Use Cases for Haze

Use Case 1: User Registration

Actors:

- **User**

Main Flow:

1. The user navigates to the registration page.
2. The user enters their username, email, and password.
3. The system validates the input (e.g., checks for a valid email format, password strength).
4. If valid, the system creates a new user account.
5. The system stores the user details in the database.
6. The system sends a confirmation message to the user's email.
7. The user verifies their account and is redirected to the login page.

Alternative Flows:

- **A1**: If the email is already in use, the system displays an error message and prompts the user to choose a different email.
- **A2**: If the password does not meet security requirements, the system displays an error message.

Use Case 2: User Login

Actors:

- **User**

Main Flow:

1. The user enters their email and password on the login page.
2. The system checks the credentials.
3. If the credentials are correct, the system logs the user in and displays their library.
4. If the credentials are incorrect, the system displays an error message.

Alternative Flows:

- **A1**: If the credentials are incorrect, the system displays an error message.
- **A2**: If the user forgets their password, they can initiate a password reset process via email.

Use Case 3: Browse Games

Actors:

- **User**

Main Flow:

1. The user navigates to the "Browse Games" section.
2. The system displays the available games.
3. The user can filter and search for games using various criteria.
4. The user clicks on a game to view more details.

Alternative Flows:

- **A1**: If no games are available, the system displays a message saying "No games available."

Use Case 4: Game Purchase

Actors:

- **User**

Main Flow:

1. The user selects a game they wish to purchase.
2. The system displays the price and purchase options.
3. The user clicks "Buy Now" and selects a payment method.
4. The system processes the payment.
5. The system adds the game to the user's library.
6. The user receives a confirmation email.

Alternative Flows:

- **A1**: If the payment fails, the system displays an error message and prompts the user to re-enter their payment details.

Use Case 5: Manage Friends

Actors:

- **User**

Main Flow:

1. The user navigates to the "Friends" section.
2. The user searches for another user and sends a friend request.
3. The recipient receives the friend request and chooses to accept or reject it.
4. If accepted, both users are added to each other's friends list.

Alternative Flows:

- **A1**: If the user is already friends with the recipient, the system notifies the user.

Use Case 6: Track Achievements

Actors:

- **User**

Main Flow:

1. The user opens their game library.
2. The user selects a game to view its achievements.
3. The system displays the list of achievements for the selected game and tracks the user's progress.
4. The user can view unlocked achievements and their descriptions.

Alternative Flows:

- **A1**: If no achievements are unlocked, the system displays a message saying "No achievements unlocked."

Domain Model

Key Entities and Relationships

1. **User**
    - Attributes: userID, username, email, password, library, friendsList, achievements
    - Relationships:
        - A user has a **Library** (1-to-1 relationship).
        - A user has many **Friends** (1-to-many relationship).
        - A user has many **Achievements** (1-to-many relationship).
2. **Game**
    - Attributes: gameID, name, description, price, genre
    - Relationships:
        - A game can belong to many **Users** (many-to-many relationship) through the Library.
3. **Library**
    - Attributes: userID, games[] (a collection of Game objects)
    - Relationships:
        - A user has a **Library** (1-to-1 relationship).
4. **Achievement**
    - Attributes: achievementID, name, description, gameID
    - Relationships:
        - An achievement is associated with a **Game** (1-to-many relationship).
5. **Friend**
    - Attributes: friendID, friendUsername
    - Relationships:
        - A user has many **Friends** (many-to-many relationship).
6. **Payment**
    - Attributes: paymentID, amount, paymentMethod, status
    - Relationships:
        - A user makes many **Payments** (1-to-many relationship).
7. **Email**

- Attributes: emailID, reci
- pient, subject, body
- Relationships:
    - A **Payment** triggers an **Email** for confirmation (1-to-1 relationship).