# San Francisco State University



Video Game Rating Application

Joystick Journal

SW Engineering CSC 648-848

Andrew Dahlstrom (Product Owner), Nadir Ali (Github Master), Aidan Bayer-Calvert (Back-end Lead), Ryan Flannery (Back-end Lead), Kayla Maa (Front-end Lead), Joyce Fu (Scrum Master), Jason Avina ( Scrum Master)

Milestone 2

10/20/2024

Section 01 Team 02

| History Table |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

# Table of Contents

### Cover Page [1](#_z6ne0og04bp5)

### [Table of Contents 2](#_35qsy46zviru)

### [Data Definitions V2 (Aidan) 3](#_40r3tdpxl1mc)

### [Functional Requirements V2 (Kayla and Jason) 5](#_76q7xj2qal59)

### [UI Mockups and UX Flows (Jason and Kayla) 9](#_x5klhqtpbgne)

### [High level Architecture & Database Organization (Ryan) 10](#_r2elsixqduig)

### [High Level UML Diagrams (Joyce) 14](#_5xphd8vrlb7m)

### [Key Risks (Nadir) 15](#_3at9u9s4e0vp)

### [Project management (Andrew) 16](#_e5zocfuhiekm)

# Data Definitions V2 (Aidan)

**User**

* **Definition:** The individual who interacts with the application. Users can register, log in, and post reviews and ratings for games. Unauthenticated users can view reviews but cannot post them.
* **Attributes:**
  + **user\_id**: (Primary Key, INT, AUTO\_INCREMENT) – A unique identifier assigned to each user.
  + **username**: (VARCHAR) – The user’s chosen display name within the application.
  + **email**: (VARCHAR) – The user’s email address, used for authentication and communication.
  + **password\_hash**: (VARCHAR) – The encrypted version of the user’s password for secure storage.
  + **created\_at**: (TIMESTAMP) – The date and time when the user account was created.
  + **updated\_at**: (TIMESTAMP) – The date and time of the user’s latest account update.

**Game**

* **Definition:** The entity representing a video game that can be reviewed or rated by users.
* **Attributes:**
  + **game\_id**: (Primary Key, INT, AUTO\_INCREMENT) – A unique identifier for each game.
  + **title**: (VARCHAR) – The title or name of the video game.
  + **description**: (TEXT) – A summary or description of the game’s content and features.
  + **genre**: (VARCHAR) – The genre the game belongs to (e.g., RPG, Action, Sports).
  + **rating:** (Decimal) - The average user rating for the game.
  + **reviews**: (LIST of review\_ids) – A list of all the review IDs associated with the game.
  + **release\_date**: (DATE) – The date the game was officially released.
  + **cover\_image**: (VARCHAR, optional) – The URL of the game’s cover image.
  + **created\_at**: (TIMESTAMP) – The date and time when the game was added to the system.
  + **updated\_at**: (TIMESTAMP) – The date and time of the latest update to the game’s data.

**Review**

* **Definition:** Represents the feedback given by a user for a particular game, including both rating and text feedback.
* **Attributes:**
  + **review\_id**: (Primary Key, INT, AUTO\_INCREMENT) – A unique identifier for each review.
  + **user\_id**: (Foreign Key, INT) – The ID of the user who submitted the review.
  + **game\_id**: (Foreign Key, INT) – The ID of the game being reviewed, linking reviews to games.
  + **rating**: (INT) – The user’s rating for the game on a scale of 1-5.
  + **review\_text**: (TEXT, optional) – A textual description or feedback on the game provided by the user.
  + **created\_at**: (TIMESTAMP) – The date and time when the review was created.
  + **updated\_at**: (TIMESTAMP) – The date and time when the review was last updated.

# Functional Requirements V2 (Kayla and Jason)

#### **FR-1001: User Authentication**

* **Description:** Users will be able to log in using their username and password to access personalized features. If credentials are incorrect, the user will be prompted to retry or reset their password.
* **Priority:** High
* **User Story Reference:** As a user (**Hardcore Gamer**, **Casual Gamer**, **Newcomer**, **Experienced Reviewer**), I want to log in with my username and password to access my account. This helps keep my information secure and gives me access to personalized features.

**FR-1001.1. Validate Username and Password**

Description: The database shall validate that the username and password meets requirements, like if the password were to have a specific length, or the username is taken.

Priority: 1 (must-have)

**FR-1001.2. Display Failed Login**

Description: There will be a message that will allow the user know that the user has inputted an incorrect username or password.

Priority: 1 (must-have)

#### **FR-1102: Search for Games**

* **Description:** Users can search for video games by title. Search results will display the game’s cover image, rating, and a link to reviews.
* **Priority:** High
* **User Story Reference:** As a gamer (**Hardcore Gamer**, **Casual Gamer**, **Newcomer**), I want to search for a video game that I am interested in and read reviews and feedback from other gamers to help decide if it's worth playing.

**FR-1102.1. Search by Title**

Description: User shall be able to search for a video game by entering the exact title.

Priority: 1 (must have)

**FR-1102.2. Search by Keywords**

Description: User shall be able to search for a video game by entering keywords from the title.

Priority: 1 (must have)

#### **FR-1103: Game Review**

* **Description:** Users can leave a review on any video game in the system, including a written review and rating. The system should allow users to edit their reviews at a later time.
* **Priority:** High
* **User Story Reference:** As a user (**Hardcore Gamer**, **Experienced Reviewer**), I want to leave and edit reviews on video games so I can share my thoughts and help others decide whether the game is worth playing.

**FR-1103.1. Edit Review**

Description: Users shall be able to edit their reviews even after submitting/publishing.

Priority: 1 (must have)

**FR-1103.2. Delete Review**

Description: Users shall be able to delete their reviews even after submitting/publishing.

Priority: 1 (must have)

#### **FR-1104: Game Rating**

* **Description:** Users can rate video games on a scale of 1 to 5.
* **Priority:** High
* **User Story Reference:** As a user (**Hardcore Gamer**, **Casual Gamer**, **Newcomer**, **Experienced Reviewer**), I want to rate video games on a 5-point scale so that I can express my opinion about the game and provide guidance for those considering playing it.

**FR-1104.1. 5 Star System**

Description: System lets the user rate the specified game on a scale of 1 to 5, lowest to highest.

Priority: 1 (must have)

**FR-1104.2. Average Rating**

Description: The system shall be able to calculate in order to find the average ratings of the video game depending on the user’s who have submitted a rating.

Priority: 1 (must have)

#### **FR-1105: Browse for Video Games**

* **Description:** Users can browse for video games by genre such as RPG, action, or sports. Results will show the game title, cover image, and average user rating.
* **Priority:** High
* **User Story Reference:** As a user (**Casual Gamer**, **Newcomer**, **Hardcore Gamer**), I want to browse video games by category so I can quickly discover games that match my interests or explore new ones.

**FR-1105.1. Browse by Category**

Description: Users shall be able to browse video games by categories to find their reviews/ratings. Categories are split based on genres of games. (can be added)

Priority: 1 (must have)

#### **FR-1106: User Profile Page**

* **Description:** Users can view and edit their profile information, such as username, and email. The page will also display a list of their submitted reviews.
* **Priority:** Low
* **User Story Reference:** As a user (**Experienced Reviewer**, **Hardcore Gamer**), I want to edit my profile and view my submitted reviews so I can track my contributions and manage my personal information.

**FR-1106.1. Editing Profile Page**

Description: Users shall be able to edit all profile information displayed to the public like usernames.

Priority: 1 (must have)

#### **FR-1107: AI Game Recommendation**

* **Description:** The system will recommend video games to users based on their past searches, reviews, and ratings. These suggestions will consider user preferences for genres, keywords, and previously rated games.
* **Priority:** Medium
* **User Story Reference:** As a user (**Hardcore Gamer**, **Casual Gamer**, **Newcomer**), I want the system to recommend games based on my past searches and reviews so I can easily find games that suit my preferences.

**FR-1107.1. Re-generate Game Recommendations**

Description: Users shall be able to click a button called “Generate New Suggestions” to refresh and generate a new set of AI Game Recommendations.

Priority: 2 (desired)

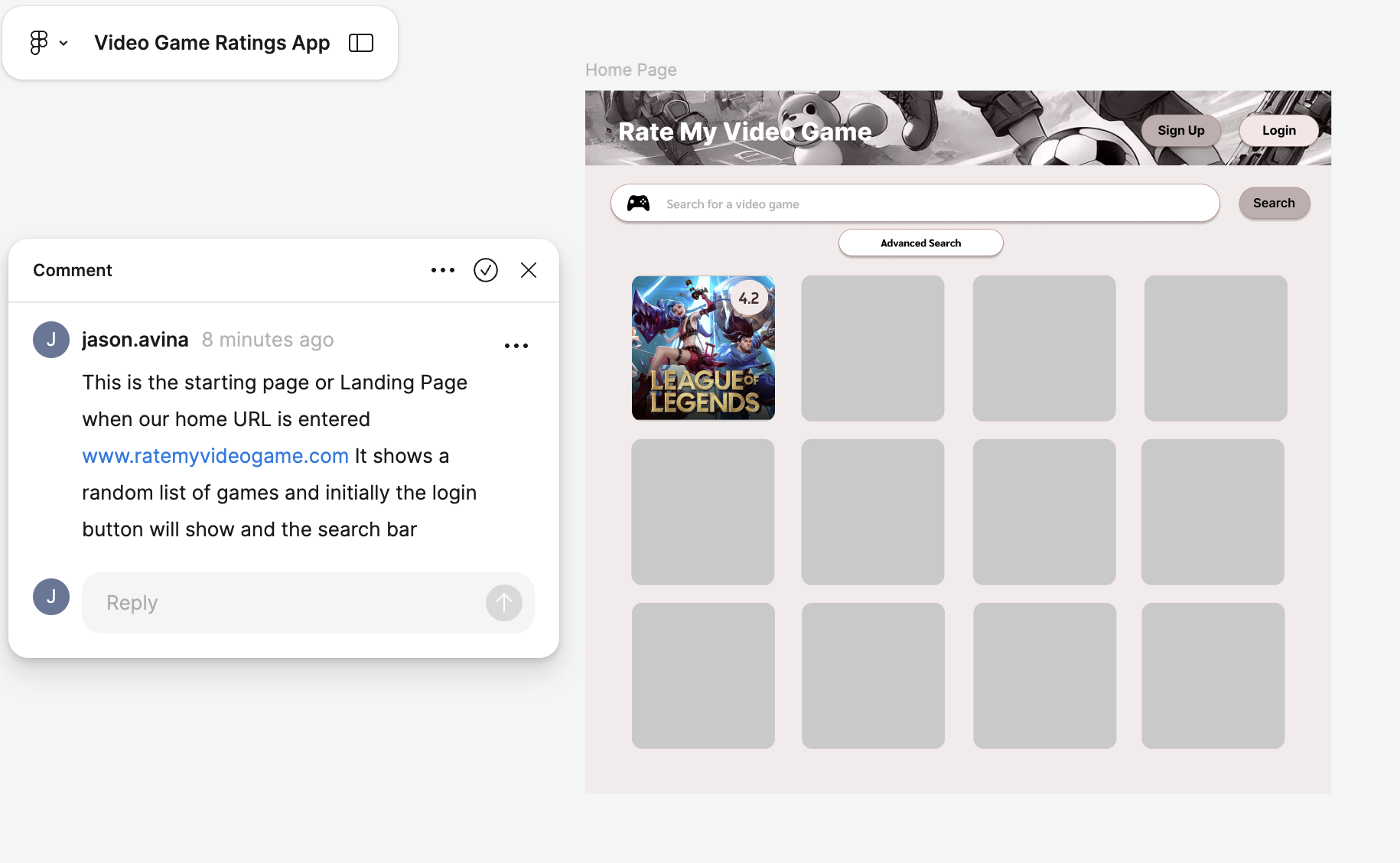
**FR-1107.2. AI Improvement Over Time**

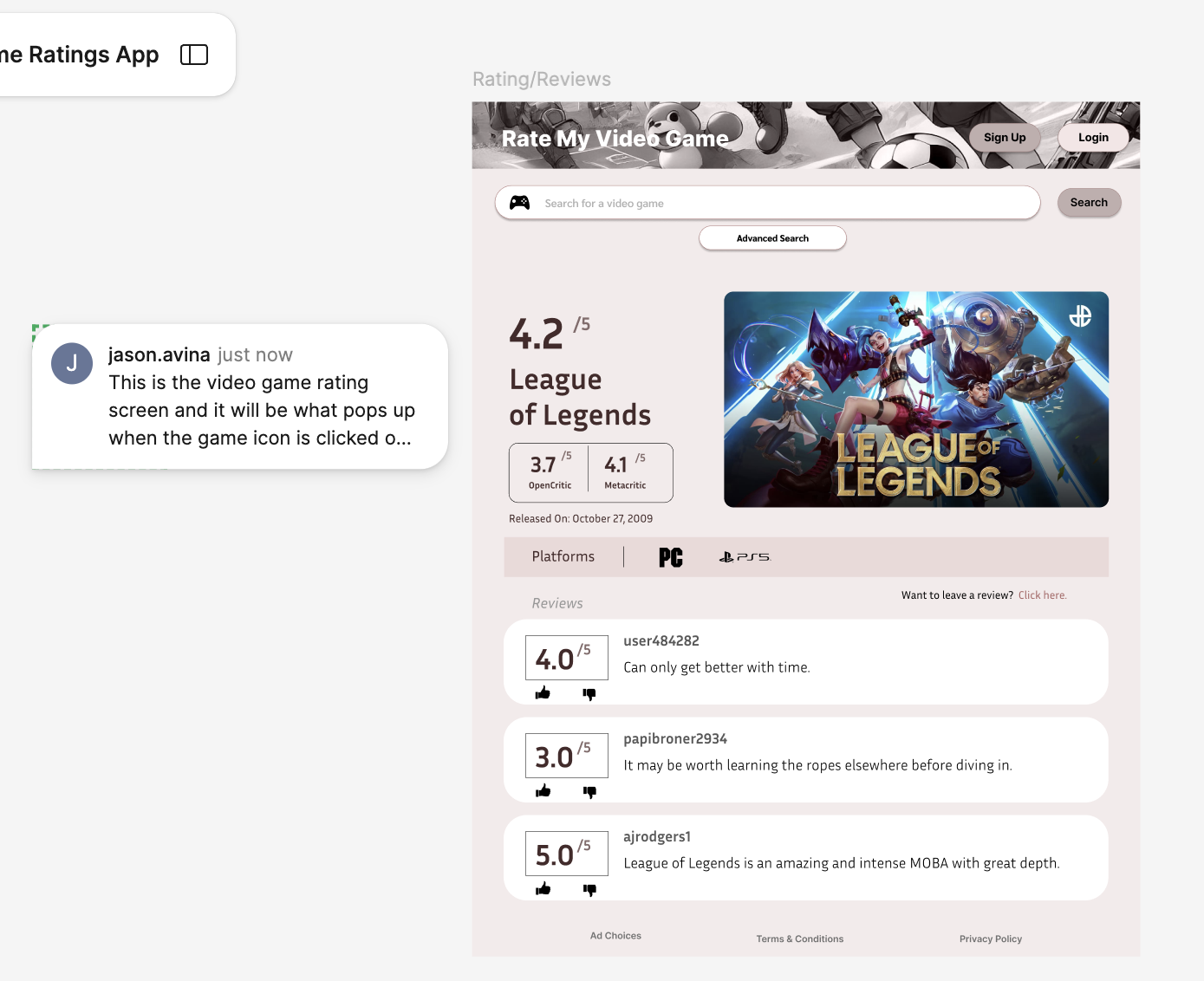
Description: AI shall improve recommendations based on the user’s interactions and learning more about the user’s interests like past searches, common categories, etc.

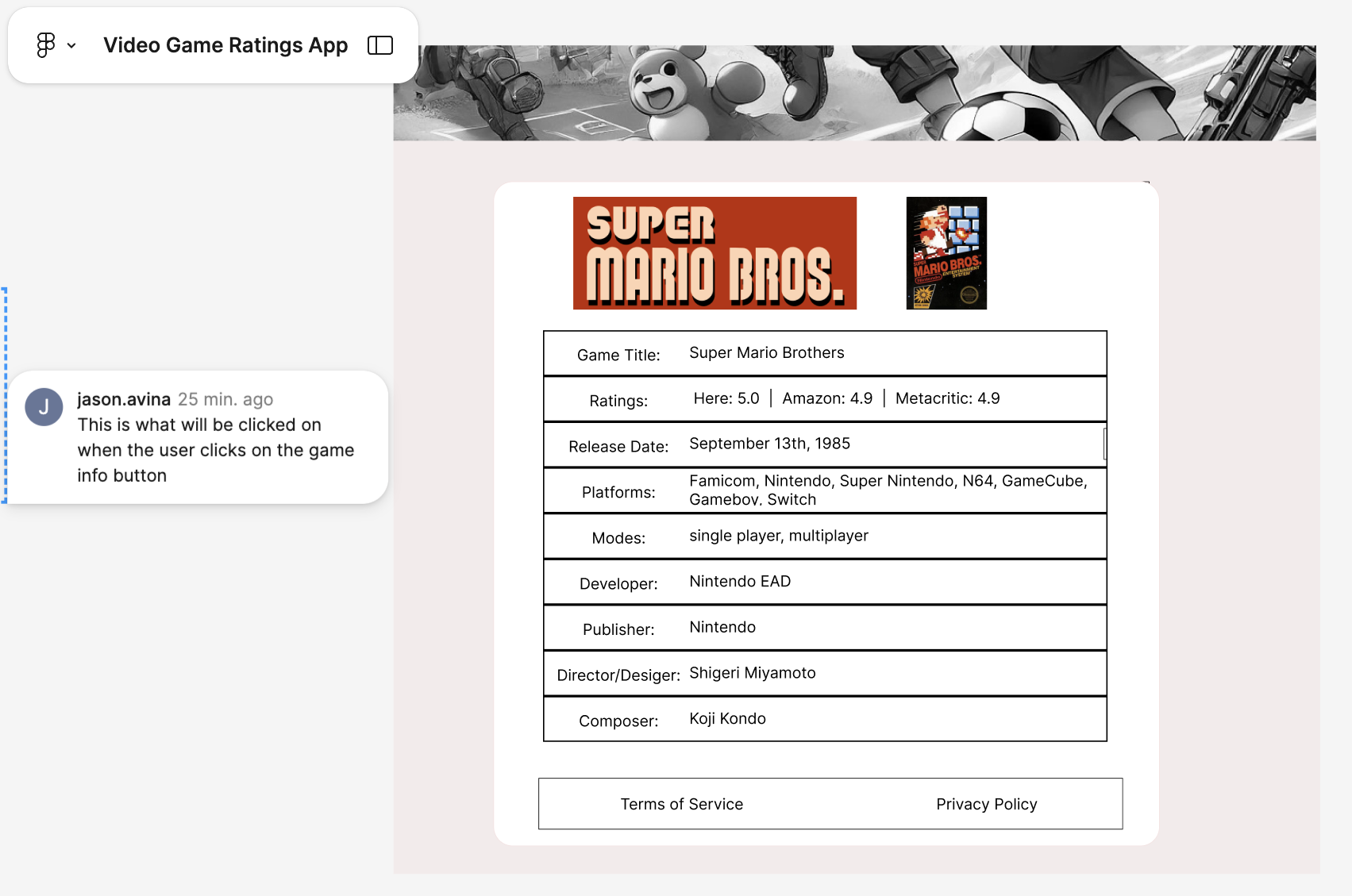
Priority: 3 (opportunistic)

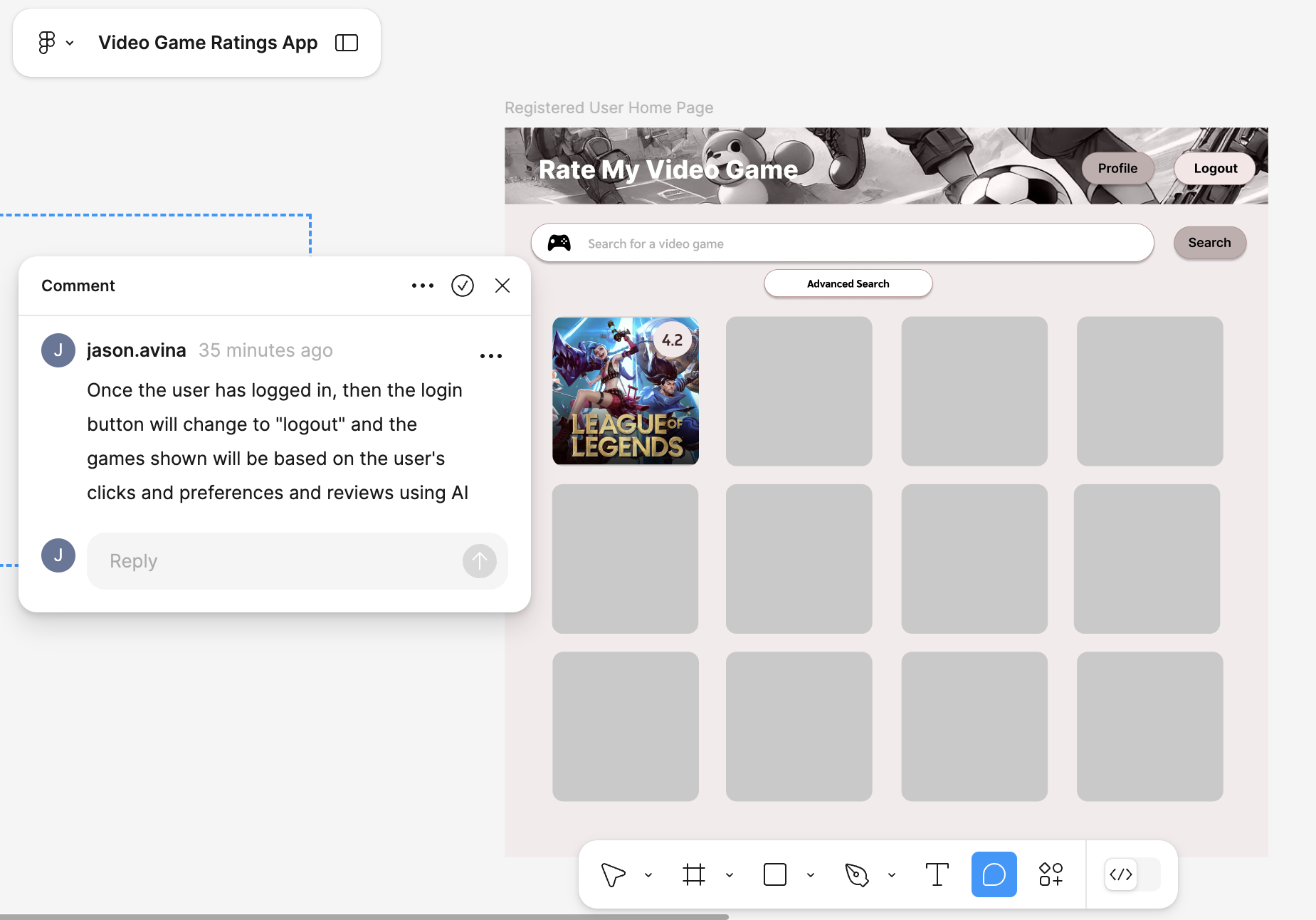
# 

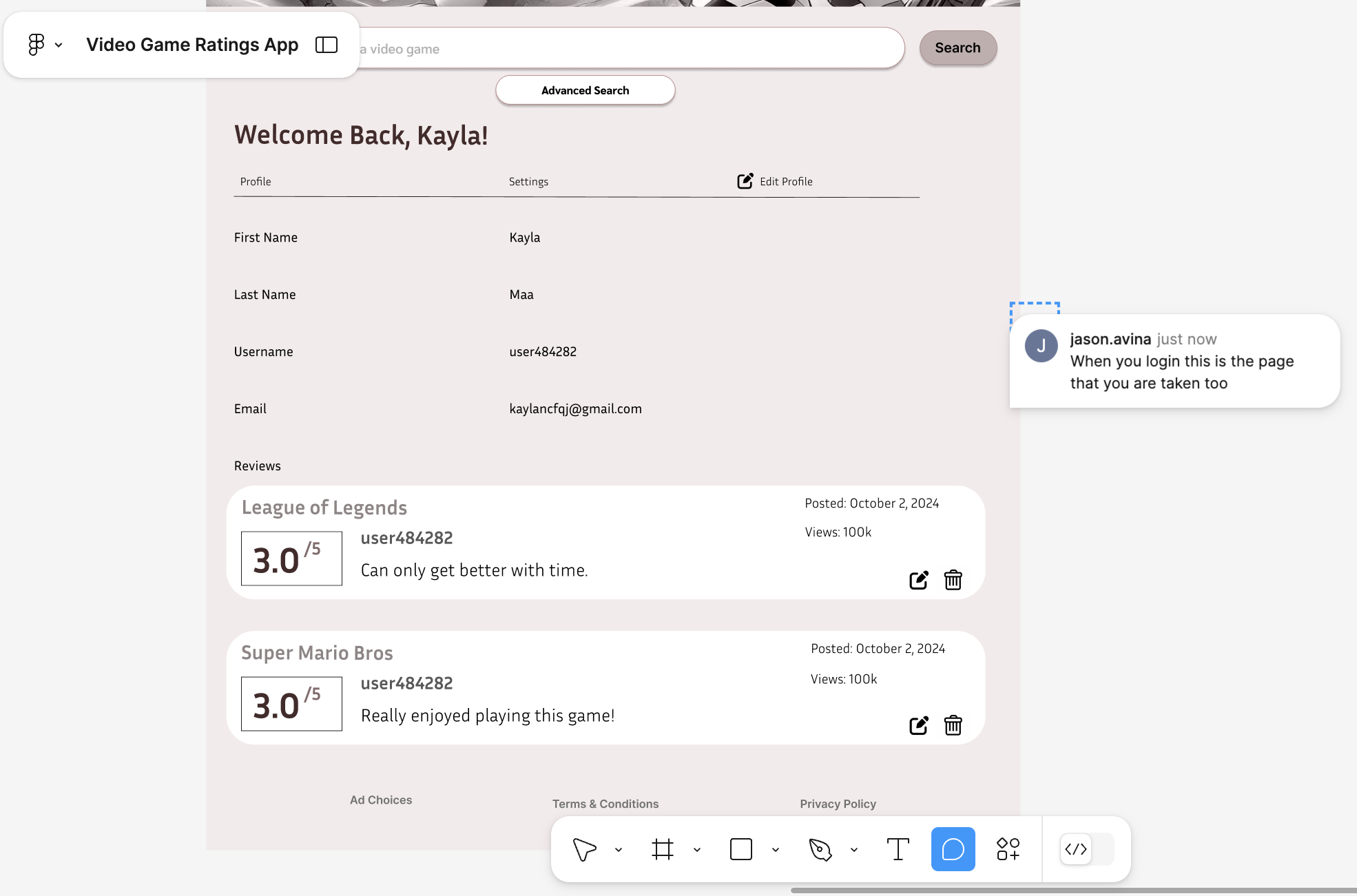
# UI Mockups and UX Flows (Jason and Kayla)

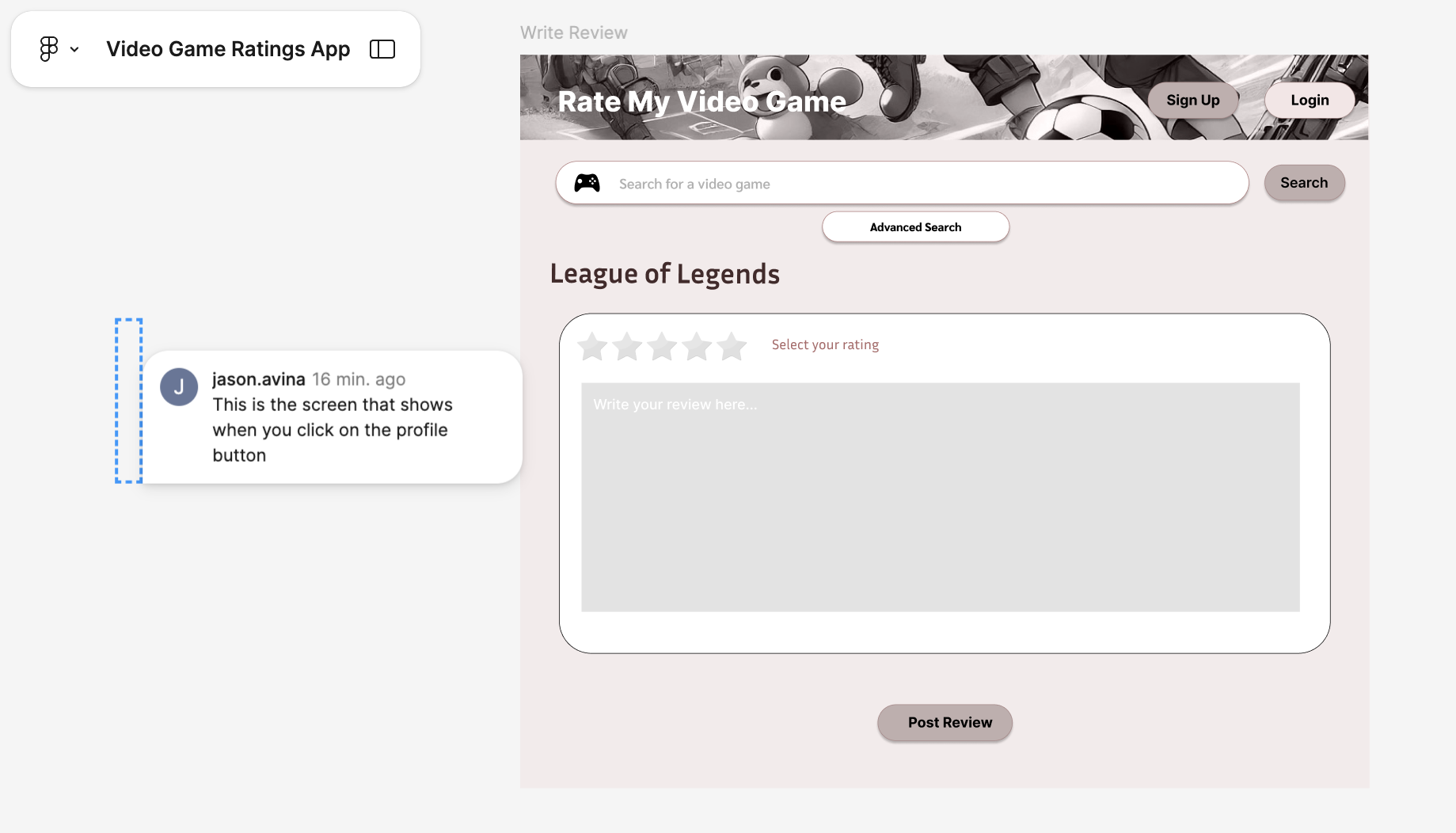


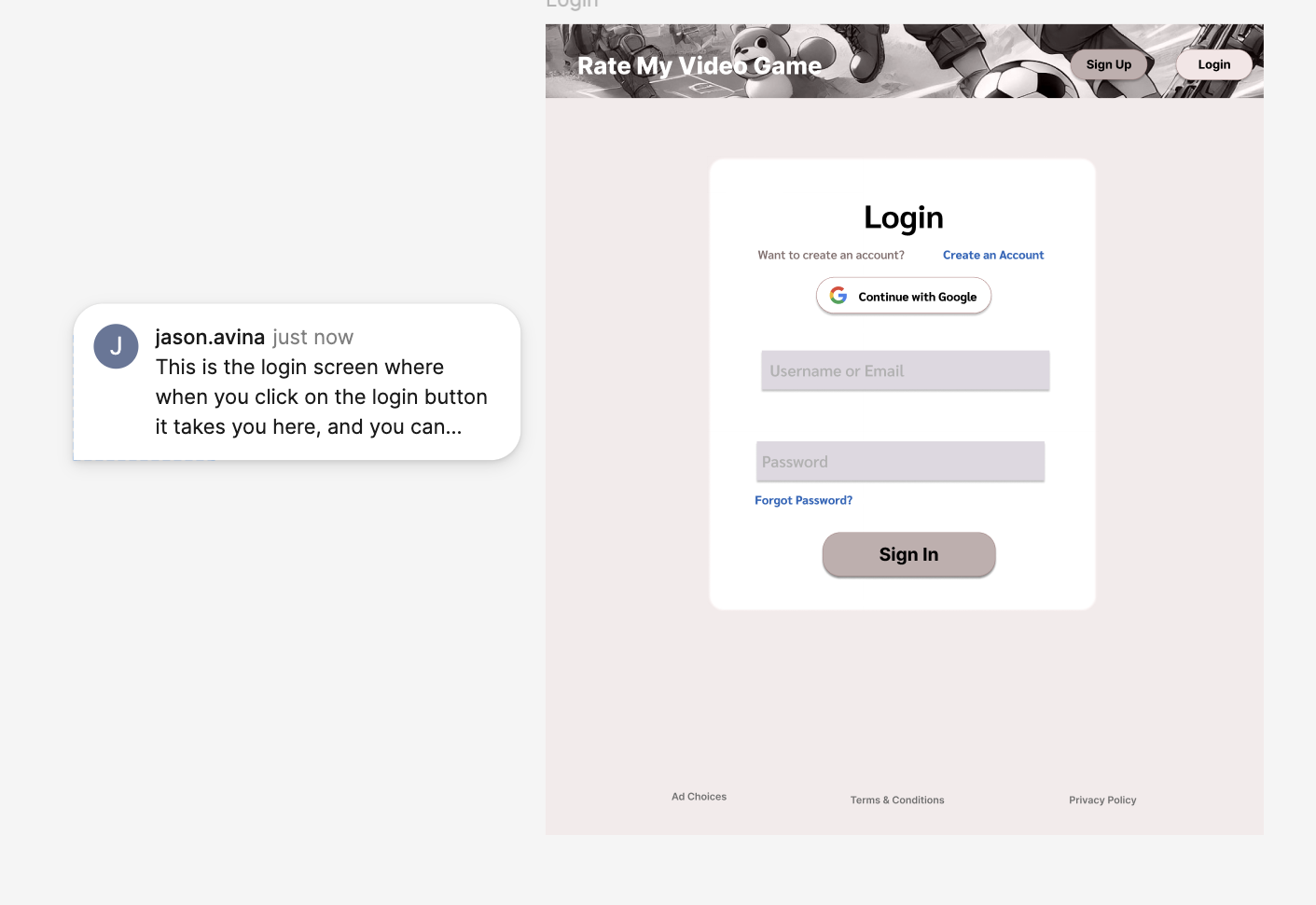


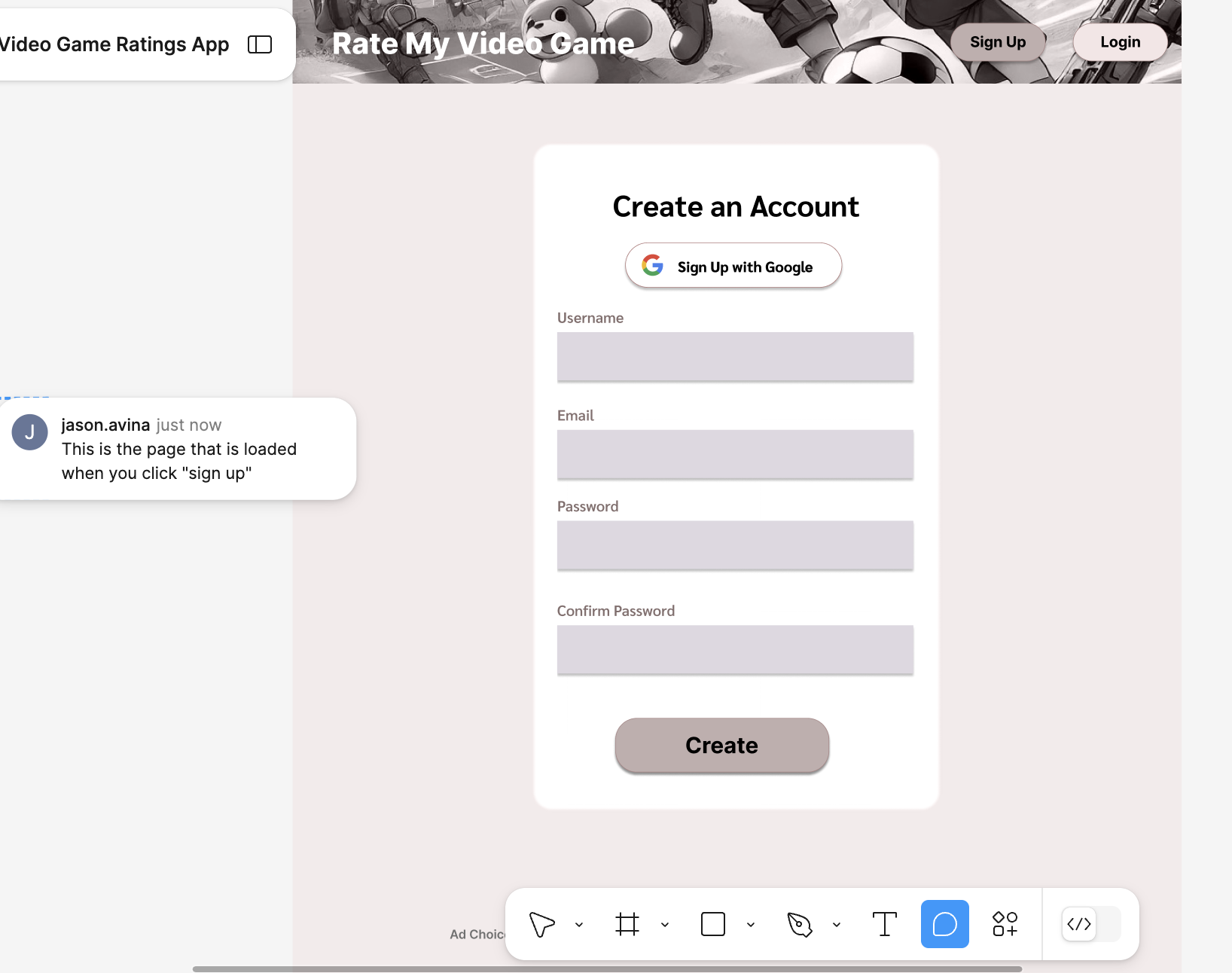










  
**Figma Link:** <https://www.figma.com/design/uIvpsuPtJgaou9RQvrtwEr/Video-Game-Ratings-App?node-id=82-137&t=nmg7458HGQJpQ2A1-1>

# High level Architecture & Database Organization (Ryan)

### Database Organization:

#### **1. Users Table**

The users table stores information about each registered user of the application, including credentials and profile details.

* **Columns**:
  + user\_id (Primary Key, INT, AUTO\_INCREMENT): A unique identifier for each user.
  + username (VARCHAR): The user’s display name.
  + email (VARCHAR): The user’s email address for login and communication.
  + password\_hash (VARCHAR): The securely encrypted version of the user's password.
  + created\_at (TIMESTAMP): The date and time when the user account was created.
  + updated\_at (TIMESTAMP): The date and time of the user's latest account update.

#### **2. Games Table**

The games table stores detailed information about the video games that users can review and rate.

* **Columns**:
  + game\_id (Primary Key, INT, AUTO\_INCREMENT): A unique identifier for each game.
  + title (VARCHAR): The title or name of the video game.
  + description (TEXT): A summary or description of the game's content and features.
  + genre (VARCHAR): The genre of the game (e.g., RPG, Action, Sports).
  + rating (DECIMAL): The average user rating for the game, calculated based on user reviews.
  + reviews (LIST of review\_ids): A list of review IDs associated with the game (for future expansion).
  + release\_date (DATE): The official release date of the game.
  + cover\_image (VARCHAR, optional): The URL for the game’s cover image.
  + created\_at (TIMESTAMP): The date and time when the game was added to the system.
  + updated\_at (TIMESTAMP): The date and time when the game's details were last updated.

#### **3. Reviews Table**

The reviews table stores the feedback that users provide for games, including both a rating and optional written comments.

* **Columns**:
  + review\_id (Primary Key, INT, AUTO\_INCREMENT): A unique identifier for each review.
  + user\_id (Foreign Key, INT): The ID of the user who submitted the review.
  + game\_id (Foreign Key, INT): The ID of the game being reviewed.
  + rating (INT): The user’s rating of the game, on a scale from 1 to 5.
  + review\_text (TEXT, optional): Optional text feedback provided by the user about the game.
  + created\_at (TIMESTAMP): The date and time when the review was created.
  + updated\_at (TIMESTAMP): The date and time when the review was last updated.

### Add/Delete/Search Architecture:

#### **1. Users Table**

* **Add**:
  + Users are added when a new user registers for the application. The backend inserts a new row with the user's username, email, password\_hash, and timestamps (created\_at and updated\_at).
* **Search**:
  + The application searches for a user during login by querying the email field to verify their identity. User data is also retrieved when accessing a user’s profile.
* **Display**:
  + User information such as username and created\_at may be displayed on the profile page or review section, but sensitive information such as email and password\_hash is never displayed to other users.

#### **2. Games Table**

* **Add**:
  + Video game entries are added by administrators or during the game's data import. Information such as title, description, genre, release\_date, and cover\_image is stored.
* **Search**:
  + Users can search for games by title or genre via SQL queries on the title or genre fields. Advanced search features can include searching by rating.
* **Display**:
  + Game details such as title, description, rating, and release\_date are displayed on the frontend search page or game detail page.

#### **3. Reviews Table**

* **Add**:
  + Reviews are added when users submit feedback and a rating for a game. Each review is tied to a user\_id and game\_id, and contains the rating, optional review\_text, and timestamps (created\_at, updated\_at).
* **Search**:
  + Reviews are queried based on game\_id when users view the reviews for a specific game. Reviews can also be filtered or sorted by rating.
* **Display**:
  + Reviews, including the rating and review\_text, are displayed on the game’s detail page for other users to view.

### APIs:

## **1. Communication Method: REST API**

The frontend and backend interface via RESTful API calls over HTTP. The backend, built with Express.js, exposes various endpoints that the React frontend can access to retrieve and modify data.

## **2. Request/Response Format**

* **Request Format**: All requests from the frontend to the backend use JSON format. Each request includes the necessary headers, such as content-type (application/json), and optionally an authorization token for secured routes.
* **Response Format**: The backend returns responses in JSON format, with standard HTTP status codes to indicate success or failure (e.g., 200 OK, 404 Not Found, 401 Unauthorized).

## **3. Authentication and Authorization**

* **Login Flow**: The frontend sends a POST request to the /login endpoint with user credentials. If successful, the backend returns a JWT (JSON Web Token). The frontend stores this token (e.g., in localStorage) and sends it in the Authorization header for future requests to protected routes.  
  **Example Request**:

POST /login { "email": "user@example.com", "password": "password123" }

**Response**:

{ "token": "JWT\_TOKEN" }

**Protected Routes**: Routes that retrieve or modify user-specific data, such as creating reviews or viewing the user's profile, require the JWT for authorization.

**Example Authorized Request**:

GET /user/profile Headers: { "Authorization": "Bearer JWT\_TOKEN" }

## **4. API Endpoints Overview**

### User Accounts

| **HTTP Method** | **URL** | **Purpose** | **Request Body** | **Response** |
| --- | --- | --- | --- | --- |
| POST | /login | Login user | {email, password} | {token} |
| POST | /signup | Register new user | {name, email, password} | {message} |
| GET | /user/profile | Get user profile | Authorization header | {user data} |

### Video Game Search and Ratings

| **HTTP Method** | **URL** | **Purpose** | **Request Body** | **Response** |
| --- | --- | --- | --- | --- |
| GET | /games/search | Search games by title or genre | {query, genre} (as query params) | {games} (filtered results) |
| POST | /reviews | Submit a new game review | {userId, gameId, rating, text} | {message, reviewId} |
| GET | /reviews/ | Get reviews for a game | N/A | {reviews} |
| POST | /ratings | Submit a rating for a game | {userId, gameId, rating} | {message} |
| GET | /ratings/ | Get average rating for a game | N/A | {averageRating} |

## **5. Frontend-Backend Interaction Flow**

### Login and User Profile Retrieval

1. User enters credentials in the frontend login form.
2. The frontend sends a POST request to /login.
3. The backend validates credentials and returns a JWT token.
4. The frontend stores the JWT token and uses it for subsequent authenticated requests (e.g., fetching the user's profile).

### Game Search

1. User enters a game title in the frontend search bar.
2. The frontend sends a GET request to /games/search with the search query and any filters (e.g., genre).
3. The backend queries the database for games matching the search query and returns the results.
4. The frontend displays the game search results to the user.

### Filter Games by Rating

1. User selects a rating from the advanced search dropdown.
2. The frontend sends a GET request to /games/search with the search query and the rating filter.
3. The backend returns only the games that match the search query and have the specified rating.
4. The frontend displays the filtered games on the search page.

### Game Review and Rating Submission

1. User selects a game and submits a review with multiple metrics (e.g., rating, review text).
2. The frontend sends a POST request to /reviews with the review data.
3. The backend saves the review and returns a success message.
4. The frontend sends a rating to /ratings to update the game’s overall rating.

### Fetching Game Reviews

1. When a user navigates to a game’s detail page, the frontend sends a GET request to /reviews/:gameId.
2. The backend responds with all reviews for that specific game, and the frontend displays them.

## **6. Error Handling**

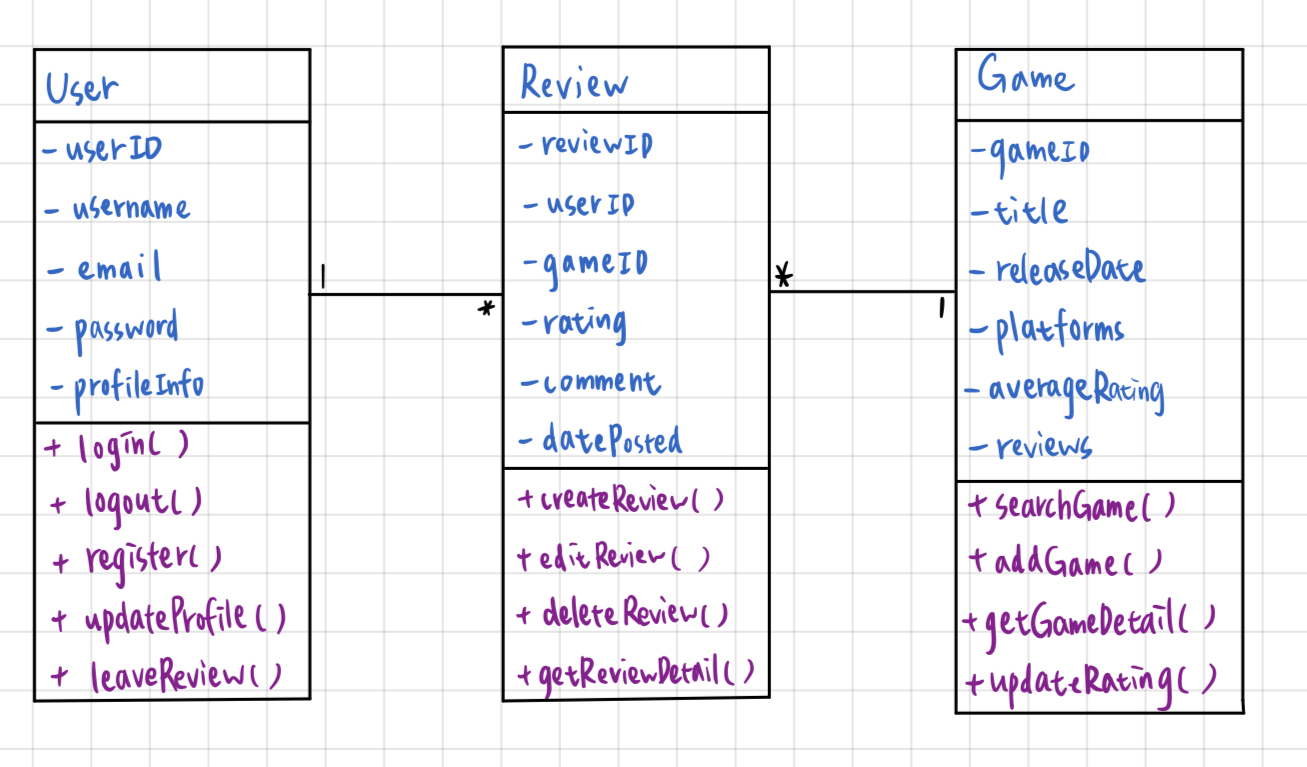
* **Frontend**: The frontend handles errors such as invalid login credentials, failed review submissions, or missing fields by displaying appropriate error messages to the user.
* **Backend**: The backend sends error messages along with corresponding HTTP status codes (e.g., 401 Unauthorized, 404 Not Found, 500 Internal Server Error). The frontend interprets and displays these errors to the user.

## **7. Security Considerations**

* **JWT Authentication**: JWT tokens are used for securing routes and are checked by the backend for each request to a protected resource.
* **Password Hashing**: User passwords are securely hashed and stored using Bcrypt.
* **Rate Limiting**: Basic rate limiting is applied to API endpoints to prevent abuse and reduce the chance of denial-of-service attacks.

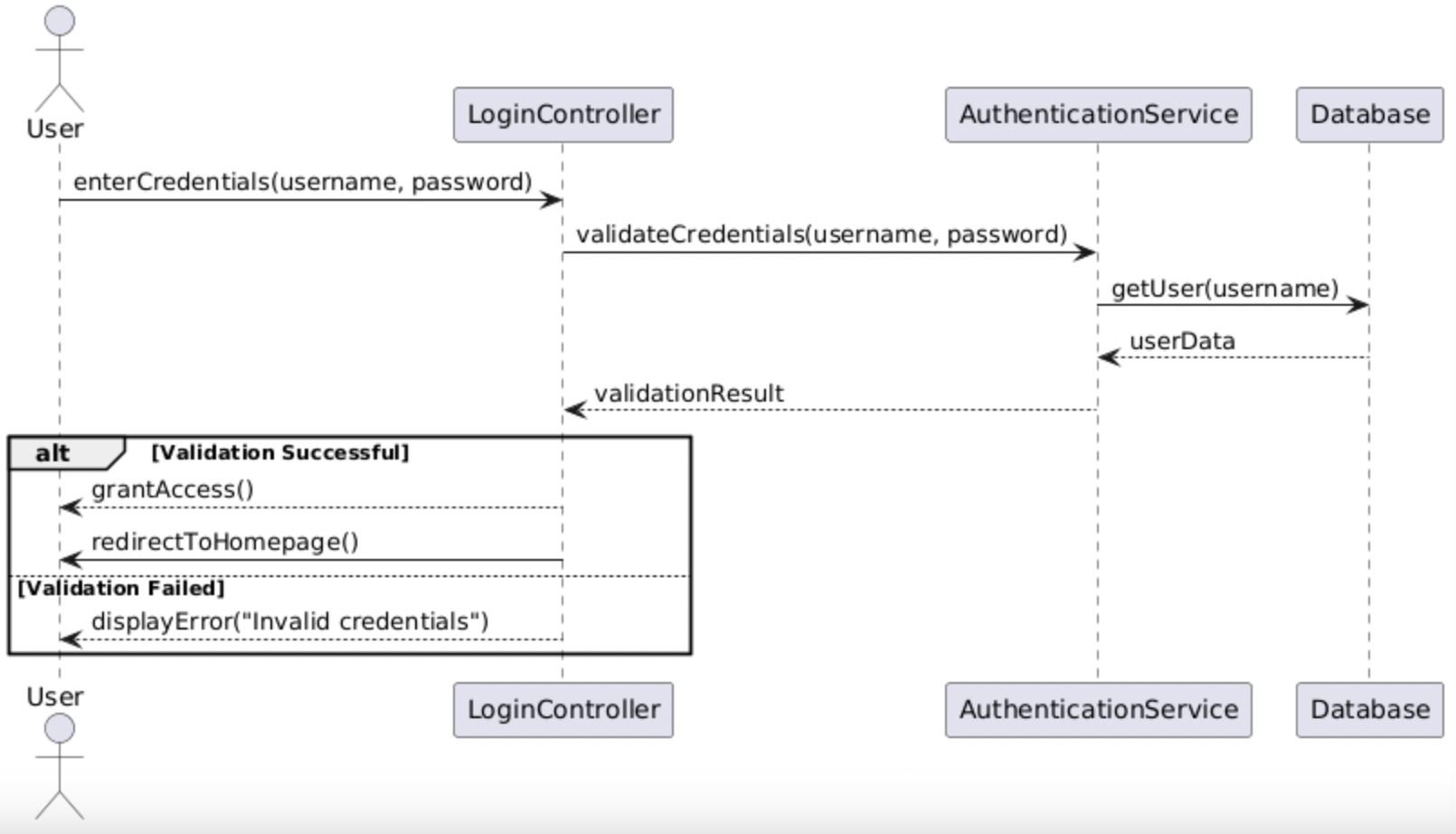
# High Level UML Diagrams (Joyce)

### High-level UML class diagrams:

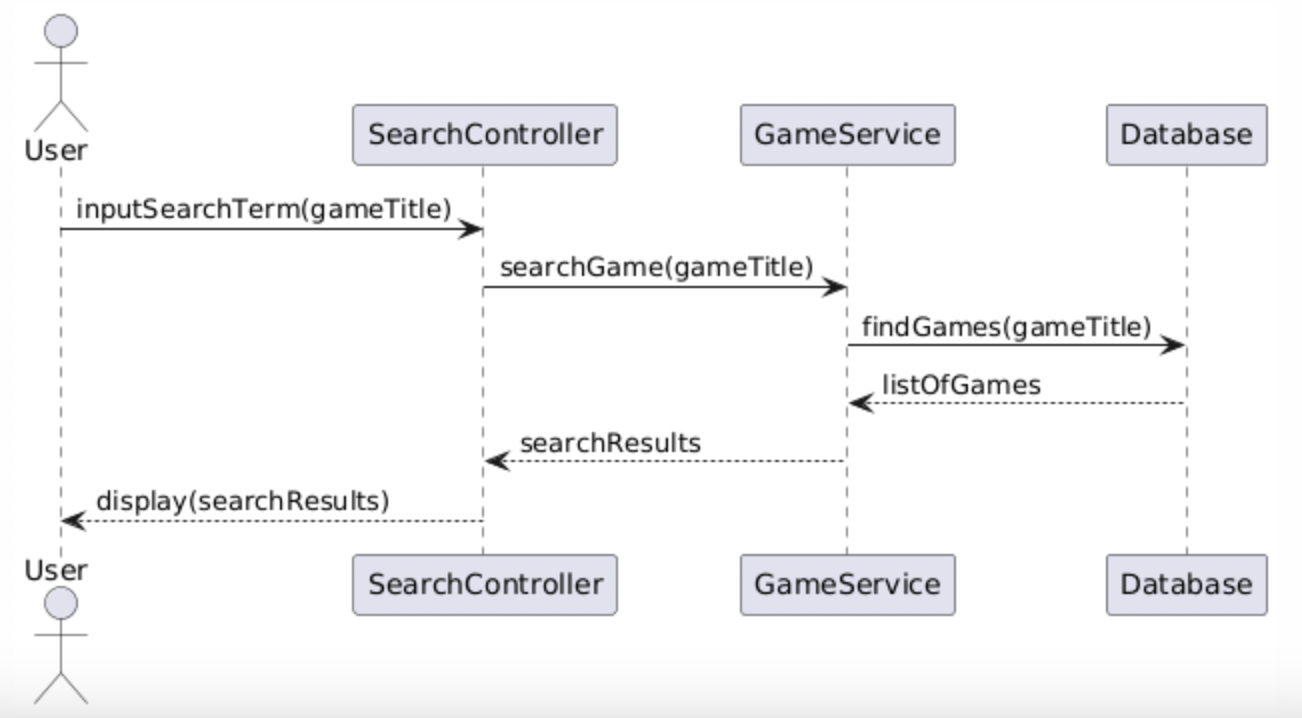


### High-level sequence diagrams:

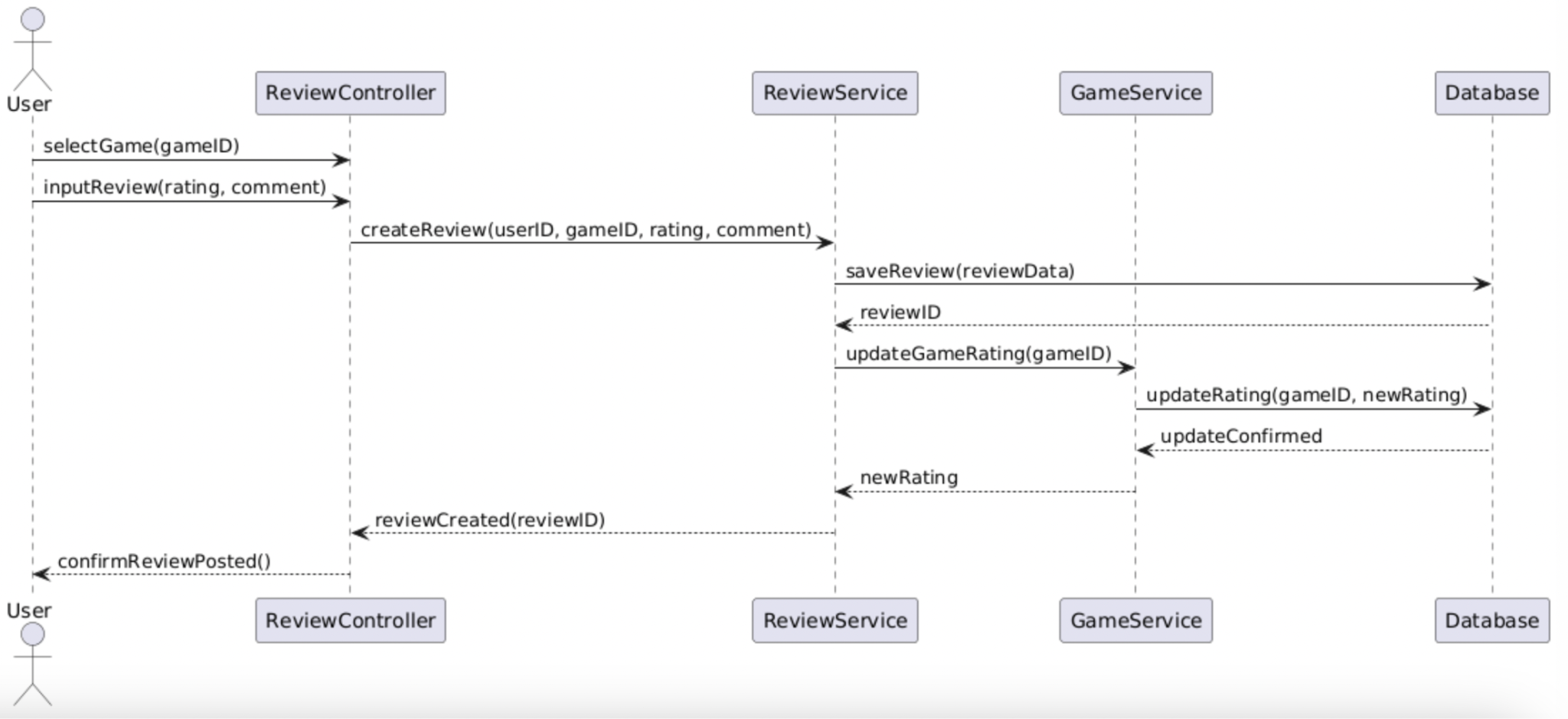
* User Login Sequence



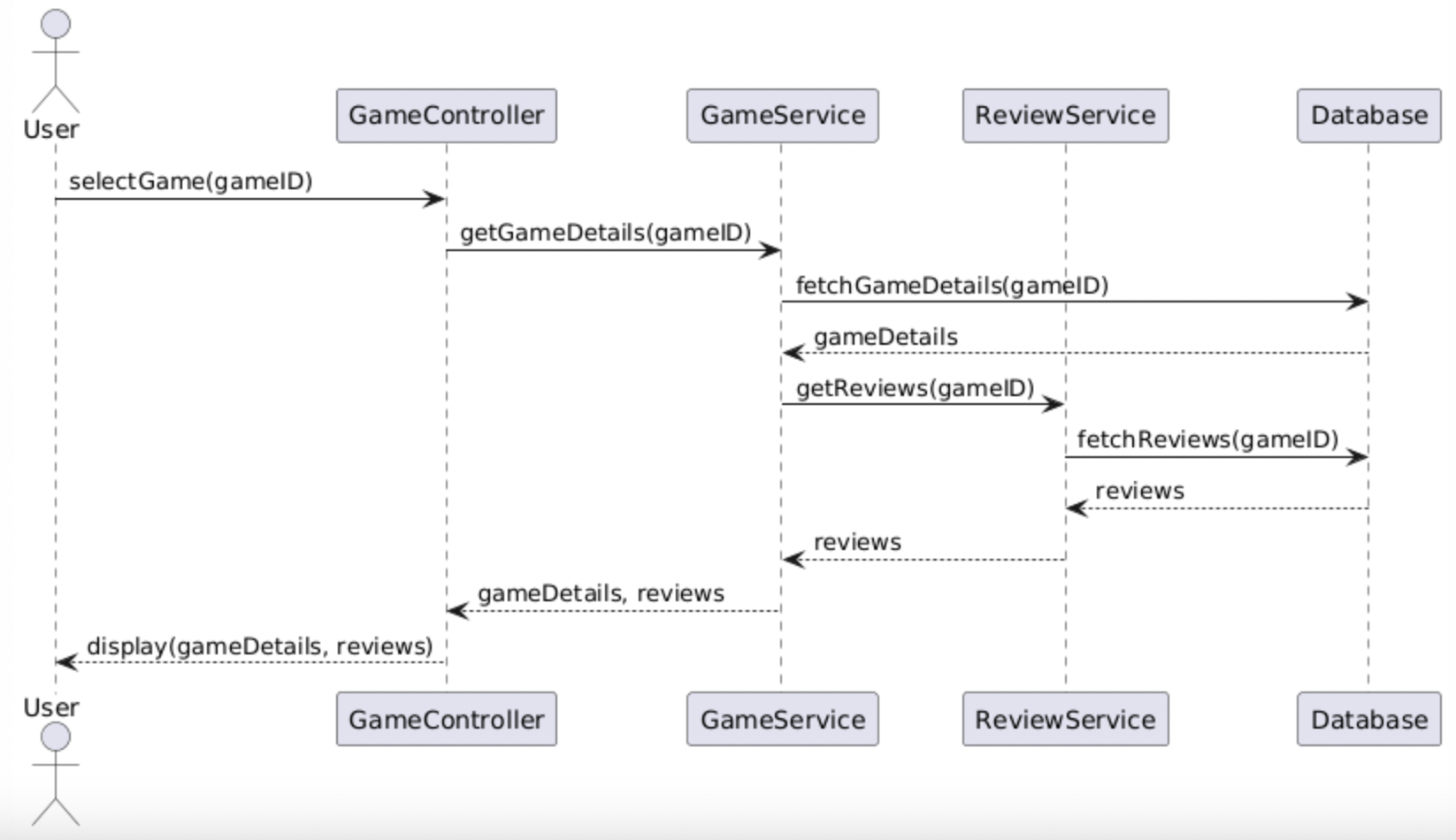
* Search Game Sequence



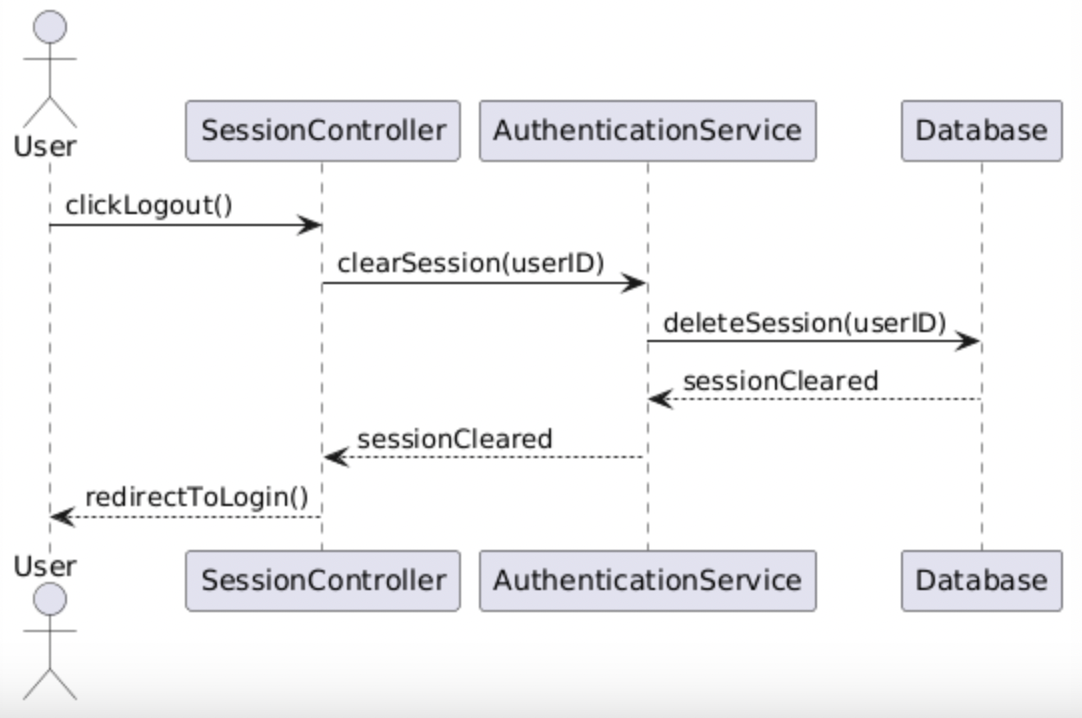
* Leave a Review Sequence



* View Game Details Sequence



* User Logout Sequence



# Key Risks (Nadir)

#### **1. Skills Risks and Mitigation Plan**

* **Risk**: While the team has a good foundation in the required technologies (e.g., React, Node.js, MySQL), there is still a risk that members may struggle with new or unfamiliar technologies, such as the deployment process to AWS EC2 or advanced search functionalities.
* **Mitigation Plan**: To address this, we have created a study plan that includes sharing tutorials, documentation, and online courses. Team members are encouraged to ask questions, explain their logic during meetings, and review each other’s code to ensure that everyone is learning and improving. Additionally, tasks have been assigned to those most familiar with certain technologies to help bridge any gaps.

#### **2. Schedule Risks**

* **Risk**: The primary risk here is if unexpected delays occur (e.g., bugs, feature misalignment) that could hinder progress on key milestones, impacting the final project deadline.
* **Mitigation Plan**: Our team is using Trello for task management, where detailed tasks and deadlines are assigned to each team member. The tool is updated regularly, and if any changes occur, such as task reprioritization or delays, they are communicated transparently through the platform. We conduct regular check-ins and adjust the schedule as needed to ensure transparency and maintain momentum.

#### **3. Teamwork Risks**

* **Risk**: While the team has generally been active, a potential risk lies in differing work paces and participation levels, which could cause bottlenecks in certain areas.
* **Mitigation Plan**: Team members have been consistently attending meetings and adhering to their assigned tasks. In case of any delays or uneven pace, we plan to mitigate this by redistributing tasks when necessary and checking in more frequently with individuals who may need extra support. A clear communication channel (Slack) ensures that if anyone is stuck, they can quickly get help and stay on track.

#### 

#### 

#### **4. Legal/Content Risks**

* **Risk**: The project involves the use of APIs and potential third-party content (e.g., game data, images), which could lead to legal issues such as unauthorized use of copyrighted material.
* **Mitigation Plan**: We are following all API usage guidelines, ensuring that content is legally sourced and licensed appropriately. For media and data used from external sources, proper attribution will be provided when necessary. Since the APIs we use are free and public, and there are no licensing fees involved, the legal risk remains low.

#### **Overall Risk Summary**

The main risks identified (skills, schedule, teamwork, legal/content) are being actively managed through regular communication, proactive learning, task management tools, and adherence to legal guidelines. By addressing each risk early, we aim to maintain smooth project progress and mitigate potential disruptions.

### 

# Project management (Andrew)

We have two alternating Scrum Masters (Jason and Joyce). During our Monday meetings, the scrum masters break down the milestone tasks for the sprint and determine who will work on each task. During our Wednesday scrum meetings, the scrum masters check-in with team members to get an update on the progress of their tasks. As product owner, I use the project management tool, Jira, to track who is assigned to each task and make sure all tasks are assigned. Using Jira, I can keep track of which tasks are in the backlog, in progress or completed to make sure we completed all tasks on schedule.

