Development of an Open-Source Application for Video Game Management

Alfie Wheeler
26380270

26380270@lincoln.ac.uk

School of Engineering and Physical Sciences
College of Health and Science
University of Lincoln

Submitted in Partial Fulfilment of the Requirements for the
Degree of BSc (Hons) Computer Science

Supervisor: Dr. Athanasios Polydoros

March 2025

Acknowledgements

Firstly, I want to express my deepest gratitude to Dr Polydoros for his continued support throughout this project. His expertise and assistance have been instrumental in the completion of this work, and it has been a pleasure working with him.

I am deeply grateful to my wonderful girlfriend Zoe, you are my rock, and I thank you for your enduring patience with all the proof-reading I asked you to do.

I would not be completing this project if it were not for my brilliant friends Daniel, Ed and Oscar. Your friendship means the world, and you have made the last three years so enjoyable.

Lastly, I want to thank both my parents for always believing in me, providing financial support and a hundred other things I do not have the space to list here. This would not have been possible without them.

Abstract

Millions of people now use media tracking applications to organise and monitor their consumption of entertainment media. Although most entertainment sectors have one or two mainstream solutions, the video game market lacks a perfect solution despite it outperforming streaming and box office revenue in 2023. As such, this project focused on the development of an open-source video game tracking application that allows users to manage and review their gaming history. The decision to build this platform as open-source reflects a commitment to transparency, user autonomy, and customisability, values often limited in today's dominant subscription-based models. The goal of the application was to provide an efficient method for users to track progress in the games they play, provide ratings and receive recommendations for new titles. The system was built using a Flutter frontend, a Flask backend, and a SQL database, integrating the IGDB API to retrieve comprehensive game data. Key features include a Levenshtein search engine, a hybrid recommendation engine and an advanced filtering solution. The application aimed to adhere to Nielsen's design principles to provide an intuitive user interface (UI) and a responsive system. Improvements were made through multiple iterations of development, including extensive functional testing and user feedback, through a survey and interviews. By combining functionality with user-centred design, the final artefact demonstrates a practical and user-friendly approach to game management. It meets the vast majority of aims, objectives, and requirements that were set out, providing a strong foundation for a product that could compete with existing applications in the market.

This research was screened under the University of Lincoln (LEAS) online ethics framework and no issues were identified (Project ID: 13320).

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1: Introduction

Media tracking applications have become a common tool for users to organise and monitor their entertainment consumption, including anime, books, and films. The gaming space, despite its massive global popularity and continued market growth, still lacks an accessible, comprehensive solution. MyAnimeList, a popular platform for tracking anime viewing, boasts around 17.5 million users at the time of writing (MyAnimeList, undated), highlighting the demand for such tools.

Although similar applications exist in the gaming space, such as Stash (Stash, undated) and GAMEYE (GAMEYE, undated), both present notable limitations. Stash locks features behind a paywall, restricting accessibility. GAMEYE is more complete but has not reached the same level of widespread adoption. Its search functionality also presents challenges, with each individual release of a game being treated as a separate entry. This level of granularity may be appealing to power users but makes adoption more difficult for casual users and fragments discussion across multiple listings.

This highlights a clear opportunity for a solution that combines comprehensive functionality with accessibility for everyday gamers. This project aimed to create an open-source application that provides that. This report details the development process, including a literature review, the design approach, the structure of the artefact, user and functional testing before showcasing the final product and outlining potential future enhancements.

The application is built using a combination of a MySQL database (MySQL, undateda), Flask server (Flask, undated), and Flutter frontend (Flutter, undateda), with game details provided by the IGDB API (IGDB, undatedb). Good user interface (UI) design practices are discussed in detail and key features such as the application’s functionality, search engine and recommendation system are highlighted.

Chapter 2: Literature Review

2.1 Background

The video game industry continues to grow steadily year after year. In 2023, the market reported $196 billion dollars in revenue, surpassing both streaming and the box office combined (Christofferson et al., 2024). With Bain forecasting 6% annual growth through to 2028, the industry will almost certainly continue to grow (Christofferson et al., 2024). In addition to this, 61% of the U.S. population report playing at least an hour of video games a week (Entertainment Software Association, 2024), showcasing the medium's widespread cultural significance.

Similarly, the mobile phone industry is experiencing rapid growth**,** with projections indicating an increase from $484.81 billion to $792.51 billion between 2022 and 2029 (Fortune Business Insights, 2025). As of December 2024, mobile phones dominated the market worldwide with a 63% share compared to desktops and tablets (Statcounter, 2024), solidifying their position as the future for most smart device users. Given the growth of both industries, it is logical to assume that mobile applications that enhance the gaming experience are becoming increasingly relevant to the end user.

2.2 The Importance of Open-Source Software

In modern society, subscription-based models have become the norm. Their convenience and safe consumption method have led to major growth in the industry since the Covid-19 pandemic (Jo et al., 2024). However, while this model prioritises convenience and accessibility, it often comes at the cost of user autonomy and transparency. In contrast to these limitations, open-source software continues to attract interest for its fundamentally different approach, centred on accessibility and user autonomy.

Open-source software has long been recognised for its economical and societal contributions, particularly its ability to allow users to modify and understand applications and software that they might not otherwise access Von Krogh and Spaeth, 2007). Alongside its direct benefits, major organisations often rely on open-source software to show that a product is viable for high-quality implementations. Companies such as Microsoft and Google openly embrace the open-source community, with projects like Kubernetes (Microsoft, undated) and Chromium (Google, undated).

Open-source software also allows for more customisation for its users. The prevalence of mass market produce has left some consumers desiring more customisation in the products they use (Randell et al., 2024). People who are strongly inclined towards individualisation hold more value in products that are tailored to them (Randell et al., 2024). Open-source software can provide this where other alternatives cannot, increasing their appeal to users seeking tailored solutions.

Finally, open-source promotes improved security and transparency. Community-maintained projects tend to receive more frequent updates and security patches (Lazarus Alliance, 2024). This collaborative approach also allows the software to be cost-effective to develop and maintain (Lazarus Alliance, 2024).

2.3 API Usage

APIs (Application Programming Interfaces) serve as the foundation for many popular applications. Services such as Uber and Spotify integrate APIs to provide both internal and external functionality, primarily around retrieving and managing large amounts of data (Thangavelu et al., 2020; Spotify, undated). Considering that this project will similarly involve handling extensive datasets, such as game

metadata and user game libraries, the successful implementation of APIs in large-scale applications validates their utilisation on the backend for this project.

Two of the most popular architectures for developing APIs are REST (Representational State Transfer) and GraphQL. REST is an architectural style of network application that relies on HTTP and set protocols to allow for stateless communication (Gupta, 2025). In contrast, GraphQL, created by Facebook in 2012, is a query language that allows clients to request exactly the data they need (The GraphQL Foundation, undated). While REST is more mature and easier to learn, GraphQL requires a substantial time investment to become proficient (Vadlamani et al., 2021). However, that time investment can be worth it, given that GraphQL helps reduce over-fetching, a predominant issue when implementing RESTful APIs (Vadlamani et al., 2021). Despite this, REST will be adopted for this project. Its simplicity will decrease development time, and its performance is well-suited to this application's mid-sized database, with room to scale if needed.

2.4 User Interface Design

UI design is a crucial factor in the success of any application. No matter the quality of information provided, if the end user cannot effectively interact with the application, then the service is a failure (Ruiz et al., 2020). As such, it is important to consider the principles of good user experience throughout the development process

Market research is the first step: understanding user motivations, identifying effective features, and recognising gaps in similar applications (Steane, 2023). This allows for project context and informs decision-making when designing (Steane, 2023).

Whilst numerous design principles exist, the most important to follow are Nielson's 10 Usability Heuristics (Nielsen, 2024). Whilst more rule of thumb than specific guide, by adhering to them, these heuristics ensure that an interface is understandable, easy to adopt, efficient and accessible. Given that this project focuses on developing a mobile application, adhering to mobile UI best practices is equally important. A crucial aspect of this is focusing on designing around the 'thumb zone', the area where it is most comfortable for a user to interact with a mobile device (Usiskin, 2025). Primary actions such as navigation should be placed near the bottom of the screen for easier reach, and when this is not possible alternative gestures should be available, such as swiping to navigate back (Usiskin, 2025). The entire application should also be tested frequently, both on physical and simulated devices to measure both usability and performance (Aldayel and Alnafjan, 2017).

Accessibility is also a key consideration when creating a mobile application. A poorly designed UI can create issues outside of the ones inherently presented by mobile devices, like small screens and text size (Zaina et al., 2022). Therefore, it is essential to tackle accessibility barriers in mobile design such as poor icon design, colour contrast issues and unreadable text. These issues can be approached through sensible icons with appropriate labelling, understandable colour schemes and appropriately sized text (Zaina et al., 2022).

2.5 Recommender Engines

The aim of a recommender engine is to provide substantial suggestions to a user based on the content they have previously consumed. The two main approaches are collaborative and content-based, with hybrid methods combining both (Melville and Sindhwani, 2010). More recently, temporal recommendation systems have also become more prevalent, due to their ability to account for context by considering factors such as time and location (Wu et al., 2022).

This project plans to leverage a recommender system to make suggestions to the user. A content-based recommender engine would be appropriate for this task due to its independence and effectiveness with sparse data (Javed et al., 2021). A strong model choice for this would be K-Nearest Neighbour (KNN) as it is effective and efficient, whilst remaining scalable and easy to interpret (Taunk et al., 2019). However, content-based models can tend to overspecialise and not provide a holistic overview of the data, potentially leading to weaker recommendations (Javed et al., 2021).

Comparatively, in a study conducted on movie recommendations, it was found that both user and item-based collaborative filtering approaches provide accurate results, with item-based filtering performing more efficiently in dynamic environments (Sahu and Saritha, 2021), such as the application being developed in this project. Alongside this more traditional recommendation system, the application will also recommend popular releases. For this, a temporal model employing time-dependent matrix factorisation is appropriate, due to its ability to track user preferences over sliding time windows, using weightings to forget outdated data (Rabiu et al., 2020).

A common issue is the cold start problem, where a lack of historical data makes it difficult to provide recommendations to new users (Yuan and Hernandez, 2023). One way this project could tackle this is that, following the initial recommendation, results could be weighted using a trust system based on the users' ratings of their friend's taste. This approach has been proven to provide better recommendations (Ozsoy and Polat, 2013) and addresses the issue by using trust propagation and indirect trust to avoid newer users having insufficient information to make reliable recommendations (Mohammadi et al., 2019).

An alternative approach to this may be adopting a hybrid model, such as the Rating Bayesian Personalised Ranking (RBPR) model proposed by Feng et al. (2021). This model combines explicit ratings via Probabilistic Matrix Factorisation (PMF) and implicit ratings through Bayesian Personalised Ranking (BPR) to improve accuracy

for cold start situations. This approach aligns well with the project's dataset as explicit metrics, such as user ratings, can be enhanced with implicit signals, such as genre frequency in a user's collection. Whilst this model has demonstrated improved accuracy, it also introduces significant computational overhead and complexity for the implementation, especially with large datasets (Feng et al., 2021).

2.6 Aims and Objectives

Aim 1: Create an application that allows users to track the games they play.

Objectives for Aim 1:

- Allow users to view and filter their 'Quest Log' by length, release date, or title.
- Display comprehensive information about each game, including a description, genre and age rating.
- Implement a search feature that allows users to find and add games to their list by title.
- Enable users to add, update or remove games to their log and track details such as completion status and rating.
- Provide personalised recommendations of new games to play based on user preference.

Aim 2: Create a complete and well-structured database of games that facilitates the user's experience.

Objectives for Aim 2:

- Populate the database with comprehensive game details from a suitable API.
- Ensure the database is atomic, normalised, and follows best practices.
- Implement an efficient and scalable search engine.
- Incorporate a range of stored functions to manage database operations, including adding or updating games in a user's log, retrieving game details, and handling user information.
- Provide a method to communicate between the frontend and backend that is responsive and ensures efficient data retrieval and updates.

Aim 3: Ensure an intuitive and accessible user interface.

Objectives for Aim 3:

- Abide by Nielson's 10 Usability Heuristics to ensure a user-friendly experience.
- Ensure a consistent and intuitive layout, with clear navigation between screens.
- Guarantee accessibility by maintaining appropriate colour contrast, readable text, and both light and dark modes.
- Refine the UI using feedback from surveys and interviews.
- Implement responsive design to ensure compatibility with different devices.

Chapter 3: Requirements Analysis

The requirements to successfully complete this project this are broken down by the keys in Figure 1 and the requirement table (Table 1). Each requirement has been categorised according to its nature and assigned a priority and complexity. This approach ensures that core features are delivered in Phase 1 to form the minimum viable product (MVP). Phase 2 includes features that the project aims to include, and Phase 3 outlines desirable functionality that may not be included in the final artefact.

Functional requirements form most of the project's scope, detailing key features the application must include. These are complimented by non-functional, operational and technical requirements focused on areas such as performance, usability, device support and the software stack.

Key	Category
O	Operational
F	Functional
NF	Non-Functional
T	Technical

Key	Priority
MH	Must Have
SH	Should Have
CH	Could Have
NTH	Nice To Have

Key	Complexity
S	Simple
M	Medium
H	High

Figure 1: Keys for the requirement table (Table 1).

REQUIREMENT LIST				REQUIREMENTS ANALYSIS			
ID	REQUIREMENT DESCRIPTION	CATEGORY	PRIORITY	COMPLEXITY	In Scope?	TEST OR VERIFICATION	PHASE
A	Create an application that allows users to track the games they play	O	MH	H	✓	Functional Unit Testing System Integration Testing User Acceptance Testing	1
A.1	Users can have individual accounts and log into the application, with hisotry entries tied to that account	F	MH	M	✓	Functional Unit Testing System Integration Testing	1
A.2	The system supports adding, updating and deleting games from a user's history	F	MH	M	✓	Functional Unit Testing System Integration Testing User Acceptance Testing	1
A.2.1	Users should be able to modify the rating, status and dates played for each entry.	F	MH	M	✓	Functional Unit Testing System Integration Testing User Acceptance Testing	1
A.3	Provide search functionality to find and add games to a user's history	F	MH	H	✓	Functional Unit Testing System Integration Testing User Acceptance Testing	2
A.4	Need to be able to view, sort and filter games in a users history based on details such as name and release date.	F	MH	M	✓	Functional Unit Testing System Integration Testing User Acceptance Testing	1
A.5	The application can provide detailed information about every game in the database.	T	MH	S	✓	Functional Unit Testing System Integration Testing User Acceptance Testing	1
A.5.1	The application can provide recommendations for new games based on the user's history.	T	SH	H	✓	Functional Unit Testing System Integration Testing User Acceptance Testing Generate reccomendations within 5 seconds	2
A.6	Be able to take notes for games in a users log	F	CH	M	✓	Functional Unit Testing System Integration Testing User Acceptance Testing	2
A.7	Users can add friends and rate their game tate to get better reccomendations	F	CH	H	X	Functional Unit Testing System Integration Testing User Acceptance Testing	3
A.8	Users can write long form reviews for games in their log	F	CH	M	X	Functional Unit Testing System Integration Testing User Acceptance Testing	3
B	Create a complete and well-structured database of games that facilitates the user's experience.	O	MH	M	✓	Bench Marking against Simalar Applications: Testing of Response Times of calls.	1
B.1	The database should be atomic, normalized and capable of scaling with additional users or games.	T	MH	M	✓		1
B.2	The interface must be fast and responsive.	NF	MH	M	✓	Testing of Page loads and refresh times	2
B.2.1	Use Stored procedures to reduce compliation time and reduce network traffic	F	SH	M	✓	Functional Unit Testing	1
C	The application should follow Microservices Aritecture, allowing for the loose coupling	T	MH	S	✓	Review of Final code Update separate parts of the artefact and test	1
C.1	Communication is handled between the frontend and backend using a RESTful API.	T	MH	S	✓	Functional Unit Testing	1
D	The application should be intuitive and have an accessible user interface	O	MH	H	✓	User Acceptance Testing User Questionaires	1
D.1	The application should conform to Nielson's usability heuristics	NF	MH	M	✓	Compare to Heuristics	1
D.2	The colour scheme must ensure readability.	NF	MH	S	✓	User Acceptance Testing User Questionaires	1
D.2.1	The application should support light and dark themes	F	SH	M	✓	Functional Unit Testing User Acceptance Testing	2
E	User accounts and data are securely stored and retrievable.	F	MH	S	✓	Passwords are hashed in database User details are retrivable	1
F	The Application should run on multiple platforms	O	NTH	M	✓	Repeat testing on different platforms	1
F.1	The appliciation supports Android devices	T	MH	S	✓	Runs on Anrdroid devices	1
F.2	The appliciation supports IOS devices	T	CH	M	X	Runs on IOS devices	3

Table 1: Requirements analysis for this project.

Chapter 4: Design & Methodology

4.1 Project Management

This section describes the approach taken to effectively manage this project. It includes the initial timeline, the development methodology, a detailed breakdown of the toolsets and environments, and a risk assessment.

4.1.1 Planning and Timeline

The Gantt chart in Figure 2 was created at the start of this project to outline its different stages. It was never intended to be followed completely but instead provide an idea of when tasks should be undertaken to ensure that the artefact was deliverable on time.

Figure 2: Project Gantt chart.

4.1.2 Methodology and Development Approach

Although the Gantt chart outlined the overall timeline, a structured development methodology was still needed to guide daily progress. Both Agile and Iterative approaches were considered for this project, however Agile was settled on due to its flexibility, risk management and improved quality (Harvard Business Review Analytic Services, 2015). Whilst Iterative development has its strengths, the risk of scope creep and complex project management (Brown, 2025) did not make it a good fit for this project due to the multiple interconnected components and time constraints.

Since Agile is traditionally team-oriented, some changes were needed to make it appropriate for solo development. The core concepts of scrum were taken and adapted to work without a team. Tasks were broken down into sensible chunks, normally with each task corresponding to a feature. These tasks were then completed in sprint cycles to provide incremental delivery. After each sprint, time was set aside to reflect on progress and assess whether development was still aligned with overarching project goals.

Tasks were identified using the requirements table and prioritised using the MoSCoW method (Agile Business Consortium, 2022). The priority of each task was recorded in the requirements table, with more important tasks being undertaken first.

Development was guided by value-driven development (VDD) and user-centred design (UCD). VDD ensured that the artefact delivered the maximum value and provided the most important features as a priority (Kwakernaak, 2019). UCD complimented this by using user feedback to identify the most important elements and ensure that potential users would be satisfied with the product (Lanter and Essinger, 2017).

Although digital project management tools such as Jira (Atlassian, n.d.) offer robust solutions for tracking progress, they were considered unnecessary for this project due to its individual nature. Instead of relying on a digital tool that could be easily ignored, a physical note board was used to provide a constant visual reminder of outstanding tasks. This mimicked the issue raising and tracking of more formal tools, reducing the likelihood of oversight and supporting consistent development progress.

4.1.3 Toolsets and Machine Environments

This project comprised four distinct components that formed the final artefact. The first was the IGDB API (IGDB, undatedb) which provided game details. Alternatives, such as MobyGames (MobyGames, undated), were considered, however IGDB was preferred due to its excellent documentation, extensive number of entries and free access. Free access was a key consideration, as the project aimed to be open source, whereas MobyGames required a paid subscription.

To store game and user data, a MySQL database (MySQL, undateda) was used due to its compatibility with MySQL Connector/Python (MySQL, undatedc). Python was always the intended backend language because of its flexibility and powerful frameworks (Chan et al., 2019), so it was necessary to select a compatible relational database management system. Although originally planned to run on a separate server, the database was hosted locally due to logistical constraints and project deadlines. If the application was deployed at a larger scale, it would be possible to migrate it to a dedicated machine.

Flask (Flask, undated) acted as the backend , handling communication between the UI and the database. Flask was selected as it allows for RESTful request handling, required for allowing the application to interface with the database. Its built-in development server also allowed for rapid testing and suited the agile development

methodology (Fox, 2023). Postman (Postman, undated) was used to test API requests and ensure correct data handling.

Flutter (Flutter, undateda) was chosen for the frontend, due to its reduced development overhead, cross-platform potential, and hot reload functionality, which allowed rapid iteration. Its substantial widget library was another determining factor, as it allowed for the creation of a professional and appealing application, without requiring significant time investment in creating UI elements (Karasavvas, 2022).

Testing was conducted using both Android Studio (Android Studio, undated) and a physical device. Android Studio enabled quick tests with emulated devices, whilst a physical phone was used to assess the feel of the application (e.g., whether button placement felt natural). Infinite Painter (Brakefield, undated) was used during the early design stages to create UI mock-ups and refine the application's layout before implementation.

4.1.4 Risk Assessment

Several major risks where identified in the project planning stage. This section highlights four of the most problematic in Table 2, assessing their potential impact and providing the mitigation strategies employed to deal with them

Key
Low = 1
Medium = 3
High = 5

Risk No	Risk	Likelihood (L)	Severity (S)	Risk Rating (L*S)	Mitigation
1	IGDB rate limit of 4 requests per second bottlenecks performance of the application	High	High	25	See Mitigation Strategy 1
2	Overloading Flask Server due to high frequency requests will cause the application to crash	Medium	High	15	See Mitigation Strategy 2
3	Inconsistent or missing IGDB API data may cause crashes or unexpected behaviour	High	Medium	15	See Mitigation Strategy 3
4	Poor search performance or accuracy may hinder the user experience	Medium	Medium	9	See Mitigation Strategy 4

Table 2: Risk table.

- **Mitigation Strategy 1:** Cached IGDB data in the local SQL database to reduce the need for repeated requests.

- **Mitigation Strategy 2:** Ensured that requests were appropriately rate-limited and not sent unnecessarily. Requests were also condensed to minimise the number required to retrieve related information.

- **Mitigation Strategy 3:** Designed the system with flexibility in mind to prevent crashes due to missing data. The database supported null entries, and fallback placeholders were implemented in the application where necessary.

- **Mitigation Strategy 4:** While the application was not expected to match the precision of commercial search engines like Google, which has evolved continuously since 2008 (Seymour et al., 2011), a combination of full-text search and a modified Levenshtein distance algorithm (fza et al., 2015) was

implemented. This approach allowed the system to efficiently handle large volumes of data while delivering accurate and responsive search results, addressing the risk of poor performance or accuracy.

4.2 Project Design

This section of the report covers the design process for the project, presenting an overview of the software in the form of a component diagram, the database's schema, a breakdown of the Flask server and a review into the UI design process.

4.2.1 Software Overview

There are four main pieces of software that work together to create the artefact, shown in Figure 3. These components can be grouped into three logical layers: a presentation layer (Flutter application), an application logic layer (Flask server), and a data layer (SQL database and IGDB API). This layered architecture supports a clear separation of concerns, allowing the system to remain modular and scalable.

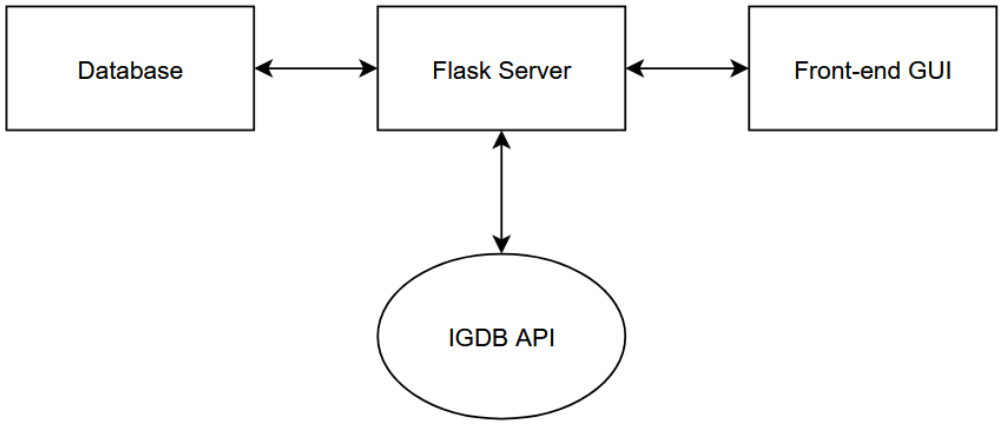


Figure 3: Top level view of architecture.

Each component is further broken into subcomponents. Figure 4 outlines their interactions, giving a clear overview of the system. The SQL database stores user details, game details, and user game history, enabling search functionality and recommendations. The Flask server accesses the IGDB API to retrieve game data, parse it, and store it in the database. It also manages communication between the Flutter application and the database, including user authentication, the recommendation engine and game data retrieval.

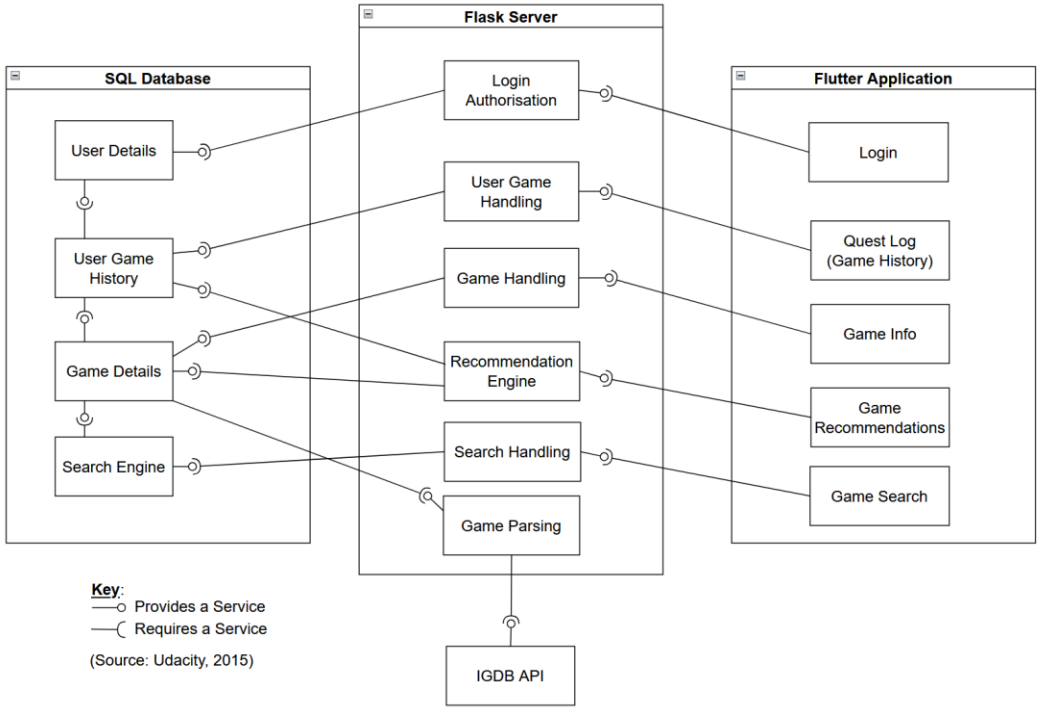


Figure 4: Component diagram (Udacity, 2015) of the software architecture.

4.2.2 Database Schema

Although game data is accessible directly via the IGDB API, the rate limit of four requests per second created a significant bottleneck (IGDB, undateda). To address this, essential data is cached in a structured database, following IGDB's

recommendation for local storage to reduce API load and give developers more control over data handling. (IGDB, undateda).

A robust database was therefore designed to efficiently store and manage approximately 250,000 records. As the foundation of the entire application, it was crucial it remained atomic, normalised, and efficient. Figure 5 provides an overview of the schema, showing how the various tables integrate. These tables can be grouped into three categories: User, Game History, and Game Information.

User tables (pink) store user data. Personal data is kept separately from public facing information to allow the table to be encrypted in a full-scale deployment. The relationships table stores friendships between users as well as the trust rating users have for their friends.

Game history tables (green) handle users' data in relation to a given game, including information such as ratings, status (playing, completed, etc.), and dates played.

Finally, game information tables (blue) relate to the games themselves. Game requires a wide range of data, with key information relating to them being stored in the games table and details that are not always required in the application being stored in separate tables. This means that functions and procedures require less operational time, which is essential given the scope of the database. A lot of data, such as genres and companies, is shared between games. Relational tables are used to establish one-to-many relationships between games and this shared information, avoiding redundancy and ensuring the database is atomic. The game_recommendations table, contains binary vectors for each game, that are used in the recommendation process.

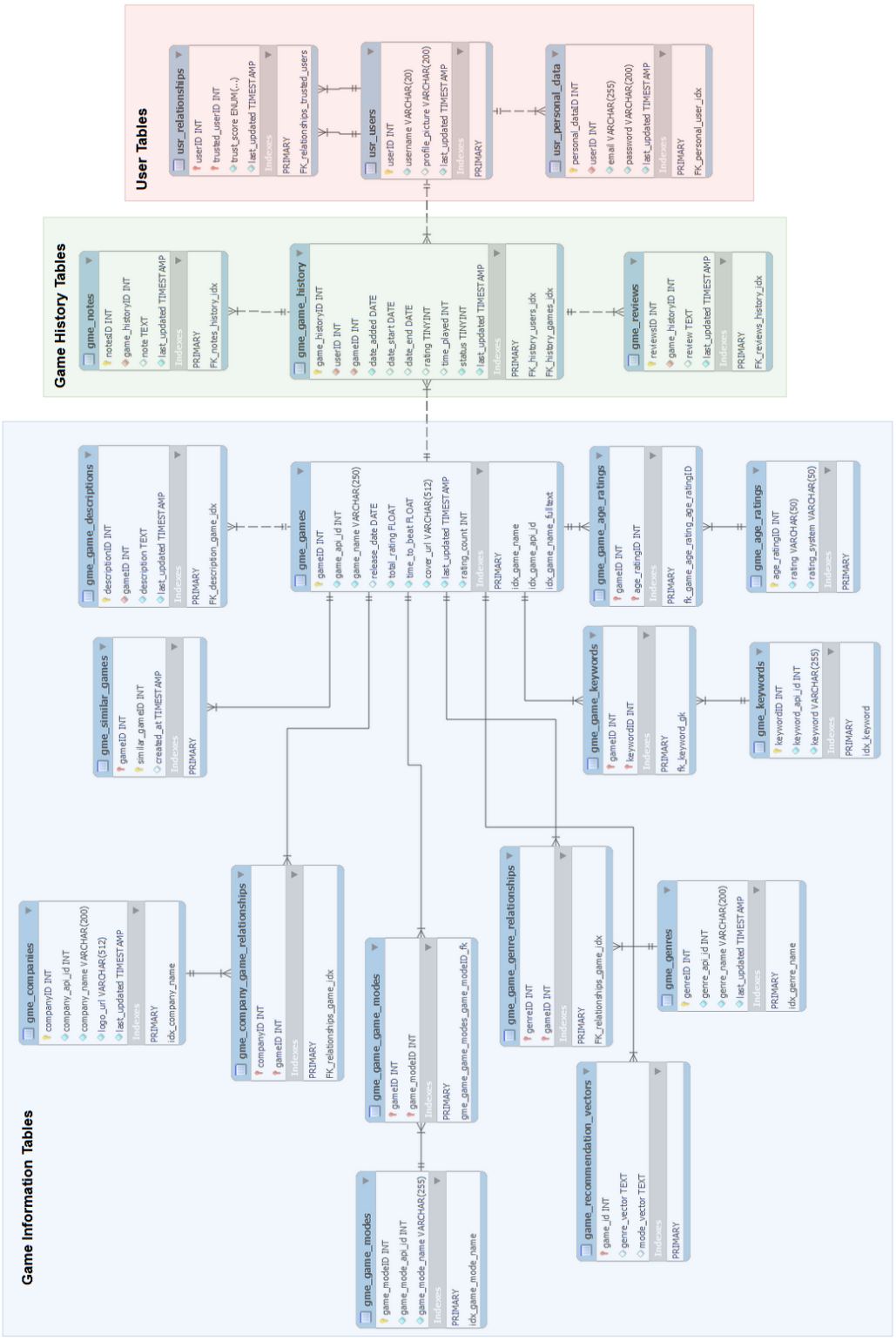


Figure 5: An Enhanced Entity-Relationship (ERR) diagram of the database.

4.2.3 Flask Server Class Diagram

The Flask server facilitates communication between the frontend of the application and the database, and handles fetching game information from the IGDB API. The diagram in Figure 6 shows this structure.

The top left of the diagram shows the classes that handle fetching and processing data from the API. The API requires authentication, which it handles through Twitch API (Twitch Developers, undated), so the TwitchAuthenticator class provides this. The IGDBAPI class handles connections to the API and GameDataParser processes the response before it is inserted into the SQL database. When retrieving data for all relevant games the populate database functions were used to save the ID of the most recently processed games to avoid losing progress

The data handler classes use the DatabaseConnection class in the centre of the diagram to manage the connection with the database, while the handler classes carry out operations. The routes classes handle the API connections that the server provides to the Flutter application. The processing classes, positioned between the handler and routes, handle data transformation and validation. This separation ensures modularity, making it easier to modify individual sections without affecting the entire system.

The UserRecommendationManager class handles recommendations of games for users. It relies on the recommender engine to generate the suggestions and is utilised by the user_game_routes class to provide them to the application.

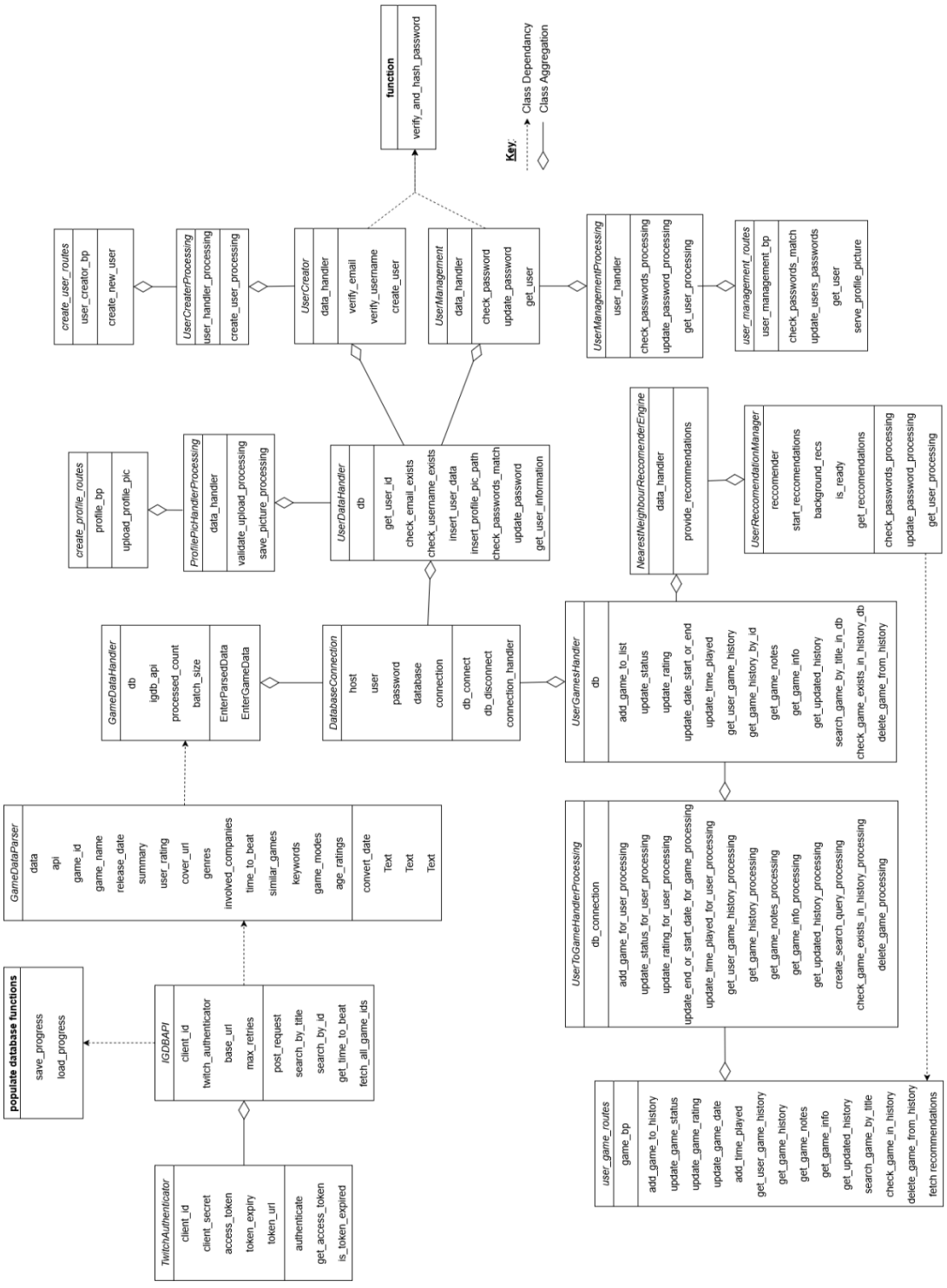


Figure 6: Class diagram of the Flask server. Classes with diamonds touching them rely on an instance of the connected class. Dotted arrows show the class depends on another class or function.

4.2.4 UI Design

The UI design was an iterative process that ensured a clear visual reference was available during frontend development. The final designs for several of the pages can be seen in Figures 7 to 13, with breakdowns of the planned functionality. These designs were adjusted based on feedback from a survey that was conducted before the development of the frontend began (Section 4.3). Due to limitations of infinite painter, all text was handwritten in these designs. However, a real typeface was used in the final implementation.

Figure 7: Main screen of the application.

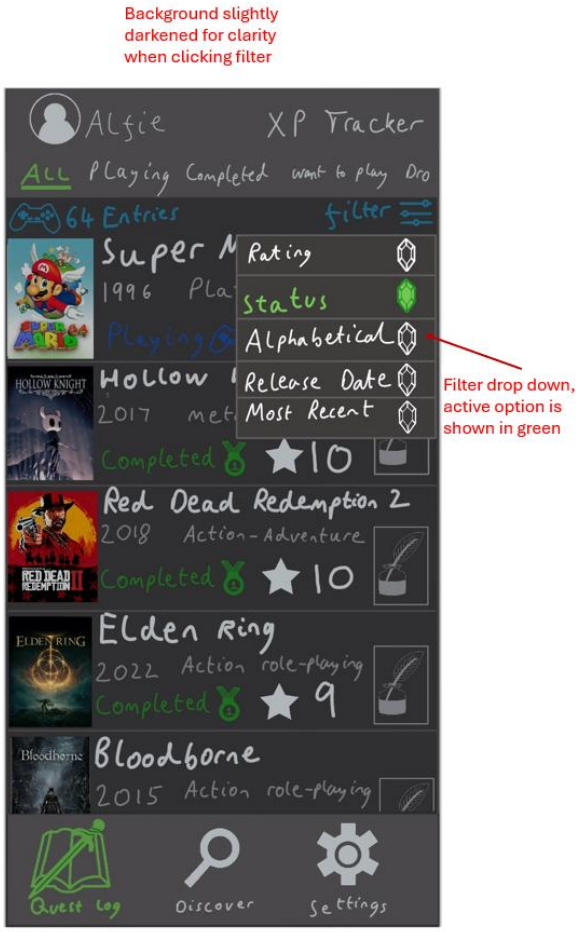


Figure 8: Example of the filter tab.

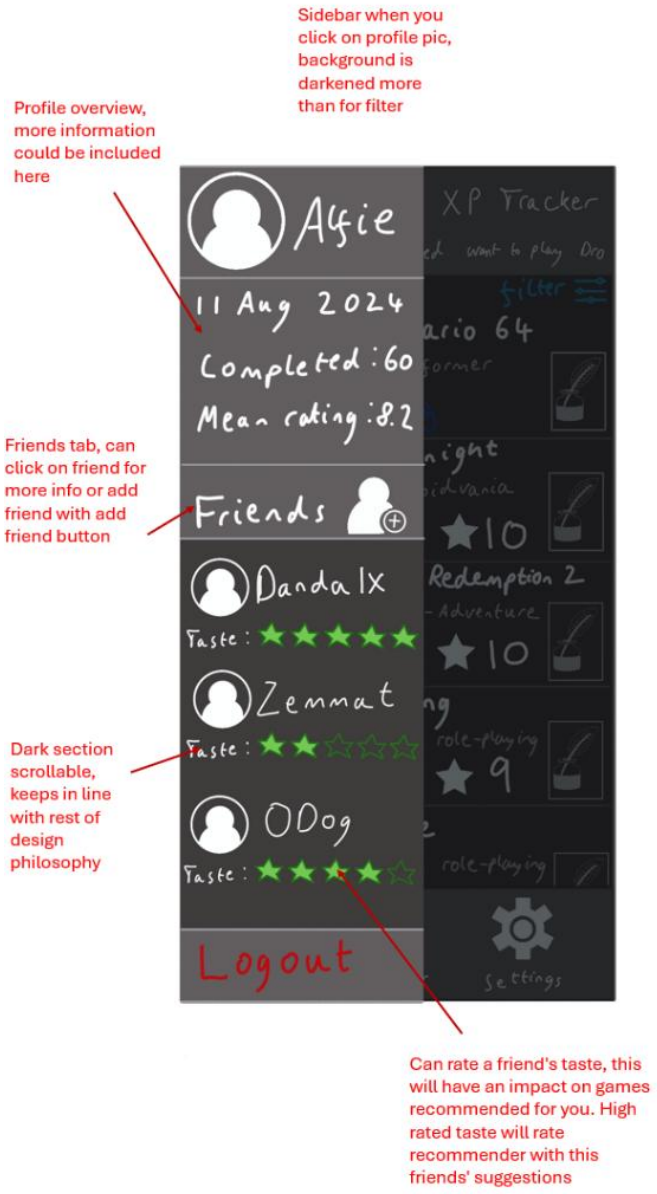


Figure 9: Sidebar with user and friends' information.

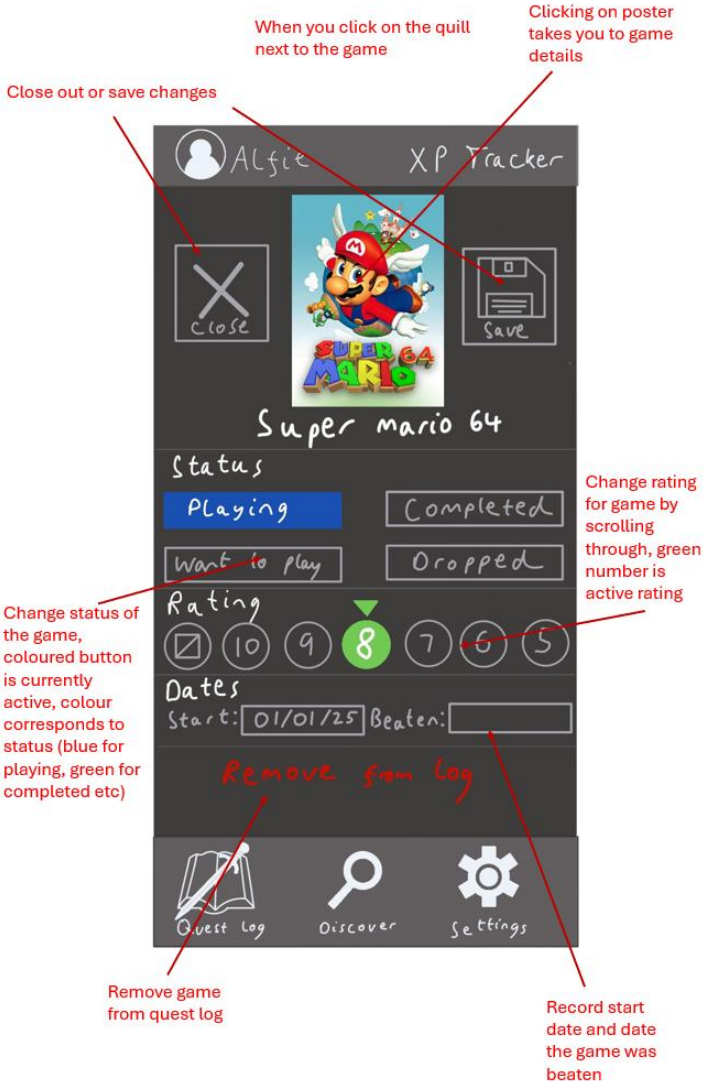


Figure 10: Edit page for a game in the 'Quest Log'.

Figure 11: Game information screen.

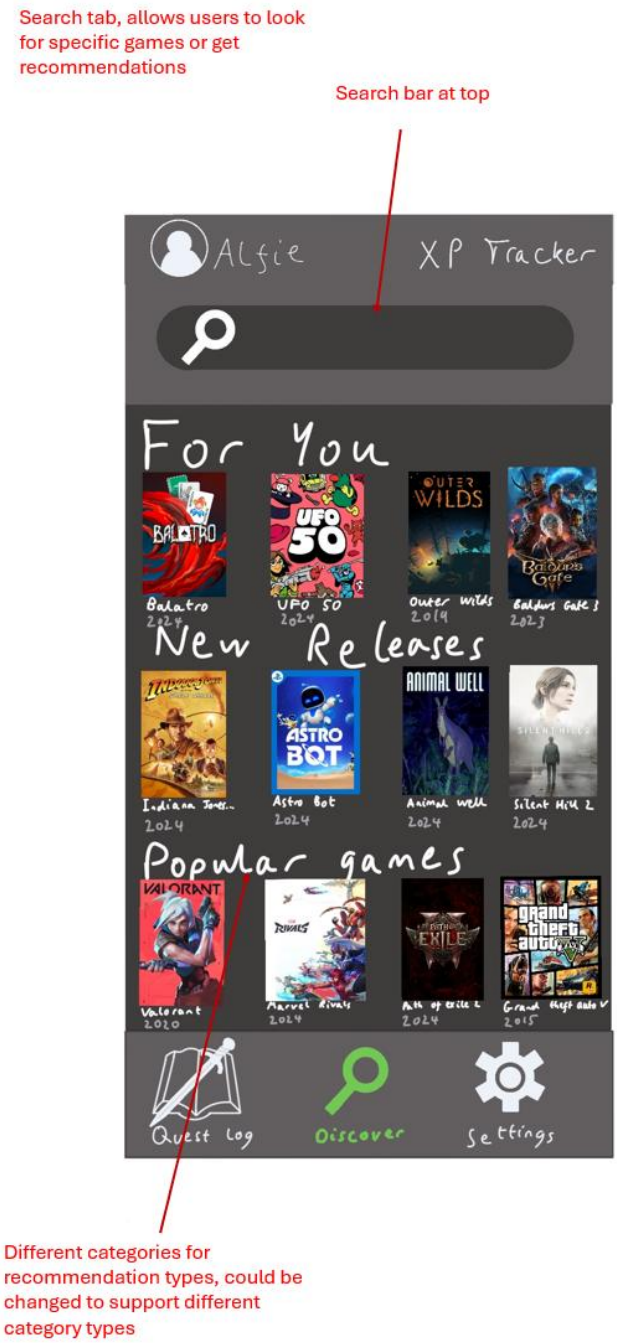


Figure 12: Discover page.

Figure 13: Application Settings.

4.2.5 Application Architecture

Figure 14 shows how navigation is handled between pages. After logging in, the user is directed to the ‘Quest Log’ page, which acts as the application’s home. From here, there are multiple ways to reach each page, represented by the arrows connecting them in the screen flow diagram.

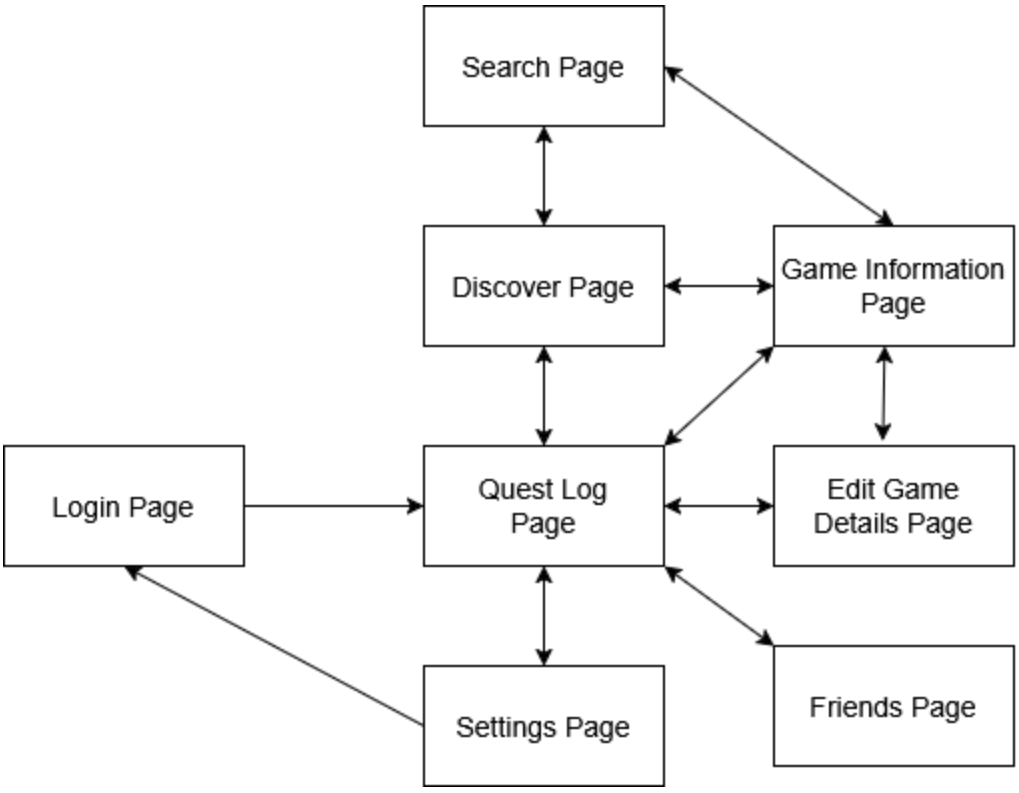


Figure 14: Screen flow diagram of app navigation.

4.3 UI Design Survey

An anonymous UI survey (Appendix A) was conducted following the initial designs outlined in Section 4.2.4, with participants being recruited through word of mouth and public posts. The survey provided an overview of the application's functionality and included questions to assess participants' prior knowledge of video games before gathering feedback on specific screens. This section summarises key findings and the resulting design changes.

4.3.1 Respondent Sentiment

The survey received 31 responses, which provided a good baseline for user opinions. Of those respondents, 17 regularly played video games, and 14 had considered keeping track of their gaming activity, but only 4 currently did, suggesting there is room in the market for this application.

Feedback on the application's design was largely positive, with most users rating screens highly (4–5/5) and describing the layout as intuitive. It was apparent from some of the negative responses given in long-form questions that some participants misunderstood some questions, or the purpose of the application. In hindsight, questions about age or technical familiarity would have helped filter out responses from outside the target demographic.

One aspect that divided opinion was the application's colour scheme. This was expected, as the app is intentionally designed with a dark theme, which may not appeal to all users. A light mode was always planned but was not shown in the survey, which may have alleviated these concerns.

The design takes inspiration from gaming culture, with elements such as the filter icons resembling gemstones from '*The Legend of Zelda'* series (Nintendo, undated) and the 'Quest Log' name evoking a classic RPG feature. The vibrant colours against the dark background reinforce this theme. As a result, respondents with little to no gaming experience may have been less receptive to the design choices.

4.3.2 Refinements and Adjustments

Most of the answers received were positive, however, some good suggestions were made, a few of which are detailed below

Most suggestions surrounded the ‘Quest Log’ page. Some respondents questioned the inclusion of the game release dates; however, these were included to help identify games (e.g. original vs remake). Some participants felt that the screen was cluttered, however an application of this nature needs to display a high level of information. The handwritten text may also have contributed to these comments as the presentation was less refined than a professional typeface.

To reduce clutter, the decision was made to remove the game genres from the ‘Quest Log’ page and to only show them in ‘Game Information’ due to their highly variable length. This was replaced with time to beat, as one respondent mentioned that they would like the option to see the length of games at a glance and it was considered useful information with a consistent length. In line with this, an option to filter by ‘time to beat’ was added.

Additional refinements included: labelling the edit button as some people were confused on how to edit the details for a game, slight colour adjustments for readability, and adding the option to filter by recently updated. These small changes resulted in a cleaner, intuitive and visually appealing implementation. Additional suggestions were acted upon for other pages of the application, such as reducing the size of buttons on the ‘Edit Game Details’ screen.

4.4 Planned Testing

Most of the planned testing focused on functional tests to ensure that features such as logging in, searching for games, and adding or removing games from a user’s list worked as anticipated. This was planned to be primarily conducted using Postman, which allowed for testing of API calls and easy validation of response data.

Usability testing was also planned to help guarantee the application was intuitive and user-friendly. This included screen layout, how information was presented, the

position of a button, or the navigation flow. The main ways that usability testing would be incorporated were through the interface survey (Section 4.3), interviews conducted at the end of development, and real-world testing on both a physical device and an emulator. Running the application on a physical device would be particularly valuable, as it would provide a more realistic feel, especially when assessing button placement and navigation. Ideally this testing would cover multiple devices across both major mobile operating systems: Android and IOS. Design decisions were planned to be guided by Nielsen's 10 Usability Heuristics (Nielsen, 2024), which would be continually referenced when making interface changes.

Edge case testing was planned to ensure long pieces of text overflowed as expected and that missing data was handled correctly. This would be particularly relevant given the reliance on external API data, where fields may occasionally be null or missing.

Performance testing targeted features most likely to be affected by latency, such as the search functionality, recommendation system and game history retrieval. Performance could not compromise accuracy, so tests to ensure that the expected outcomes occur would coincide with performance testing.

Finally, basic security testing would be conducted to ensure that that users could only access their own data, password checks worked as expected, and API data was validated. If the application was to be released publicly, more extensive security testing would be required.

Chapter 5: Implementation

This chapter of the report details the technical implementation for each section of the application, showing how the desired functionality was provided, and how the project aims were met.

5.1 Database

5.1.1 Stored Procedures

The SQL server contains stored procedures which provide essential services for the application. They are primarily used to insert, update or retrieve information relating to the various games and users. A few of them are highlighted in this section, but not all are covered due to their similarities. The full range of stored procedures can be seen in the artefact.

5.1.1.1 Adding a Game to the Database

When adding new games to the database, the application uses a handful of procedures to handle inputting the relevant information. The most significant of these is the InsertOrUpdateGame procedure, which accepts the game's core details, its description, and an array of similar game IDs as parameters. Details such as related companies and genre information are handled by separate procedures for clarity, but transactional changes are only committed when all operations have been enacted successfully. This process is covered in Figure 15.

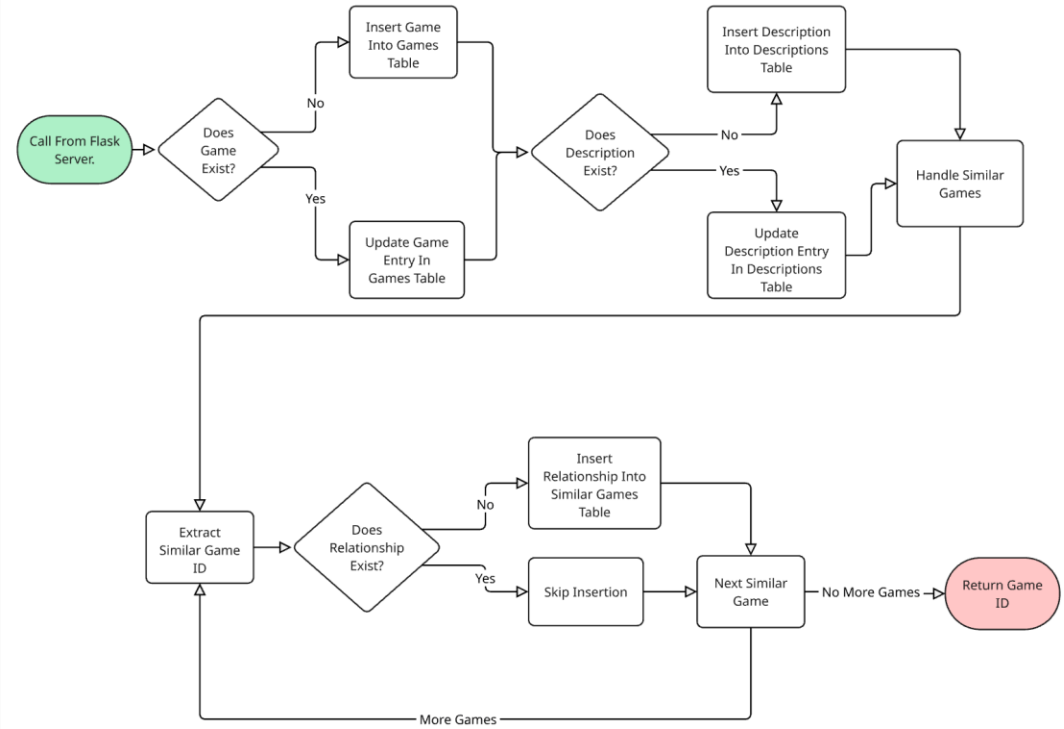


Figure 15: Flow of adding a new game to the database.

5.1.1.2 Retrieving Game Information

The GetGameInfo stored procedure is the most complex query in the database, and handles the return of all information that is shown on a game's detail page. The stored procedure has the game's ID provided and returns all relevant information for that game, by joining together the data from multiple relational tables. As some attributes contain multiple values, such as the involved companies, GROUP_CONCAT is used to return comma-separated lists that can be used by the frontend.

Furthermore, the procedure also contains a subquery that compiles a JSON array of similar games. Each similar game is returned with its overview data, enabling users to access related titles directly

This procedure is important because it significantly reduces the need for individual queries from the application, thus improving performance and simplifying the logic for the Flask server. The data retrieved from each table can be seen in Figure 16.

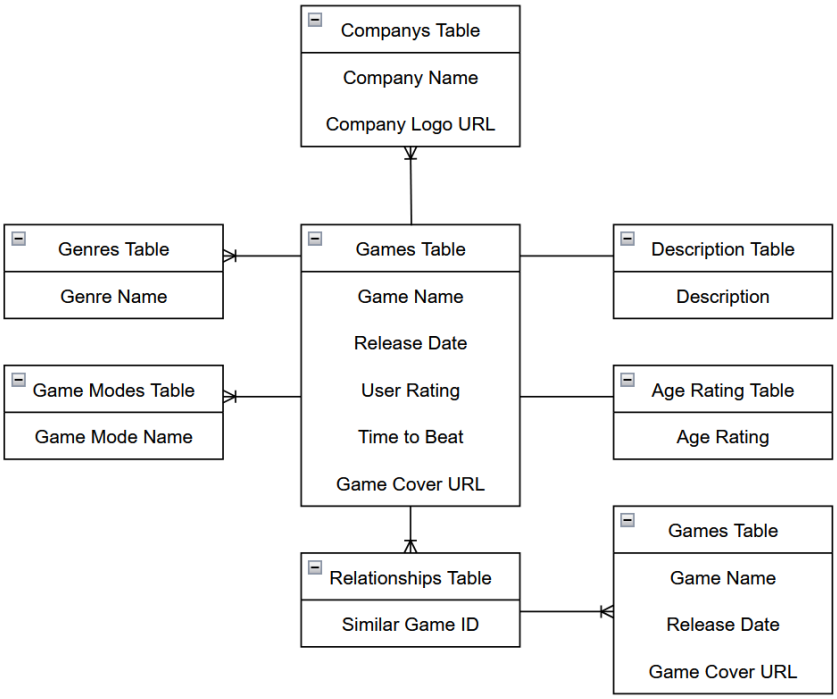


Figure 16: The tables involved in GetGameInfo.

5.1.1.3 Adding a Game to a User's History

AddToGameHistory handles a user adding a game to their Quest Log. This procedure takes in the user and the game's ID as well as the integer corresponding to the status the user selected for the game (playing, want to play, completed or dropped). Whilst other information alongside the status is stored in this table, only the status is required when making a new entry. The procedure contains several layers of validation before committing to the database. The process is shown in Figure 17.

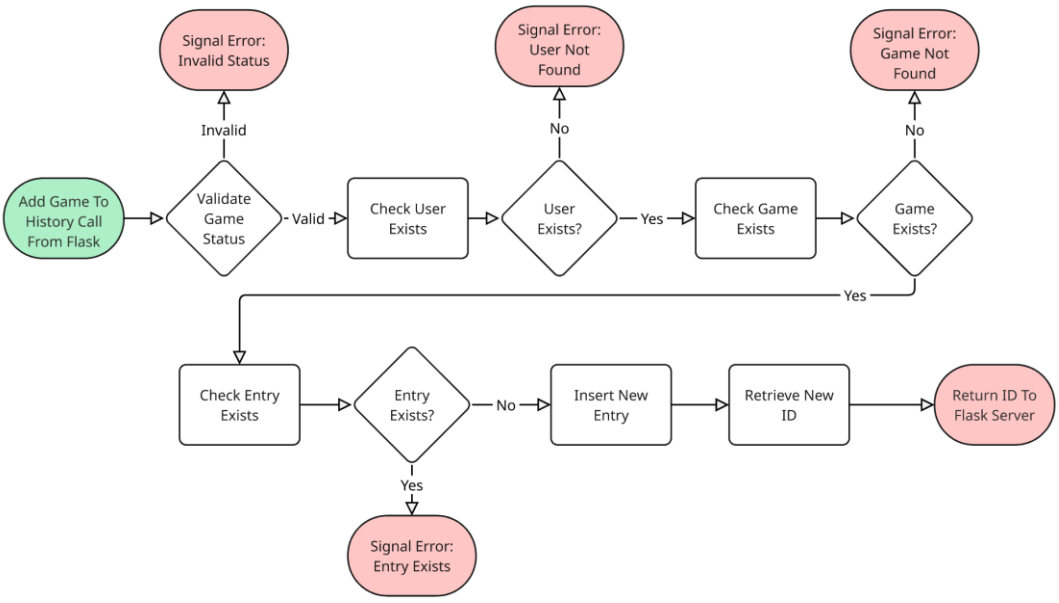


Figure 17: Flow of adding a game to a user's history.

5.1.2 Search Engine

The database also facilitates the search engine. This comprises a procedure that processes the request, a full-text search, and a string distance algorithm to narrow down the search results.

The SearchGamesByTitle stored procedure receives a varchar input. This input is compared against every game title in the gme_games table. This table contains a full-text index, consisting of all the game titles. Natural language full-text search is performed, which interprets the input string as a human-language phrase (MySQL, undatedb). This search is extremely fast, even with a large dataset, however, it returns too many results to be useful for accurate suggestions. For example, searching for 'Super Mario 64' returns over 1,000 results, with the intended match appearing in the 13th row (Figure 18). Consequently, full-text search alone was deemed insufficient for accurate result generation.

gameID	game_name	release_date	cover_url
246961	Super Mario XP: Super Mario Land	NULL	https://images.igdb.com/igdb/image/upload/t_cover_big/co93w2.jpg
247502	The Super Mario Bros. Super Show: Mario's Greatest Movie Moments - ...	2002-12-31	https:NO URL
182144	Super Mario World: Mario to Yoshi no Bouken Land	1991-12-31	https://images.igdb.com/igdb/image/upload/t_cover_big/co826d.jpg
141500	Super Super Super Super	2015-01-26	https://images.igdb.com/igdb/image/upload/t_cover_big/co5egx.jpg
311	Super Mario Bros.	1985-09-13	https://images.igdb.com/igdb/image/upload/t_cover_big/co6pib.jpg
912	Super Mario Bros.: The Lost Levels	1986-06-03	https://images.igdb.com/igdb/image/upload/t_cover_big/co23bj.jpg
913	Super Mario Bros. 3	1988-10-23	https://images.igdb.com/igdb/image/upload/t_cover_big/co7ozx.jpg
914	Super Mario Land	1989-04-21	https://images.igdb.com/igdb/image/upload/t_cover_big/co7o14.jpg
915	Super Mario World	1990-11-21	https://images.igdb.com/igdb/image/upload/t_cover_big/co8lo8.jpg
916	Super Mario Land 2: 6 Golden Coins	1992-10-21	https://images.igdb.com/igdb/image/upload/t_cover_big/co7gxg.jpg
917	Wario Land: Super Mario Land 3	1994-01-21	https://images.igdb.com/igdb/image/upload/t_cover_big/co216h.jpg
918	Super Mario World 2: Yoshi's Island	1995-08-05	https://images.igdb.com/igdb/image/upload/t_cover_big/co2kn9.jpg
919	Super Mario 64	1996-06-23	https://images.igdb.com/igdb/image/upload/t_cover_big/co721v.jpg
920	Super Mario Sunshine	2002-07-19	https://images.igdb.com/igdb/image/upload/t_cover_big/co21rh.jpg
921	New Super Mario Bros.	2006-05-15	https://images.igdb.com/igdb/image/upload/t_cover_big/co21rm.jpg
922	Super Mario Galaxy	2007-11-01	https://images.igdb.com/igdb/image/upload/t_cover_big/co21ro.jpg

Figure 18: A segment of the search results from 'Super Mario 64' using full-text search.

To improve accuracy, a distance algorithm was incorporated. Among the available options, the Levenshtein distance algorithm (Algorithm 1) was selected due to its effectiveness in comparing string similarity based on edit distance, where a lower number indicates higher similarity (Nam, 2019). Algorithm 2 converts this distance into a normalized ratio (0–100%) for easier thresholding. A MySQL-compatible implementation of the Levenshtein algorithm was sourced from GitHub (fza et al., 2015) and adapted for use with the video game dataset. Whilst it does provide accurate results, the algorithm is too computationally expensive to be applied to the entire dataset.

To mitigate this, the full-text search was combined with the Levenshtein algorithm. As seen in Algorithm 3, the full-text search is first used to narrow down the search results before the Levenshtein algorithm filters them by similarity score using the Levenshtein ratio and minimum rating. A ratio threshold of 35 was chosen to ensure meaningful textual relevance whilst accommodating minor typos. The implementation of this search can be seen in the final application in Section 6.1 and testing in Section 6.3.1.

Algorithm 1: Levenshtein Distance Calculation

```
Function LEVENSHTEIN(s1, s2)
    Input  : Strings s1, s2 (max length 255)
    Output: Integer edit distance
    DECLARE s1_len, s2_len, i, j, c, c_temp, cost: INT;
    DECLARE s1_char: CHAR;
    DECLARE cv0, cv1: VARBINARY(256);
    SET s1_len = CHAR_LENGTH(s1),
     s2_len = CHAR_LENGTH(s2);
    SET cv1 = 0x00, j = 1, i = 1, c = 0;
    if s1 = s2 then
        return 0
    else if s1_len = 0 then
        return s2_len
    else if s2_len = 0 then
        return s1_len
    else
        while j ≤ s2_len do
            SET cv1 = CONCAT(cv1, UNHEX(HEX(j))), j = j + 1;
        while i ≤ s1_len do
            SET s1_char = SUBSTRING(s1, i, 1), c = i,
             cv0 = UNHEX(HEX(i)), j = 1;
            while j ≤ s2_len do
                SET c = c + 1;
                if s1_char = SUBSTRING(s2, j, 1) then
                    SET cost = 0;
                else
                    SET cost = 1;
                SET c_temp =
                 CONV(HEX(SUBSTRING(cv1, j, 1)), 16, 10) + cost;
                if c > c_temp then
                    SET c = c_temp;
                SET c_temp =
                 CONV(HEX(SUBSTRING(cv1, j + 1, 1)), 16, 10) + 1;
                if c > c_temp then
                    SET c = c_temp;
                SET cv0 = CONCAT(cv0, UNHEX(HEX(c))),
                 j = j + 1;
            SET cv1 = cv0, i = i + 1;
    return c;
```

Algorithm 1: Levenshtein Distance algorithm.

Algorithm 2: Levenshtein Ratio Calculation

Function *LEVENSHTEIN_RATIO(s1, s2)*

DECLARE $s1\_len$, $s2\_len$, max_len: INT;
SET $s1\_len = LENGTH(s1)$, $s2\_len = LENGTH(s2)$;
if $s1\_len > s2\_len$ **then**
    SET $max\_len = s1\_len$;
else
    SET $max\_len = s2\_len$;
return $ROUND\left((1 - \frac{LEVENSHTEIN(s1,s2)}{max\_len}) \times 100\right)$;

Algorithm 2: Levenshtein Ratio calculation.

Algorithm 3: SearchGamesByTitle

Input: search_title: user input string
Output: Top 5 games with fields: `gameID, game_name, release_date, cover_url`

1. Perform full-text search on `game_name` for `search_title`, limit to 100 games.
2. Filter results where:
   - `LEVENSHTEIN_RATIO(game_name, search_title) > 35`
   - `total_rating > 0`
3. Sort remaining games by `LEVENSHTEIN_RATIO` (descending).
4. Return top 5 games.

Algorithm 3: Pseudocode for the SearchGamesByTitle algorithm.

5.2 IGDB Integration

The IGDB API is used to populate the database with game details. This process is facilitated by the Flask server.

5.2.1 Retrieving Game Data

Integration with the IGDB API begins by acquiring an access token via the Twitch API (Twitch Developers, undated), a process handled by the TwitchAuthenticator class. Since the access token can expire, the class ensures that it is valid prior to use, requesting a new one if required.

The IGDBAPI class uses this token whenever a request is made. The class contains a generic post request function that retries the request up to 3 times in the event of failure. The other functions of the class use this method to make requests for games, searching by various identifiers.

During development, the initial requests were made manually for testing. However, to support search functionality being integrated, it was important to have the information for every game locally. To populate the local database, a batch process was introduced to fetch every game ID from IGDB in chunks of 500. The process then iterated through each ID, retrieving all relevant game data. Due to IGDB's data structure, each game required two separate calls, and with the platform’s rate limits in place, runtime equated to around 4 days. Helper functions were implemented to periodically save progress and resume the operation as needed. A flowchart illustrating this process is shown in Figure 19. At present, the system does not automatically fetch new games, but future iterations could include syncing with changes in the IGDB database.

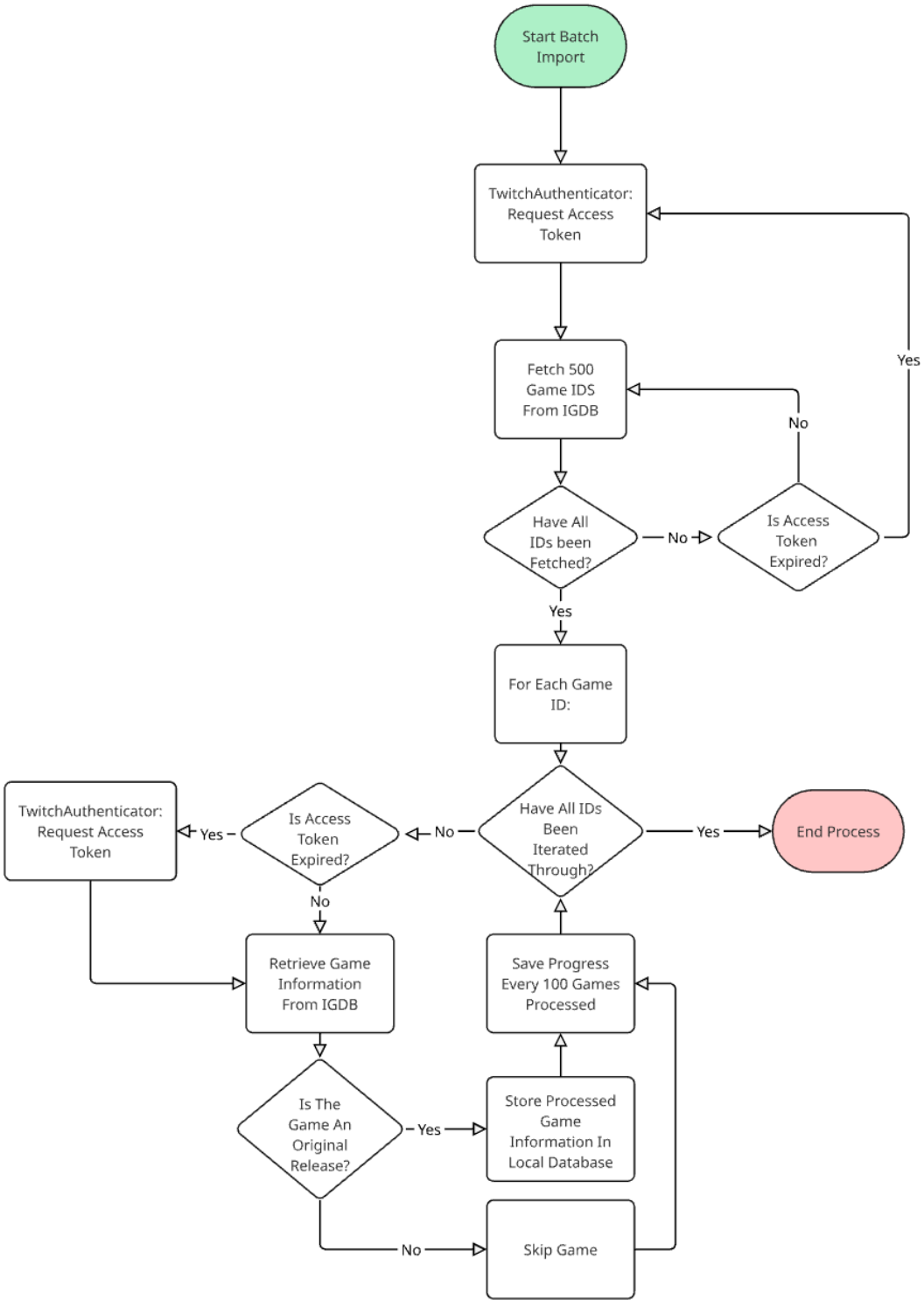


Figure 19: The process of retrieving all game information.

5.2.2 Inserting Game Data

Game data received from the IGDB API is parsed into the desired format by GameDataParser. For example, dates are converted from Unix timestamps to a YYYY-MM-DD format, the game length is converted from seconds into hours, and a dictionary is created consisting of the involved companies. The formatted data is passed as a dictionary to the GameDataHandler class for database insertion.

The DatabaseConnection class manages database connections using the MySQL connector module. It employs a context manager to ensure connections are safely opened and closed, preventing leaks and improving reliability.

Once connected, the GameDataHandler class uses its EnterParsedData method to insert the data via stored procedures, including the one covered in Section 5.1.1.1. Changes are only committed once all stored procedures have been successfully executed to maintain database atomicity. This process is displayed in Figure 20.

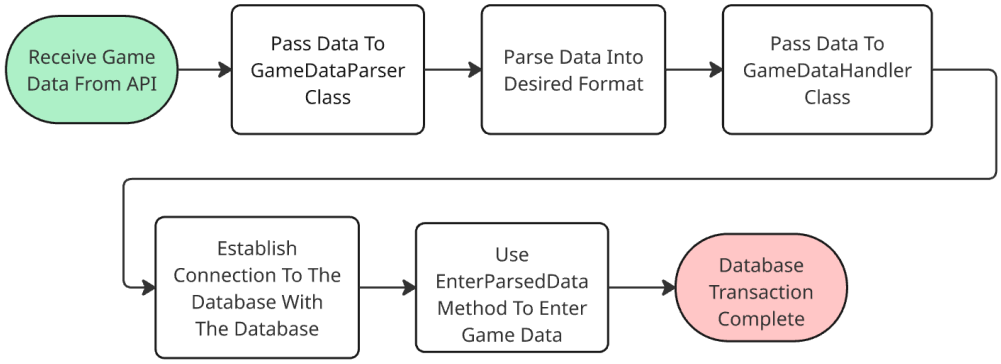


Figure 20: The process of inserting a game into the database.

5.3 Backend

The Flask server enables communication between the application and the SQL database, with functionality split across 4 route classes. These routes handle creating

a new user, other user management (password verification, changing details etc), profile pictures, and user details for games. The routes handle receiving requests and returning responses to the frontend.

Rather than embedding all logic within the route classes, processing is delegated to dedicated frontend connection classes. This separation supports modular development, keeps the codebase clean, and allows for future expansion. Database operations are abstracted further into respective handler classes, maintaining a clear separation of operations within the backend.

Outside of communication, the server also hosts the recommendation system, generating game suggestions when needed.

5.3.1 User Management

When creating a new user, the request is first received by the create_user_routes class. Once confirmed that no fields are missing, the UserCreator class verifies the email address and username are in appropriate format using regex patterns and checks to make sure that neither already exist within the database. The provided password is verified and hashed before all the details are inserted into the database. Hashing is handled using the bcrypt Python module (Python Cryptographic Authority, undated) with a salt generated using 12 rounds, providing resistance to brute-force attacks. The hash is stored in the database and checked with the salt key when needed. This approach is appropriate for an application of this scale, but a more secure hashing algorithm such as argon2id would be preferred for a full release (Python Cryptographic Authority, undated).

The UserManagment class deals with processes relating to existing users, such as checking an inputted password matches the password stored in the database, updating an existing password, or retrieving a user ID. All user related operations are handled by this class except for changing a profile picture. This is because

saving a profile picture involves writing the file directly to the computer's storage, with only the file path stored in the database for efficiency.

5.3.2 Game Management

Operations relating to a user and games in their ‘Quest Log’ are facilitated by the UserGamesHandler class. This is the most feature rich segment, due to the sheer number of functionalities the application provides. It includes functions such as: retrieving a user’s game history, adding new games to the history, searching for new games, or updating details for a specific game. These operations were designed to be robust and handle errors gracefully, ensuring reliability during frequent day-to-day use.

5.3.3 The Recommendation System

While advanced models were explored, a KNN model was chosen for the final artefact (scikit-learn, undated). This was deemed as acceptable, since the solution is relatively lightweight and provides results in approximately 5 seconds. This solution is ideal for the responsive user experience the application is aiming for, and can generate recommendations quickly without a large amount of computational overhead.

Whilst the Rating Bayesian Personalised Ranking model proposed by Feng et al. (2021) was not used due to its computational complexity being too high to apply to the large dataset, it inspired the use of a hybrid approach. Both content-based filtering, via genre and mode similarity, and elements of collaborative filtering using user-specific preferences derived from historical ratings were incorporated. This improves relevance and adapts to user-specific tastes without requiring large amounts of user data. This is applied to both general recommendations (For You)

and recent releases, using the same model. The only difference is that recent releases are pre-filtered by release date before being passed to the model, creating temporally relevant suggestions.

To ensure a high level of performance, games are pre-filtered based on their average IGDB user rating and the number of users who contributed to that rating. This guarantees only high-quality, relevant games are suggested using the GetGameSuggestions stored procedure. After some testing, it was determined that filtering results to only games rated over 70 and with over 40 votes provided the best suggestions. Games that are already in the user's history are filtered out to ensure novelty in the recommendations provided.

Each game is represented using binary vectors for both genre and game modes. A user profile is constructed by computing a weighted average of the vectors from the games in the user's log, with weights based on user ratings. This average vector is then compared against candidate games using cosine similarity to determine the closest matches. Cosine similarity was chosen as it is not affected by the scale of the vectors, it is proven to work well for text analysis, and the data being used is relatively low-dimensional (Wang and Dong, 2020).

Upon a user logging in, a background thread is started to generate their recommendations. The number of recommendations desired is 15, however if less games are available then the number of neighbours is reduced to match the sample size. Once complete, they are stored in an instance of the UserReccomendationManager class, and a flag is set to indicate they are ready. When a request is received, the recommendations stored within the class are returned immediately if they are available. This process is represented in Algorithm 4.

Algorithm 4: Nearest Neighbour Recommendation Process

User ID u, recent flag r **Preprocessing: Filter candidates in SQL**
Execute stored procedure `GetGameSuggestions(`u`,` r`)` to retrieve up to 20,000 candidate games meeting:

- Minimum rating of 70 and more than 40 votes
- Valid genre and mode vectors (at least one non-zero bit)
- Not already played/rated by user u
- Released within 9 months (if r = TRUE) or past 12 years excluding recent releases

Let D be the filtered dataset and U be the user's game history with ratings
Vector Conversion each game g in D Convert g's binary genre and mode strings to numpy arrays Stack genre and mode vectors to form matrix X
User Profile Generation Filter U to keep only entries with non-null ratings Convert each game in U to genre and mode vectors Stack vectors to form matrix X_u and extract ratings vector r_u Compute weighted average user vector: $v_u = \frac{X_u^T r_u}{\sum r_u}$
Nearest Neighbour Search Let $n_{samples} = |X|$ Set $n_{neighbors} = \min(15, n_{samples})$ Fit `NearestNeighbors(n_neighbors=`$n_{neighbors}$`, metric='cosine')` on X
Query v_u to get $n_{neighbors}$ nearest games in X based on cosine distance
Postprocessing each recommended index i Retrieve full game info from database using game ID Add similarity distance and game details to result
List of recommended games with metadata

Algorithm 4: Pseudocode for the recommendation system.

The one downside of this solution was that, since a friend system was not implemented, it was not possible to provide trust weighted recommendations. This would need to be included in future iterations of the application to address the cold start problem. Another solution could be to provide a welcome questionnaire so that users can choose game genres they are interested in. Trending and well-rated games in those genres could then be suggested until sufficient data for accurate

recommendations is obtained. If the questionnaire is skipped, then the application could suggest trending and critically acclaimed titles to ensure initial engagement.

5.4 Flutter Application Implementation

Due to the complexity of the Flutter application and word count constraints, only key aspects are covered in detail. The full program can be seen within the artefact. Different screens of the application are broken into pages, and these will be covered in the order of natural navigation. The settings page is not covered, as in the current version it only includes the ability to toggle light/dark mode and log out of the application.

5.4.1 Login Process

The login page allows users to sign in to their account. The page uses two instances of a text controller to receive the username and password, with togglable obfuscation for the password field. In the current version, there are placeholders for a forgot password and register button, but these are not implemented due to issues surrounding automating two factor authentication emails. When the user presses the sign in button, the inputted details are verified using the check_passwords_match route. If the details are correct, then the application stores the user ID into shared preferences for future operations and loads into the home page with a custom transition.

Upon loading, the home page fetches the user's profile picture and username (in case email login is implemented in future) as well as the top bar and bottom navigation bar. The space in between the two bars contains either the 'Quest Log', 'Discover' or 'Settings' page. These pages can be navigated between using either the buttons at the bottom or swiping on the screen.

5.4.2 Quest Log Page

The ‘Quest Log’ page is where the application opens by default. At the top of the page are the multiple filter options and the entry count, each of which are loaded from their component files. Animations for the filter and entry information drop downs are controlled here. Whenever this page is loaded, a request is made to retrieve the users game history. This information is then filtered based on the selected status and passed into the GameList widget. This widget takes up most of the page, showing all the games in the user’s log. Before any games are displayed, the games are ordered based on the selected filter (alphabetical, rating, release date, etc.). Five games are displayed at a time, and the list can be scrolled through. The game information that has been passed in is loaded into appropriate variables and formatted as required. Of note here, a Hero widget is used for the game cover images as this allows them to ‘fly’ between screens, creating a smoother transition (Flutter, undatedb). Each entry receives its own item in a list, with all the relevant information displayed. An edit button is also present for each entry and can be used to access the screen where users can edit the details for that entry. If the user presses anywhere else on an entry, they are taken to a screen that provides information on that game.

5.4.3 Game Information Page

The game information page provides information on the selected title. When loaded, this information is retrieved using the Flask server and formatted as needed. This page must be flexible as different entries contain different levels of detail. Two areas of interest here are the age rating and the similar games information. The age rating is only returned as text, so the corresponding jpegs are stored as assets within the application and are loaded when needed. The similar games are passed in as an array of dictionaries and displayed in a scrollable list. When one of these is clicked, the user is taken to the page for that game. To allow easy browsing, the page includes

both a back arrow (to return to the previous game) and a cross icon to return to the home page.

This page was the most problematic part of development because of data consistency. In the bottom right corner is a floating action button that allows users to either edit or add the game to their log. The icon changes depending on the context (editing vs adding). To ensure that this button displays the correct symbol, and all the details for the game stay up to date, a query must be made to the corresponding history entry whenever the page is loaded. If this is skipped, then the application ends up out of sync with the database. Implementing this correctly took time, but the final version performs smoothly and ensures that app state remains accurate. This functionality was crucial, as it significantly improved the app's overall flow and reliability.

5.4.4 Edit Entry Page

The edit entry page allows users to change their details for a game (status, rating, etc.) It can be accessed from both the 'Quest Log' and the game information pages. Users select the status with the corresponding buttons, the rating with a scrollable list of numbers, and the start and end dates with date fields. The colour of both the status buttons and rating numbers are adjusted based on the value selected for enhanced visual feedback. The save button is only tappable when a change is detected to avoid unnecessary API requests. Games can be removed from the users log with the button at the bottom of the page. If this button is pressed a confirmation dialog is shown to confirm games are not removed by mistake.

5.4.5 Discover Page

The discover page is the main way to find new games. It consists of a search bar at the top of the page, and two tabs of recommendations. Tapping the search bar navigates the user to the search page, following similar user experience (UX) patterns to applications such as Spotify (Spotify, undateda) or Instagram (Instagram, undated). A search request is triggered 300ms after the user stops typing to reduce unnecessary API calls. A flag ensures only one search runs at a time, preventing overlap. Results appear below the search bar, and tapping one navigates to its information page.

Recommendations are separated into two tabs, providing personalised suggestions for new releases and all games in the database. Games are displayed in a 3x5 grid format, and clicking on a game takes the user to the respective information page. If the recommendations are not yet available when requested, a loading symbol is displayed, and the request is retried every 2 seconds, ensuring content is delivered as soon as it is ready.

One issue with the current implementation is that recommendations are not reloaded when a user alters their ‘Quest Log’; instead, the suggestions are only refreshed when logging in. This should theoretically be a simple problem to fix, however a consistent trigger point for the reload could not be found during development. As a result, this is a limitation that would need to be addressed in future iterations.

Chapter 6: Results & Discussion

This chapter of the report presents the results of this project, demonstrating how the key functional requirements have been met. It comprises an overview of the final application, a summary of testing outcomes, a discussion of insights gathered from user interviews, and a UI design evaluation.

6.1 Final Application Overview

All figures in this section are screenshots taken of an emulated Google Pixel 8 Pro in Android Studio. The white bar at the bottom is a device feature, not part of the application. Red circles indicate taps, and red arrows show navigation flow. Figures 21 to 36 provide a comprehensive overview of the application's functionality.

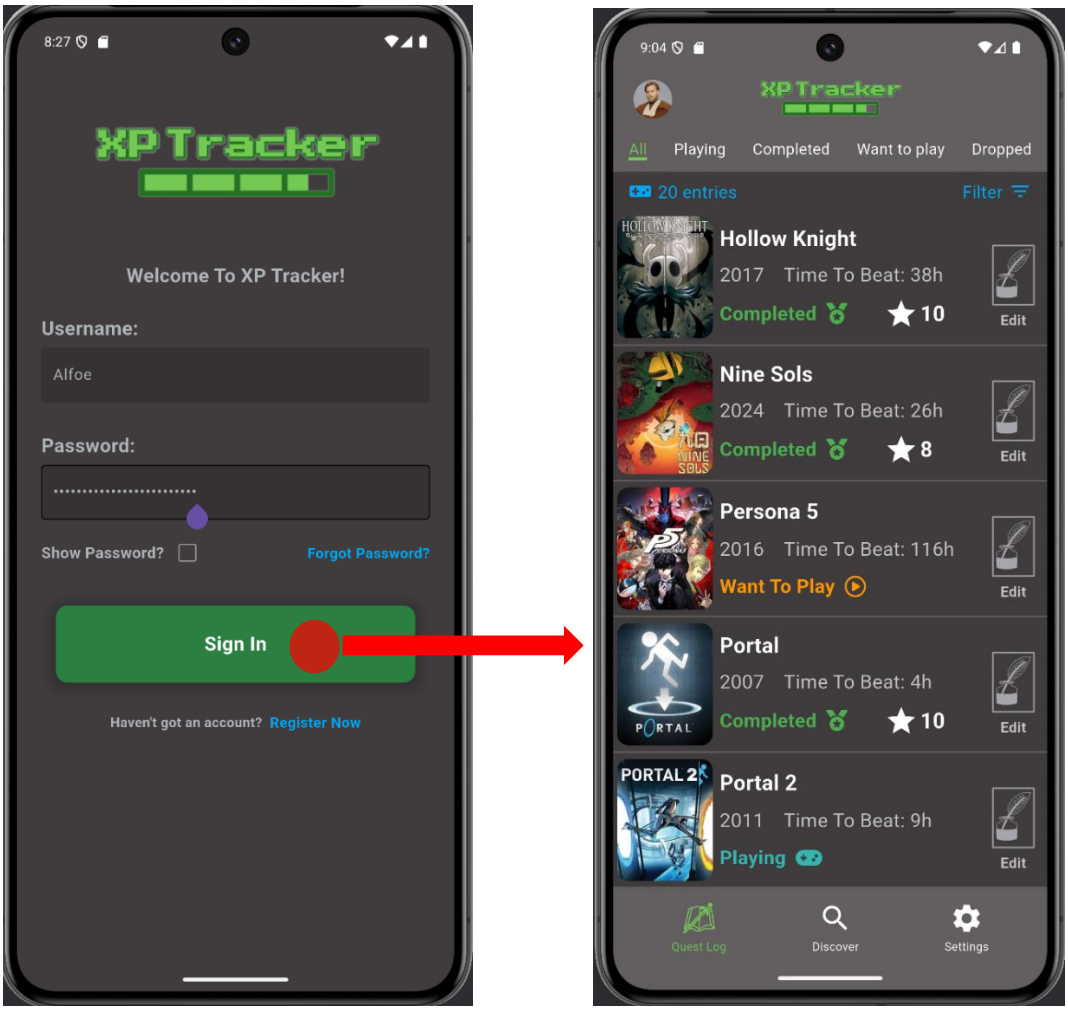


Figure 21: Navigation from the login screen to the home page (Requirement A.1: User accounts and logging in).

Figure 22: Methods to filter the Quest log (Requirement A.4: View and filter games).

Figure 22 demonstrates ways games can be filtered. By clicking on the options along the bar at the top, only games matching the selected status are displayed (22a). By tapping the blue filter button on the right, a drop down is displayed, allowing users to change the order their games are shown in (22b). Clicking an option when it is already selected reverses the order. The background is darkened for better clarity of the active section of the application, and clicking on the filter button again, or anywhere in the darkened area, closes the drop down.

Figure 23: Clicking on the entry count in the top left shows an overview of all games currently being displayed.

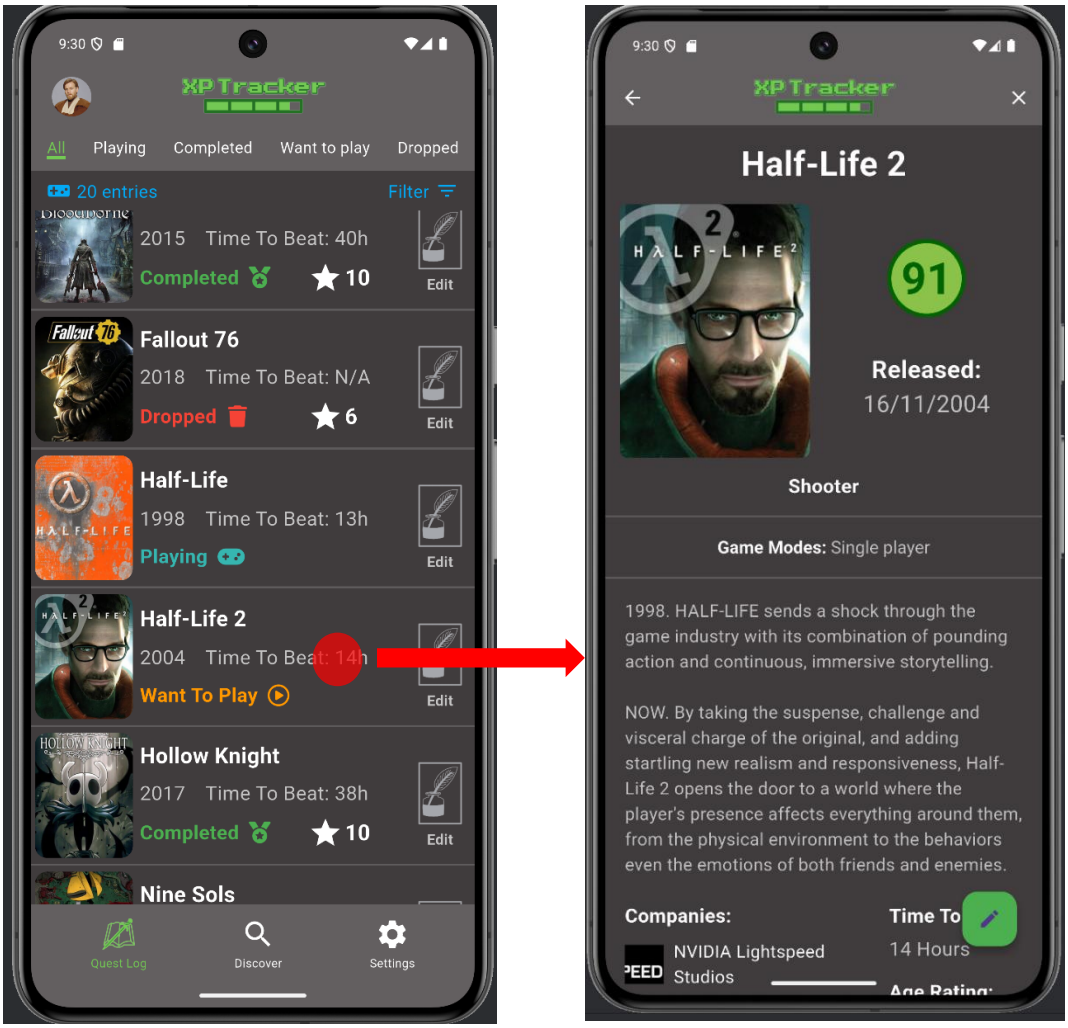


Figure 24: Navigating to the game information page by clicking on an entry (Requirement A.5: Detailed game information).

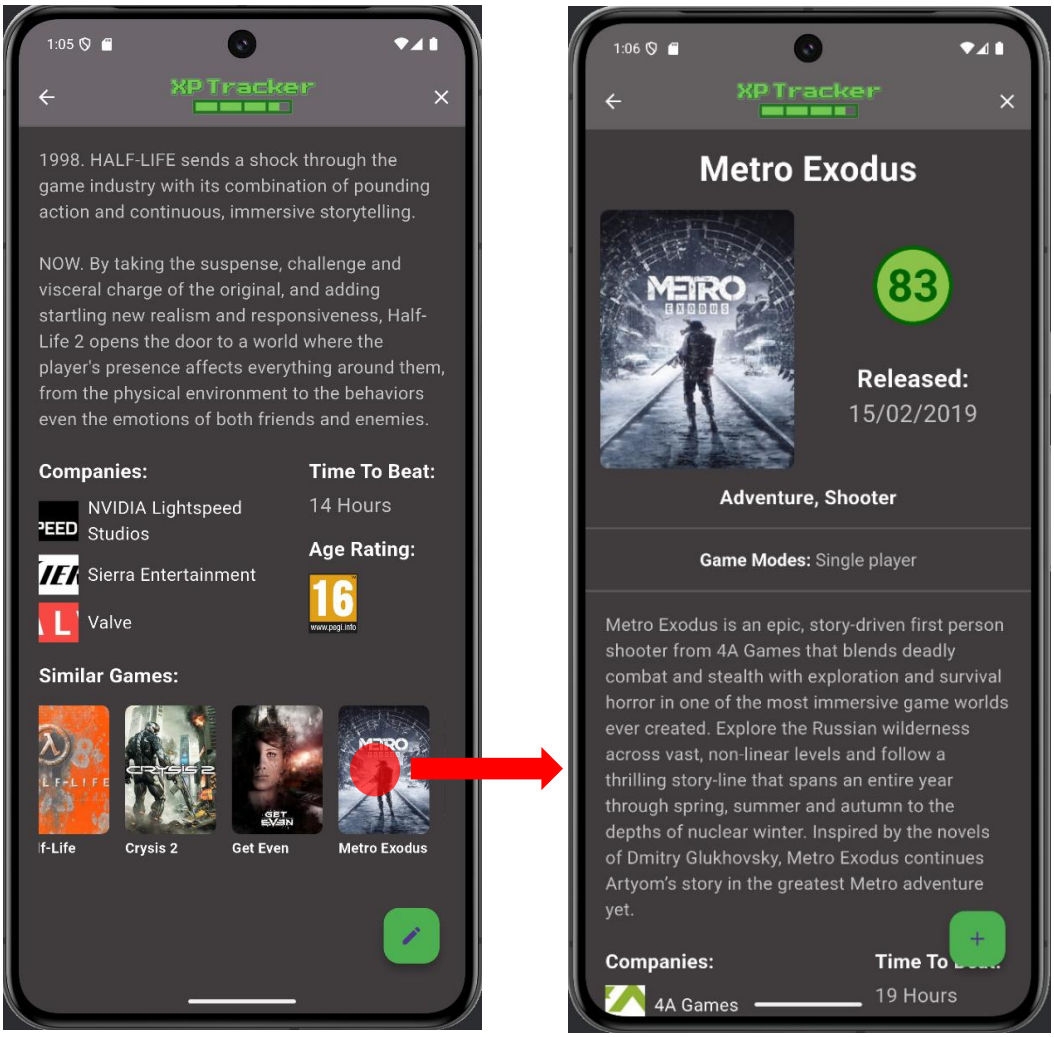


Figure 25: Navigating to a game from the Similar Games list.

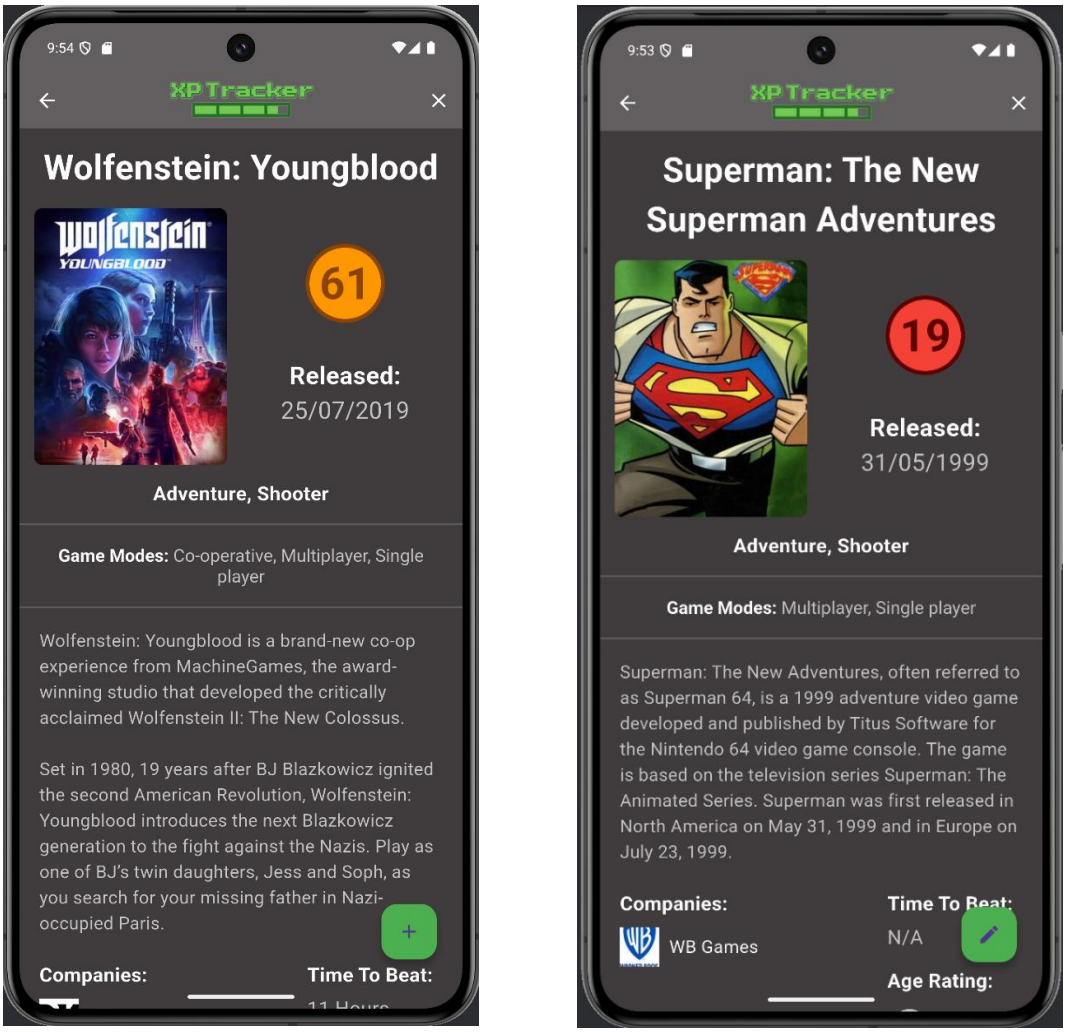


Figure 26: Two game information pages, showing how the application deals with different title lengths, and changes the colour of the user rating based on its value.

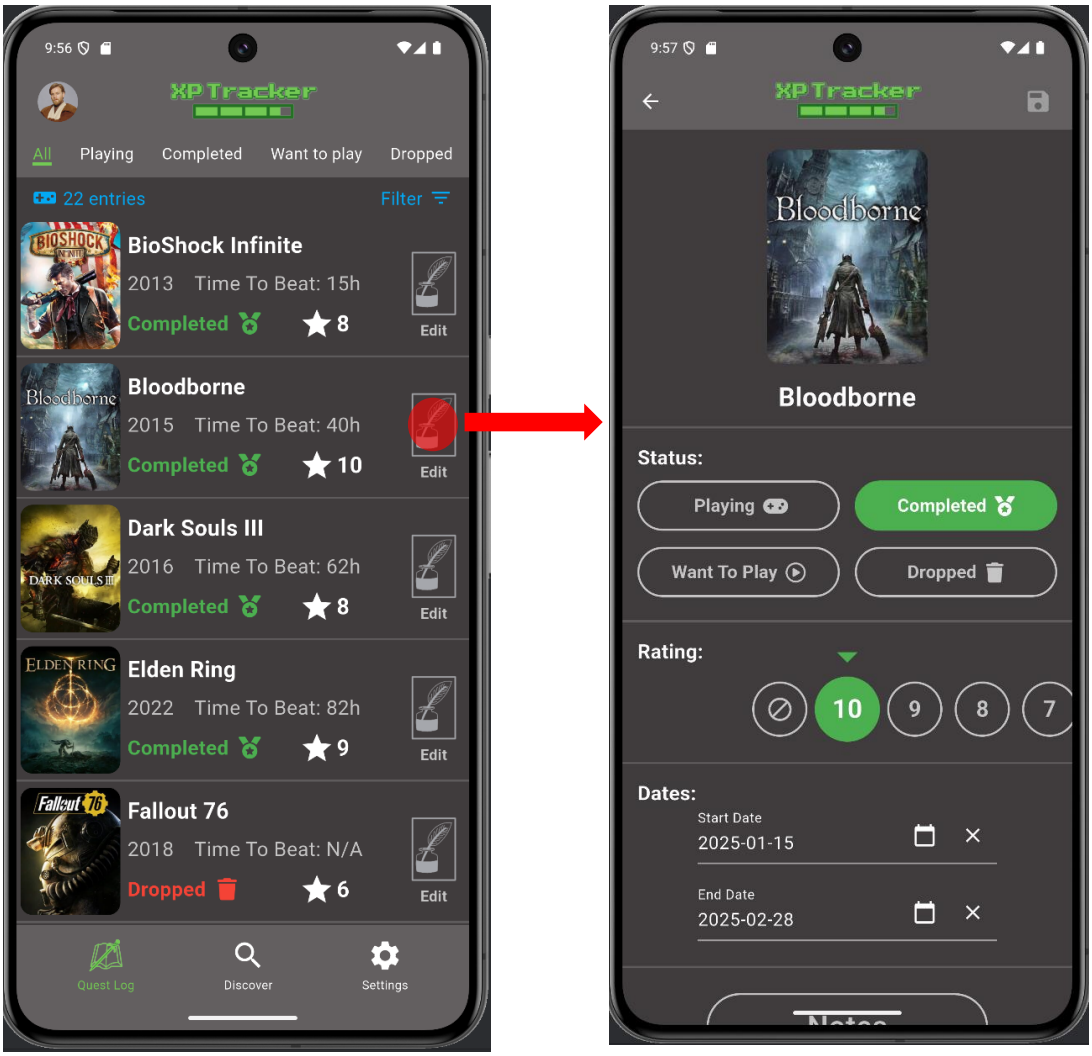


Figure 27: Navigating to the edit page for a game (Requirement A.2: Editing game details).

The edit entry page seen in Figure 27 can also be reached using the green floating action button in the bottom right corner of Figure 26. The pen symbol indicates editing an existing entry, while the plus is present when adding a new game. The save button is greyed out here because no changes have been detected on the page.

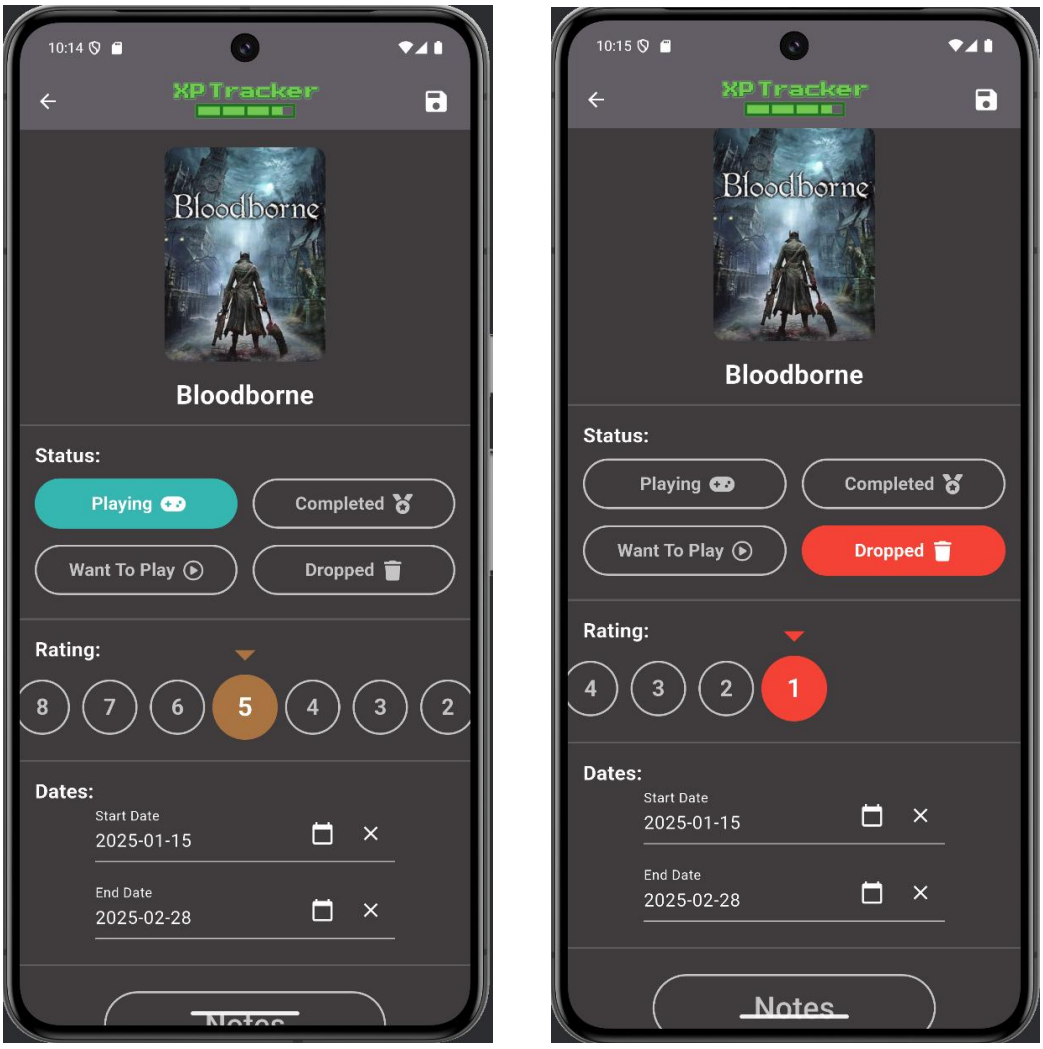


Figure 28: Colour changes for the different status or rating selected. The colour for the rating operates on a gradient from green to red (Requirement A.2.1: Status, rating and dates).

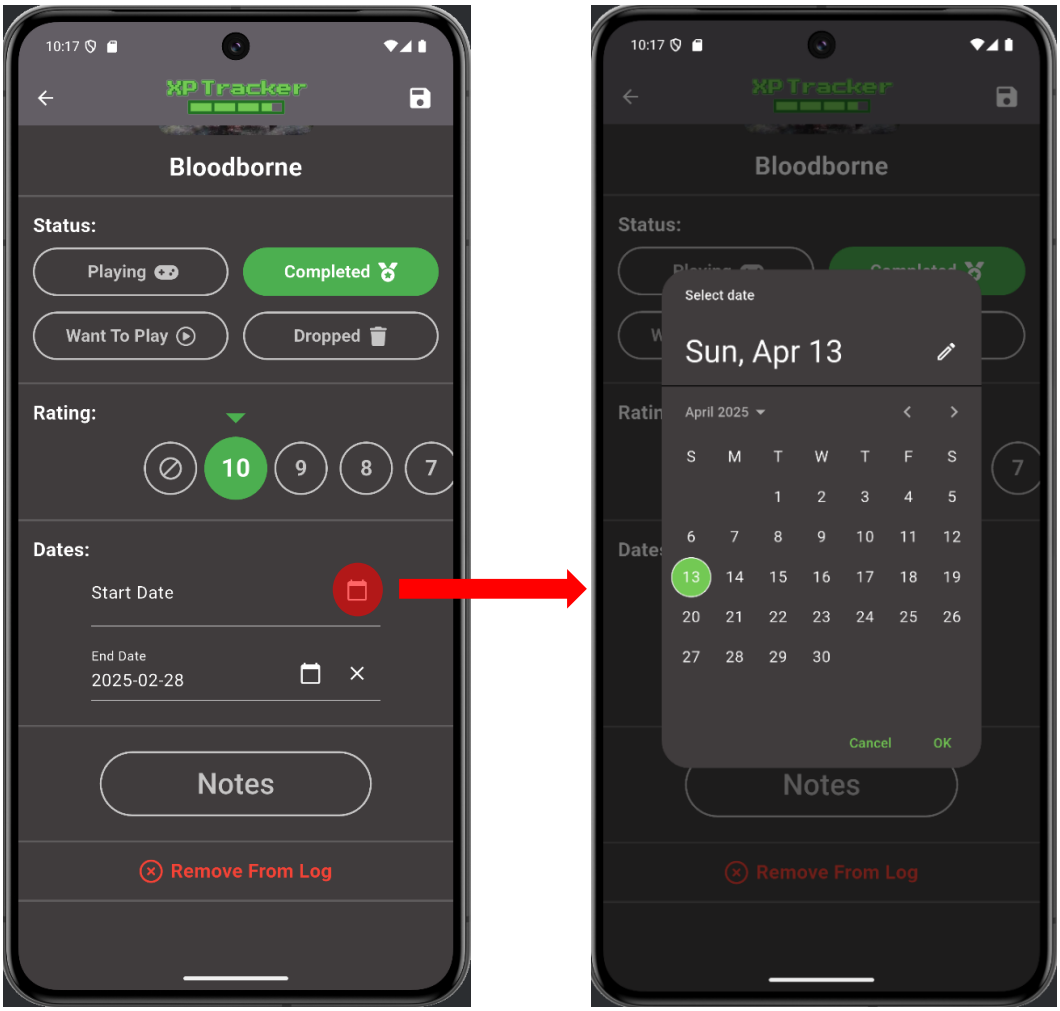


Figure 29: Date selection. Clicking the cross removes an existing date.

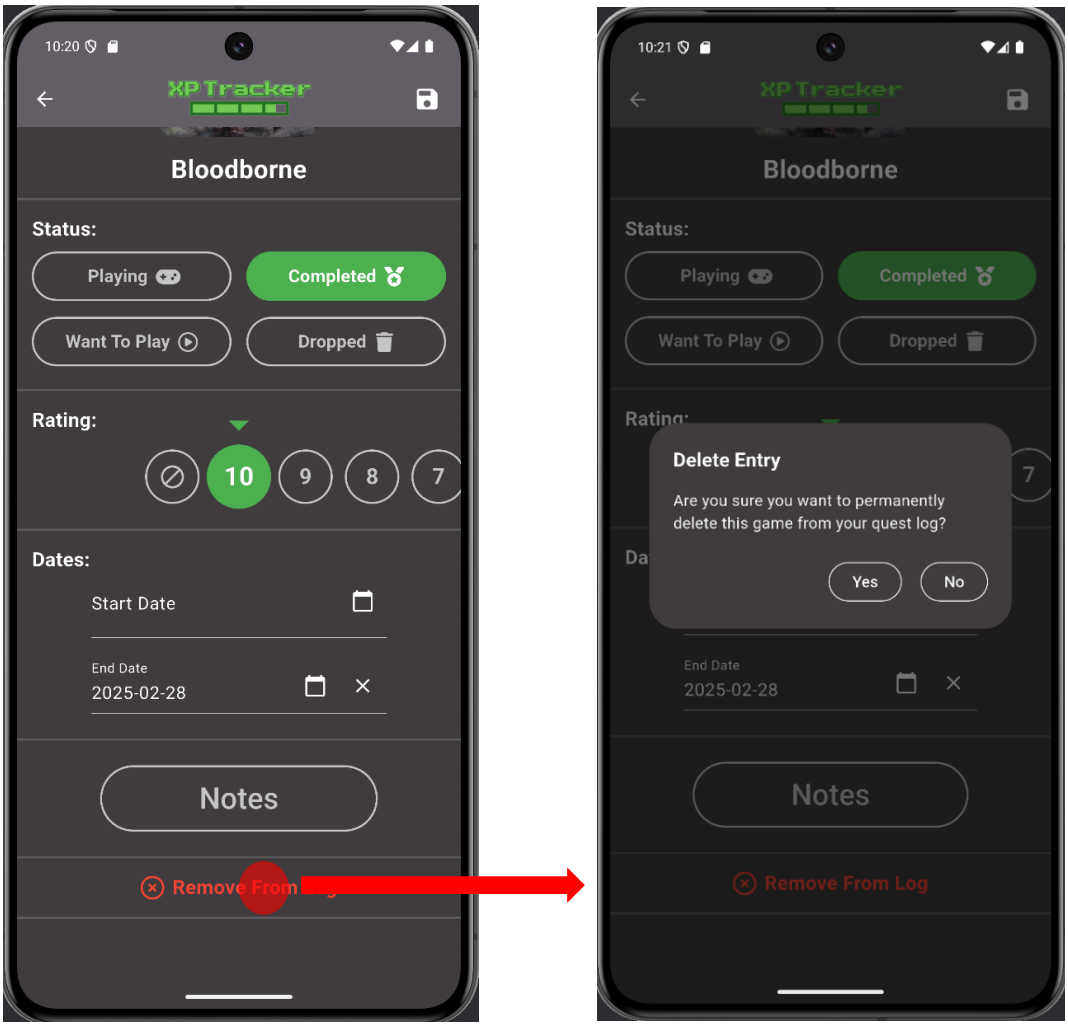


Figure 30: Removing a game from the 'Quest Log' and the confirmation dialog.

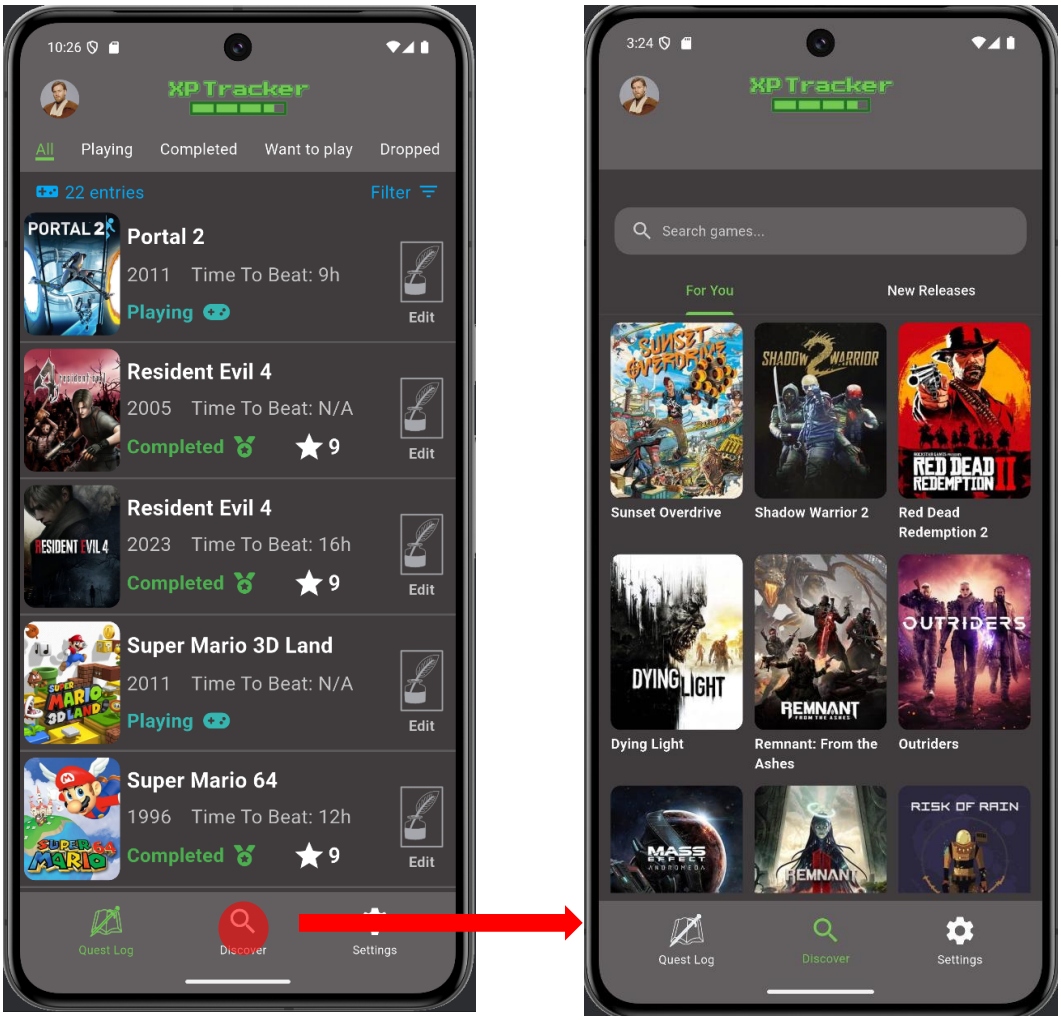


Figure 31: Navigating to the 'Discover' page. It can also be reached by swiping left anywhere on the 'Quest Log' page (Requirement A.5.1: Providing recommendations).

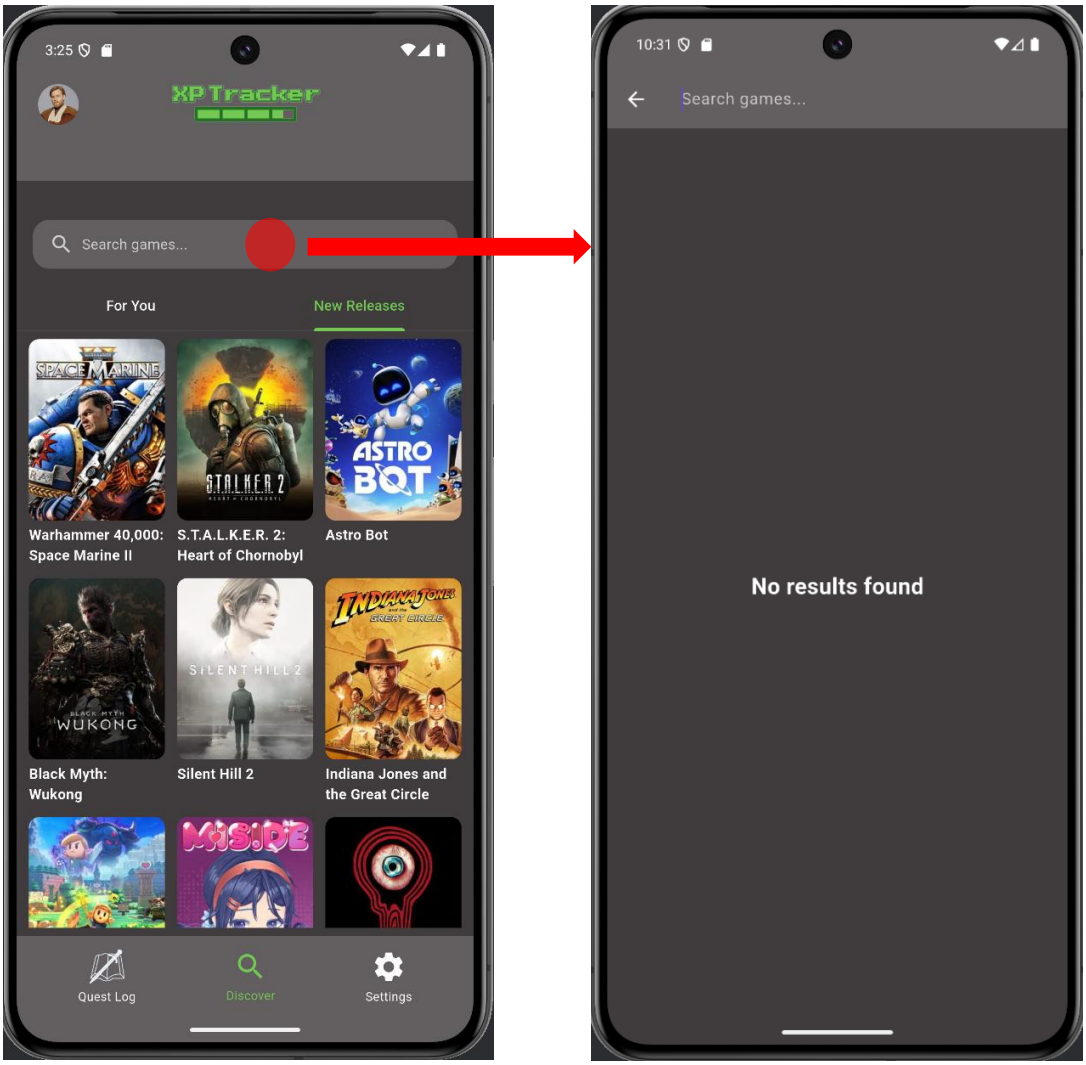


Figure 32: Navigating to the Search Page.

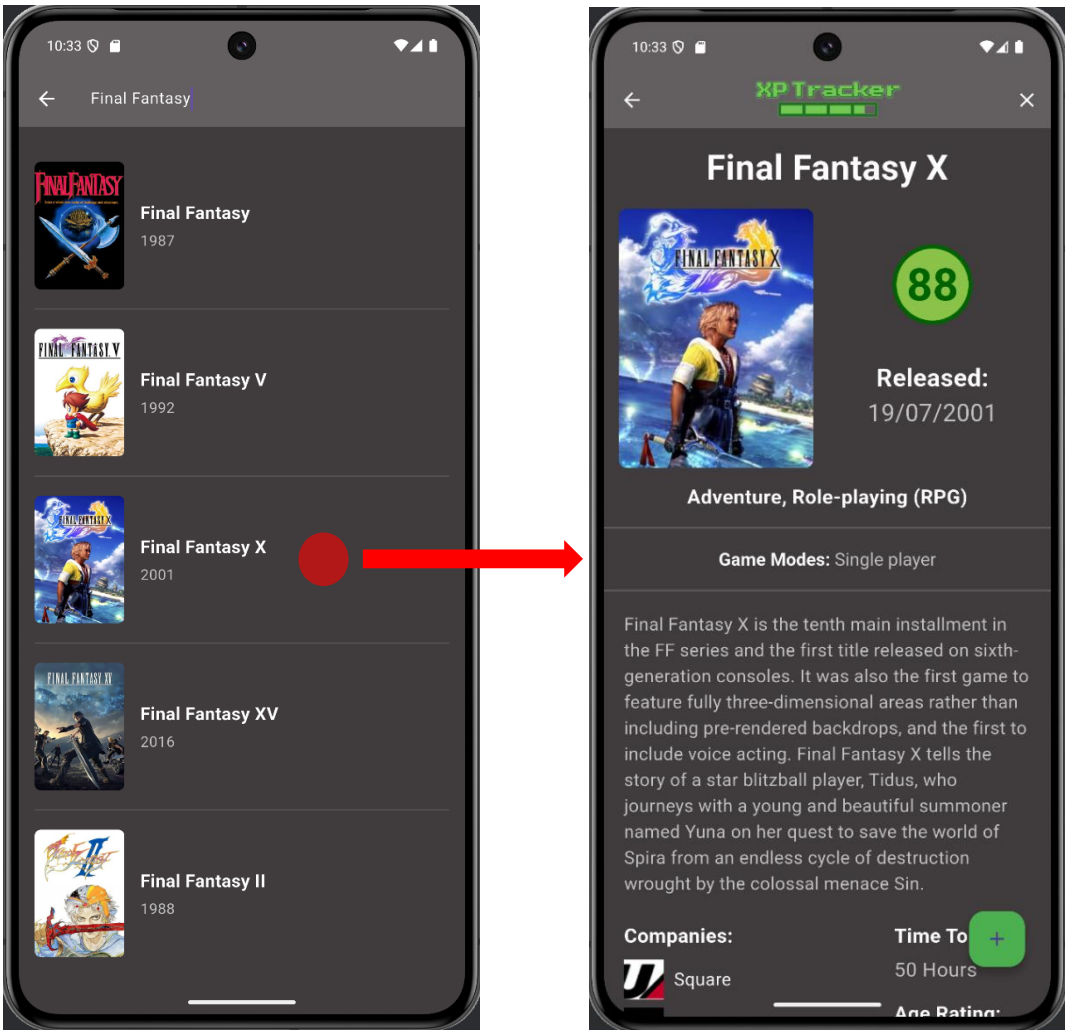


Figure 33: Search result selection (Requirement A.3: Searching for games).

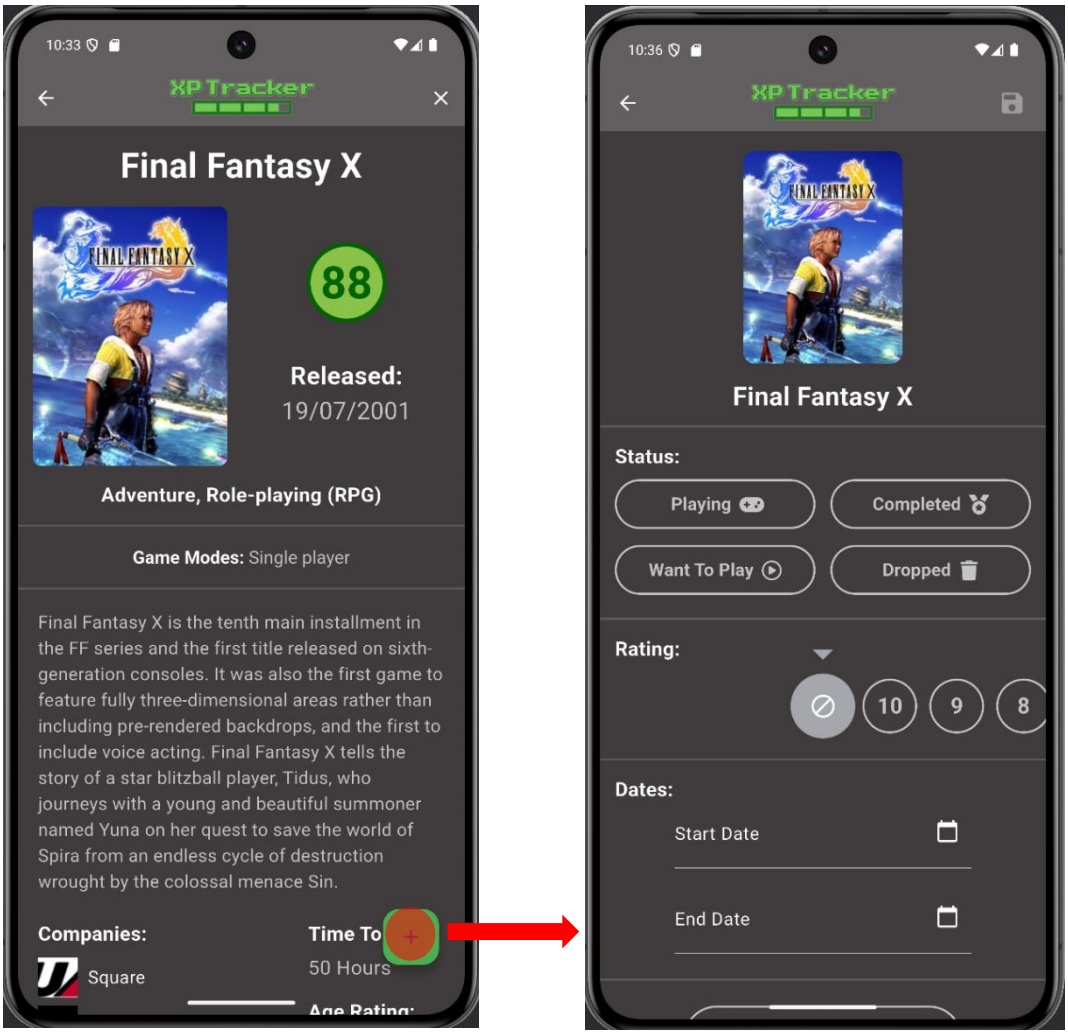


Figure 34: Adding the searched for game to the 'Quest Log'. Once the user has selected a status, the save button will become white, and the game can be saved to the log by pressing it.

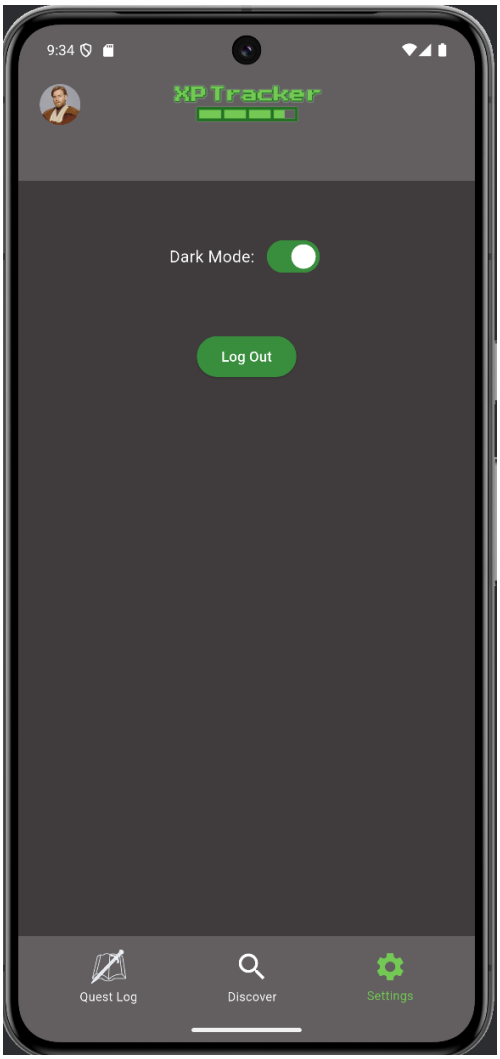


Figure 35: The settings page. The full page could not be finished within the designated time.

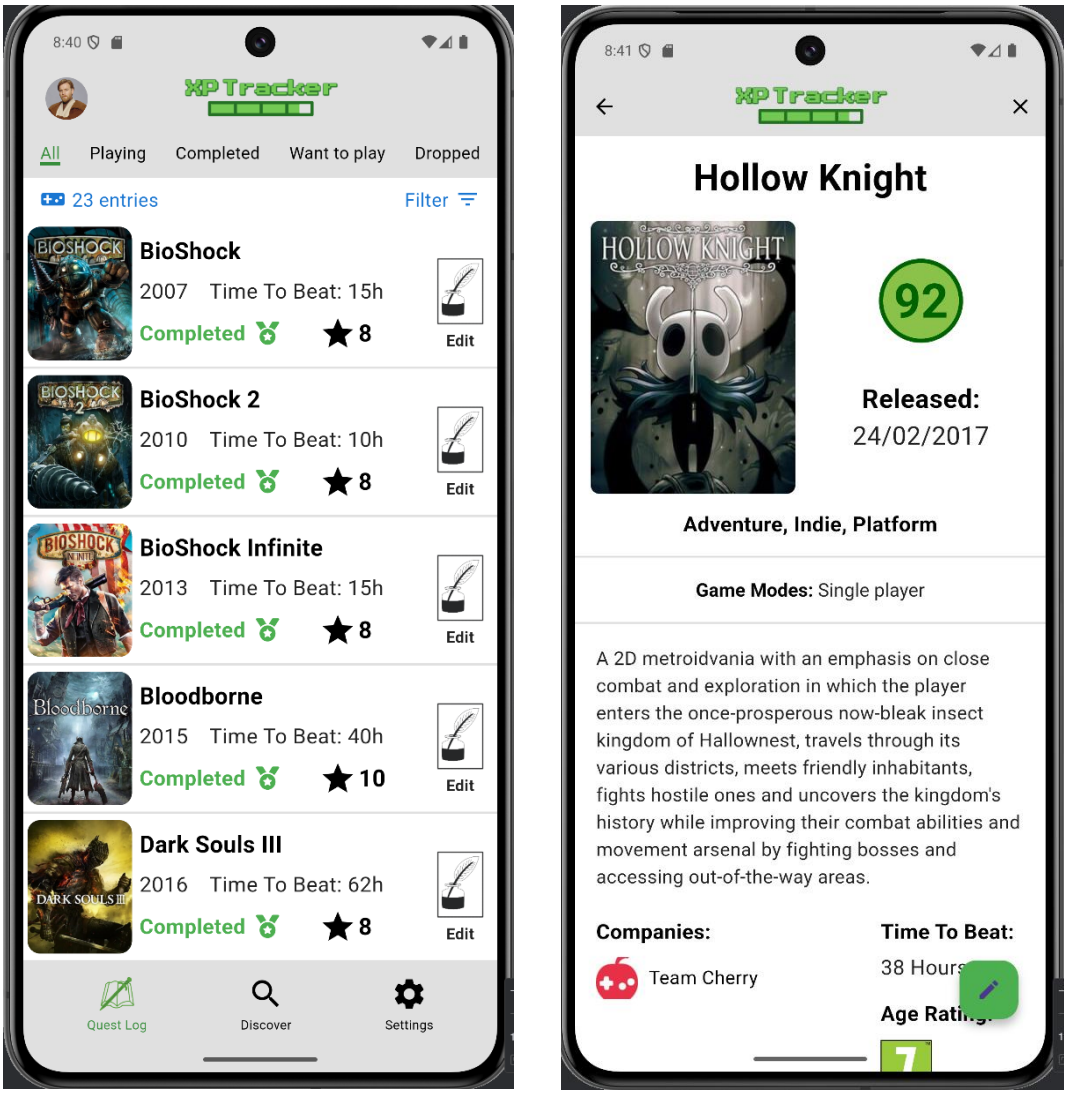


Figure 36: Light mode examples (Requirement D.2.1: Light and dark modes).

6.2 Evaluation of Application

During development, 3 specific aims were defined to evaluate the success of the application. Aim 1 focused on the creation of an application that allowed users to track the games that they play. The above figures demonstrate that this aim has been achieved, and all key functionality is operational. Not every feature outlined in the requirements has been implemented, most notably the friend system (Requirement

A.7). However, this was anticipated, as the requirements were intentionally designed with future development in mind. One limitation resulting from the absence of this feature is the inability to weight recommendation results based on friend ratings as initially proposed during the literature review. Features such as this were ultimately not implemented due to an underestimation of the project's scope at the proposal stage. Nevertheless, the core objectives required to satisfy Aim 1 have been successfully achieved and are fully functional within the final artefact.

The first aim would not have been possible without the creation of a complete database, as detailed by Aim 2. The database is the foundation of this project, and it provides essential functionality. It stores all the game data seen within the application alongside the procedures needed for a search engine and recommendation system, whilst adhering to best practices of database design. One area the database is flawed is in how up to date it is, since new game entries stopped around three months ago from the time of writing. This has been appropriate for development, but maintaining a complete database would require ongoing synchronisation with the IGDB API. Although this task is not technically challenging, it was deemed less important than other core features.

The Flask server complements the database, providing an efficient way for the front and backends to communicate as well as providing additional functionality such as data parsing and generating recommendations. If the database and Flask server were not responsive, this aim would be considered a failure, so Section 6.3, Testing, evaluates various performance metrics for the application.

Aim 3 focused on the design of the UI, and so discussion surrounding it is in Sections 6.4 and 6.5.

6.3 Testing

To ensure the application performed as expected and was responsive (Requirement B.2), testing was conducted on key components. These tests ensured the database was capable of scaling with multiple entries (Requirement B.1) and highlighted the strengths and weaknesses of the implementation. Outside of performance and accuracy, other areas of testing were also conducted, including API calls, edge case and device testing.

6.3.1 Search Engine Performance and Accuracy

The search engine is a large part of Aims 1 and 2 and directly pertains to Requirement A.3. To ascertain its performance, response times were recorded for 25 pieces of test data. It was able to accurately return almost every search term with a mean execution time of 269 ms (Table 3). Execution times varied substantially, ranging from 4 ms to 1080 ms. Faster results usually occurred when only a few games matched the search term (Figure 37a), while slower results were because of longer search terms that may have had multiple matches, such as '*The Legend of Zelda: Breath of the Wild*'.

Of the 25 search queries tested, only one, *'Minecraft'* did not return at the top of the results. This was because the game is separated by editions in the IGDB database, and there is no game entry of a matching name (Figure 37b). Despite this, even the slowest search times remained excellent given the database size, resulting in a responsive and smooth UX.

Search Term	Correct Game In Results? (Yes/No)	Game At Top of Results (Yes/ No)	Execution Time Measured By The Server (ms)
Dark Souls	Yes	Yes	171
Super Mario 64	Yes	Yes	309
Hollow Knight	Yes	Yes	204
Super Mario Odyssey	Yes	Yes	410
God of War Ragnarök	Yes	Yes	329
Red Dead Redemption 2	Yes	Yes	378
Bloodborne	Yes	Yes	4
The Legend of Zelda: Breath of the Wild	Yes	Yes	1080
Hades	Yes	Yes	12
Resident Evil Village	Yes	Yes	469
Final Fantasy VII Remake	Yes	Yes	601
Portal 2	Yes	Yes	126
Terraria	Yes	Yes	10
DOOM Eternal	Yes	Yes	202
It Takes Two	Yes	Yes	244
Dead Cells	Yes	Yes	162
Cyberpunk 2077	Yes	Yes	213
Minecraft	Yes	No	408
Stardew Valley	Yes	Yes	255
Valorant	Yes	Yes	2
Animal Crossing: New Horizons	Yes	Yes	501
Celeste	Yes	Yes	13
Donkey Kong Country	Yes	Yes	349
Elden Ring	Yes	Yes	193
Persona 5	Yes	Yes	80
		Average Execution Time	269

Table 3: Execution times for 25 different search terms.

a

gameID	game_name
6547	Bloodborne
23906	Bloodborne 2
144485	Bloodborne PSX
185013	Bloodborne Kart
NULL	NULL

b

gameID	game_name
100876	Minecraft Earth
93801	Minecraft Dungeons
164002	Minecraft: Legends
7449	Minecraft: Story Mode
110	Minecraft: Java Edition

Figure 37: Search results for 'Bloodborne' (b) and 'Minecraft' (c).

While the search engine performs well in most cases, it does have limitations. Table 4 illustrates that minor spelling mistakes and search terms with missing words do still return the correct game, however shorthand terms and significant misspellings can result in either poor matches or no results. Searches using abbreviations or acronyms also tend to fail.

Game Name	Search Term	Correct Game In Results? (Yes/No)	Game At Top of Results (Yes/ No)
The Legend of Zelda: Breath of the Wild	Zelda Breath Wild	Yes	No
Elden Ring	Eldan Ring	Yes	Yes
God of War Ragnarök	GOW Ragnrok	No	No
Hollow Knight	Holo Knight	Yes	Yes
Stardew Valley	Star Dew Valley	Yes	Yes
Stardew Valley	Stardew	Yes	Yes
Resident Evil Village	Resident Evil 8	No	No
Portal 2	Portal Two	Yes	No
The Witcher 3: Wild Hunt	Witcher Wild Hunt	Yes	Yes
Super Mario Odyssey	mario odyssey	Yes	No
Grand Theft Auto V	GTA Five	No	No
God of War Ragnarök	God War Ragnarok	Yes	Yes
Hades	Haades	No	No
Cyberpunk 2077	Cyber Punk 2077	Yes	Yes
Doom Eternal	Doom Eternall	Yes	Yes

Table 4: Search results for shorthand terms and misspelt games names.

Although more advanced implementations such as Elasticsearch (Elastic, undated) could be considered for future iterations, the current solution strikes a balance between accuracy and efficiency while remaining suitably lightweight for the application's scope.

6.3.2 Recommender Engine Performance and Accuracy

Aim 1 required a recommendation system to provide new game suggestions to users. Requirement A.5.1 stipulates that for this to be successful, recommendations must be generated within 5 seconds. Once recommendations have been generated, they are returned almost immediately, so this section will focus on the time taken for the background thread to generate recommendations.

Table 5 illustrates that the recommendation system performs better than designated in Requirement A.5.1, and as such can be deemed as successful. Testing was performed with a range of 15 to 40 games in the 'Quest Log', with a

mean total execution time of 3.7 s, well within the 5 s requirement. The shortest execution time for one set of recommendations was for recent releases on test 5 (1.49 s), where many recent releases were added to the ‘Quest Log’ to purposely reduce the candidate set. The application visually handles this appropriately using the dynamic grid, as shown in Figure 38.

Test Number	General Reccomendations Generation Time (s)	Recent Reccomendations Generation Time (s)	Total Time Taken (s)
1	2.04	1.86	3.9
2	2.04	1.79	3.83
3	2	1.85	3.85
4	1.87	1.94	3.81
5	1.9	1.49	3.39
6	1.81	1.75	3.56
7	2.03	1.91	3.94
8	1.86	1.84	3.7
9	1.9	1.85	3.75
10	1.78	1.81	3.59
Average Time Taken (s)	1.923	1.809	3.732

Table 5: Recommendation generation times.

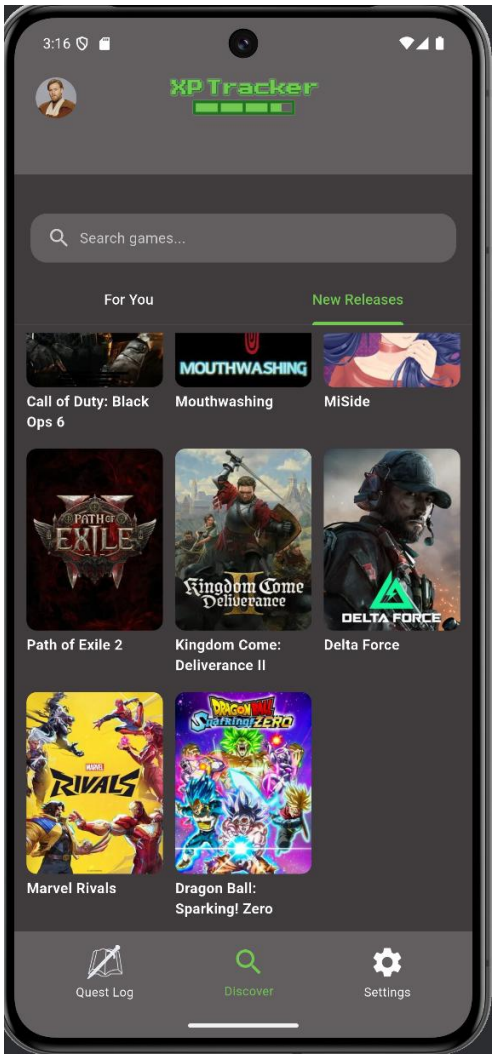


Figure 38: Dynamic grid that gracefully adjusts when fewer than 15 recommendations are available.

While formal metrics, such as click-through-rate, are hard to discern accurately without a large online testing group, extensive tests and sufficient funding (Silveira et al., 2017), it was observed that the genre and game mode of the games suggested closely align with existing games in the user's log. Alongside this, participants in the final interviews (see Section 6.4) were asked their opinion on the quality of the recommendations, with all of them responding that they felt logical. This constituted

the most appropriate offline evaluation method available under the circumstances (Silveira et al., 2017).

6.3.3 System Performance

As part of Aim 2 and Requirement C.1, communication between the front and backend should be handled by a RESTful API. The Flask server enables this, but it must also demonstrate acceptable responsiveness to fulfil Requirement B.2. To assess this, multiple API calls were timed under both local and remote conditions. Table 6 shows mean response time and variability.

a

API Call Type	Test 1 (ms)	Test 2 (ms)	Test 3 (ms)	Test 4 (ms)	Test 5 (ms)	Mean (ms)	Range (ms)
Login	485	426	449	491	411	452.4	80
Fetch User History	312	339	336	279	282	309.6	60
Add Game To History	286	301	275	249	252	272.6	52
Delete Game From History	253	215	250	216	260	238.8	45
Edit Game Status	256	278	220	312	243	261.8	92
Get Game Info	241	255	268	285	218	253.4	67
					Mean	298.1	66

b

Local Testing Response Time

API Call Type	Test 1 (ms)	Test 2 (ms)	Test 3 (ms)	Test 4 (ms)	Test 5 (ms)	Mean (ms)	Range (ms)
Login	232	273	230	226	240	240.2	47
Fetch User History	54	34	45	46	33	42.4	21
Add Game To History	31	36	34	25	24	30	12
Delete Game From History	24	27	23	19	26	23.8	8
Edit Game Status	43	23	38	23	21	29.6	22
Get Game Info	70	46	40	42	33	46.2	37
					Mean	68.7	24.5

Table 6: Response times for API calls under (a) remote and (b) local conditions.

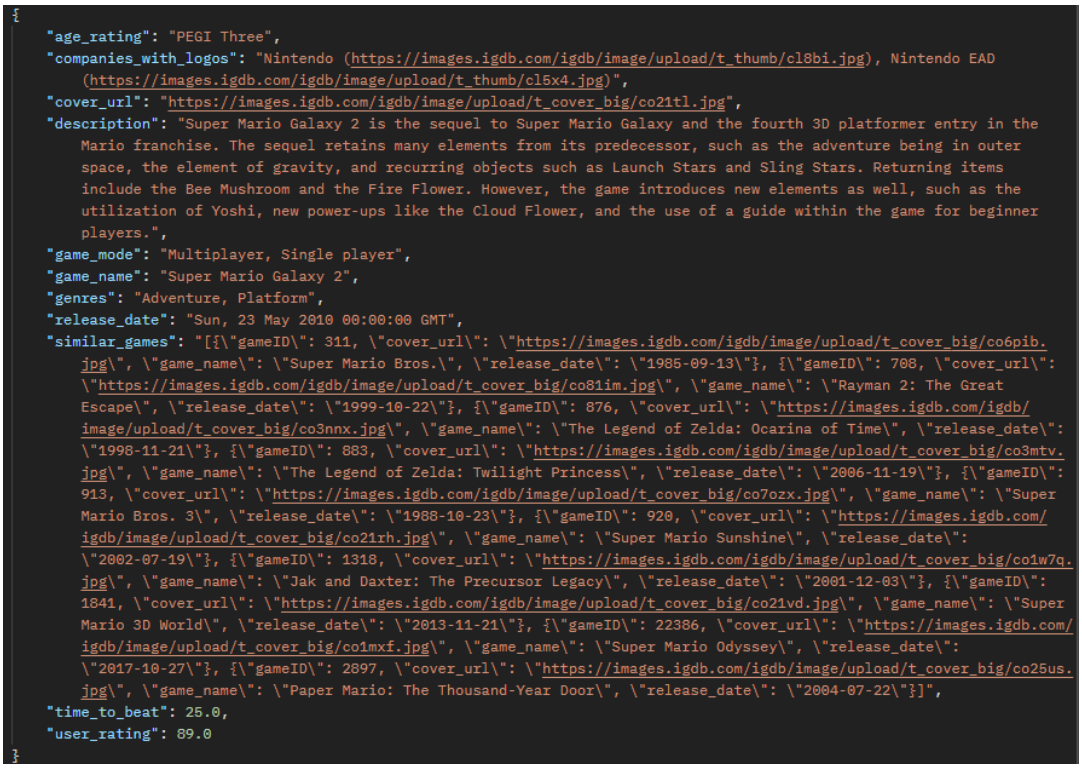
According to widely accepted usability standards, delays under 100 milliseconds are perceived as instantaneous, while delays under 1000 milliseconds are generally considered acceptable for maintaining user engagement (Nielsen, 1993). On the local network, the application provided an average response time within 68.7 ms, indicating near-instantaneous performance. Remotely, the response time increased

by a factor of around 4.3; however, the resulting 298.1 ms remained comfortably within acceptable bounds.

API response times are consistent, with the average variance under 100ms in both tests. Logging in takes noticeably longer than other calls, likely due to the cryptographic operations it employs, such as password hashing. Despite this, even this outlier remained well below the 1000 ms threshold, and overall, the system can be considered responsive and fit for purpose.

6.3.4 API Testing

Most API testing involved checking that requests returned the expected data using Postman. This tool was invaluable to this project, and reduced the time it took to identify issues. An example of how data was verified can be seen in Figure 39.



```
{
    "age_rating": "PEGI Three",
    "companies_with_logos": "Nintendo (https://images.igdb.com/igdb/image/upload/t_thumb/cl8bi.jpg), Nintendo EAD
        (https://images.igdb.com/igdb/image/upload/t_thumb/cl5x4.jpg)",
    "cover_url": "https://images.igdb.com/igdb/image/upload/t_cover_big/co21tl.jpg",
    "description": "Super Mario Galaxy 2 is the sequel to Super Mario Galaxy and the fourth 3D platformer entry in the
        Mario franchise. The sequel retains many elements from its predecessor, such as the adventure being in outer
        space, the element of gravity, and recurring objects such as Launch Stars and Sling Stars. Returning items
        include the Bee Mushroom and the Fire Flower. However, the game introduces new elements as well, such as the
        utilization of Yoshi, new power-ups like the Cloud Flower, and the use of a guide within the game for beginner
        players.",
    "game_mode": "Multiplayer, Single player",
    "game_name": "Super Mario Galaxy 2",
    "genres": "Adventure, Platform",
    "release_date": "Sun, 23 May 2010 00:00:00 GMT",
    "similar_games": "[{\"gameID\": 311, \"cover_url\": \"https://images.igdb.com/igdb/image/upload/t_cover_big/co6pib.
        jpg\", \"game_name\": \"Super Mario Bros.\", \"release_date\": \"1985-09-13\"}, {\"gameID\": 708, \"cover_url\":
        \"https://images.igdb.com/igdb/image/upload/t_cover_big/co81im.jpg\", \"game_name\": \"Rayman 2: The Great
        Escape\", \"release_date\": \"1999-10-22\"}, {\"gameID\": 876, \"cover_url\": \"https://images.igdb.com/igdb/
        image/upload/t_cover_big/co3nnx.jpg\", \"game_name\": \"The Legend of Zelda: Ocarina of Time\", \"release_date\":
        \"1998-11-21\"}, {\"gameID\": 883, \"cover_url\": \"https://images.igdb.com/igdb/image/upload/t_cover_big/co3mtv.
        jpg\", \"game_name\": \"The Legend of Zelda: Twilight Princess\", \"release_date\": \"2006-11-19\"}, {\"gameID\":
        913, \"cover_url\": \"https://images.igdb.com/igdb/image/upload/t_cover_big/co7ozx.jpg\", \"game_name\": \"Super
        Mario Bros. 3\", \"release_date\": \"1988-10-23\"}, {\"gameID\": 920, \"cover_url\": \"https://images.igdb.com/
        igdb/image/upload/t_cover_big/co21rh.jpg\", \"game_name\": \"Super Mario Sunshine\", \"release_date\":
        \"2002-07-19\"}, {\"gameID\": 1318, \"cover_url\": \"https://images.igdb.com/igdb/image/upload/t_cover_big/co1w7q.
        jpg\", \"game_name\": \"Jak and Daxter: The Precursor Legacy\", \"release_date\": \"2001-12-03\"}, {\"gameID\":
        1841, \"cover_url\": \"https://images.igdb.com/igdb/image/upload/t_cover_big/co21vd.jpg\", \"game_name\": \"Super
        Mario 3D World\", \"release_date\": \"2013-11-21\"}, {\"gameID\": 22386, \"cover_url\": \"https://images.igdb.com/
        igdb/image/upload/t_cover_big/co1mxf.jpg\", \"game_name\": \"Super Mario Odyssey\", \"release_date\":
        \"2017-10-27\"}, {\"gameID\": 2897, \"cover_url\": \"https://images.igdb.com/igdb/image/upload/t_cover_big/co25us.
        jpg\", \"game_name\": \"Paper Mario: The Thousand-Year Door\", \"release_date\": \"2004-07-22\"}]",
    "time_to_beat": 25.0,
    "user_rating": 89.0
}
```


Figure 39: Example of data verification in Postman.

6.3.5 Edge Case and Exception Handling

The primary focus of edge case testing was to ensure proper handling of missing data and correct overflow of lengthy text entries. Some data for certain games may be missing, so it was important that placeholders were implemented correctly. An example of this can be seen in Figure 40, where the game '*Ultrakill*' is missing an age rating, so a symbol is displayed in its place. Overflow working as intended can be seen in Figure 41, with the game '*Supersonic Acrobatic Rocket-Powered Battle-Cars*' having its title displayed gracefully.

Figure 40: Placeholder for missing age rating.

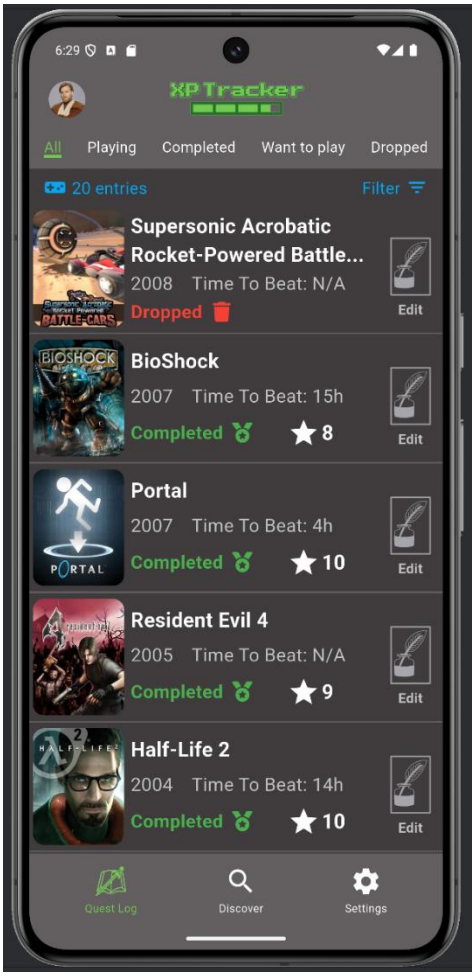


Figure 41: Long title being handled gracefully.

6.3.6 Device Testing

One notable limitation of this project was the lack of physical device testing. This was primarily due to limited access to a variety of mobile devices. No testing was carried out on IOS due to the required changes to the code base and a lack of access to the platform. IOS support (Requirement F.2) was always designated as a non-essential feature and was never considered vital to the project, hence its placement in Phase 3 of development.

The primary phone used for development was a 6.7-inch Pixel 8 Pro. A range of emulated Android devices were also tested on, but this occurred late in the development cycle, leading to minor issues surrounding scaling of text across a range of screen sizes, as seen in Figure 42. Although the application does function on Android devices, there is not true support for the entire platform yet. To fully meet Requirement F.1 and Aim 3's compatibility requirements, additional work would be needed to ensure proper scaling across a broader range of Android devices.

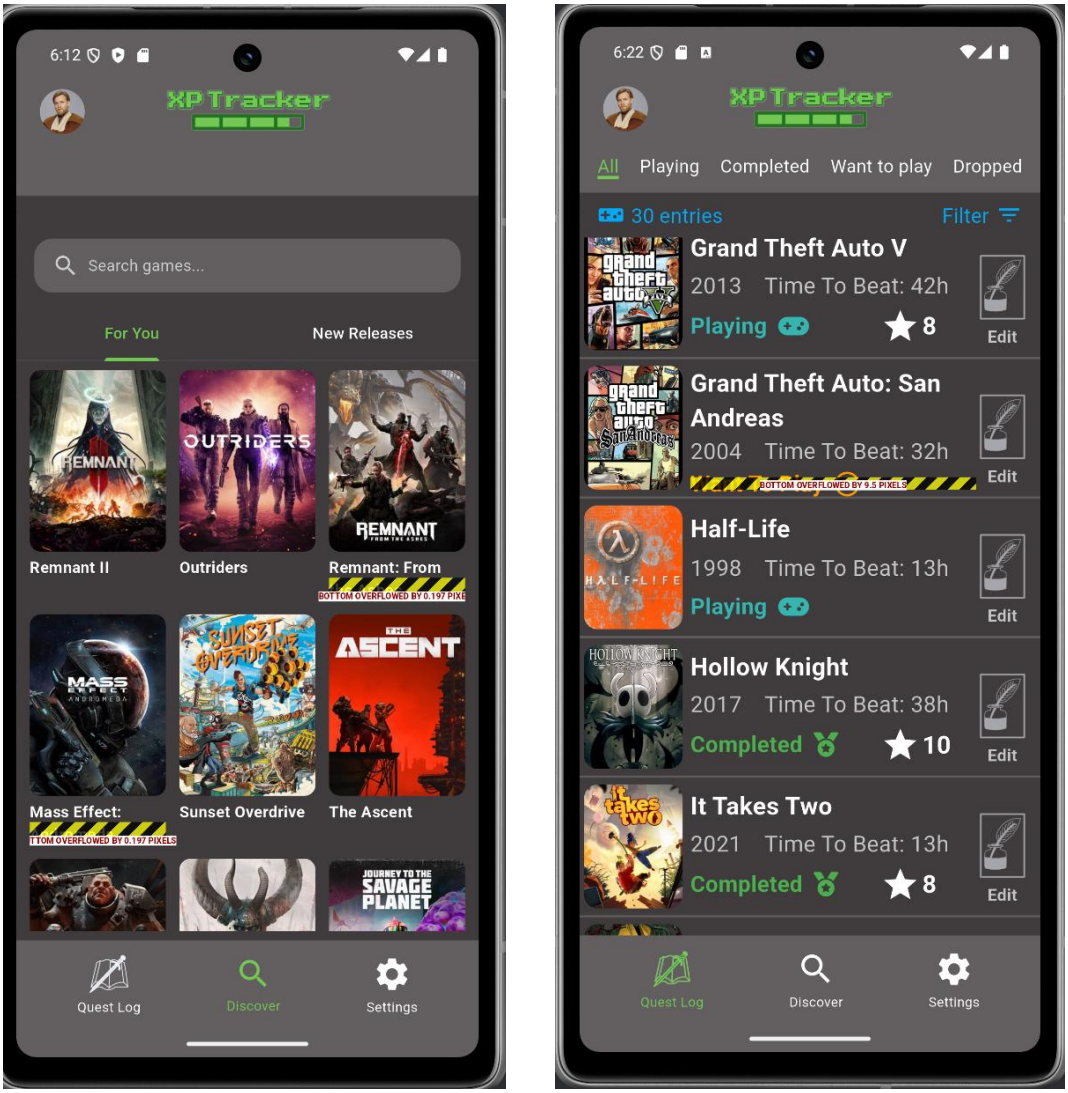


Figure 42: The two places in the application on a 6.1-inch Pixel 7a where there is a scaling issue (pixel overflow on titles across two lines).

6.3.7 Security Testing

Security was not a major concern during development, as the application was not intended for public release within the scope of the project. As such API requests were left unencrypted, and the security of data in transit was not a major consideration. All user data used during development was test data only, so no personal details were stored.

Despite that, password protection was still handled responsibly (Requirement E), and only salted and hashed passwords were stored in the database. To confirm this and ensure that plain text passwords were never stored, 5 test users were created with identical passwords: '*TestPassword123!'*. Figure 43 shows that none of the passwords appear in plaintext within the database. Furthermore, each stored password hash was unique, which verifies that salting was correctly applied.

personal_dataID	userID	email	password
17	17	test_email1@test.com	$2b$12$edJ/BtIcO6Yr4LyvUumbeeRJuKcVa4H6roA1SwSXS04yUQyOE2ZPK
18	18	test_email2@test.com	$2b$12$G4wtoKKBOtBoHenrFjnUT.oh6zlTfcEi/xmc1hQoK5phodwcIaNbK
19	19	test_email3@test.com	$2b$12$s2eLjje88fwTRTjmb4Nzxu1Lbdnad.4GCAATxkslYW8bI9L52Eup6
20	20	test_email4@test.com	$2b$12$gOYAdRk6uVNrjRNtik2GEe6Yek6gsubo9.dToG6MC299mQxNh0ewW
21	21	test_email5@test.com	$2b$12$m4JMeXJX5iupk6e3CMqe9e08zGMxUFW8oMQhQ.LHEuTIo9zIqdQiK

Figure 43: 5 Test users with identical passwords being stored salted and hashed.

6.4 UI Design Evaluation

Aim 3 of this project was to ensure that the application had an intuitive and accessible UI. As such Requirement D.1 stated that Nielsen's 10 Usability Heuristics (Nielsen, 2024) should be used to guide design decisions throughout development.

The application demonstrates alignment with almost all these heuristics. For example, the application provides the user with clear information about the status of the games in their log (Heuristic 1) and uses icons that will appear familiar to

users (Heuristic 2), such as the save icon, to clearly communicate functionality. Consistency and standards (Heuristic 4) are followed throughout, with the application handling navigation like other applications in the space and placing elements such as the search bar in a conventional location. The application also supports recognition over recall (Heuristic 6), maintaining a consistent layout and ensuring functionality is provided in logical locations. This is most apparent with the edit page being accessible from both the ‘Quest Log’ and game details page to ensure users can always access it. However, the application does not currently have any welcome guidance or tool tips for users and so would require improvements to its help and documentation (Heuristic 10) if it were to be released in the future.

Further insights into the usability of the interface were also gathered during user interviews (Section 6.5).

6.5 User Acceptance Testing

At the end of development, in-depth interviews were conducted (Appendix B) with 5 anonymous participants to assess user sentiment towards the application.

Users completed various tasks and rated their difficulty (Table 7), followed by answering several qualitative questions about the application.

Task Number	Task Summary	Participant 1	Participant 2	Participant 3	Participant 4	Participant 5	Task Mean
1	Navigate to Quest Log, Discover, Settings	5	5	5	5	5	5.00
2	Filter the games in Quest Log, both forwards and backwards	4	5	5	4	5	4.60
3	Group the games by status	4	5	4	5	3	4.20
4	Find out the average rating of all games in the Quest Log	3	4	5	3	3	3.60
5	Navigate to the edit page for a game entry in the Quest Log	5	5	5	5	5	5.00
6	Change the details for a game entry and save the change	5	4	4	4	4	4.20
7	Remove a game entry from the Quest Log	5	5	5	5	5	5.00
8	From the Quest Log page navigate to the details for a game entry	5	5	5	5	5	5.00
9	From the details page, navigate to the edit entry page	5	5	5	5	5	5.00
10	From the details page for a game, navigate to a similar game.	5	5	5	5	5	5.00
11	Add a similar game to the Quest Log	4	5	4	4	5	4.40
12	From the discover page, add a recommended game	4	5	5	4	5	4.60
13	Search for a game and add it to the Quest log	4	4	5	3	4	4.00
14	Change the application's theme (light/dark mode)	5	5	5	5	5	5.00
	Participant Mean	4.50	4.79	4.79	4.43	4.57	4.61

Key:
1 = Difficult
2 = Hard
3 = Average
4 = Simple
5 = Easy

Table 7: Task difficulty ratings.

The data illustrates that most users found the application easy to use, with the mean task rating being 4.61, and 7/14 tasks receiving a perfect score. This provides strong evidence that Aim 3 and Requirement D (intuitive and accessible UI) have been largely achieved, including Requirement D.2 (readable colour scheme), aside from the previously mentioned scaling issues.

Task 4 (finding the average rating for all games in the log) was rated most difficult. This was expected, as this feature is intended for power users, and accessing this information is not immediately obvious. Task 13 (search and add a game to the 'Quest Log') received the second lowest rating, with one user struggling with the search engine being sensitive to spelling mistakes. As covered in Section 6.3.1, this is a known weakness but was deemed as acceptable due to project scope.

The mean participant rating was 8.6/10. Participants praised the application's responsiveness, layout, level of information, recommendations, colour scheme,

theming and functionality. One participant commented that the presentation is professional, and it would be difficult to tell this application apart from a published one. This was the general sentiment received from all participants, with most issues being around incomplete features, rather than faults in existing ones. All participants stated they would use this application to track their game collection.

The interviews revealed a few design issues. One participant suggested implementing scroll bars to better indicate scrollable content, after initially missing that the edit and game details pages were scrollable. Another user found some buttons in the filter tab slightly too small and wanted a way to scale them up. The most prominent issue surrounded changing a game's rating, with 3 users wanting to tap the rating number to input it instead of swiping through them. This was an oversight during development and would need to be addressed in future.

Several additional features were suggested, however most of them were already considered for future development and are covered in Section 7.3.

Chapter 7: Conclusion

7.1 Project Conclusion

Aim 1 of this application was to create an application that allows users to track the video games that they play. The artefact meets every objective for this aim, and as such it can be considered successfully met. The results of this project support this conclusion, with 19/22 expected requirements fully implemented. Furthermore, performance testing shows the application is responsive, with API calls averaging 68.7 ms locally and 298 ms remotely, both comfortably within the usability thresholds outlined by Nielsen (1993).

Furthermore, Aim 2 was to create a complete and well-structured database that facilitated the user experience. This aim has been successfully met, with the minor exception that the database is not fully up to date with the latest game releases. The database is atomic, normalised, and includes multiple stored procedures to support efficient data operations. Performance was strong even with a large dataset; for example, search queries returned results in an average of 269 ms, with all exact matches producing the expected output and only one case where the game searched for was not ranked first. While accuracy for mistyped search terms was not perfect, this was considered acceptable within the scope of the project. Additionally, recommendation generation averaged 3.73 seconds, with all users reporting that the suggestions were relevant, and that the genres and game modes aligned well with entries in their 'Quest Log'.

Finally, Aim 3 focused on delivering an intuitive and accessible design. This aim has been achieved, with the application offering a clean, user-friendly interface shaped by user feedback. This is evidenced by the heuristic evaluation, which aligned with most of Nielsen’s usability principles (Nielsen, 2024), and by user feedback on the final artefact. The application received a mean user rating of 8.6/10,

and a mean task difficulty score of 4.61/5 (with 5 indicating ease), suggesting that users found the application intuitive and easy to use.

In conclusion, the final implementation is a holistic artefact that performs well, provides a high level of functionality, handles large amounts of data well and includes a well-designed, intuitive UI. It meets the vast majority of aims, objectives, and requirements that were set out, resulting in a robust and polished application.

7.2 Project Reflection

This project has been an insightful experience into the development process of an application and has involved a lot of learning about different methodologies, techniques and toolsets. It has been deeply rewarding and can be considered a success. Extensive research into similar platforms, UI principles, database design, recommender/search engines and API structure was conducted and underpinned both architectural and design choices. All key objectives within the aims have been achieved and risks mitigated successfully. Almost every expected requirement has been met, with a few minor issues. The methodology suited development well, allowing adjustments as issues were uncovered and survey feedback received, ensuring progress was timely and key features were delivered.

One of the key regrets for this project concerns the approach taken towards the development of the application. A lack of prior experience with Flutter meant that insufficient focus was placed on scaling the application appropriately to different screen sizes. Whilst scaling works well for widgets and the overall interface, text elements do not scale appropriately, leading to some overflow errors on smaller devices. This problem was not identified earlier in the development process, primarily due to a lack of access to a range of physical devices for testing. However, Flutter provides a comprehensive set of tools for implementing responsive and adaptive design. Should development of the application continue, addressing this

limitation would be prioritised to enhance usability and accessibility across all devices.

The largest unforeseen problem came when trying to ensure data consistency when a user made a change to an entry. This required additional API calls to properly manage, as well the restructuring of some internal logic, resulting in a larger portion of development time being needed to implement the edit page than expected. This issue highlighted the importance of considering the flow of data in more detail earlier in development, a lesson for future projects.

The main limitations of this project involve features not being implemented rather than issues with existing ones, outside of the scaling problem and recommendations not automatically refreshing when a user updates their ‘Quest Log’. Not every feature proposed was expected to be included so this is acceptable, and they could be implemented through further development.

7.3 Future Development

At this stage, the application is not yet ready to be published for open-source access due to several minor issues that still need to be addressed. To support widespread use, the MySQL database and Flask backend would need to be hosted on a cloud service so that full functionality is available for all end users. Prior to this, additional security measures must be implemented to safeguard both user data and the system’s integrity and a modest level of funding would be required to support hosting and deployment.

Moreover, the application would benefit from several additional features, which would enhance its usability and bring it closer to a complete state. Once these improvements were made, the application could be confidently released. Planned features include:

- Dynamic scaling
- In-app user creation and password recovery
- Keeping the local database in sync with the IGDB database
- Automatically refreshing recommendations
- A friend system and recommendation weighting based on reviews of friends' tastes
- Expanded settings page, including completed accessibility options
- Full length review support
- Note taking
- Introductory guidance and tool tips

The absence of these features does not diminish the project's success. Instead, it establishes a clear roadmap for future development, providing a strong foundation for a more polished and complete product which could be released as an open-source application.

References

Agile Business Consortium (2022). *Chapter 10: MoSCoW Prioririsation*. [online] www.agilebusiness.org. Available from: https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html [Accessed 1 May 2025].

Aldayel, A. and Alnafjan, K. (2017). Challenges and Best Practices for Mobile Application Development. *Proceedings of the International Conference on Compute and Data Analysis - ICCDA '17*, pp.41–48. Available from: https://doi.org/10.1145/3093241.3093245.

Android Studio (undated). *Android Studio and SDK tools*. [online] Android Developers. Available from: https://developer.android.com/studio [Accessed 17 Mar. 2025].

Atlassian (undated). *Jira*. [online] Atlassian. Available from: https://www.atlassian.com/software/jira [Accessed 21 Mar. 2025].

Brakefield, S. (undated). *Infinite Painter - A powerful sketching, painting and illustration app.* [online] Infinitestudio.art. Available from: https://www.infinitestudio.art/painter.php [Accessed 24 Mar. 2025].

Brown, L. (2025). *Agile Vs. Iterative: Key Differences Explained*. [online] Invensis Learning Blog. Available from: https://www.invensislearning.com/blog/agile-vs-iterative-model/ [Accessed 1 May 2025].

Chan, J., Chung, R. and Huang, J. (2019). *Python API development fundamentals: develop a full-stack web application with Python and Flask*. Birmingham: Packt Publishing Ltd.

Christofferson, A., Videbaek, A., Egan, A., Rowland, T. and Madden, M. (2024) *Gaming Report 2024 - Gamer Survey: Young Players Reshape the Industry*. [online] Bain & Company. Available from: https://www.bain.com/globalassets/noindex/2024/bain-report_gaming-report-2024.pdf [Accessed 22 Mar. 2025].

Elastic (undated). *Elasticsearch: The Official Distributed Search & Analytics Engine*. [online] Elastic. Available from: https://www.elastic.co/elasticsearch [Accessed 9 Apr. 2025].

Entertainment Software Association (2024) *2024 Essential Facts About the U.S. Video Game Industry* Available from https://www.theesa.com/resources/essential-facts-about-the-us-video-game-industry/2024-data [accessed 23 January 2025].

Feng, J., Xia, Z., Feng, X. and Peng, J. (2021). RBPR: A hybrid model for the new user cold start problem in recommender systems. *Knowledge-Based Systems*, 214. Available from: https://doi.org/10.1016/j.knosys.2020.106732

Flask (undated). *Welcome to Flask — Flask Documentation (3.0.x)*. [online] Palletsprojects.com. Available from: https://flask.palletsprojects.com/en/stable/ [Accessed 17 Mar. 2025].

Flutter (undateda). *Flutter - Beautiful native apps in record time*. [online] Flutter.dev. Available from: https://flutter.dev [Accessed 17 Mar. 2025].

Flutter (undatedb). *Hero animations*. [online] docs.flutter.dev. Available from: https://docs.flutter.dev/ui/animations/hero-animations [Accessed 13 Apr. 2025].

Fortune Business Insights (2025) *Smartphone Market Size, Share & COVID-19 Impact Analysis, By Operating System (Android, iOS, Windows and Others), By Distribution Channel (OEMs Stores, Retailer and E-commerce) and Regional Forecast, 2022-2029.* [online] Fortune Business Insights. Available from https://www.fortunebusinessinsights.com/industry-reports/smartphone-market-100308 [accessed 23 January 2025].

Fox, L. (2023). *Why Should You Use Flask Framework For Web Development?* [online] Medium. Available from: https://medium.com/@lauren-fox/why-should-you-use-flask-framework-for-web-development-f5a7233e17a6 [Accessed 17 Mar. 2025].

fza, zhil and afoeder (2015). *Levenshtein distance function for Doctrine and MySQL*. [online] github. Available from: https://github.com/fza/mysql-doctrine-levenshtein-function [Accessed 21 Mar. 2025].

GAMEYE (undated). *Homepage | GAMEYE*. [online] GAMEYE. Available from: https://www.gameye.app [Accessed 27 Mar. 2025].

Google (undated). *Google Open Source*. [online] opensource.google. Available from: https://opensource.google [Accessed 3 Apr. 2025].

Gupta, L. (2025). *What is REST – Learn to create timeless REST APIs*. [online] Restfulapi.net. Available from: https://restfulapi.net/ [Accessed 2 May 2025].

Harvard Business Review Analytic Services (2015). *Agile practice: The competitive advantage for a digital age - Sponsored by Atlassian*. [online] Harvard Business School Publishing. Available from: https://www.atlassian.com/dam/jcr:d9ce12ec-c1a1-4495-b7dc-16c38d6b594e/Atlassian_Harvard_Business_Review_Agile_Report.pdf [Accessed 1 May 2025].

IGDB (undateda). *IGDB API docs*. [online] IGDB.com. Available from: https://api-docs.igdb.com [Accessed 17 Mar. 2025].

IGDB (undatedb). *IGDB: Video Game Database API*. [online] IGDB.com. Available from: https://www.igdb.com/api [Accessed 17 Mar. 2025].

Instagram (undated). *Instagram*. [online] Instagram. Available from: https://www.instagram.com [Accessed 13 Apr. 2025].

Javed, U., Shaukat, K., A. Hameed, I., Iqbal, F., Mahboob Alam, T. and Luo, S. (2021) A Review of Content-Based and Context-Based Recommendation Systems. *International Journal of Emerging Technologies in Learning (iJET)*, 16(03), pp. 274–306. Available from: https://doi.org/10.3991/ijet.v16i03.18851

Jo, H., Wang, J., Lee, J.Y., Shin, Y., Jeong, Y. and Kwang Lee, J. (2024). Analyzing Consumer Adoption in Subscription Services: Perceived Benefits, Sacrifices, and Innovativeness. *Journal of KIIT*, 22(8), pp. 165–176. Available from: https://doi.org/10.14801/jkiit.2024.22.8.165.

Karasavvas, T. 'Theo' (2022). *Why Flutter is the most popular cross-platform mobile SDK*. [online] Stack Overflow Blog. Available from: https://stackoverflow.blog/2022/02/21/why-flutter-is-the-most-popular-cross-platform-mobile-sdk [Accessed 17 Mar. 2025].

Kwakernaak, M. (2019). Making Products That Matter: How Value-driven Development Accelerates Your Teams. *Journal of Creating Value*, 5(2), pp.150–163. Available from: https://doi.org/10.1177/2394964319868366

Lanter, D.P. and Essinger, R. (2017). User-Centered Design. *The International Encyclopedia of Geography.* Available from: https://doi.org/10.1002/9781118786352.wbieg0432

Lazarus Alliance (2024). *The Role of Open Source Software in Cybersecurity: Benefits, Challenges, and Key Tools*. [online] Lazarusalliance.com. Available from: https://lazarusalliance.com/the-role-of-open-source-software-in-cybersecurity-benefits-challenges-and-key-tools [Accessed 3 Apr. 2025].

Melville, P. and Sindhwani, V. (2010) Recommender Systems. *Encyclopedia of machine learning*, 1 829–838. Available from https://www.vikas.sindhwani.org/recommender.pdf [Accessed 15 Mar. 2025].

Microsoft (undated). *Featured Projects*. [online] Microsoft Open Source. Available from: https://opensource.microsoft.com/projects [Accessed 3 Apr. 2025].

MobyGames (undated). *Video Games Database. Credits, Trivia, Reviews, Box Covers, Screenshots - MobyGames*. [online] MobyGames. Available from: https://www.mobygames.com [Accessed 17 Mar. 2025].

Mohammadi, V., Rahmani, A.M., Darwesh, A.M. and Sahafi, A. (2019) Trust-based recommendation systems in Internet of Things: a systematic literature review. *Human-centric Computing and Information Sciences*, 9(21). Available from https://doi.org/10.1186/s13673-019-0183-8

MyAnimeList (undated). *Forums- MyAnimeList*. [online] Myanimelist.net. Available from: https://myanimelist.net/forum [Accessed 27 Mar. 2025].

MySQL (undateda). *MySQL*. [online] Mysql.com. Available from: https://www.mysql.com [Accessed 17 Mar. 2025].

MySQL (undatedb). *MySQL :: MySQL 8.4 Reference Manual :: 14.9 Full-Text Search Functions*. [online] dev.mysql.com. Available from: https://dev.mysql.com/doc/refman/8.4/en/fulltext-search.html [Accessed 8 Apr. 2025].

MySQL (undatedc). *MySQL Connector/Python Developer Guide*. [online] dev.mysql.com. Available from: https://dev.mysql.com/doc/connector-python/en [Accessed 17 Mar. 2025].

Nam, E. (2019). *Understanding the Levenshtein Distance Equation for Beginners*. [online] Medium. Available from: https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0 [Accessed 9 Apr. 2025].

Nielsen, J. (1993). *Response Times: The 3 Important Limits*. [online] Nielsen Norman Group. Available from: https://www.nngroup.com/articles/response-times-3-important-limits/ [Accessed 20 Apr. 2025].

Nielsen, J. (2024) *10 Heuristics for User Interface Design.* [online] Nielsen Norman Group. Available from https://www.nngroup.com/articles/ten-usability-heuristics [accessed 26 January 2025].

Nintendo (undated). *The official home for The Legend of Zelda - Home*. [online] The Official home for The Legend of Zelda - Home. Available from: https://zelda.nintendo.com [Accessed 24 Mar. 2025].

Ozsoy, M.G. and Polat, F. (2013) Trust based recommendation systems. *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp.1267-1274. Available from: https://doi.org/10.1145/2492517.2500276

Postman (undated). *Postman | The Collaboration Platform for API Development*. [online] Postman. Available from: https://www.postman.com [Accessed 17 Mar. 2025].

Python Cryptographic Authority (undated). *bcrypt: Modern password hashing for your software and your servers*. [online] PyPI. Available from: https://pypi.org/project/bcrypt/ [Accessed 13 Apr. 2025].

Rabiu, I., Salim, N., Da'u, A. and Osman, A. (2020) Recommender System Based on Temporal Models: A Systematic Review. *Applied Sciences*, 10(7) 2204. Available from: https://doi.org/10.3390/app10072204.

Randell, S., Kester, A., German, J.D. and Janice, M. (2024) The need for individualization: An open innovation perspective on the case for customized products. *Acta Psychologica*, 249. Available from: https://doi.org/10.1016/j.actpsy.2024.104473.

reaperhulk (undated). *bcrypt: Modern password hashing for your software and your servers*. [online] PyPI. Available from: https://pypi.org/project/bcrypt/ [Accessed 13 Apr. 2025].

Ruiz, J., Serral, E. and Snoeck, M. (2020) Unifying Functional User Interface Design Principles. *International Journal of Human–Computer Interaction*, 37(1) 47-67. Available from: https://doi.org/10.1080/10447318.2020.1805876

Sahu, M. and Saritha, K. (2021) Study on Various Collabrative Filtering Techniques to Recommend Movies. *2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT)*, pp. 414-420. Available from: https://doi.org/10.1109/CSNT51715.2021.9509623

scikit-learn (n.d.). *1.6. Nearest Neighbors — scikit-learn 0.21.3 documentation*. [online] Scikit-learn.org. Available from: https://scikit-learn.org/stable/modules/neighbors.html [Accessed 23 Apr. 2025].

Seymour, T., Frantsvog, D. and Kumar, S. (2011). History Of Search Engines. *International Journal of Management & Information Systems (IJMIS)*, 15(4), p.47. Available from: https://doi.org/10.19030/ijmis.v15i4.5799.

Silveira, T., Zhang, M., Lin, X., Liu, Y. and Ma, S. (2017). How good your recommender system is? A survey on evaluations in recommendation. *International Journal of Machine Learning and Cybernetics*, 10(5), pp.813–831. Available from: https://doi.org/10.1007/s13042-017-0762-9.

Spotify (undateda). *Spotify*. [online] Spotify. Available from: https://open.spotify.com/ [Accessed 13 Apr. 2025].

Spotify (undatedb). *Web API | Spotify for Developers*. [online] developer.spotify.com. Available from: https://developer.spotify.com/documentation/web-api [Accessed 4 Apr. 2025].

Stash (undated). *Stash - Games tracker*. [online] Stash.games. Available from: https://stash.games/ [Accessed 27 Mar. 2025].

Statcounter (2024) *Desktop vs Mobile vs Tablet Market Share Worldwide | StatCounter Global Stats* Available from: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet [Accessed 23 Jan. 2025].

Steane, J. (2023) *The Principles and Processes of Interactive Design*, 2nd edition. London: Bloomsbury Publishing.

Taunk, K., De, S., Verma, S. and Swetapadma, A. (2019). *A Brief Review of Nearest Neighbor Algorithm for Learning and Classification*. [online] IEEE Xplore. Available from: https://doi.org/10.1109/ICCS45141.2019.9065747.

Thangavelu, M., Medisetty, U.K. and Astakhov, P. (2020). *Designing Edge Gateway, Uber's API Lifecycle Management Platform*. [online] Engineering | Uber Blog. Available from: https://www.uber.com/en-GB/blog/gatewayuberapi/ [Accessed 4 Apr. 2025].

The GraphQL Foundation (undated). *GraphQL: A query language for APIs*. [online] Graphql.org. Available from: https://graphql.org/ [Accessed 8 Apr. 2025].

Twitch Developers (undated). *Twitch API*. [online] Twitch Developers. Available from: https://dev.twitch.tv/docs/api/ [Accessed 19 Mar. 2025].

Udacity (2015). *UML Structural Diagrams: Component Diagram - Georgia Tech - Software Development Process*. [online] YouTube. Available from: https://www.youtube.com/watch?v=ipKJwnPsst8 [Accessed 23 Mar. 2025].

Usiskin, A. (2025). *Designing for the Thumb: Why More UX Designers Should Prioritize the Thumb Zone*. [online] Linkedin.com. Available from: https://www.linkedin.com/pulse/designing-thumb-why-more-ux-designers-should-zone-aaron-usiskin-gtrte [Accessed 3 Apr. 2025].

Vadlamani, S.L., Emdon, B., Arts, J. and Baysal, O. (2021). Can GraphQL Replace REST? A Study of Their Efficiency and Viability. *2021 IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*. Available from: https://doi.org/10.1109/ser-ip52554.2021.00009.

von Krogh, G. and Spaeth, S. (2007) The open-source software phenomenon: Characteristics that promote research. *The Journal of Strategic Information Systems*, 16(3) 236–253 Available from: https://doi.org/10.1016/j.jsis.2007.06.001

Wang, J. and Dong, Y. (2020). Measurement of Text Similarity: A Survey. *Information*, 11(9), p.421. Available from: https://doi.org/10.3390/info11090421

Wu, L., He, X., Wang, X., Zhang, K. and Wang, M. (2022) A Survey on Accuracy-oriented Neural Recommendation: From Collaborative Filtering to Information-rich Recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 35(05) pp. 4425 - 4445. Available from: https://doi.org/10.1109/TKDE.2022.3145690

Yuan, H. and Hernandez, A.A. (2023). User Cold Start Problem in Recommendation Systems: A Systematic Review. *IEEE Access*, 11, pp.136958–136977. Available from: https://doi.org/10.1109/access.2023.3338705

Zaina, L.A.M., Fortes, R.P.M., Casadei, V., Nozaki, L.S. and Paiva, D.M.B. (2022). Preventing accessibility barriers: Guidelines for using user interface design patterns in mobile applications. *Journal of Systems and Software*, [online] 186 Available from: https://doi.org/10.1016/j.jss.2021.111213

Appendices

Appendix A: UI Design Survey Questionnaire

31 anonymous participants provided answers to a UI Design Survey Questionnaire distributed through JISC Online Surveys. A transcript of these questions is provided below; in the original online survey the design screenshots were provided at full page width for clearer viewing and can be found in Section 4.24 of this document. The questionnaire could only be completed once all four tick boxes on Question 1 had been checked to obtain consent.

1. CONSENT TO PARTICIPATE IN RESEARCH

- ☐ 1. I confirm that I have read the information sheet dated 08/02/2025 (version 2.0) for the above study. I have had the opportunity to consider the information, and ask questions
- ☐ 2. I understand that as I have completed the study anonymously it will not be possible to remove any information I have provided, as you will not be able to identify me in any way.
- ☐ 3. I understand that individuals from the University of Lincoln may look at research data collected during the study, to ensure that the study is conducted appropriately.
- ☐ 4. I agree to take part in the above study

Structure of this questionnaire
• Section One will include some questions focusing on background information
• Section Two will focus on the design of the application

Section 1
2. Are you interested in video games?
[Yes / No]

3. How often do you play video games?
[Never / Rarely / Monthly / Weekly / Daily]

4. Have you considered keeping track of the video games you play?
[Yes / No]

5. Have you used any existing tools or apps to track your gaming activity?
[Yes / No]

6. If yes, which ones and what do you like/dislike about them?
[text box]

7. When completing a task on your phone, do you prefer using an application or a website?
[Application / Website]

8. Please explain your choice
[text box]

9. Does a recommendation system for what to play next appeal to you
[Yes / No]

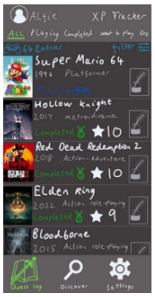
10. Why/Why not?
[text box]

11. Would you like the option for automatic tracking, such as syncing with game platforms, or would you prefer to track games manually?
[Automatic Tracking / Manual Tracking / Both]

12. Outside of features listed in the study details, what features (if any) would you want a video game tracking app to have?
[text box]

Section Two

This section will contain some examples for the design of the user interface in the proposed application and offer you the option to provide some feedback on it. The text on these screens is purely a placeholder currently, so the focus of this section is on layout, functionality and design choices



Main Screen of the Application: The screen below will act as the home of the application. It allows users to view and manage their game entries.

13. What are your first impressions of this screen?
[text box]

14. Does the layout feel intuitive?
[Yes / No]

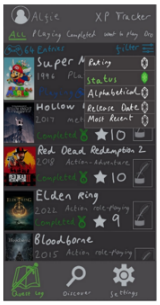
15. Why/Why not?
[text box]

16. Is it clear how to filter and sort game entries?
[Yes / No]

17. How would you go about editing a game entry?
[text box]

18. How do you feel about the colour scheme and layout of the screen?
[Poor / Okay / Good / Excellent]

19. What would you rate this screen on a scale of 1 – 5?
[1 / 2 / 3 / 4 / 5]



Filter Tab: This screen shows the options for filtering the game entries

20. What do you think about the filter tab design?
[text box]

21. Are there any other filter options you would like included?
[text box]

22. What would you rate this screen on a scale of 1 – 5?
[1 / 2 / 3 / 4 / 5]

Edit Game Entry: The next two screens show the options for editing a game entry. These screens are accessed by clicking on the quill next to an entry in the log

23. Do you feel as though this interface is intuitive? *
[Yes / No]

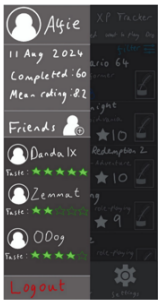
24. Why/Why not?
[text box]

25. How do you feel about the colour scheme and layout of the screens? *
[Poor / Okay / Good / Excellent]

26. Would you change or add anything to the design? If so what?
[text box]

27. Are there any additional options for tracking you would like included?
[text box]

28. What would you rate this screen on a scale of 1 – 5?
[1 / 2 / 3 / 4 / 5]



User Information and friends: This tab is accessed by clicking the user profile in the top corner of the screen. It offers an overview of the users and access to the user's friends. Here you can rate your friend's tastes to have the games they have played influence your recommendations

29. Do you feel as though this interface is intuitive? *
[Yes / No]

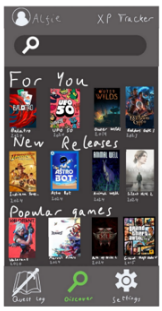
30. Why/Why not?
[text box]

31. How do you feel about the colour scheme and layout of the tab? *
[Poor / Okay / Good / Excellent]

32. Would you change or add anything to the design? If so what?
[text box]

33. Are there any other options you would expect to access from this tab?
[text box]

34. What would you rate this screen on a scale of 1 – 5?
[1 / 2 / 3 / 4 / 5]



Discover Screen: This screen allows users to search for games to add to their collection, and also discover new games that they may enjoy

35. Do you feel as though this interface is intuitive? *
[Yes / No]

36. Why/Why not?
[text box]

37. Is it obvious how to navigate to this screen? *
[Yes / No]

38. If not then why?
[text box]

39. Would you naturally come to this tab to find games to add to you list? *
[Yes / No]

40. If not then why?
[text box]

41. How do you feel about the colour scheme and layout of the tab? *
[Poor / Okay / Good / Excellent]

42. Would you change or add anything to the design? If so what?
[text box]

43. What would you rate this screen on a scale of 1 – 5?
[1 / 2 / 3 / 4 / 5]

Game details: After searching for a game, this screen will be displayed, allowing users to see the details of the game, and also give them the option to add the game to their log

44. Do you feel as though this interface is intuitive? *
[text box]

45. Why/Why not?
[text box]

46. How do you feel about the colour scheme and layout of the screens? *
[Poor / Okay / Good / Excellent]

47. Would you change or add anything to the design? If so what?
[text box]

48. What would you rate this screen on a scale of 1 – 5?
[1 / 2 / 3 / 4 / 5]

Survey End: Thank you for your time in answering the questions in this survey, your input is highly valued and will have a strong impact on the development of this application

Appendix B: User Interview Framework

The interview was conducted with 5 anonymous participants after obtaining consent in line with the LEAS application. The framework for it can be seen below:

This interview will consist of two main stages. In stage one you will have the opportunity to explore the application and attempt to complete a list of tasks. After this you will be asked to answer some questions and provide your thoughts on the application.

Stage One

Please take a moment now to become comfortable with the application. When you are ready, please say so, and you can begin working through the tasks below. Please indicate how easy you found each task on a scale from 1-5, with 5 being easy, and one being difficult.

1. Navigate to these pages: Quest Log, Discover and Settings
2. Filter the games in the Quest Log alphabetically and by one other option, both forwards and backwards.
3. Group the games by status.
4. Find out the average rating of all games in the Quest Log.
5. Navigate to the edit page for a game entry in the Quest Log.
6. Change the status, rating and date for a game entry and save the change.
7. Remove a game entry from the Quest Log
8. From the Quest Log page navigate to the details for a game entry.
9. From the details page, navigate to the edit entry page for that game entry.
10. From the details page for a game, navigate to a similar game.
11. Add a similar game to the Quest Log
12. From the discover page, add a recommended game to the Quest Log.
13. Search for a game and add it to the Quest log.
14. Change the application's theme (light/dark mode)

Stage Two

1. What would you rate the application on a scale from 1-10?
2. Did you find the menus easy to navigate?
3. What did you like and dislike about the application?
4. Do you think all the relevant information about a game is displayed on its detail page?
5. Do the games recommended make sense/appeal to you?
6. Are there any features that you think are missing?
7. Would you use this application if you wanted to track the video games you have played. If yes, then why and if not then why not?
8. Do you have any other comments?