Designing for Data Informed User Experience

Project 3: Interactive Project - Production Journal

Website: https://54x1.github.io/aniDistro/

Nathan TSE | U3200943

Description
This group project will produce a novel interactive work that responds to an identified design problem.

Aim
The project aims to demonstrate your technical understanding of web APIs through the creation of novel interactive interfaces.

Task
Define a design problem and create a functional solution using dynamic data retrieved from APIs.

Design Problem
There isn't a centralised place in the current anime community where people can search for the anime, they want to watch at a legal service provider. So, for this project 3, I have decided to address this issue by utilising two different API's StreamData and Jikan. The StreamData API will provide data on legal streaming services like Crunchyroll, AnimeLab and the Jikan API, which will help gather all the necessary data on an anime such as image, title, related genres, etc.

Production Journal
Research
The initial research I did was trying to find APIs suitable for solving the problem presented in this project. I started researching pre-existing databases and APIs on the internet, such as JustWatch, simpleanime API from RapidAPI and MyAnimeList.

One of the only websites that somewhat solves this projects problem is JustWatch. They have an extensive collection of where to stream TV shows and movies. Though, upon searching for anime, JustWatch's database could not display where to stream for some titles. Even though JustWatch's has an extensive API collection, it was only available for commercial use.

In addition to researching JustWatch, I tried using the MyAnimeList's API and simpleanime's API to return anime data. Even though MyAnimeList's API has a vast anime database and simpleanime's API has an extensive collection of where to stream anime, I quickly discovered it wasn't easy to set up the API's in JavaScript. Therefore, I replaced the MyAnimeList API with Jikan, an unofficial MyAnimeList API database and simpleanime API with StreamData.

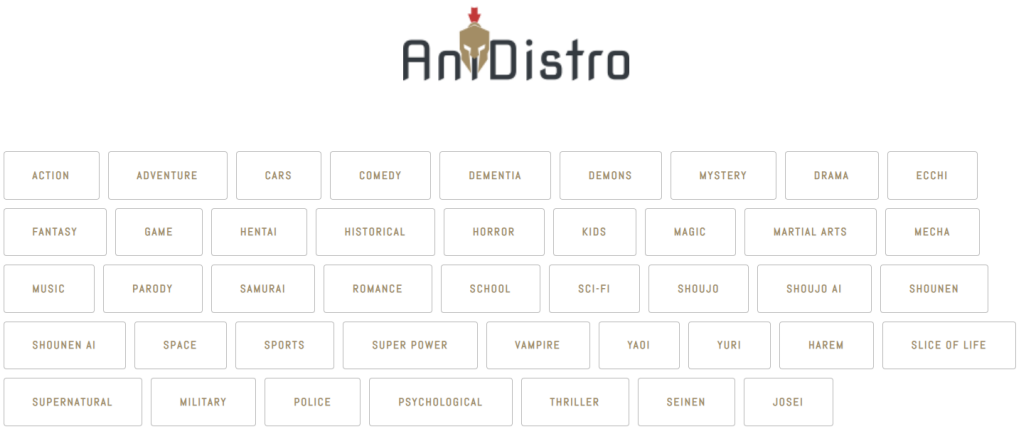
Jikan's API documentation far surpasses the original MyAnimeList API as myanimelist API had virtually no documentation. Jikan is free to use open-source PHP & REST API. It is available in 15 different types of wrappers (such as .net, python, java, etc.). It has a friendly, supportive community and is regularly updated, making it one of the most popular and widely used anime database API's.

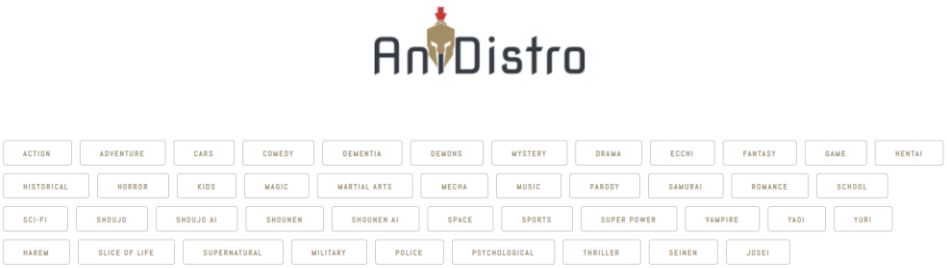
While Jikan being so thoroughly maintained, StreamData, on the other hand, is no longer updated as of May 2020. According to BecauseMoe, an associate of StreamData, on his Twitter account, he tweeted, "Due to updates and anti-scraping implementations on a couple of our sources, because.moe will be unable to update for the forseeable future. The listings will be accurate as of May 27th". Because of the new updates to anime streaming websites, it now prevents web scrapping. Because.moe can't depend on StreamData's API to display where to watch anime as they can no longer update their anime data.

The other part of the research was discovering how anime databases display their information. Since I already use a couple of anime databases, such as MyAnimeList.net and AniList.co, these inspired me and gave me the foundation to design what I wanted in the project.
Development Process
StreamData CORS Error
Users must download or enable a CORS extension for the StreamData API to function and display correctly while browsing the website. This is because the SteamData API is no longer maintained or updated. Therefore, it might have internal API errors such as certificates or SSL issues.

Displaying genre on home page
In the initial stages of developing the genre buttons for the home page, I manually wrote down all 40+ genres into an array and then used a key and value loop to display them. This worked when I only had 40 or so items in an array, but I quickly figured it'd be too much work for the type producers with over a thousand items. Thankfully I revisited the Jikan API for documentation, as they had recently released an improved and updated version of the API. Jikan v4.0 is still in beta stages, so not everything is perfect. But even so, this new version could filter and list genres as well as filter and display different seasons by years, licensors, studios, and producers, etc.

Buttons





I changed all the sizes of buttons to a larger size to be more direct and filling towards to user. The Aired (Season, Year) button is a great example that displays a user's feedback when they hover or click this button. As the button's background is highlighted in red, and the text is changed to white,

suggesting to the user that this button is clickable and leads to more information. When hovered or clicked upon, the button underneath Type is muted so that only the text and border colours are greyed out to suggest to the users that the button is only for display.

Hover Effect

Like many sites that display content in a gallery format, such as MyAnimeList, Netflix, Crunchyroll, etc., its convention that when a user hovers over an item, something happens like a backdrop shadow or title text appearing, to give the user feedback that they have or about to select the item. Thus, on my website, I have also added this user feedback feature to aid the user in finding the title of that item as well as aiding in exploration.

Design Decisions and Challenges

Design Decisions and Challenges

Framework

In this project, I originally tried to use Skeleton as my framework, but due to how barebones it was, I added Materialize v0.100.2 on top of Skeleton's framework. The Materialize v0.100.2 beta release is a framework created by Google. It is a mobile-friendly framework that allows users to design grids and interactive buttons easily. It is also showcased in interactive maps and eCommerce.

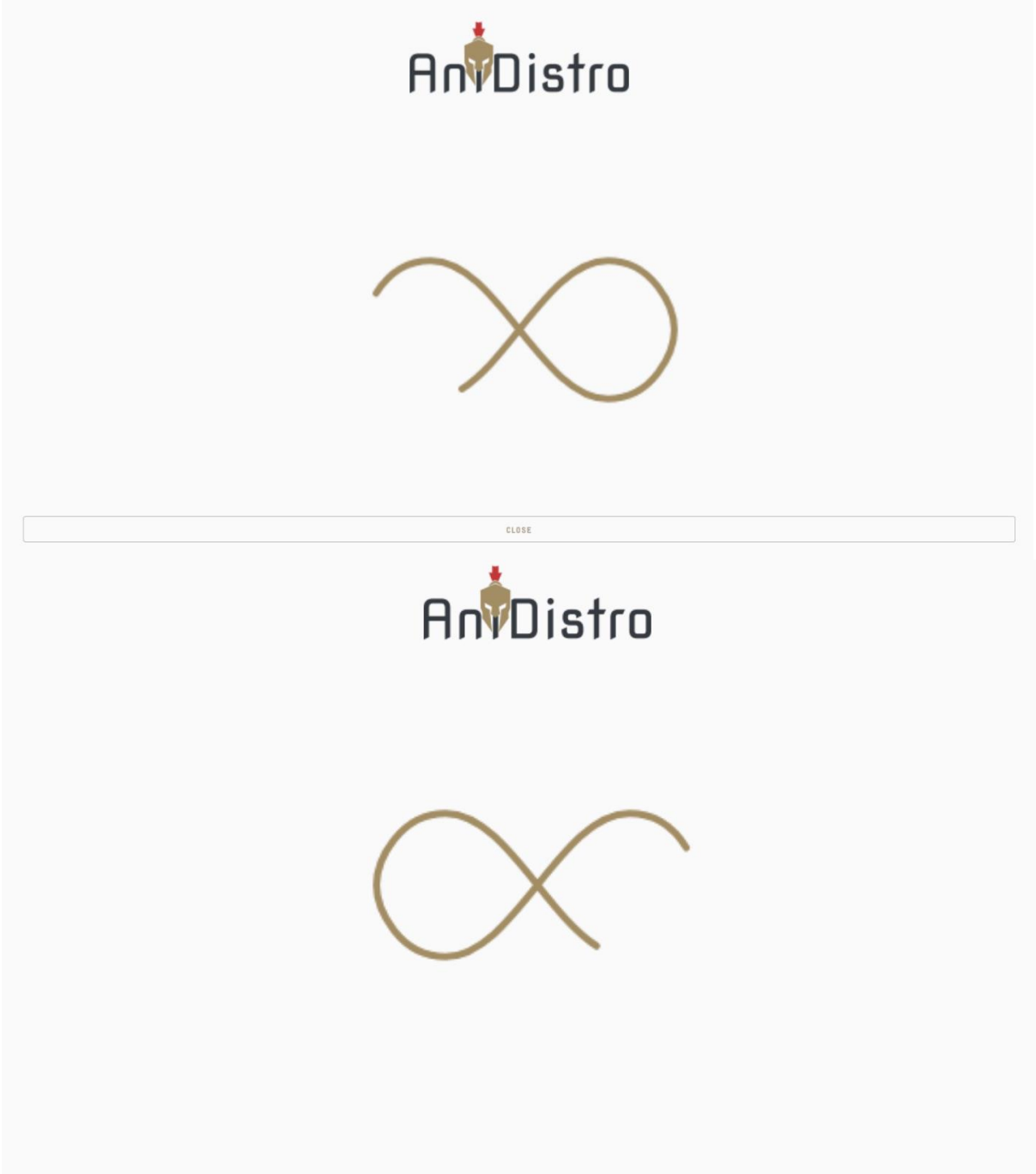
Modal

In the initial design of the modal that contained all the anime's information, I displayed all the data of what I wanted on the anime to the modal, so I could check and see if everything was working. I had barely made it mobile responsive at this stage as I just wanted everything not to break.

After successfully pulling all the data from both API's Jikan and StreamData, I then thoroughly designed the modal with the help of a few suggestions from my tutor and other colleagues who tested the site. To avoid uneven content, I placed the anime image, synopsis and trailer into a div,

then "clear both" the div so that the rest of the other content would remain underneath the anime image, synopsis and trailer div. I used this method for the other two content divs.

Since the data from both API's takes time to load and display, I placed a loading gif while the data is loading then removed the gif once the content has loaded. At first, I had a close button at the same time as the loading gif, but this caused some bugs where the modal wouldn't close or cancel the request correctly. Therefore, I disregard the close button at the loading gif stage and instead appended it to the anime content modal.



Resizing Ratio

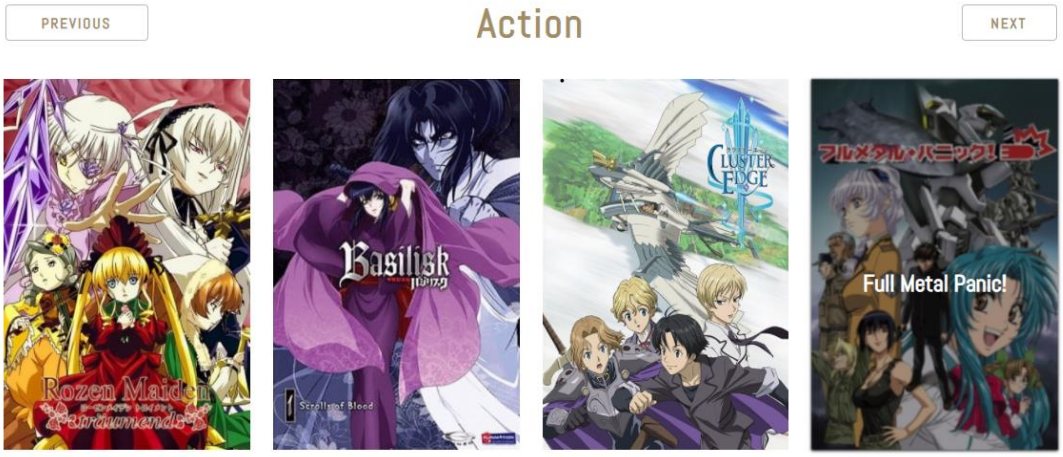
One of the initial challenges of this project was the responsive image gallery set up I made. I had issues with the height of the anime image since many of the images had different dimensions. I initially dealt with the height issue using too many media queries. So, I thought up a better way and

used jQuery to dynamically load and resize the height of all the anime images.

Next and previous buttons

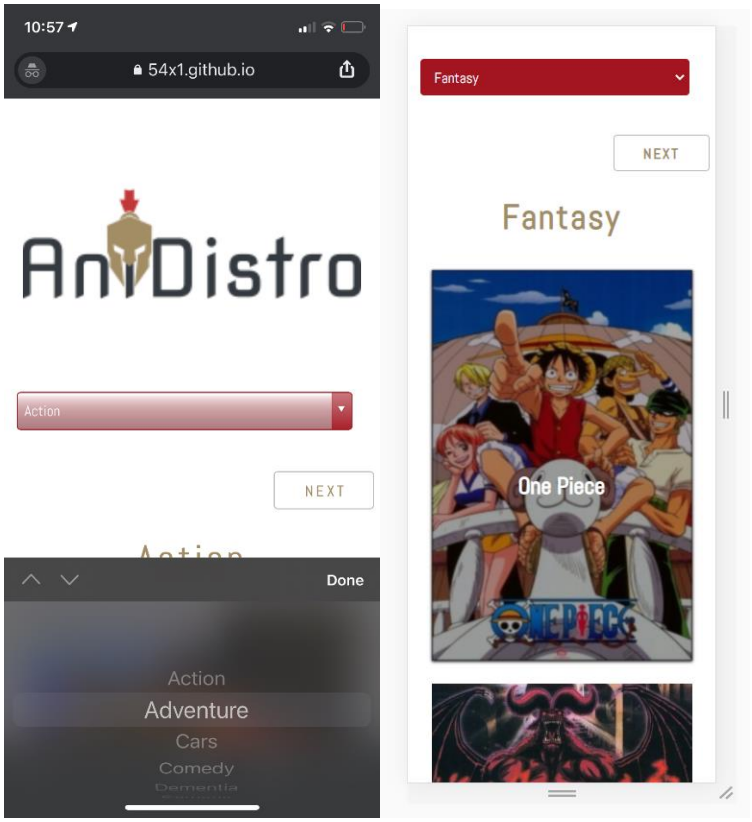
One of the features I added to the project was the next and previous button. Since the results are paginated, I could use this to display the next page or previous page. This proved to be more challenging than expected; the previous page was very difficult to code as the variable for the current page determined the previous page. But since the current page variable would be undefined, making the previous variable call an error when fetching the URL request. In the end, I managed to

get the next and previous buttons functioning on genres and producer queries by appending and selecting the different values from the buttons.
But for the season by year, pagination for the next and previous became too overwhelming to process. Instead of only passing one variable like in the genres next and previous buttons, two different variables passed through the season by year functions, making this very hard to manage and code. If I had more time on this assignment, I would next time try to make the season by year next and previous buttons functional.



Actual Mobile issue

One challenge in this project I could not overcome was fully mobile compatible since my genres select option would not display the genre when the user selects done. My thoughts on this were that since the genres data is appended, it's not officially in the DOM when the site loads. However, the genres data does load when in the device console. So, this would appear the genres as based on click instead of touch or tap. All in all, I don't have an idea of how to fix this bug on an actual mobile device.



Conclusion

Overall, I'm delighted with the work I've done; I have made a functional example of what the design of this project could look be. However, I am very dissatisfied that the SteamData API was discontinued. Consequently, only fetching data from about a year ago. Due to time restraints, I couldn't add subtle features such as the top 10/100 anime or even a random anime generator button. This would have vastly improved the website and made it more interactive with the user.