GamerQuest Website

PROJECT

Pattarapol Tantechasa | COS30043 Interface Design and Development | 21 May 2025

Introduction

GamerQuest is a social web application where users can discover new games, write reviews, add and manage their personal game library. It combines the features of a community review platform with a dynamic game tracking system, designed using Vue.js (frontend), PHP (backend), and MySQL (database). The app will be deployed on the university's Mercury server and MariaDB. As well as there are more technologies used in this project like Bootstrap for styling, Vuetify for form validation and Vue.Draggable.Next for drag and drop to change user's game status.

Main Functionality

- **User Authentication**: user can sign up for their personal account, log in to their account. Logged in user will see different user interface and able to use features like add game library, rate and review game.
- **Game Catalogue Browsing**: All users can browse a list of games fetched from an external RAWG Video Games Database API, search or filter by title and genre. As well as able to click link to learn more detail about the game.
- **Game Library Management**: a logged in users can add game to their personal library, organize them by status (Wishlist, Playing and Completed) using drag and drop interface.
- **Review System**: a logged in users can write review, edit and delete their own review for the game. As well as, all users can see all reviews for the game, including reviews that created by other users.
- **News Page**: a searchable and paginated news feed read data from a JSON file.
- **About Page**: basic information about GamerQuest, welcome message based on user input and interactive image toggle using radio button.
- **Home Page**: a welcome message, call to action button and carousel of images about different type of gamer.

Technical Components and Tools

Front-End:

- Vue 3 (composition API, vue-router, directives)
- Bootstrap and custom CSS for responsive layout
- Vuetify for form validation and accessible UI component
- Vue.Draggable.Next for implementing drag and drop interactions

Back-End:

- **PHP**: connect the website to database, handle request from client side to server side. (e.g. Account signs up or Add game review).
- **MySQL** relational database:
    - **MAMP** for local development environment
    - **MariaDB** for production environment
- **Postman API**:
    - RAWG APIs testing to see what response look like, easier development process.

Deployment:

- Uses **VITE** to build the project for deployment.
- Hosted on Swinburne's **Mercury** server
- Create and connected the deployed website to MySQL database on **MariaDB**

Highlight Features and Approaches

- **Use of External RAWG Video Games API**: The app integrates the RAWG API to fetch real-time data on popular games, allowing users to explore up-to-date game information. As well as allows to have less table on the database which avoid complexity during development.
- **Role-Based UI**: The interface dynamically adapts to whether a user is logged in or not and displays different features.
- **Environment Configuration with ".env" file**: Sensitive data and API URLs are stored in a .env file, making the development environment more secure, maintainable, and adaptable across setups.
- **Use of Axios over Fetch**: Axios was used instead of the native fetch API for its cleaner syntax, better error handling, and ease of working with JSON by default. Axios is widely used in the industry for REST API communication.
- **Clean Vue Component Structure**: Each Vue component follows a consistent structure using *<script setup>*, *<template>,* and *<style scoped>,* making the codebase easier to read and maintain
- **Postman API Testing**: During development, Postman was used to test and document all RAWG API calls. The API test collection was exported as part of the submission as "*GamerQuest_RAWG_PostmanAPI.json*"
- **Separation of Concerns**: Each PHP file in the project adheres to the single responsibility principle. Components, Views, and backend APIs functions are separated into their own folders, improving maintainability and scalability of the project.
- **Vuetify for Forms and Validation**: Vuetify components simplify the creation of input forms and make validation rules more readable.

- **SCSS for Custom Styling**: SCSS enables reusable class definitions, such as “bg-brown”, which integrates smoothly with Bootstrap and helps enforce consistent styling across the app.

Reflection

Some of challenges I faced during this project is the integration of PHP into Vue 3 project. I never work with full-stack website with this set of tech stack. The setup and development of this project is kind of complex when the project needs to be deployed on Mercury as well. On my deployment first try of this project that PHP version I used in the development and Mercury are different. Mercury got an older version of PHP; therefore, some of modern syntax or PHP built in method like “*password_verify()*” isn’t working on Mercury. Luckily, I’m able to solve this problem with an integration of compatibility library by including a “*password.php*” file from a community-supported source in the “*/backend/lib/”* directory. This taught me the importance of checking the server compatibility early in the development process and finding reliable fallback solutions when working with legacy systems.

I faced many errors on my PHP file because I have minimal experience of PHP. I watched many tutorials on YouTube that taught about PHP and MySQL. As well as, asking ChatGPT for some guide to solve some specific problems during my development. I used it to learn and get better understand of issues I faced, try to implement in my own way, which I believe it is the best way to use AI.