MuseNews Report

OVERVIEW

MuseNews is a source for news and information about today's top artists developed by Team 7. The site contains detailed information about a variety of songs and artists, while also allowing access to the latest music news. This report will detail the status of the project at the end of phase two.

Team Members

- Venkata Ravila, vkr339, venkataravila@gmail.com, GitHub Profile Link: vravila
- Sam Dauenbaugh, sad2929, s.dauenbaugh@gmail.com, GitHub Profile Link: sDauenbaugh
- Alexander Tekle, at36953, tekle.alexander@gmail.com, GitHub Profile Link: AlexanderTekle
- Daniel Walsh, dws956, danwalsh98@gmail.com, GitHub Profile Link: donuthole55
- Kedar Raman, kvr336, kedar.v.raman@gmail.com, GitHub Profile Link: kedaraman

Github Repo Link: https://github.com/vravila/MuseNews

Website Link: http://musenews.appspot.com

Canvas Group: afternoon-7

Project Name: Muse-News

MOTIVATION AND USERS

We were motivated to create this application because we wanted to create a database for music modeled after the IMDB database. We felt like this would be beneficial to the world because, although there are many different sources of information on music artists and songs, they are rarely consolidated. We wanted to combine the artists and their songs with news about each of them to allow users to get up-to-date news about their favorite artists and songs. Artists communicate to their audience through social media applications like Twitter and Instagram, and there is constantly new news about them and their music. We wanted to combine this information into one application, so that users can quickly explore artists and their lives.

The target audience for this application is anyone interested in music. Whether someone is a passionate fan of a certain artist or a novice seeking out new music, our application allows people to explore artists and be up-to-date with their work.

REQUIREMENTS

The requirements for this project were obtained from our team's collective brainstorming as well as our customer team. Our customer team is Team 7.

User Stories for Phase II

Customer User Stories “Team 7”:

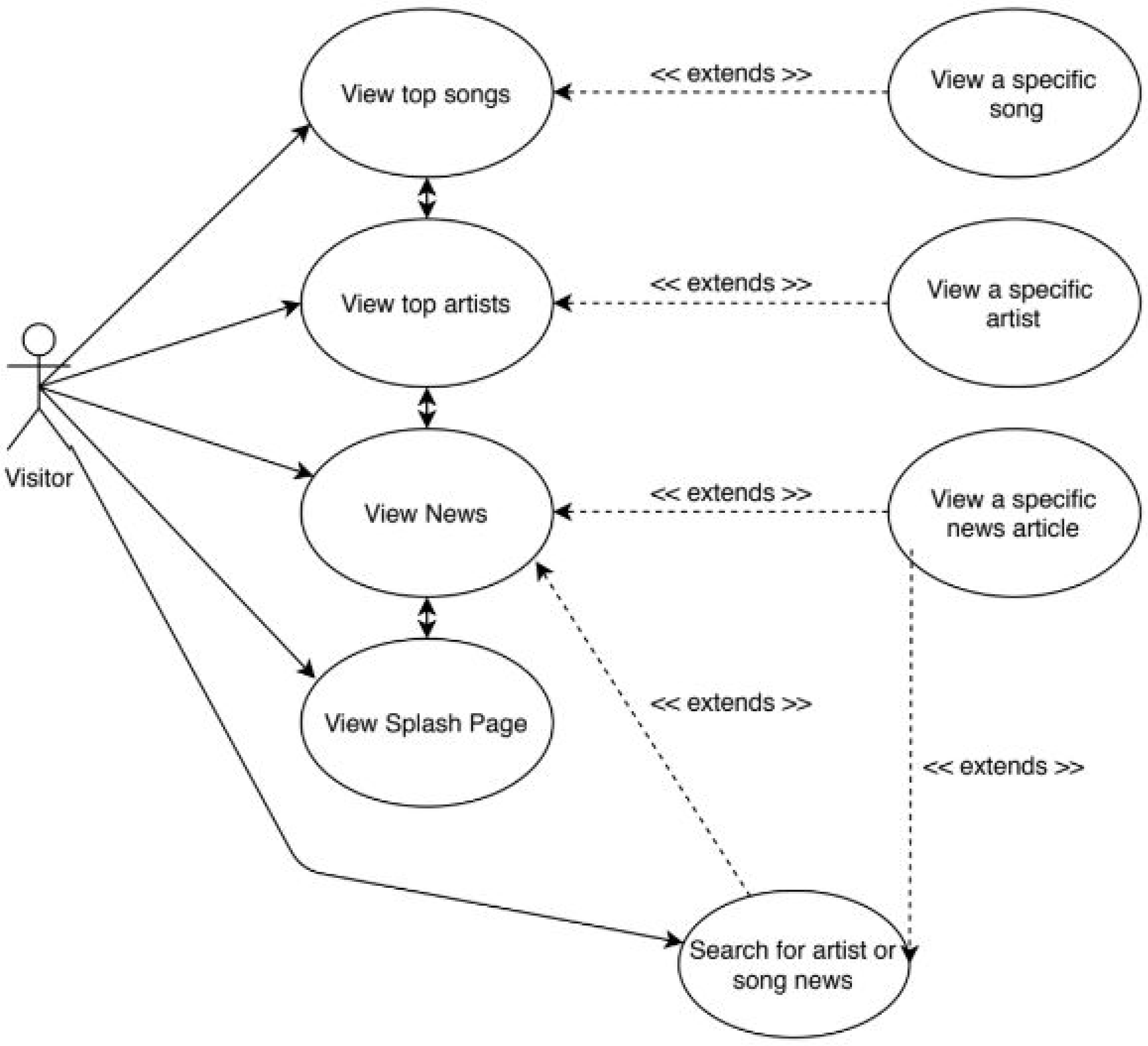
None for this phase.

Our User Stories:

- As a fan of The Weeknd, I want to be able to see the latest Tweets about The Weeknd, so that I can be up-to-date and see what people are saying about him and his work.
  - Estimated Time: 12 hours
  - Actual Time: 10 hours
  - Description: We want to add Tweets to the instance page of each artist. This way, when users load the page of an instance of an artist (e.g. The Weeknd's page), they will see information about the artist as well as the most recent tweets about the artist. This furthers Muse-News's goal of giving users interconnected and up-to-date information about music.
  - Assumptions: The user wants to just see the tweets about a particular artist when they visit their page, not search or sort Tweets as a separate entity.
- As a Metallica fan, I want to be able to search for the latest news on them, so that I can discover articles about them and get the latest updates about them.
  - Estimated Time: 20 hours
  - Actual Time: 25 hours
  - Description: We want to implement searching for news articles by artist, so that we can link artists to relevant news articles about them. This will consist of searching either a database or the Google News API based on the artist name, getting the results, and displaying it.
  - Assumptions: none
- As an avid music listener, I want to be able to quickly navigate between artists/songs and news articles about them, so that whenever I find an interesting song or artist, I can quickly search for news and reviews about it.
  - Estimated Time: 15 hours
  - Actual Time: 10 hours

    - Description: We want to add links from the instance pages of each song or artist to relevant news articles about the respective song or artist. So on the song instance page, there should be a link to all of the news articles about that song. On the artist instance page, there should be a link to all of the news articles about that artist.
    - Assumptions: User wants to be directed from an instance page to a list of all news articles about that instance, not have a list of all news articles on the instance page itself.
- As a marketing consultant at IHeartRadio, I want to see Tweets about a song that is currently being played, so that I can choose Tweets to display on the ticker while the song is being played.
    - Estimated Time: 7 hours
    - Actual Time: 4 hours
    - Description: We want to add Tweets to the instance page of each song. This way, when users load the page of an instance of a song (e.g. the Blinding Lights page), they will see information about the song as well as the most recent tweets about the song. This furthers Muse-News's goal of giving users interconnected and up-to-date information about music.
    - Assumptions: The user wants to just see the tweets about a particular song when they visit their page, not search or sort Tweets as a separate entity.
- As a music fan, I want to be able to quickly get multiple pages of artists and songs, so that I can go through many artists and songs and find those that interest me.
    - Estimated Time: 20 hours
    - Actual Time: 30 hours
    - Description: We want to get many instances of songs and artists into a database, so that we don't have to make API calls each time we load a page of songs or artists. We want to create a server that accesses this database with one API call for every page to load the page quickly.
    - Assumptions: none

Use Case Diagram



DESIGN

The website consists of three models (songs, artists, news), a splash page, an about page, and separate pages for each instance of each model. The home page contains links to the about page and the pages for each of the three models. The about page contains relevant information about the project, the developers, and the tools used. The songs page consists of 30 pages each containing 10 cards that serve as links to instance pages for songs. Each song instance page has information about the song including its title, a link to its artist, a picture of the album art, a description, number of listeners, play count, tags, and tweets about the song. The song instance page also contains a link to the latest headlines of the song and its artist. This is a link to the news page with search results for that song and artist. The artists page consists of 20 pages each containing 10 cards that serve as links to instance pages for artists. Each artist instance page has information about the artist including their name, a picture of the artist, a description, whether they are on tour, number of listeners, playcount, tags, similar artists, and recent tweets. The artists page also has links to all of their top songs in the muse-news database and a link to the latest headlines of that artist. This is a link to the news page with search results for that artist. The news page contains many pages of 10 cards each that serve as links to instance pages for each news article. Each news article instance page has information about the article title, author, the picture on the article, a short summary, and a link to the full article. Also included are links to the artist that the article is about and links to that artist's top songs.

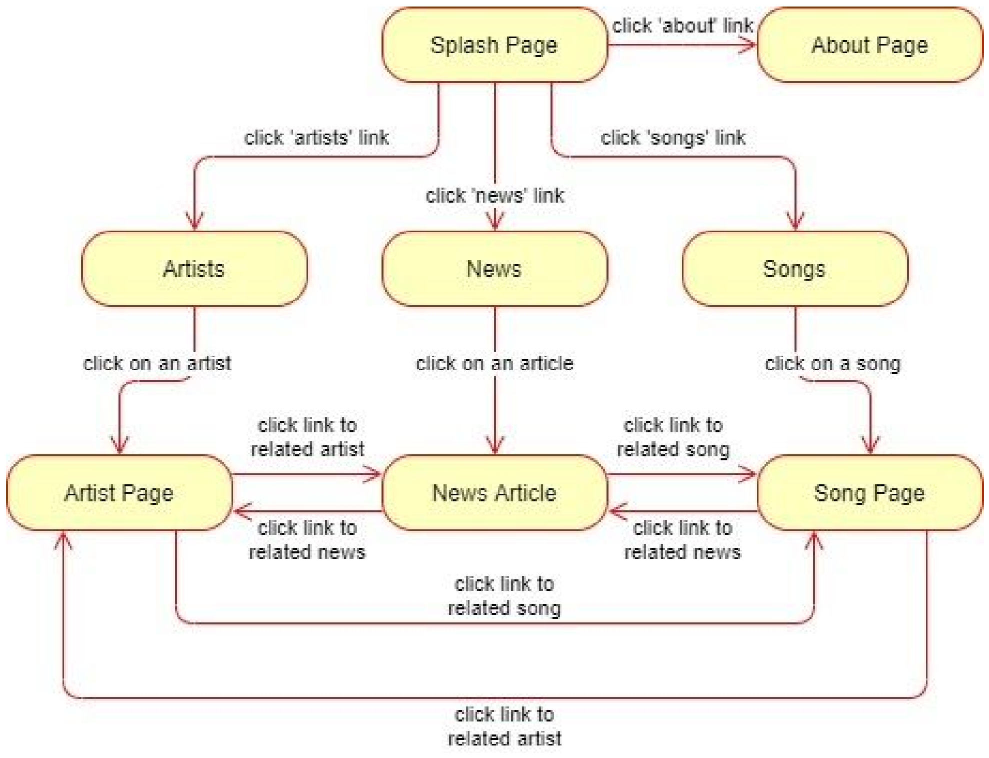
To design the application, we started off by splitting it by model. We designed a home page that linked to a separate page for each model. Each model homepage was developed individually for functionality. Pages were added from each model homepage for each instance. We use MongoDB to store many instances of data for each model. We created Python scripts (in /muse-news/backend_unittests) to populate MongoDB based on REST API calls. This directory also contains Python unit tests for the MongoDB population code. For the artists model, we have an artists collection in MongoDB. Our script fills this collection with data from the LastFM API. For the songs model, we have a songs collection populated by the LastFM API as well. For the news model, we have a news collection populated by the Google News API. For this model, we store the first 5 pages of data in MongoDB. In the songs and artists model, we also search the Twitter API to display the five random tweets featuring that song or artist. The rest of the pages are dynamically populated with calls to the Google News API itself. We chose this design because we should be displaying the latest news, so storing all of the news articles in MongoDB and constantly updating it was not optimal. It made the most sense to directly call the Google News API to get the most up-to-date information. However, since we only had a limited number of API calls, we decided to store the first few pages in MongoDB, because otherwise simply visiting the News page would use up API calls.

For the back-end code, we used Node.JS and the Express.JS framework. We created endpoints for our server in the /muse-news/routes directory. The base script to run the server is /muse-news/index.js. These endpoints directly access our MongoDB database. The routes are divided up by model, so we have a JavaScript file for news routes, artist routes, and song routes. The root endpoint for our server API is musenews.appspot.com/api. From there, you go to either /artists, /news, or /songs. Each model then has its own endpoints based on what functionality was necessary. For example, a call to /api/artists/getArtistByName/The Weeknd will get the JSON object for The Weeknd from our MongoDB database. We decided what routes to create based on the needs of our application and each model. For artists, we can get artists by rank or name, or get a list of artists given a range of ranks. For songs, we can get songs by rank, artist, or song name and artist. We can also get a list of songs given a range of ranks. For news, we can get news articles by what page we are on. The routes for songs and artists also contains an endpoint to use the Twitter API to get tweets about a song or artist. These endpoints were tested extensively with Postman throughout the development process as we figured out how to structure our back end code. We also thoroughly tested the endpoints of our server through Mocha tests in the /muse-news/test directory.

To create the front-end of our application, we use React.JS. Most of the front-end was developed in Phase 1. The major change we had to make for Phase 2 was making calls to our server API instead of directly to the Google News API or the Last FM API. In order to integrate our front-end and back-end code, we use the fetch routine to make GET requests to the endpoints we created for our server API. For example, on the artists page, to get the artists for the first page, we make a GET request to /api/artists/getArtistByRankRanges/1/10. This means that we want to get all artists with rank between 1 and 10, since our first page consists of the Top 10 artists. This request returns a JSON object that we parse and create cards out of which link to the artist instance page. Each artist instance page uses the /api/artists/getArtistByName endpoint, which returns a JSON object with information about that artist. We also get tweets using the /api/artists/getArtistTweets endpoint. Similar logic is used for the songs and song instance pages, where we get a JSON object from GET requests to our server API and use this to create our page. For the News page, we make GET requests to our server API for the first few pages, and then make GET requests to the Google News API for the remaining pages. The logic behind this decision was described in the second paragraph. Both of these return JSON objects that are formatted the same way, since our MongoDB collection was created by making a call to the Google News API.

State Diagram

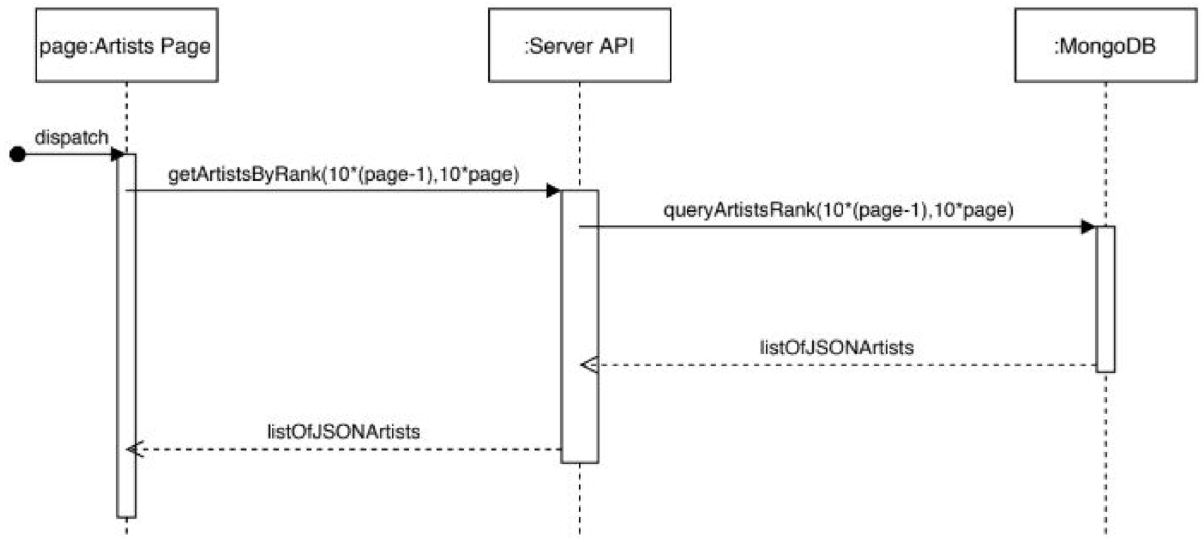
The internal links embedded in the site's pages can be seen in our state diagram. The state diagram does not contain any searches or transitions through the navbar or through external links.



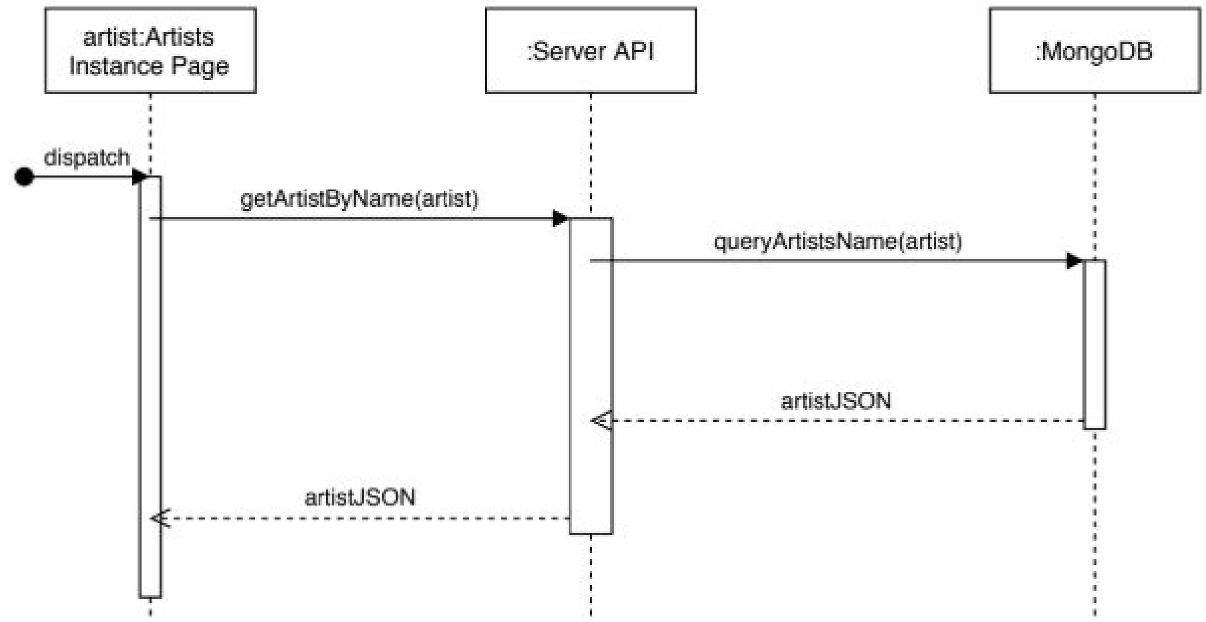
Sequence Diagrams

The relationship between loading a page and how it calls our back-end server can be seen in the sequence diagrams below. The call upon dispatch is made by a React.js fetch() call when the page is rendered. The data returned from the Server API is parsed as a JSON object to fill the page with relevant data.

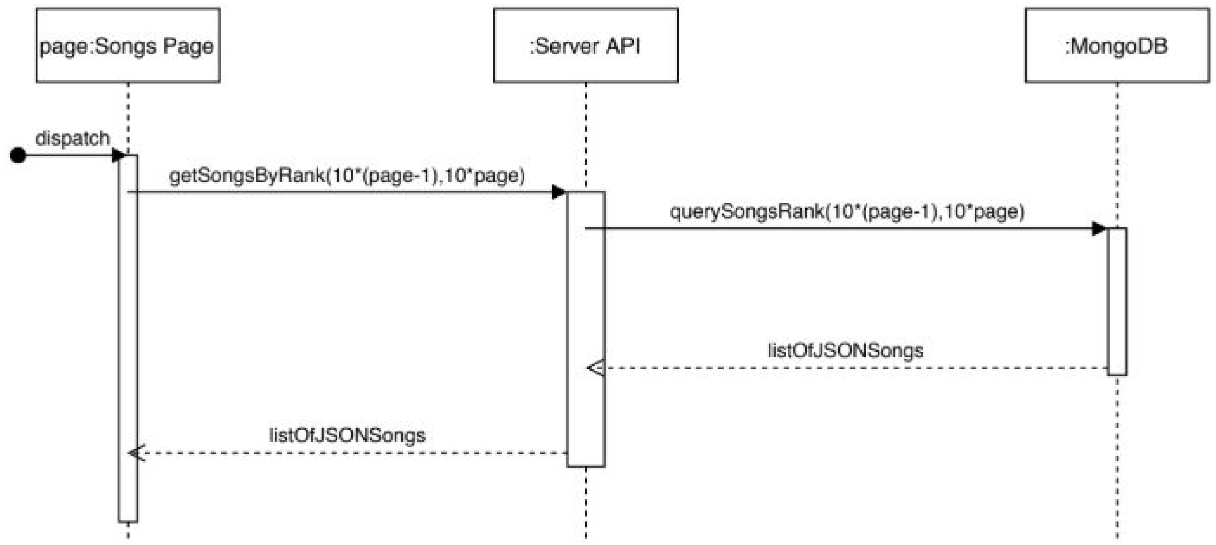
Loading Artists Page:



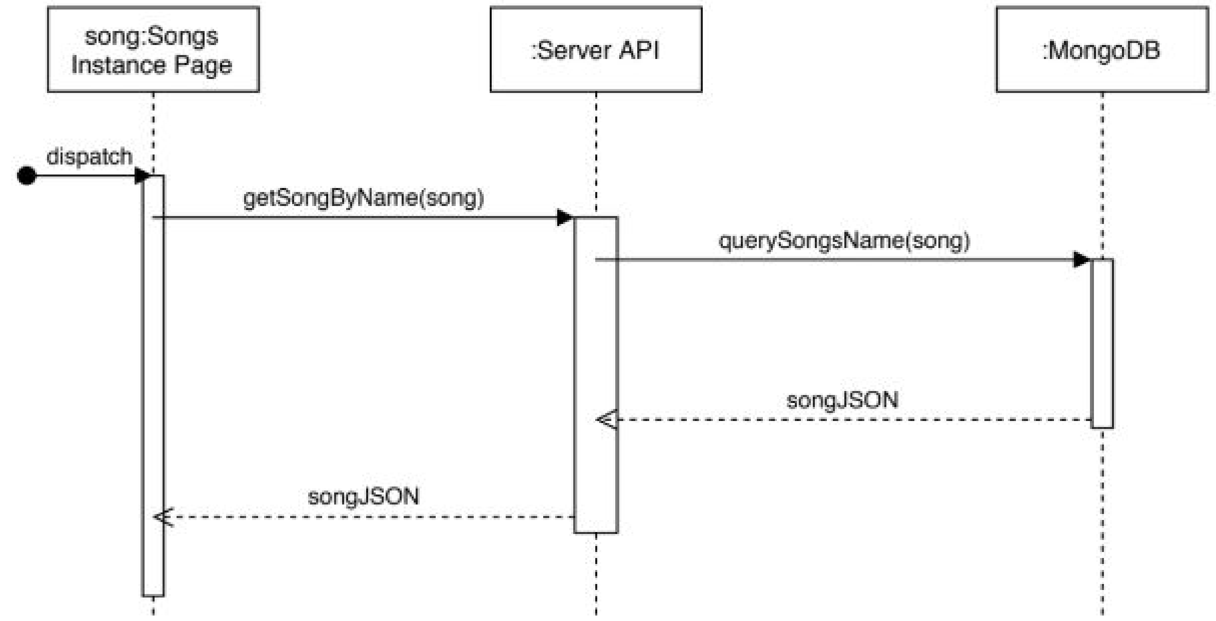
Loading Artists Instance Page:



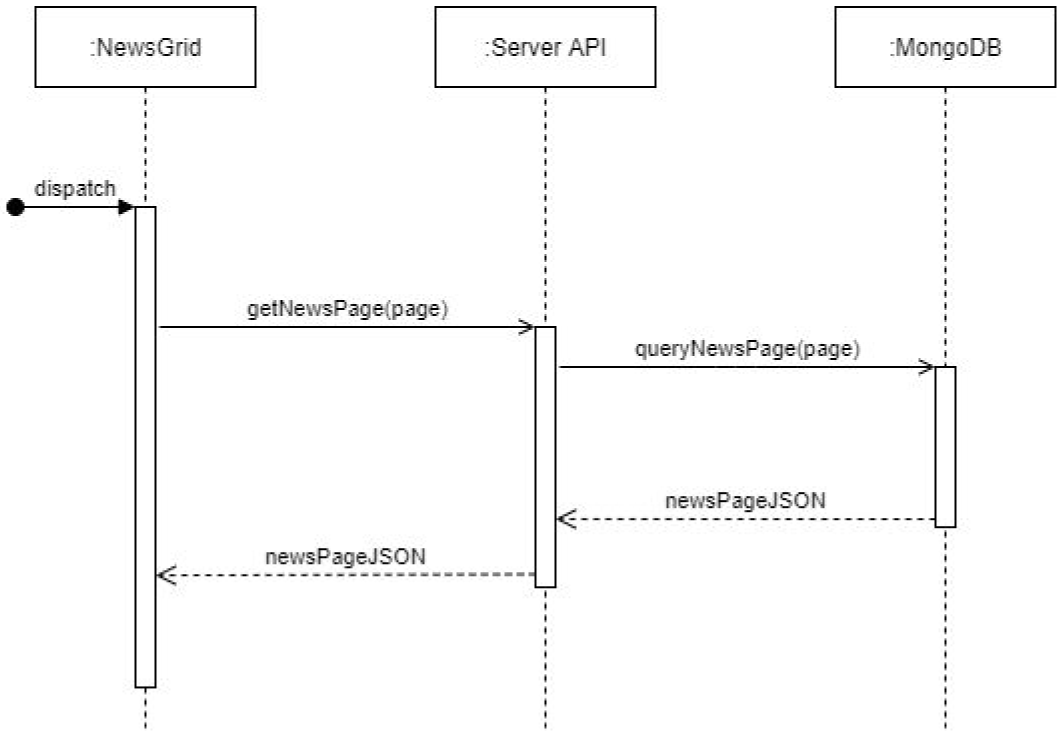
Loading Songs Page:



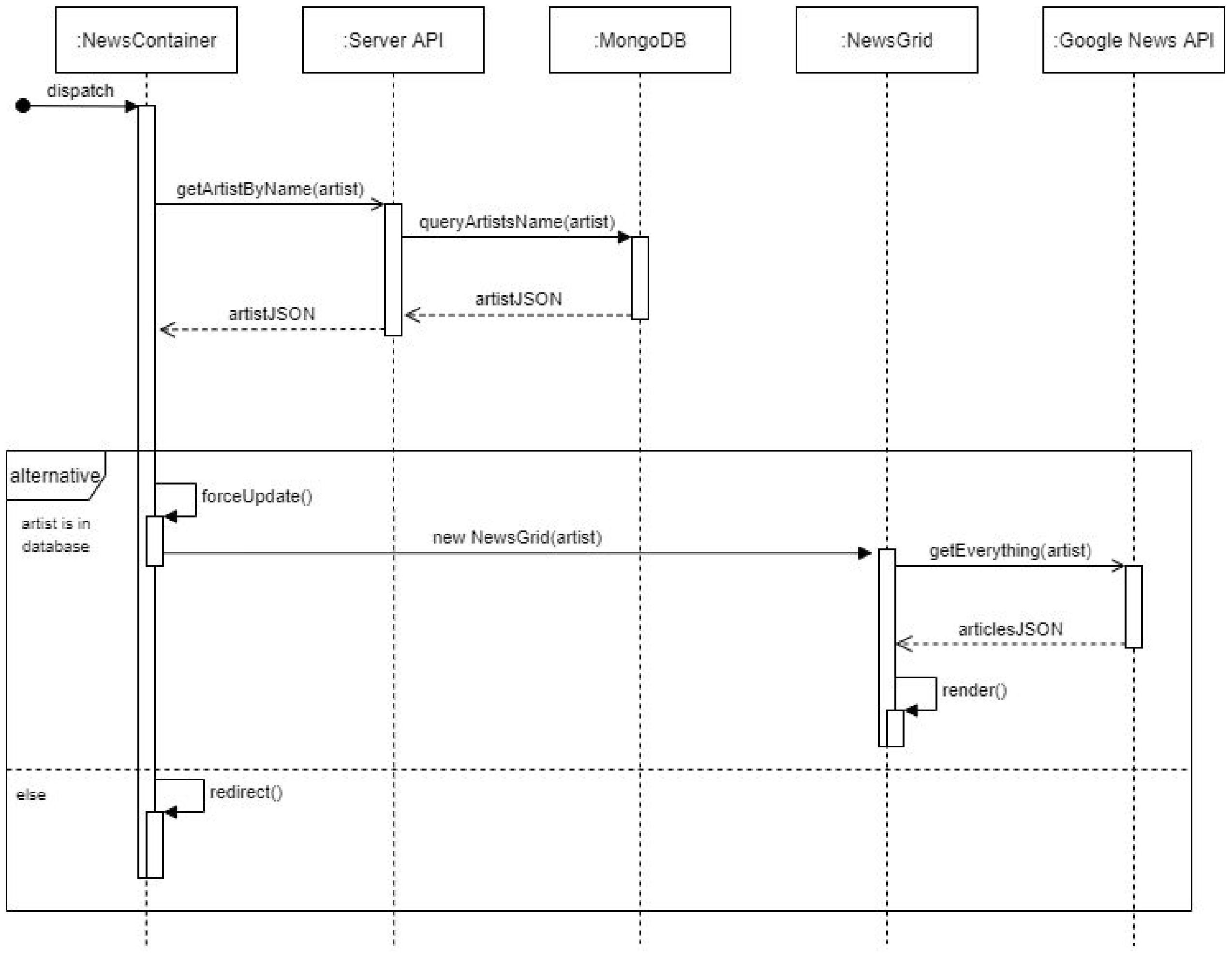
Loading Songs Instance Page:



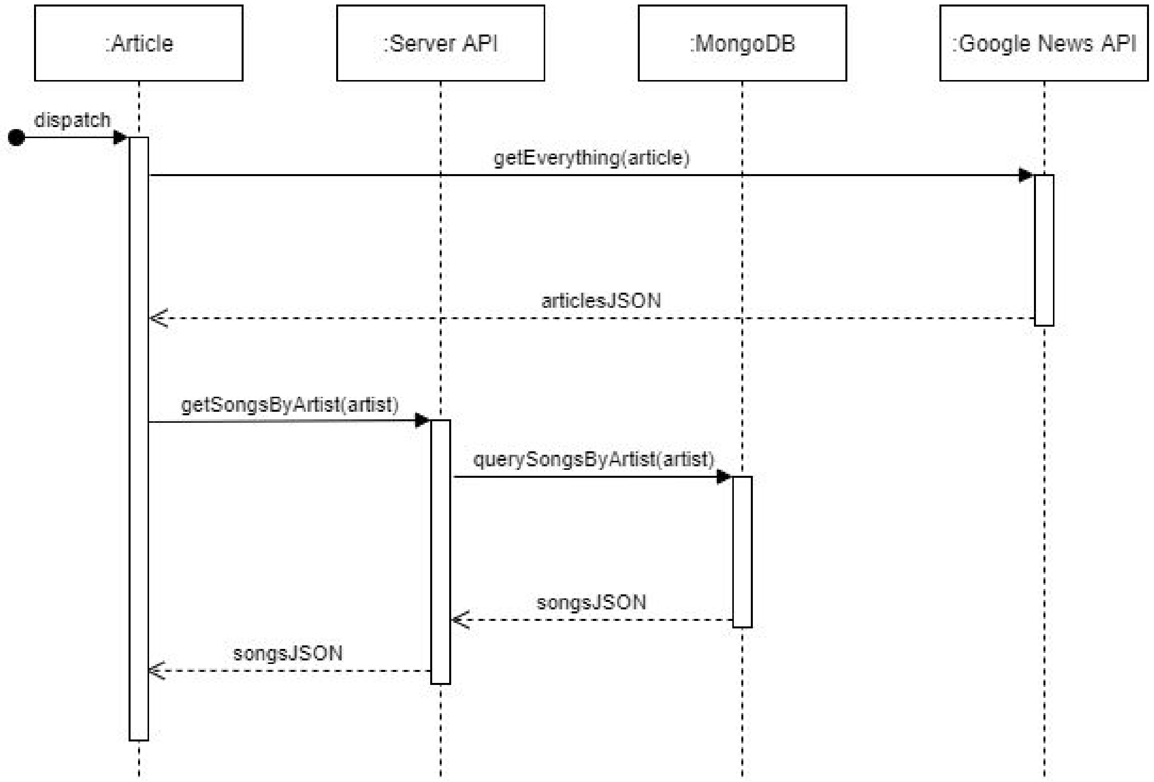
Loading News Page:



Searching News:



Loading News Instance:



TESTING

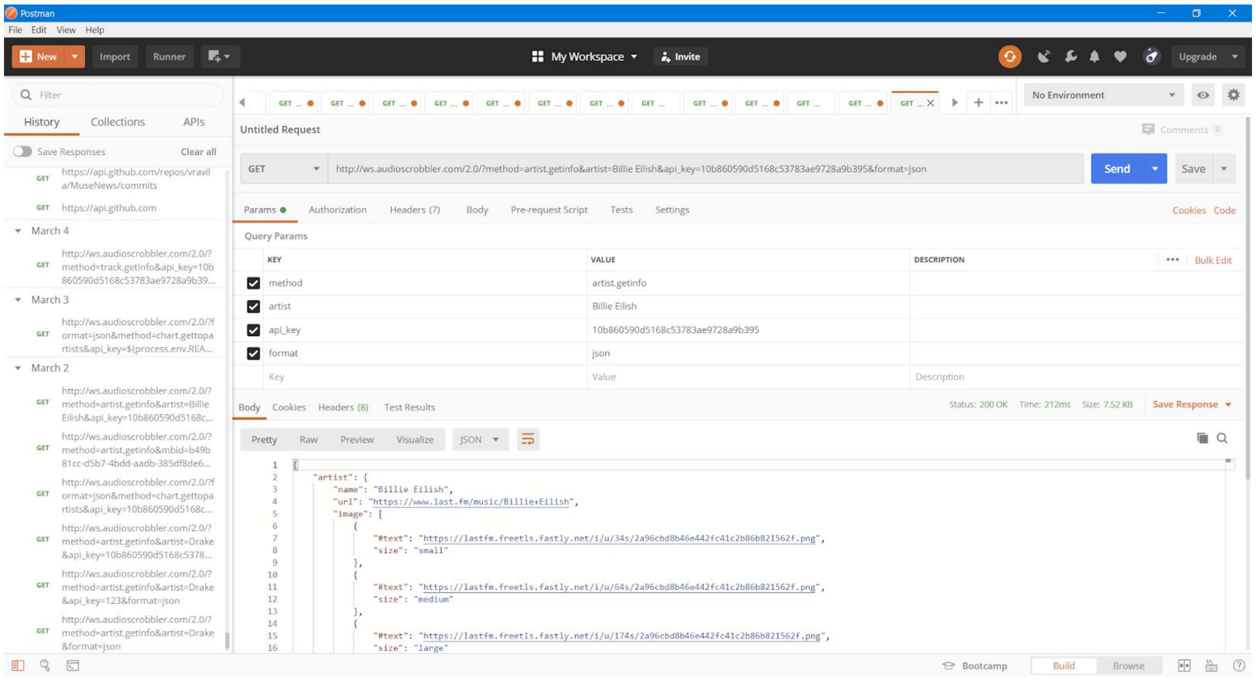
For Phase 2 of this project, we extensively tested for bugs in our code. We used Selenium tests to test our front-end code. We used Mocha tests to test our back-end Node.JS code by testing our endpoints. We used the Python unittest library to test our Python scripts that create the MongoDB database and populate its collections. Throughout the development process, we also used Postman to test the endpoints of the APIs we were accessing and the server API that we created.

Frontend

To test the front-end elements of the site, we created Selenium tests. For this phase, all of the elements that we could interact with were links or buttons, and we had no input boxes to test. Thus, our selenium tests tested the buttons and ensured that they linked to the correct link and that the correct data was displayed. We had a total of 14 Selenium tests for this phase. We tested the links between instances of pages (e.g. from artist to song, article to artist, song to artist). We also tested the previous and next page buttons for the artist and song pages. The one test where we did have an input was testing the news search bar functionality. We input “Lady Gaga” in the search bar and ensured that we got back results about Lady Gaga. We also tested the More button on the News model page to ensure that it goes to the correct News article instance. Finally, we tested the navigation bar links to ensure that they worked. The front-end was also thoroughly tested manually throughout the development process to find any bugs that the test cases may not find.

Backend

The back end of this phase was tested by Python unit tests, Postman, and Mocha tests. Throughout the development process, Postman was used to test that the API calls we made would work. For example, in the first image below, we make a call to the LastFM API where we are trying to get information about the artist Billie Eilish. We used Postman to see how the JSON was structured and write our code accordingly. Once we got our server API endpoints set up, we also used Postman to manually test them and make sure they were producing the right results. An example of this is seen in the second picture, where we make a call to https://musenews.appspot.com/api/artists/getArtistByRank/50". This returns a JSON object of the artist at rank 50, in this case Green Day. We looked at the JSON to make sure that it was formatted the same way as what the LastFM format was.



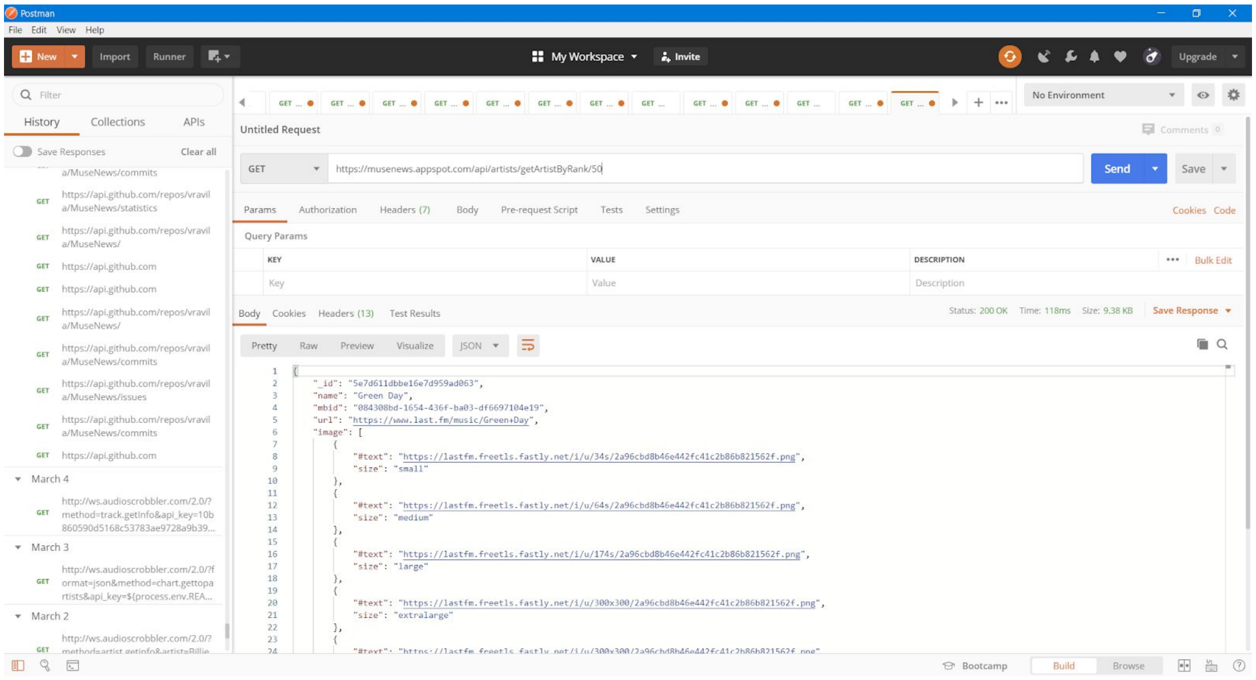
```
GET  http://ws.audioscrobbler.com/2.0/?method=artist.getinfo&artist=Billie Eilish&api_key=10b860590d5168c53783ae9728a9b395&format=json
```

KEY	VALUE	DESCRIPTION
✓ method	artist.getinfo	
✓ artist	Billie Eilish	
✓ api_key	10b860590d5168c53783ae9728a9b395	
✓ format	json	

Status: 200 OK Time: 212ms Size: 7.52 KB

```
{
    "artist": {
        "name": "Billie Eilish",
        "url": "https://www.last.fm/music/Billie+Eilish",
        "image": [
            {
                "#text": "https://lastfm.freetls.fastly.net/i/u/34s/2a96cbd8b46e442fc41c2b86b821562f.png",
                "size": "small"
            },
            {
                "#text": "https://lastfm.freetls.fastly.net/i/u/64s/2a96cbd8b46e442fc41c2b86b821562f.png",
                "size": "medium"
            },
            {
                "#text": "https://lastfm.freetls.fastly.net/i/u/174s/2a96cbd8b46e442fc41c2b86b821562f.png",
                "size": "large"
```

Image 1: GET request to Last FM API for Billie Eilish



```
GET  https://musenews.appspot.com/api/artists/getArtistByRank/50
```

Status: 200 OK Time: 118ms Size: 9.38 KB

```
{
    "_id": "5e7d611dbbe16e7d959ad063",
    "name": "Green Day",
    "mbid": "084308bd-1654-436f-ba03-df6697104e19",
    "url": "https://www.last.fm/music/Green+Day",
    "image": [
        {
            "#text": "https://lastfm.freetls.fastly.net/i/u/34s/2a96cbd8b46e442fc41c2b86b821562f.png",
            "size": "small"
        },
        {
            "#text": "https://lastfm.freetls.fastly.net/i/u/64s/2a96cbd8b46e442fc41c2b86b821562f.png",
            "size": "medium"
        },
        {
            "#text": "https://lastfm.freetls.fastly.net/i/u/174s/2a96cbd8b46e442fc41c2b86b821562f.png",
            "size": "large"
        },
        {
            "#text": "https://lastfm.freetls.fastly.net/i/u/300x300/2a96cbd8b46e442fc41c2b86b821562f.png",
            "size": "extralarge"
        },
        {
```

Image 2: GET request to server API for getting artist at rank 50 (Green Day)

To test the scripts that populated our MongoDB database, we created Python unit test suites. We have 3 files that populate our database: fillDBSongs.py, fillDBArtists.py, and fillDBNews.py.

We wrote 3 test suites that test different parts of the code that fills our database. All of these files are located in /muse-news/backend_unittests. In order to test fillDBArtists.py, we created a test suite called TestFillArtists.py. This test suite tests all of the functions involved in filling the artists collection. We test the call to Last FM to get the top artists on the chart and ensure that it returned a JSON object of the right format. We test the call to Last FM to get data for a certain artist and ensure that it has all information necessary for our website. We test the call to the Bing Images API and ensure that it returns a JSON object with a URL to an image that we can use for our website. Finally, we test that the script successfully populated our MongoDB collection by ensuring that it put 50 artists in the database. Since the songs collection is created the same way as the artists collection, we used 4 similar tests to ensure that fillDBSongs.py successfully filled the songs collection with 50 songs that are of the correct JSON format and have a URL from the Bing Images API. For fillDBNews.py, we only have two tests since the calls are formatted differently. We test the GetNewsArticle method to ensure that, when we call the Google News API with a certain search query, we get an article that has all of the fields necessary to populate our news article instance page.

Once we tested that our MongoDB database was filled properly, we created Mocha tests to test that our server worked properly. Our server was created using Node.JS and Express.JS, so it made sense to use Mocha to test that it worked. We used the Chai assertion library. The Mocha tests are located in the /muse-news/test directory and can be run with npm test in the /muse-news directory. We extensively test each endpoint to ensure that they work with the right data (return a 200 status) and fail with incorrect data (return a 404 status). To test the /api/artists endpoints, we have a total of 9 tests. We test the /api/artists/getArtistByRank endpoint with a valid rank (10) and invalid ranks (-10,1000). We test the /api/artists/getArtistByRankRanges endpoint with a valid range (11-20) and invalid ranges (-30 to -20, 1000 to 1015). We test the /api/artists/getArtistByName endpoint with a valid name (The Weeknd) and an invalid name not in the database (Kedar). We test the /api/artists/getArtistsTweets endpoint with a valid artist (The Weeknd). Since this endpoint will return tweets for any query, there is no invalid query. To test the /api/songs endpoints, we have 12 test cases that are structured the same way as the artists test cases (1 valid test case, 1 or more invalid test cases based on how queries can be partitioned if applicable). To test the /api/news/getNewsPage endpoint, we have 3 test cases that partition the input (valid, invalid positive, and invalid negative).

MODELS

The site has three models pertaining to songs, artists, and news. Each model is linked to other related instances from other models. For example, an artist page will link to the artist's songs and news about the artist.

Songs

The first model shows a song's title and rank on the charts, gathered from the Last FM API, and the album art, gathered from the Bing Images API. By clicking on the card, the user is taking to the instance page for that song. The individual song's page shows detailed information about that particular song. Data includes a description of the song, how many listens and plays it has, and its tags. This data was pulled from the LastFM API. In addition, the instance will have a link to a related news article and the song's artist, connecting it to the other models. Furthermore, tweets are displayed based on a query sent to the Twitter API to display relevant tweets about the song.

Artists

The second model shows an artist's name and rank on the charts, gathered from the Last FM API, and a picture of the artist, gathered from the Bing Images API. By clicking on the card, the user is taken to the instance page for that artist. The second model details an artist by getting their biographical data, number of listens and plays, tags, similar artists, and whether they are on tour through the LastFM API. In addition, each artist is linked to news and songs related to the artist in order to connect the models together. Furthermore, tweets are displayed based on a query sent to the Twitter API to display relevant tweets about the song.

News

The third model describes recent news related to the artists and songs stored in our database. The landing page of the news model shows a collection of articles about the top 10 artists. The user can then browse to other pages to see more articles, or the user can search for a specific artist. The articles are displayed as cards in a grid using data available through the Google News API. Each card in the grid shows an image, the article title, and a brief description of the article.

If the user clicks on a specific article, they are redirected to a page with more detailed information about the article. This page shows the title, author, date, image, and preview text associated with the article. It also has links to the original source of the article and related instances of other models. The artists associated with the article have a link and their top songs are also linked.

When the user performs a search, we check our database to determine if the search terms the user provided exist. If not, the user is redirected to a page stating that the search was invalid. In the event of a successful search, articles about the artist will be displayed for the user to view. The landing page of the news has data stored in our database, while individual searches call on the Google News API to generate the most recent news articles. This is illustrated in the sequence diagram below.

SOFTWARE

The bulk of the website's code is written in JavaScript using Node.js and React.js. In addition to those frameworks, we used four different RESTful APIs to gather the information necessary for the models and their instances.

Frameworks

In order to make the website easier to design and build we decided to use the MERN model (MongoDB, Express.js, React.js, and Node.js). For this phase specifically, we used the Node.js and React.js frameworks. These frameworks helped aid in the development of the pages and allowed us to integrate the APIs and JavaScript code with the HTML elements displayed on screen.

React

React is a JavaScript library that assists with web development. It includes functions that make pages easier to manage and makes code more readable. It also allows object oriented approaches to problems through the use of Components. It also provides libraries for numerous elements, which was helpful with front-end development. We used the fetch() functionality to make calls to our back end and get the data needed for each page. The usage of React.js can be seen in the /muse-news/client component. The front-end code is run through npm start in /muse-news/client.

Node.js and Express.js

Node.js is a JavaScript framework that helps with the back-end development of a website. The framework provides tools for building, compiling, testing, and running code. We used Node.js to create our server API that allows access to our MongoDB database. We use the Express.js framework to structure our endpoints in a clean manner, using routes. Node.js and Express.js were used together to create the endpoints necessary for our server. Their usage can be seen in the /muse-news/index.js and /muse-news/routes components. The back-end server is run through npm start in /muse-news

APIs

Five different API's were used in making this site. Each API related directly to the information being presented in one of our models.

Bing Images (Azure Cognitive Services)

The Bing Images API allows us to access images from the web. We use this API to get the album art of a certain song or a picture of the artist for a certain artist. We created a Microsoft Azure Cognitive Services Resource Group to use the Bing Images API.

Last.fm

The Last.fm API allows access to information about songs and artists. It is used in the songs model to get a description of the song, how many listens and plays it has, and related tags. It is used in the artists model to get a biography of the author, how many listens and plays they have, tags, similar artists, and whether they are on tour.

Google News

The Google News API allows access to hundreds of news articles from various sources across the internet. This API is searchable by a number or parameters which is used in the news model to get relevant news articles and to link directly to the source of the article.

Twitter

The Twitter API provides access to several of Twitter's core features. We implemented querying an artist or song name through the Twitter API to find and display relevant tweets.

Github

The Github API allows access to statistics about the development of the project. This is used in the about page of the site to show the contributions of each developer on the team in terms of commits and issues.

Tools

In order to aid the development of the site, our team used a variety of frontend and backend tools.

Bootstrap

Bootstrap allows for the creation of user-friendly interfaces by providing style sheets and JavaScript components. Bootstrap is used throughout the site to create a consistent look and feel. Components provided through bootstrap help create simple GUIs for the models and the individual instances.

REFLECTION

Looking back, our team did a great job with starting this phase early and making sure we added new functionality while fulfilling the phase's requirements. We were not very familiar with using Node.js, Express.js, and MongoDB (outside of the tutorial with Python), so there was a somewhat steep learning curve to develop the back end of the application. We also had to learn how to use the Python unit test library and Mocha. Overall, we did a great job of learning these technologies quickly and figuring out how to apply them to our application. We used Github better by using more branches and only committing code to the master branch when we were sure it worked. Looking to the future, we should do a better job of communicating the progress we are making towards our goals and coming together more often to plan out the phase. We could have done a better job planning out our work for this phase, as we had to change some stuff up at the end to ensure that everything worked smoothly.