Phase 3 Report

EE461L

Github repo: https://github.com/chrisjoswin/EE461L_Project

Team Information:

Jacob Grimm	jacobgrimm@utexas.edu	jacobgrimm
Jerad Robles	sebastian.robles@utexas.edu	JSRobles
William Gu	williamgu@utexas.edu	Minalinnski
Christopher Erattuparambil	chris.joswin@utexas.edu	chrisjoswin
Josh Kall	joshuakall@utexas.edu	j-ka11
Haosong Li	hl27346@utexas.edu	hdlee9885

Team Canvas Group: Morning-9

Project Name: Internet Comic Database (ICDb)

Project URL: https://icdb-phase1-270405.appspot.com/

Motivation and Users

Although there are many comic databases online, the official websites only have comics from the same universe while unofficial ones are usually messy and ad-heavy. Our team intended to integrate online sources and build a clean, user-friendly database for all famous comics. The intended users are all comic readers, no matter if they are fans of a character or a creator, or they are interested in reading the origins of the newly released movie.

User Stories

Phase III User Stories

- Customer: As a fan of Stan Lee, I want to filter Comic Issues by his Name, so I can read more comics he wrote.
    - Time estimate: 1 hr

    - Actual Time required: 1.5 hrs
- Customer: As someone who loves nicknames, I want to search by aliases, so I can find their accompanying character. (i.e. Searching Web Slinger returns Spider-Man)
    - Time estimate: 45 mins
    - Actual Time required: 30 mins
- Customer: As an amateur story writer, I want to sort author names in alphabetical order, so I can discover new authors.
    - Time estimate: 30 mins
    - Actual Time required: 30 mins
- Customer: As a Marvel fan, I want to search a character's superhero name, so I can find more information on my favorite characters.
    - Time estimate: 45 mins
    - Actual Time required: 30 mins
- Customer: As a fan of storylines, I want to search the title of a series, so I can see all the comic issues in that series.
    - Time estimate: 30 mins
    - Actual Time required: 30 mins
- As a fan of my favorite comic book, I want to filter by the issue name, so I see the characters associated with the issue.
    - Time estimate: 30 mins
    - Actual Time required: 30 mins
- As a follower of Jack Kirby, I want to search his name, so I can find and visit his Author page.
    - Time estimate: 45 mins
    - Actual Time required: 1 hr
- As someone who goes by a nickname, I want to search by an author's pen name/aliases, so I learn more about them.
    - Time estimate: 30 mins
    - Actual Time required: 45 mins
- As a novice comic reader, I want to sort comic titles in descending alphabetical order, so I can find some interesting issues.
    - Time estimate: 30 mins
    - Actual Time required: 15 mins
- As a fan of The Hulk, I want to filter comic issues that contain The Hulk, so I read more stories with The Hulk in them.
    - Time estimate: 30 mins
    - Actual Time required: 30 mins

Phase II User Stories

- As a fan of writers, I want to know when my favorite authors were born, so I can mark them on my calendar.
  - Time estimate: 30 mins
  - Actual Time required: 45 mins

- As an artist, I want to see an issue cover page, so that I can draw it and be a comic book artist one day.
  - Time estimate: 1hr
  - Actual Time required: 1.5 hrs

- As a fan of heros, I want to see their aliases, so I can understand which character a comic is referring to when an alias name is used.
  - Time estimate: 45 mins
  - Actual Time required: 1 hour

- As an artist, I want to know the general appearance of a character, so I can create my own version of the superhero using the hero's standard appearance.
  - Time estimate: 1hr
  - Actual Time required: 1.5 hours
- As a comic book newbie, I want to know the real names of all the superheros, so that I can understand who is who while reading comic books or watching movies.
  - Time estimate: 1hr
  - Actual time required : 2 hours

Phase I User Stories

- As a fan of Dan Slott, I want to search by his name so that I can find other works he wrote.
  - Time Estimate: 1 hour
  - Actual Time Required: 2 hours
- Customer: As a lover of Captain America, I want to find issues that he is featured in so I can read more comics with Captain America in them.
  - Time Estimate: 1 hour
  - Actual Time Required: 1.5 hours
- Customer: As an amateur comic writer, I want to learn more about who wrote my favorite comic issue so I can be more familiar with their work.
  - Time Estimate: 1 hour

    - Actual Time Required: 1.5 hours
- Customer: As a hardcore Iron Man fan, I want to read his bio page so I can learn everything about him.
    - Time Estimate: 2 hours
    - Actual Time Required: 2 hours
- Customer: As a new comic reader, I want to learn more about all the characters in my first comic book so I can broaden my comic knowledge.
    - Time Estimate: 1 hour
    - Actual Time Required: 1 hour
- As a comic collector, I want to find issues from the Silver Age (1970-1984) so I can see if my collection is complete.
    - Time Estimate: 2 hours
    - Actual Time Required: 1.5 hours
- As a comic enthusiast, I want to see issues where two of my favorite characters crossover so I can read them.
    - Time Estimate: 2 hours
    - Actual Time Required: 1 hour
- Customer: As an International Reader, I want to find writers from my country so I can read their comics.
    - Time Estimate: 2 hours
    - Actual Time Required: 1.5 hours
- As a casual comic reader, I want to find issues from a new series so I can start reading that series.
    - Time Estimate: 1.5 hours
    - Actual Time Required: 2 hour
- As a fan of hero groups, I want to see what groups a character belongs to so I can see how that character fits in the comic universe.
    - Time Estimate: 1 hour
    - Actual Time Required: 1.5 hours

Use Case Diagram

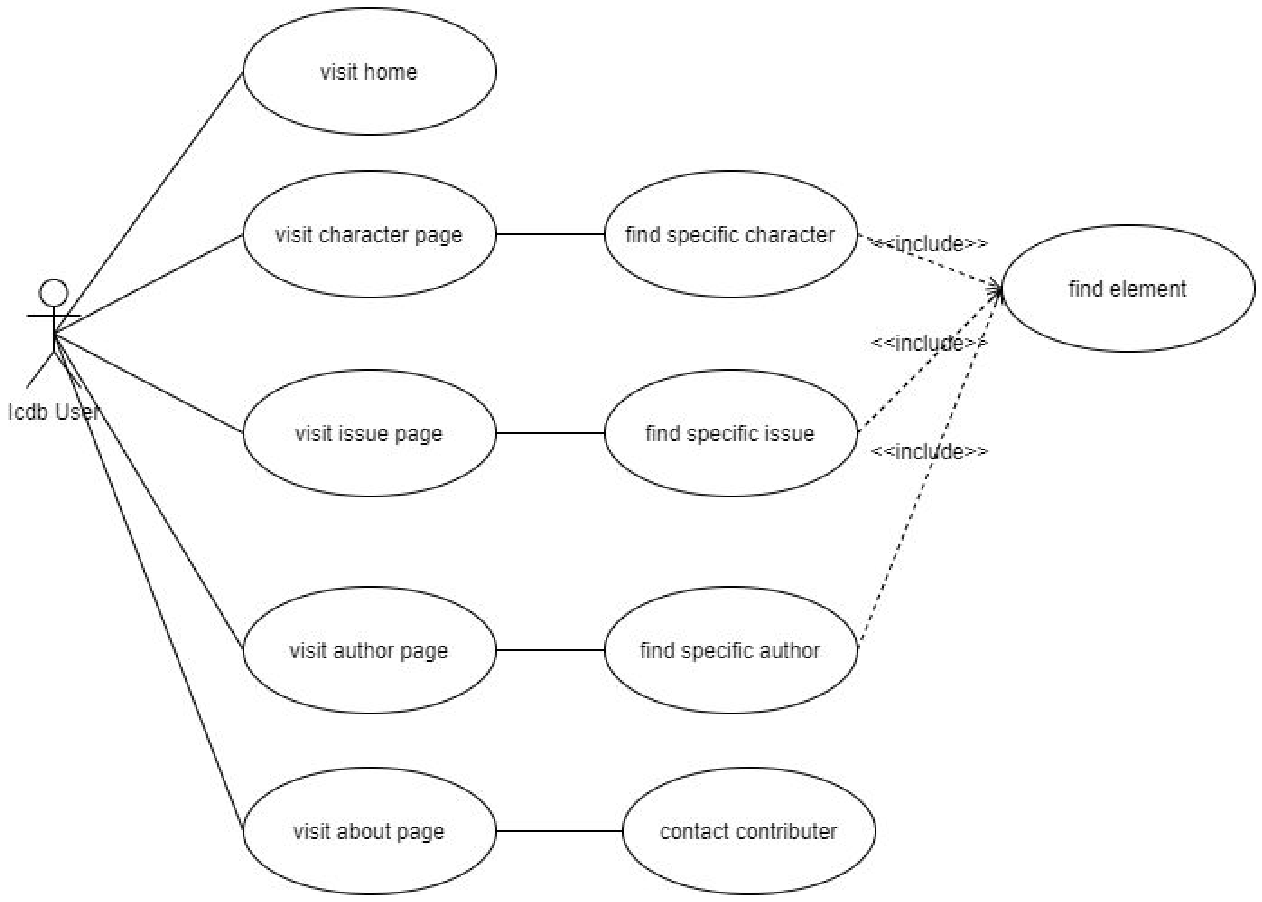

Figure 1: Use Case Diagram for Icdb user

Design

Front end:

The front end currently consists of 4 different types of pages constructed using Angular. The 8 pages are the home page, this provides a clean entry in our website and allows the user to have easy access to website features. The about page allows the user to learn more about our team and progress on the website's development. The model page provides access to all instances of that particular model, there are 3 of these pages, one for each model. Lastly is the corresponding instance page for each model, this gives the user all the information attached to that particular instance. We have created one instance page for each model and pass the information that the user clicks on the to instance page to be displayed. This way we would not have to generate a new html file as our collection of instances for each model grows.

Phase III: In phase III, we added a search page. When a user types a search term in the search bar, the Angular router navigates to this page and displays a preview of all the results that the backend returns. From here, a user may click on one of the search results and the application will detail that result. We also added a couple new components that aren't linked to their own page, specifically a search-bar component and a filter component. The search bar was inserted into every main page we have that has its own URL. This way, you are able to search our database no matter where you are in the application. The filter component was inserted into our three model pages that implement pagination. The search-bar component's responsibilities consisted of receiving the keyword that the user wants to search for and handing the term off to the parent. The reason the search-bar simply hands the input off to the parent is because some of the parents needed to handle the input differently, specifically the search-page parent. The filter component's responsibilities were similar to the search-bar component's in that it needed to receive the user input for the filter term, but the filter component is able to handle the input on its own since the three parents handle the input in the same way. To handle the filter input, we had our database service add a custom header to the GET request for pagination and set the value as the keyword. From here, we would receive a new, filtered list of items for our pagination pages.

We did not choose to implement sorting with a new component due to the fact we could simply add buttons to the HTML. When a user clicks the sort ascending button, we add a header to the GET request for pagination that notifies the backend we would like our results sorted in ascending fashion. A similar process occurs when the user clicks the sort descending button.

Back end:

The back end consists of several python scripts that pull relevant information to a specific model from a specified API and outputs the information into a JSON file. For the Character model the Superhero API is used, for the Issues model the Marvel Comics API is used, and for the Authors model the Comic Vine API is use. With these JSON files we build our MySQL database hosted on Google Cloud SQL.

Then to take requests from the front-end we built a Flask-API that receives the specific request that the front-end makes at designated endpoints. This API is hosted on Google App Engine, and connects to our MySQL instance. Once our flask-API receives a request, we then make the necessary SQL query from our API, format the data into a desired JSON instance.

<u>Phase III:</u>

Our essential changes we made was we added an extra endpoint for our search bar. ‘/search/<term>’ and in this endpoint we query the SQL database with the given search term passed in as <term> and look for the term in each table in our database.

Then for sorting and filtering through the model pages, we made one other essential change. We asked the front end to pass in the sort of filter term in the headers of the request they were making so the headers would be {‘sort’ : ‘<True or False’>, ‘filter’ : <filter term>}. And with these headers we would sort alphabetically if a sort header was passed in. And if a filter header was passed in, we would filter through our SQL tables looking for the filter term.

Overall these changes were quite necessary and these design changes helped make everything work.

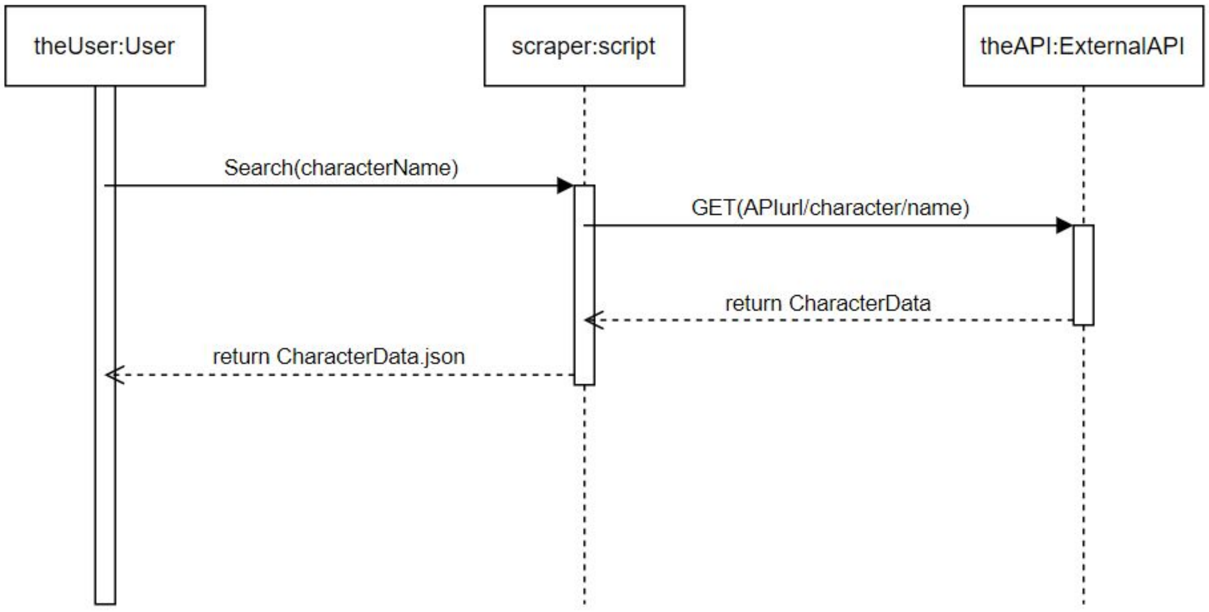

Figure 2: Sequence Diagram for data retrieval script. Scraper collects data on Instance and then returns it to the front-end for them to do the work.

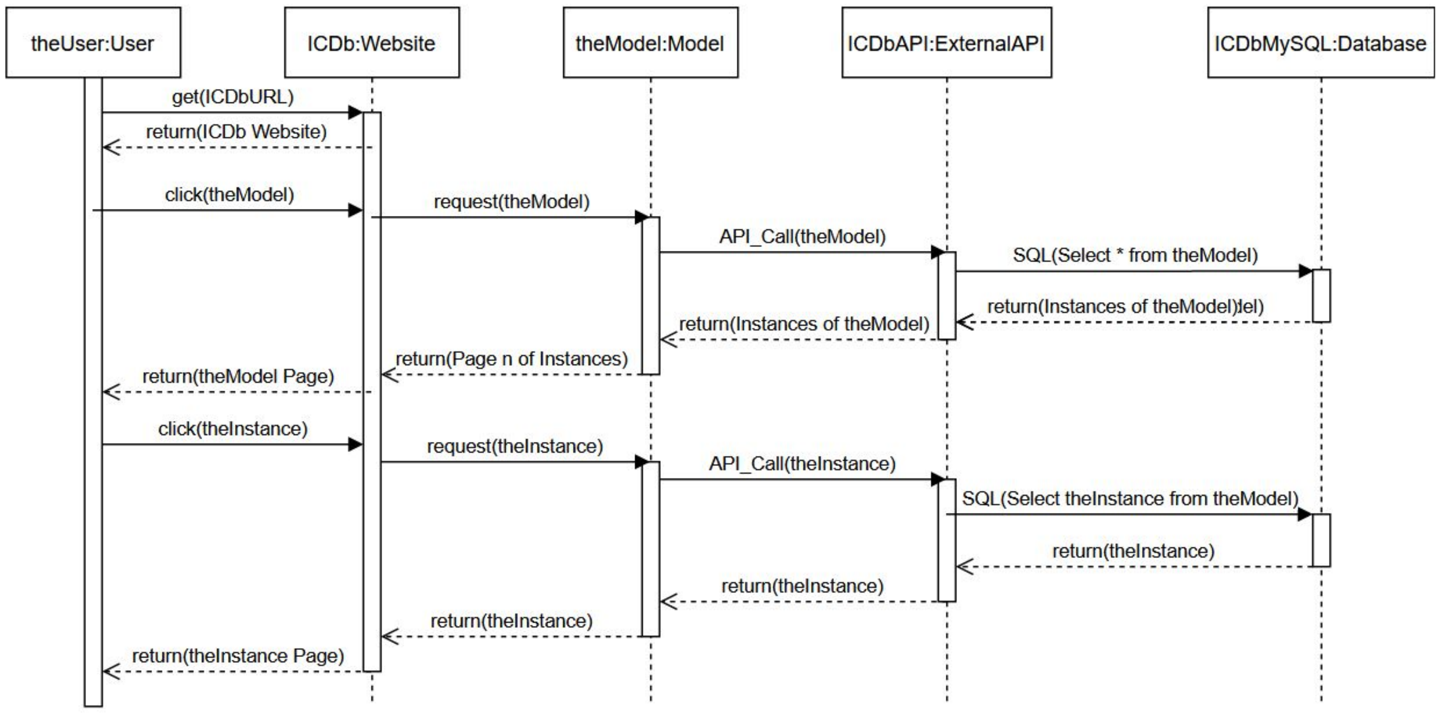

Figure 3: Sequence Diagram for Interactions between the user, website, API, and database. The diagram shows the user requesting the Model page and then a page of a particular Instance.

Pagination

The backend team provided an API endpoint that returns a list of model instances by page number. On the frontend, we created a component for each model's pagination page. When the component is initialized, we query the backend for the first page of model instances. To perform pagination, the component keeps track of a current page number and makes use of back and forward buttons. When the user presses the forward button, the component gets the next page of model instances as designated by the current page value and increments the current page value. We implement the back button in a similar fashion. We also used conditional directives in the component's view template to decide if we should render the back and forward buttons depending on the current page.

Phase III: We took feedback from phase 2 so that the results are now displayed in a grid format with 9 instances. We also added pagination to work on our new search page and made sure that pagination functioned properly with the addition of sorting and filtering on our existing model pages.

Testing

After our testing methods we are quite confident in the robustness of our web application and error-free programming that we have done. Of the many testing techniques we have done, the most significant front-end testing was clicking and examining the success of the links between each of our static pages. As well, we tested our navigation bar on both mobile and pc browsers, and the rate of reply was quick and it returned the proper responses.

In regards to our back-end testing, we tested our scrapers on a number of different API resources and each run returned exactly what we wanted. So we are fairly confident in the success rate of it being quite high.

Phase II GUI Testing

For GUI testing we used Protractor which is built on top of Selenium and specifically for Angular projects. We created tests that were user-oriented and tested the functionality of each button on each page. We also made sure to verify the content of each page was exactly what the user would expect once a certain click was made and that it was displayed properly.

Phase II JavaScript/TypeScript Testing

Angular provides its own testing framework that uses Karma/Jasmine. First we tested to make sure each schematic is able to be created. For the components, we just made sure to cover every statement since complex branching isn't present yet. To test the services that perform HTTP GET requests, we made sure that the requests were successful and that they were returning the correct json objects. The third service we had is in charge of storing state information, so we created some test objects to set and get to make sure it was working properly. An…interesting component to test was our component that handles the about page. In order to cover everything, we needed to create a rather large amount of test data. We gave this data to the component and ensured that it rendered the information correctly. In fact, this was where we found the most amount of bugs.

Phase II Backend Testing

We tested the built API with python unittest. We have a homepage that lists all instances we have in the database, and pages for each module and instances. We tested each api request has a valid connection to the database and not missing instances. We also tested the response data to ensure it's in the right format and contains all the information needed for frontend functionality. We also tested sql to ensure data is not written into the database redundantly. With these tests we figured out some files that break the scripter, which was very hard to track with error report on gcp.

Phase III GUI Testing:

For GUI testing we continued using Protractor built on top of Selenium. We basically created tests on newly-added features, filtering, sorting, and searching. For each new feature, we tested the functionality of each button on the page and we made sure the content of the page was correct. Moreover, we added input boxes for searching and filtering so that we tested to see if the values were input and the result was what the user wanted. Furthermore, filter buttons could be hidden when clicking on the remove button and we made sure this was implemented properly.

Phase III Javascript/Typescript Testing

In this phase, our javascript testing consisted of making sure our new components worked properly. With the search-bar component, we made sure that it was correctly receiving the input term and correctly giving the term to the parent. To test the filter component, we again made sure it was correctly receiving the user input. However for this component, we also made sure that it was making a correct call to our database service.

Phase III Backend Testing:

In this phase we continued using pytest to ensure all new API requests are working properly. For search we have a new api request that reads search context as well as pagination indexes. Test makes sure the server responds with all necessary results and adapts to all cases, ordering and substrings. For sort and filtering, it is sent by headers and the default is scrambled. If sort is true then return in ascending order, false for descending. Filter works similar to search but the term is passed by headers. In addition, we made sure the preflight request is properly responded by the database.

Models

Our database website consists of three models: Comic Book Characters, Issues, and Authors. Every instance of each model will contain data and media obtained from a source. The data and media for each model as well as the source will be as follows.

The Comic book characters model we are using will consist of a few distinct elements, with a few holding an important relationship to the others. The elements held in each hero model will consist of some of the superhero's fictional biographical data, it is fictional because as far as we know, none of these superheroes exist outside of their respective comic books or movies. As well, we will be including information about the powers they have, specific/notable issues that the respective character is featured in, as well as the author and the hero's first appearance. We will also include a nice thumbnail image of each character, so the user can better identify who the hero is. The sources for this information are from each the SuperHero API, the Marvel Comics API, and the ComicVine API.

- Comic Book Characters
  - Data: Biographical information, powers, issues appeared in, name of the creator, and creation date, appearance.
  - Multimedia: Image of the character, alternate images, and video if available.
  - Source: SuperHero API

Another model we are opting to pursue, is individual comic book issues, and the information associated with them. The certain elements of each model include, the name of the issue, the name of the series that the issue is a part of, a meaningful summary/description of the issue, the characters included in the issue, and the writers of the issue. As well, we plan to include a picture of the front cover of each particular issue so that the user will be able to recognize if they have viewed it in the past, and also understand the artists' style a little better. This information will be gathered from the Marvel Comics API and the ComicVine API.

- Comic Book Issues
  - Data: Name of Issue, name of Series, description of issue, characters featured, and story authors.
  - Multimedia: Image of the Issue cover, frames from comic if available, and series image if available.
  - Source: Marvel Comics API

Our final and least significant model is the Comic Book authors model. The elements we will try our best to include are the author’s biographical data, which is not fictional, because the authors are real people. This is a stark contrast to the existence of the characters, a powerful irony considering the authors write the characters, but the characters do not exist in our sense of existence. As well we will add elements of what specific issues the authors wrote, the characters they have helped write for, and the social media handles of each other, in case the user is inspired to tweet at the writer. We also plan to include images of the author, of the covers of the comics they made, and interview videos they are featured in (if available). The primary source for this information will be the ComicVine API, with maybe some slight help from the Marvel API.

- Comic Book Authors
    - Data: Biographical information, issues written and characters affiliated with
    - Multimedia: Image of Author, covers of issues written, and interview videos if available.
    - Source: Comic Vine API

A very interesting aspect of our models is that they are all interconnected through one aspect of each other. The author models are connected to the issues and characters, through the issues and characters they’ve written. The character models are connected to the authors and issues, by the authors that have written them, and the issues they are featured in. And the issue models are connected to the characters by the characters in the specific issue, and the authors by the author of each specific issue. The significance of this is that we will be able to create convenient and nice hyperlinks to the author models quite easily, making our website more robust.

Tools, Software, and Frameworks

The tools we used for this section were quite interesting. For the frontend we elected to use Angular. In the implementation, it proved somewhat difficult to configure and setup, but it seems as though the efforts will be worth it for the ensuing phases. Angular is a front-end framework that uses javascript to help you make more robust websites. The frontend team really learned a lot about angular and how to develop a nice website with angular components. We used Protractor, Jasmine and Karma to do testing on our Angular app.

Another tool we used was more in the backend, but we programmed a few python scripts to scrape the necessary data we needed from the APIs. This proved to be surprisingly interesting! The scripts essentially made the calls, grabbed the We really enjoyed writing the scripts and cannot wait to implement them into a pipeline next phase to put the data we acquire from the API into our database!

One final tool we used was Postman, we used this to test out API calls that we were trying to make and examine the return content. Postman is a tool that can make API calls from a very friendly user interface, it is very useful for API testing. We found it quite useful just to ensure that we were calling the APIs properly.

Phase II Frontend Tools

For testing the GUI portion we used Protractor which is similar to Selenium but is specifically designed for Angular projects. Protractor allows for Angular designs to render unlike Selenium. We used Protractor to create test suites and run individual tests on the front end of our website. By using this software we were able to easily simulate a user experience with our website.

For the JavaScript/TypeScript portion of our code we are using Jasmine and Karma which is similar to Mocha. One of the reasons we decided to use Jasmine instead of Mocha is because Jasmine is set up with our Angular command line interface. Jasmine also gives us the capability to test our typescript files as we do not have any explicit Javascript files in our Angular project. Karma is our test runner for Angular projects and can also be accessed through the Angular command line interface. By using Jasmine and Karma we were able to make sure that our data is being properly received from our backend and is accessible and displayed on our front end. It helped in verifying that data being passed from component to component is being done correctly.

In addition to testing, we started using the Material module in the Angular dev environment. The Material module is extremely vast, so we mainly chose to utilize their table and card components. This module is useful for displaying data in a user friendly way.

We continued our usage of Angular’s HTTP service and Observables in conjunction with Interfaces. Since our data is not hardcoded into the frontend anymore, we need to continuously make GET requests to our backend to provide data for the user.

Phase II Backend Tools

For the construction of our backend API, we used Flask for python. We chose to use it because of its simplicity, along with the super easy deployment and maintenance that comes along with it. Defining endpoints with flask is super easy and straightforward, all that's required is the tag, @app.route(), to set up URL mappings to functions. And the biggest example of it's simplicity is using Python, one of the simplest languages to program in.

As well, we used MySQL to create the database. We chose MySQL, because we liked the idea of storing our data in a relational-database, and some of us had previous experience making SQL schema and maintaining a SQL database.

To connect our Flask-API to our MySQL instance, we used the Python module SQLAlchemy. SQLAlchemy has a ton of functionality that we have not yet implemented, but currently we have been using it as our SQL client to make whatever query we need. It functions by providing it with the login info for the SQL server, then it establishes the connection, and then we can make queries. It returns the answers in an easily accessible format, so all we have to do is build our JSON file with the data we are provided, and send it to the front-end.

Phase III Frontend Tools

In this phase, we mainly made use of two new features of Angular. The first feature is Angular's ability to create custom HTMl tags that come with their own HTML, CSS, and Javascript. These child tags can then be inserted into parent components. The two new components we created that implement this were the search-bar component and the filter component, as stated in the design section. Angular provides a buffet of communication methods between parents and children like this, and we only got our feet wet with everything we could do.

The other feature of Angular we made use of was their interface for adding headers to http methods. Angular provides constructors that create header objects for us, which come with a handy append method that lets us add any header we want. This is where we ran into issues with CORS, which ultimately was fixed on the backend.

Phase III Backend Tools:

In this phase we utilized the same core tools of FLASK and SQLAlchemy, but we expanded our usage heavily.

In flask we kept running into CORS issues when our front-end was making requests to our API, so we imported this new python module called flask_cors, when you initialize your flask-app with flask_cors at bootstrap of the server, the flask_cors module automatically takes care of all issues, so that you don't have to worry about CORS errors at all!

In the previous phase with SQLAlchemy we didn't utilize complex SQL queries, and instead relied on simple **SELECT * From *Table*** queries which we would then disect from the python code. We realized we weren't optimizing SQL to the best of our abilities, when SQL is built to handle difficult queries more efficiently then we are able to. So we implemented much more advanced queries to lower the amount of necessary work to be done on our python server side, and instead shifted that work to the SQL server.

Reflection

Our team performed very well in bringing our individual skill sets together. We did not run into any problems with getting particular technologies to work for us and because of this, we were able to complete a lot of the technical work fairly quickly. One thing our team needs to improve on is our communication, especially between sub-teams. There were a couple of instances where the front-end team did not know what the back-end team was doing or what technologies they were planning on implementing and vice-versa. It's not easy to move around and understand the new techniques we just learned for either frontend or backend in a short time, but we should try to eliminate the cutoff between sub-teams by understanding the othere's works. We learned that skill-wise our team is very capable, but that we need to work on improving our communication and scheduling of project components and team deadlines.

Phase II

Our team once again worked great as a team to finish this project especially considering the pandemic that is currently sweeping the nation. The team really committed to communication through slack as usual, but most importantly Zoom, as we are unable to meet in person. Even though it becomes much harder to be productive as a group through only these methods we were able to overcome technical issues and obstacles to get this phase done. Due to everyone being on Spring break we all took some time off and ended up deciding that the front and back end teams stay the same so that we can continue to function at a high level. This worked well, although each team is still not in full understanding of how the other team is able to make their part work. The frontend team worked well together to learn new software products to test Angular applications and we primarily expanded our Angular knowledge into State

Services and Angular Material Design. The backend team also was very collaborative and learned new software such as Sql to get our database up and running. Since we were all apart we were committed to making our daily standups at a scheduled time and it was pretty useful.

Phase III : After months of work as a team, we have created a beautiful website with lots of functionality. We were able to more effectively communicate with each other as we got used to virtual communication. We also had a zoom session in which the backend team taught the frontend team about the basics or what they did and the frontend team then explained how Angular and database calls were used to create the final product. This helped everyone's understanding and was helpful down the road as it was easier to explain team needs to each other when problems arose. The frontend team learned how to bind functions in Angular, use the EventEmitter object and use headers in the Http request for getting information from the database. In our backend, we delegated roles quite wonderfully, having worked together for almost a semester, we have learned how to best accommodate each of our skill sets to maximize efficiency. The backend learned more about the utility of SQLAlchemy and dealt with certain obnoxious CORS errors together that allowed us to grow incredibly in our understanding of backend API development.   Looking back it was a great learning experience for us all and we really enjoyed working as a team to accomplish making this website.