WASHINGTON STATE UNIVERSITY
VANCOUVER

CS 320 Course Project Final Report

for

Retro Game Exchange Database

Prepared by

Group Name: Team 22

Logan Tan	11631408	Logan.tan@wsu.edu
John McEchron	011749059	johnathan.mcechron@wsu.edu
Brandon Huang	011725342	brandon.huang@wsu.edu
Daniel Lee	011546543	daniel.lee4@wsu.edu

Date: 12/16/2020

1 Introduction

This project describes an online system to support the sharing and exchange of Retro Video Games. This system is intended for users to share their collections, search for games by different criteria, and exchange games with other users. Since this system is intended for Retro Video Games it does not cover the current or future generations of console games and also does not include online, digital, or mobile games that lack physical media to share.

1.1 Project Overview

The purpose of this system is to provide an online portal for individuals to showcase their collections of Retro Video Games and Systems, and to allow others to view, search, and exchange those games. The system will allow users to input details about their collections to identify which games and consoles they own. The system will allow users to search for games based on different criteria like Release Console, Release Year, and Genre to find Retro Video Games that they may be interested in playing. The system will allow users to exchange games that other users make available to share and track which games have been borrowed. The goal of the system is to provide a hub of interaction for Retro Gaming enthusiasts to show off their own collections, learn more about the games, and share them with other enthusiasts to attract new fans of these classics.

The scope of this system will be limited to Video Games that were released in the United States for 20th Century platforms including the Atari 2600, ColecoVision, Commodore 64, Mattel Intellivision, Nintendo NES, Sega Genesis, and the Sony Playstation. This system will exclude newer generations of games for consoles released in the 21st Century like the Nintendo Wii, Xbox, PS3, iOS, and other Digital Platforms. This system will also exclude support for sharing any re-releases of Retro Games for newer platforms, limited-edition versions of games, games released in markets outside the US, and games released in non-English languages.

1.2 Definitions, Acronyms and Abbreviations

AWS = Amazon Web Services
C64 = Commodore 64
Cart = Video Game Cartridge
GCP = Google Cloud Platform
HTTPS = Hyper Text Transport Protocol Secure
NES = Nintendo Entertainment System
PS1 = Sony Playstation

1.3 References and Acknowledgments

Information about the Video Game Consoles, Titles, Genres, Developer, Publisher, and Release Dates will be collected from the following sources:

[1] Wikipedia, *List of Intellivision Games,* 18 August 2020. Accessed on 05 November, 2020. [Online] Available: https://en.wikipedia.org/wiki/List_of_Intellivision_games

[2] Wikipedia, *List of Atari 2600 Games,* 29 October 2020. Accessed on 05 November, 2020. [Online] Available: *https://en.wikipedia.org/wiki/List_of_Atari_2600_games*

[3] Wikipedia, *List of ColecoVision Games,* 31 August 2020. Accessed on 05 November, 2020. [Online] Available: *https://en.wikipedia.org/wiki/List_of_ColecoVision_games*

[4] Wikipedia, *List of Commodore 64 Games,* 04 August 2020. Accessed on 05 November, 2020. [Online] Available: *https://en.wikipedia.org/wiki/List_of_Commodore_64_games*

[5] Wikipedia, *List of Nintendo Entertainment System Games,* 02 November 2020. Accessed on 05 November, 2020. [Online] Available: *https://en.wikipedia.org/wiki/List_of_Nintendo_Entertainment_System_games*

[6] Wikipedia, *List of Playstation Games (A - L),* 01 November 2020. Accessed on 05 November, 2020. [Online] Available: *https://en.wikipedia.org/wiki/List_of_PlayStation_games_(A%E2%80%93L)*

[7] Wikipedia, *List of Playstation Games (M - Z),* 01 November 2020 Accessed on 05 November, 2020. [Online] Available: *https://en.wikipedia.org/wiki/List_of_PlayStation_games_(M%E2%80%93Z)*

[8] Wikipedia, *List of Sega Genesis Games,* 05 November 2020. Accessed on 05 November, 2020. [Online] Available: *https://en.wikipedia.org/wiki/List_of_Sega_Genesis_games*

[9] Wikipedia, *List of Sega Master System Games,* 05 November 2020. Accessed on 05 November, 2020. [Online] Available: *https://en.wikipedia.org/wiki/List_of_Master_System_games*

Code written for this system includes publicly-available libraries from the following sources:

[10] Open JS Foundation, *Node.js version 14.15.1*, 16 November, 2020. Accessed on 21 November, 2020. [Online] Available: https://nodejs.org/en/

[11] Open JS Foundation, *Express version 4.17.1*, 25 May, 2019. Accessed on 28 November, 2020. [Online] Available: https://expressjs.com/

[12] JQuery Foundation, *RequireJS version 2.3.6*, 5 April, 2020. Accessed on 5 December, 2020. [Online] Available: https://requirejs.org/

[13] Evan You, *VUE.js Framework version 2.6.10*, 19 March, 2019. Accessed on 28 November, 2020. [Online] Available: https://vuejs.org/

[14] Marcos Moura, *VUE Material for VUE.js version 1.0.0-Beta-15*, 13 August, 2020. Accessed on 28 November, 2020. [Online] Available: https://vuematerial.io/

[15] Daniel Stenberg, cURL version 7.73.0, 13 October, 2020. Accessed on 28 November, 2020. [Online] Available: https://curl.se/

2 Design

2.1 System Modeling

2.1.1 Use Case Diagram

Unchanged from the SRS submitted with Milestone 1.

2.1.2 Activity Diagram

Unchanged from the SRS submitted with Milestone 1.

2.1.3 Sequence Diagrams

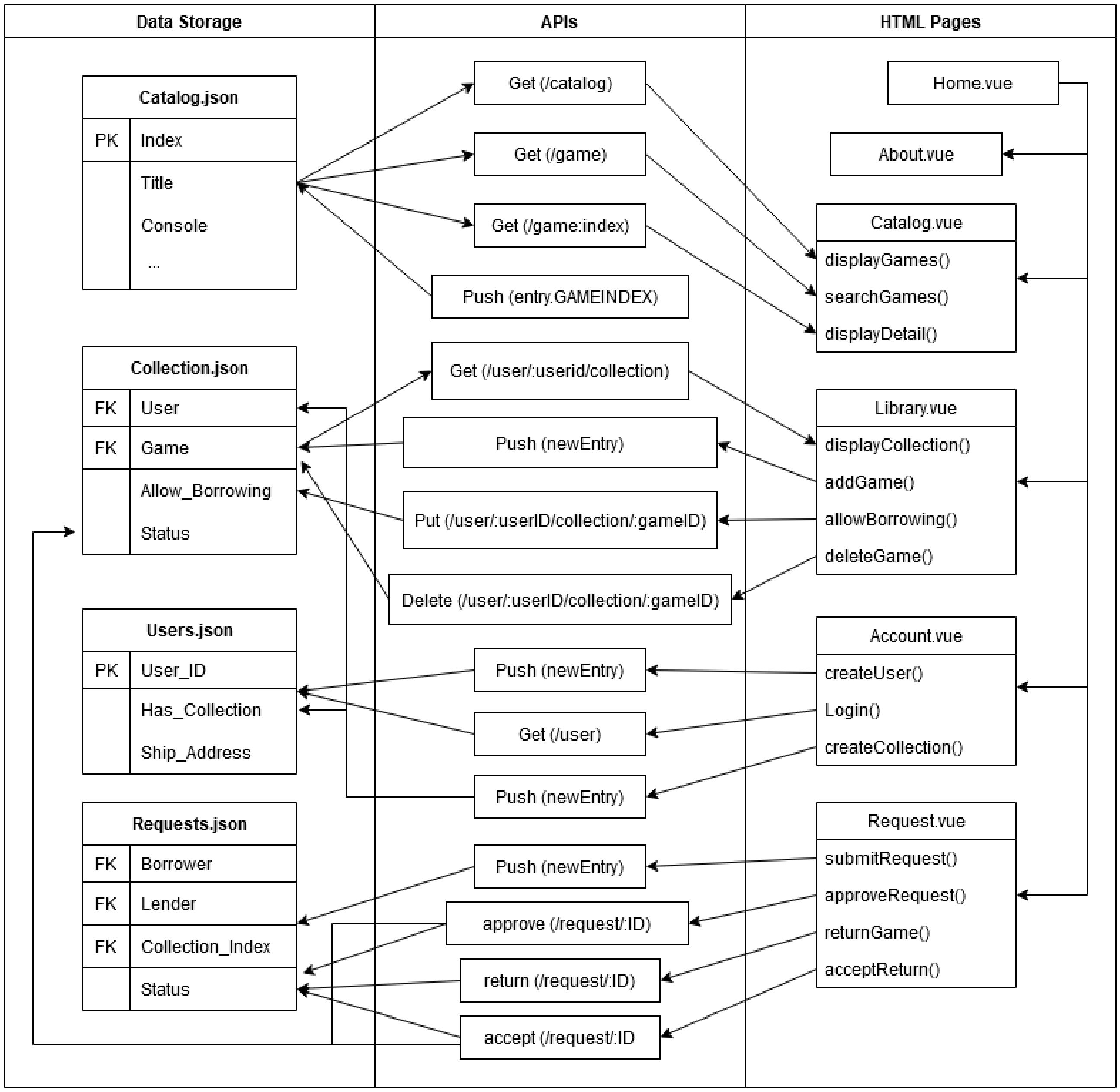
Unchanged from the SDD submitted with Milestone 2.

2.1.4 Class Diagram

Unchanged from the SDD submitted with Milestone 2.

2.1.5 **API Diagram**

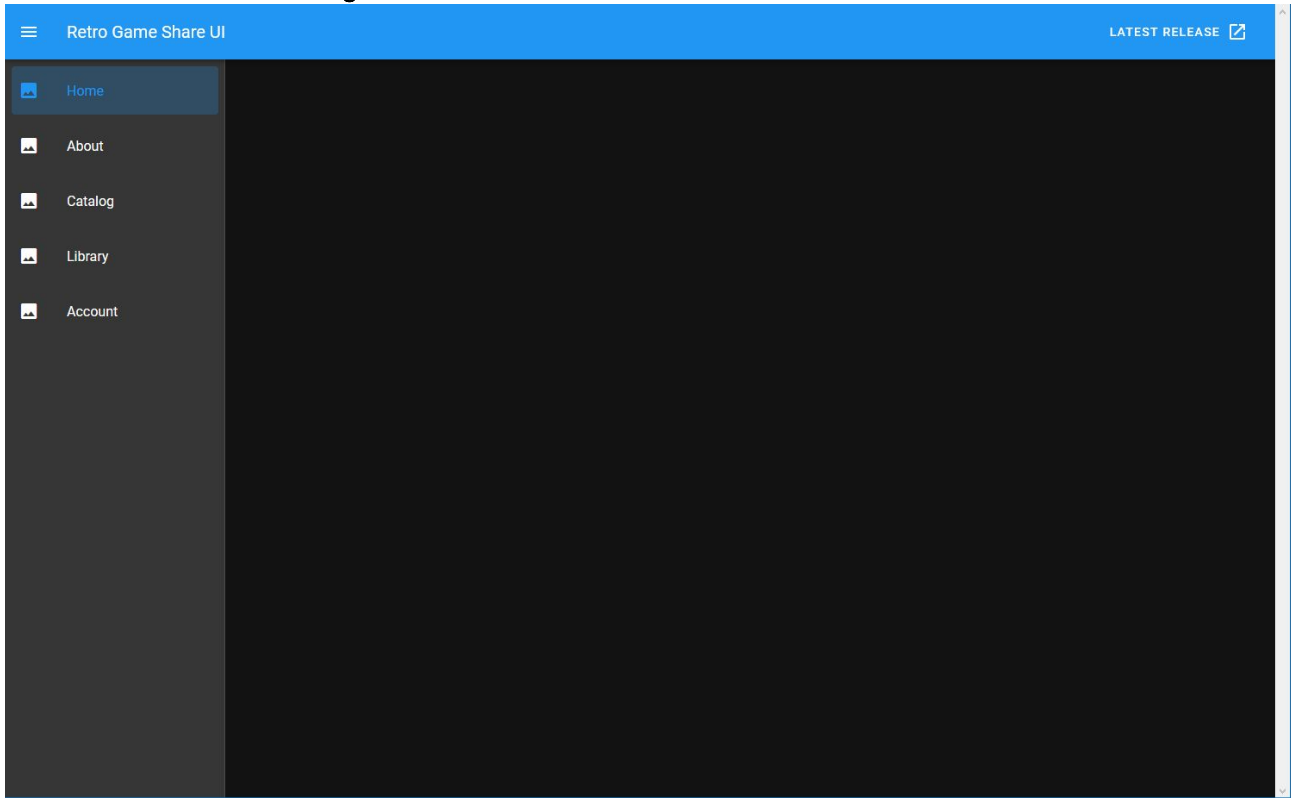
During the implementation of our system we decided to use API calls to separate the Data stored in JSON files from the HTML Views presented to the user. The diagram below was not included in our MIlestone 2 submission of the initial SDD.



2.3 Interface Design

Our Interface Design work began with the mock-ups below intended to outline the different pages we would need to build. We selected the VUE.js framework to help with the implementation of these pages and their respective methods to get, push, and display data.

Common Framework Design:



Home - provides introduction and links to each page

About - provides more information about the application and links to user-facing documentation

Catalog - provides access to the entire Retro Video Game database and Search functions

Library - provides access to User Collections of games and related functions to manage the user's collection, submit a request to borrow, and manage existing requests

Account - provides access to user-related functions like Login, Create Account, and Manage Account

3 Implementation

3.1 Development Environment

The Retro Game Exchange system classes were developed primarily in JavaScript. We used publicly-available JavaScript libraries including Node.js, Express, and RequireJS to aid in implementation of our class functions. Development work was primarily done in the Visual Studio Code IDE and IntelliJ IDE. Version Control and code sharing among team members was provided by GitHub.

The Retro Game Exchange interface was developed in HTML using the publicly-available VUE.js and VUE Material libraries. The publicly-available Express.js and Require.js libraries were also used to develop the APIs that link the online interface to the stored data. Front-end development work was primarily done in the Visual Studio Code IDE.

3.2 Task Distribution

Tasks were distributed through the Agile method of self-selection so each team member could work on elements of the project that aligned with their skills. Each team member contributed to the documentation delivered for Milestones 1 and 2, but Milestone 3 introduced more specialized needs for team members to focus on specific areas:

Daniel led the selection of design frameworks we used for the system, including the selection of VUE.js for the user interface and the use of Express.js and Require.js for the API calls.

Brandon wrote a first draft of the JavaScript methods that would be used in the interface and wrote the API using Express.js, Require.js and Node.js

Logan arranged and recorded notes for most of the team meetings and made the mockup for the front end UI. Along with organizing the and maintaining the Github.

John gathered information to build the game catalog and other JSON data files and contributed to the Final Report document.

3.3 Challenges

The primary challenge we faced was a lack of familiarity with developing web-based software applications that were required for this project. Some team members had experience in specific areas like front-end or back-end development, but none possessed a complete knowledge of the different skills the project required. Due to the amount of time required to learn these new skills we did decide to drop some requirements, such as the use of Google Authentication or OAuth to validate and track users that create accounts. We chose instead to use a more simple object-based approach to user management without third-party authentication. We also chose to scale back the inclusion of Accessories in the Retro Game Exchange database after gathering information for 5,000 games. Including all accessories for the selected gaming consoles, such as the custom button overlays included with Intellivision games, would have added many more data points and required additional ways to filter and organize the catalog so we chose to limit the catalog to games only.

4 Testing

4.1 Testing Plan

Tests will be based on the requirements provided in the SRS and linked through the Requirement ID given in that document. Developers will be expected to complete Unit Testing before checking code into GitHub. Integration and Acceptance Testing will be done according to the following schedule and any bugs or concerns will be discussed at the weekly Saturday meetings:

November 21	Initial function drafts and design document review
November 28	Data structure and web page mockup review
December 5	Data structure revisions, draft search algorithm, user interface screen layout, and JavaScript Function review
December 12	Data API review and integration with user-facing functions
December 15	Final acceptance testing and bug fixes

4.2 Tests for Functional Requirements

4.2.1 Test Cases:

3.2.1.a A New User is able to create an account when the appropriate information is supplied

3.2.1.b When a New User tries to create an account but provides incomplete or invalid information the system displays an error message that identifies the data field that is missing or invalid.

3.2.1.c When a New User is created successfully their User_ID and Email_Address are stored in the users.json data file.

3.2.2.a When the Catalog page is loaded it contains a Data Grid displaying the game catalog with columns for Console, Game Title, and Owner

3.2.2.b When the Catalog loads, it contains data that is organized correctly to match the Catalog.JSON file. This can be verified by selecting at least two random entries in the data grid and comparing the listed data matches what is found in the JSON file

3.2.2.c When the Catalog loads, the unique Index ID number for each row is not displayed on-screen.

3.2.3.a When a user enters Search Criteria that does not match any Catalog entries (ex: “QQQ”), then the system displays a message “No data available” indicating no matching results were found.

3.2.3.b When a user enters Search Criteria that matches more than one Catalog entry (ex: “Mario”), then the system displays all Titles that contain the search term in alphabetical order.

3.2.3.c When a user enters Search Criteria that matches at least one Catalog entry, then the system displays the information stored for all matching entries in the Catalog.json file including Console, Title, and Owner

3.2.4.a When a user clicks on the URL for a Game entry in the Catalog then the Web Browser should open the Wikipedia page associated with that Game. The page opened should match the URL stored in the JSON file for that Game.

3.2.5.a If the user has not logged into an existing User Account then the Library button displays an error message.

3.2.5.b If the user has logged into an existing User Account then the Library button is functional.

3.2.5.c If an existing User that already has a Collection clicks the Create Collection button then the system will display an error message stating that they cannot have more than one Collection. No new entry should be created in the Collections.JSON file.

3.2.5.d If an existing User without a Collection clicks the Create Collection button, then a new row is entered into the Collections.JSON file with the User’s User_ID.

3.2.6.a If an existing User with a Collection uses the Search function, then the results grid will include an Add to Collection button for each row that contains a Game that is not currently in their Collection.

3.2.6.b If an existing User with a Collection clicks the Add to Collection button, then that Game is added to their Collection. Verify that the Collections.JSON file includes a new row with the User_ID and Game_ID associated with this action.

3.2.6.d When a User adds a new Game to their Collection, then the Game will initially be assigned an Availability value of Private and a Status of Unavailable.

3.2.7.a When a User with a Collection clicks the View Collection page then they will see a data grid with a row for each Game in their collection. The grid contents will be sorted by the Index value in the Catalog.json file, in ascending order. (The Catalog index is sorted by console, then alphabetically by Title)

3.2.7.b When a User with a non-empty Collection views their collection, they will see two options available for each Game in their collection: Change Availability and Remove

3.2.7.c When a User with a non-empty Collection views their collection and selects the Change Availability button for a Private game, then the game will become Available for Borrowing. The Collections.JSON file will be updated so the row associated with that User_ID and Game_ID has an Allow_Borrowing value of 'true'.

3.2.7.d When a User with a non-empty Collection views their collection and selects the Change Availability button for an Available game, then the game will no longer be Available for Borrowing. The Collections.JSON file will be updated so the row associated with that User_ID and Game_ID has an Allow_Borrowing value of 'false'.

3.2.7.e When a User with a non-empty Collection views their collection then the grid will correctly display the Status value contained in the Collections.JSON file.

3.2.8.a When a User with a non-empty Collection views their collection and selects the Remove option for a game, then that Game will be removed from the user's Collection. The row associated with that User_ID and Game_ID will be deleted from the Collections.JSON file.

3.2.9.a If a User without an account views a Collection, then that user will not see the option to Request to Borrow any games.

3.2.9.b When a User with an account views another user's Collection, then the system will display a Request to Borrow option for any Game where the Status is Available and Allow_Borrowing is true.

3.2.9.c When a User with an account views another user's Collection, then any game in the user's Collection with an Allow_Borrowing value of false will be marked as Private and the Request to Borrow option will not be available for that game.

**Note: For the remaining test cases, the Authorized User that submits the Borrowing Request will be referred to as the Borrower, and the User who owns the game being requested is referred to as the Lender.

3.2.9.c When a Borrower submits a new Request for a Game, then a new row will be entered in the Requests.JSON file that contains the Borrower's ID, the Lender's ID, and the Game_ID for the Game that has been requested. The initial Status value for this row should be 'Requested'.

3.2.10.a When a Borrower submits a new Request for a Game, then the Lender should see this request listed when they access the system and open the Request page.

3.2.10.b When a new Borrow Request has been submitted and the Lender opens the Request page, the Lender will see each active Request listed in the Active Requests data grid. Each row in the Active Requests grid will contain the Borrower's User ID, the Game_ID, the Status, and options to Approve or Deny the Request.

3.2.10.c If a Lender Declines an active Borrow Request, then the Status of the request will be changed to 'Declined'. The Borrower will see this Denied Request when they access the Requests page with no further actions available for that request.

3.2.11.a If a Lender Approves an active Borrow Request, then the Status of the request will be changed to 'Approved'. The Borrower will see this Approved Request when they access the Requests page and the system will display the Borrower's Ship_Address.

3.2.12.a The Borrower will see a new option on the Requests page to Return Game for each Approved Borrow Request associated with their account and with a status of Approved.

3.2.12.b When the Borrower selects the Return Game option, they will see the Shipping Address of the Lender. When the Borrower confirms that they have recorded the Return Address, then the system will update the Status of this Request to Returned.

3.2.12.c When the Borrower selects the Return Game option and confirms, then the status of this request will be 'Returned' when the Lender views the Requests page. The Requests.JSON file will be updated with a status of 'Returned' for this request row.

3.2.12.d For each request that has a status of Returned, the Lender will see an option on the Requests page to Accept Return.

3.2.13.a When the Lender selects the Accept Return option for a request, then the system will update the status of this Request to Accepted. The Requests.JSON file will be updated with a status value of 'Accepted' for this request row.

3.2.13.b When the Lender selects the Accept Return option for a request, the Collections.JSON file will be updated with a status of Available for this game so that it can be borrowed again.

4.2.2 **Test Case Status:**

Test Case	Status	Date Tested	Testing Notes
3.2.1.a	Pass	12/15/20	
3.2.1.b	Pass	12/15/20	Fields like email validated and error displayed, but we don't have limits on long strings
3.2.1.c	Pass**	12/15/20	Warning: The user is logged out after creating an account - could be a bug but not critical
3.2.2.a	Pass	12/15/20	
3.2.2.b	Pass	12/15/20	
3.2.2.c	Pass	12/15/20	
3.2.3.a	Pass	12/15/20	
3.2.3.b	Pass	12/15/20	
3.2.3.c	Pass	12/15/20	
3.2.4.a	Pass	12/15/20	
3.2.5.a	Pass	12/15/20	
3.2.5.b	Pass	12/15/20	
3.2.5.c	Pass	12/15/20	Unable to create duplicate collection for a single user
3.2.5.d	Pass	12/15/20	
3.2.6.a	Pass	12/15/20	
3.2.6.b	Pass	12/15/20	
3.2.6.d	Pass	12/15/20	
3.2.7.a	Pass	12/15/20	
3.2.7.b	Pass	12/15/20	
3.2.7.c	Pass	12/15/20	
3.2.7.d	Pass	12/15/20	
3.2.7.e	Pass	12/15/20	
3.2.8.a	Pass	12/15/20	

3.2.9.a	Pass	12/15/20	
3.2.9.b	Pass	12/15/20	
3.2.9.c	Pass	12/15/20	
3.2.9.c	Pass	12/15/20	
3.2.10.a	Pass	12/15/20	
3.2.10.b	Pass	12/15/20	
3.2.10.c	Pass	12/15/20	
3.2.11.a	Pass	12/15/20	
3.2.12.a	Pass	12/15/20	
3.2.12.b	Pass	12/15/20	
3.2.12.c	Pass	12/15/20	
3.2.12.d	Pass	12/15/20	
3.2.13.a	Pass	12/15/20	
3.2.13.b	Pass	12/15/20	

4.3 Tests for Non-functional Requirements

4.3.1 Test Cases:

4.1.a The Status of a Game in the Collections.JSON file will be updated within 2 seconds of the game being Approved, Returned, or Accepted through the User Interface.

4.1.b The Catalog Search feature will take no longer than 5 seconds to display the search results.

4.3.2 Test Case Status:

Test Case	Status	Date Tested	Testing Notes
4.1.a	Pass	12/15/20	<1 second
4.1.b	Pass	12/15/20	<1 second

4.4 Hardware and Software Requirements

Testers must have the latest version of code from GitHub and are expected to have the latest browser and framework versions as listed below:

- Mozilla Firefox version 83 or above
- Google Chrome version 87 or above
- Node.js version 14.15 or above
- Express.js version 4.17 or above
- Require.js version 2.3 or above
- VUE.js version 1.0-Beta-15 or above

Testers are expected to use hardware compatible with the latest Windows, LINUX, or Android Operating System for system testing.

Hardware used for integration testing must have a reliable connection to the Internet with at least 1.5 Mbps upload and download speeds.

5 Analysis

Logan Tan:
I spent approximately 30 Hours on this project most of the time spent on the 2nd and 3rd Milestones. This is because these Milestones had the most specific requirements, and I missed a key meeting in deciding how the database would be implemented. This led to me spinning my wheels on Milestone 2 and on Milestone 3 I had personal life create issues for me to actively participate and by time my schedule started to clear the team was working in Vue and I had some issues learning the system, I should have spent more time trying to communicate with the team to better understand how to work the system. While I would say I spent around 8 hours scanning and thinking about Milestone 1, it was 10 hours working in circles for Milestone 2, and the remainder plucking at Milestone 3. Much of the time in milestone’s 2 and 3 were unproductive.

John McEchron:
I did not think to track my time spent on this project, but I would estimate my time as around 10 hours for Milestone 1, 12 hours for Milestone 2, and more than 20 hours for Milestone 3. During Milestone 3 I focused on building the database of games and the documentation since I did not have any experience with APIs.

Brandon Huang:
I would estimate that I spent about 34 hours on this project. Milestone one I spent around 8 hours actively working on the SRS or in meetings about the project. Milestone two I spent about 6 hours either in meetings brainstorming or working on the SDD. Milestone three I spent the most hours at approximately 20 hours, most of the work done implementing a backend API for the system. It’s worth noting that these times are rough estimates as I didn’t think to track the time I worked on the project over the quarter. Milestone 3 was definitely the most effort, as the majority of the actual

coding took place here. I have no experience in using express.js or even writing API's in general and it took a lot of experimenting and reading documentation to figure out how to do it.

Daniel Lee:
I spent an estimated 3-4 total hours on milestone 1 and about 35 hours on milestone 3 for a total of about 40 hours. For milestone one, I spent most of the time with research, brainstorming features and functionality. For milestone three, I spent a majority of the time researching web application implementations, javascript frameworks, and actually implementing the UI. The most difficult milestone was milestone three. Towards the end, there were dozens of features to complete with decreasing amounts of time, so I really had to grind through to get every feature in the UI implemented, ensure the api connections and communications were working as expected, and clean up the frontend client code to keep things legible while keeping the user interface itself clean.

6 Conclusion

We learned a great deal about working together as a development team and how it differs from individual software coding tasks like our homework assignments. Since we were forced to collaborate remotely we made use of online tools including Google Documents, Discord, and GitHub, but collaboration became more difficult as we tried to finalize our design work and move to implementation. We did start using the screen sharing feature of Discord in December, but that still limited us to a single person editing the code at one time. Similar to what we learned about eXtreme Programming, that method of coding was very informative to the observers but it may have slowed our progress as the more experienced programmer answered questions and explained their actions.

We also learned new skills from each other as we reached the implementation stage. Each of us had different skill levels in areas like requirements, diagramming, APIs, and web-based applications that were required for this project. We self-selected tasks that let us use our skills most effectively, but we also tried to share that knowledge with others so that we each had an understanding of system aspects that we may not have been familiar with before this project.

Appendix A - Group Log

Team communication was done primarily through Discord. We would use the chat feature to exchange information during the week and had two timeslots for voice chat on Thursday nights and Saturday afternoons. Meetings were suspended for the Thanksgiving holiday week.

Communication through Discord was effective in some ways, but not in others. The ability to chat through Discord and have files shared through GitHub was very helpful when we had competing schedules that prevented us from working on the project at the same time. However, the online communication format was not very good for specific collaboration tasks like brainstorming and design where we had to repeat ourselves often when two people spoke at once. It was also more time consuming to share ideas digitally when we had tasks like web page mockups or data relationship diagrams that could have more easily been done together on a whiteboard.