

CS 4850-01 Spring 2024
Video Game Blogging and Review Website
INDY-11



[Website Link](#)

[Github Link](#)

Lines of code: 852

Components: 16

Sharon Perry April 10, 2024
| Michael Roth | Matteo Staciarine | Jeremiah Nienburg |

Table of Contents

| | |
|--|-----------|
| Table of Contents | 2 |
| Introduction | 3 |
| Project Plan | 3 |
| Requirements | 4 |
| Design and Architectural Diagrams | 5 |
| UI Mockup | 7 |
| Development: | 11 |
| Testing Process | 14 |
| Conclusion | 17 |

Introduction

The video game blogging site "PixelText " was created for the purpose of giving gamers a place to read and write about player-side experience with games. As an indie team, the three members worked on separate parts of the project, mainly focusing on developing the experience through interface and database functionality. The website has been hosted through the Raspberry Pi 4 model B, the frontend code completed on Visual Studio, and the databases created on MariaDB. The website is self-hosted with a domain name that is to be bought. Features of the website include the ability to create an account, navigate through all the website tabs, navigate the games index, and post a personal review. User interface features for the necessary functionality of the website were implemented allowing users to navigate through all website implemented features smoothly. When creating an account, the user will open up access to the profile page of the website and have a permanent account for use. This account information is encrypted through Bcrypt; the majority of the functionality of the website requires users to create an account. The absence of an account doesn't allow users to post reviews or interact with any that exist.

Project Plan

The primary functions of our website include the ability for our users to create a personal account, which holds their unique username and chosen profile picture as well as a brief description. After account creation, a user can browse through a number of games listed on the website to find one that they are passionate about. Once they select a video game, they may create a personalized essay for their particular game as well as leave a ranking of that game if they choose. A given user will have access to markdown to add some extra functionality beyond a simple plain text document.

Various tools have been used in the planning and implementation of our website:

- **Draw.io:** Used for diagramming and mockups of the website and database layout.
- **Visual Studio Code:** the primary IDE used for coding our website in the following languages: PHP, HTML, and CSS.
- **Bcrypt:** used as a hashing algorithm to securely encrypt and store the user's password.
- **Github/Git:** A code sharing platform and distributed version control system used to store the website's code for collaboration between our team members.
- **Apache Web Server:** establishes a connection between the users and website in a client-server structure; compatible with major operating systems.
- **Secure Shell Protocol (SSH):** Used to remotely connect to the Raspberry Pi as necessary by our team members and make concurrent changes.
- **Remote Desktop Protocol (RDP):** Used to remotely connect to the desktop environment of the Raspberry Pi.

For hosting our website we decided to use a Raspberry Pi 4 Model B. This was chosen as it was already in the possession of our team and was the least costly investment. A number of factors had to be considered because of this choice. The security of the teams personal network, the ability for the server to run 24/7 without interference from outside circumstances such as power or service outages, physical negligence, notifying the entire team of any changes to the device, setting up SSH connection to the

device. To ensure that the device could fulfill these requirements and also be physically accessible, the following was done:

- The device is securely placed in a location that does not have any unnecessary traffic.
- The device is connected through an ethernet cable to the team members personal router.
- Private key encryption ensures only team members may remotely access the device.
- The device is connected to a secondary battery source that may supply the necessary voltage if a local power outage occurs. This is to ensure the device's internal storage does not become corrupted. Another device monitors if the Pi is receiving power.

Requirements

Our website will allow users to compose and publish reviews and blogposts using the website GUI. We refer to the system by which users will accomplish the task of content creation as the Post Creation and Publishing System (PC/PS). The PC/PS will require users to select a game to write about and will require the user to supply a title. Each post and reply is attached to a game or parent post and is displayed accordingly.

We will additionally allow for users to create and customize profiles for their accounts. The account creation process will consist of a user supplying their email, username, and password. After account creation, the user can then log into the site using their username and password. The password is encrypted using Bcrypt and stored into the database only after encryption. After account creation, the user will be able to access their personal profile page, where they can then edit account settings such as supplying a profile picture and creating an "About Me" section.

The website itself will feature a homepage by which the main components of the website - the games directory, one's profile, and the ability to view posts - are immediately accessible. A user shall be able to view data and user-generated posts about a game upon clicking its link in the games directory page. The homepage, outside of acting as a hub to the other parts of the website, will also feature popular posts automatically.

All user-supplied information, from account data to generated posts, will be stored permanently in a database. The user will be able to perform CRUD operations on their profile or any post using the website interface. All other operations will be locked behind administrative accounts.

The website will maintain functionality on all modern browsers, maintain accessibility standards, and use best security practices (encryption, safe storage of sensitive information, etc). Users should be able to easily navigate all the main features of the website, including the profile page, games, and the extraneous contact and about pages. Upon entering these pages, the user should be greeted with exactly the information they would expect upon entrance. The games page should provide a clear route towards users posting information about the games that they would like to review or speak about, the profile page should provide all information necessary for the user profile, the about page should state a summary of the website's purpose and how it came to be, and the contact page should provide a clear method for contacting the developers of the website. Each developer's role should be clear to understand for contact purposes on part of the user.

Generally, the user should be able to understand and enjoy the experience that the website provides visually and interactively. With the customization of their profile, there should be a sense of individuality and personalization that the website aimed to implement. All the operations for this was intended on part of the design and coding of the website's frontend and backend.

Design and Architectural Diagrams

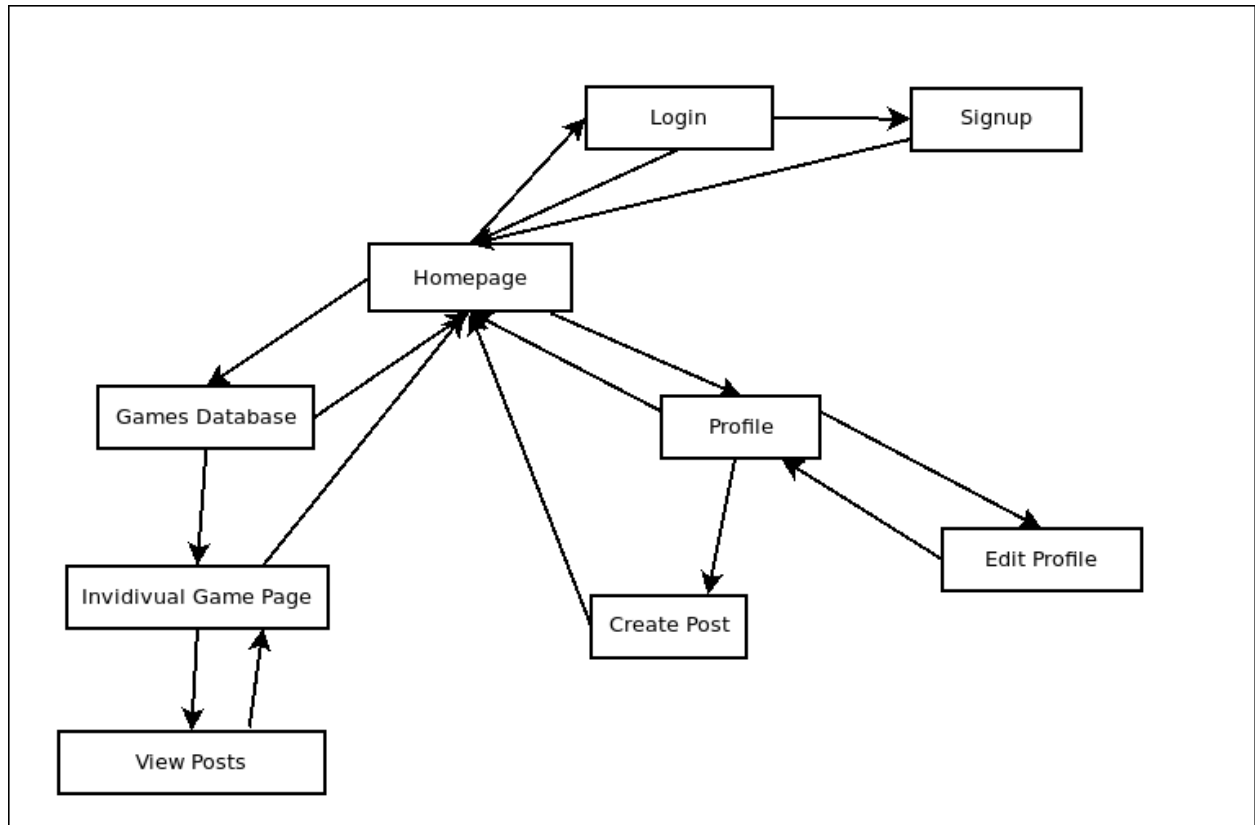
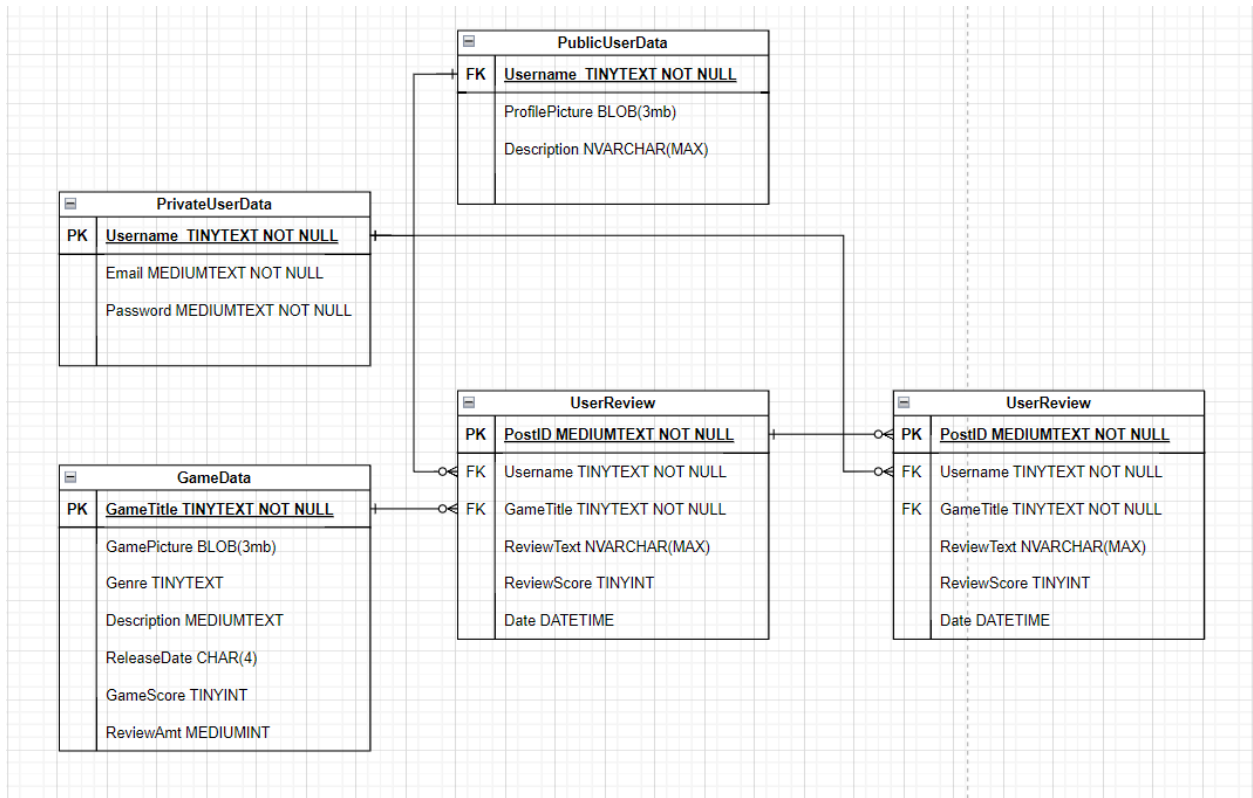
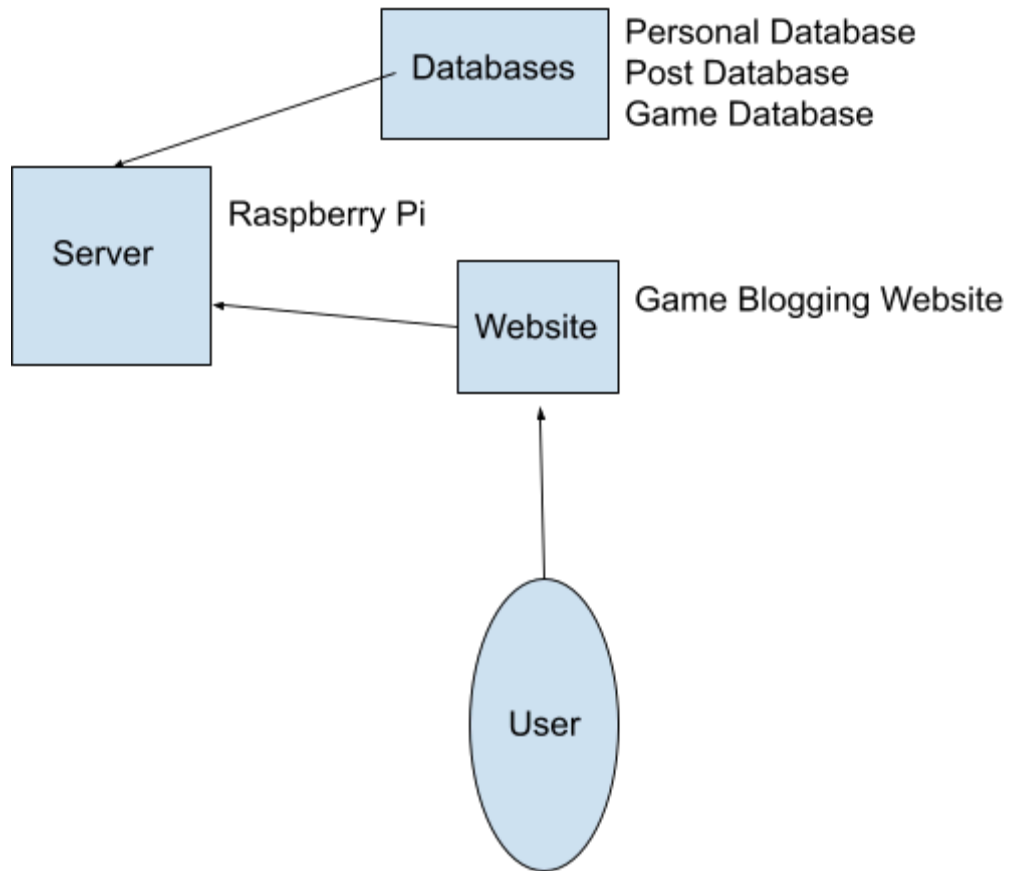


Fig. 4.1. State transition diagram that depicts how each page in the website is accessed and what it is linked to.





UI Mockup

Homepage

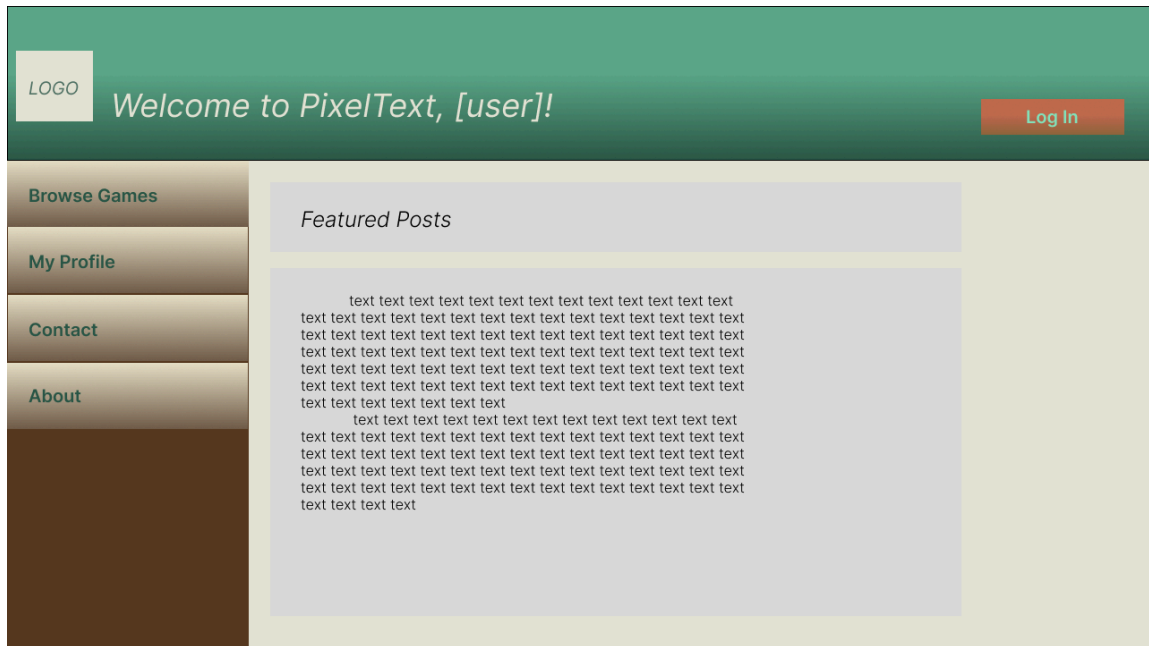


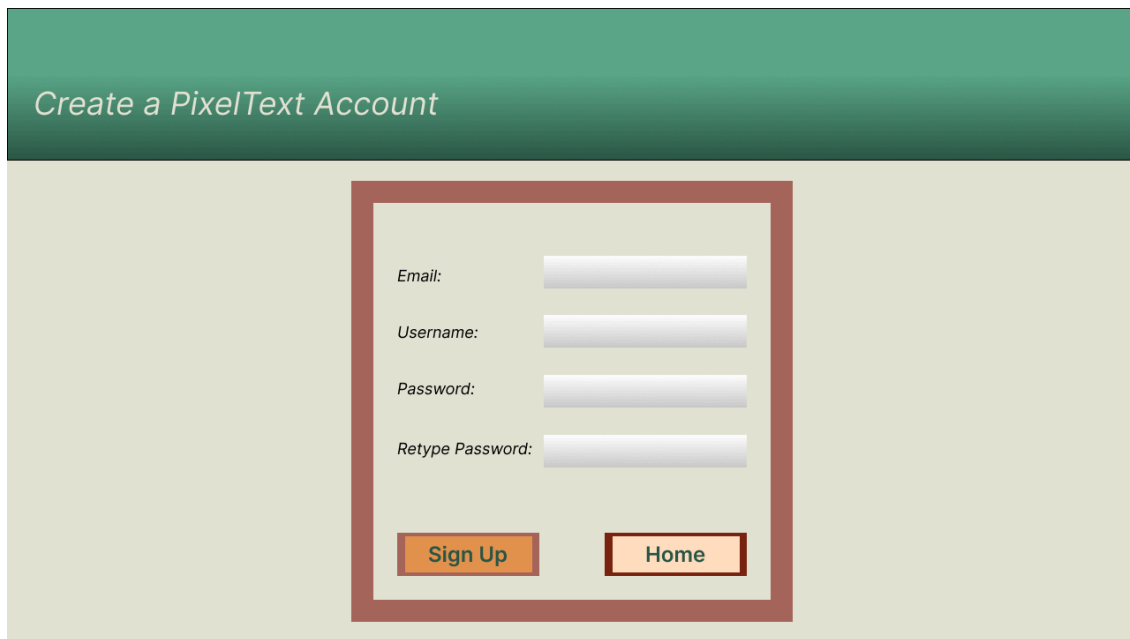
Fig. 5.1. Our original design for PixelText's homepage.

Login

The mockup shows a login screen layout. At the top is a dark green header bar with the word "Login" in a light green font. Below the header is a light beige main content area. In the center of the main area is a large, light gray rectangular box with a thick red border. Inside this box, there are three input fields. The first is labeled "Username:" and the second is labeled "Password:". Below the password field is a third input field labeled "Retype Password:". Below the input fields is a small red text label that says "Error Message". At the bottom of the box are two orange buttons: "Login" on the left and "Sign Up" on the right.

Fig. 5.2. Mockup of our Login Screen

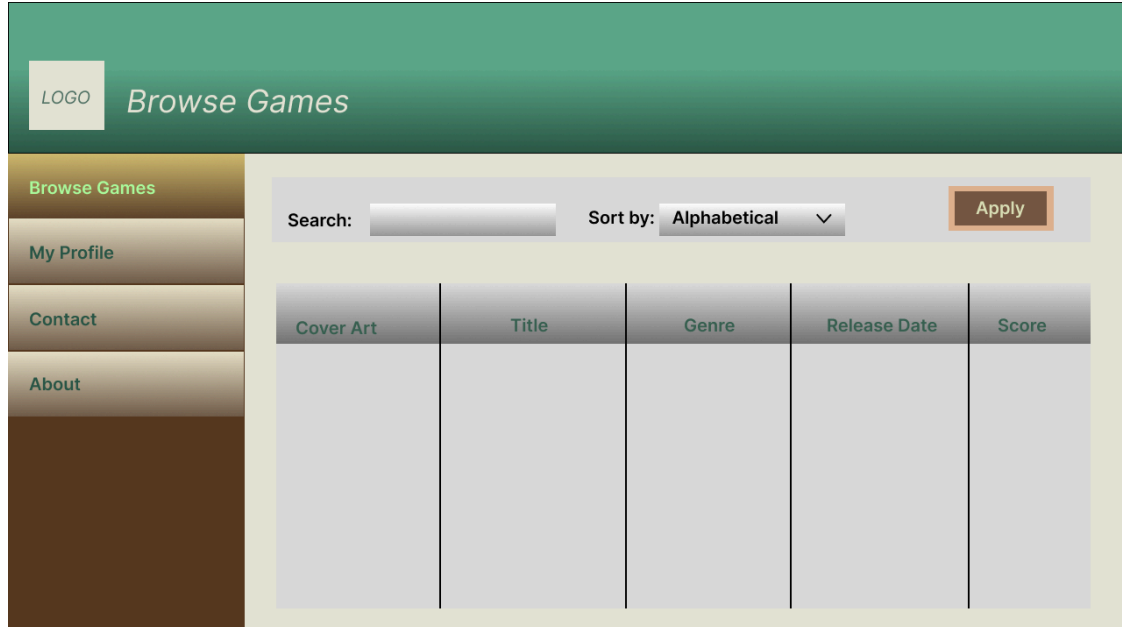
Signup



The sign-up screen features a dark green header with the text "Create a PixelText Account". Below the header, a light beige background contains a central registration form enclosed in a red border. The form includes four input fields: "Email:", "Username:", "Password:", and "Retype Password:". At the bottom of the form are two buttons: "Sign Up" (orange) and "Home" (orange with a red border).

Fig. 5.3. Mockup of the site's sign-up screen

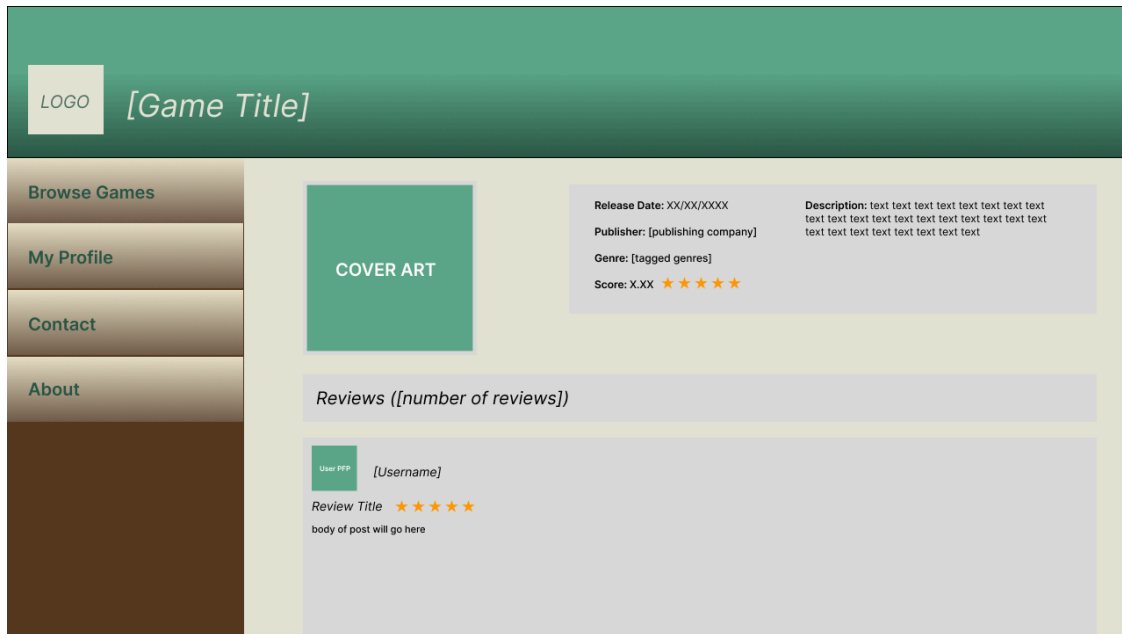
Games DB



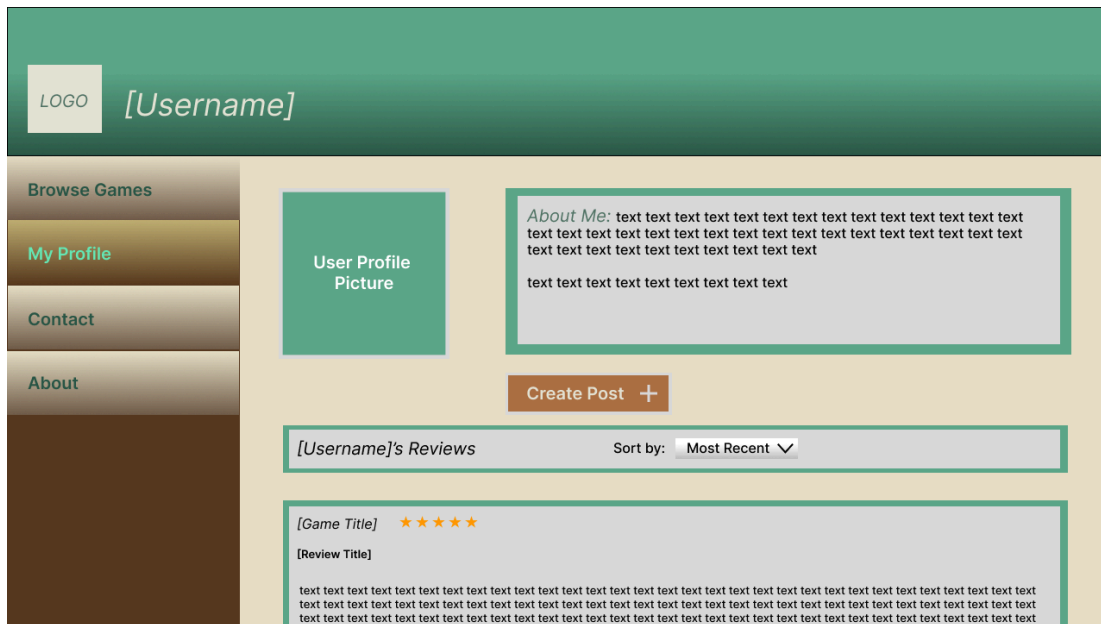
The Games DB screen has a dark green header with a "LOGO" placeholder and the text "Browse Games". A left sidebar contains four menu items: "Browse Games" (highlighted in green), "My Profile", "Contact", and "About". The main content area has a search bar, a "Sort by: Alphabetical" dropdown, and an "Apply" button. Below this is a table with five columns: "Cover Art", "Title", "Genre", "Release Date", and "Score".

| Cover Art | Title | Genre | Release Date | Score |
|-----------|-------|-------|--------------|-------|
| | | | | |

Game Infopage



My Profile



Version Control:

For version control we used the popular codesharing platform *Github* alongside the distributed version control system *Git*. *Git* was installed on all team members computers as well as the Raspberry Pi for distributed modification to our codebase. The primary website-code has two branches: Master branch, Testing branch. All changes made to the Testing branch once debugging and testing completed, would be moved to the Master branch. For the server-code of our website only one main branch is used as there is only one developer working on this portion of the website.

Development:

At the beginning, there was a big collaboration between all of us to plan the project together. A lot of the architecture, UI planning, and general putting-together of the framework was divided up amongst all three groupmates. Each of us had a number of elements that we needed to discuss and agree on before they were declared as a part of the final product. When it came to the project, there were many changes we had to agree on together because of us learning new material that was either more efficient, easier, or looked better. The collaboration on github was essential to putting the elements of the project together on part of the coding especially. The assets created were also transported easily over github, easing the process for this immensely.

When designing the website, we tried to keep the number of interconnecting frameworks that were used to run and manage it to a minimum. Throughout the actual development process, we have mostly stuck to this principle, but we have also found tools that were better suited to our website's needs and libraries that completed complex client-side tasks in a few lines of code.

The main framework we used to build the frontend is a standard for most websites one comes across: page layouts were created in HTML, styling was done in CSS, and client-side logic was done using Javascript. Since our website makes extensive use of a database, we also decided to take advantage of the jQuery library for Javascript to assist with the automatic loading of database information into a site. Due to the basic nature of our planned user interface motifs, pure HTML and CSS have been enough to achieve the look and feel we wanted.

Server-side development underwent a tumultuous process throughout the software development life cycle. Initially, we planned to use Node.js for server-side operations such as CRUD (create, read, update, delete) operations. However, after conducting more research into PHP and examining how a lot of the complex functions for database management were provided in its default libraries, we made the decision to use PHP and remove Javascript from our server-side codebase entirely. An example of the rationale for the switch includes the issue of password storage. PHP provides an easy-to-use command called `password_hash()` that uses Bcrypt to hash and salt password strings before storage in our database, whereas Javascript would require an external library. We used MariaDB as our database management system due to a member's familiarity with MySQL (and MariaDB's similarities to said software). Finally, we are using Apache Web Server as our server software of choice to make our website accessible from the internet. We chose to use Apache for its documentation, ease of configuration, and the fact that it is readily available from Debian's repositories (which was a factor when choosing all of our server-side tools).

We used a plethora of developer tools to create our website and to document all changes to our codebase. We used git to make changes to our codebase while retaining our code's history, and we used

GitHub as our version control system to host our codebase due to its accessibility for new and seasoned users. Since a team member owns the hardware we are hosting our website on, any team member that wished to access the server side to make changes needed to do so via a remote connection. We opted to use Secure Shell (SSH), as it is a standard protocol among modern computers (even in our Raspberry Pi's arm64 architecture), and the setup process is made simple using GUI interfaces such as PuTTY. Additionally, while our lack of GUI prevented us from testing code (a browser would be necessary), operations for writing code were entirely doable with a command line. For testing, our team member who had physical access to the Raspberry Pi tested all code pushed to it in Mozilla Firefox.

The main idea to build a website centered around reviews and blogs about video games was agreed upon and initially conceived by the team leader. The team was anxious to test out some parts of development that were either new or completely foreign to our team's skills. A few other ideas were presented, however the idea to create a review and blog site for people who liked writing about video games became the frontrunner going into the project.

As none of the team had experience in creating a website, research for tools and essential programs was paramount to steady development. An initial high-level understanding of what the site should accomplish was drafted by the team leader.. Our team leader focused mostly on writing out software requirements specification (SRS) while the other two members focused on logistics in the software design document (SDD). For the SRS, functional and nonfunctional requirements were drafted and light research into the necessary resources for those requirements was done. This was an important action that saved lots of time during development as it let the team know what was possible during our given timeframe.

For setting up the Raspberry Pi 4, A Linux distribution with a basic command line environment, which would later be changed to a full desktop environment for front-end testing purposes, is flashed to the Raspberry Pi. The Pi is then configured to connect to the internet automatically during boot. The ports necessary for development included 3306 (MariaDB), 80 (HTTP), and 22 (SSH). The Pi is remotely connected to using SSH for initial interfacing. For security, an initial username and password were set up for access to the Pi. This was later deemed too insecure and instead a key pair was generated for each of our team members. The database management system of choice was initially MySQL but upon further research, MariaDB (A fork of MySQL) was chosen for its marginally faster query times and other performance improvements.

The database was set up with a unique password and an initial user for setup. This initial user would have access to all portions of the database for creating the database. Multiple different tables were set up under the "websiteDB" database. The only distinction of note is that some portions of the data are to remain private and encrypted. For instance, when a user initially sets up their account, their password will be encrypted and stored in the database. As a second layer of security, a second user is created that does not have access to the tables that have sensitive information pertaining to the users.

After we completed the planning phase of the project, we focused on setting up our development environments before formally beginning extensive work on building our frontend. As we are self-hosting our website using a Raspberry Pi, the issue of collaboration was compounded upon due to the fact that only one person has local access to the machine. Our backend developer used an SSH connection between the Pi and personal desktop PC, and mostly wrote and debugged server code over the SSH connection using Vim. Some assistance was offered to set up an SSH connection to a team member's Pi using PuTTY, and later configured authentication with SSH keys since, in real-world use cases for SSH, passwords are being phased out in favor of keys.

When the development process began, there was immediate deviation from the planned server code.. While writing server code using Node.js was a fine choice, many of the tools we would need for our website are bundled in PHP, whereas an external library would need to be installed using Node.js. The decision to swap server-side code to PHP very early on was ultimately one of the most influential decisions made during the course of the project. PHP includes a Bcrypt function for hashing and verifying passwords, so what was anticipated to be a difficult part of the project became incredibly easy to implement. Similarly, the usage of a library to accomplish the task of using a .env file to store database credentials was easily done in PHP. By the time our team showcased our prototype presentation, our back-end developer felt comfortable among many facets of web development, and I had learned more about client-server design and Linux system administration than initially anticipated.

For work in the front end a particular style had to be decided on. For inspiration, websites in the 2000's/1990's that aspired for homely, personal design which reflected the values and practices of the time period. There was an emphasis on personality shining from the users, and the website would similarly reflect that kind of personality. With that said, the final product had some of these ideas toned down for the sake of a healthier and simpler user experience, but the design of the elements which make it up still have this flare.

Having drawn the art used for clickable entities on the website. The color scheme wasn't initially thought about, but going with the cream color was initiated by our team leader, and the decision was made to offset this with a dark blue. The result was a clean, warm looking webpage that expressed not the smoothness of newer websites, but a style reminiscent of the early internet. All buttons came with a slight animation to show when hovering over them. Each of the elements on the homepage which are clickable reacts to a mouse hover with an animation, and the sidebar navigation highlights whenever a mouse hovers over each element. The logo is a clickable home button along with the home button for easier navigation to the home page for users who would like to navigate to any other part of the website. A lot of the elements on each page had to be manually layed out for the cleaner look we achieved on the website, where elements would be adjusted to the margins by seemingly appropriate values. The favicon was a quick reduction of the initial drawn logo that was added quickly along with the header bar that included the login option.

Testing was difficult to do as there were difficulties in how we could communicate remotely and the modification of code on a bare command line interface. This is when the decision to install a full desktop environment onto the Raspberry Pi was made. Due to the limitations of the ISP's of our team, remotely connecting to the desktop environment was not an option. Therefore all of the testing when it came to the Raspberry Pi was locally via instruction from our developer. Because the developer of our website could not locally test the code on the Pi, screenshots of the relevant errors were exchanged until the given error could be solved.

With the contact and about me page, each element was similarly layed out manually. The references from the UI mockup were essential for giving reference to how the user experience would be at a skeletal level. As the website was designed, there were some changes that came along with general layouting and design decisions that felt superior to what the plans intended.

Through this project, the team has acquired an increased knowledge of Linux system administration, database entry and interpretation, web server hosting, and much more. While some team members were quite comfortable using Linux and have been learning about how to use Linux environments casually, this project pushed us to learn the ins and outs of the provided tools and not just how to use said tools. Because we had to configure SSH, Apache Web Server, and MariaDB from the

command line using config files, a deeper knowledge of the system was necessary. Navigating the Linux filesystem to make full use of the software was incredibly important.

Testing Process

The testing process of our website took place in multiple stages. The first stage is writing the code for a new module of our website. The code would be pushed to the testing branch of our github repository. This would be locally tested on the developers system to check for errors on their personal system. These errors would include general syntax errors or basic functionality testing. The test branch would then be pulled to the Raspberry pi. The new code would then be tested for errors locally on the Raspberry Pi. This testing would be to thoroughly test the system given a reasonable application of scenarios. First the individual module would be tested for its full functionality, then entering and exiting the module from the other linked modules. If no errors are spotted on the testing of the Raspberry pi, then the branch would be merged to the master branch.

A number of difficulties arose during the testing of our website. There was difficulty remotely connecting to the Raspberry pi and the limitations of a terminal interface when attempting to debug code. The ISP's (Internet Service Providers) of our team members were limited in the ports they were able to connect over, thus all attempts to remotely connect the Raspberry Pi's local desktop environment were unsuccessful. Due to this, anytime the testing of a particular module needed to take place on the Pi, the testing had to be done locally rather than handled remotely. This caused development time to increase due to the varying schedules of the team members and when a module could be tested and approved.

Test Report:

| Case | ID | Pass |
|---|--------|------|
| The system shall allow the user to select a game upon creating a new post. | PCPS-1 | X |
| Games shall be searchable via either ID or title if the user types into the "Game" field of a new post. | PCPS-2 | |
| Text written into the post creation field shall be Unicode characters. | PCPS-3 | X |
| The post creation field shall support the formatting language Markdown. | PCPS-4 | |
| The basic tools Markdown provides (bold, italics, underlining, and creation of headings) shall be made accessible via button elements. | PCPS-5 | |
| The post creation field shall display Markdown formatting in real time (e.g., text should appear as bold instead of **bold** in the post creation field). | PCPS-6 | |
| The text within the post creation field shall be given an ID, relevant information (such as game ID/post ID if post is a reply) upon clicking the "Publish" button. | PCPS-7 | X |

| | | |
|---|--------|----------|
| A post shall be made accessible using the following URL template: https://www.[finalsitename].net/posts?id=[postID] | PCPS-8 | |
| Upon clicking the “Delete” button under a post the user owns, the DBMS shall search the post database for a matching post ID and delete the post from the database. | PCPS-9 | |
| The system shall include a login/signup page where users can submit username and password information to access their accounts, and new users can submit username and password information to create an account. | PM-1 | X |
| Password information shall be hashed using Bcrypt before storage inside the profile DB. | PM-2 | X |
| If there is a mismatch between the provided password and a password from a user’s associated account, an error shall be printed. | PM-3 | X |
| If the user clicks the “Forgot password” link, the system shall send a recovery link via email to the user’s inbox. | PM-4 | |
| The user shall be redirected to the website’s signup page if the “Sign Up” button is clicked. | PM-5 | X |
| The user’s provided email and encrypted password information shall be stored inside the profile DB - along with a default profile template - once the user has completed the signup process by clicking the “Sign Up” button. | PM-6 | X |
| The user shall be directed to the website’s homepage upon successful login. | PM-7 | X |
| The user shall be directed to their personal account if they click the “My Account” button. | PM-8 | X |
| The user shall be redirected to the “Customize Profile” section of the Preferences page upon clicking the “Customize” button. | PM-10 | X |
| The user shall be directed to the site’s homepage upon successful login. | GSR-1 | X |
| The homepage shall contain the top two blog posts of the website under a div element called “Featured.” | GSR-2 | |
| The “Featured” element shall contain a hyperlink that redirects the user to the website’s “Featured” page. | GSR-3 | |
| The Featured page shall contain a curated list of the top 20 posts on the website that are one week old or newer. | GSR-4 | |

| | | |
|--|--------|----------|
| The homepage shall contain a hyperlink that redirects the user to the list of games to view. | GSR-5 | X |
| The game list page shall contain 25 elements per page. | GSR-6 | |
| The game list shall be sorted by a hidden metric called “points.” | GSR-7 | |
| The hidden metric “points” shall be determined by multiplying average user score with the amount of user reviews. | GSR-8 | |
| The user shall be redirected to the Preferences page upon clicking the “Preferences” button. | GSR-9 | X |
| The “Preferences” section shall contain a section where users can customize their profile (see PM-10). | GSR-10 | X |
| The “Preferences” page shall contain a General section where users can access basic options such as notification settings, toggle dark theme, and text size (full list TBD). | GSR-11 | |
| The “Preferences” page shall contain a Security section where the user can change their password and set their profile visibility (full list TBD). | GSR-12 | |
| The profile database shall create a new entry when a user with a unique email signs up for the service. | DBR-1 | X |
| Each user inside the profile database shall be given a private user ID. | DBR-2 | X |
| The password element inside the database shall be encrypted using Bcrypt before storage. | DBR-3 | X |
| An entry within the profile database shall be removed if the user chooses to delete their account. | DBR-4 | |
| All entries in the post DB associated with a user ID to be deleted shall be deleted. | DBR-5 | |
| An entry within the post DB shall be created once a user publishes a post. | DBR-6 | X |
| An element within the post DB shall reference a user ID and either a game ID if a new post or another post ID if a reply. | DBR-7 | X |
| If the user edits a post and saves changes, the entry within the post DB shall be updated to reflect the changes. | DBR-8 | |
| If the user saves changes to settings pertaining to their profile or preferences, their entry within the profile DB shall be updated. | DBR-9 | X |

Conclusion

During our undertaking of building a content-oriented review website from scratch, our team has branched into a field unfamiliar to all members before - we each learned quite a bit about web development. The project has taught us about selecting the right tools for the job, Linux system administration, and implementing secure development practices. From the setting up of the Raspberry Pi, to the database development, to the implementation of the website elements, to the linking between the two, to the hosting of the website in general, we put in a lot of work to the breadth of the website. We faced many challenges with this novel form of development. There were many elements we were unsure of and unfamiliar with at the beginning of development, and one of the biggest challenges was linking the database and frontend of the website for user input. Conquering this challenge was a huge feat on our part, and there was a lot learned through that process. Outside of any technical skills gained in the area of computer science, we have each gained insight on the best way to tackle major projects. The senior project is a challenge conducive to real-world cases of our line of work; while it has been very difficult at times, we also feel massively rewarded and are proud of the work we put in.

For the role of team leader, learning how to take charge of a large project was initially very difficult. My “leader” role pushed me to work hard alongside my teammates just so I would not assume the role of being a “boss” by giving myself all the credit for simply handing out work for others to do (I had seen my fair share of that when pursuing a software engineering minor). I elected to take a more laissez-faire approach by letting my teammates find their niches and assign roles to themselves, which has been both a blessing and a curse. It has been a blessing that I have wonderful teammates I am proud to work with - a blessing that there have been no real altercations in regards to division of work. However, it is a curse in regards to the fact that I have felt a bit directionless myself since I wouldn’t set clear goals for everyone. As deadlines were fast approaching and I felt as if the project was not in the place it needed to be, I learned I had to start setting goals for not just myself, but my entire group. I hate this process myself, but once I started setting clear goals, I feel like we avoided crunching ourselves nearing the end of our development period before our presentation. While it has been an enlightening experience, I will leave the leader role to those who are born leaders in the workforce.

With all this said, I look back on how far we’ve come as a group and believe that, no matter the grade or the state our final product is in, we came out of our senior project twice the developers we were when we came in. I felt really out of my depth in the beginning, and I still do feel out of my depth, but I recognize that feeling out of one’s depth is growth. The hardest thing about computer science is that there are entire fields I have never touched and know nothing about, and web development was one of those fields. Now, while I may not be a truly experienced web developer, I at least know a few more things, which is more than before.

Going into this I did not have a very clear grasp on what we were partaking in. I had never actually hosted or helped create a website. While I did volunteer for the database portion of our website, it should be known that I did not do very well in my database oriented classes. A lot of the things related to the Raspberry Pi felt very niche and exclusive to the device so much of my time was spent researching navigating a Raspberry Pi and MariaDB. Self hosting was a decision we made for the challenge of it. It definitely slowed development more than using an already established web hosting service.