

4/25/2024

TECHNICAL FINAL DELIVERABLE

GAME NIGHT RECOMMENDER

PREPARED BY: GROUP 22

ADAM SHOAIB KAREEM SAB (AKAREEMS)

MANISHA MALHAR RAO DESHPANDE (MDESH10)

DIYA ROSHAN SANGHVI (DSANGHV3)

NAHEER FATIMA (NNOLAS33)

SOWMYA TANIKONDA (STANIKON)

SPONSOR:

PROF. RUBEN ACUNA

TABLE OF CONTENTS

1. Project Description.....	2
a. Overview.....	2
b. Key Requirements.....	2
c. Deliverables.....	6
d. Acronyms and Abbreviations.....	10
2. User Manual.....	12
3. Installation Guide.....	14
4. Issues.....	18
5. Outlook.....	22
6. Video Presentation.....	25

1. PROJECT DESCRIPTION

1.1. OVERVIEW

Our team has developed the Game Night Recommender Tool for group play among friends. This tool streamlines the process of selecting video games by integrating automatically gathered data and user-provided preferences. The final deliverable is a user-friendly interface that offers ranked game recommendations accompanied by explanations.

Existing challenges in choosing suitable video games for group play have prompted the need for this tool. By leveraging both automatically gathered data and user input, the tool addresses issues related to individual preferences, game availability, and player online status. Its goal is to enhance the overall gaming experience for users.

The Game Night Recommender Tool operates within the gaming and social interaction domain, targeting groups of friends who enjoy multiplayer gaming. It integrates with platforms like Discord for seamless communication and leverages APIs such as Steam for game data retrieval. The motivation behind this project is to create a more efficient and enjoyable gaming experience for users.

Primary users of the tool are groups of friends engaging in multiplayer gaming sessions. They range from casual gamers to enthusiasts and are likely familiar with gaming platforms and communication tools. The tool caters to their diverse preferences and gaming habits, providing personalized recommendations tailored to their various contexts such as collective game ownership, total and recent playtime, ratings, interest and genre preferences.

The Game Night Recommender Tool has been developed using the MERN stack, which includes MongoDB, Express.js, React, and Node.js. This stack was chosen for its efficiency in handling asynchronous requests and its scalability. React is utilized for the frontend to create a responsive and dynamic user interface, while Express.js and Node.js form the backbone of the server-side logic. MongoDB serves as the database, offering a flexible schema that accommodates the diverse data needs of our application. This technology stack ensures that the tool is robust, maintainable, and capable of evolving with future enhancements.

To conclude, the Game Night Recommender Tool addresses the challenge of selecting video games for group play by integrating automatically gathered data and user preferences. The tool enhances the overall gaming experience by providing personalized recommendations in a user-friendly interface.

1.2. KEY REQUIREMENTS

FUNCTIONAL REQUIREMENTS:

1. User Authentication and Security:

- 1.1. Users must be able to register, log in, and log out securely. Users must be able to register within 5 minutes, log in, and log out within 5 seconds.
- 1.2. Passwords must be hashed in the database. Passwords must be hashed using industry-standard hashing algorithms (bcrypt).
- 1.3. SteamID must be encrypted in the database. SteamID must be encrypted using industry-standard encryption algorithms (crypto).
2. Game Interaction:
 - 2.1. Users can rate games they own, express interest in games they do not own and choose their preferred genres. Users can update their ratings and preferences for games.
 - 2.2. Users must be able to rate games they own within 3 clicks.
 - 2.3. Users must be able to express interest in games they do not own within 3 clicks.
 - 2.4. Users must be able to update their ratings and preferences for games within 3 clicks.
 - 2.5. Users must be able to select preferred game genres within 3 clicks.
3. Game Discovery:
 - 3.1. Users can view:
 - 3.1.1. Games they own.
 - 3.1.2. Games their friends are playing.
 - 3.1.3. Top-rated/featured games on the platform.
 - 3.1.4. Browse through all games on the platform.
 - 3.2. Games must be loaded and displayed within 3 seconds.
 - 3.3. Clicking on a game card must reveal a short description, tags, and genres within 1 second.
4. Server and Voice Channel Selection:
 - 4.1. Users can select a Discord server and Discord voice channels associated with it. (The application bot must be invited to at least one Discord server the user is present in)
 - 4.2. Users must be able to select a server and voice channels associated with it within 5 seconds.
 - 4.3. The bot must be invited to at least one server the user is present in within 2 minutes.
5. Member Selection:
 - 5.1. Users can view (only those members that are present in the selected server on Discord and have an account on our application) :
 - 5.1.1. Active members in the selected voice channel.
 - 5.1.2. Members online on Discord.
 - 5.1.3. Members offline.

- 5.2. Users must be able to select a final list of members for recommendations within 2 minutes.
- 6. Recommendation Algorithm:
 - 6.1. The system fetches details of selected members and their owned games and creates a member pool.
 - 6.2. The system creates a game pool which includes the union of multiplayer games owned by the selected users with ratings or expressed interest.
 - 6.3. Recommendation algorithm should consider:
 - 6.3.1. Game ownership.
 - 6.3.2. Ratings and interest.
 - 6.3.3. Preferred genres.
- 7. Recommendation Output:
 - 7.1. The system recommends 5 games. Recommendations are accompanied by reasoning. Recommendations are posted to the selected voice channel's text channel via the application's bot.
 - 7.2. The system must recommend 5 games with reasoning within 10 seconds.
 - 7.3. Recommendations must be posted to the selected voice channel's text channel within 5 seconds.

TECHNICAL REQUIREMENTS :

- 8. Authentication and Security:
 - 8.1. Implemented secure user authentication mechanisms with at least 99.9% uptime.
 - 8.2. Utilized encryption techniques for storing passwords in the database with no known vulnerabilities.
- 9. Database Management:
 - 9.1. Implemented a database schema to store user details, game information, and ratings with a response time of under 100 milliseconds for most queries.
 - 9.2. Ensured data backup and recovery processes can restore data within 1 hour in case of failures.
- 10. Integration with Discord API:
 - 10.1. Implemented integration with Discord API for user and server interactions with an average response time of under 200 milliseconds.
 - 10.2. Ensured the system can handle Discord API rate limits without service interruptions.
- 11. Integration with Steam API:
 - 11.1. Implemented integration with Steam API for retrieving list of games and related information with an average response time of under 400 milliseconds.

- 11.2. Ensured the system can handle Steam API rate limits without service interruptions.
- 12. Recommendation Algorithm:
 - 12.1. Developed an algorithm that considers game ownership, ratings, interest, and preferred genres with an average processing time of under 5 seconds.
 - 12.2. Ensured the algorithm efficiently handles large datasets to provide timely recommendations with at least 15 selected members.
- 13. Hosting:
 - 13.1. Hosted the application on a reliable server with at least 99.99% uptime.
 - 13.2. Implemented monitoring mechanisms to detect and address potential issues within 5 minutes of occurrence.

Non-Functional Requirements:

- 14. Performance:
 - 14.1. The system must provide quick responses to user actions with 95% of requests processed within 2 seconds.
 - 14.2. Response times for recommendation generation and database queries must be within acceptable limits, with 99% of queries completing within 1 second.
- 15. Usability:
 - 15.1. The user interface must be intuitive and user-friendly, with minimum cognitive load for most features and 99% of the users should be able to navigate without any help.
 - 15.2. Users should find it easy to navigate and interact with the application with an average task completion time of under 5 minutes for common tasks.
- 16. Security:
 - 16.1. Implemented measures to prevent unauthorized access to user data.
- 17. Reliability:
 - 17.1. The system must be reliable and available whenever users require it, with scheduled downtime of no more than 2 hours per month for maintenance.
 - 17.2. Implemented backup and cache mechanisms to ensure data integrity and availability in case of API failures, with data restored within 1 hour in case of interruptions.

1.3. DELIVERABLES

1. User Authentication and Security Module: (Requirement 1., 8., 16.)

1.1. Work Scope:

The project involved designing and implementing user registration, login, and logout interfaces, alongside developing backend processes for user authentication and session management. Integration of bcrypt hashing ensures secure password storage, while implementation of secure communication protocols safeguards data transmission.

1.2. Deliverable Completion Criteria:

1.2.1. Functional registration, login, and logout interfaces.

1.2.2. Secure backend processes for user authentication and session management.

1.2.3. Passwords are Hashed using bcrypt.

1.2.4. SteamID must be encrypted.

1.2.5. Secure communication protocols are implemented.

2. Game Interaction Module: (Requirement 2.)

2.1. Work Scope:

The project involved designing and implementing user interfaces for rating games, expressing interest, and selecting preferred genres. Backend processes were developed to manage user interactions with games, while integration of frontend and backend components ensured a seamless user experience. Usability testing was conducted to ensure intuitive interaction flows, and error handling mechanisms were implemented for user input validation.

2.2. Deliverable Completion Criteria:

2.2.1. Functional user interfaces for rating games, expressing interest, and selecting genres.

2.2.2. Backend processes handle user interactions accurately.

2.2.3. Seamless integration of frontend and backend components.

2.2.4. Usability testing reports confirming intuitive interaction flows.

2.2.5. Error handling mechanisms in place for input validation.

3. Game Discovery Module: (Requirement 3., 11.)

3.1. Work Scope:

We have designed and implemented user interfaces to display owned games, friend's games, top-rated/featured games, and allowed browsing through all games. Backend processes were developed to efficiently retrieve and display game data. Frontend components were optimized for fast loading and responsiveness. Dev performance testing ensured that games loaded within

specified timeframes. Additionally, click event handlers were implemented to display game details upon card selection.

3.2. Deliverable Completion Criteria:

- 3.2.1. User interface elements for editing preferences are implemented and functional.
- 3.2.2. Backend processes handle preference updates accurately and securely.
- 3.2.3. Validation mechanisms prevent invalid data input and ensure data consistency.
- 3.2.4. Editing preferences feature is seamlessly integrated into the application.
- 3.2.5. Testing confirms the functionality works as expected without errors or issues.

4. Server and Voice Channel Selection Module: (Requirement 4., 10.)

4.1. Work Scope:

The project entailed designing and implementing user interfaces enabling users to select Discord servers and voice channels. Backend processes were developed to retrieve and manage server and voice channel data efficiently. Integration with the Discord API will facilitate seamless functionality for server and channel selection. Thorough testing was conducted to ensure smooth integration with the Discord.

4.2. Deliverable Completion Criteria:

- 4.2.1. Functional interfaces displaying various game discovery options.
- 4.2.2. Backend processes retrieve and display game data accurately.
- 4.2.3. Frontend components optimized for quick loading and responsiveness.
- 4.2.4. Performance testing reports confirming games load within specified timeframes.
- 4.2.5. Click event handlers successfully display game details.

5. Editing Preferences Module: (Requirement 2., 9.)

5.1. Work Scope:

The editing preferences feature involves designing intuitive user interface elements for adjusting preferred genres, game ratings, and interest. Backend functionality is implemented to handle user requests for preference updates, with validation mechanisms ensuring data integrity and consistency. Integration of frontend and backend components ensures a seamless editing experience. Additionally, guidance has been provided on inviting the application bot to Discord servers to ensure smooth operation.

5.2. Deliverable Completion Criteria:

- 5.2.1. Functional interfaces for selecting Discord servers and voice channels.
 - 5.2.2. Backend processes handle server and voice channel data accurately.
 - 5.2.3. Seamless integration with Discord API for server and channel selection.
 - 5.2.4. Testing confirms successful integration with Discord API.
6. Member Selection Module: (Requirement 5., 10.)
- 6.1. Work Scope:

The project involves designing and implementing user interfaces to view active members in voice channels, online members on Discord, and offline members. Backend processes were developed to retrieve member information from Discord, while frontend components were integrated with backend processes to display member lists. Thorough testing was conducted to ensure accurate retrieval and display of member information. Additionally, user interaction was implemented to enable the selection of members for recommendations.
 - 6.2. Deliverable Completion Criteria:
 - 6.2.1. Functional interfaces displaying active, online, and offline members.
 - 6.2.2. Backend processes retrieve member information accurately from Discord.
 - 6.2.3. Frontend components integrate seamlessly with backend processes.
 - 6.2.4. Testing confirms accurate retrieval and display of member information.
 - 6.2.5. User interactions for selecting members for recommendations are implemented.
7. Recommendation Algorithm Implementation: (Requirement 6., 12.)
- 7.1. Work Scope:

The recommendation algorithm was designed and developed to factor in game ownership, ratings, interest, and preferred genres and total and recent playtime. Backend processes were implemented to retrieve details of selected members and their owned games. A game pool is created, comprising multiplayer games owned by selected users with ratings or expressed interest. Integration of the algorithm with the recommendation output module was carried out. Testing was conducted to ensure the algorithm generates relevant recommendations within specified timeframes. Comprehensive documentation was provided detailing the algorithm's design and implementation details.
 - 7.2. Deliverable Completion Criteria:
 - 7.2.1. Recommendation algorithm developed considering specified factors.
 - 7.2.2. Backend processes fetch member and game details accurately.
 - 7.2.3. Game pool creation based on selected users' multiplayer games with ratings or interest.
 - 7.2.4. Integration of the algorithm with the recommendation output module.

- 7.2.5. Testing confirms relevant recommendations provided within specified timeframes.
 - 7.2.6. Documentation detailing algorithm design and implementation.
8. Recommendation Output Module: (Requirement 7., 9., 12.)
- 8.1. Work Scope:
The project involves designing and implementing user interfaces to showcase recommended games alongside reasoning. Backend processes were developed to generate recommendations using the recommendation algorithm. Additionally, Discord API integration was implemented to post recommendations to selected voice channels' text channels. Testing was conducted to ensure recommendations are generated and posted within specified timeframes.
 - 8.2. Deliverable Completion Criteria:
 - 8.2.1. Functional interfaces displaying recommended games with reasoning.
 - 8.2.2. Backend processes generate recommendations based on the algorithm.
 - 8.2.3. Recommendations are posted to selected voice channels' text channels via Discord API.
 - 8.2.4. Testing confirms recommendations are generated and posted within specified timeframes.
9. Database Management and Integration: (Requirement 9.)
- 9.1. Work Scope:
The project entails designing and implementing a comprehensive database schema to efficiently store user details, owned games information, and ratings. Backend processes were developed to manage data storage, retrieval, and optimization of database queries for enhanced performance. Additionally, robust data backup and recovery processes were implemented to ensure data integrity and availability. Rigorous testing was conducted to validate database functionality and performance.
 - 9.2. Deliverable Completion Criteria:
 - 9.2.1. Database schema designed and implemented for storing user and game data.
 - 9.2.2. Backend processes handle data storage, retrieval, and management accurately.
 - 9.2.3. Database queries optimized for efficient data retrieval and performance.
 - 9.2.4. Data backup and recovery processes ensure data integrity and availability.

9.2.5. Testing confirms database functionality and performance meet specified criteria.

10. System Hosting and Monitoring: (Requirement 13.)

10.1. Work Scope:

The project involves selecting and configuring a dependable server to host the application, deploying all necessary components to the hosting environment. Monitoring mechanisms would have to be implemented to promptly detect and address potential issues, with alerting systems configured to notify administrators of critical events. Comprehensive testing will ensure the application maintains optimal availability and responsiveness.

10.2. Deliverable Completion Criteria:

10.2.1. Application deployed and running on a reliable hosting server.

10.2.2. Monitoring mechanisms in place to detect and address potential issues.

10.2.3. Alerting systems configured to notify administrators of critical events.

10.2.4. Testing confirms application availability and responsiveness.

1.4. ACRONYMS AND ABBREVIATIONS

ACRONYMS	Definition
API	Application Programming Interface : it is the service exchange between two applications. API is used to serve as a request when the client sends a request.
GNR	Game Night Recommender: This is the name of this project.
MVC	Model-View-Controller: A system design pattern used to develop interfaces, business logics and manage data.
HTML	HyperText Markup Language: it is a markup language to design documents in the web browser.
CSS	Cascading Style Sheets: used to style HTML documents.
HTTP	Hypertext Transfer Protocol: is used to retrieve information from the network to the user device using hypertext links.

IDE	Integrated Development Environment: is an application using which developers can write the code and develop the application efficiently.
CORS	Cross-Origin Resource Sharing: used for communication between client and server of different domains.
CRUD	Create, Read, Update, and Delete operations for a database.
DOM	Document Object Model: is an interface which treats HTML as a tree structure where each node is a part of the webpage.
JWT	JSON Web Token: is a standard for creating string patterns with signatures and encryption which can be converted to some valid information on decryption.
DB	Database: is a collection of data stored in a device or the cloud.
PR	Pull request: is a request to merge changes in the code from one branch to another.

2. USER MANUAL

This User Manual provides a comprehensive guide to the features implemented in the Game Night Recommender (GNR) tool. It is intended for software engineers and developers who will be maintaining/extending the GNR tool. Each feature is described with its functionality and usage instructions.

2.1. IMPLEMENTED FEATURES

No.	REQUIREMENT	DESCRIPTION & USAGE DETAILS
1.	User Authentication and Security	Secure registration, login, and logout processes. Hashed passwords and user steam IDs.
		<ol style="list-style-type: none"> 1. Users can register an account by providing a username, email, and password. 2. Additionally, the users' Steam ID, Discord username, preferred video game genres and ratings are requested for minimum recommendation context. 3. Users can log in and out with their email and password credentials. The sign-up process is designed to be completed within the specified time constraints (registration within 5 minutes, login/logout within 5 seconds). 4. Additional security constraints exist, such as Password and Steam ID encryption using bcrypt and crypto (AES algo) libraries, respectively.
2.	Game Discovery	Users can view owned games and top multiplayer games, as well as browse all games on the Steam platform.
		<ol style="list-style-type: none"> 1. The dashboard allows users to browse and search all Steam games. 2. Users can filter games by genres, tags and features, and sort games by release date, price and review score. 3. Users can also sort owned games by total and recent (last 2 weeks) playtime. 4. Clicking on a game card reveals a short description, tags, and genres within 1 second. 5. The game information is requested with params from external services: Steam Web APIs and Gamalytic APIs.
3.	Game Interaction	Users can rate owned games, express interest in unowned games, and select preferred genres.
		<ol style="list-style-type: none"> 1. Within the user dashboard, users can rate games they own, express interest in games they do not own, all within 3 clicks. 2. The rating for owned games is saved as a number in the 0-5 scale and the interest expressed in an unowned game (love, like, dislike) is saved as dislike (1, 0.75, 0).

	3. Ratings and genre preferences can be tracked and updated at any time in the 'Edit Preferences' tab.	
4.	<i>Discord Server and Channel Selection</i>	Users can select a Discord server and an associated voice channel.
	<ol style="list-style-type: none"> 1. At Sign up, users can invite the application bot to a Discord server and upon trying to generate recommendations, users can select a server and voice channels within 5 seconds. 2. The bot must be invited to at least one server the user is present. 	
5.	<i>Discord Member Selection</i>	Users can view and select members for game recommendations.
	<ol style="list-style-type: none"> 1. Users can view active members in the selected voice channel, members online on Discord, and members offline. 2. A game pool is generated based on the final list of members for recommendations. 	
6.	<i>Multicriteria Scoring Recommendation Algorithm</i>	The system generates multiplayer game recommendations based on varying priorities for group's collective ownership, genre preferences, user interest and ratings, as well as play time.
	<ol style="list-style-type: none"> 1. The recommendation algorithm considers game ownership, ratings, interest, preferred genres and playtime to recommend 6 games with reasoning within 10 seconds. 2. The algorithm is detailed in these documents: Multicriteria scoring - Google Sheets Algorithm idea-2 - Google Docs 3. A 'API Key limit reached' error should be displayed to the user if the external call to fetch user information starts to fail. 	
7.	<i>Recommendation Output</i>	The system recommends games and posts them to the selected voice channel's text channel via the application's bot.
	<ol style="list-style-type: none"> 1. The system displays 6 game recommendations with reasoning on the UI which can be shared to the Discord text channel within 5 seconds of the click of a button. 	

3. INSTALLATION GUIDE

3.1. OVERVIEW

The below steps provide step-by-step instructions for technical users to set up and run the application. The application is built using MERN (MongoDB, Express.js, React.js, Node.js) stack, which contains MongoDB for database, Express and Node for the backend server and React for the frontend user interface.

3.2. PRE-REQUISITES

3.2.1. OPERATING SYSTEM

This installation guide assumes you are using either a Linux or Windows operating system to run the application.

3.2.2. TERMINAL

You should be familiar with using the terminal or the command line interface (CLI) for executing the commands required to run the application.

3.2.3. BROWSER

There should be a browser installed on the computer if you are using Windows then Microsoft Edge will be preinstalled and for macOS, safari browser will be preinstalled.

3.3. INSTALLATION STEPS

1. INSTALL NODE.JS AND NPM:
 - 1.1. Download the Node.js and npm from the official website ([Link](#)).
 - 1.2. Follow the installation instructions for the operating system being used.
2. INSTALL GIT
 - 2.1. FOR UBUNTU/DEBIAN:
 - 2.1.1. On the command line run `sudo apt install git`
 - 2.2. FOR MacOS:
 - 2.2.1. On the command line run `brew install git`
 - 2.2.2. Note: If homebrew is not installed then download homebrew from the website ([Link](#)).
 - 2.3. FOR WINDOWS:
 - 2.3.1. Download git from the official git website ([Link](#)).
 - 2.3.2. Run the installer and follow the instructions to complete the installation.

- 2.4. TO VERIFY THE GIT INSTALLATION
 - 2.4.1. Run ``git --version``
 - 2.4.2. This should give you the version of the git without any errors.
- 2.5. CONFIGURE GIT
 - 2.5.1. Set up a username and password to access the git repository.
 - 2.5.1.1. Run the following command in the terminal
 - 2.5.1.1.1. `git config --global user.name "Your Name"`
 - 2.5.1.1.2. `git config --global user.email "your_email@example.com"`
3. CLONE THE PROJECT FROM GITHUB:
 - 3.1. Open the terminal.
 - 3.2. Navigate to the directory where you want the project to be cloned.
 - 3.3. Run the command
 - 3.3.1. `git clone <repository_url>`
 - 3.3.2. Replace `<repository_url>` with the URL of the repository.
4. INSTALLING NODE MODULES I.E PROJECT DEPENDENCIES
 - 4.1. Navigate to the project directory.
 - 4.2. Navigate to the backend folder
 - 4.3. Run ``npm install``
 - 4.4. Navigate back to the project directory.
 - 4.5. Navigate to the frontend folder.
 - 4.6. Run ``npm install``
 - 4.7. Navigate back to the project directory.
5. SET UP MONGODB ATLAS (CLOUD DATABASE):
 - 5.1. In the web browser open the MongoDB Atlas website ([Link](#))
 - 5.2. Create a new account or log in to your MongoDB Atlas account.
 - 5.3. Click on the "Build a Cluster" button.
 - 5.4. Select the required Cloud Provider and Region.
 - 5.5. Choose a Cloud Provider and Region
 - 5.5.1. Select any cluster tier of your choice.
 - 5.6. Click on "Create Cluster".
 - 5.7. Once the cluster is created, click on the "Connect" button to get the cluster details.
 - 5.8. Select the "Drivers" under "Connect to your application".
 - 5.9. Select "Node.js" as the driver with the latest version available.
 - 5.10. Click on "Review setup steps"

- 5.11. Copy the “Connection String”, this will be the MongoDB URL which you need to paste under Step 5.3 for the key MONGO_URL.

6. SET UP ENVIRONMENT VARIABLES

- 6.1. Navigate to the backend folder.
- 6.2. Create a file and name it `.env`.
- 6.3. Add the necessary environment variables for the backend values as shown below:
 PORT=8080
 MONGO_URL=<MONGO URL - created in Step 6>
 JWT_SECRET_KEY=<JWT SECRET KEY>
 RAWG_API_KEY=<RAWG API KEY HERE>
 STEAM_API_KEY=<STEAM API KEY HERE>
 BOT_TOKEN=<DISCORD BOT TOKEN>
 AGE_SENSITIVE_TAGS=
 STEAM_SECRET_KEY=<STEAM SECRET KEY>
 STEAM_SECRET_IV=<STEAM SECRET IV>
- 6.4. Navigate to the frontend folder.
- 6.5. Create a file and name it `.env`.
- 6.6. Add the necessary environment variables for the frontend with values as shown below: REACT_APP_BASE_URL=<http://localhost:8080/api>

7. START THE APPLICATION

- 7.1. Navigate to the project directory.
- 7.2. Run the command `./start_app.sh`
- 7.3. The application should automatically be opened in the browser.
- 7.4. In case the application does not automatically open then
 - 7.4.1. Open the browser.
 - 7.4.2. Navigate to <http://localhost:3000>.

3.4. DEPLOYMENT STEPS

1. CREATE AWS ACCOUNT

- 1.1. To deploy the MERN stack application create an AWS account in the amazon website ([Link](#)).

2. CREATE AN EC2 INSTANCE AND DEPLOYMENT
 - 2.1. Once the AWS account is created, create an EC2 instance to deploy the application.
 - 2.2. Select the required OS and storage space according to the budget, any OS and storage will work for this application.
 - 2.3. Once the Instance is created, click on connect on the EC2 instance and run the ssh command on the terminal.
 - 2.4. Once the connection is done we need to clone the project on the instance for this follow Steps 2 and Step 3 from the project local installation guide written above.
 - 2.5. Once the project is cloned, navigate to the project folder and create an '.env' file and paste the same contents which was given in Step 6 from the project local installation guide written above.
 - 2.6. Now run the command 'npm install' inside the backend folder.
 - 2.7. Once this is complete, run the backend on the EC2 instance by executing the command 'pm2 start npm -- start' so that the instance is always running even when you close the terminal.
 - 2.8. Now in a new terminal navigate to the frontend folder and change the '.env' value of the REACT_APP_BASE_URL to the EC2 instance URL.
 - 2.9. Run the command './build_prod.sh', to build the project, when the prompt asks for permission to open the instance type 'yes' and git enter.
 - 2.10. The above step will automatically connect to the EC2 instance, once connected execute the command './deploy.sh' to run the front end on the EC2 instance.
 - 2.11. Now access the EC2 instance public URL to access the application.
 - 2.12. For redeployment of the application follow the steps from 2.4 to 2.11.

4. ISSUES

In this section, we highlight critical issues impacting our application's performance and user experience. These include constraints such as rate limits with the Steam API, inconsistent response times from the Gamalytic API, absence of pagination support, lack of a password reset feature, and scalability limitations in our algorithm. Addressing these issues is essential to ensure smoother operation, better user engagement, and long-term competitiveness.

4.1. RATE LIMIT WITH STEAM API KEY

4.1.1. ISSUE DESCRIPTION

The current implementation of our application relies on the Steam Web API to fetch data. However, we are encountering a constraint due to the rate limit imposed by Steam. As per our observation, the limit allows only one hundred thousand (100,000) calls to the Steam Web API per day. This limitation poses a bottleneck, especially during peak usage periods, hindering the seamless operation of our application.

4.1.2. RECOMMENDATION

To mitigate this issue, we need to implement strategies such as caching frequently accessed data locally to reduce the frequency of API calls. Additionally, we should explore alternative solutions, such as optimizing the API calls to minimize unnecessary requests and ensure efficient utilization of the allotted quota. Another solution would be to allow the user to generate and provide their own steam API key to the GNR system for more reliable access.

4.2. GAMALYTIC API RESPONSE TIME

4.2.1. ISSUE DESCRIPTION

Our application integrates with the Gamalytic API to retrieve gaming-related data. However, we have observed instances where the API response time is slow, leading to delays in processing user requests. This inconsistency in response time impacts the overall user experience and may result in frustration among our user base.

4.2.2. RECOMMENDATION

To address this issue, we should conduct a thorough analysis of the factors contributing to the variability in response time. This analysis may involve assessing server load, network latency, and API endpoint optimization. Additionally, we should consider implementing caching mechanisms to store frequently accessed data locally, reducing reliance on real-time API calls and mitigating the impact of fluctuations in response time. Another research-heavy approach would be to explore more open source databases that provide vast information on Steam games.

4.3. MISSING PAGINATION IN GAMALYTIC API

4.3.1. ISSUE DESCRIPTION

Currently, when retrieving game lists from the Gamalytic API through an API call, our application lacks proper pagination support. As a result, only the first fifty games from the retrieved list are displayed to the users. This limitation prevents users from accessing the full catalog of games and hampers their ability to explore the available options comprehensively.

4.3.2. RECOMMENDATION

To address this issue, we need to implement pagination support in our application, allowing users to navigate through large datasets seamlessly. This involves modifying our API integration to handle paginated responses from the Gamalytic API efficiently. By incorporating pagination, we can enhance the usability of our application and provide users with a more intuitive browsing experience.

4.4. CONFIGURATION SAVING FOR PARAMETER WEIGHTS

4.4.1. ISSUE DESCRIPTION

Our application lacks a feature that allows users to save configurations of parameter weights for personalized preferences. Currently, users are unable to preserve their customized settings, forcing them to reconfigure the parameters every time they use the application.

4.4.2. RECOMMENDATION

To improve user convenience and streamline the customization process, we should implement functionality to enable users to save and load configuration profiles. This feature would allow users to store their preferred parameter weights for future use, enhancing the personalization aspect of our application.

4.5. ALGORITHM OPTIMIZATION FOR SCALABILITY

4.5.1. ISSUE DESCRIPTION

Our application's algorithm exhibits performance degradation when operating with a combined game pool exceeding 300 games. Beyond this threshold, the algorithm's efficiency diminishes, resulting in prolonged wait times for users. This scalability limitation hampers the application's ability to accommodate larger user bases and limits its potential for growth.

4.5.2. RECOMMENDATION

To address this scalability issue, we need to undertake algorithm optimization efforts aimed at improving performance and reducing wait times, particularly when handling

large datasets. This optimization may involve revisiting the algorithm's design and implementation to identify inefficiencies and bottlenecks. Additionally, we should leverage techniques such as parallel processing, caching, and algorithmic enhancements to enhance scalability and ensure optimal performance, even under heavy load conditions. Regular performance testing and profiling should be conducted to assess the effectiveness of the optimization efforts and fine-tune the algorithm accordingly.

4.6. PASSWORD RESET FUNCTIONALITY ABSENT

4.6.1. ISSUE DESCRIPTION

Our application currently lacks a password reset feature, which poses a significant inconvenience to users who have forgotten their passwords or wish to update them for security reasons. Without this functionality, users are unable to regain access to their accounts independently, leading to frustration and potential loss of engagement with our platform.

4.6.2. RECOMMENDATION

To address this critical usability issue, we should prioritize the development and implementation of a password reset feature. This feature would enable users to initiate the password recovery process by providing their email address or username associated with their account. Upon submission, users should receive an email containing a secure link or temporary token allowing them to reset their password securely.

4.7. CROSS-PLATFORM BROWSER COMPATIBILITY OVERSIGHT

4.7.1. ISSUE DESCRIPTION

Currently, our application is optimized for Google Chrome and primarily caters to larger screens, neglecting compatibility with other web browsers and smaller devices. This oversight restricts accessibility for users who prefer alternative browsers or access the application from smaller screens, leading to suboptimal user experiences.

4.7.2. RECOMMENDATION

To address this issue, we must prioritize enhancing cross-platform browser compatibility to ensure seamless operation across a variety of web browsers, including but not limited to Google Chrome. This involves conducting comprehensive testing and debugging on popular browsers such as Mozilla Firefox, Safari, and Microsoft Edge to identify and resolve compatibility issues. Additionally, adopting responsive web design principles and implementing CSS media queries can optimize

the application's layout and functionality for smaller screens, enhancing accessibility and user satisfaction across diverse devices and browsing environments.

5. OUTLOOK

This section outlines key enhancements aimed at advancing our application's capabilities and user features. These include expanding platform support, introducing friend management functionality, diversifying recommendation systems, predicting friends' online activity, and enabling users to save custom parameter configurations. These enhancements aim to enrich user experiences, foster social connectivity, and streamline personalization processes, ensuring our application remains competitive and user-centric. The following features listed are ordered by priority.

5.1. MULTI-PLATFORM SUPPORT

5.1.1. ENHANCEMENT DESCRIPTION

Expanding our application's support beyond Steam to include all gaming platforms is crucial for broadening our user base and accommodating the diverse gaming preferences of our audience. By integrating with multiple gaming platforms such as PlayStation, Xbox, Nintendo, and PC gaming platforms like Epic Games Store and Origin, we can provide a more comprehensive gaming experience to our users.

5.1.2. TECHNICAL IMPLEMENTATION

Implementing multi-platform support entails developing robust APIs or SDKs for each gaming platform to facilitate seamless data integration and interoperability with our application. This involves establishing secure authentication mechanisms, handling platform-specific data formats, and ensuring compliance with each platform's terms of service. Additionally, we need to enhance our backend infrastructure to scale efficiently and accommodate the increased data volume and diversity resulting from multi-platform integration.

5.2. FRIEND MANAGEMENT FEATURE

5.2.1. ENHANCEMENT DESCRIPTION

Introducing a friend management feature enables users to connect and interact with their peers within our platform, fostering a sense of community and enhancing social engagement. Users can add friends, view their online status, communicate via messaging, and participate in multiplayer gaming sessions together.

5.2.2. TECHNICAL IMPLEMENTATION

Implementing the friend management feature involves developing user-friendly interfaces for adding, removing, and managing friends within the application. This requires backend support for storing and retrieving user friend lists securely, as well as real-time communication mechanisms to update friends' online statuses dynamically. Integration with third-party authentication providers or social media platforms may be necessary to facilitate friend discovery and connection.

5.3. ENHANCED RECOMMENDATION DIVERSITY

5.3.1. ENHANCEMENT DESCRIPTION

Expanding our recommendation system to provide users with diverse recommendations in spite of having the same parameters enhances the personalization and discovery aspects of our application. By offering a variety of game suggestions tailored to users' preferences, we can cater to a broader range of gaming interests and preferences.

5.3.2. TECHNICAL IMPLEMENTATION

Enhancing recommendation diversity involves refining our recommendation algorithm to consider additional factors such as genre diversity, user play history, and emerging trends in gaming. This requires sophisticated data analysis techniques, machine learning models, and collaborative filtering algorithms to generate diverse and relevant recommendations.

5.4. FRIENDS' ONLINE ACTIVITY PREDICTION

5.4.1. ENHANCEMENT DESCRIPTION

Introducing a feature that predicts when users' friends are likely to come online based on their past activities enhances the social connectivity and coordination within our platform. By providing users with insights into their friends' gaming patterns, we can facilitate better coordination for multiplayer gaming sessions and enhance the overall gaming experience.

5.4.2. TECHNICAL IMPLEMENTATION

Implementing the friends' online activity prediction feature entails analyzing historical user activity data to identify patterns and trends in friends' online behavior. Machine learning algorithms, such as time series analysis and predictive modeling, can be employed to forecast friends' online presence accurately. Real-time synchronization with friends' gaming platforms or APIs is essential to ensure timely updates and notifications regarding friends' online status changes.

5.5. SAVED PARAMETER WEIGHT CONFIGURATIONS


5.5.1. ENHANCEMENT DESCRIPTION

Enabling users to save and retrieve custom parameter weight configurations streamlines the personalization process and enhances user convenience. By allowing users to store their preferred parameter settings, we empower them to tailor the recommendation system to their unique preferences effortlessly.

5.5.2. TECHNICAL IMPLEMENTATION:

Implementing the saved parameter weight configurations feature involves developing user interfaces for managing and storing custom weight configurations securely. Backend support is required to store user-specific configurations persistently and ensure seamless retrieval during subsequent sessions.

6. VIDEO PRESENTATION

 Game Night Recommendation System | SER517 | Group-F-22 | Application Demo