San Jose State University

Department of Computer Science, College of Science

CS 157A

Spring 2024

Professor Aravind Rokkam

Video Game Library Deliverable

Group Members:

Aaron D'Souza

Alex Bhandari

Christian Garo

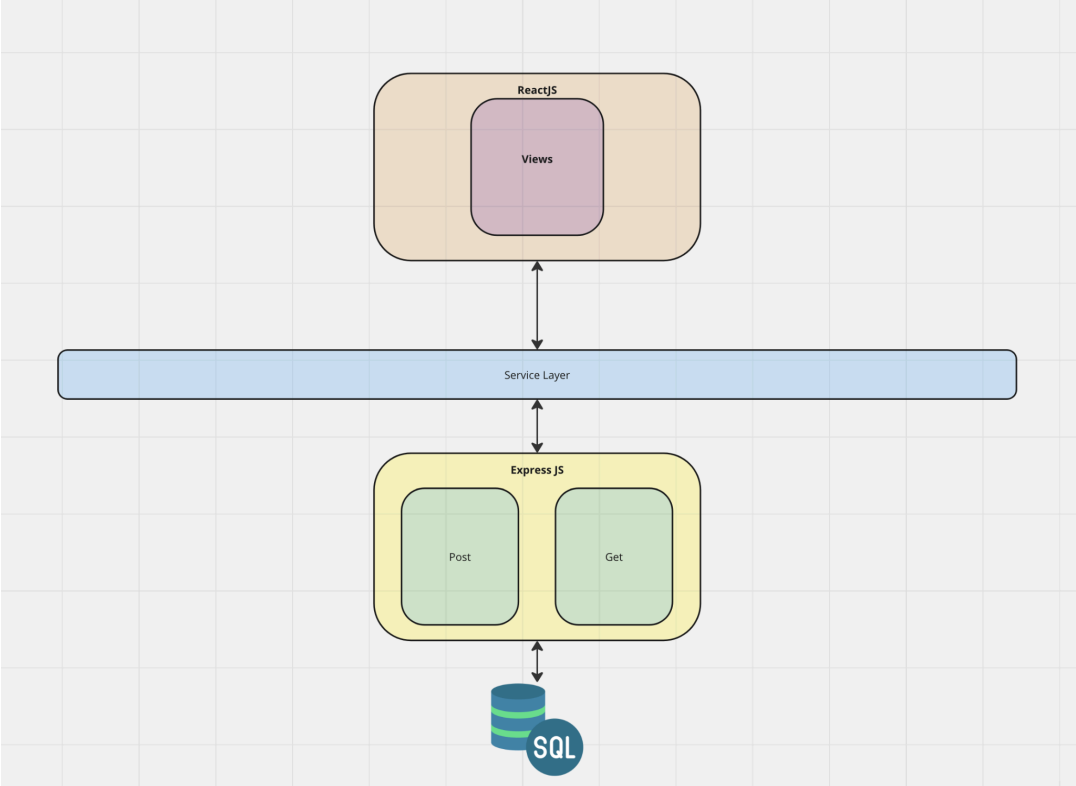
Project Introduction

Despite the ever-evolving industry of creative application development from recipe tracking applications to POS applications, there is one domain that generally receives little appraisal: the gaming industry. As a team, we realized that there is a general lack of attention given to gamers in terms of utility tools. As game enthusiasts, we recognize that we're constantly looking for popular games once complete with a game series. Finding games over the web, although efficient, can often be overwhelming to the average user. Gamers will often find multiple games of their liking but end up only playing a few since there were simply too many good selections to keep track of. To address these issues, our team decided to create a game management site that would enable users to organize their future games in a wishlist. The team decided to create a project that tailors the needs of gamers by allowing users to rate, review, and use these points to determine the personal fit of a game.

Objective

The goal of the project is to create a web platform for gamers to track games they may currently have or want across all their platforms. The objective is to allow users to rate, review, look for potential interesting games, and add them to their wishlist. The function of ratings and reviews will foster site-wide community-building and make it easier for users to understand games played by others and decide if they're worth adding to their wishlists. This will better help users find games they might like and thereby save time when it comes to searching for new games.

Project High-Level Design



Overall Summary

The project follows a very standard level of clean code architecture. The main theme reflected by the project follow the key components of modularization where the Views are responsible for the sole purpose of displaying the data, the service layer which handles logic to handle data requests, the networking layer which handles the “GET” or “POST” request specified by the service layer to which the data is queried from the SQL database.

React JS Layer

This portion of the architecture is solely for the front-end. These include all the views within the application. It is wrapped around a React JS layer as this library is the key component to making sure all the pages are properly rendered and every necessary state management is handled accordingly. This includes the **Home.js, Login.js, Profile.js and Signup.js.** The main responsibility of these pages is to interact with the user, presenting information and capturing input. It is the overall interface which allows users to navigate and engage with the functionality of the application. Furthermore, through React JS, these pages reactively update based on user interaction and changes within the data. This is to make sure users are able to have a seamless experience while facilitating fully functional state management and is consistent within the overall application.

Service Layer

The main purpose of the service is to differentiate the views from the logic. It plays a critical role to make sure there is a separation between the user React JS layer and the data handling portion facilitated by Express JS. The services within this layer include **userAuthService, loginServiceAuth,** and **App.js.** Together they manage the user authentication, session and overall state management within the application. It is important to acknowledge these abstractions from the presentation layer as the Service Layer makes sure that the Pages are focused solely for rendering and interaction, making the React components lightweight. By following these coding conventions, it enhances the overall modularity and maintainability

which not only helps in the development process for testing, but helps make the overall user experience even more seamless. Furthermore, it allows the system to be more scalable and easier to manage and user base grows.

Express JS Layer

The Express JS layer serves as the spine for the server-side functionality within the application. It utilizes Node.js library, Express JS, which handles all the routing for the server-side logic. The **server.js** utilizes important middleware such as "cors" for cross-origin resource sharing, **express.json**() to parse any JSON requests and any essential file serving capabilities to make sure the backend is running smoothly.

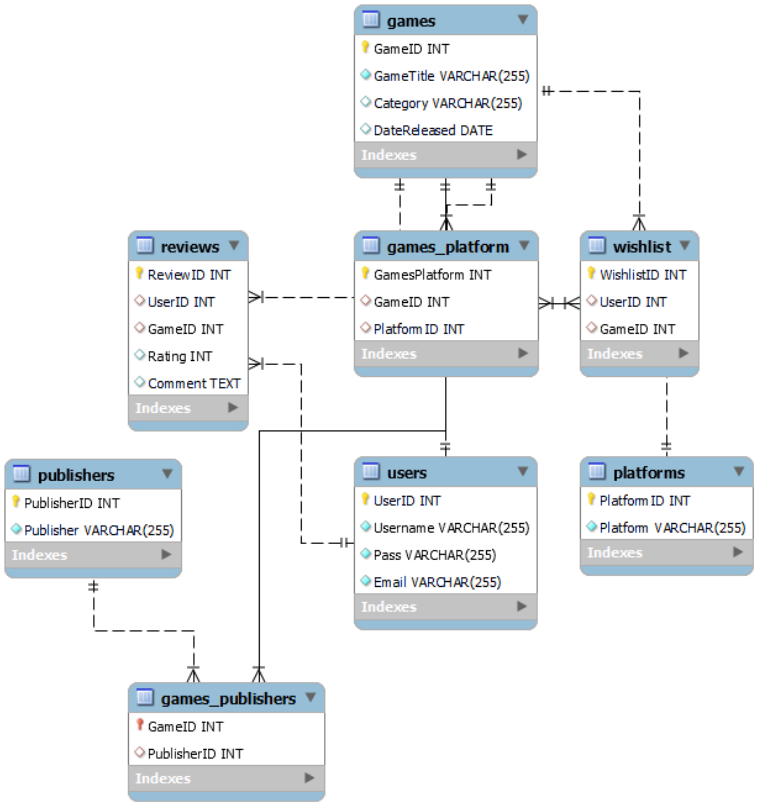
Furthermore, this integrates with the MySQL database to handle various endpoints for authentication, registration, data retrieval, and data manipulation. While there are many routing functionalities, the important ones are the **/signup, /login, user/details,** and **/games/reviews.** Signup and Login handle the user creation and validation by communicating directly with the SQL database and executing queries securely to handle the respective service. Just like login and signup, user details and game reviews perform specific SQL queries to fetch or update according to the client request or requested by the service layer.

Lastly, this layer is absolutely necessary for handling errors and status updates within the entire application. This is not only important for the development side, but in case there are issues that are floating, the user understands and is able to move forward accordingly. It ensures robust backend management and system's overall reliability.

Database Layer

The SQL Database layer holds the overall foundation of data management within the application. With predefined objects and intricately designed models, it leverages the robust relational database system to store and organize the application's data. The layer in itself is designed with tables for Users, Games, Reviews, Publishers, Platforms, Games_Platform, Game_Publishers, and Wishlists. These tables are connected with appropriate relationships allowing for proper query functionalities, updates, and data management. This layer ensures that all the data interaction is consistent and usable. Overall, the database layer is important in the sense that it allows for dynamic content management and a consistent user experience across the gaming platform.

Database Design with Information of All Tables and Relationships (ER Diagram)



The database schema for the Video Game Management Library is designed to support the application's main functionality. Each table within the data serves a big importance not only in defining the entire database, but the application too.

Database Information

Users: The **Users** table stores all the information about the user. **UserID** is the primary key for this table and will automatically increment whenever a new user signs up. The table also contains a **Username, Pass, and Email** column. Each user will have a unique **UserID** tied to their username, password, and email so that other parts of the application can reference the user

appropriately. When a user logs in, the database will search for the user's **Username** and **Pass** to see if it matches the input. If it does, the user will be able to log in. To prevent any duplicate usernames and emails, the **Username** and **Email** fields are set to be unique.

Publishers: The **Publishers** table stores all the publishers for a particular game. **PublisherID** contains the ID for the publisher, is an integer, and is the primary key for the table. **Publisher** is a VARCHAR (a string) that stores the publisher's name.

Games: The **Games** table stores information about the games for the website. **GameID** is the primary key for the table. Each game will have a specific **GameID** tied to it so that it can be referenced in the other tables and in the code. When adding additional games to the **Games** table, **GamesID** will auto-increment to account for additional games being added. **GameTitle** will contain the name of the game, and **DataReleased** will contain the release date for the game. By default, **Category** and **DateReleased** are set to null, while **GameTitle**'s default value needs to be a string.

Wishlist: The **Wishlist** table functions as a junction table between **Users** and **Games,** with **WishlistID** acting as the primary key. To prevent duplicate games from being added to the wishlist, **GameID** is set to be unique to prevent multiple games from being added to a wishlist. This table also stores information about a user's wishlist. This is done to prevent any violation of the second normalization form when placed in either the **Users** or **Games** table, as a **WishlistID** alone cannot determine a **UserID** or a **GameID. UserID** and **GameID** are set to be foreign keys of the **Users** and **Games** table so that **Wishlist** can point to a particular user and game.

Reviews: The **Reviews** table stores reviews made by a user. **ReviewID** is the primary key of the table. A user can make a **Comment** about a particular game, which is stored in this database as text. In order for a review to be added, the **UserID** and **GameID** must not be null, including the rating. **UserID** and **GameID** are set to be foreign keys of the **Users** and **Games** table as well so that **Reviews** can point to a particular user and game.

Platforms: The **Platforms** table stores information about the available platforms the game is available on. By platforms, this means the consoles (like Playstation 5, Playstation 4) and operating systems (like Microsoft Windows) a user is able to play the game on. The table contains the primary key **PlatformID** and the column **Platform**, where it stores the name of the platform. When more platforms are added to the table, **PlatformID** is automatically incremented to account for additional platforms.

Games_Publishers: Games_Publishers acts as a junction table to prevent any violations of the third normalization form. This table contains **GameID** (primary key) and **PublisherID** and will act as a reference for a game's publisher. **GameID** and **PublisherID** are set to be foreign keys of the **Games** and **Publishers** table so that **Game_Publishers** can point to a particular game and platform specified.

Games_Platforms: In the same manner as **Games_Publisher**, this acts as a junction table to prevent any violations of the third normalization form. This table also contains a **GameID** (primary key) and **PlatformID**. This table acts as a reference for the platform the game is

available on. **GameID** and **PlatformID** are set to be foreign keys of the **Games** and **Platforms** table so that **Game_Platforms** can point to a particular game and platform specified.

ER Relationships

Starting with the **Users** table, a **many-to-one** relationship is formed between the **Reviews** table and the **Users** table since multiple users can add a review for a game and it will be stored in the table. A **many-to-one** relationship is also formed between the **Wishlist** table and the **Users** table, since users can add multiple games to their wishlist. One notable thing to note is that the **Wishlist** and **Reviews** table depends on an existing **UserID,** indicating that there is a weak entity relationship with the **Users** table.

The **Games** table forms multiple relationships with different tables. **Games_Publishers** forms a strong entity many-to-one relationship with the Games table. The games_publishers table contains multiple GameIDs that correspond to a specific game. Each publisher can publish different games, meaning many of the same PublisherID can be in the database. **Reviews** forms a many-to-one relationship with the **Games** table as well because multiple reviews can be placed for a game. The relationship is a weak-entity relationship because the **GameID** in the **Review** table depends on the GameID present in the Games table to be present to form the review. **Games_Platform** forms a many-to-one relationship with the **Games** table because multiple platforms are available for a single game. It is also a weak-entity relationship because it needs reference the **GamesID** in the **Games** table to define what a games_platform is. Lastly, **Wishlist** forms a many-to-one relationship with the **Games** table since a wishlist can contain many games. This is also a weak entity relationship because a foreign key needs to be established **GameID** in

the **Wishlist** table with the primary key **GamesID** in the **Games** table. This is done to ensure that a **Wishlist** entry can be created in the table.

The **Games_Publishers** table forms a many-to-one relationship with the **Publishers** table. This relationship is considered a many-to-one relationship because there can be multiple publishers for a single game. This is also considered a **weak-entity relationship** due to how **PublisherID** in the **Games_Publishers** table relies on the primary key **PublisherID** in **Publishers** to be able to define what a game's publisher is. **Games_Publishers** also forms a many-to-one relationship with the **Games** table because there can be multiple games stored in the **Games_Publisher** table, while only one kind of game can be places in the **Game_Publishers** table.

The **Reviews** table forms a many-to-one relationship with the **Games** table. This relationship is a many-to-one relationship because multiple reviews can be placed for a video game. However, only one game will have its own dedicated reviews. This relationship is also a weak-entity because both **UserID** and **GameID** depend on the primary keys of **UsersID** in **Users** and **GameID** in **Games** respectively. This is to ensure that a review can be created properly for a particular user with the respective game.

The **Games_Platform** table forms a many-to-one relationship with the **Platforms** table. This relationship is a many-to-one relationship because games can be on multiple platforms, but only one platform exists for a single game. This relationship is also a weak-entity relationship because **PlatformID** relies on the primary key for **PlatformID** in **Platforms.** This ensures that a platform for a game can be uniquely identified and can reference a platform in the **Platforms** table.

Normalization of tables

First Normal Form

```
CREATE TABLE Reviews (
  ReviewID INT AUTO_INCREMENT PRIMARY KEY,
  UserID INT NOT NULL,
  GameID INT NOT NULL,
  Comment TEXT,
  Rating INT NOT NULL,
  FOREIGN KEY (UserID) REFERENCES Users(UserID),
  FOREIGN KEY (GameID) REFERENCES Games(GameID)
);
CREATE TABLE WishList (
  WishlistID INT AUTO_INCREMENT PRIMARY KEY,
  UserID INT NOT NULL,
  GameID INT NOT NULL,
  FOREIGN KEY (UserID) REFERENCES Users(UserID),
  FOREIGN KEY (GameID) REFERENCES Games(GameID)
);
CREATE TABLE Games (
  GameID INT AUTO_INCREMENT PRIMARY KEY,
  GameTitle VARCHAR(255) NOT NULL,
  Category VARCHAR(255) DEFAULT NULL,
```

```
  DateReleased DATE DEFAULT NULL,
  Platform VARCHAR(255),
  Publisher VARCHAR(255)
);
CREATE TABLE Users (
  UserID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  Username VARCHAR(255) NOT NULL,
  Pass VARCHAR(255) NOT NULL,
  Email VARCHAR(255) NOT NULL,
  UNIQUE KEY (Username),
  UNIQUE KEY (Email)
);
```

To satisfy 1NF all the columns in each of the tables can only hold an attribute of a single value and has a primary key that is unique enough to identify its row. To fulfill this criteria, there are no repeating groups and each column has a primary key: **UserID, GameID, ReviewID, WishlistID.**

Second Normal Form

```
CREATE TABLE Games (
  GameID INT AUTO_INCREMENT PRIMARY KEY,
  GameTitle VARCHAR(255) NOT NULL,
  Category VARCHAR(255) DEFAULT NULL,
  DateReleased DATE DEFAULT NULL
```

```
);

CREATE TABLE GamePlatforms (
  GameID INT NOT NULL,
  Platform VARCHAR(255) NOT NULL,
  FOREIGN KEY (GameID) REFERENCES Games(GameID)
);

CREATE TABLE GamePublishers (
  GameID INT NOT NULL,
  Publisher VARCHAR(255) NOT NULL,
  FOREIGN KEY (GameID) REFERENCES Games(GameID)
);
```

As for the 2NF, the first criteria means that the database would need to fulfill the 1NF requirement. Once that is satisfied, the second portion to fulfill is that every detail within the table must be dependent completely on the primary key for the whole table, not just part of it. **Games** originally contained **Platform** and **Publisher** fields which means that these attributes aren't fully dependent on **GameID.** The problem arises when it would be redundant if multiple games share the same platform or publisher, for example. Hence to fix this error, it makes sense to create a whole different table **Game_Platforms** and **Game_Publishers** and having them linked with a foreign key to **Games.**

Third Normal Form

```
CREATE TABLE Users (
  UserID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  Username VARCHAR(255) NOT NULL,
  Pass VARCHAR(255) NOT NULL,
  Email VARCHAR(255) NOT NULL,
  UNIQUE KEY (Username),
  UNIQUE KEY (Email)
);
CREATE TABLE Games (
  GameID INT AUTO_INCREMENT PRIMARY KEY,
  GameTitle VARCHAR(255) NOT NULL,
  Category VARCHAR(255) DEFAULT NULL,
  DateReleased date DEFAULT NULL
);
CREATE TABLE Platforms (
  PlatformID INT AUTO_INCREMENT PRIMARY KEY,
  Platform VARCHAR(255) NOT NULL
);
CREATE TABLE Publishers (
  PublisherID INT AUTO_INCREMENT PRIMARY KEY,
  Publisher VARCHAR(255) NOT NULL
);
CREATE TABLE WishList (
```

```
    WishlistID INT AUTO_INCREMENT PRIMARY KEY,
    UserID INT DEFAULT NULL,
    GameID INT DEFAULT NULL,
    UNIQUE KEY(WishListID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (GameID) REFERENCES Games(GameID)
);
CREATE TABLE Reviews (
    ReviewID INT AUTO_INCREMENT PRIMARY KEY,
    UserID INT NOT NULL,
    GameID INT NOT NULL,
    Comment TEXT,
    Rating INT NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (GameID) REFERENCES Games(GameID)
);
CREATE TABLE Games_Platform (
  GamesPlatform INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  GameID INT DEFAULT NULL,
  PlatformID INT DEFAULT NULL,
  FOREIGN KEY (GameID) REFERENCES Games(GameID),
  FOREIGN KEY (PlatformID) REFERENCES Platforms(PlatformID)
);
```

```
CREATE TABLE Games_Publishers (
  GameID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  PublisherID INT DEFAULT NULL,
  FOREIGN KEY (GameID) REFERENCES Games(GameID),
  FOREIGN KEY (PublisherID) REFERENCES Publishers(PublisherID)
);
```

To fulfill the Third Normal Form, the database would need to be in 2NF and also follow the rule where there should be no transitive dependency for non-key attributes. This means that no non-key attribute should depend on another non-key attribute. This is more apparent in the **Games** where the separation of **Platforms** and **Publishers** results in fulfilling both 2NF and 3NF by removing transitive dependencies. This would've been the case prior where **Platform** had a big dependency on **GameID** indirectly. Furthermore, **GamePlatforms** and **GamePublishers** help avoid transitive in the **Games** table by differentiating platform and publisher attributes into separate tables that are instead linked by foreign keys, which in turn results in the attributes within **Games** table dependent on **GameID** (primary key).

Importance of Normalization

This process of normalization is important to maintain integrity and efficiency of the overall database. It allows for faster queries and more importantly, avoiding data anomalies. This helps with efficiency and helps the application be scalable.

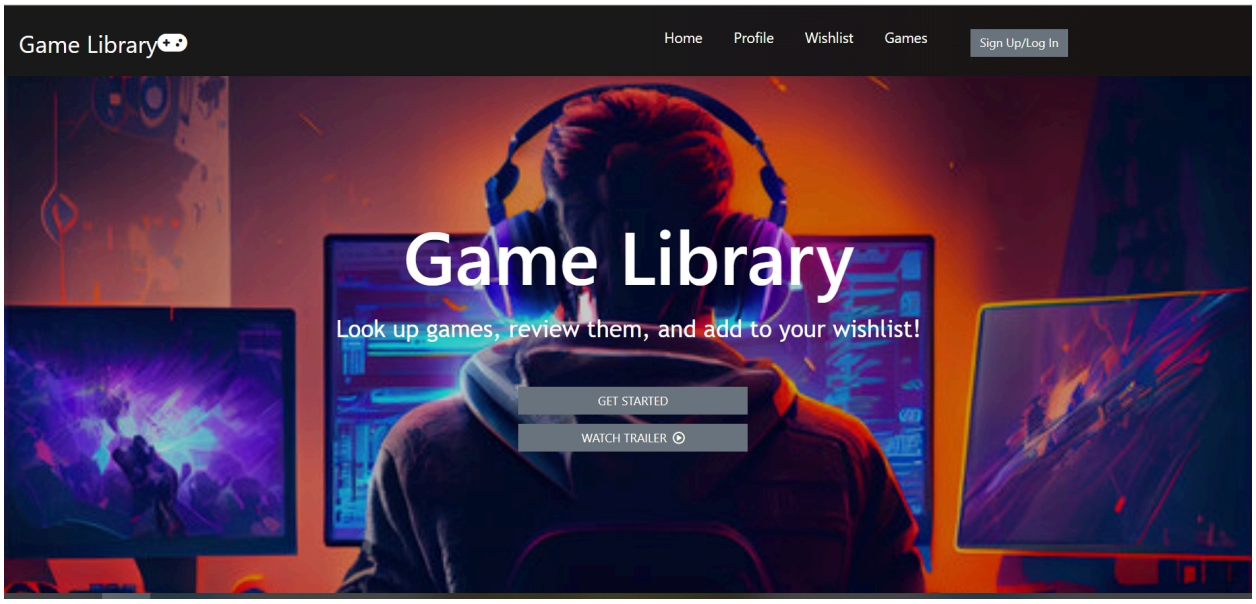
Results

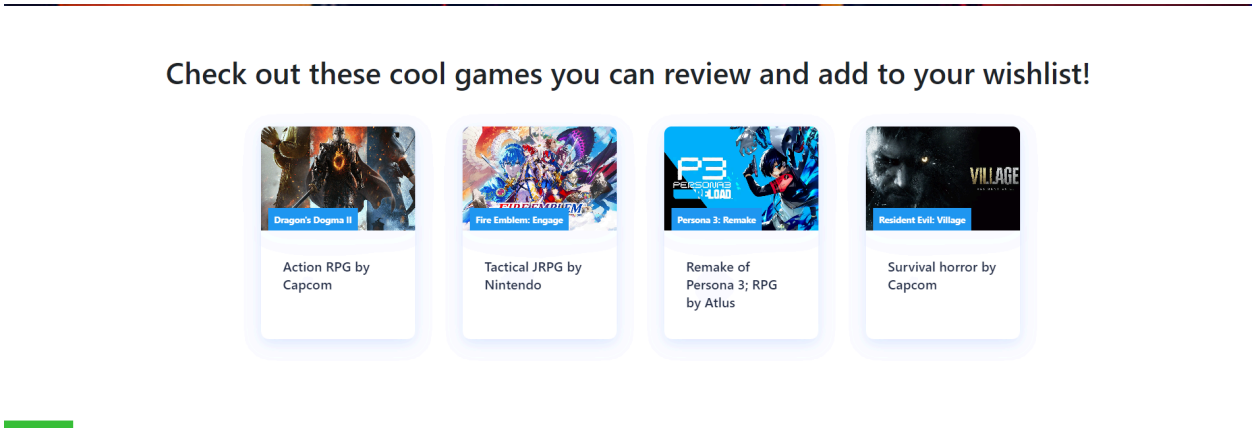
The final result of the project included a robust, multi-page react application. The site included a home page with a limited, sample set of games displayed as cards and redirection links to the wishlist, profile page, and login/signup. Clicking on game cards will take users to the respective game page, from which they can add that game to their wishlists. Game pages will allow users to rate, drop a review, and add that game to their wishlist. Users will also have a profile page that will display their site-wide wishlists and reviews. A dedicated game page will be available in a simpler list layout displaying a larger variety of all games available on the site. Of course, future versions that stretch beyond the scope of this class will include amplified networking between users live. This would entail a blog/community form platform of users sharing reviews and ratings about games they played to better offer input for users considering a particular game. In addition, pulling information about different games from an API will be done in the future. This is more efficient than manually adding new games to the database, giving us flexibility if we ever want to expand this project. On the current note, the end product is a simple, user-friendly, game manager that tailors the basic utility of gamers and game enthusiasts alike.

Screenshots of the Application with All the Views

Front-End (External Level)

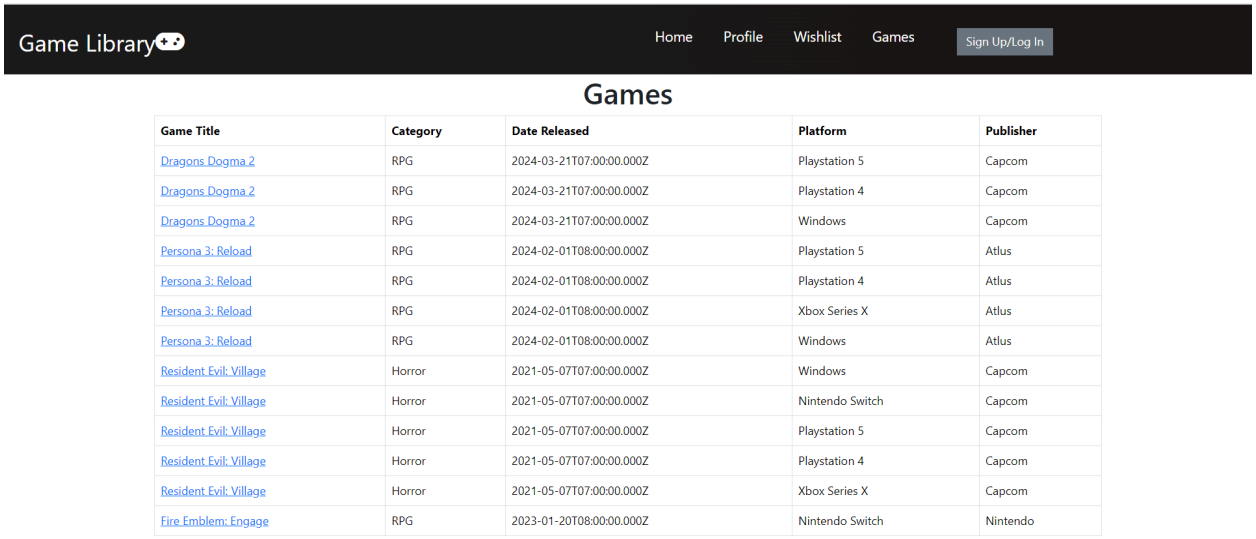
Home Page





The homepage allows you to navigate and add games to the wishlist as shown below.

Games Tab



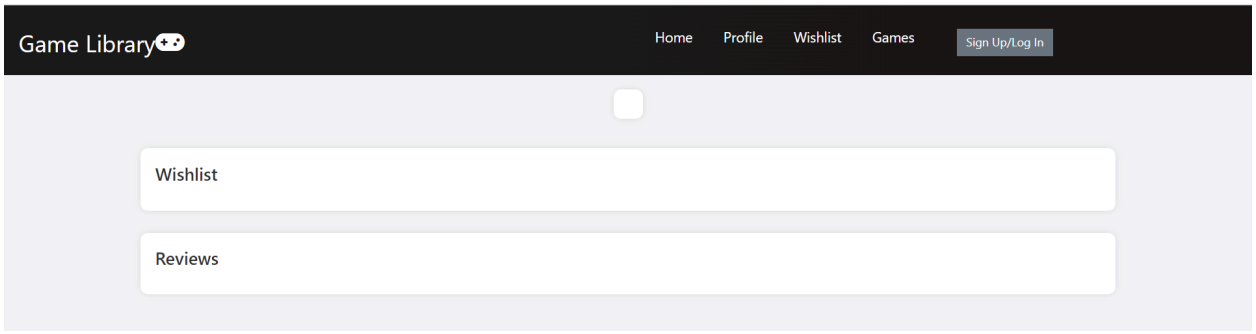
Game Title	Category	Date Released	Platform	Publisher
Dragons Dogma 2	RPG	2024-03-21T07:00:00.000Z	Playstation 5	Capcom
Dragons Dogma 2	RPG	2024-03-21T07:00:00.000Z	Playstation 4	Capcom
Dragons Dogma 2	RPG	2024-03-21T07:00:00.000Z	Windows	Capcom
Persona 3: Reload	RPG	2024-02-01T08:00:00.000Z	Playstation 5	Atlus
Persona 3: Reload	RPG	2024-02-01T08:00:00.000Z	Playstation 4	Atlus
Persona 3: Reload	RPG	2024-02-01T08:00:00.000Z	Xbox Series X	Atlus
Persona 3: Reload	RPG	2024-02-01T08:00:00.000Z	Windows	Atlus
Resident Evil: Village	Horror	2021-05-07T07:00:00.000Z	Windows	Capcom
Resident Evil: Village	Horror	2021-05-07T07:00:00.000Z	Nintendo Switch	Capcom
Resident Evil: Village	Horror	2021-05-07T07:00:00.000Z	Playstation 5	Capcom
Resident Evil: Village	Horror	2021-05-07T07:00:00.000Z	Playstation 4	Capcom
Resident Evil: Village	Horror	2021-05-07T07:00:00.000Z	Xbox Series X	Capcom
Fire Emblem: Engage	RPG	2023-01-20T08:00:00.000Z	Nintendo Switch	Nintendo

---

This portion is the available games within the database itself. This is where the users are able to click on the games and add reviews or wishlists to it.

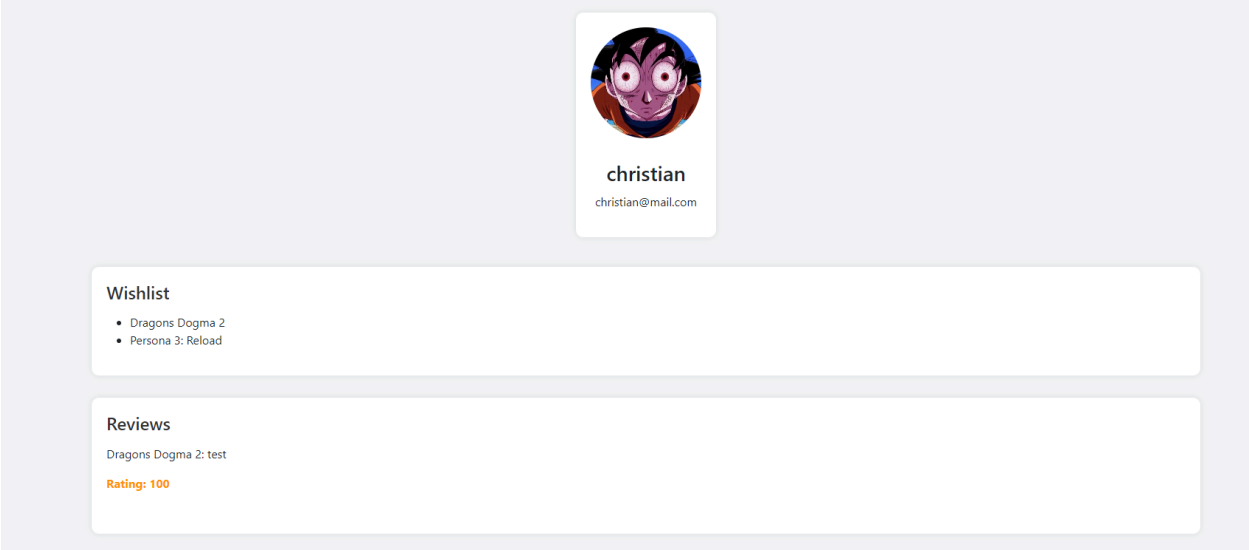
Profile Tab

Not Logged In:



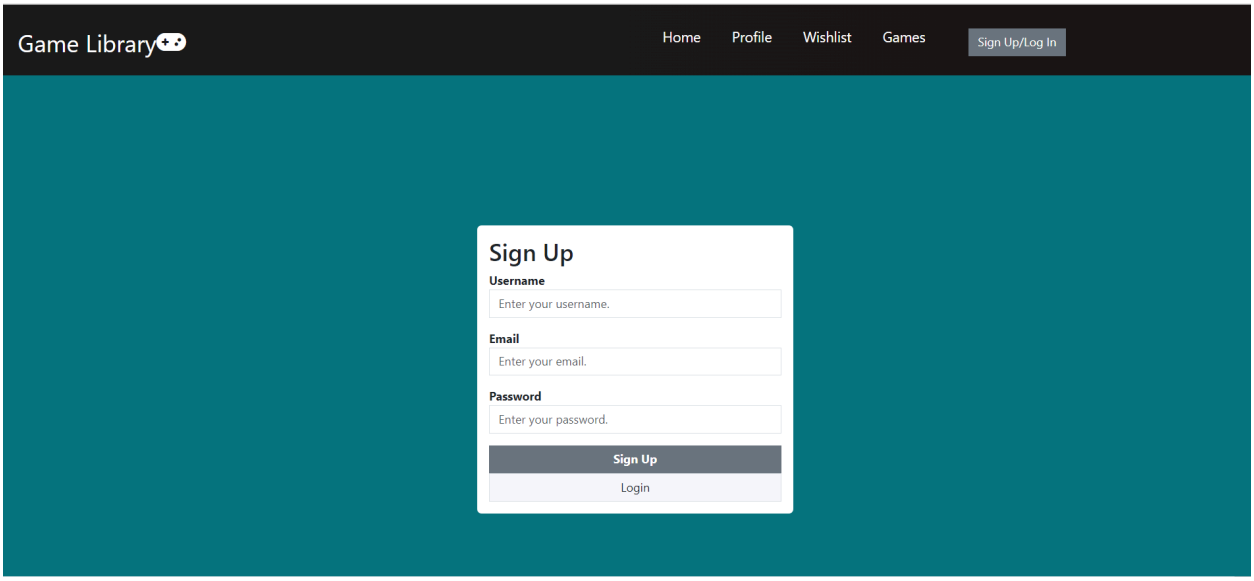
This is how the profile page looks like when the user is not logged in. As shown, there is no profile picture, reviews, or wishlist being shown.

Logged In:

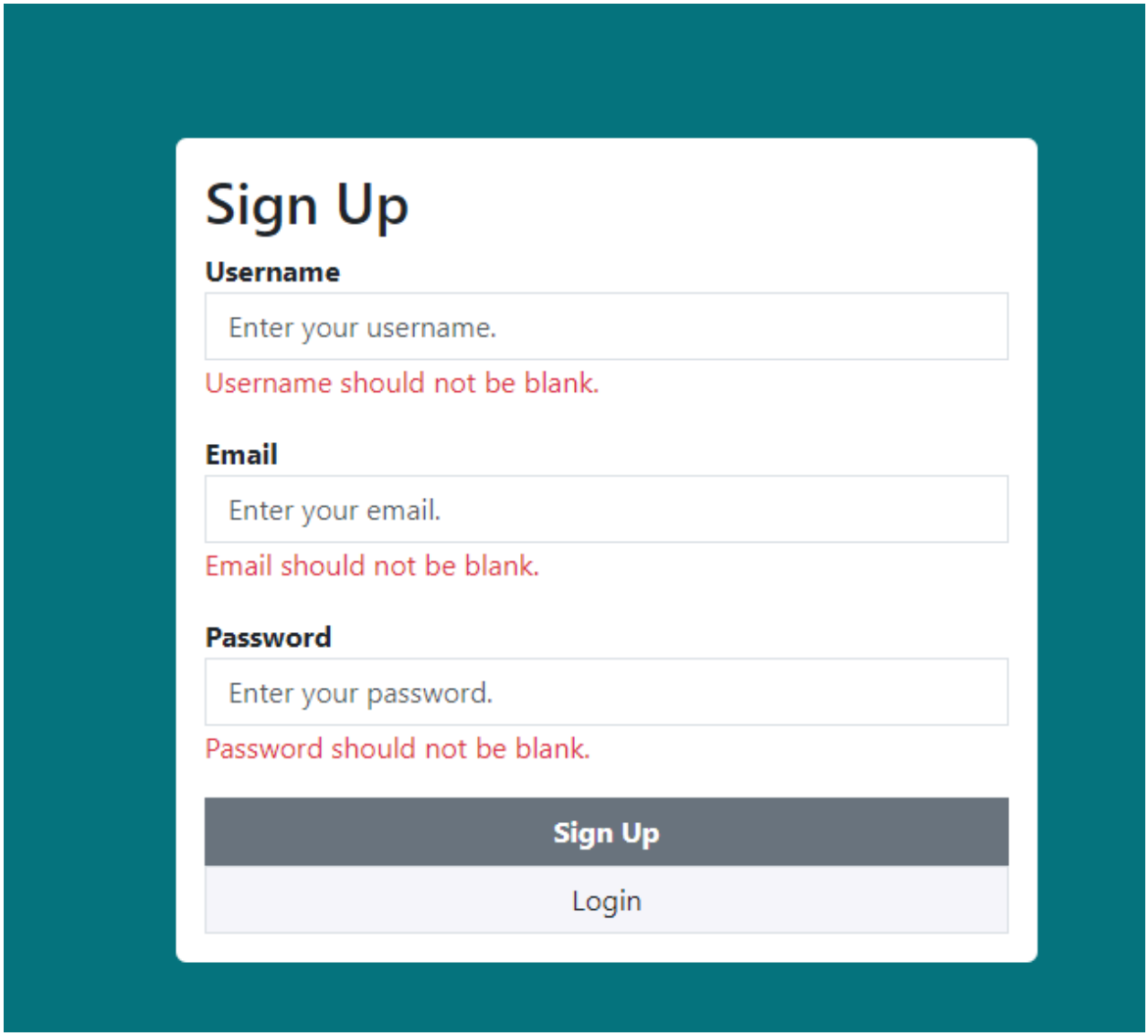


This is how the profile page looks like when the user is logged in. In this screenshot, a review for Dragons Dogma 2 by the current user is shown. The user has put two games in their wishlist: Dragons Dogma 2 and Persona 3: Reload.

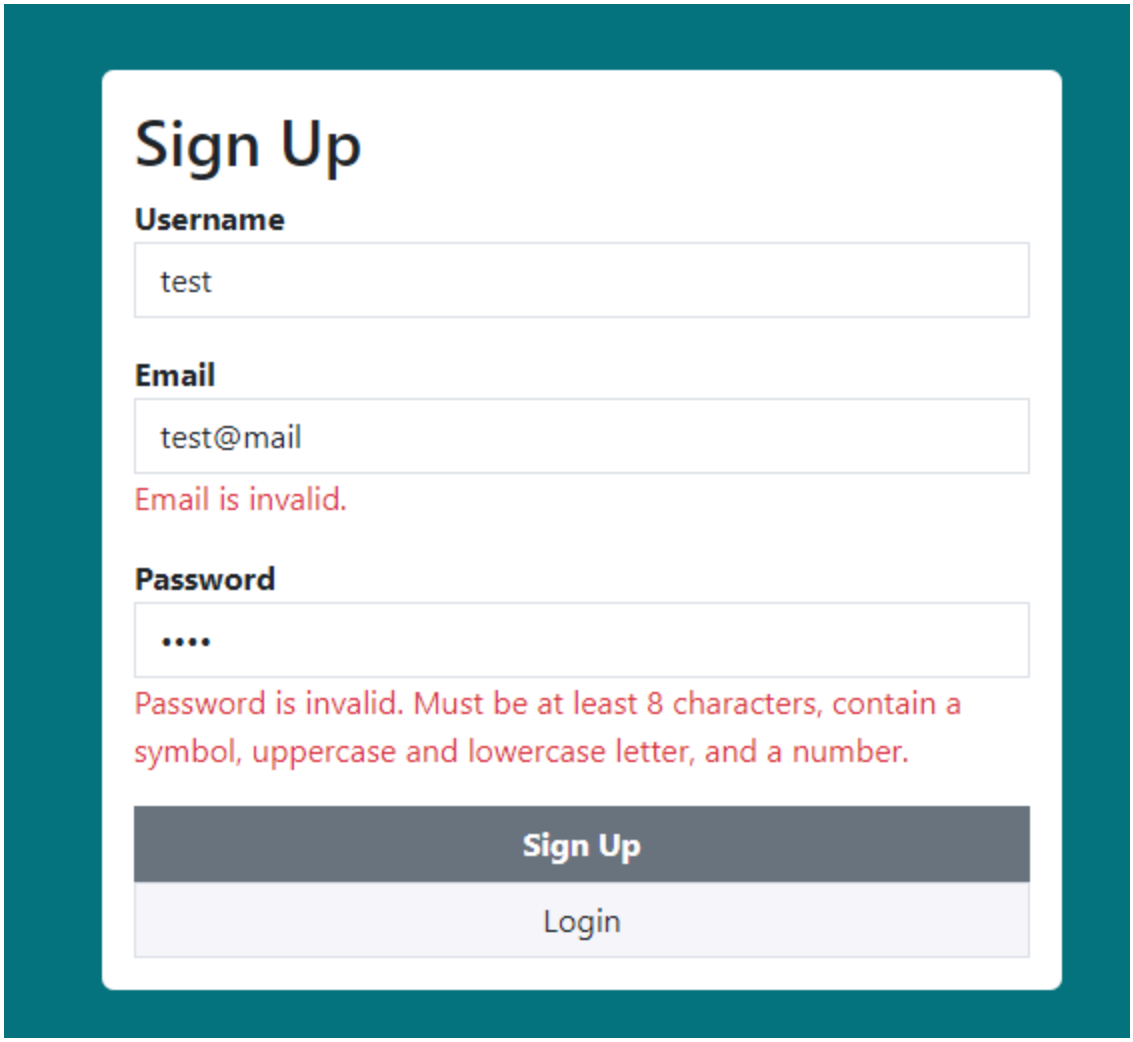
Sign Up



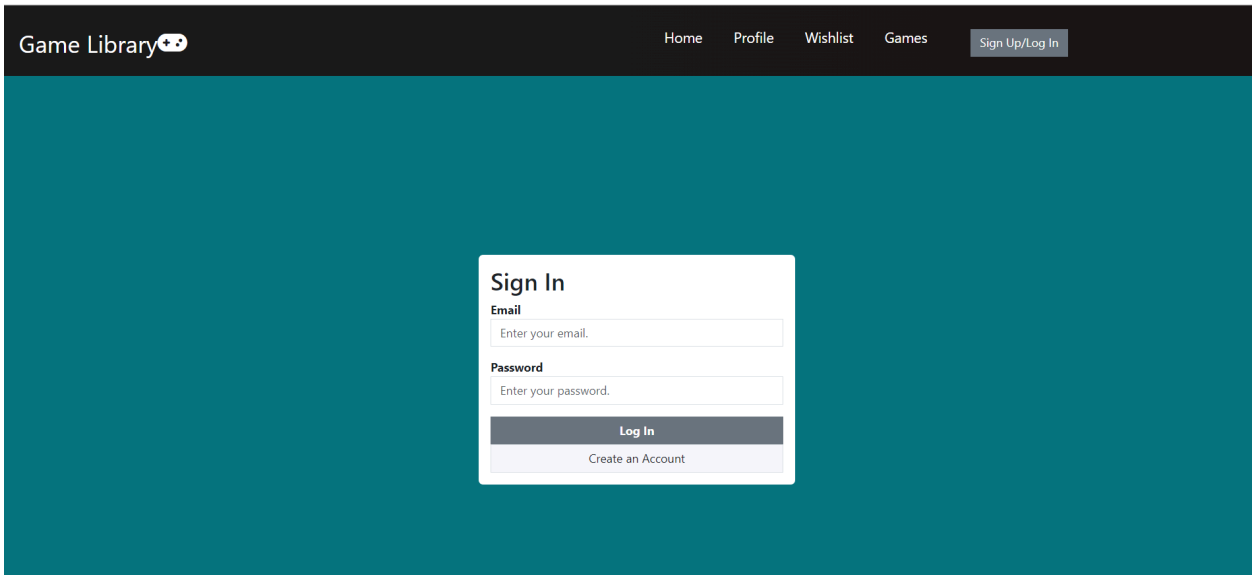
This is how the sign up page looks like. The user will type in a username, email, and password before signing up. There are checks in place to prevent any null entries from being added to the database, shown below.



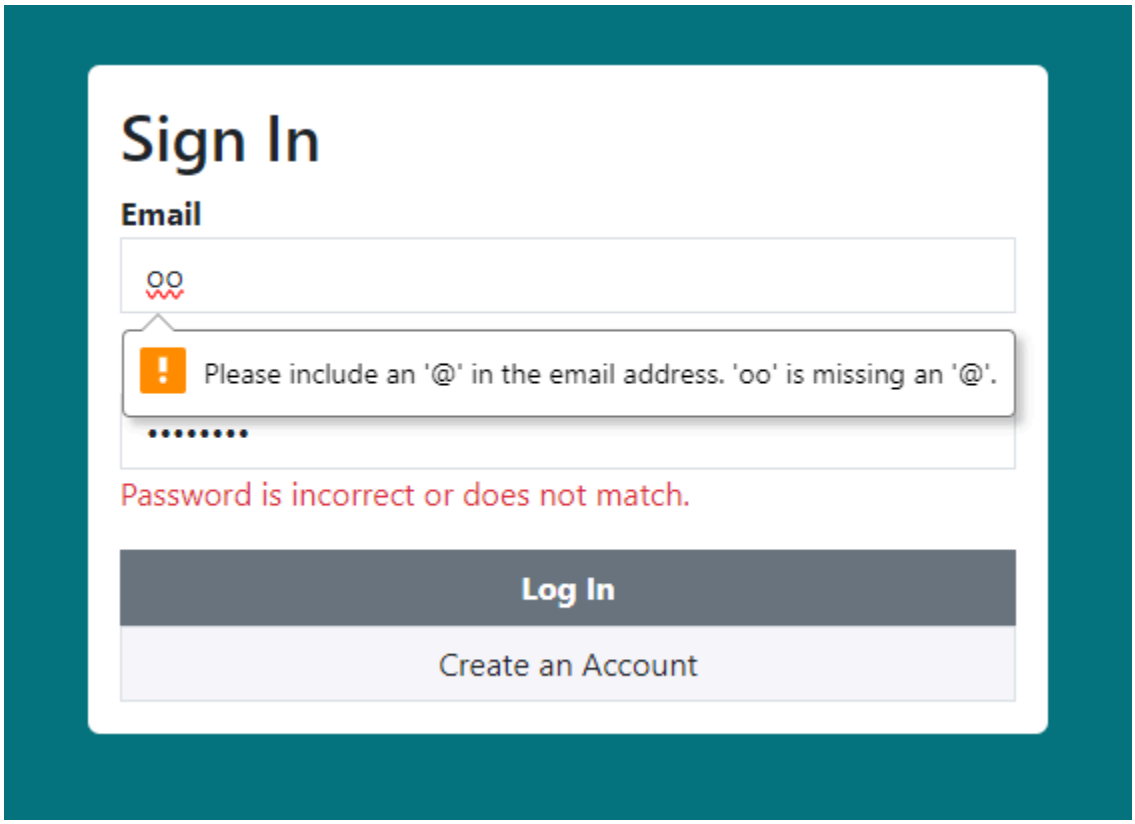
There are also checks in place to ensure that the user enters a valid e-mail and password, shown below.


Sign Up
Username
test
Email
test@mail
Email is invalid.
Password
••••
Password is invalid. Must be at least 8 characters, contain a symbol, uppercase and lowercase letter, and a number.
Sign Up
Login

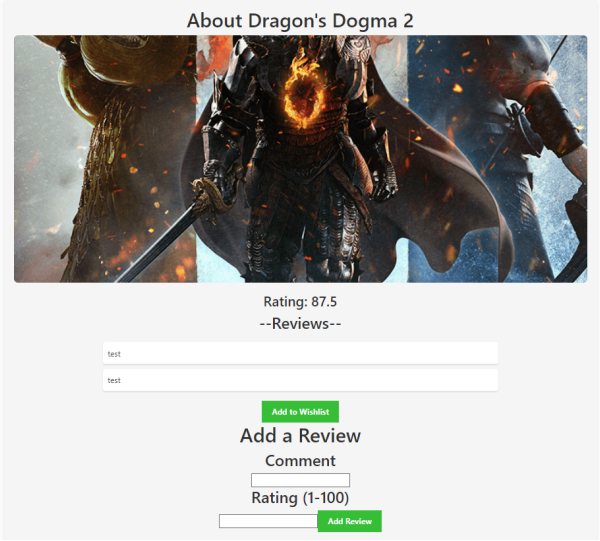
Log In



This is the log in page that allows users to log in and see their profile and wishlist. There are checks to validate if an e-mail is valid and to check if the password is correct, shown below.

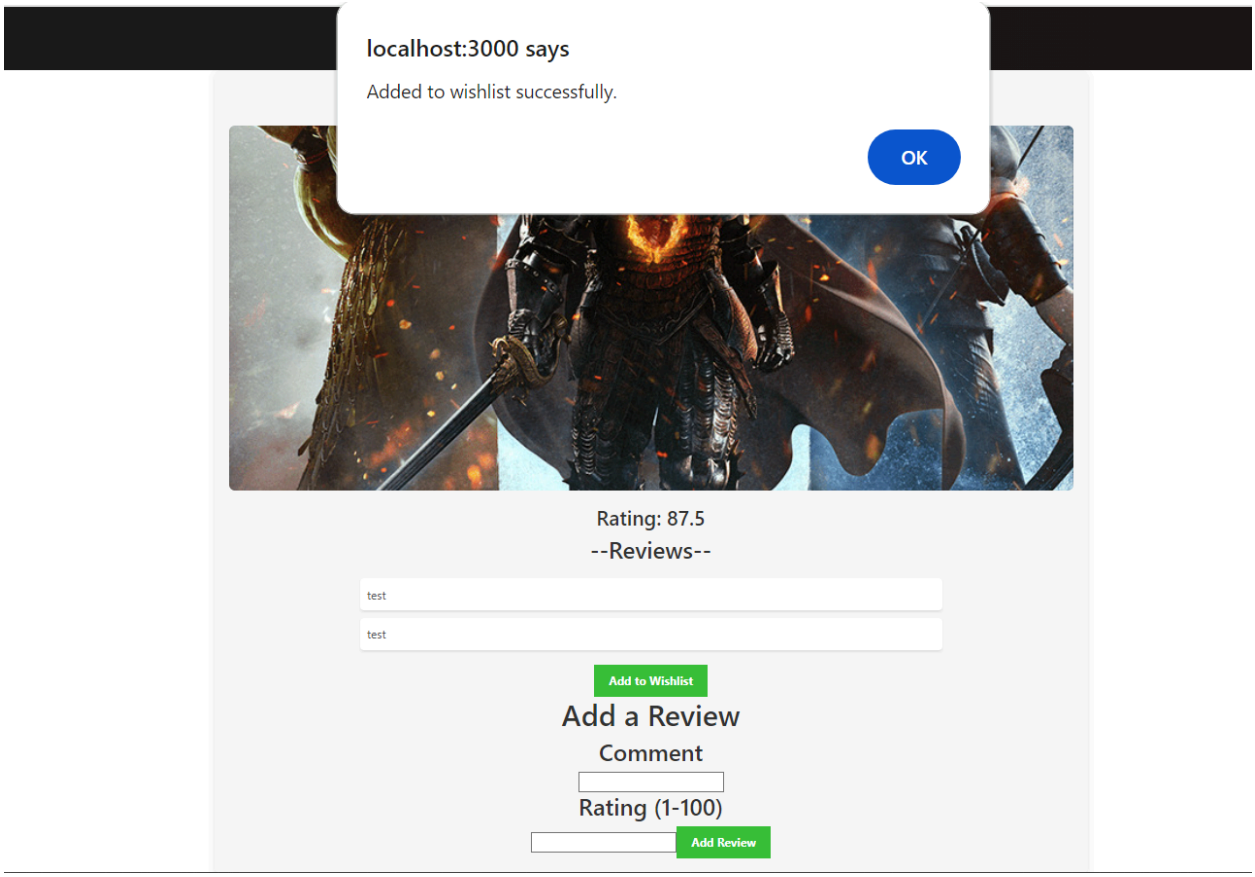


Information About a Game



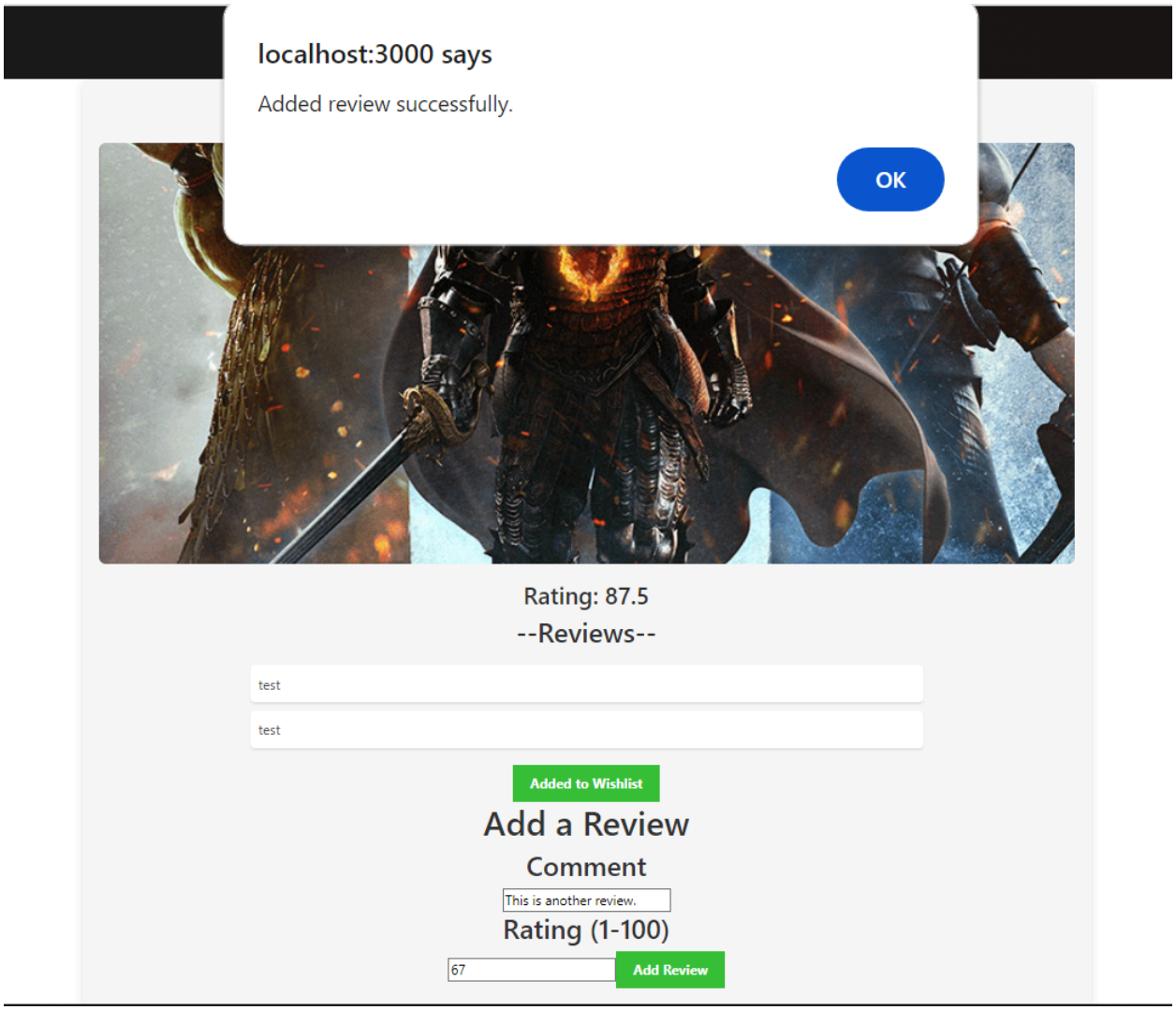
This shows the reviews and average rating for the game. The ability to add a review and comment are shown as well.

Adding to Wishlist

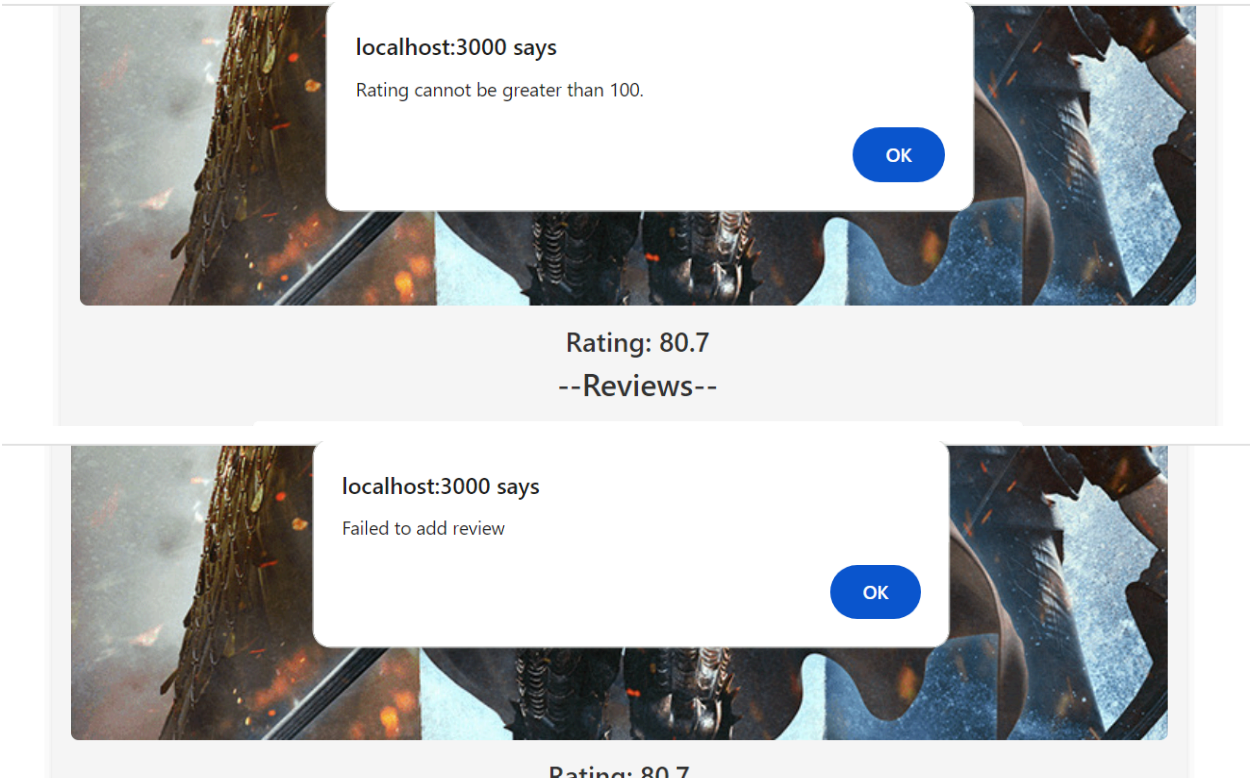


This shows the notification that appears when a game is added to the wishlist successfully.

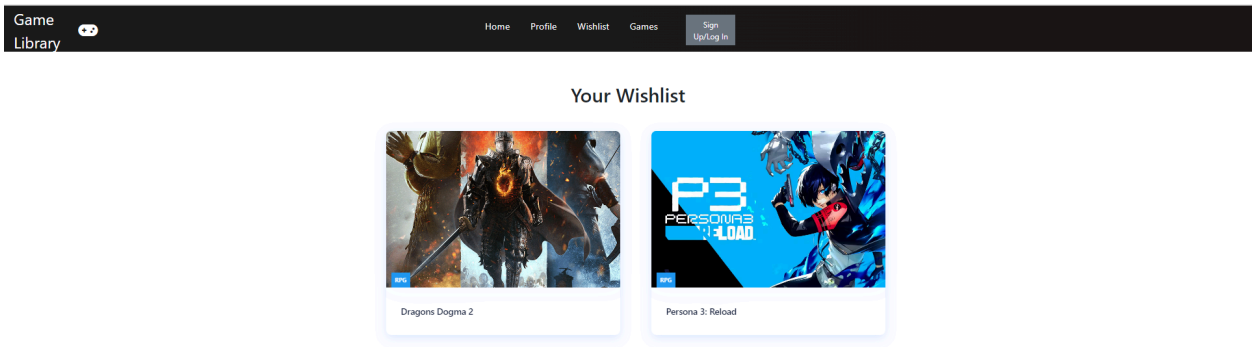
Adding Review



This is the notification that appears when a review has been added successfully. If the user attempts to put in text for the rating, it will not add it to the database as shown below. If the user tries to rate it over 100, it will also notify the user of the error, shown below as well.



Wishlist



This shows the wishlist for the current user. The wishlist will contain a card for all the games the user puts on their wishlist.

Contribution/Work Done By Each Team Member

Aaron: Worked on the individual game pages, the initial structure of the wishlist page, and tidying up game cards on the home page. Worked on the initial code for allowing users to add a game to a user's wishlist as well. Also set up the original schema for the database. Worked on the initial set up for the backend server.

Alex: Worked on the Login/Signup Functionality. Handled the overall user state management. Worked on Profile page and querying to get reviews and games within wishlist. Designed the overall architecture for the project.

Christian: Worked on setting up the home page for the website. In other words, created the buttons and styled them with Bootstrap, created the Cards that display the game on the homepage, created the Hero Section of the website, and created a navigation bar. Also worked on the UI for the login and signup pages. Set up the backend server to handle post requests for logging in and signing up. Also added a game page to display the games in the database. Also fixed the SQL queries to add reviews, wishlists, displaying user's wishlist and reviews. Fixed the tables for the database to ensure that it was in the 3rd normalized form.

References

Build a full-stack authentication app with react, node, express, mysql: Login, registration, Logout. YouTube. (2023, April 3). https://youtu.be/0i86B4mU-vw?si=t6rXhqUZQLOK3FUM

"Login and Registration Form Using REACT + Node + Mysql | Login and Sign up Form with Validation." *YouTube*, YouTube, 13 Mar. 2023, www.youtube.com/watch?v=F53MPHqOmYI.

"React Website Tutorial - Beginner REACT JS Project Fully Responsive." YouTube, YouTube, 11 Aug. 2020, www.youtube.com/watch?v=I2UBjN5ER4s&t=3126s.

YouTube. (2022, June 13). NextJS MySQL Example. get mysql data into a react app using node JS. YouTube. https://www.youtube.com/watch?v=aprLiG34b50&list=PLz2l9OPXFZr-8oH7tFHoIhbh6C9oYzo0G&index=21

YouTube. (2022, October 21). *REACT: How to get input value (Dynamic Text Input Field)*. YouTube.

https://www.youtube.com/watch?v=91TIUURx5JM&t=235s

YouTube. (2023, July 17). *React JS node JS Express add and fetch all data from mysql database | axios*. YouTube.

https://www.youtube.com/watch?v=_77ie-arQs4&t=710s