VIETNAM NATIONAL UNIVERSITY – HCM

INTERNATIONAL UNIVERSITY

Final Report

Topic: Web-based Rhythm Game

Group Starlight

No.	Name	Student ID	Contribution
1	Tạ Trung Hiếu	ITCSIU21180	28.3%
2	Phạm Tiến Đạt	ITITIU21172	28.3%
3	Nguyễn Thị Thanh Thảo	ITITWE21111	28.3%
4	Nguyễn Nguyên Bảo Phú	ITCSIU21216	15%

Assoc. Prof. Nguyễn Văn Sinh

Department: School of Computer Science and Engineering

Course: Web Application – IT093IU

Ho Chi Minh city, Vietnam, 2024

Table of Contents

List of Figures

Chapter 1: Introduction

The “Star Light” web application is a rhythm-based game designed to engage users in a dynamic and interactive music experience. It allows players to match rhythmic patterns to music tracks while adjusting key bindings, accessing a music library, and viewing beat count accuracy feedback on their performance. The game offers a variety of difficulty levels that cater to both novice and experienced players, each level presenting a unique musical mood and rhythm complexity. This app aims to provide a blend of fun and challenge by simulating the fast-paced excitement of rhythm games, while also allowing for score tracking, peer-to-peer multiplayer modes, and community-created levels in future iterations.

In the development of "Star Light," we have leveraged a range of frontend and backend technologies, a database for user and system data management, and a systematic project management approach. Using the Scrum methodology has allowed the team to incorporate user feedback iteratively, focusing on an agile workflow to refine gameplay mechanics and user interface. By the end of this report, readers will gain an understanding of the project’s objectives, features, technologies, and planned future developments.

Chapter 2: Background

1.1 Project Background

In the development of "Star Light," we have leveraged a range of frontend and backend technologies, a database for user and system data management, and a systematic project management approach. Using the Scrum methodology has allowed the team to incorporate user feedback iteratively, focusing on an agile workflow to refine gameplay mechanics and user interface. By the end of this report, readers will gain an understanding of the project’s objectives, features, technologies, and planned future developments.

- Project Scopes

- **User Account Management:**
  - Secure login and registration system with user credential storage.
  - Basic profile features allowing for personalized user settings.
- **Core Gameplay Mechanics:**
  - Vertically scrolling rhythm gameplay with various difficulty levels (easy to expert).
  - Real-time performance feedback, including beat accuracy judgments (Critical Perfect, Perfect, Good, Bad, Miss).
  - Scoring system based on user input timing, accuracy, and combo bonuses.
- **Configurable Settings:**
  - Customizable key bindings for personal input preferences.
  - Adjustable audio settings to synchronize sound and gameplay.

- **Music Library and Levels:**
  - Access to a music library with selectable songs and associated beat maps (charts).
  - Integration of song attributes such as BPM, song duration, and artist information.
- **User Feedback and Performance Tracking:**
  - Dashboard with heatmap visualization showing timing accuracy across gameplay.
  - Detailed scoring breakdowns and accuracy heatmaps per song.
- **Future Expansion (Planned Features):**
  - Community-created levels to encourage user-generated content and replayability.
  - Peer-to-peer multiplayer mode for competitive gameplay.
  - Offset adjustment settings to allow users to fine-tune latency for their specific devices.

- Project Objectives:

The primary objectives of this project include:

- Creating an engaging rhythm-based gameplay experience that caters to different skill levels.
- Developing a scalable system architecture to support real-time performance feedback, including accuracy and timing metrics.
- Implementing configurable user settings, such as key binding and latency adjustments, to improve accessibility.
- Establishing a robust, future-ready backend and database for user data management, score tracking, and peer-to-peer interaction.
- Building a modular framework to enable future additions, such as multiplayer functionality, enhanced visual effects, and community level sharing (Future Work).

1.2 User Roles & Permissions

"Star Light" incorporates several user roles, each with specific permissions to enhance gameplay security and personalization. At present, the roles are streamlined to include basic user roles, with potential for expanded administrative roles in future updates.

- **Players:** Registered players can access all core functionalities, including music library access, beat count accuracy, gameplay feedback, personalized settings, and score tracking on the dashboard. They also have access to their

profile page, where they can view statistics, adjust settings, and modify their profile picture.

- **Administrator:** Administrators hold the highest level of access within "Star Light," enabling them to manage, maintain, and monitor the entire system. Their responsibilities extend beyond gameplay to include overseeing player accounts, song management, system optimization, and database access. The admin role includes tools for:
  - ***Account Management:*** Admins can create, modify, and delete player accounts, addressing any issues users may face and maintaining community standards.
  - ***Content Management:*** Admins can add or edit songs, maintain the music library, and manage associated metadata to ensure a seamless user experience.
  - ***Database Access and Management:*** Admins have direct access to the system database to ensure data integrity, update records, and troubleshoot any data-related issues.
  - ***System Monitoring and Performance Analytics:*** Admins are equipped with tools to monitor system performance, track usage statistics, and analyze data trends. This monitoring ensures smooth operation and helps to identify areas for system optimization.
  - ***Maintenance, Backup, and Recovery:*** Admins handle the critical tasks of regular updates, system maintenance, and backups. In case of data loss, they are responsible for executing recovery protocols to restore user data and system functionality.

Chapter 3: Project approach

1.3 Work Breakdown Structure

The “Star Light” web application project is structured into five essential phases: **Planning & Analysis, Design, Development & Integration, Testing,** and **Deployment & Maintenance**. Each phase builds on the previous one, ensuring a smooth and efficient workflow toward project completion.

a. Planning & Analysis

The Planning & Analysis phase establishes the project’s foundation by identifying objectives, scope, and user needs. This phase ensures all stakeholders have a shared vision for “Star Light” and defines essential features for initial development.

- **Define Project Scope & Objectives**: Clarify goals, primary features, and the project’s vision for a rhythm game with an intuitive interface and immersive gameplay.

- **Define User Roles & Permissions**: Identify expectations of key user roles and permission, including admins and players, ensuring all core features (e.g., music selection, scoring, and real-time feedback) align with user needs.
- **Feature Prioritization**: List and prioritize features for the MVP (minimum viable product) and future updates, focusing initially on gameplay, scoring, and essential feedback mechanisms.

b. Design

In the Design phase, the system architecture and user experience components are created. This phase provides a visual and functional blueprint for development, aligning design elements with project requirements and user feedback.

- **System Architecture**: Develop use case, class, and sequence diagrams to visualize interactions and relationships among components, ensuring a scalable structure.
- **UI/UX Mockups**: Create wireframes and prototypes for key screens (e.g., landing page, gameplay interface, and profile settings), refining layout and user flow.
- **Database Schema Design**: Define and structure data tables and relationships to efficiently store user profiles, scores, and gameplay data.
- **Gameplay Mechanics & Dashboard Layout**: Outline scoring logic, accuracy tracking, and dashboard elements to ensure a smooth and responsive user experience.

c. Development & Integration

This phase involves building and integrating the application's frontend, backend, and database. Each component is developed, tested for interoperability, and optimized for performance.

- **Frontend Development**: Develop the user interface with React, HTML, CSS, and JavaScript to deliver an interactive gameplay.
- **Backend & API Development**: Create server-side logic and RESTful APIs for user authentication, gameplay data handling, and leaderboard tracking.
- **Database Integration**: Implement and connect the database for user data, scores, and real-time performance tracking, ensuring secure and efficient data handling.
- **Game Engine Integration**: Implement core gameplay mechanics (timing, scoring, feedback) using Unity, and synchronize with backend systems for seamless functionality.

d. Testing

The Testing phase focuses on verifying functionality, stability, and usability across all components. Rigorous testing minimizes potential issues and improves the user experience.

- **Functionality & Integration Testing**: Test all core features and interactions (e.g., login, gameplay mechanics, feedback dashboard) to ensure components work as a cohesive system.
- **User Acceptance Testing (UAT)**: Conduct usability tests with users to validate gameplay experience, identifying and addressing any issues with UI/UX or feature functionality.
- **Performance & Load Testing**: Assess system response and stability under varying loads, ensuring it can support high concurrent user counts.
- **Bug Fixing & Optimization**: Document, prioritize, and address issues found during testing, ensuring a polished and reliable product.

e. Deployment & Maintenance

The final phase involves deploying "Star Light" for public use and ensuring ongoing maintenance, updates, and support.

- **Deployment**: Configure and launch the application on a production server, optimizing for security and performance, and verifying all core functionalities work in the live environment.
- **Monitoring & Analytics**: Implement monitoring tools to track system performance, user engagement, and game metrics, allowing data-driven improvements.
- **Regular Updates & Feature Enhancements**: Maintain system updates and add features (e.g., multiplayer, community levels) based on user feedback, continually enhancing gameplay.
- **Backup & Recovery Protocols**: Establish regular data backups and implement recovery procedures to prevent data loss, ensuring system integrity and continuity.

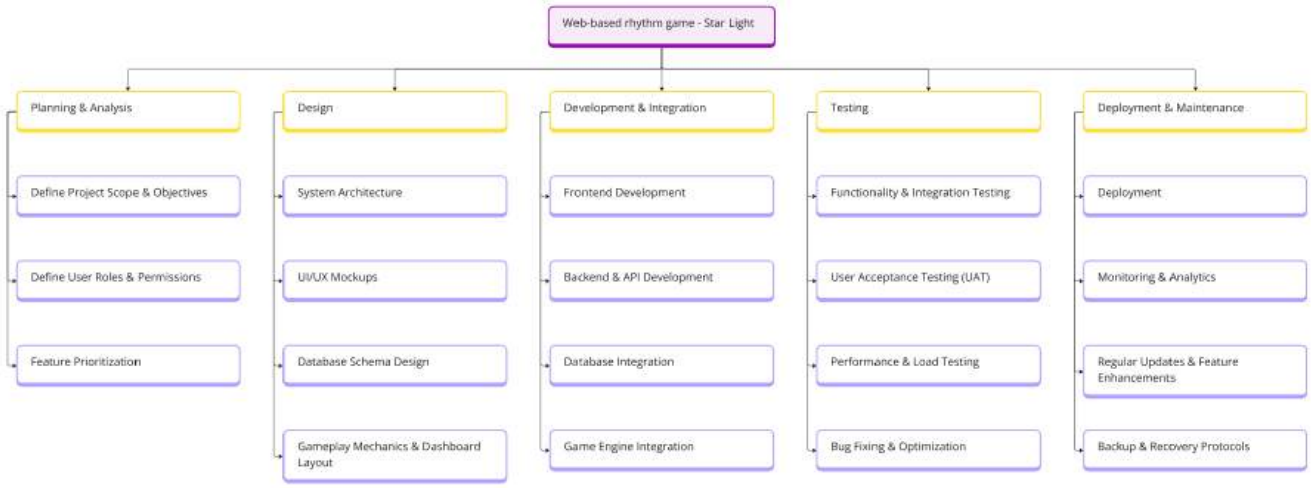

Figure 1: Work Breakdown Structure

1.4 Applying Methodology

- ✓ The project employs an Agile methodology approach and manages sprints and tasks through the Jira software website.
- ✓ The Project was divided into 6 sprints, each sprint lasts around 12 to 24 days. The lists of sprints content respectively are Project Setup and Requirements Gathering, Design, Development- part 1, Development- part 2, Deployment, and Buffer Time.
- ✓ This methodology allows us to freely get feedback and immediately modify our system each sprint until it is complete.

1.5 Software Features

The "Star Light" application is designed with a range of system features to provide a dynamic, engaging, and customizable experience for players. These features have been developed to ensure high usability, immersive gameplay, and an intuitive interface that appeals to a wide variety of users. Below are the primary features of the system:

1. User Authentication

- **Login and Registration**: Users can register for an account and log in securely using their credentials. This feature is essential for tracking player progress, scores, and personalization options.
- **Session Management**: Once logged in, users maintain a session, allowing them to play multiple games or return to their dashboard without re-authenticating.

2. Configurable Gameplay Settings

- **Key Binding Options**: Players can customize key bindings to suit their preferred controls, making the gameplay accessible and comfortable for different play styles.
- **Audio Latency Adjustment**: An offset adjustment feature allows players to fine-tune audio timing to synchronize better with their input, minimizing latency discrepancies for a smoother experience.

3. Music Library

- **Interactive Song Library**: Players can browse a curated selection of songs organized by difficulty and genre, helping users select tracks that align with their preferred music style.
- **Beat Maps/Charts (Levels)**: Each song includes a beat map with specific timing challenges
- **Preview and Selection**: Users can preview song details such as BPM, artist, and Best score of the song before starting the game. The intuitive layout enables players to quickly navigate between tracks

4. Real-Time Gameplay and Scoring System

- **Accuracy-Based Scoring**: The game employs a scoring system based on user input accuracy, judging timing precision to award different scores: Critical Perfect, Perfect, Good, Bad, or Miss.
- **Feedback Heatmap**: A real-time feedback heatmap displays timing accuracy over the duration of the gameplay, showing players their performance at a glance and helping them identify areas for improvement.
- **Combo Scoring and Bonuses**: Players earn combo bonuses for consecutive accurate notes, and advanced scoring options are available for more challenging gameplay modes (e.g., tournament mode).

5. Performance Dashboard

- **Score Tracking and History**: Users can view their best scores and play history for each song, helping them track their progress and compare performances over time.
- **Heatmap Analysis**: The dashboard displays a heatmap of performance over time for each song, allowing users to review their timing accuracy in different intervals.
- **Fast/Late Timing Bar Charts**: Additional visuals provide insights into timing patterns, indicating whether a player was consistently early or late, which helps improve timing accuracy.

6. User Profile Management

- **Profile Customization**: Players can update their profile picture, username, and display preferences, making the gaming experience more personal.
- **Experience and Song Record Tracking**: The system tracks total game time, experience points, and levels achieved, allowing players to see their cumulative progress.
- **Achievements and Badges**: Unlockable achievements and badges are awarded based on milestones or high scores, encouraging continued play and skill development.

1.6 System's Workflow

The system workflow follows these steps:

Landing Page

The Landing Page is the starting point of the rhythm game. It handles user authentication (register and login) and ensures that only logged-in users can access the main features of the game. Here's how it works:

1. **Starting the Game Without Logging In**:
    - When a user clicks the "Start Game" button without logging in, the system immediately checks whether the user is authenticated.
    - If the user is not logged in, an **error popup** appears, asking the user to either log in or register. This ensures that all users have valid accounts before accessing the game.
2. **Registering a New Account**:
    - If the user chooses to register, they must input their email and password.
    - The system validates these inputs:
        - If the inputs are invalid (e.g., missing characters or incorrect format), the system displays an **error message** with suggestions to fix the issue.
        - If the inputs are valid, the system stores the user's account information in the database, and a **success popup** appears to confirm the registration.
    - The user can now log in with their new credentials.
3. **Logging In**:
    - When a user attempts to log in, the system checks their input credentials against the database.
    - If the credentials are incorrect, an **error popup** is displayed, asking the user to try again.
    - If the credentials are correct, the system grants access, and the user is redirected to the Main Menu Pages.

Main Menu Pages

Once logged in, the user enters the Main Menu Pages, where they can navigate through different features of the game. Here's how this section works:

1. **Fetching Data**:
    - As soon as the user logs in, the system fetches essential data using APIs:
        - **Song Data**: All available songs are retrieved from the server for display on the Song Page.
        - **User Information**: The system also retrieves user-specific details like profile information and history.
2. **Navigating Between Pages**:
    - The user can move between the following pages using a **header navigation bar**:
        - **Song Page**: A list of all available songs, allowing users to pick and play a track.
        - **History Page**: Displays the user's past gameplay scores and achievements.
        - **Event Page**: Shows any special events or in-game challenges.
        - **Store Page**: Allows the user to browse and purchase in-game items or content.
3. **Interacting with the Burger Menu**:
    - A **burger menu** on the header displays all the available songs. Users can:
        - **Search for Songs**: Use a search bar to quickly find specific tracks.
        - **Filter Songs**: Apply filters to narrow down song options based on certain criteria.
        - **Select Songs**: Clicking on a song in the menu redirects the user to the Song Page with details for that specific track.
4. **Playing a Song**:
    - On the Song Page, users can click the **"Play" button** next to a track.
    - This action sends the song's ID to the Gameplay Interface, where the actual rhythm game begins.

Gameplay Interface

The Gameplay Interface is where the core rhythm game mechanics take place. This section focuses on the player's interaction with the game during a song playthrough.

1. **Preparing the Song**:
    - Before the gameplay starts, the system retrieves and processes the song data (such as the beatmap, notes, and timing information).

    - This data is essential for generating notes and ensuring the gameplay matches the rhythm of the selected track.
2. **Game Loop**:
   The gameplay runs in a continuous loop until the song ends. During this loop, the following mechanics are active:
    - **Note Tracking and Generation**: The system generates visual notes on the screen in sync with the song's rhythm. These notes fall toward specific target areas, guiding the player on when to press corresponding keys.
    - **Player Input**: The player interacts with the game using their keyboard, pressing keys in time with the notes. They can also pause the game if needed.
    - **Time Evaluation**: The system evaluates the player's inputs to determine their timing accuracy.
    - **Real-Time Updates**: The game updates the player's score, combo count, and accuracy percentage in real-time based on their performance.
3. **Ending the Gameplay**:
    - Once the song ends, or if the player chooses to exit, the system performs the following actions:
        - **Record Score**: The player's performance (score, accuracy, combos, etc.) is saved to the database.
        - **Redirect to History Page**: The user is taken to the History Page, where their most recent score is displayed.

Profile Page

The Profile Page allows users to view and manage their personal information, gameplay records, and achievements.

1. **Viewing Profile Data**:
    - The system retrieves and displays the user's profile information, including:
        - **Song Record**: A detailed list of all songs the user has played, along with their scores and performance data.
        - **Achievements**: A summary of the milestones and accomplishments the user has unlocked.
2. **Updating Account Settings**:
    - Users can update their account details (username, email, or password).
    - The system validates the inputs during this process:
        - If the inputs are invalid, an **error message** is displayed.
        - If the inputs are valid, the changes are saved to the database, and a **confirmation popup** appears.
3. **Profile Tabs**:

- The Profile Page has tabs to switch between:
  - **Song Records**: Displays all the tracks the user has played.
  - **Achievements**: Lists all the unlocked in-game achievements.
  - **Account Settings**: Provides options to update user details.

4. **Exiting the Profile Page**:
   - When the user finishes viewing or updating their profile, they can exit the Profile Page.
   - The system displays a confirmation popup before redirecting them back to the Main Menu.

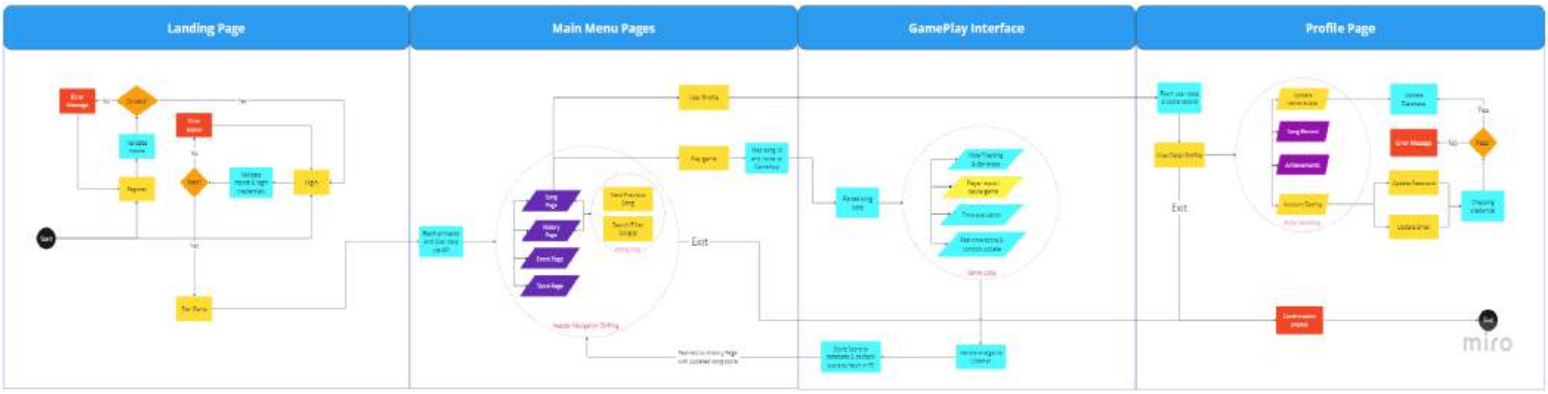

Figure 2: System Workflow

1.7 Project Timeline

ID	Task Name	Start Date	Finish Date	Duration (days)	% Complete
1	**Project Setup and Requirements Gathering**	**9-Sep**	**23-Sep**	**14**	**100%**
2	Identify scopes, objectives, User Roles & Permissions	9-Sep	16-Sep	7	100%
3	Draft Design Workflow	17-Sep	23-Sep	7	100%
4	List and prioritize features for the MVP	23-Sep	23-Sep	1	100%
5	**Design**	**24-Sep**	**15-Oct**	**21**	**100%**
6	Create wireframes and prototypes for key screens, refining layout and user flow.	24-Sep	1-Oct	7	100%
7	Develop user stories, use case, class, and sequence diagrams	1-Oct	15-Oct	15	100%
8	Define and structure data tables and relationships	10-Oct	15-Oct	5	100%
9	Outline scoring algorithm logic, accuracy tracking, and dashboard elements	13-Oct	15-Oct	2	100%
10	**Development - Part 1**	**1-Oct**	**28-Oct**	**28**	**100%**
11	Starts building the FE setup, user interface based on the mockups (Landing Page, Main Menu Page, Profile	1-Oct	28-Oct	28	100%
12	Unity Gameplay mechanism discovery, develop a script to convert song rhythm to beat	1-Oct	7-Oct	7	100%
	Gameplay Interface FE Structure Setup	8-Oct	21-Oct	14	100%
13	Starts setting up the database	1-Oct	5-Oct	5	100%
14	Create server-side logic and RESTful APIs swagger for user authentication, gameplay data handling.	6-Oct	28-Oct	23	100%
15	**Development - Part 2**	**29-Oct**	**26-Nov**	**28**	**100%**
17	FE test API connection	29-Oct	15-Nov	17	100%
18	Implement and connect the database to FE	16-Nov	26-Nov	11	100%
19	Implement core gameplay mechanics, and synchronize with backend systems for functionality.	29-Oct	26-Nov	28	100%
20	**Deployment System**	**27-Nov**	**8-Dec**	**12**	**100%**
21	Functionality & Integration Testing	27-Nov	3-Dec	7	100%
22	User Acceptance Testing (UAT)	27-Nov	3-Dec	7	100%
23	Bug Fixing	27-Nov	8-Dec	12	100%
24	**Make Final Report + Presentation**	**27-Nov**	**8-Dec**	**12**	**100%**

Figure 3: Project timeline

Figure 4: Timeline gantt chart

Chapter 4: System Analysis and Design

1.8 Requirements Analysis

Functional Requirements:

a. **Authentication and Authorization**:
   - The system must validate user registration inputs (email, password) and check if they meet required constraints .
   - The system must verify login credentials by checking input against stored database records.
   - The system must reject invalid credentials and return error messages for failed logins or registrations.

b. **Data Storage**:
   - The system must store newly registered user data in the database.
   - Gameplay scores must be stored in the database immediately after a song is completed, tagged with user ID and song ID.

c. **API Management**:
   - The system must expose an API to fetch all available song tracks.
   - The system must expose an API to fetch user data after successful login.
   - The system must expose an API to fetch user scores for a specific song.

- After gameplay, the system must expose an API to fetch the user's most recent scores.

d. **Data Validation**:
   - Validate song, user, and gameplay data before storing in or retrieving from the database to prevent corrupted or invalid data entries.

e. **Navigation Bar**:
   - The system must route requests between different pages (e.g., Song Page, History Page, Event Page, Store Page, Gameplay Interface, Profile Page) using React routing.
   - It must handle and validate routing parameters (e.g., songId passed to the Gameplay Interface).

f. **Song Search Functionality**:
   - The system must provide a fuzzy search mechanism for songs, allowing query-based filtering by song title.
   - Search results must be returned in a song title matching track name in the frontend sidebar.

g. **Gameplay Data Processing**:
   - The system must parse song data using songId to generate gameplay sequences (e.g., notes, beats) dynamically.
   - It must track and evaluate gameplay performance (e.g., scores, combos, beat judgements, accuracy percentage) in real-time.

h. **Score and Achievement Updates**:
   - When gameplay is completed, the system must calculate and store the user's total score and update achievements in the database (if any milestones are met).
   - It must provide APIs for querying all scores and achievements for a specific user.

i. **Profile Management**:
   - The system must update user data (e.g., username, email, password) when requested and validate changes before committing them to the database.
   - Profile stats such as total playing time and experience points must be recalculated and fetched from stored gameplay data.

j. **Error Handling**:
   - The system must return appropriate error codes and messages for all failed operations, including invalid inputs, missing data, or unauthorized access.
   - It should log errors and exceptions for debugging and monitoring purposes.

k. **Environment-Specific Configuration**:
   - The system must distinguish between development and production environments, using separate databases and API endpoints.
   - The production environment must use the domain **starlight.swyrin.id.vn.**

l. **Session and State Management**:
   - The system must maintain user session states to track logged-in users and allow seamless transitions between pages.
   - Session expiration or invalidation must be handled securely to prevent unauthorized access.

Non-Functional Requirements:

a. **Usability Requirements**:
   - The system must implement clear and consistent APIs for fetching and updating data to ensure smooth integration with the frontend.
   - All API endpoints should adhere to RESTful design principles and return JSON responses.

b. **Performance Requirements**:
   - API endpoints must process requests and respond within **200ms** under normal load conditions.
   - The gameplay engine must process inputs (e.g., keyboard interactions) and update game states in **real-time**, ensuring a maximum delay of **50ms** for smooth rhythm gameplay.
   - The system must support at least **10 concurrent active users** without significant performance degradation.

c. **Space Requirements**:
   - The database must be optimized to store:
     - User accounts, song data, gameplay scores, and achievement records.
     - Space-efficient indexing must be implemented for fast query responses.
   - Media assets (e.g., song files, profile pictures) must be stored on a scalable and reliable cloud storage platform.

d. **Dependability Requirements**:
   - The system must provide **99.9% uptime** for production environments.
   - Data backup mechanisms must be in place to recover user accounts, scores, and achievements in case of system failure.

e. **Security Requirements**:
   - All sensitive data (e.g., passwords) must be securely hashed and stored using encryption standards.
   - Communication between the frontend and backend must be encrypted using **HTTPS**.
   - API endpoints must enforce authentication and role-based authorization for sensitive operations.

f. **Environmental Requirements**:
   - The system must support two distinct environments:

- CI/CD pipelines must automate deployments to production.

g. **Operational Requirements**:
   - The system must log key operational events, such as user logins, data updates, gameplay completions, and API errors.
   - Monitoring tools must track system performance, API health, and resource usage.

h. **Development Requirements**:
   - The system must use scalable and modular architecture to allow future feature extensions (e.g., adding new game modes or events).
   - The backend can be built with a modern, reliable framework .
   - All components must be thoroughly tested (unit tests, integration tests, and system tests) before deployment.
   - Code should follow best practices for maintainability, including clear documentation and version control.

1.9 Use Case Diagram

Use case Diagram for the whole software:

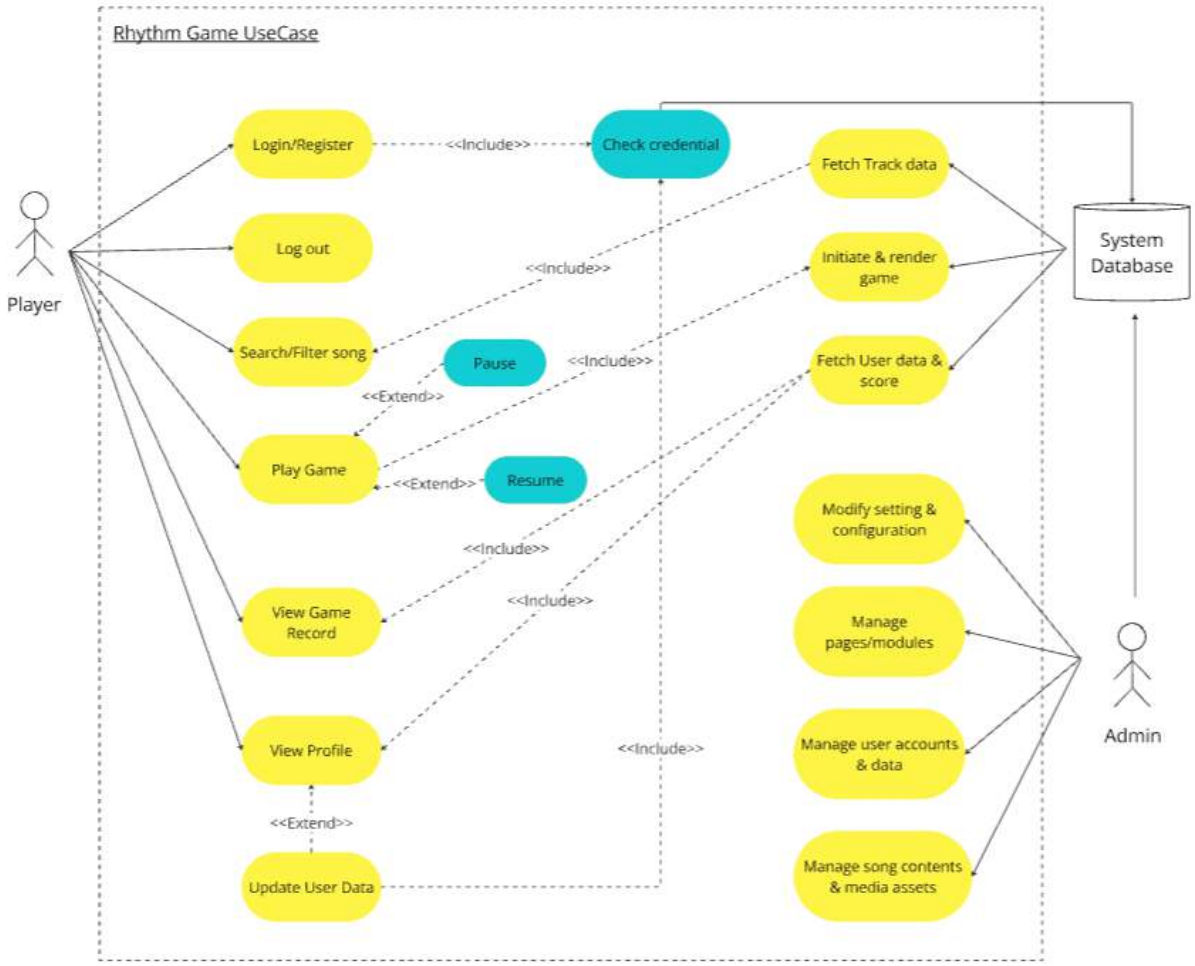

Figure 5: Use Case Overall System

1.9.1. Use Case Login, Register, Forgot password

- **User Story:**

a. Scenario 1: User Registration

- As a new user, I want to register an account so that I can access the game and save my progress
- Scenario: Successful Registration
    - Given I am on the registration popup
    - When I enter a valid email, password, and player name
    - And I click the "Sign up" button
    - Then the system should verify the email format and password strength

- And it should check if the email already exists in the database
- And it should create a new account in the database
- And display a popup message "Registration Success"

b. Scenario 2: Registration with Invalid Email

- Scenario: Invalid Email Format
  - Given I am on the registration popup
  - When I enter an invalid email format
  - And I click the "Sign up" button
  - Then the system should display an error popup "Invalid Email Format"

c. Scenario 3: Duplicate Account Registration

- Scenario: Duplicate Account Registration
  - Given I am on the registration popup
  - When I enter an email that already exists in the database
  - And I click the "Sign up" button
  - Then the system should display an error popup "Email already exists"

d. Scenario 4: Successful Login

- As a registered user, I want to log in so that I can access my account and game progress
- Scenario: Successful Login
  - Given I am on the login popup
  - When I enter valid email and password credentials
  - And I click the "Login" button
  - Then the system should verify the credentials with the database
  - And it should log me into my account
  - And display the "Song Page"

e. Scenario 5: Login with Invalid Credentials

- Scenario: Invalid Login Credentials
  - Given I am on the login popup
  - When I enter an invalid email or password
  - And I click the "Login" button
  - Then the system should display an error popup "Invalid email or password"

f. Scenario 6: Forgot Password

- As a user, I want to recover my forgotten password so that I can regain access to my account
- Scenario: Successful Password Recovery
  - Given I am on the "Forgot Password" popup
  - When I enter my registered email address
  - And I click "Submit"
  - Then the system should verify the email exists in the database
  - And send a password recovery link to my email

- And display a popup "Password recovery email sent"

- **Use case diagram:**

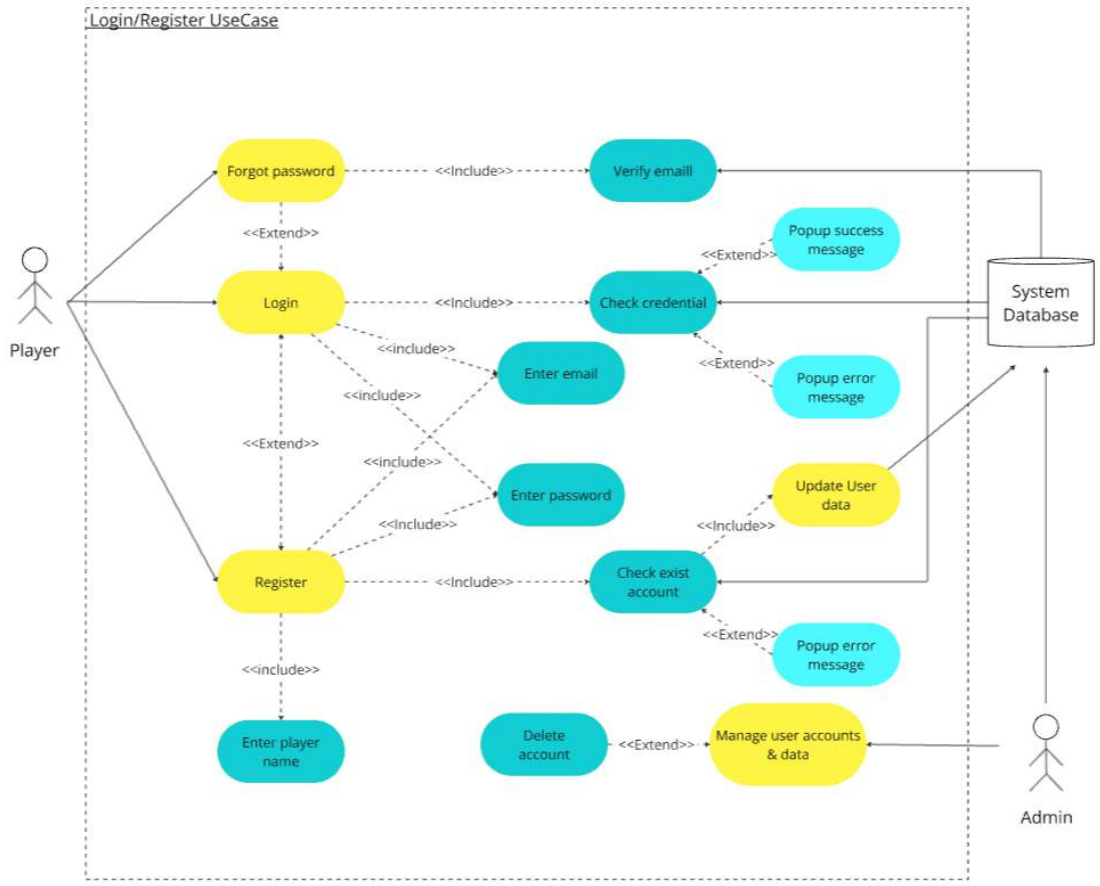

Figure 6: Use Case Login/Register

- **Use case using a table format:**

Table 1: Use Case table for Login/Registration

Use case name:	Use Case Login/Register
Actors:	Player, Admin, System Database
Description:	Allows the player to register for a new account, log in with existing credentials, and reset their password.
Preconditions:	● The player has the web open. ● The system database is accessible. ● Predefine the valid email and password formats..

Postconditions:	● New user account is created for registration. ● Player is successfully logged in after validation. ● Invalid credentials trigger errors.
Normal Flows:	1. Player enters email and password. 2. System validates the input. 3. For registration, the system checks email uniqueness and creates an account. 4. For login, the system checks credentials and grants access to the main menu.
Alternative Flows:	1. If email already exists during registration, the system prompts the user. 2. For login, system allows password reset if credentials fail.
Exceptions:	• System database failure during input validation. • Network issue prevents API call to verify credentials.
Note and issues:	

1.9.2. Use Case Main Menus

- **User Story**:
  - ***a. Scenario 1: Fetching Song Data***
  - As a logged-in user, I want to see a list of available songs in the sidebar so that I can choose a song to play
  - Scenario: Fetching Song Data
    - Given I am on the “Song page”
    - When the page loads
    - Then the system should fetch all song tracks using the API "/api/track/all"
    - And display the songs in the list
  - ***b. Scenario 2: Searching for a Song***
  - As a logged-in user, I want to search for a specific song so that I can find it quickly
  - Scenario: Search for a Song
    - Given I am on the Song page
    - When I enter a song title in the search bar
    - And I click the "Search" button
    - Then the system should filter the song list based on my input
    - And display the matching songs
  - ***c. Scenario 3: Starting a Game***

- As a logged-in user, I want to start a game so that I can play the selected song
- Scenario: Successfully Starting a Game
    - Given I am on the "Main Menu" page
    - When I click the "Play" button on a song
    - Then the system should route me to the gameplay interface
    - And pass the correct "songId" to the game engine
    - And the game engine should load the selected song

d. Scenario 4: Viewing Game Record History Page

- As a logged-in user, I want to view my gameplay history so that I can track my performance and progress
- Scenario: Viewing Gameplay History
    - Given I am on the "History Page"
    - When the page loads
    - Then the system should fetch my song scores and history using the API "/api/score/{songId}" and "/api/score/recent"
    - And display my last-played scores and best scores for each song

e. Scenario 5: Logging Out

- As a logged-in user, I want to log out of my account so that I can exit the system securely
- Scenario: Logging Out
    - Given I am on the Song page, History Page, Event Page, or Store Page
    - When I click the "Leave" button
    - Then the system should clear my session data
    - And redirect me to the "Login Page"
    - And display a confirmation popup "You have successfully logged out"

- **Use case diagram:**

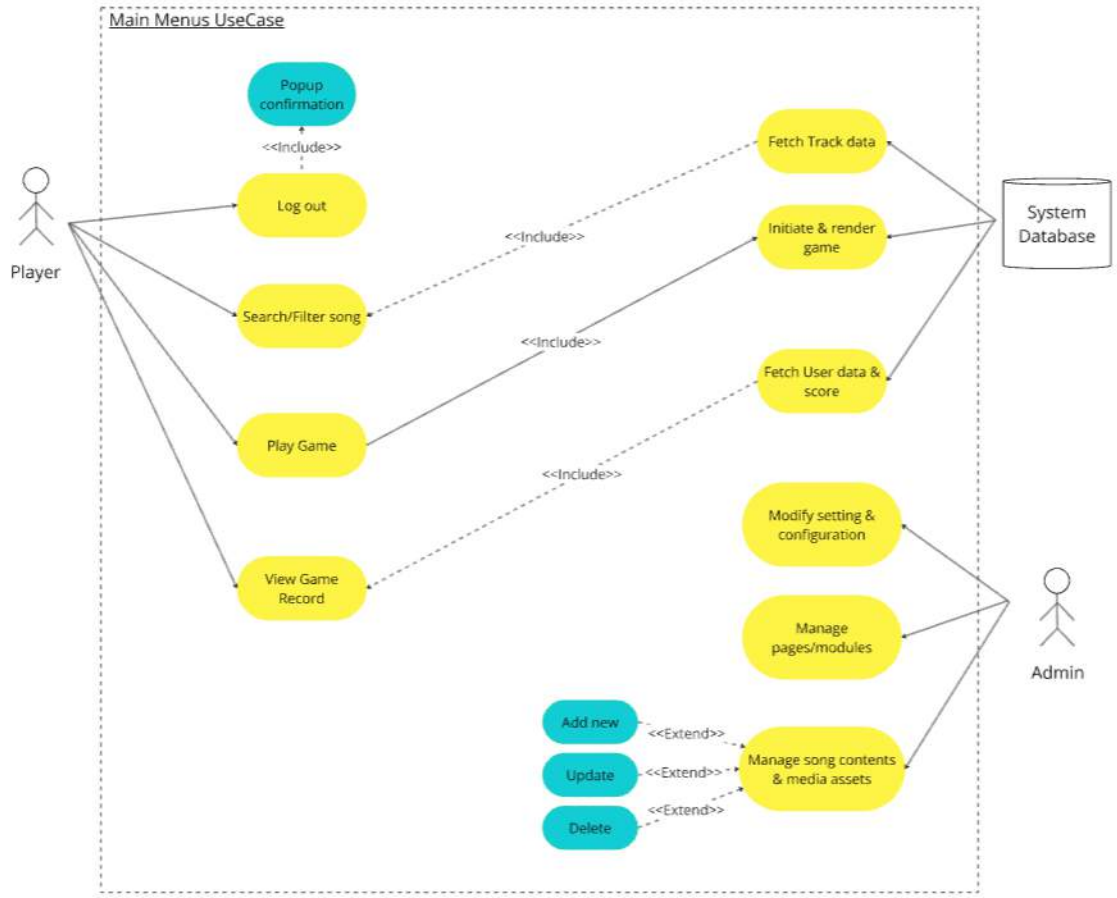

Figure 7: Use Case Main Menus

- **Use case using a table format:**

Table 2: Use Case Table for Search CV format

Use Case name	Main Menus
Actors:	Player, Admin, System Database
Description:	Provides navigation options for players, allowing access to songs, history, events, and store pages.
Preconditions:	• Players must be logged in. • Navigation headers and buttons must be functional. • Backend APIs must be available for song data.
Postconditions:	• Selected pages (e.g., Song Page, History Page) are loaded. • Relevant data is fetched from the database. • Game is initiated if selected.
Normal Flows:	• - Player selects a page via navigation. • - System fetches data (e.g., tracks, scores, events). • - Selected page content is rendered.
Alternative Flows:	• If no tracks are available, the system displays a message. • If network issues occur, the system retries data fetching.
Exceptions:	• API endpoint failure prevents song data fetching. • System navigation fails due to invalid routing.
Note and issues:	

1.9.3. Use Case GamePlay

- **User Story:**
  - ***a. Scenario 1: Starting the Gameplay***
  - As a logged-in user, I want to play a selected song so that I can enjoy the rhythm game experience
  - Scenario: Successfully Starting the Gameplay
    - Given I am on the "Song Page"
    - When I click the "Play" button for a selected song
    - Then the system should route me to the "Gameplay Interface"
    - And it should pass the correct "songId" to the game engine

- And the game engine should map the songId to the corresponding track data
- And render the song with the gameplay interface

b. Scenario 2: Playing the Game

- As a player, I want to interact with the gameplay interface using my keyboard so that I can play the rhythm game
- Scenario: Interacting with Gameplay
  - Given I am on the "Gameplay Interface"
  - When the song starts playing
  - Then the system should display falling notes on the screen
  - And track user keyboard inputs to match the song beat
  - And calculate beat judgments (e.g., Perfect, Good, Miss,..)
  - And display real-time score, combos, and accuracy percentage

c. Scenario 3: Completing the Game

- As a player, I want my gameplay performance to be recorded after completing a song so that I can track my progress and scores
- Scenario: Successfully Completing a Song
  - Given I am playing a song in the "Gameplay Interface"
  - When the song ends
  - Then the system should calculate my total score and final accuracy
  - And store the score in the database using the API
  - And automatically redirect me to the "History Page"
  - And the system should fetch my recent score from the database using the API
  - And display my score on the "History Page"

d. Scenario 4: Pausing the Game

- As a player, I want to pause the gameplay so that I can temporarily stop the game if needed
- Scenario: Pausing a Song
  - Given I am playing a song in the "Gameplay Interface"
  - When I press the "Pause" button
  - Then the system should pause the falling notes
  - And stop the song playback
  - And display the pause menu

e. Scenario 5: Resuming the Game

- As a player, I want to resume the game after pausing so that I can continue playing the song
- Scenario: Resuming a Song
  - Given the game is paused
  - When I press the "Resume" button
  - Then the system should close the pause menu
  - And restart the song playback and falling notes from the paused position

f. Scenario 6: Exiting the Game

- As a player, I want to exit the gameplay interface so that I can return to the Song Page without completing the song
- Scenario: Exiting During Gameplay
    - Given I am playing a song in the "Gameplay Interface"
    - When I press the "Exit" button
    - Then the system should pause the game
    - And display a confirmation popup with "Exit Game" and "Cancel" options
    - When I confirm the "Exit Game" option
    - Then the system should discard my score
    - And redirect me back to the "Song Page"

- Use case diagram:

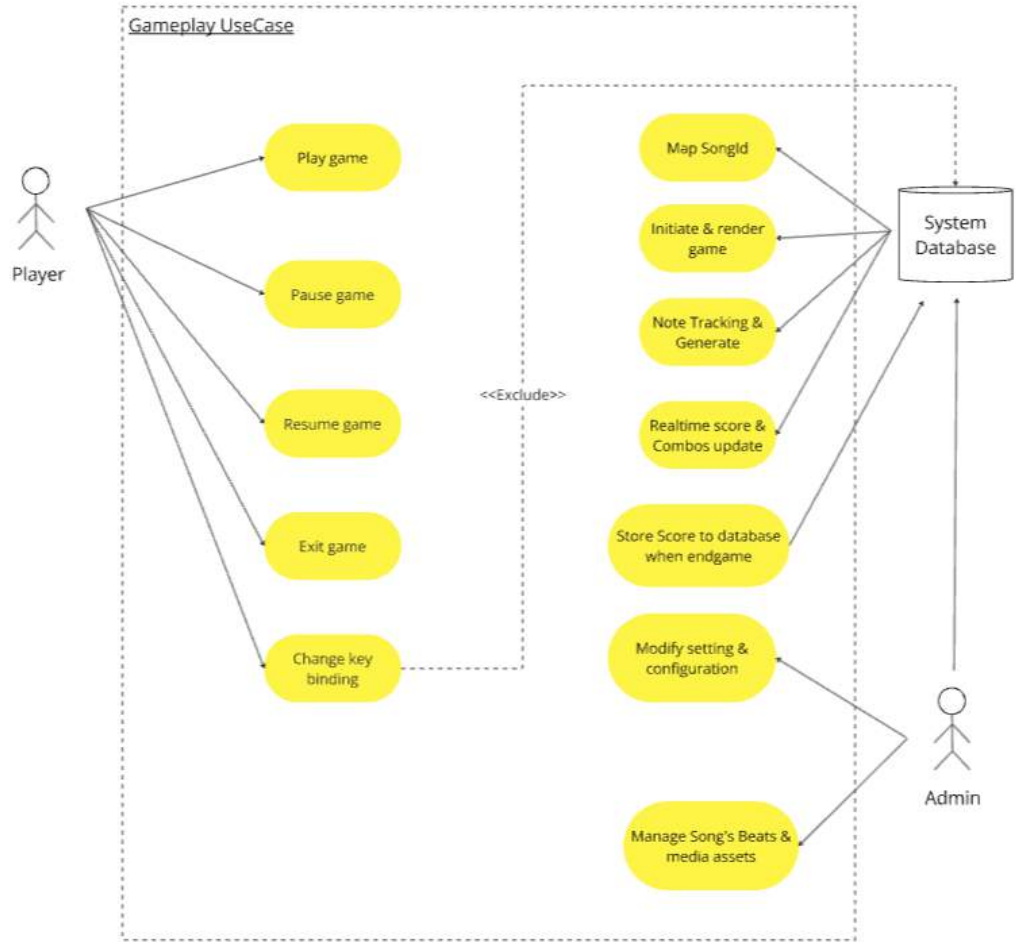

Figure 8: Use Case Gameplay

- **Use case using a table format:**

Table 3: Use Case table for Create CV and Access profile

Use case name:	Gameplay
Actors:	Player, Admin, System Database
Description:	Handles the actual gameplay of a selected song, including score tracking, note evaluation, and result storage.
Preconditions:	• Player must select a song to play. • Game engine must have the song data mapped to songId. • Keyboard inputs must be enabled.
Postconditions:	• Gameplay results are stored in the database. • Player is redirected to the History Page to view scores. • Any game errors are logged.
Normal Flows:	1. System loads the song and gameplay interface. 2. Player interacts with notes using the keyboard. 3. System calculates scores and judgments in real-time. 4. Results are stored after the song ends, and the History Page is displayed.
Alternative Flows:	1. Player pauses the game and resumes later. 2. Player exits the game before completing it. 3. System handles fail states (e.g., health bar depletion).
Exceptions:	• Game fails to load due to corrupted song data. • API call to store the score fails. • Keyboard inputs are unresponsive.
Note and issues:	

1.9.4. Use Case Profile Page

- **User Story:**
  a. Scenario 1: Viewing Profile
  - As a logged-in user, I want to view my profile so that I can see my exp, song record, play time, and achievements
  - Scenario: Viewing Profile Details
    - Given I am on the "Profile Page"

- When the page loads
- Then the system should fetch my user data using the API "/api/user"
- And display my total playtime, experience points, and achievements

b. Scenario 2: Updating Profile Information

- As a logged-in user, I want to update my profile information so that I can keep my account details up-to-date
- Scenario: Successfully Updating Profile
  - Given I am on the "Profile Settings" section
  - When I change my username, email, or password
  - And I click "Save"
  - Then the system should validate the input data
  - And update the database with the new information
  - And display a popup "Profile updated successfully"

- Use case diagram:

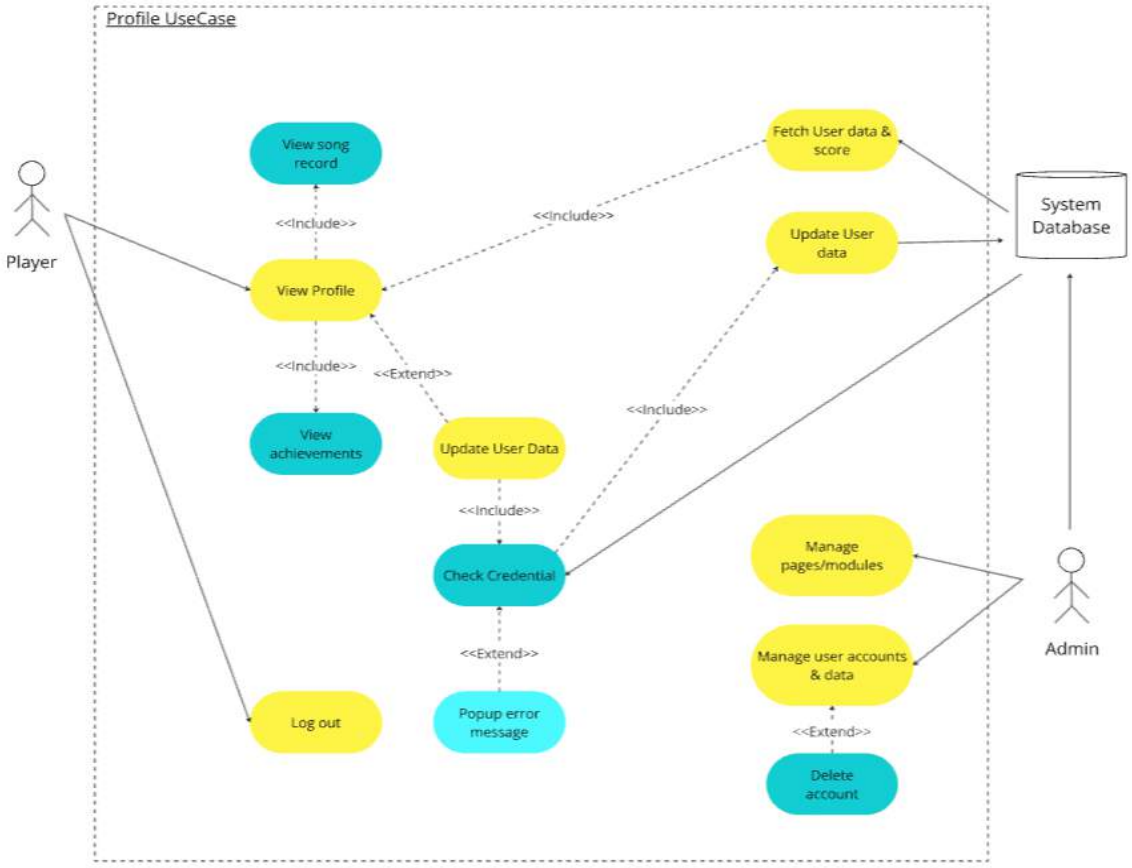

Figure 9: Use Case Profile Page

- **Use case using a table format:**

Table 4: Use case for Save, Download, and Payment

Use case name:	Profile Page
Actors:	Player, Admin, System Database
Description:	Displays player's song records, achievements, and account settings. Allows updates to username, email, or password.
Preconditions:	• Players must be logged in. • APIs to fetch user data must be operational. • The database must support account updates.
Postconditions:	• Updated account details are saved in the database. • Song records and achievements are fetched and displayed. • Players can view or edit their profile.
Normal Flows:	1. Player navigates to the Profile Page. 2. System fetches and displays song records and achievements. 3. Player updates the username, email, or password. 4. System validates and stores updates in the database.
Alternative Flows:	1. Player cancels an update before submitting. 2. System displays an error if invalid input is detected.
Exceptions:	• API call to fetch profile data fails. • Database update for account settings fails. • Network issues interrupt profile updates.
Note and issues:	

1.10 Class Diagram

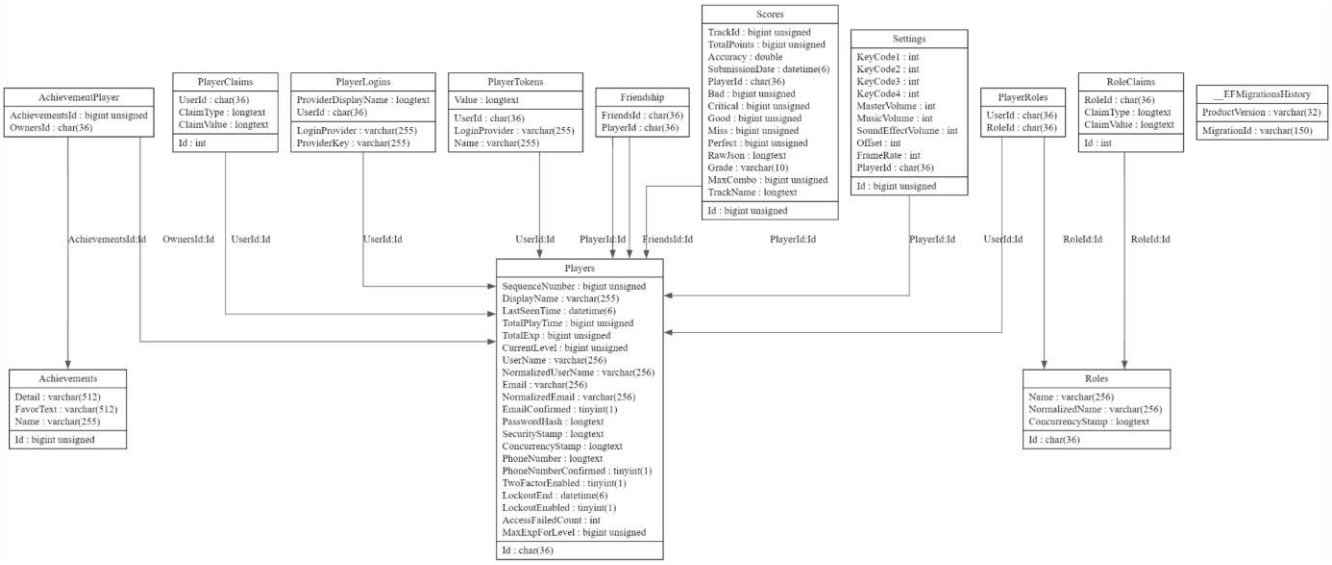

Figure 10: Class Diagram

1.11 Sequence Diagram

- **User Login-Register- Forgot Password Diagram:**

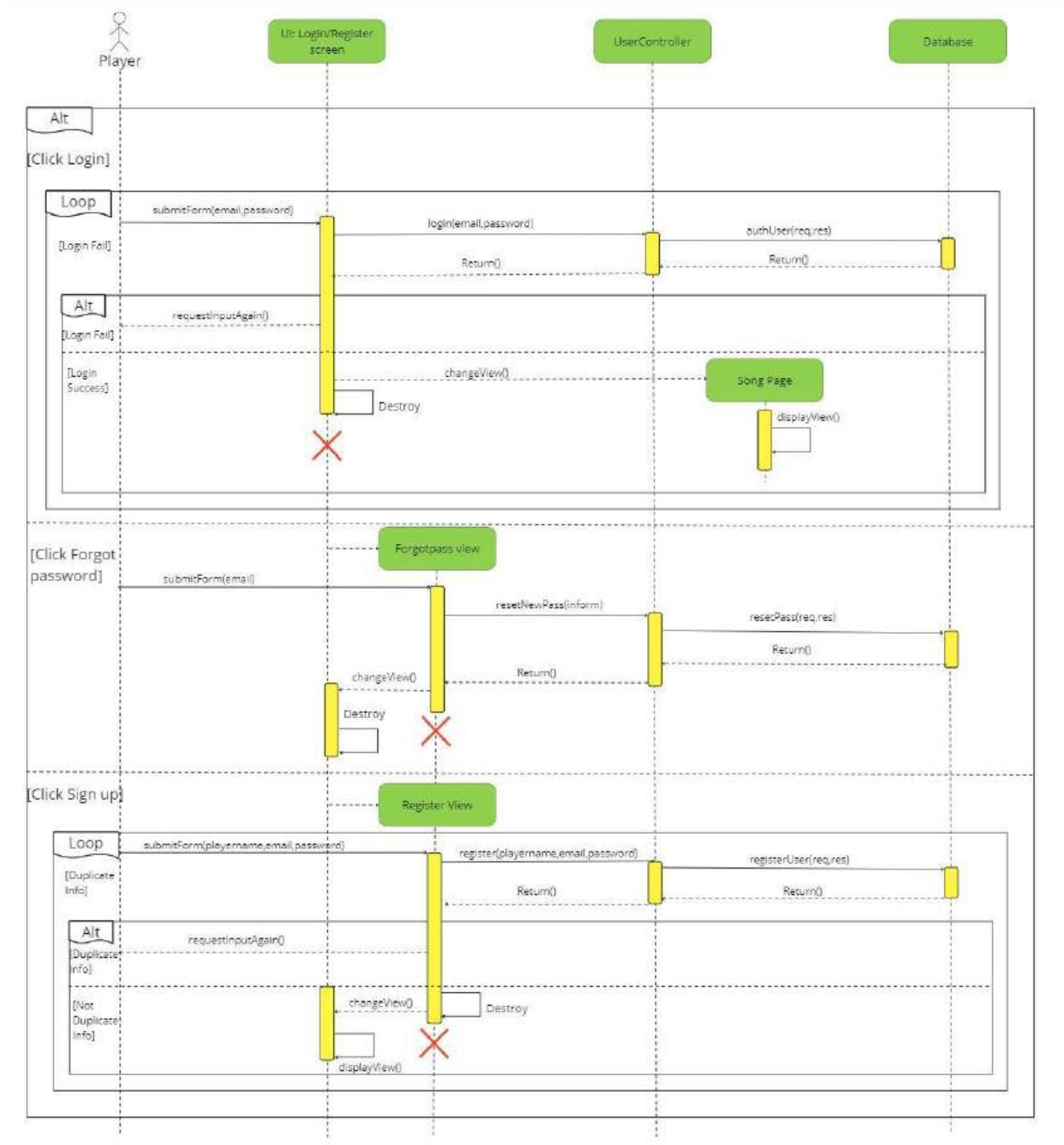

Figure 11: Sequence Diagram Login/Register/Forgot password

- **Main Menus & Gameplay Diagram:**

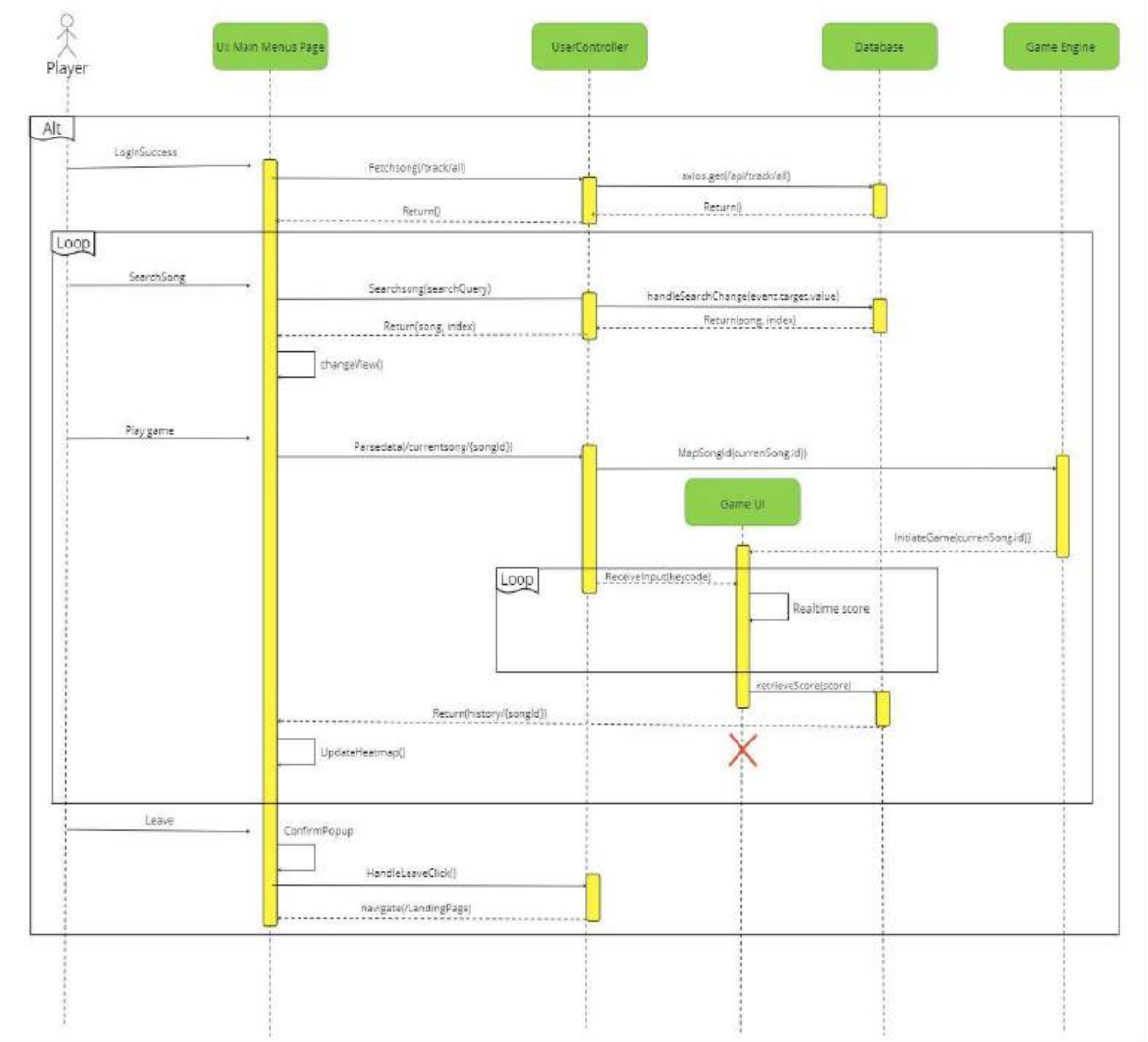

Figure 12: Sequence Diagram Main menu & Gameplay

- **Profile Page Diagram:**

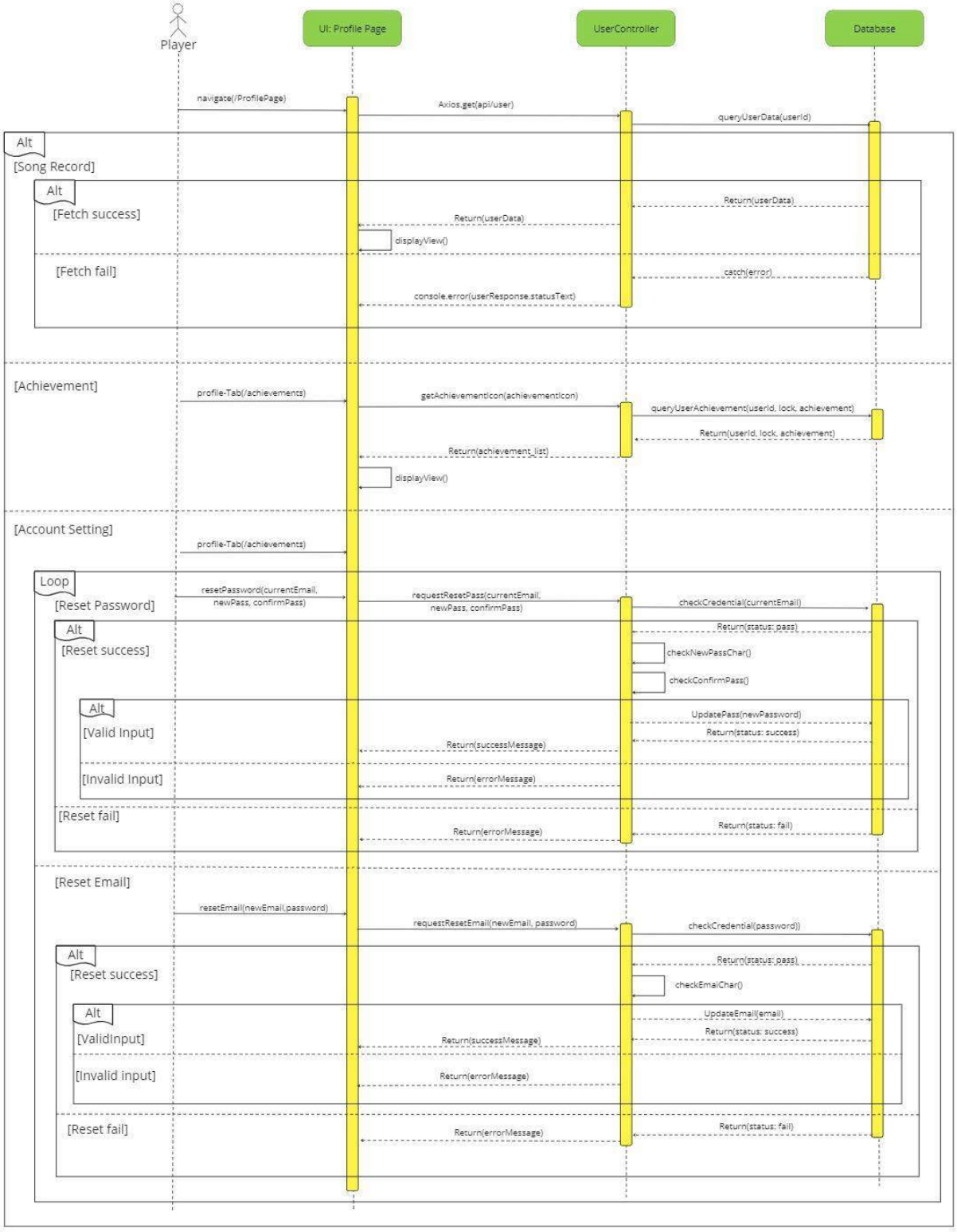

Figure 13: Sequence Diagram Profile Page

Chapter 5: Backend System Architecture

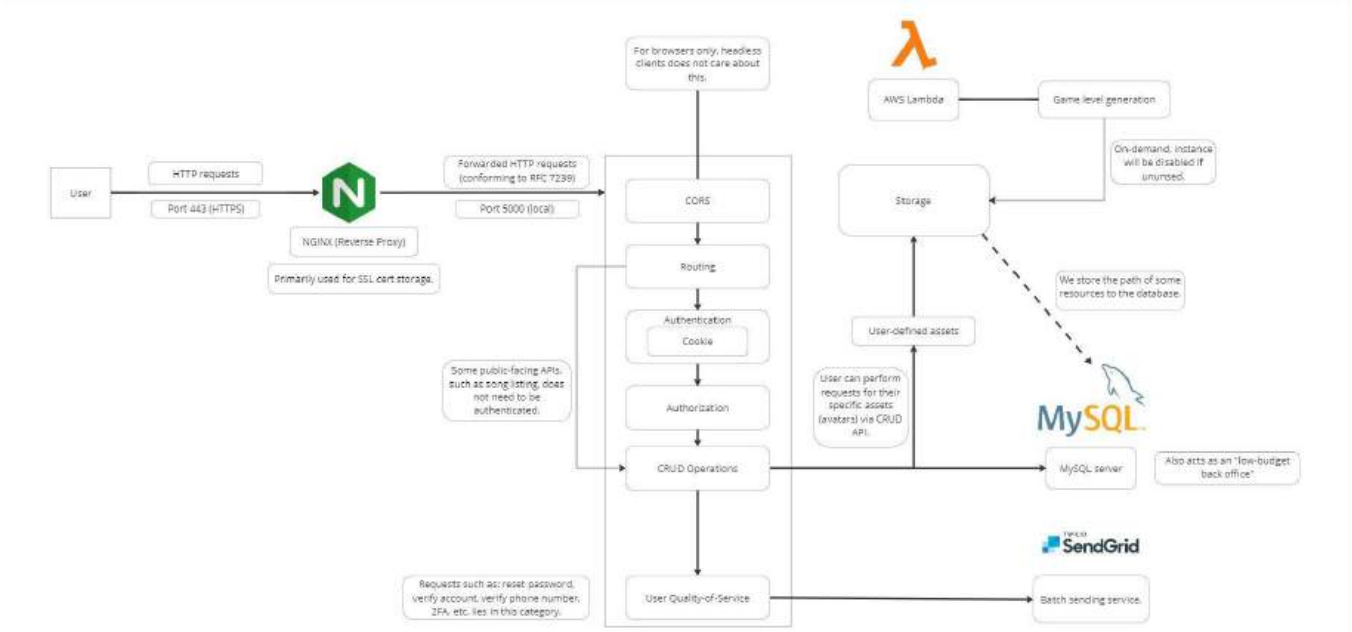

Figure 14: System Design for Backend

User Interaction and Entry Point

- **User Requests**:
  Firstly, users interact with the system by sending HTTPS requests over **Port 443**, which ensures secure communication. These requests serve as the primary entry point for all operations.
- **NGINX Reverse Proxy**:
  Secondly, the requests are forwarded to an **NGINX reverse proxy**, which plays a pivotal role in the architecture.
    - It acts as a gateway to route incoming traffic to backend services.
    - The reverse proxy stores **SSL certificates**, enabling encrypted connections and secure transmission of user data.
    - Furthermore, the proxy ensures that forwarded HTTP requests comply with **RFC 7239** standards, maintaining reliability and consistency in header information.

Backend Core Logic

The backend logic comprises several critical processes to manage incoming requests and execute application logic efficiently.

- **CORS (Cross-Origin Resource Sharing)**:
  To begin with, the backend implements CORS to allow browser-based applications to access resources securely from different domains.
    - This feature is specifically useful for modern web applications, enabling cross-domain communication without compromising security.
    - Notably, headless clients, such as scripts or automated systems, do not rely on CORS for functionality.
- **Routing**:
  Once requests pass the reverse proxy, they are routed to the appropriate service endpoints based on the URL and method specified. This module ensures that the system can handle various functionalities, from authentication to data processing.
- **Authentication**:
  Authentication plays a central role in ensuring the security of user data.
    - The system relies on **cookie-based session management** to identify and authenticate users.
    - Importantly, some APIs, such as those listing publicly accessible data like song collections, are designed to bypass authentication for convenience and performance.
- **Authorization**:
  After authentication, the system performs authorization checks to validate whether users have the appropriate permissions to access specific resources or execute particular actions.
    - Sensitive requests, such as resetting passwords, enabling **two-factor authentication (2FA)**, or verifying accounts, are authorized strictly under this module.
- **CRUD Operations**:
  The backend supports **Create, Read, Update, and Delete (CRUD)** operations to manage resources efficiently.
    - This includes managing user-specific assets such as avatars or profiles.
    - Additionally, it facilitates user-driven quality-of-service improvements, ensuring an optimized user experience.

Data Management and Resource Handling

The system architecture incorporates various components for managing data storage and retrieval seamlessly.

- **AWS Lambda**:
  Firstly, **AWS Lambda** is leveraged for on-demand tasks, particularly for **game level generation**.
    - This serverless compute service activates only when required, ensuring cost-efficiency by deactivating itself when idle.
    - As a result, Lambda optimizes both scalability and resource utilization.
- **Storage**:
    - The backend also includes a storage module for user-defined assets.

    - Users can interact with this storage via the CRUD API to manage assets like avatars and other game-related resources.
    - The system stores file paths and metadata in the database, rather than storing large files directly, to enhance database performance.
- **MySQL Database**:
  Moreover, a **MySQL database** serves as the primary data repository.
    - It stores critical information, including user accounts, gameplay statistics, and paths to uploaded assets.
    - Beyond data storage, the database doubles as a "low-budget back office" for processing operational data.
    - This dual-purpose usage exemplifies the cost-effectiveness of the architecture.

External Integrations

To ensure seamless communication and external functionality, the system integrates with reliable third-party services.

- **Twilio SendGrid**:
  Twilio SendGrid is utilized to handle **batch email services**, which include sending account verification emails, password recovery links, and notifications.
    - By outsourcing this functionality, the system ensures scalability and reliability in handling email-related operations.

Security and Scalability

- **HTTPS Communication**:
  To protect user data, all communication between the user and the system is encrypted using **HTTPS**. This ensures that sensitive information, such as login credentials, is transmitted securely.
- **Dynamic Scaling**:
  Finally, the system leverages dynamic scaling through AWS Lambda to adapt to varying workloads. This approach optimizes performance during high-traffic periods while minimizing costs during idle times.

Chapter 6: Testing

1.12 Testing Strategy

Table 5: Table for testing strategy

	Coder	Customer
Focus on the actor requirement	• Focus on the requirements • Test other unit (components, framework) • Test functional • Test non-functional • Test regression • UAT (User Acceptance Testing) • Check performance • CI (Continuous Integration) • CD (Continuous Deployment)	• UX (User Experience) • Beta testing • Feedback loops • Documentation review • Improvement review • Check user cases (Story) • Localization and Internationalization • Test accessibility • Performance from the customer perspective

Test for coder:

Step 1: Analyze software requirements

Step 2: Test plan outlining objectives, scope, resources, schedule, and deliverables

Step 3: Test scenarios based on requirements

Step 4: Set up the testing environment to mirror the production environment

Step 5: Conduct unit testing to validate individual components.

Step 6: Test interactions and interfaces between integrated components

Step 7: Perform comprehensive testing of the entire system.

Step 8: Functional Testing

Step 9: Non-functional Testing

Step 10: Ensure new changes don't impact existing functionalities.

Step 11: Involve end-users to validate the software against expectations.

Step 12: Release a beta version to a limited user group collect feedback and identify issues.

Step 13: Review and validate user documentation.

Step 14: Test the software deployment process.

Step 15: Implement monitoring tools in the production environment and conduct additional testing as needed.

Step 16: Gather feedback from testing phases and post-release usage and use feedback to improve testing processes.

Step 17: Summarize testing activities, results, and lessons learned in a test closure report.

Customer test:

Step 1: Communicate the purpose of testing.

Step 2: Define the scope of testing and what is expected from customers.

Step 3: Set realistic expectations regarding their role in the testing process.

Step 4: Provide an overview of the testing timeline and milestones.

Step 5: Ensure customers understand how to use the software and report issues.

Step 6: Customers have the necessary access and environments.

Step 7: Provide customers with realistic test data.

Step 8: Streamline the bug-reporting process.

Step 9: Customers focus on key scenarios.

Step 10: Establish effective communication channels.

Step 11: Encourage active participation through incentives and recognition.

Step 12: Keep customers informed and engaged.

Step 13: Conclude the testing phase and gather insights.

1.13 Test Cases and Results

Table 6: Table of Test Cases and Results

Test Scenario	Test Case	Pre Conditions	Test step	Test Data	Expected Result	Actual Results	Pass/Fail
User Registration	Register with valid credentials	The registration popup is displayed; valid email and password formats are predefined	1. Enter valid email, password, and username. 2. Click the "Sign Up" button.	Email: test@gmail.com Password: Valid123! Username: Player1	Account is created successfully; "Registration Success" popup is displayed.	Matches expected result.	Pass
	Register with an invalid email format	Registration popup is displayed.	1. Enter an invalid email (e.g., missing @). 2. Click "Sign Up."	Email: ahihi@mai.com Password: Valid123! Username: Player1	Error message "Invalid Email" is displayed.	Matches expected result.	Pass
User Login	Login with valid credentials	Login popup is displayed; user exists in the database.	1. Enter valid email and password. 2. Click the "Login" button.	Email: test@gmail.com Password: Valid123!	User is redirected to the "Song Page."	Matches expected result.	Pass
	Login with invalid credentials	Login popup is displayed.	1. Enter incorrect email or password. 2. Click "Login."	Email: wronguser@gmail.com Password: WrongPass	Error popup "Invalid email or password" is displayed.	Matches expected result.	Pass
Main Menu - Song List	Fetch available songs	User is logged in and on the "Song Page."	1. Load the page. 2. Fetch data from API /api/track/all.		Songs are displayed in the sidebar.	Matches expected result.	Pass

Profile Management	View profile	User is logged in and on the "Profile Page."	1. Navigate to "Profile Page." 2. Fetch user data via /api/user		Profile details (playtime, experience, achievements) are displayed.	Matches expected result.	Pass
	Update profile information	User is logged in; valid input provided.	1. Navigate to "Profile Settings." 2. Update email, username, or password. 3. Click "Save."	Email: updated@gmail.com Username: NewPlayer	Data is validated, saved in the database, and a "Profile updated successfully" popup appears.	Matches expected result.	Pass
Song Page	Fetching all available songs	User is logged in; API /api/track/all is operational.	1. Navigate to "Song Page." 2. The system calls API /api/track/all.		All available songs are displayed in the sidebar	Matches expected result.	
	Search for a song	User is logged in; songs are fetched and displayed.	1. Enter a song title in the search bar. 2. Click the "Search" button.	Search Term: Song A	Song list is filtered to display only matching songs.	Matches expected result.	
History Page	View gameplay history	User is logged in and on the "History Page."	1. Load the "History Page." 2. Fetch data from APIs /api/score/{songId} and /api/score/recent.		The history table displays recent scores and best scores for each song.	Matches expected result.	

Store Page	Display store items	User is logged in and on the "Store Page"; API for fetching store items is operational.	1. Navigate to "Store Page." 2. Fetch store items via API /api/store/items.		Store items (e.g., avatars, themes) are displayed	Matches expected result.	
Song Page - Play Song	Start playing a selected song	User is logged in; song exists in the system.	1. Click the "Play" button on a song. 2. The system passes the songId to the game engine.	Song ID: 12345	The game interface loads with the selected song.	Matches expected result.	
Log out	Successfully logout	User is logged in and on any page (e.g., Song, History, Store).	1. Click the "Leave" button. 2. Clear session data. 3. Redirect to the landing page.		User session is cleared, redirected to login page, and "Successfully logged out" popup appears.	Matches expected result.	

Chapter 7: Deployment and Maintenance

1.14 Deployment Strategy

We plan to have our software to be deployed using the following steps:

- **Step 1 (Testing):** To make sure the program is reliable, functional, and easy to use, we have to test the software whether it is both manually and automatically then give solutions to resolve any potential flaws or errors.
- **Step 2 (Version Control):** We use Git to track and manage the changes to the code. Ensure the latest stable version of the application is on the main branch.
- **Step 3 (Deployment):** we have used tool named ngrok to utilize a program to connect to a private server and copy the application files to the required place. To further expose the program to the internet, we employ port forwarding to connect our private local server.

1.15 Maintenance Plan

We plan to have our software to be maintained using the following steps:

- **Regular Updates:** Based on user feedback and market need, we want to update the application's dependencies to the most recent versions and add new features or enhancements.
- **Monitoring:** We have used many monitoring tools to evaluate and analyze the software's performance so that if there are any problems or issues arise, the software will announce the developers to fix.
- **User and Stakeholder Feedback:** We have implemented a method for users to report defects or request enhancements, and examine and act on this input regularly. Furthermore, our team will regularly collect and evaluate customer feedback to determine user requirements and expectations, as well as to increase user happiness and loyalty.

Chapter 8: Minimum Viable Product

1, Users access to Landing Page, if they press start game without having logged in, it popup an error

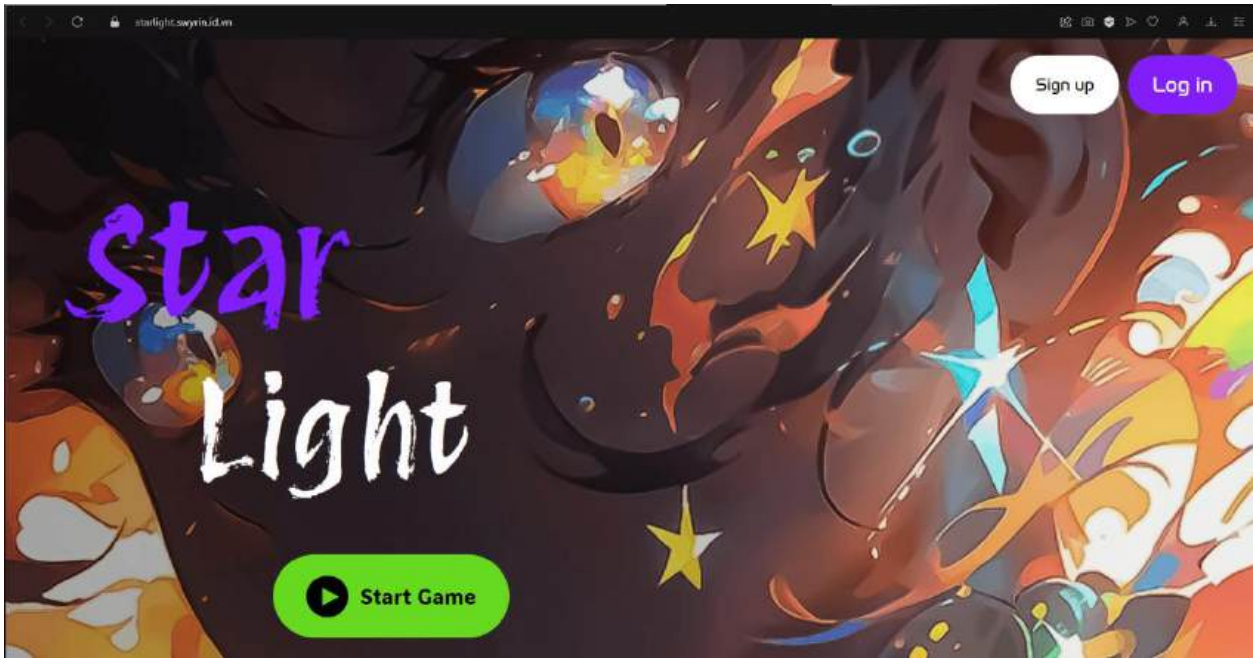

Figure 15: Demo Landing Page

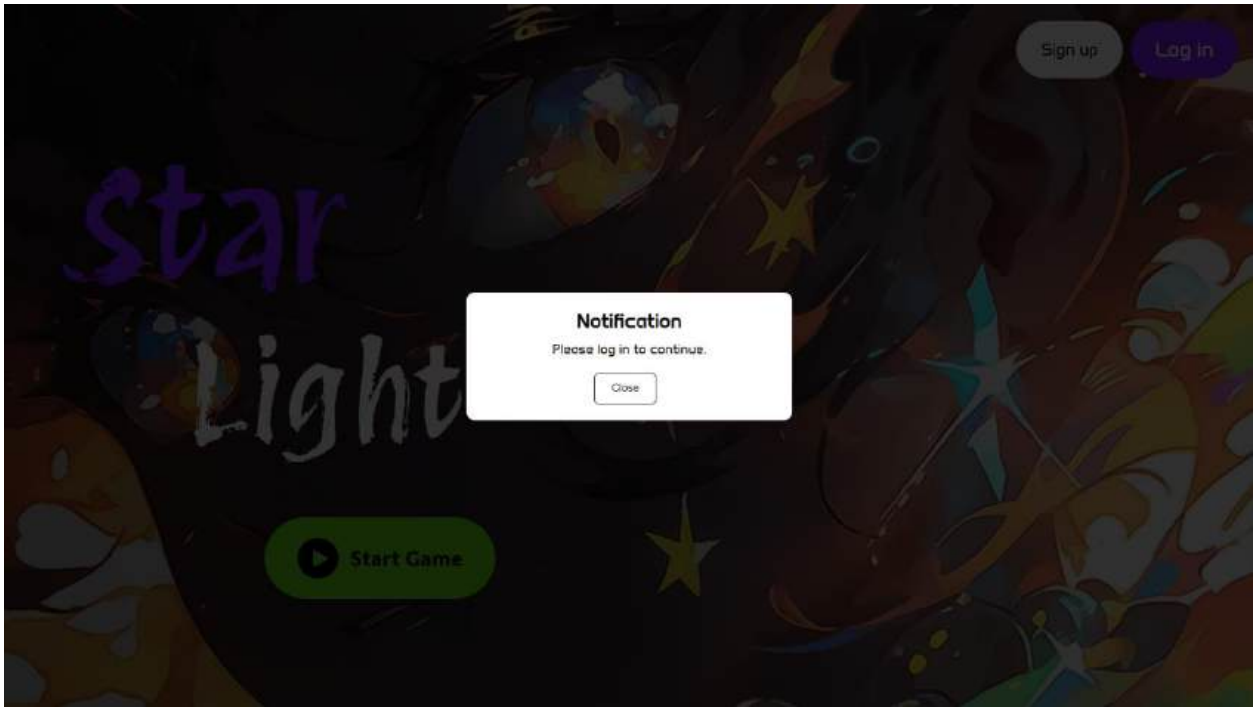

Figure 16: Demo Landing Page - Notification

2, Secondly, If User choose to access their account, they click the Logvin

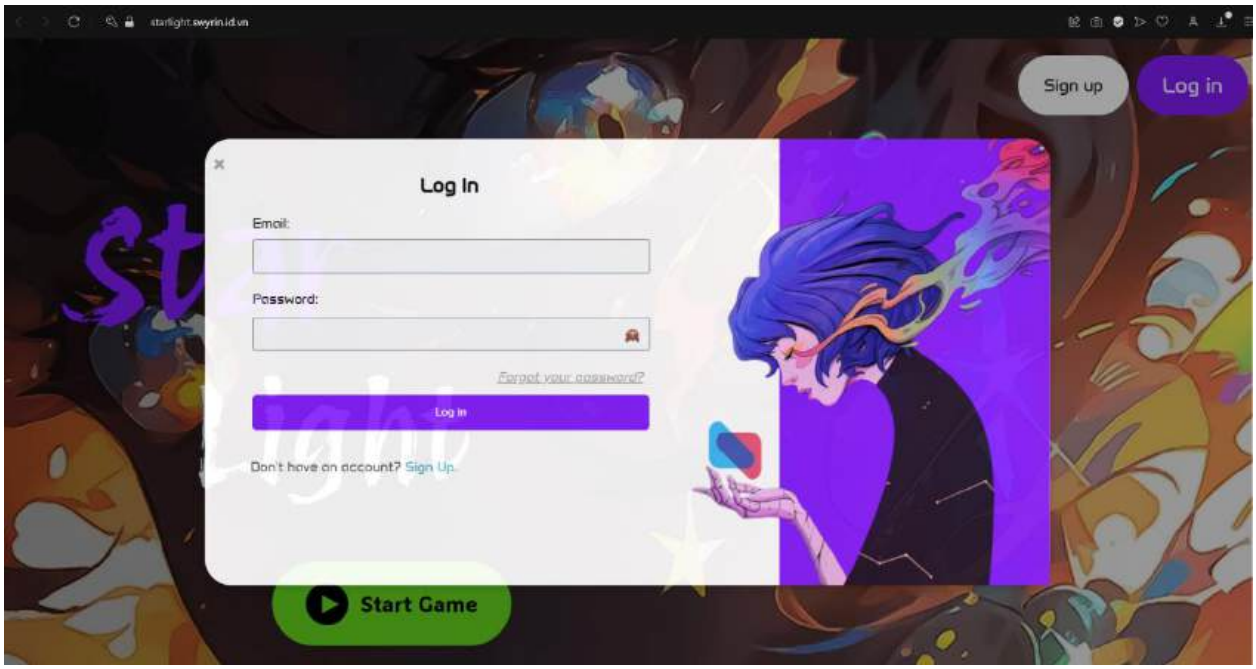

Figure 17: Demo Login

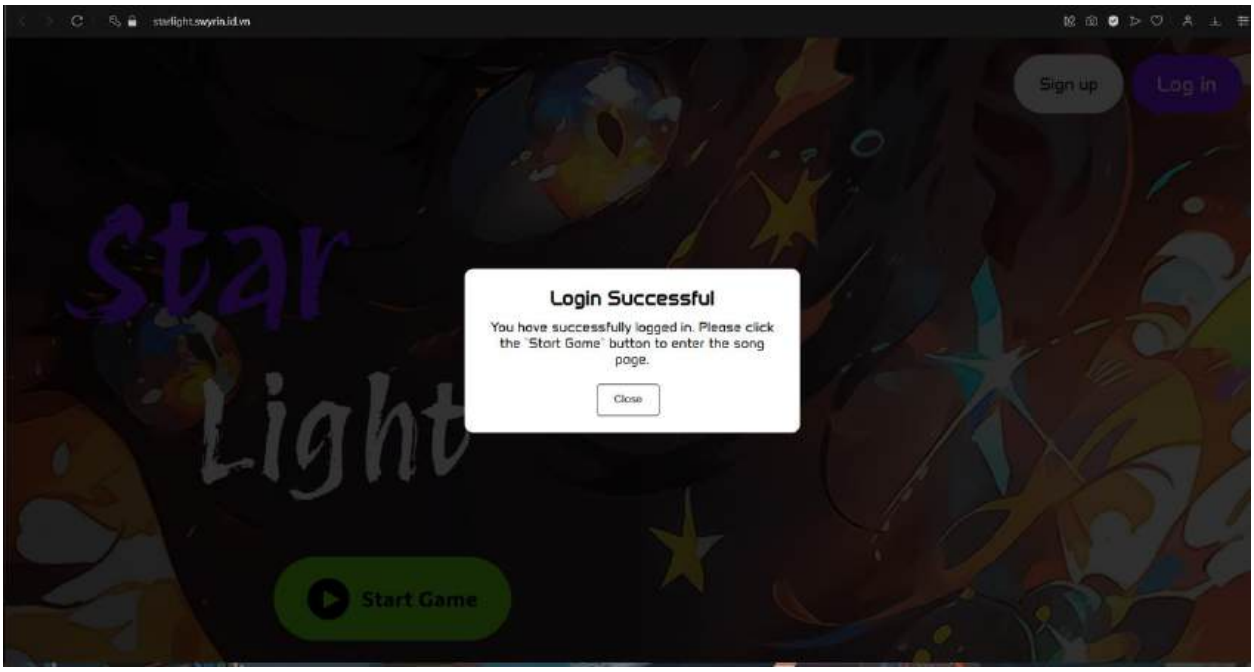

Figure 18: Demo Login - Notification

3, If the User does not have an account, They can Sign Up

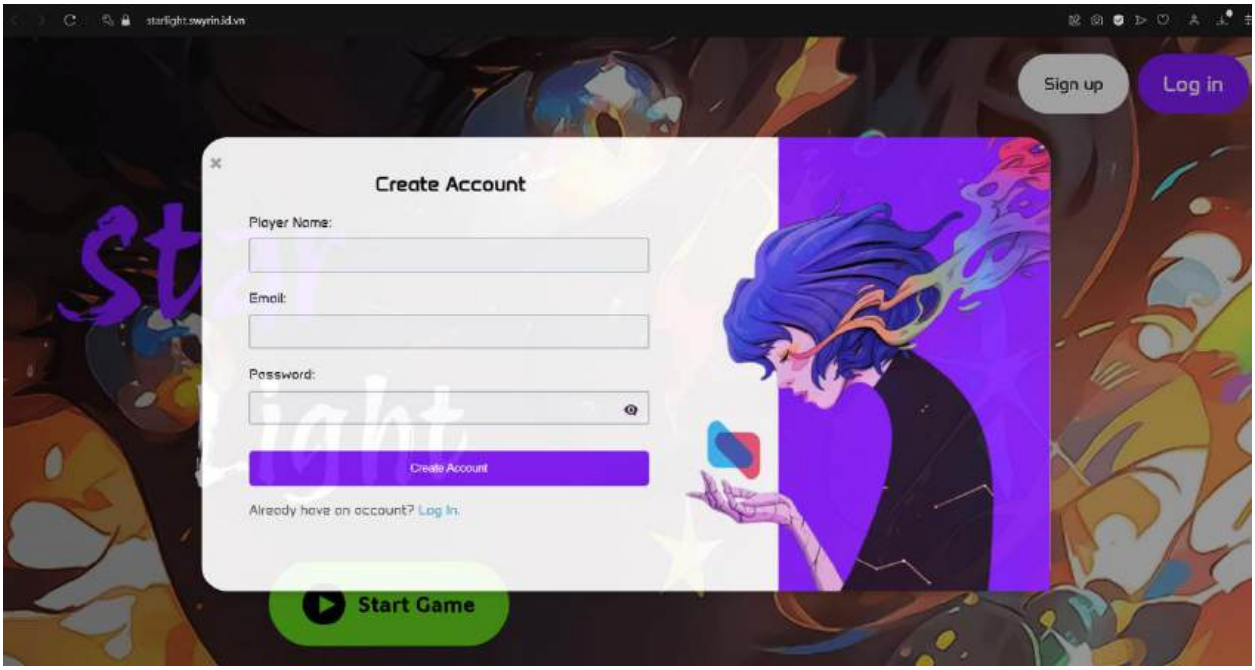

Figure 19: Demo Register

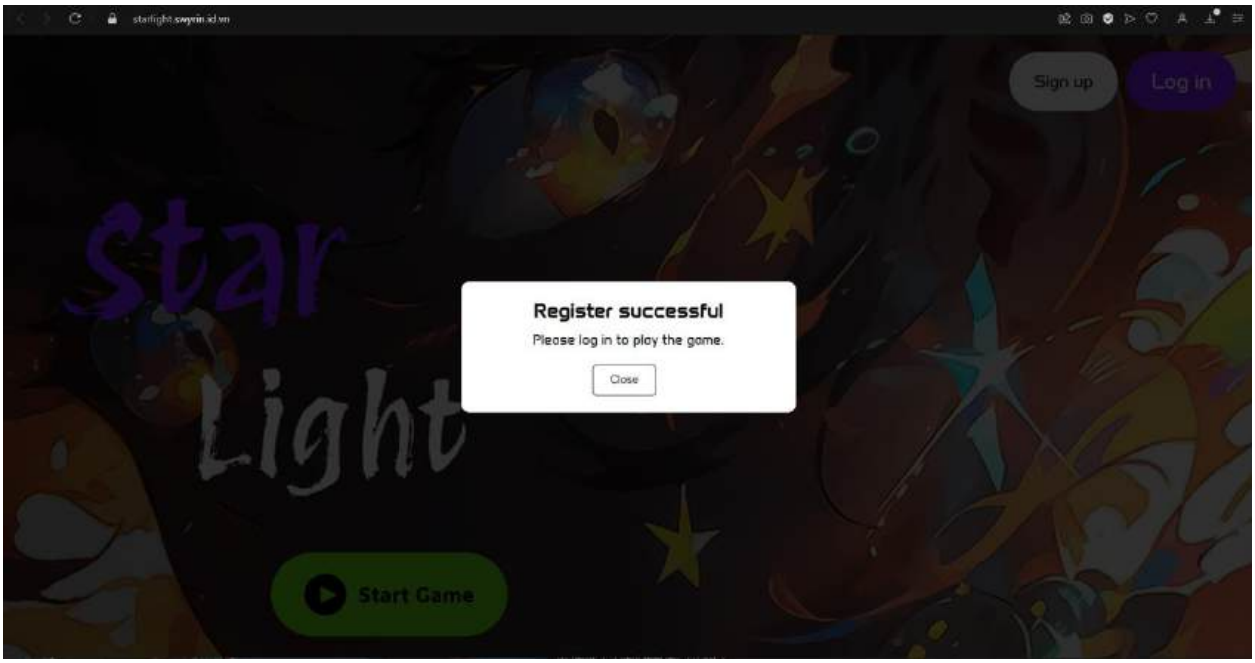

Figure 20: Demo Register - Notification

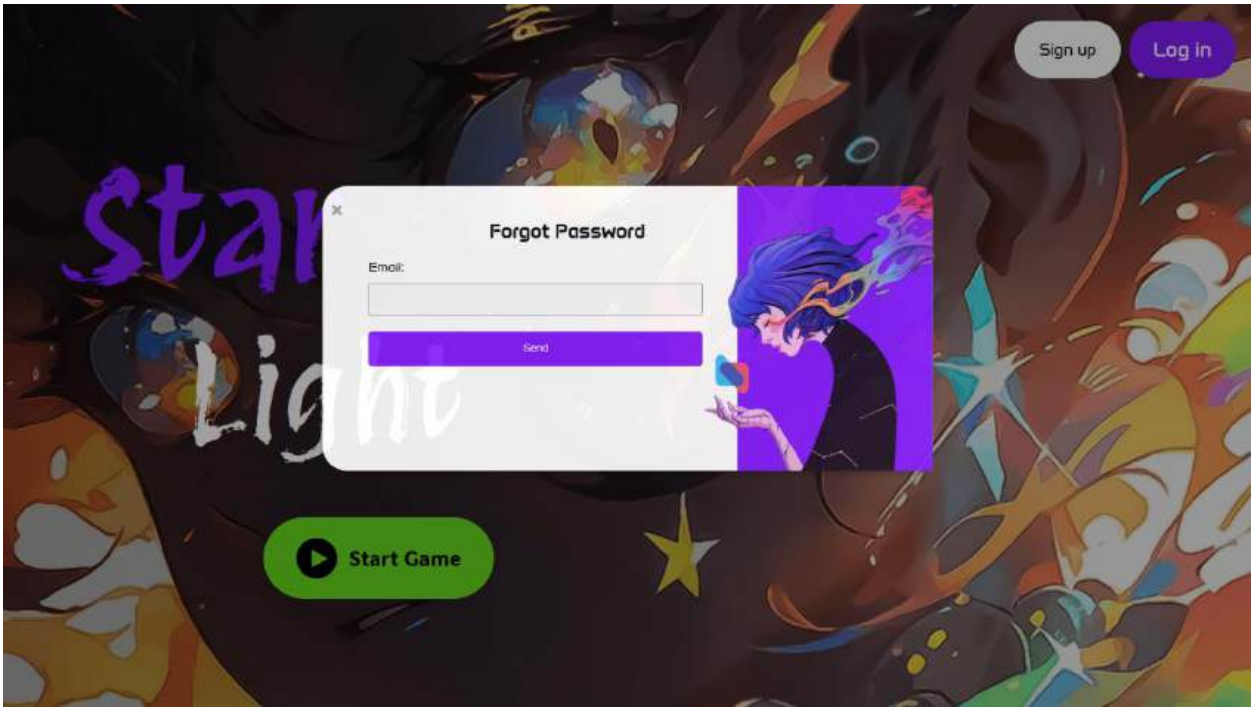

Figure 21: Forgot Password

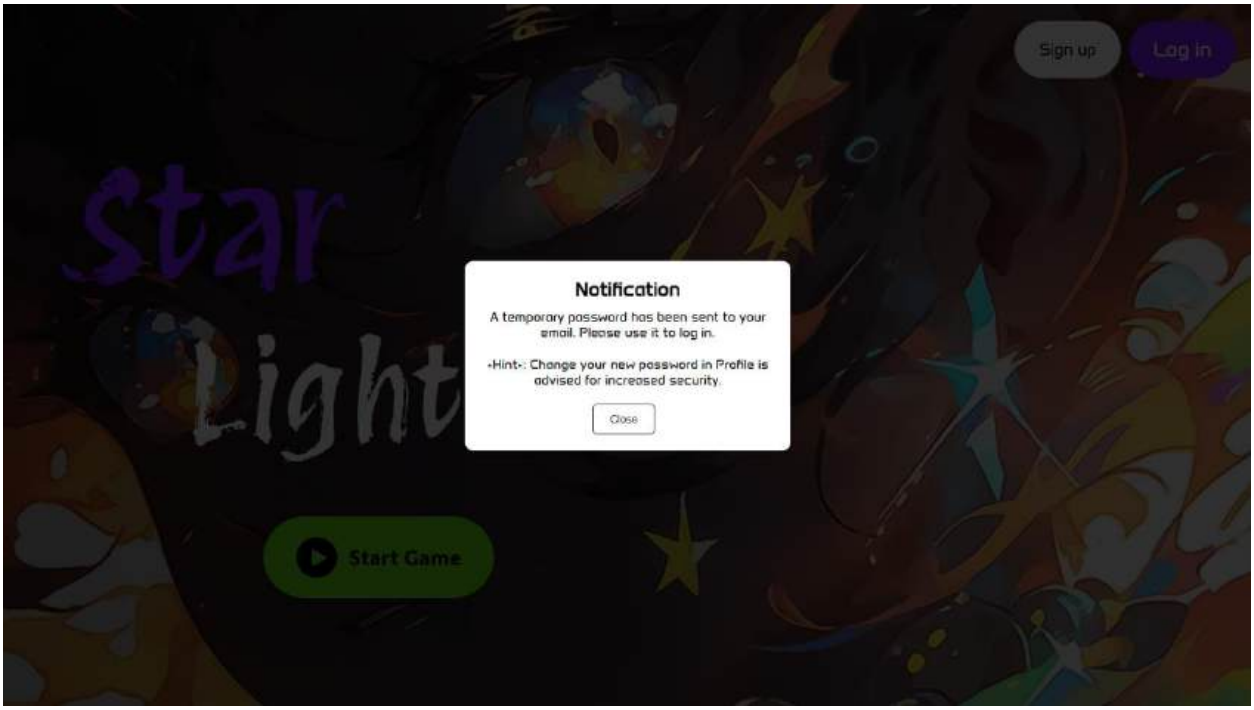

Figure 22: Forgot Password - Notification

4, After successfully Login, the system redirects user to Song Page with fullscreen, where they can open the burger menu to search or select song track, move back and forth the song using the keyboard, shift to another page via Header, access profile page via user avatar, Leave game, or Play Game

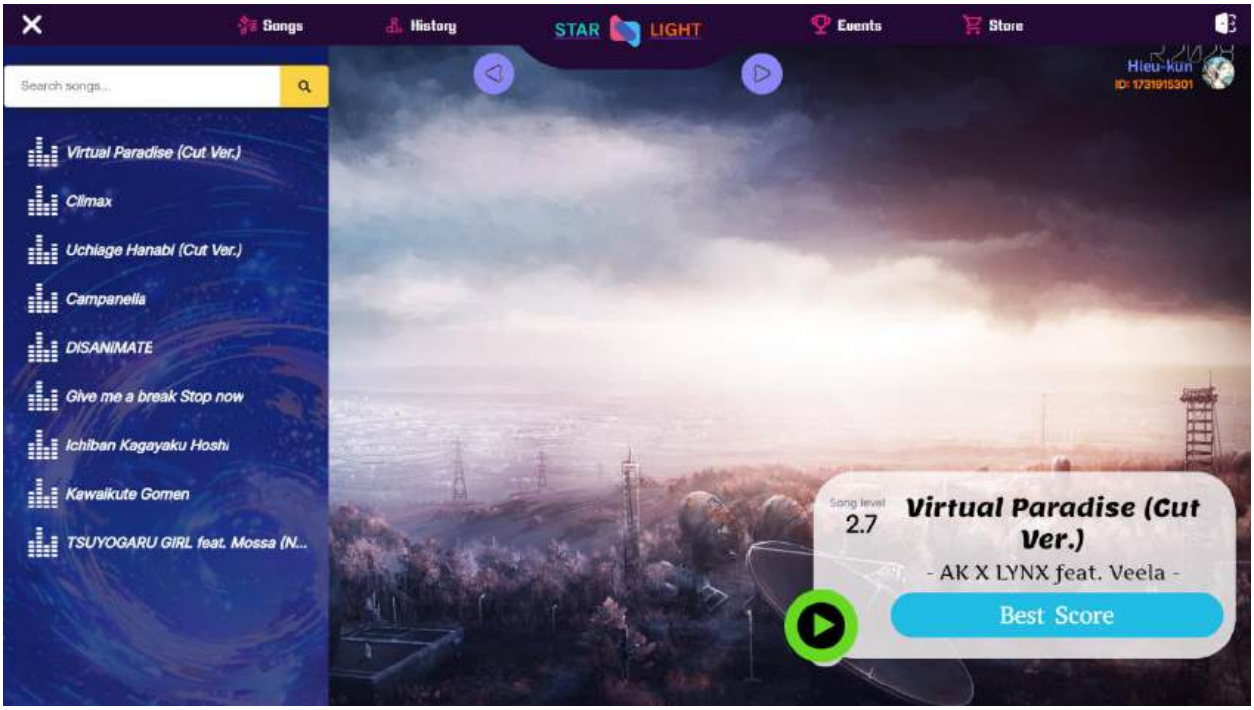

Figure 23: Demo Song Page

5, In History Page, the user can use all the functionality of the header. Thus, they can interact with the heatmap to view their overscore and beat the accuracy they scored in an interval of time

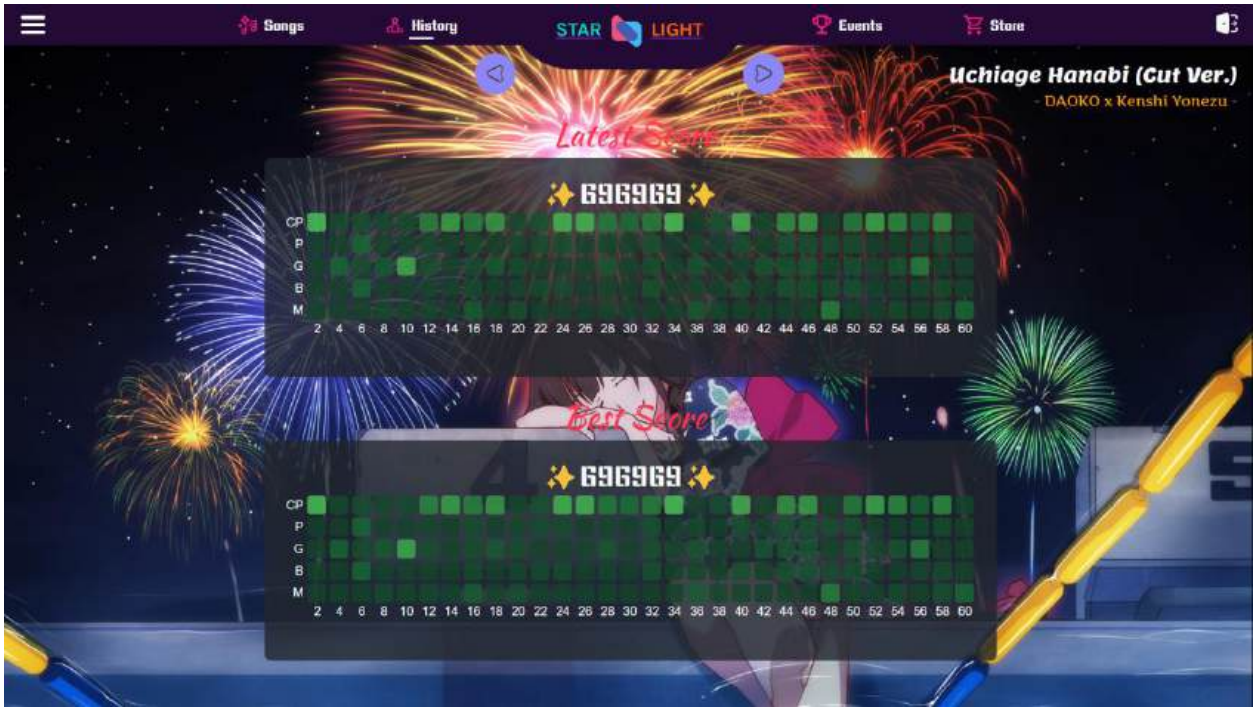

Figure 24: Demo History Page

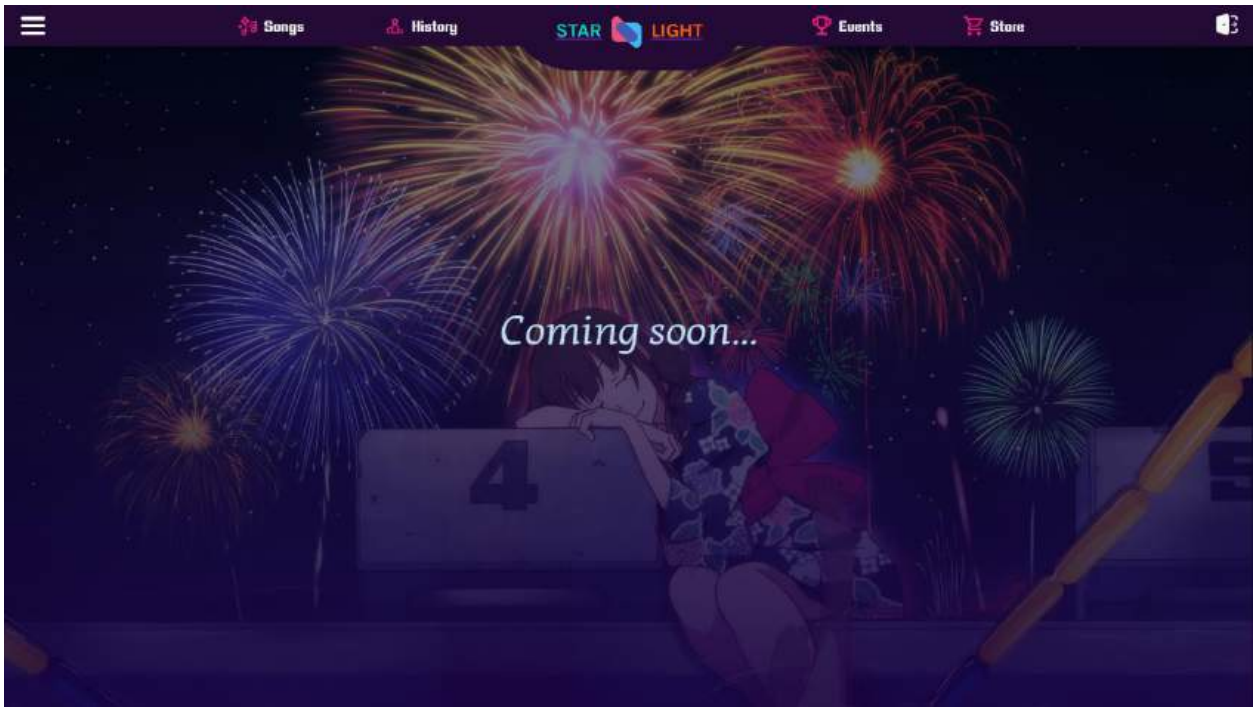

Figure 25: Demo Event & Store Page

6, From Song Page, clicking Play button will navigate user to corresponding song gameplay

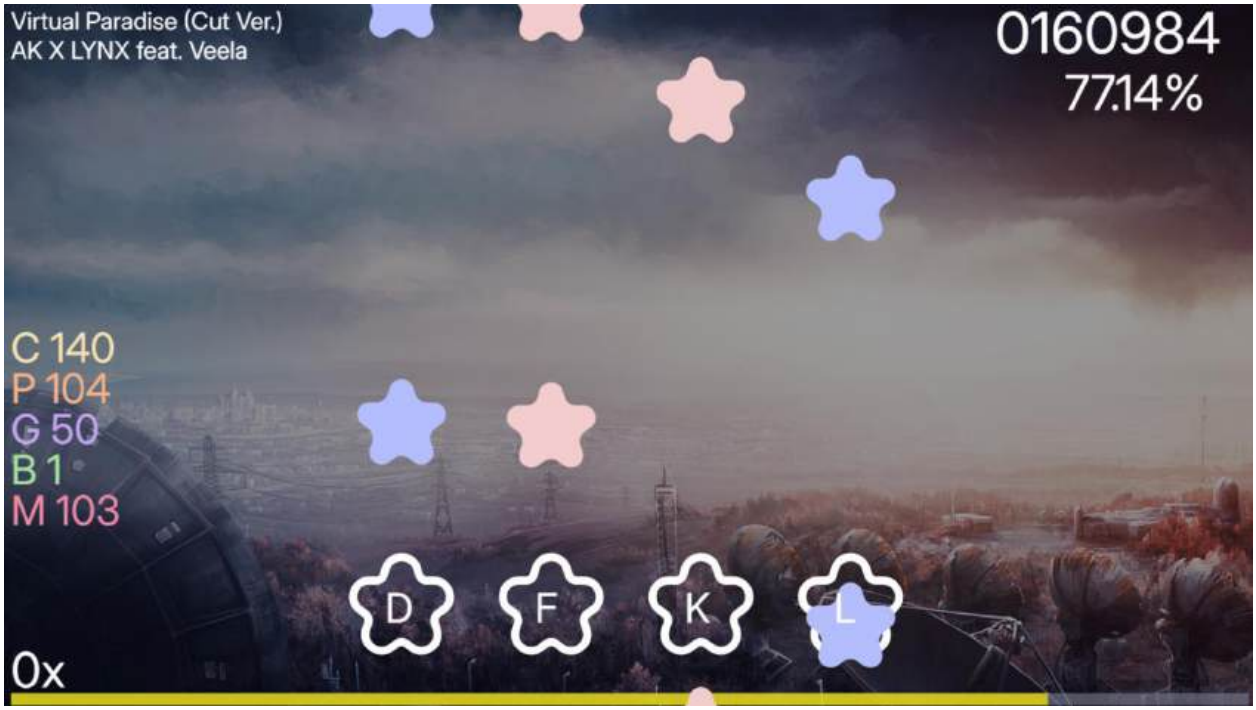

Figure 26: Demo Gameplay

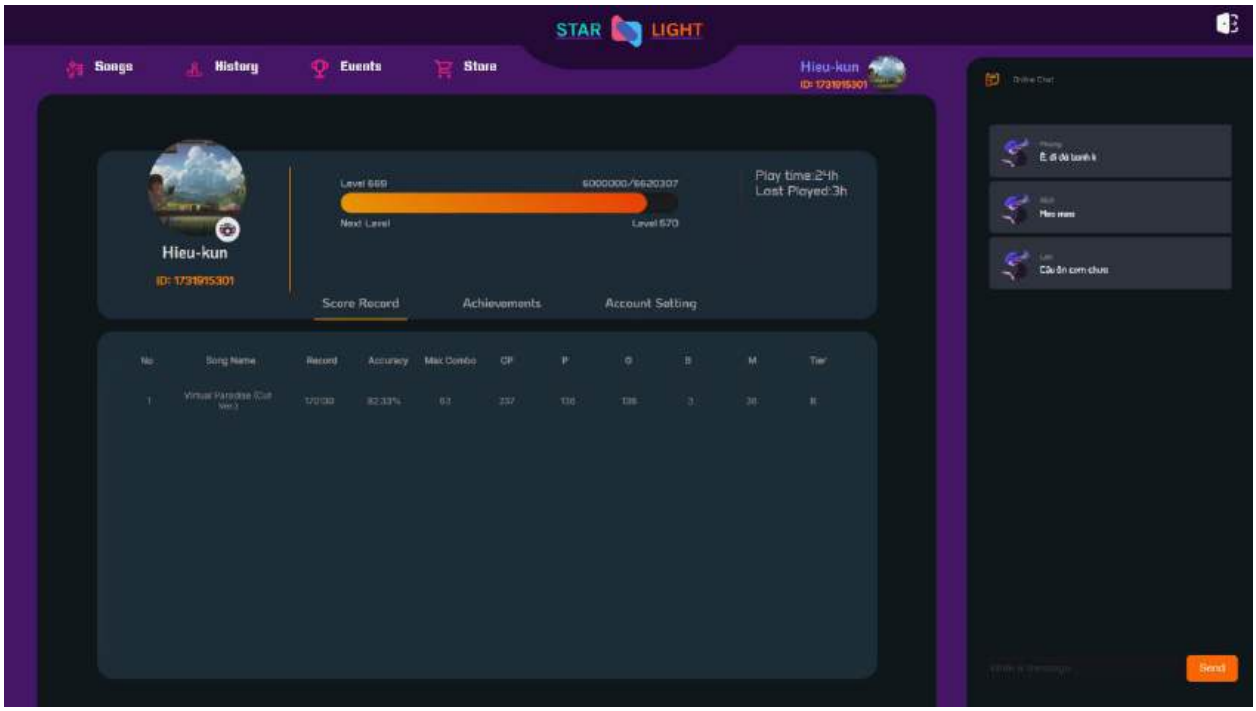

Figure 27: Demo Profile Page - Song Record

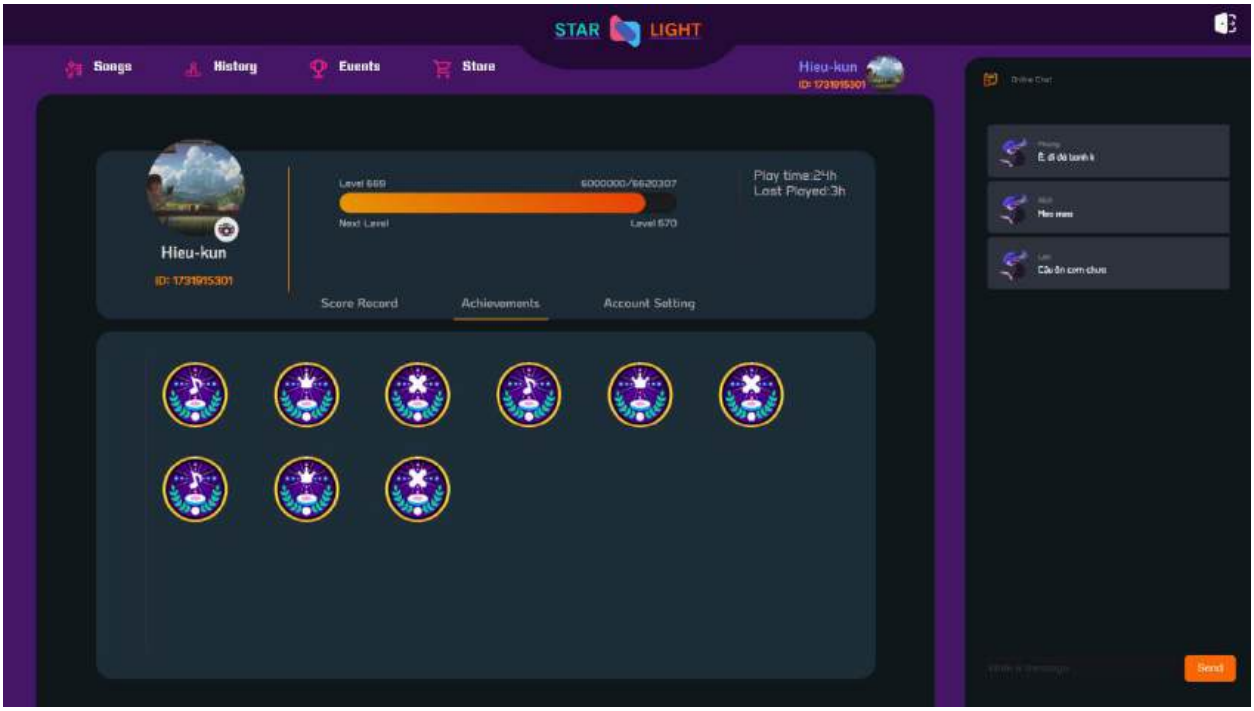

Figure 28: Demo Profile Page - Achievement

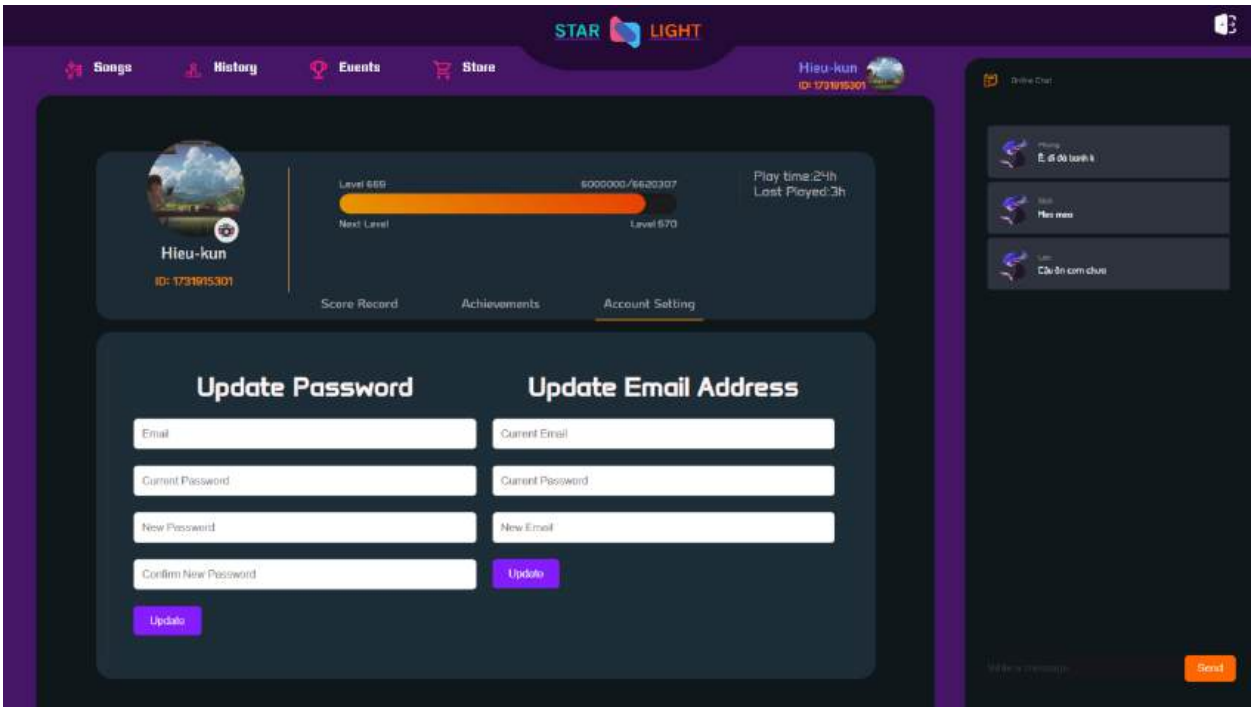

Figure 29: Demo Profile Page - Account Setting

Chapter 9: Achievements and Future Plan

Achievements
During the development of "Star Light," several milestones and successes were achieved that set a strong foundation for future growth and innovation. Key achievements include:

- **Core Gameplay Mechanics**: Implementing the core mechanics of a vertically scrolling rhythm game (VSRG) was a primary accomplishment, incorporating precise timing, varied difficulty levels, and mood-based gameplay modes. This functionality allows users to experience and enjoy rhythm gaming in a web-based format, accessible without installations.
- **Real-Time Feedback and Heatmap Dashboard**: One of the standout features is the real-time feedback system that tracks player performance with high accuracy. The feedback integrates with a heatmap dashboard, allowing players to visually gauge timing and accuracy over gameplay intervals. This feature enhances the player experience by giving them detailed insights into their strengths and areas for improvement.
- **User Data Management and Progress Tracking**: Successfully implementing a secure backend that manages user profiles, login credentials, and progress tracking was another key achievement. By establishing a database to store user data and scores, players can monitor their progress and compare performances across different levels.
- **Interactive Music Library and Configurable Settings**: The project included developing a music library with selectable songs and levels, enabling players to choose tracks that match their preferences. Additionally, the inclusion of configurable key bindings allows players to tailor the controls to their comfort, improving accessibility for users with different gameplay styles.
- **Role-Based Access Control for Admins and Players**: The implementation of role-based permissions, particularly for the Admin role, ensures secure and efficient management of the platform. Admin tools for user account management, song updates, and monitoring help streamline maintenance tasks, allowing administrators to oversee user interactions, monitor system performance, and address issues proactively.

Future Work
Despite these successes, there are many promising areas for development that will enhance "Star Light" and expand its functionality. Planned future work includes:

- **Offset Adjustment Feature**: Introducing a customizable offset adjustment setting will allow players to fine-tune their input timing to account for device or individual latency preferences. This improvement will enhance gameplay accuracy and appeal to competitive players who prioritize precision.
- **Peer-to-Peer Multiplayer Mode**: Multiplayer functionality is an anticipated feature that will allow users to compete against each other in real-time. Integrating a peer-

to-peer network setup will facilitate multiplayer sessions, and the planned use of WebSockets will enable smooth, low-latency communication. The multiplayer feature will increase user engagement by adding a social and competitive dimension, encouraging players to challenge friends or compete in online matches.

- **Community-Generated Levels**: Allowing users to create and share custom beat maps and levels will foster a community-driven ecosystem within "Star Light." This feature will involve designing tools that let players edit, save, and publish their own levels, which others can play and rate. Community levels will add variety, boost engagement, and encourage creative participation.
- **Enhanced Scoring and Combo Systems**: Future updates to the scoring system will include combo bonuses and a more competitive "Tournament" mode, with stricter judgment criteria and heightened difficulty. This mode will reduce the timing windows for accurate inputs, providing an additional challenge for advanced players who seek competitive scoring.
- **Expanded Mood and Difficulty Options**: Adding more nuanced moods and difficulty levels will cater to a broader audience, from beginners to expert players. Future iterations may also incorporate custom BPM (beats per minute) options and mood settings that allow players to experiment with different rhythmic intensities, increasing the personalization options available.
- **Advanced Visual and Audio Settings**: Planned improvements to visual and audio settings will allow users to adjust visual effects, frame rates, and sound volumes with more precision. This update will include accessibility options, such as colorblind mode and visual contrast adjustments, to make the game inclusive for users with varying needs. Additionally, latency calibration will allow players to adjust audio-to-visual synchronization based on their devices.
- **Improved User Profile and Achievement System**: To enhance player engagement, the user profile will feature an expanded achievement system that allows players to unlock rewards or badges for reaching specific milestones. Achievement icons and tooltips detailing progress and scores will incentivize continued play, with higher achievements highlighting top-performing players.

Chapter 10: Conclusion

In summary, "Star Light" represents a well-rounded web-based rhythm game that delivers an engaging and dynamic music gaming experience. The app provides core functionalities, including real-time gameplay feedback, customizable key bindings, and a user-friendly interface, all developed using modern web technologies and an agile development approach. By focusing on user-centered design and incorporating a modular architecture, the game is prepared for future expansion with features such as multiplayer mode and community-level sharing.

The combination of entertainment, skill-building, and user interaction positions "Star Light" as a promising addition to the genre of rhythm games. The team is committed to refining and expanding the game, building on its initial success to develop an even more

compelling and versatile gaming platform that meets the evolving needs of the gaming community.

Chapter 11: References

- GitHub Link: https://github.com/team-nameless/starlight.git
- Official Game Domain: starlight.swyrin.id.vn
- Miro (Diagram workspace): Starlight Miro Drawing