Team E14 EE461L Project
Anime

Shoyon Kermany, Neil Narvekar, Muhammed Mohaimin Sadiq,
Jason Siegel, Xylon Vester,
The University of Texas at Austin
November 1, 2020

TEAM INFORMATION:

Our team group is E14 and the project github is https://github.com/UT-SWLab/TeamE14

Name	E-mail	GitHub Username
Shoyon Kermany	Shoyon.Kermany@utexas.edu	ShoyonK
Neil Narvekar	neil.narvekar@utexas.edu	narvekarneil
Muhammed Mohaimin Sadiq	m.mohaiminsadiq@utexas.edu	mohaiminsadiq
Jason Siegel	jadamsiegel@gmail.com	jas23823
Xylon Vester	xvest10@utexas.edu	XyScripting

INTRODUCTION

This technical report is intended for individuals interested in the design and functionality of the anime website created by Team E14. The website is created for users that are interested in learning more about anime, anime characters, and tv studios/staff members related to anime production. For the purposes of this website, anime is considered to be the genre of film and television animation created in or influenced by the traditional style of Japanese 2D animation. There are thousands of different anime and finding information on a specific anime may be difficult which motivated our team to create a website that allows users to gather information immediately. Currently, the website provides useful information regarding anime, anime characters, anime studios, and studio staff using three APIs.

REQUIREMENTS

User Stories

The following section divides the user stories by phase and gives a brief description of each.

Phase I

As a user, I would like the website to be easily accessed on my mobile device, so that I can lookup anime info on the go (Estimated: 2 hours; Actual: 2 hours)

> This user story calls for creating responsive web design using bootstrap, so that the website functions in an easy to use way when viewing it on a mobile device as well as a computer.

As a user, I would like an easy-to-use, visually appealing model pages that display instances of the model page I am using (Estimated: 5 hours; Actual: 6 hours)

> This user story calls for creating four separate pages for anime, characters, staff, and studios which will display a grid of specific instances of anime, characters, staff, and studios that users can browse.

As an anime fan, I would like to easily be able to see the most popular anime, characters, and studios so that I can look for what to watch next (Estimated: 3 hours; Actual: 2 hours)

> Assuming that instances of a model have a popularity rating, this user story calls for the instances displayed on the model pages to be displayed by popularity instead of in a random order.

As a user, I would like visually appealing instance pages that display information about the instance (Estimated: 5 hours; Actual: 8 hours)

> This user story calls for the creation of instance pages that will display at least three attributes from a database that was populated by an API. The instance pages must also have a nice design so users can easily navigate the information.

As a user, I would like a visually appealing splash page to capture my attention enough to use the website (Estimated: 4 hours; Actual: 8 hours)

> This user story calls for a home page to be created that will hook and engage a viewer. It was done by adding a carousel linking to the model pages and a frame-by-frame moving background image of a popular anime.

As a user, I would like to be able to access model pages through links in the splash page so that I can easily navigate the website (Estimated: 1 hour; Actual: 1 hour)

This user story calls for a navigation mechanism that allows the user to access all model pages from the home page. This was done by adding a navbar with links to the model pages as well as putting links in the splash page carousel.

As an anime fan, I would like an about page so that I can figure out how I can make use of this website (Estimated: 1 hour; Actual: 1.5 hours)

Assuming the user does not know how to navigate a website, this user story calls for an about page that displays information regarding the developers, the design of the website, the API's that information was gathered from, and the github link to all source code

As a user, I would like to be able to access instance pages of each model through links in each model page so that I can easily navigate the relevant instances (Estimated: 2 hours; Actual: 2 hours)

Assuming that each model page has relevant instances, this user story calls for a navigation mechanism that will take the user to an instance page that will show all relevant anime, characters, and staff. This was done by using a carousel and adding preview cards of all relevant instances to the database.

Phase II

As an anime fan, I would like to see the faces of the voice actors behind my favorite characters and learn more about them. (Estimated: 3hours; Actual: 3 hours)

This user story calls for displaying the character's main voice actor on the character instance page and linking to the voice actor's instance page. The main voice actor for a character was found using the Anilist API.

As an anime fan, I would like to see reviews of the anime, so that I can decide if I want to watch it. (Estimated: 2 hours; Actual: 4 hours)

This user story calls for accessing the anime's written reviews from the API call and displaying them on the anime instance page.

As a user, I would like to be able to view anime trailers on instance pages so that I can see if I would like the anime. (Estimated: 10 mins; Actual: 30 mins)

This user story calls for embedding the youtube video located in the database for an anime's trailer on their instance page using the link to the trailer video.

As an anime fan, I would like to see related searches for the instance, so that I can look up similar anime that I may like. (Estimated: 1 hour; Actual: 1 hour)

This user story calls for the use of API information to display related searches of an instance somewhere on anime instance pages.

As an anime fan, I would like to see the anime's genre prominently on the anime's page so that I can quickly decide if it suits my taste. (Estimated: 1 hour; Actual : 1 hour)

Assuming the API gives us information about the anime's genre, this user story calls for displaying the genre prominently such that it is easy for the viewer to find.

As an anime fan, I would like to not accidentally see spoilers on the character bios, so that the story is not ruined for me. (Est: 1 hour; Actual: 1 hour)

Assuming our APIs give us an indication if certain parts of the description contain spoilers, this user story calls for creation of a button that only reveals parts of the description that contain spoilers whenever it is clicked.

As a user, I would like to be able to view many instances of a model on the model page, so that I can see if the website offers the anime I am interested in. (Estimated: 4 hours; Actual: 16 hours)

This user story calls for displaying multiple instance cards of a model type on a model page, including the ability for the user to click on different pages to access new instance pages.

Use case diagram

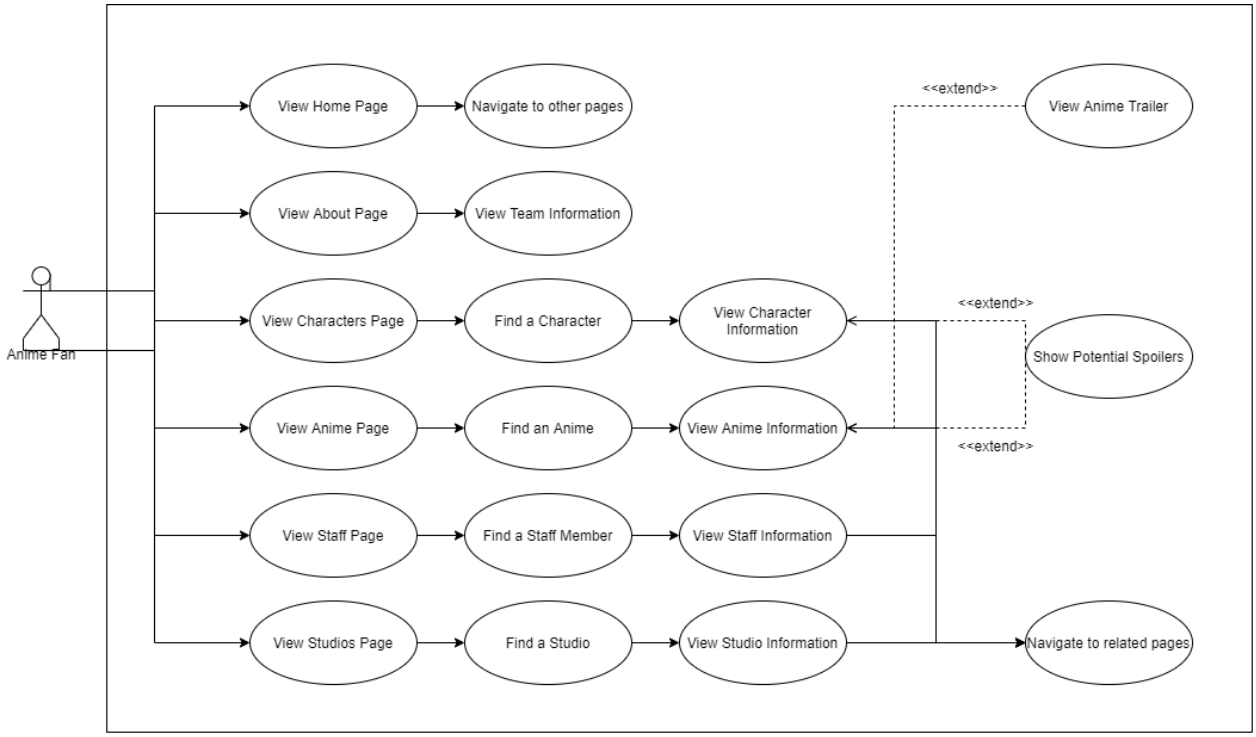

Figure 1. UML Use Case Diagram

DESIGN

In this website, Bootstrap is used as a CSS framework, Flask as a web framework, and Python is used for routing and data collection/analysis. As shown in the UML class diagram in Figure 2 on the next page, the website was designed using a logically connected hierarchy of HTML templates. The base template includes the navigation bar and relevant metadata and jQuery, CSS, and Bootstrap CDNs. All other templates are extended from this base template. The website consists of four models: anime, characters, staff, and studios with each model having different attributes. A visually appealing splash page with a carousel and a means of navigating to model pages is provided. Through the splash page, the user can access the different model pages. All the model pages logically follow a model page template consisting of a brief description about that model and a grid of the most popular instances of that model. Instance cards of a model consist of at least three parts: an image, three attributes, and a link to that instance's page. Each instance page has a basic layout consisting of a brief description about that instance and links to

the other three models. For example, a character instance page will provide links to related anime, related staff (voice actors), and related studios. These can be in the form of three-card dynamic carousels, or simple lists of hyperlinked text. However, each model's instance page is customized based on the unique attributes of that model. For example, an anime instance will have an embedded trailer video and fan reviews, which no other model's instances will have. There is also an about page that provides information about the website and the team.

For styling and adding functionality to the HTML pages we used common css and js files as far as possible. For example all the instance pages use one css and js file, and all the model pages use one css and js file. The splash page however has its own css and js file as it is unique.

The anime website gathers information on models and instances shown from four different API's: Anilist, Jikan, Wikipedia, and Pytrends. MongoDB was used as the database to store information on instances gathered from the different API's due to its NoSQL property. The API's return information as a nested dictionary or list of nested dictionaries so a key-value database like MongoDB was optimal and each API has its own database on the MongoDB server. All data elements in the database have an ID, and the corresponding elements from different databases have a linking ID. When a model or instance loads, a call is made to the separate databases by ID to gather information and the information is then passed into a template that is rendered by a Flask function.

The class diagram below shows the relationship between the different types of objects in the website.

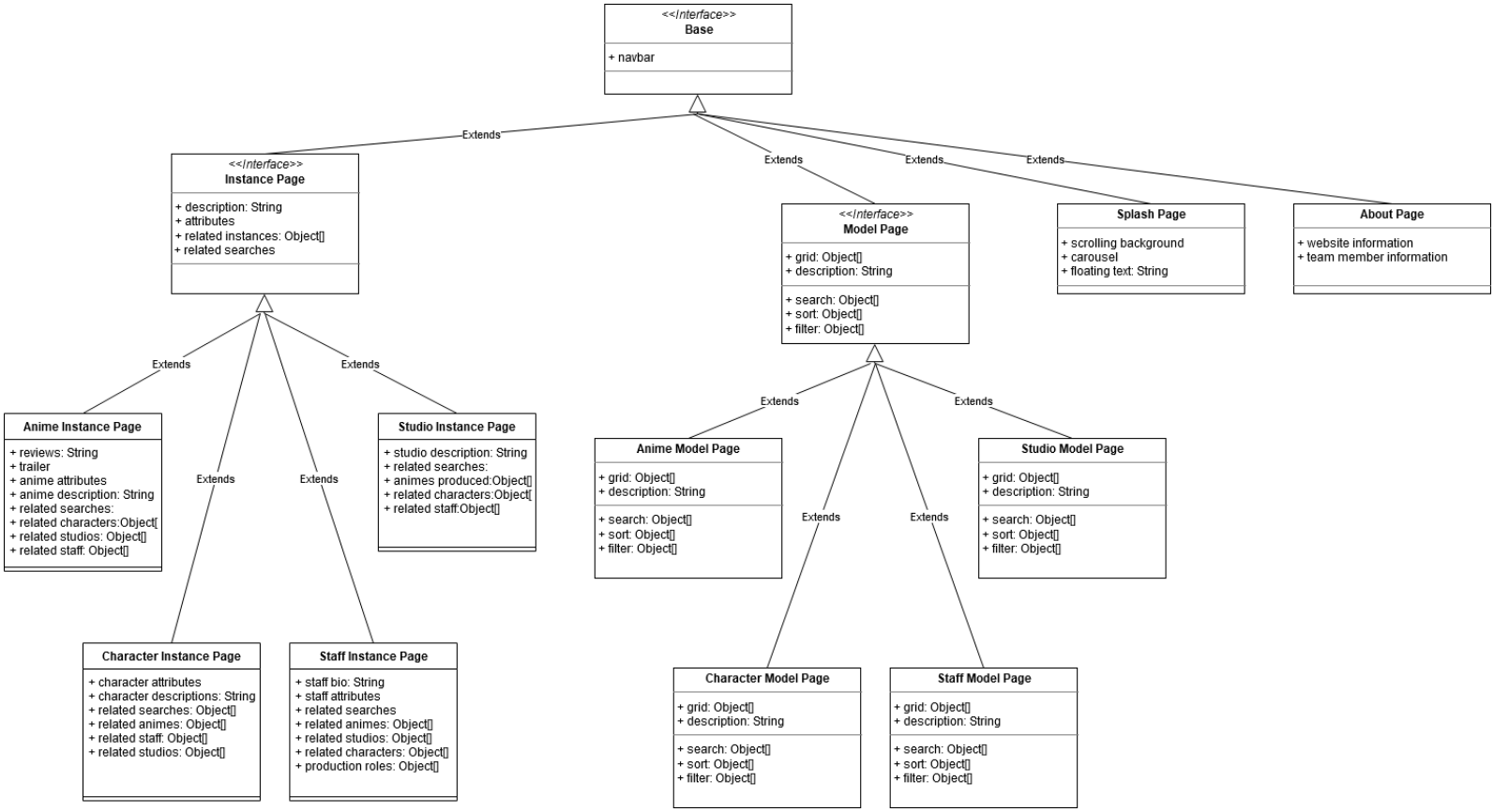

Figure 2. UML Class Diagram

TESTING

The following section describes the frontend, backend, and GUI testing done for the project.

Frontend (Mocha)

The Mocha test framework was used to test the frontend code for the instance and model pages which were written using the jQuery library. (Please note that the 'node_modules' folder necessary to run the mocha tests has not been included in the GitHub repository because it is too large. The list of relevant dependencies, however, can be found in the package.json and package-lock.json files in the testcases directory.)

Model Pages

jQuery was used for dynamic pagination on the model pages which meant that the number of instances on the model pages could be changed without having to modify the pagination code. The correct number of pages with the constraint of nine instances on each page would be dynamically populated. The tests were to:

i) Ensure that the number of instances on each page is 9

ii) Ensure that the formula (ceil(numInstances/9)) for calculating the number of pages to populate given the total number of instances on the model page was correct. For example there should be 12 pages for 100 instances, and 5 pages for 40 instances.

iii) Ensure that the jQuery code was actually dynamically modifying the DOM to include the right number of buttons in the pagination bar by running the code on the HTML and then checking the modified DOM.

iv) Ensure that the jQuery code was correctly dynamically saving calculated data attributes like the number of pages to the pagination component in the DOM.

Instance Pages

On the instance pages, jQuery was used to generate the carousel which can show 3 of the other models' instance cards simultaneously. We ensured that the variable that needed to be set for exactly 3 cards to appear simultaneously was set correctly.

Backend (Unittest)

We used the unittest unit testing framework to test functions in our backend code.

Tests for Parsing Descriptions

The descriptions of the character and voice actor instances in their raw format included other useful attributes for the instances (which were delimited by double underscores) as well as miscellaneous links to external websites (which were enclosed in parentheses). We wrote functions to clean up the descriptions and extract the relevant data.

Attribute extractor test

We tested that the attributes that were delimited by double underscores were properly extracted from the raw description by the function we wrote.

Parentheses, HTML tags, square brackets, and tilde remover tests

We tested that single and multiple instances of regular strings and website urls that were enclosed in parentheses or triple tildes (~~~), or contained lingering HTML tags or square brackets in the raw descriptions were removed by the function we wrote.

Spoiler finder tests

Spoilers in the character descriptions are enclosed in special delimiters (~! and !~). We tested that these were separated from the general description properly by the function we wrote.

Tests for Database

We used MongoDB for our database.

CRUD tests

We created a dummy database called CRUD with a crud collection inside it to test simple create, read, update, and delete operations in a MongoDB database.

GUI (Selenium)

The Selenium framework was used to conduct various GUI tests across the site to ensure the display and website functionality were working correctly.

The initialization method loads the website as a graph from an input file, but can also perform a traversal to load the graph (which will take a very long time to crawl through all the pages). This graph is used to perform some other tests.

Splash Page Text Scrolling Test

In this test, Selenium scrolls through the splash page in order to make sure that the introductory text loads and displays properly. It fails if any element fails to load before the automated browser reaches the bottom of the page, or displays in the incorrect order.

Link 404 Test

A random subset of link nodes are selected. The test crawls through each one to make sure that no instance returns a “404 Not Found” error.

Instance Link Test

A random subset of instances are selected. The test defines a passing threshold and a batch size. A select number of instances must have correct links to other instances. This is a soft test because some pages can redirect.

Anime Instance Content Test

In this test, a random subset of anime instances are selected. The test defines a passing threshold and a batch size. A select number of instances must pass very thorough content checks on the page, or the test will fail. (Title, Description, Attributes, Image, Trailer, Reviews, and Carousel)

Character Instance Content Test

In this test, a random subset of character instances are selected. The test defines a passing threshold and a batch size. A select number of instances must pass very thorough content checks on the page, or the test will fail. (Title, Description, Attributes, Image, and Carousel)

Staff Instance Content Test

In this test, a random subset of staff instances are selected. The test defines a passing threshold and a batch size. A select number of instances must pass very thorough content checks on the page, or the test will fail. (Title, Description, Attributes, Image, and Carousels)

Studio Instance Content Test

In this test, a random subset of studio instances are selected. The test defines a passing threshold and a batch size. A select number of instances must pass very thorough content checks on the page, or the test will fail. (Title, Description, Attributes, Image, and Carousel)

Model Pagination Test

This test navigates through all model pages, ensuring that the pagination function is working properly by checking the cards loaded and the active property of the selected button.

MODELS

The website has the three main model types Anime, Characters, and Staff, with an extra fourth model type Studios. These models are the most relevant information that would be required by an anime fan using the website. Each model page has a grid of cards that link to a page with more information about that specific instance of the model type.

The Anime pages contain a variety of information from the Anilist API, Jikan API, and Pytrends API. The data presented on this page includes the anime's English and Japanese titles,

description, genres, rating, reviews, related searches, links to the anime's characters and the character's voice actors, and the studios who made it. The Character pages contain information from the Anilist API and Jikan API. They display the character's description, number of anime and manga titles appeared in, links to animes they appear in and their voice actor, and if provided by the API, attributes such as age, height, and birthday are also displayed. The Staff pages contain information from the Anilist API, Jikan API, and Pytrends API. They display staff descriptions, number of voice acting and production roles, their language, related searches, links to animes they had a production role in, and links to characters and anime they were a voice actor for. Finally, the Studio model uses information from the Anilist API, Pytrends API, and Wikipedia API. They display the studio's description, related searches, and links to their most popular animes and characters.

TOOLS, SOFTWARE, AND FRAMEWORKS

The following section describes the tools, software, and frameworks we used each phase.

Phase I

All tools, software, and frameworks introduced in phase I were also used in phase II onwards.

Flask

This was the Python framework that was used to build the backend of the website. This framework allows different HTML templates to be rendered depending on the url the user is on, allowing us to easily display our different model and instance pages depending on what links the user selects.

Bootstrap

This was a frontend framework to make our website have a more modern look for the user. It was used to create cards on the model page to be clicked on, create carousels to navigate to

related instance pages on any instance page, create jumbotrons on every page to display information, have responsive web pages for use on a computer or phone, and to create a navbar the constantly at the top of the user's screen to allow for easy navigation around the website.

HTML

HTML was used to create base templates for each of the web pages displayed on the website. These templates were expanded on by various languages and frameworks like JavaScript, CSS, and Bootstrap to make the pages look more modern, as would be expected by a user of the website. Additionally, these HTML pages provide the linking between different website pages.

JSON

The JavaScript Object Notation (JSON) framework was used to take a response from the Anilist API's GraphQL query and convert it into a usable Python dictionary.

JavaScript

JavaScript was used to create the "animated" background for the splash page using a frame-by-frame image transition in response to scrolling. It was also used to create the fade in/out floating text on the splash page.

jQuery

jQuery is a JavaScript library that allows more succinct syntax when it comes to accessing elements in the Document Object Model (DOM). jQuery was used for dynamic pagination for the grid of instances on the model page. It was also used to create the carousels on the instance pages which can show up to three instance cards simultaneously unlike the default carousel which can only show one at a time.

CSS

This frontend framework allowed us to build off of our base HTML templates and develop a better looking frontend interface that included background colors, text colors, text fonts, and various other improvements.

Phase II

All tools, software, and frameworks introduced in phase II were also used in phase III onwards.

MongoDB

This was the database used to store documents queried from the APIs. Our group set up one shared MongoDB that the backend team used to store our data from various APIs in one place. This allowed fast access to pages than the slower use of API calls. In future phases, the databases created in this phase will be used for searching and sorting the database.

Unittest

This Python testing framework was used to test if specific functions in our program produced the expected output using assert methods. For instance, it was used to test if create, read, update, and delete (CRUD) functions on our MongoDB database were performing their expected tasks.

Selenium

Python GUI testing framework allowing our group to test the website's overall interface functionality. Selenium was used to verify the proper content was loading on each page, and that the interaction elements, like buttons and links, were performing the correct actions. By using selenium to test every single link on our model pages, we were able to pinpoint exactly which links were not working as expected and debug why they were not working.

Mocha

Mocha is a Javascript testing framework which we used to write unit tests for the JavaScript and jQuery portions of our code.

REFLECTION

The following sections will discuss our reflections on the project.

Phase I

Overall, all team members were fairly new to web development which means there was a steep learning curve for the entire team in terms of both frontend and backend work. We learned how to manage and organize a fairly large project with multiple members, not just in terms of code organization, Git best practices, proper commenting, descriptive file naming, and documentation, but also in terms of working around different people's personalities, schedules, work rates, and work ethic. We learned a couple of different mechanisms to extract data from an API and how to dynamically allocate the data on instance/model pages.

Five things we struggled with:

1. The Jikan API was easy to extract data from using Jikanpy, but some information such as studios could not be directly extracted with jikanpy and had to be requested through a url.

2. Planning was not done thoroughly enough. In general, information about studios was difficult to come by, so we may have to rethink studios as a main model page unless we can find an API with more information on studios.

3. Development was started very late as the team took a while to decide on a common time to meet and decide on a project. This also led to many questions regarding requirements that arose during development not being clarified until very close to the deadline. This should be resolved now that the basic architecture and idea of the project has been established and understood.

4. Merge conflicts were not handled properly at first resulting in broken code being pushed to the master branch. The team members had to get familiar with best Git practices, including creating feature branches followed by pull requests, and pulling the most up to

date version of the code before starting work, in order to successfully avoid and/or resolve merge conflicts.

5. As was expected with phase I, general team coordination, task allocation, and timeboxing were not smooth. The team members still needed to learn each other's work patterns and work rates. Internal deadlines need to be set in order to finish each phase smoothly before the deadline and not rush at the end.

Five things that worked well:

1. We were able to come to a consensus on a project topic (Anime) that every member on the team was interested in. This will allow each member to have a vested interest in making the website as good as possible.

2. We set up the website so that it is dynamically populated by extracting data from at least one API without hardcoding. This will make scaling much easier for the coming phases, as we have already displayed and linked almost all of the instances together.

3. The front end of the web pages turned out better than expected because we were able to use banner images from one of our APIs as background images, so that each instance page has a customized background image relevant to that instance. We were also able to find good resources on setting up scrim-based text boxes (to enhance readability) and card-based carousels (to enhance the visuals).

4. Despite the initial struggle with merge conflicts, one thing we did do well with regards to Git was issue tracking. We made issue specific branches and used issue resolution keywords in the pull requests to automate issue closing. This also allowed the issue resolution to be automatically documented with the linked pull requests.

5. The tasks were broadly broken down into frontend, backend, and documentation. Around half the team worked on the frontend with the other half working on the backend. Everyone contributed to the documentation, involving issue tracking, user stories, the technical report draft, pull requests, etc. The team effectively used Slack to communicate, by breaking down the Slack channel into sub-channels based on the task category and helping each other out and offering useful advice as necessary.

Phase II

We learned how to create and use a MongoDB database to find, insert, delete, and update information. The operations and functions we learned in order to accomplish this will greatly benefit us for the upcoming phase. For our project, we used jQuery for pagination on the model pages and for the carousels on the instance pages. JQuery works on the DOM in a similar way to JavaScript, but has a different syntax which we had to learn. We learned how to test code using Selenium, Mocha, and unittest. These testing frameworks were new to the team so there was a bit of a learning curve when it came to testing.

Five things we struggled with:

1. Initial task assignment may not have been granular enough, as there was some confusion about which part of a larger task, for example testing, was assigned to which team member. Tasks need to be thought all the way through and divided as appropriate.

2. We struggled with collecting adequate data from some of the APIs which meant having to rethink what our third model should be. This was a consequence of not having fully thought through what each model would need and what APIs were available at the beginning of the project.

3. Based on the nature of our project, the graph of linked instances was very dense. In order to not make our database too large, we had to restrict our project to the most popular instances. However, this meant that we needed to keep checking for broken links as the popular instances could be linked to less popular instances in the original API but not in our database.

4. The Anilist API and the Jikan API have different id's for the same characters so searches on Jikan had to be done through name instead of ID. This sometimes led to wrong information being selected to be placed in the Jikan database. To deal with this, we created a log as the code would run to see what was being inputted, and we had to manually go through and correct the failures. This ended up taking a long time, and we will work on creating a better process for this.

5. Modifying the database was not done in an organized way sometimes. Because of this, some members would end up deleting documents from the database they believed were unnecessary, causing the website to not work for a time. We need to be more careful in modifying documents in the database in the future, since if we have a problem with the database while our project is being graded we would lose a lot of points.

Five things that worked well:

1. Overall, work was more evenly distributed than Phase I. We were more organized with task assignments which we did very early on. We had two team members primarily doing backend work, and two team members doing front-end work. One team member was focused on testing, while others also contributed to testing their parts of the code.

2. In this phase, we set mid-week and weekly internal deadlines for our tasks which helped us manage our time better so that we were not overwhelmed close to the deadline. We started work almost immediately after the end of the last phase and held a retrospective to analyze and correct our past mistakes.

3. We used git best practices more effectively in this phase. There were no merge conflicts which broke the code. We were able to learn about git practices such as pull requests and branching in the last phase, and get good practice with them this phase.

4. We used issue tracking more effectively as a means of communication. Any team member who found a bug on the website would create a GitHub issue so that not only

was there no chance of forgetting to address it, but since the entire team could see it, the relevant team member whose expertise was in that domain could resolve it quickly.

5. Overall, communication was better. Team members were more responsive on Slack and the scheduled bi-weekly meetings had a higher attendance. Our team meeting with Dr. Eberlein probably helped with getting everyone on track.

REFERENCES

Used for card alignment on the model page grid:
https://stackoverflow.com/questions/48639792/bootstrap-4-cards-as-grid-with-the-same-height-and-width

Used for the splash page to create a frame-by-frame moving image on scroll effect:
https://levelup.gitconnected.com/how-to-create-frame-by-frame-moving-image-on-scroll-effect-30ce577c63c2

Used to create the carousels on the instance pages which show 3 cards simultaneously:
https://www.codeply.com/go/EIOtI7nkP8/bootstrap-carousel-with-multiple-cards

Used to create the scrim based text boxes on all the pages:
https://travishorn.com/responsive-scrim-6f286da5b6a5

Google Trends API for Python:
https://towardsdatascience.com/google-trends-api-for-python-a84bc25db88f

Wikipedia API for Python:
https://pypi.org/project/wikipedia/

JikanPy Reference Docs:
https://jikanpy.readthedocs.io/en/latest/

Mocha Reference Docs:
https://mochajs.org/

Used for dynamic pagination on the model pages:
https://stackoverflow.com/questions/43521242/how-to-make-pagination-with-description-of-specific-contents-in-each-page

Used to style the speech bubbles for the anime reviews:
https://codingislove.com/css-speech-bubbles/

Used to make the show less/show more buttons for the reviews and spoilers:
https://stackoverflow.com/questions/52332952/can-we-use-partial-collapse-concept-in-bootstrap-4

Used to test jQuery code using Mocha:
https://gist.github.com/robballou/9ee108758dc5e0e2d028

Used to perform verification tests using Selenium:
https://selenium-python.readthedocs.io/locating-elements.html

Used to make verification and interface functionality checks with Selenium:
https://stackoverflow.com/questions/53571352/how-to-scroll-down-on-my-page-through-selenium-and-python