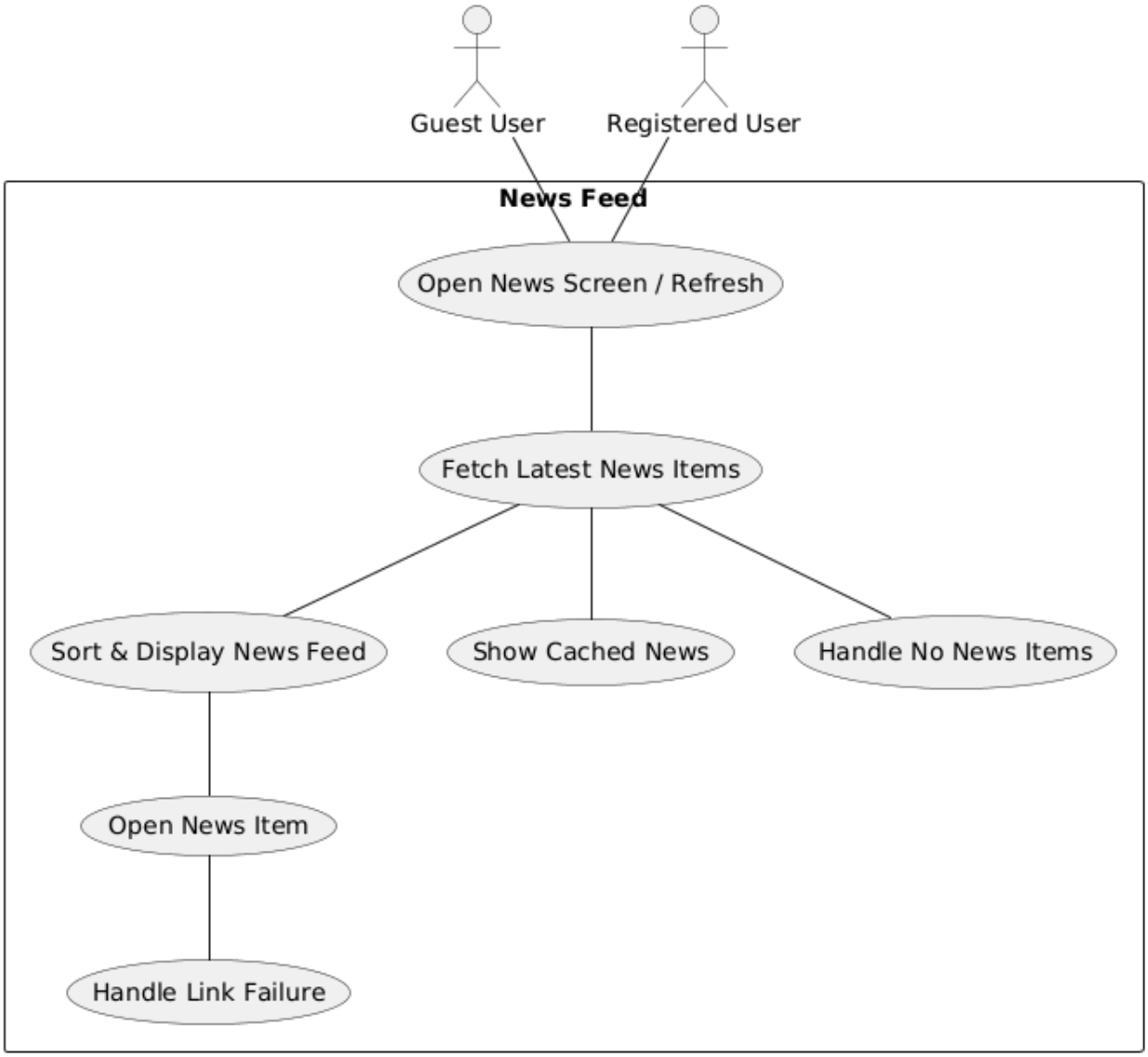
5.6.2 USE CASE SEQUENCE DIAGRAM



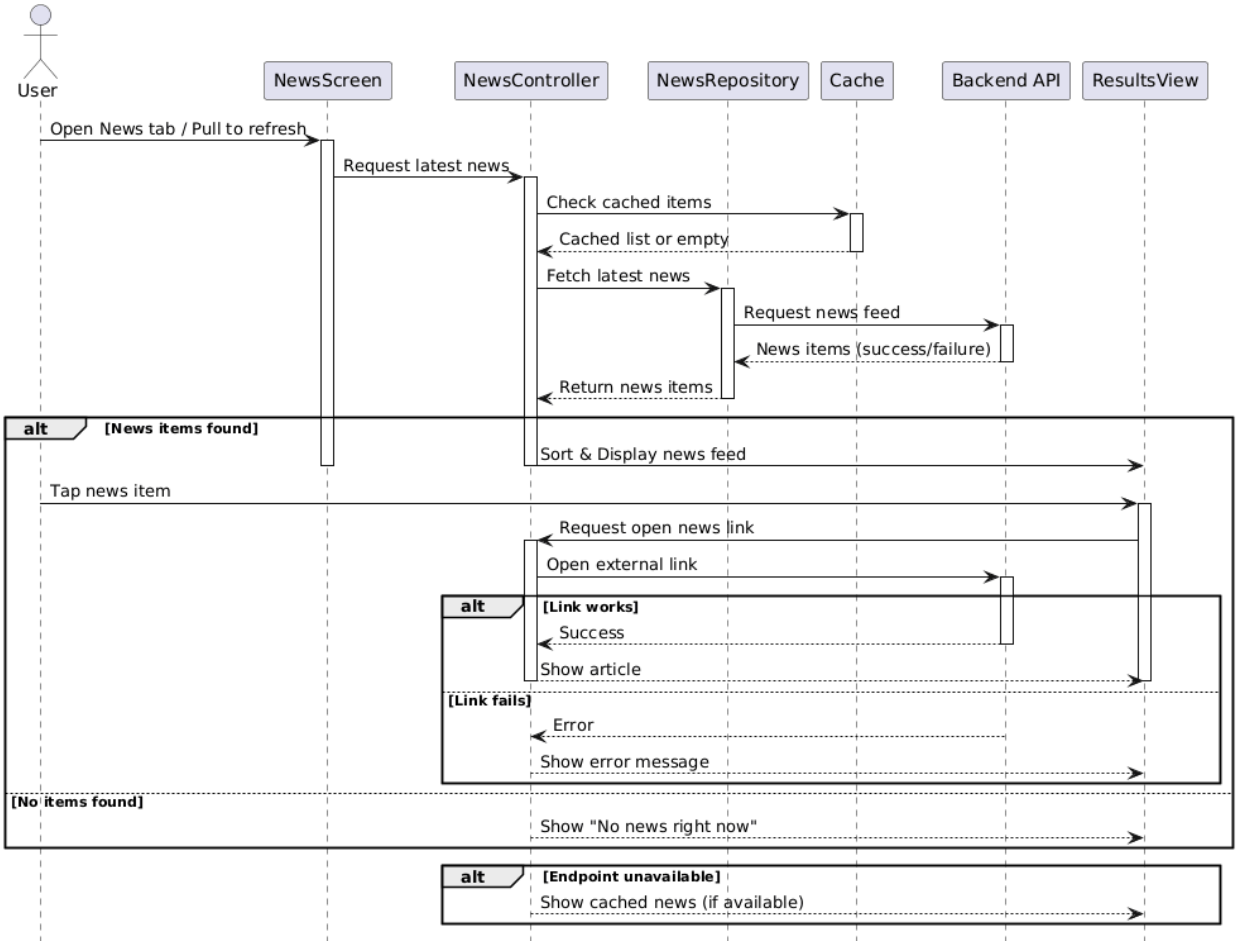
5.7.1 USE CASE

Title:	Live News
Description:	The user views a real-time news feed with headlines and summaries about players, matches, and tournaments.
Actors:	Guest User, Registered User
Stimulus: (Trigger)	The user opens the News screen or pulls to refresh.
Preconditions:	1. News service endpoint available 2. device online (or cached items exist).
Postconditions:	1. A sorted list of recent news items is displayed with links to full articles.
Main Success Scenario:	1. The user navigates to the News tab. 2. The system fetches the latest news items (title, source, timestamp, summary, link). 3. The system sorts by freshness and relevance and displays the feed. 4. The user taps an item to read its details or open the source link.
Extensions:	2a. Endpoint unavailable → System shows cached news and a refresh option. 3a. No items found → System shows “No news right now.” 4a. External link fails → System shows error and suggests retry or different item.
Priority:	6

5.7.2 USE CASE DIAGRAM



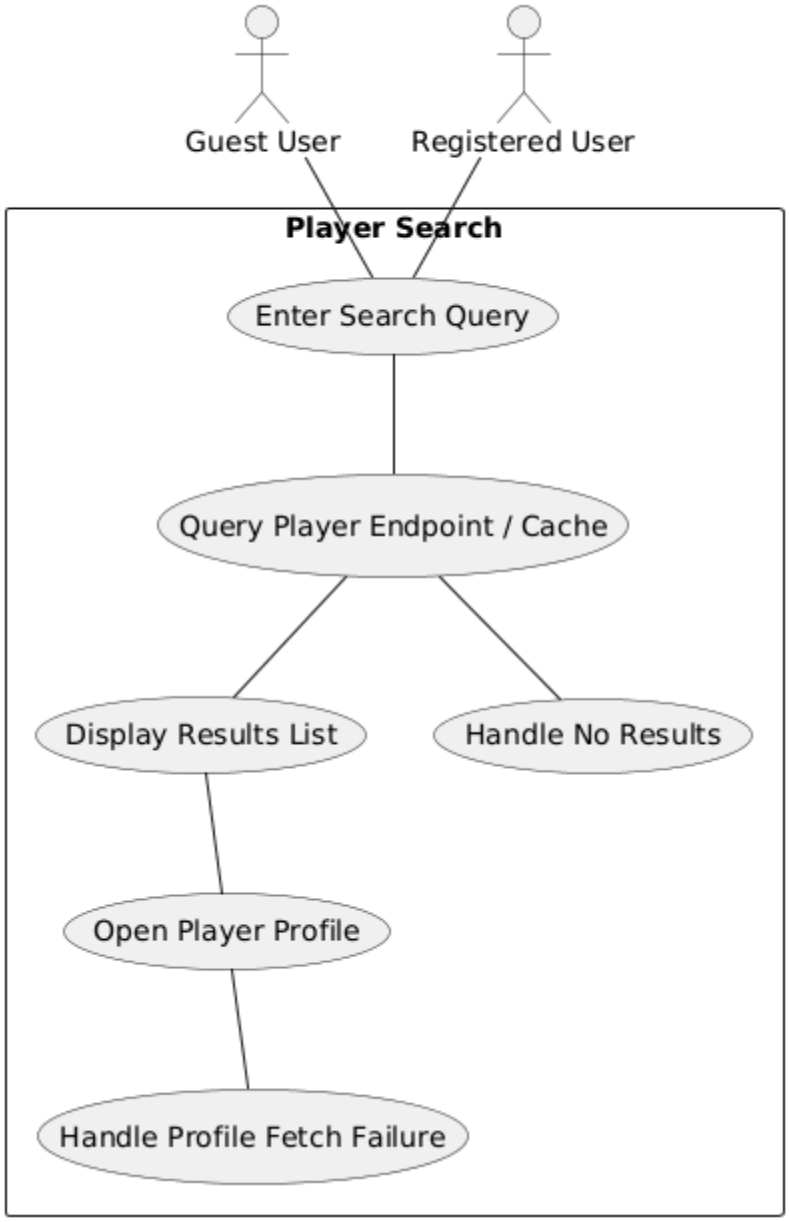
5.7.3 USE CASE SEQUENCE DIAGRAM



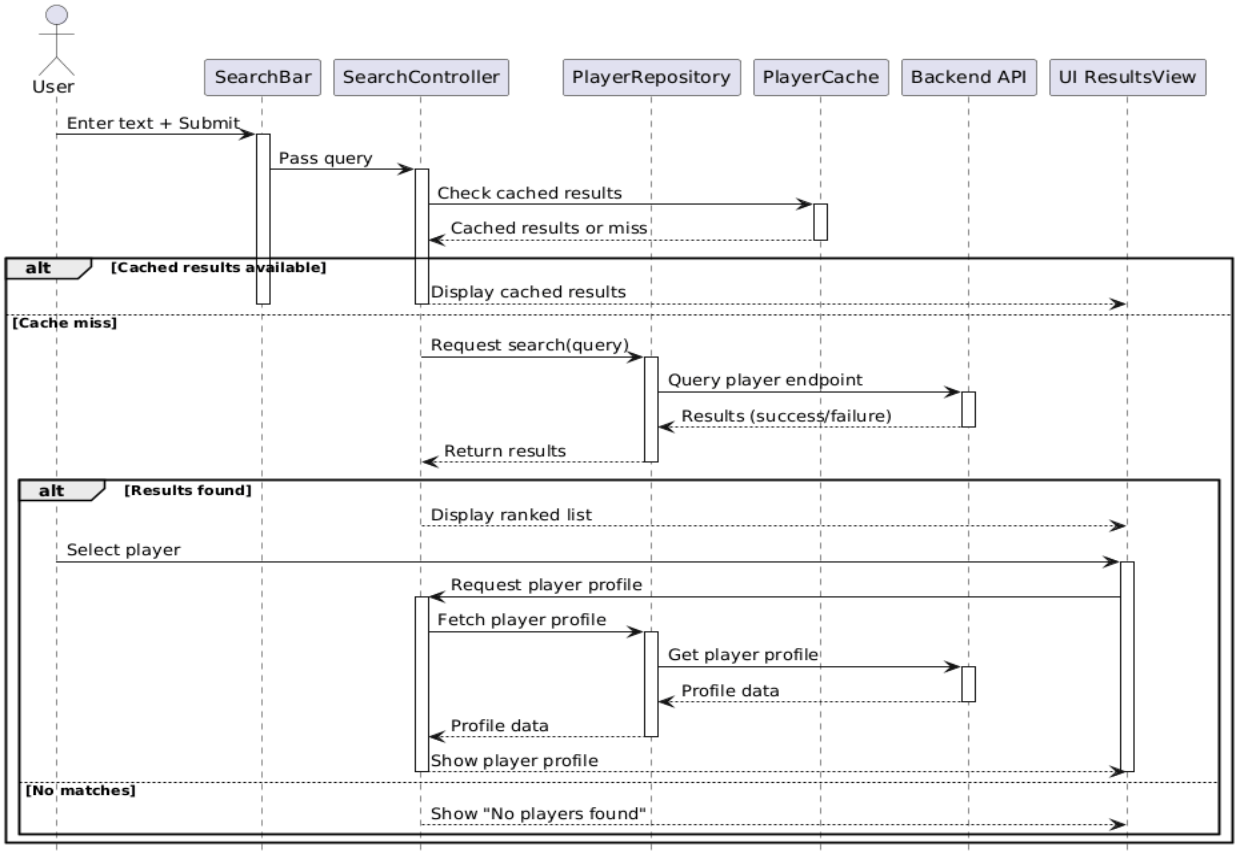
5.8.1 USE CASE

Title:	Search for Players
Description:	The user searches by player name to find profiles.
Actors:	Guest User, Registered User
Stimulus: (Trigger)	The user enters text into the search bar and submits.
Preconditions:	1. Player index is available 2. device is online (unless cached results exist)
Postconditions:	1. Search results are shown or a “no results” message is displayed.
Main Success Scenario:	1. The user types the player name and submits. 2. The system queries the player endpoint or local cache. 3. The system displays a ranked list of matching players. 4. The user selects a player to open the profile.
Extensions:	No matches → System shows “No players found. Check spelling.” Profile fetch fails → System shows retry inline.
Priority:	3

5.8.2 USE CASE DIAGRAM



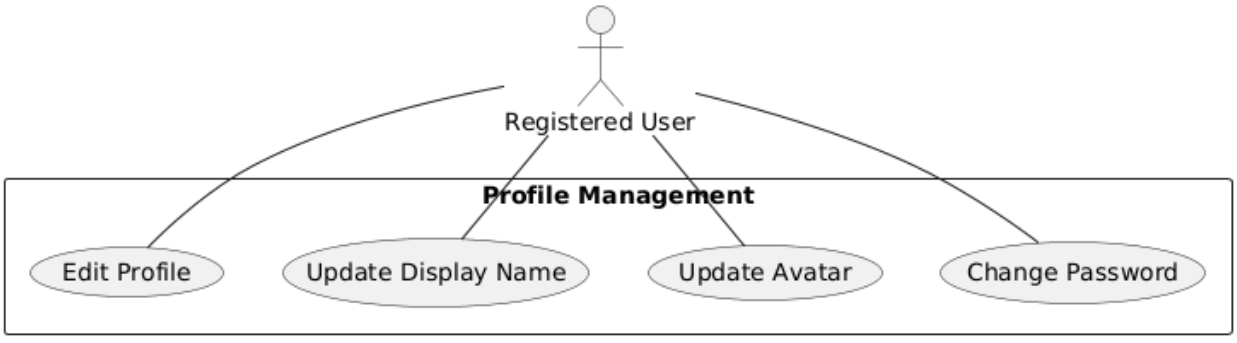
5.8.3 USE CASE SEQUENCE DIAGRAM



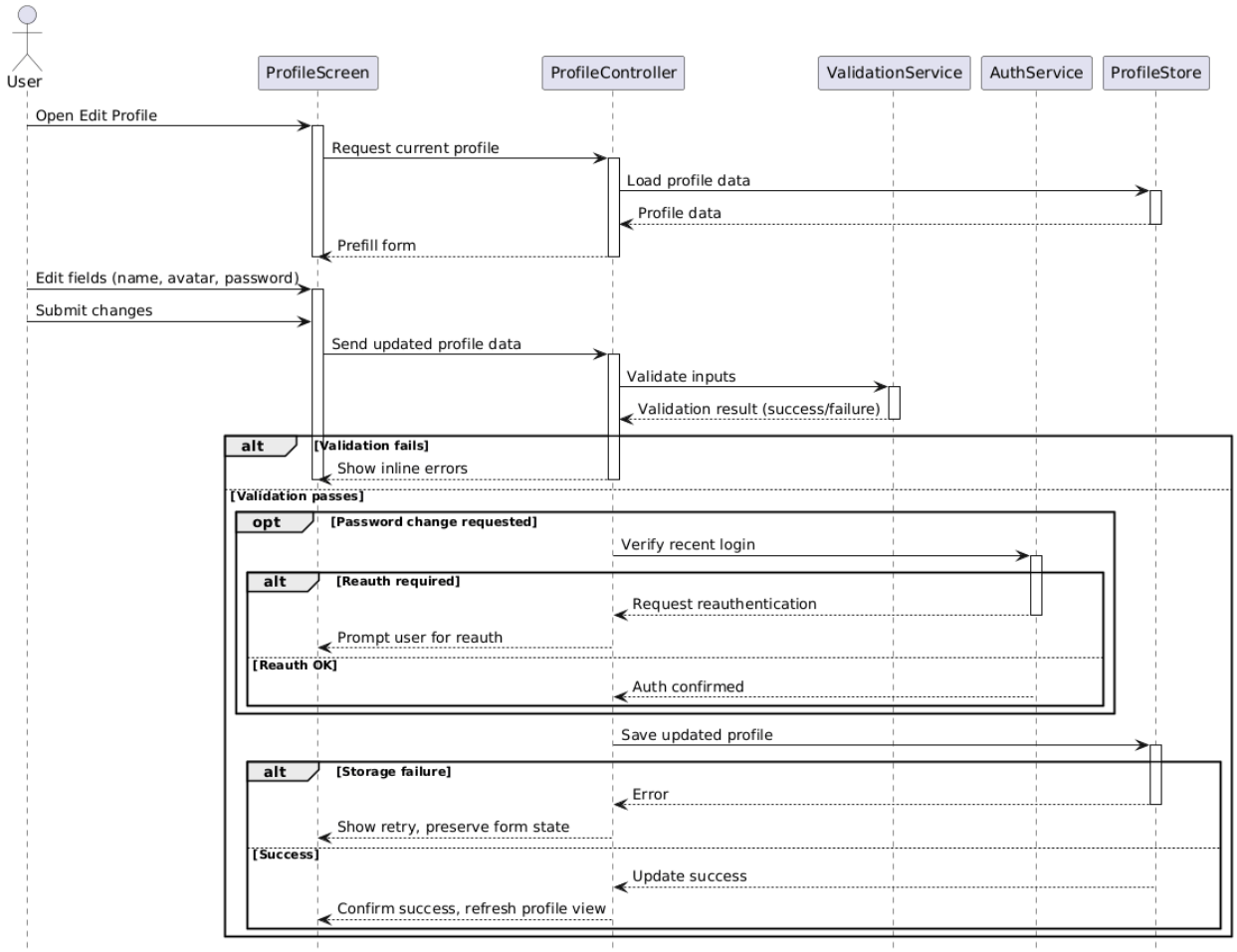
5.9.1 USE CASE

Title:	Manage User Profile
Description:	The user updates account information such as display name, avatar, and password.
Actors:	Registered User
Stimulus:	User opens Profile → Edit, submits changes.
Preconditions:	1. The user is authenticated 2. Profile record exists
Postconditions:	1. Profile changes are validated, persisted, and reflected in the UI.
Main Success Scenario:	1. The user opens Edit Profile. 2. The system loads current profile data into a form. 3. The user edits fields (display name, avatar image; password via secure flow). 4. The user submits changes. 5. The system validates inputs and updates profile store; reauth required for password. 6. The system confirms success and refreshes profile view.
Extensions:	1. Validation fails (e.g., name length, avatar size) → System shows inline errors. 2. Password update requires recent login → System prompts reauthentication. 3. Storage failure → System preserves form state and shows retry.
Priority:	9

5.9.2 USE CASE DIAGRAM



5.9.3 USE CASE SEQUENCE DIAGRAM



6. Diagrams

Fig 6.1 User Interface Overview

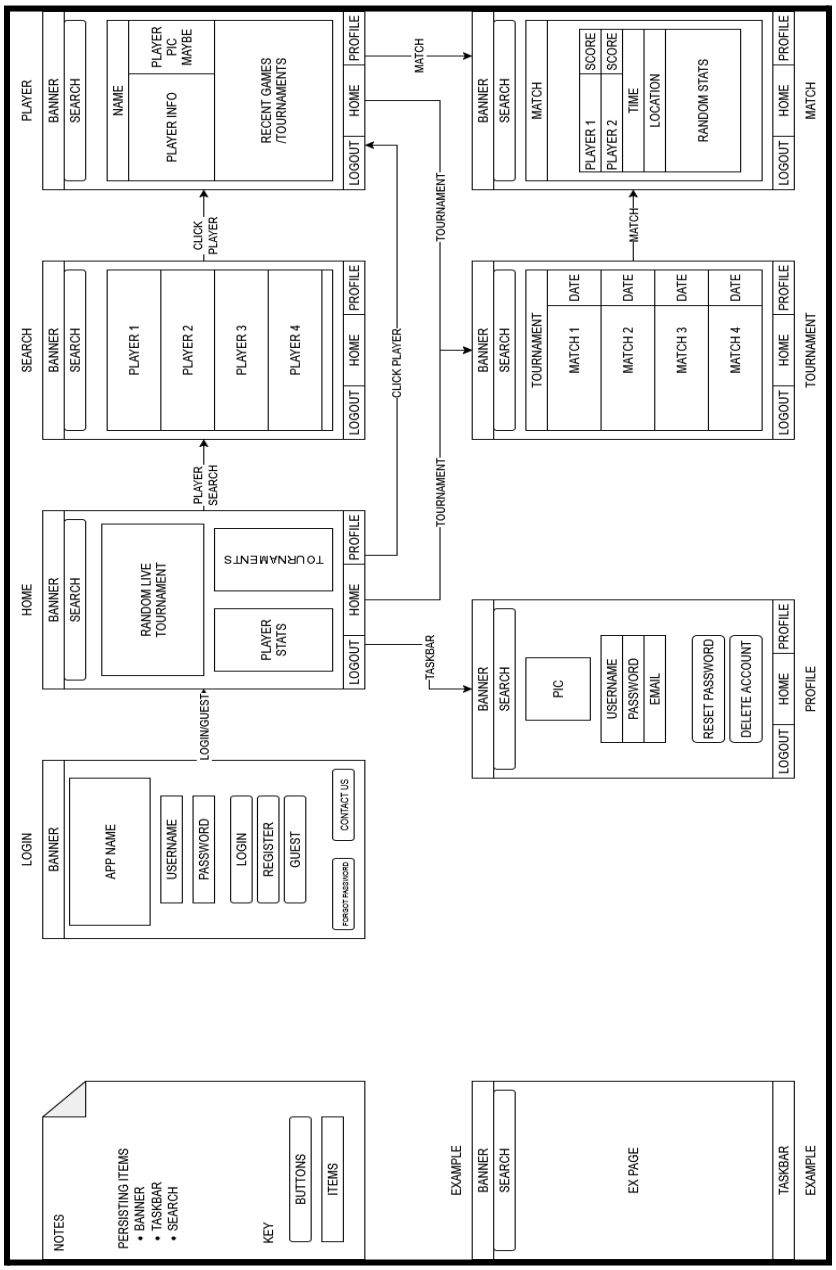


Fig 6.2 Interaction Flowchart

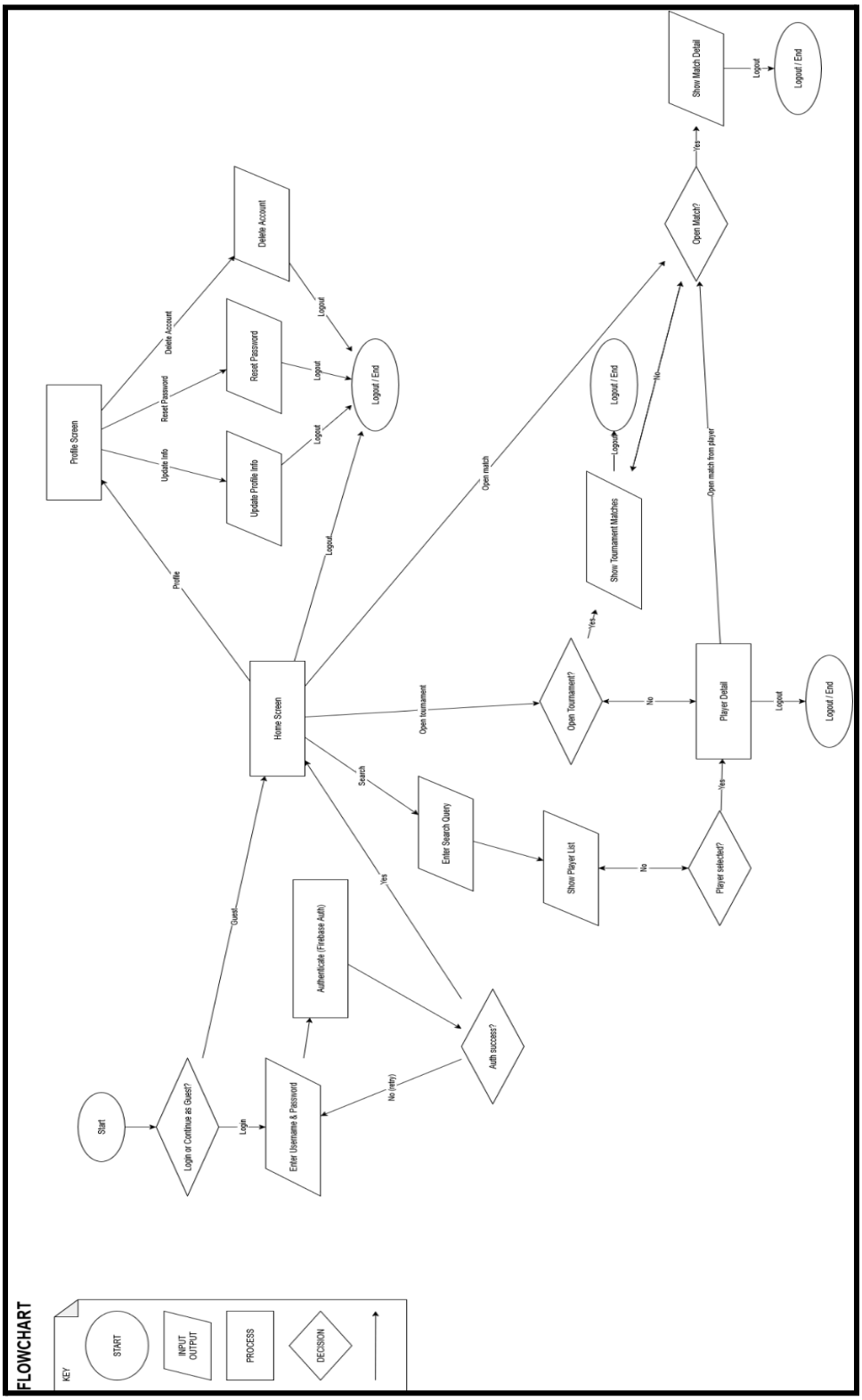


Fig 6.4 Authentication Interface Overview