

Phase 3: Technical Report

Canvas Group: morning-1

Project Name: Game-DB

Website URL: <http://gamedb.us-east-1.elasticbeanstalk.com/>

GitHub Repo Link: <https://github.com/numan201/game-db>

Phase Leads:

Phase 1: Numan Habib

Phase 2: John Nguyen

Phase 3: Alejandro Rodriguez

Motivation:

We want to make a useful video game database for gamers. By incorporating data from various APIs, other websites, and communities, we can have a variety of useful information all in one place for our users.

Users:

All types of gamers, people interested in learning about different games, people shopping for games, and people who like to consume gaming related entertainment.

Members:

Albert Garza

- **GitHub ID:** possumrapture
- **Email:** albertsgarza@utexas.edu

Numan Habib

- **GitHub ID:** numan201
- **Email:** numanhabib@gmail.com

John Nguyen

- **GitHub ID:** johnnguyen3196
- **Email:** johnnguyen3196@gmail.com

Alejandro Rodriguez

- **GitHub ID:** JustAlejandro
- **Email:** rodriguezalejandro@utexas.edu

David Wolf

- **GitHub ID:** rambisco
- **Email:** david.wolf@utexas.edu

User Stories:

Phase 3 User stories:

1. As a gamer, I want to have a dark color scheme so I will be comfortable using Game-DB at night.
 - a. Estimated: 1 hour
 - b. Actual: 5 hours
 - c. Assumptions: User wants option for dark mode
2. As someone without a lot of free time, I'd like to know how long a game is before I start playing it so I don't end up playing a game for more time than I expected.
 - a. Time Estimated: 1 hour
 - b. Time Taken: 3 hours
3. As someone who wants to find new games, I want to be able to see a random game so I can explore games outside of my comfort zone, which are usually just the popular games.
 - a. Estimated: 1 hour
 - b. Actual: 3.5 hours
4. As someone who likes to share their opinion, I want to be able to write reviews so people will be informed of my good opinion.
 - a. Estimated: 4 hours
 - b. Actual: 7 hours
5. As a social media influencer, I want to share information about a game to my social media followers with a simple button click so we can experience the game together.
 - a. Estimated: 3 hours
 - b. Actual: 3 hours
6. As a college student with a tight budget, I would like to immediately know the cost of games when learning about them so I can include them in my budget.
 - a. Estimated: 6 hours
 - b. Actual: 12 hours
7. As a gamer, I would like to be able to search a game by its name in the database so I don't have to scroll through pages of games to find it.
 - a. Estimated: 3 hours
 - b. Actual: 4 hours
8. As a gamer who only likes puzzle games, I would like to filter by puzzle games so I can find a new game to play easily.
 - a. Estimated: 4 hours
 - b. Actual: 6 hours

9. As a gamer who likes to stay up to date with the best titles, I would like to sort by user rating so I can know what games are the best according to users.
 - a. Estimated: 3 hours
 - b. Actual: 4 hours
10. As a potential game buyer, I would like to have a sneak peak at the user achievements so I know what are some of the objectives of the game.
 - a. Estimated: 1 hour
 - b. Actual: 1 hour

Phase 2 User stories:

1. As a user, I want to see the latest and top games and gaming news so I don't have to spend a long time searching through the website to find these games and news.
 - a. Estimated: 8 hours
 - b. Actual: 8 hours
2. As a gaming enthusiast, I want to check the news for specific games so I can make conversation with my friends and know what games have interesting gameplay.
 - a. Estimated: 5 hours
 - b. Actual: 7 hours
3. As a user, I would like to be able to easily view all games, developers, and publishers in the database page by page (pagination) so the website is more pleasant to view.
 - a. Estimated: 3 hours
 - b. Actual: 3 hours
 - c. Assumptions: page by page means pagination
4. As a gamer looking to purchase a video game, I would like to see how many people are currently playing the game on Steam so I can know if I will be able to find an online lobby quickly.
 - a. Estimated: 2 hours
 - b. Actual: 1 hours
5. As a Game-DB user, I would like to keep track of games I would like to purchase so I can get them in the future when I have enough money.
 - a. Estimated: 4 hours
 - b. Actual: 6 hours

Phase 1 User stories:

1. As a grandmother with a grandchild who likes Fortnite, I want to know what Fortnite is so that I know my boy is not playing inappropriate games.
 - a. Estimated: 5 hours
 - b. Actual: 8 hours
 - c. Assumptions: user has no gaming experience and does not normally keep up with the current games
2. As a fan of Rockstar Games, I want to know some of the games that they made so that I can find other cool sandbox games that I can spend all day on.
 - a. Estimated: 5 hours
 - b. Actual: 6 hours
3. As a potential game buyer, I want to preview screenshots and videos of the game so I have an idea of the game type and how fun it can be.
 - a. Estimated: 3 hours
 - b. Actual: 3 hours
4. As a software engineering professor, I want to know who is making commits to a GitHub repo so I know how much each person is contributing.
 - a. Estimated: 3 hours
 - b. Actual: 12 hours
5. As a person who likes to watch Twitch streams, I want to watch a stream for a game I find in the database so that I can see its gameplay live.
 - a. Estimated: 6 hours
 - b. Actual: 8 hours
6. As a frugal video game purchaser, I want to know how many 5/4/3/2/1 star reviews there are for a game so that I can make an educated decision in purchasing a game and not waste time on a bad game.
 - a. Estimated: 3 hours
 - b. Actual: 1 hour
 - c. Assumptions: user does not have much free time because they are normally busy with work
7. As an owner of a PlayStation 3, I want to know what platforms each game is on so that I know if I can play it on my console
 - a. Estimated: 1 hour
 - b. Actual: 2 hours
8. As a video game historian, I want to know the release date of my favorite games, so that I can be factually correct in my discussions with peers.
 - a. Estimated: 1 hour
 - b. Actual: 1 hour

9. As a software engineer interested in game development, I want to know which studio made my favorite games, so that I can send in an application.
 - a. Estimated: 2 hours
 - b. Actual: 4 hours
10. As a software engineer interested in web development I want to know the tools used to make a website, so that I have an idea of the capabilities of tools and which tools I should invest time in learning.
 - a. Estimated: 0.1 hours
 - b. Actual: 0.5 hours

Design:

Phase 1:

For our design we used modern tools to expose us to technologies commonly used in the industry right now. Our backend is Node.js with MongoDB, and our website is hosted through AWS. For our framework, we chose Express since it is a common framework used to make web applications with Node. It makes web development with Node easier and more structured. When a user visits a URL on our domain, the framework routes it to the appropriate router file which will handle the request. The router will query and process data from MongoDB/APIs necessary to render the page. After this, it will provide this data in an object to a template file (.ejs) specific to this page. The .ejs file created for this page will render the provided data in HTML to serve the browser request\generate web page. Our .ejs files use various CSS classes and Bootstrap 4 to make our pages aesthetically pleasing and keep design simple.

Phase 2 Update:

1. User Login\Wishlist

To create a user system for wishlists, we used Passport.js to help us create our user authentication. Passport.js assisted us in creating user sessions and implementing OAuth login through Google Accounts. Once logged in, users can add and remove games from their wishlist.

Phase 3 Update:

1. User Review

When a user is logged in and on a single instance of a model (such as a single game), two text box forms are shown under the reviews graph. The user can write a title for their review and their actual review of the specific instance. When the user submits their review, a GET request is sent to a reviews router and relevant user data and their review is stored onto a specific collection in our Mongo database. When each instance is loaded, it searches for itself in the reviews collection and displays any user reviews.

2. Vendor/Prices

In the process of going to a game page, the server makes three asynchronous get requests to Steam, Microsoft, and Sony's databases. The request to Steam uses the steam game ID, which is already saved to our database and uses the Steam Web API to retrieve price data on the game. The request to Microsoft mimics the call the Microsoft Store page makes to give search results to users. Unfortunately the Microsoft Store has no easy way to obtain price data this way, so price data for Xbox One games is not available and we can only provide store links. To request from the PlayStation Network (Sony) for PS4 games we use a similar technique as the Microsoft store where we mimic PlayStation's search URL and return the closest match to the title of the game in our database. With Sony we are also able to get price data this way and are even able to retrieve the price for a game for PlayStation+ members (the default for our site). Using PlayStation and Steam vendor data we can also give some plaintext names (non-linkable) for publisher and developer as a fallback for game's whose publisher and developer aren't in our database.

3. Caching

When a game, publisher, or developer page is loaded by a user, the server first checks if there is a relevant 'cached' entry in our caching database so we can serve the page as fast as possible to the user. This means the API calls that need to be made on the first load don't need to be made again, resulting in much faster load times (typically 3-4 times faster). In the case of games, this data decays over time with a lifespan of 5 minutes. This is because data such as Steam player counts and Twitch streams are constantly changing and are not reliable for longer cache times. For the games page, even if there is a cached result the server will still make the API calls to update the cached result after serving the page. This results in the game page data on a cache hit being one page load behind the live data, in exchange for much faster load times. Since publisher and developer pages are static (outside of reviews), those caches have infinite lifespan and do not update like the game pages do.

We also cached information that came from the News API and the SteamNews API. The front page serves up gaming news on every load. The relevant articles don't change very often, so in order to not hit our API limit, these results are now stored for 24 hours. SteamNews, likewise, does not have game specific articles getting posted very often, so those results also are cached for 24 hours.

4. Steam Achievements

When a user navigates to a game that is on the Steam store, we make an Steam API request that returns a JSON object that contains an array of all achievements for a game that Steam tracks. The first ten achievements are displayed on the page.

5. Dark Mode

In the navbar, there is a bootstrap style toggle switch for the user to switch between a lite and dark mode similar to most modern websites. When the user clicks on the switch, a small JS function toggles between a lite and dark stylesheet.

6. Filtering

On the left side of each model's page, there is a form that allows a user to filter each instance of a model by clicking on each checkbox and submitting the form. The form sends a GET request to the specific model's router and the router creates a MongoDB query to filter the models.

Note: Users can deselect a filter by holding shift and click

7. Sorting

On the top right side of each model's page, there is a dropdown menu that the user can use to sort each model by clicking on a single option in the dropdown menu. The model's router checks if a user used the dropdown menu and creates a MongoDB query to sort the model.

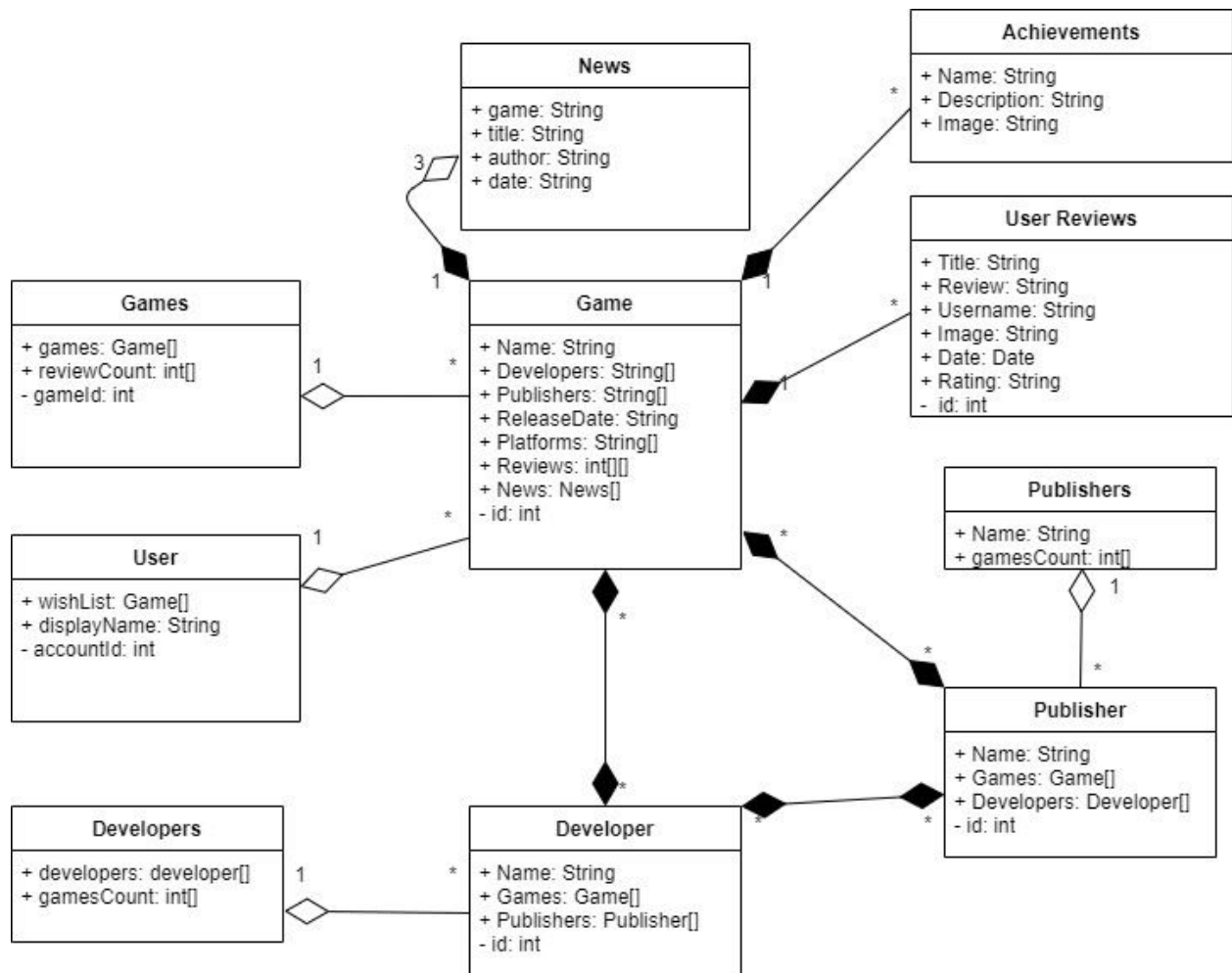
8. Search

On the very top of every page, there is a search box with a dropdown menu. The dropdown lets you specify what you would like to search. You can either search Games, Developers, or Publishers. The search form routes the query to the appropriate type of model (game, developer, publisher) and creates a Mongo query to return appropriate results.

9. How Long to Beat

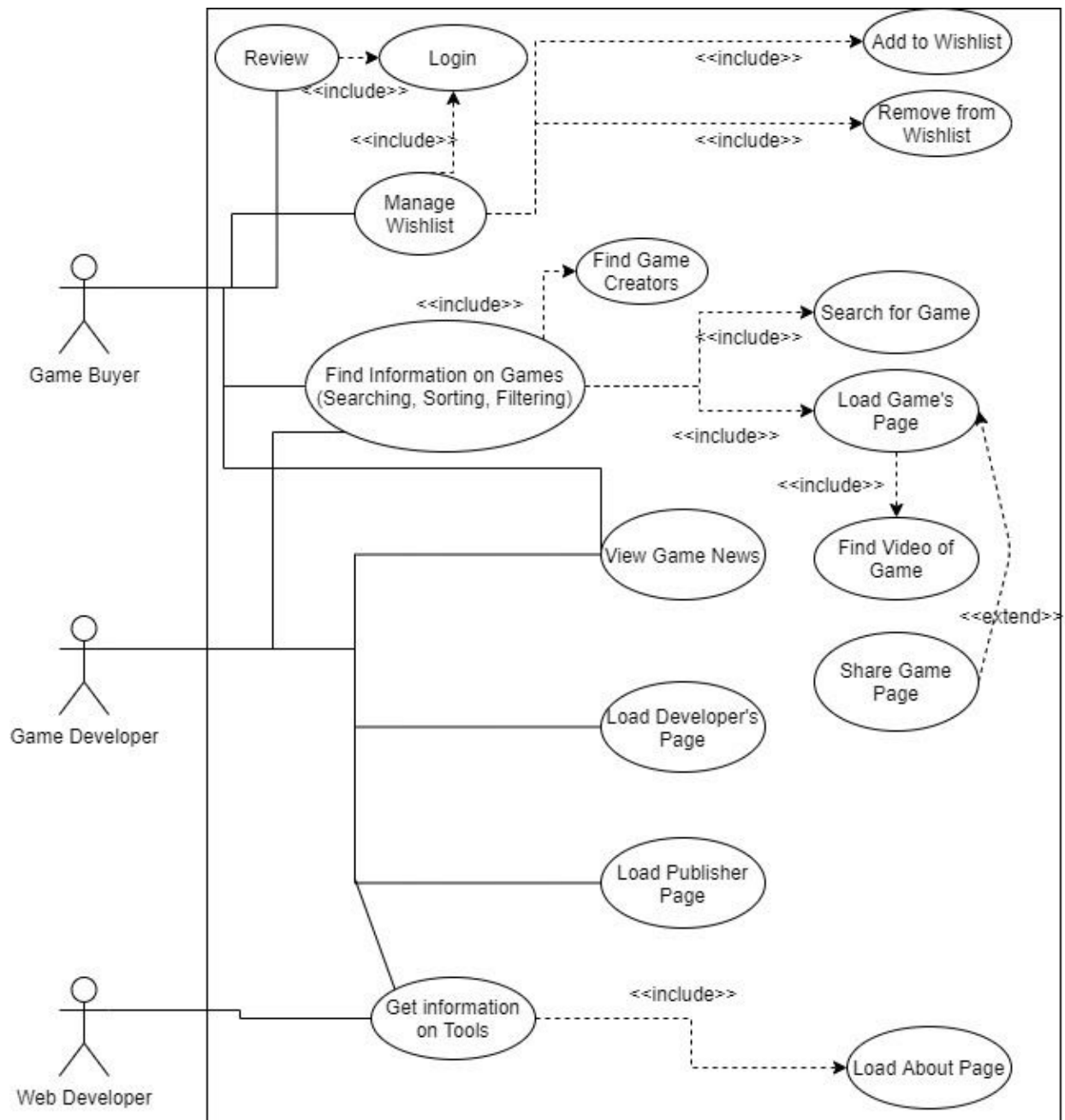
Underneath each game's 'Platforms' section, there is a listing of the average time to complete the game using How Long to Beat's API. We make a call to the API using the game's title and the API returns a closest match to its database. We then check if the title's are reasonably similar to each other and show the user the result if we think it's a match.

UML:



Class Diagram, Figure 1

Class diagram showing the relationships between different models, and the functionality that can exist between each.



Use-Case Diagram, Figure 2

Diagram showcasing the different actors for the website, and the processes they can go through to interact with the system.

Testing:

Phase 1:

Manually tested publicly viewable pages and all of their features.

Phase 2:

We did GUI testing by using Selenium to automatically test different parts of our webpage. All unique types of connections were tested such as: Developers -> Games, the GitHub Repo Link, pagination, and the tab to go to the Games model. Since we have a login feature, there are tests exclusive to being logged in and logged out. There are 5 test suites: Logged In, Logged Out, Doesn't Matter If Logged In/Out, Smoke Test, and All Tests (which is used to keep track of which tests have not been assigned to a suite). To make a test, we used the Selenium IDE which allows you to record a series of inputs, such as typing text into a text box, resizing your window, and clicking buttons. The IDE is really useful because you can quickly and easily add tests by navigating as you normally would and you do not have to worry about the tedious and slow process of typing so many tests and finding the correct button to click.

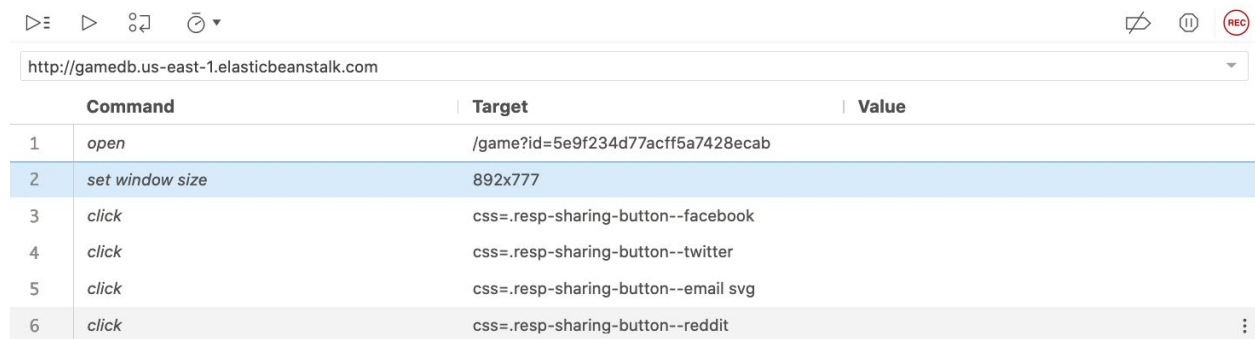
We tested JavaScript code by ensuring that all pages and JavaScript routes return a successful code. Each URL is routed to a JavaScript file which is then rendered with a JavaScript template. We tested URLs that did not require parameters by ensuring a successful load, and tested URLs that did have parameters by getting sample objects from the database and ensuring that loads still completed successfully.

Phase 3:

We added more tests by looking at the new features that were added, such as sorting, searching, and filtering, and creating tests for each new feature. The test suites Doesn't Matter If Logged In/Out, Smoke Test, and All Tests were updated. While manually testing and creating tests, we were able to find several bugs that were missed during development. It was much easier to test for Phase 3 than for Phase 2, even if there was more stuff to test for Phase 3, because of the experience gained during Phase 2. During Phase 2, we learned to think about what tests should be covered quickly, how to create the tests in an order that makes it faster to make more tests

(making tests that cover similar functionality across the different models sequentially instead of making all the tests for one model at a time), and how to use Selenium IDE.

Selenium test used to check buttons for sharing a link to a game with others.



The screenshot shows the Selenium IDE interface. At the top, there's a toolbar with icons for running, stepping through, and other actions. Below the toolbar is a URL bar containing 'http://gamedb.us-east-1.elasticbeanstalk.com'. The main area displays a table of test commands:

	Command	Target	Value
1	open	/game?id=5e9f234d77acff5a7428ecab	
2	set window size	892x777	
3	click	css=.resp-sharing-button--facebook	
4	click	css=.resp-sharing-button--twitter	
5	click	css=.resp-sharing-button--email svg	
6	click	css=.resp-sharing-button--reddit	

We simulate several buttons clicks that are used to share a game through four platforms: Facebook, Twitter, e-mail, and Reddit. We check if the buttons open correctly and do not make our website give an error. When we run a test, Selenium will act exactly as the user did when the test was recorded.

With mocha, we wanted to have an easy way to see if we had reached any API limits, since with certain APIs we would hit their limits from testing all day. Using axios, we made mocha tests to make sample API requests to different services with our keys, to make sure they returned successfully and contained the data we were expecting.

Models (required 3 are underlined):

1. Games:

a. Attributes:

- i. Using RAWG API
 - 1. Name
 - 2. Release Date
 - 3. Developers
 - 4. Publishers
 - 5. Platforms
 - 6. Reviews
 - 7. Related Screenshots
- ii. Using YouTube API
 - 1. Related Videos
- iii. Using Twitch API
 - 1. Related streams
- iv. Using Steam API
 - 1. Steam player count
 - 2. News
 - 3. Achievements
 - 4. Price data
- v. Using HLTB API (How Long to Beat)
 - 1. Time to complete a game
- vi. Using Microsoft Store
 - 1. Xbox One store link
- vii. Using PlayStation Network
 - 1. Price data

2. Developers:

a. Attributes:

- i. Using RAWG API
 - 1. Name
 - 2. # of Games made
 - 3. List of games made
 - 4. List of publishers worked with

3. Publishers:

a. Attributes

i. Using RAWG API

1. Name
2. Developers worked with
3. Games published

4. News

a. Attributes

i. Using Steam API

1. Title
2. Author
3. Date

Tools:

1. Express JS
 - a. Framework to create websites with Node more easily.
2. Axios
 - a. Library to make HTTP requests easily.
3. Bootstrap:
 - a. Used to make our website look better and more modern.
4. Passport.js
 - a. Authentication library to make user authentication system.
5. Postman
 - a. Test get requests for APIs
6. MongoDB
 - a. Database for storing site data. Also helps to reduce API calls.
7. Amazon Web Services (AWS)
 - a. Host website
8. WebStorm
 - a. IDE used to run website and debug
9. Visual Studio
 - a. IDE used to run website and debug
10. Visual Studio Code
 - a. Text editor with syntax highlighting
11. Google Chrome (Chrome Developer Tools)
 - a. Used to find bugs on website with Inspect Element tool
12. RAWG API
 - a. Used to pull information about games, developers, and publishers
13. YouTube API
 - a. Used to get relevant Youtube videos of a game
14. Twitch API
 - a. Add videos of people livestreaming games
15. Github API
 - a. Dynamically get commit and issue numbers from our repository
16. Google News API
 - a. Used to get overall gaming news on frontpage
17. Steam Web API
 - a. Used to fetch player numbers
 - b. Serves relevant news for each game
 - c. Returns price data for each game

18. draw.io

- a. Make UML diagrams to describe our website design

19. Selenium

- a. GUI testing

20. Mocha

- a. Unit Testing for JavaScript

21. HLTB API (How Long to Beat)

- a. Fetched user playtime statistics for the game's page

Reflection:

Phase 1:

We started working on this project early and finished 95% of Phase 1 half a week before it was due. We were also very communicative by having daily SCRUM meetings, properly distributing work, and reallocating workloads when necessary. The frameworks and APIs that we used were also chosen early in the planning process. Some improvements that we can make is planning a full schedule for meetings, instead of figuring out what we want to do for the meeting when we arrive at the meeting place. We can also plan weekly meetings at a specific time that works for everyone, instead of playing it by ear. We all learned more about web development tools. We also learned that we work effectively on our individual assignments when being in a group because we can easily use each other as resources instead of having to always find help online.

Phase 2:

During Phase 1, we made our website dynamic and stored a lot of data on MongoDB, so for Phase 2 we were able to focus more on additional features beyond the requirements such as adding a wishlist for games. There was significantly less communication between group members during this phase, compared to the previous phase, because we could not meet in person, due to Spring Break and the ongoing pandemic. However, we extensively communicated through Slack and helped each other debug our issues. We learned that in-person meetings make it easier to engage everyone and communicate more effectively because it is easier to understand visual cues and share what is on our screens.

Phase 3:

Our team fixed bugs quickly when they were reported in the group chat. In addition, when help was needed, we were quick to help out our teammates. Both the bugs and help asked for were finished within the same day that the team was notified of them. We did a good job of delegating responsibilities to members, so everyone felt comfortable finishing their work on time. Later on, there were last minute bugs relating to sorting and searching, and we ended up finishing much later than planned for. Initially, we planned to finish a week before the current deadline. We used two of our slip days on this project and ideally we would have completed this phase by the original due date. Since we

finished a few of our features late, much of the testing for these features also came in late, so we found bugs right before our project had to be finalized.