Phase III Report

Team Information

Github Link: https://github.com/Mattgallant/AustinConcerts

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

Canvas Group: afternoon-2

Project Name: Austin Data Bass

Name	**EID**	**GitHub**	**Email**
Matt Gallant	mdg3344	mattgallant	mattgallant24@gmail.com
Dylan Wolford	ddw2379	dylanwolford	dylanwolford@mac.com
Zander Tedjo	zbt86	zandertedjo	zander.b.tedjo@gmail.com
Will Worthington	wjw692	willworthington	willworthington99@yahoo.com
Michael Hilborn	mth2433	michaelhilborn	michaelthilborn@gmail.com
Guy Farmer	gcf375	farmerguycf	farmerguycf@gmail.com

Motivation

Our site serves as a database for all of Austin's upcoming Concerts. We aim to make finding upcoming Concerts a seamless experience by providing all of the Venue, Artist and Concert details in one easy-to-use website. We know how hard it can be to find all the information about one Concert in one place. Our users are anyone from an avid Concert-goer, an Artist looking to find a Venue or someone looking for their first Concert to attend.

Requirements: Users

Our User Stories

Phase I

1. As a person who has bought tickets, I want to see information of the event, including the artist, venue, and details, so that I can be prepared for the event on the set date.
    - 2 hr estimated, 4 hr actual
    - We are assuming the user would find all of the information they need across the three models we have chosen. The primary model they would visit would be the concert model, and then relevant links to the other models would be listed.

2. As an owner/manager of the venue, I want to see and manage the information about my venue so that it accurately reflects my location and business, and can improve the management as needed.
    - 1.5 hr estimated, not yet implemented fully
    - We are assuming this would be a pretty difficult feature to add and we have not yet added. This would require a lot of additional features to be added…. At the very least, an owner/manager can see their information, just not manage it.

3. As a manager of the artist, I want to see and manage the information about the artist I am affiliated with so that the information accurately reflects my client and presents positive PR.
    - 2 hr estimated, not yet implemented fully
    - We are assuming this would be a pretty difficult feature to add and we have not yet added. This would require a lot of additional features to be added…. At the very least, a manager can see their information, just not manage it.

4. As a fan of the artist, I want to see where my favorite artists have performed and are going to perform next so that I can obtain tickets and see them live, as well as find more information about them.
    - 1.5 hr estimated, 1 hr actual
    - We are assuming this requirement would be fulfilled by the artist model instances. Each artist model will link to upcoming concerts to the concert model where they can see all the information they need. Since our site is currently focused on upcoming concerts, we have not yet implemented seeing past concerts.

5. As a fan of the venue, I want to know what interesting performances are coming to that venue so that I can obtain tickets and see how the venue adapts to different types of performances and artists.
    - 2 hr estimated, 2 hr actual
    - This requirement will be fulfilled via the venue model. The venue model will display all relevant shows

Phase II

1. As a concert goer, I want to see data about concerts, venues, and artists in a well-organized grid format.
    - 2 hr estimate, 2 hr actual
    - For this we decided to make a 3x3 grid of 9 instances per page. This was also an official assignment requirement.
2. As a music enthusiast, I want to see a wide variety and many instances of concert, venue, and artist data.
    - 50 hr estimate, 70 hr actual
    - This involves getting all of our API data and getting into the database. We also made scripts to get all of the upcoming concerts and the corresponding venue and artist data. After this, we need to display all of our database data onto the frontend in a clean and organized fashion
3. As a concert browser, I want to see data broken up into individual pages for easier browsing.
    - 1 hr estimate, 1 hr actual
    - We are assuming that this means breaking up the Model index pages into a grid, and also have each concert, artist and venue have a unique page for each instance.
4. As a detail-oriented person, I want to see 3 attributes per instance of each Concert, Venue, and Artist on the respective grid pages.
    - 30 hr estimate, 50 hours actual
    - This story requires pulling all the data from APIs and storing it in the database. We then will create a grid page using HTML, CSS, Bootstrap and Django's template language to display the information and the 3 attributes per instance.
5. As a visually oriented person, I want to see a picture for each instance.
    - 1 hr estimate, 2.5 hr actual
    - This involved getting pictures from both the spotify API and yelp API. We simply passed image links into the database and then into the frontend in order to display the images.

Phase III

1. As an interested music fan who is easily overwhelmed, I want to be able to filter concerts so I can easily decide which concert to attend.
    - 5 hr estimate, 10 hours actual

    - This involved filtering the database based on button inputs from the concert grid page. Using a time object will not display any past dates.
2. As a person wanting to explore more artists, I want to sort artists by popularity so that I can check out other popular artists that are coming to Austin.
    - 3 hr estimate, 7 hours actual
    - This pulls the artist's popularity from the spotify api, it is loaded as a property of the artist model. The sorting accesses the database and displays the artist based on the input from the sorting section
3. As an Austin music fan with a specific concert in mind, I want to be able to search for that concert and see the information about it.
    - 5 hr estimate, 7 hours actual
    - This takes in a text input from the user and compares the name with concert, artist, and venue models
4. As a concert-goer that will be in Austin later this year, I want to see more concert instances for dates all throughout the year.
    - 10 min estimate, 1 hour actual
    - This accessed the data in the database based on the upcoming dates and used it based on a time object
5. As a high-maintenance concert-goer, I want to filter venues by rating so that I can assure that I am at a good concert location.
    - 5 hr estimate, 10 hours actual
    - This compares venue ratings using both the yelp api and the venue models in the database

Customer user stories

Phase I

1. As someone who wants to go to a concert, I want to see upcoming concerts.
    - 1 hr estimated, 30 min actual
    - We display this information in multiple different places. We assumed that the user would want to see upcoming concerts for each artist and for each venue, which we have included. The user can also see all upcoming concerts through the Concert model itself.

2. As someone on a budget, I want to see the price of concert tickets.
    - 1 hr estimated, 1 hr actual
    - We have had trouble getting the exact ticket price of a concert via an API. For now, we are simply including a price range of the venue on the venue instance. We are assuming that this will suffice as a ticket price.

3. As a concert goer, I want to see where the concert venue is located.
    - 1 hr estimated, 20 min actual

    - We will complete this requirement both by displaying the address of the venue and also by including an interactive map that displays the surrounding area.

4. As someone concerned about the weather, I want to know if the venue is indoors or outdoors.
    - 1 hr estimated, 30 min actual
    - This information is hard or impossible to find from an API. We assumed that a simple picture of the venue would be enough to determine this information.

5. As a fan of specific genres, I want to know the genre of the concert and artist.
    - 1 hr estimated, 1.5 hr actual
    - This user story is fairly straight forward. We will simply include the genre(s) of each artist on their individual instance. The concert instances will link to the artist instance and will therefore be able to see genre for the concert.

Phase II

- Not required for this phase

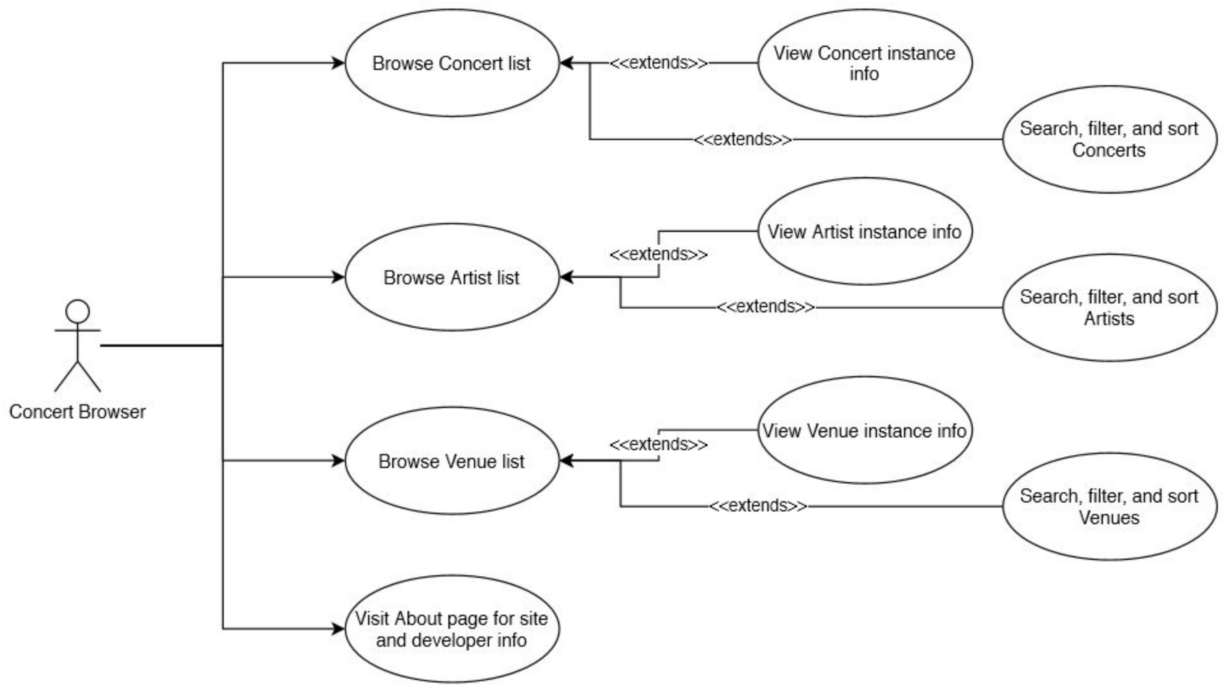
Phase III

1. As a concert goer, I want to be able to search for artists and venues in my area.
    - 1 hr estimated, 1 hour actual
    - This takes in a text input from the user and makes a keyword search that compares the name with concert, artist, and venue models
    - The search implements Django's SearchVector, SearchQuery, and Rank libraries to add weights to the models' data fields, so the keyword search returns a sorted list of relevant instances to the user
2. As a punctual concert goer, I want to be able to filter my search results by its date of event and start time.
    - 1.5 hr estimated, 6 hour actual
    - This involved filtering the database based on button inputs from the concert grid page. Using a time object will not display any past dates.
3. As a person who is not technologically savvy, I would like a reset button that will let my filter results go back to default.
    - 15 min estimated, 15 min actual
    - This took a button input and reset the page to its original results without any parameters. It simply redirects the user to the grid page before any filter/sort parameters were entered.
4. As a person who likes to ball on a budget, I would like to filter my venue choices by price and popularity.
    - 5 hr estimated, 10 hour actual
    - This compared the venue object's price and popularity and sorted it based on user input, then the result was loaded into the venue instance page from the database.

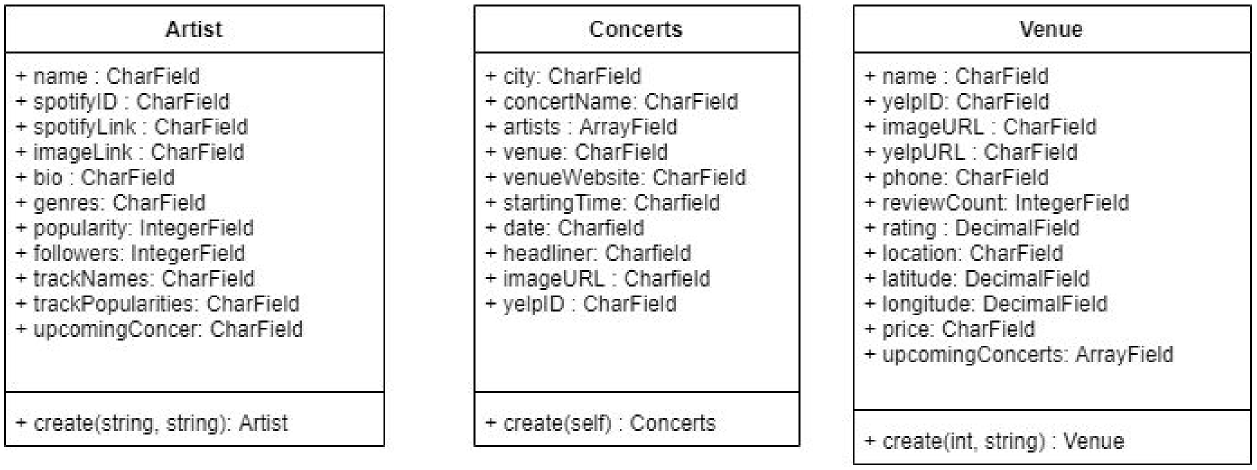
5. As a person who likes all information in one place, I would like a map indicating the venue available on the website.
    - 1 hr estimated, 30 min actual
    - This required us using the coordinates given to us by the Yelp API, along with Google Maps API. This was easily implementable with the Google Cloud Platform: we merely inserted the correct function to get a map display, with our venue’s latitude and longitude as input parameters.

Design

UML Use Case Diagram:


Browse Concert list
<<extends>>
View Concert instance info
<<extends>>
Search, filter, and sort Concerts
View Artist instance info
<<extends>>
Browse Artist list
Search, filter, and sort Artists
<<extends>>
Concert Browser
View Venue instance info
<<extends>>
Browse Venue list
Search, filter, and sort Venues
<<extends>>
Visit About page for site and developer info

UML Class Diagram:



The application uses data from instances of three different models, Artist, Concert, and Venue, to generate a dynamic website that allows for users to learn more about upcoming Concerts in Austin. We pulled our information from three different APIs: Spotify, Songkick, and Yelp. The information for each instance was stored in a PostgreSQL database. By storing the data in this database, we are able to dynamically populate the pages to display all instances of each Model in a paginated fashion, as well as create the individual instance pages based on a universal template specified for each Model. We paginated the instances into pages of 9 instances each via a python library. We hosted our website on Google Cloud which allows us to access all necessary servers and functionality. We also pulled from the GitHub API to get our repository stats, but this information was directly fed to the frontend rather than stored in our database.

USER EXPERIENCE:

To access the Austin Data Bass, a user first lands on the splash page, which includes a carousel of relevant images and a navigation bar to access the three pages of model instance lists, as well as the About Us page. Clicking on the About Us page provides the user with information about the creation of the website and the six developers of this project, complete with a picture, bio, and numbers of commits, issues, and unit tests.

Through the use of the nav bar, the user can click on the Artists tab to be directed to the list of all Artists in the database. This list includes the name of the Artist, hyperlinked to their instance webpage, their genre, and their relative popularity percentage. Clicking on a specific Artist takes you to the instance page of that Artist, which includes an image of the artist, a short bio, and a collection of stats from the Spotify API. Lastly, the Artist instance includes the Upcoming Concerts in Austin, with a link to the Concert instance page and the Venue instance page for the Concert on that specific date.

A user can either click on links to one of the specific instances of the models, or use the nav bar to access the list of model instances for the other two models. By clicking on the Venue tab in the nav bar, the user is taken to the list of Venues in our database, with a preview of the Venue on the right hand side of the web page. Clicking “See Venue Page” takes you to the specific Venue instance that you have selected. On this page is an image of the venue, with the Location and Contact information below, alongside a map of the location in Austin. There is also a rating of the Venue from the Yelp API with a link to the Yelp page for that Venue.

In the section below, we have a list of upcoming Concerts for that Venue, where a user can click on the hyperlinked date to be directed to the Concert instance on that date, or the user can click on the hyperlinked Artist name to be directed to that Artist’s instance page. Finally, we have compiled other relevant information at the bottom of the page, including the Venue Capacity, Food & Drink available, restrictions on allowed items, and the year the Venue opened.

The user can click on a specific instance of a Concert or an Artist from the Venue instance page, or they can click on the Concert tab in the nav bar to be directed to the list of Concerts in the Austin area. From the lists of concerts, you can click on an upcoming concert to be directed to the Event instance page. On each Concert instance page there is an image of the Artist performing at the Concert, a hyperlink to the Artist’s instance page, and the Event date and time. The Venue is listed below and is linked to the webpage of the Venue instance. Lastly, the Ticket Price Range and Opening Performances are included as well.

We have also added searching to the upper right of the navbar. The search bar allows you to choose which model you would like to search, or to search through all three models at once using the “All” option. The search bar will match your keywords to information in the given model and return the most relevant results broken up by model. The user can then click on the given instance to be directed to that instances page, just like our model grids.

The user also has the option of filtering and sorting each model using relevant options based on each instance. For example, the artist model allows you to filter by the Spotify popularity level and the genre, and sort by popularity, followers and name. Using these options will reload the grid page with the relevant results and card order. The user can also click the center “Reset” button to return the grid to its original unsorted and unfiltered state.

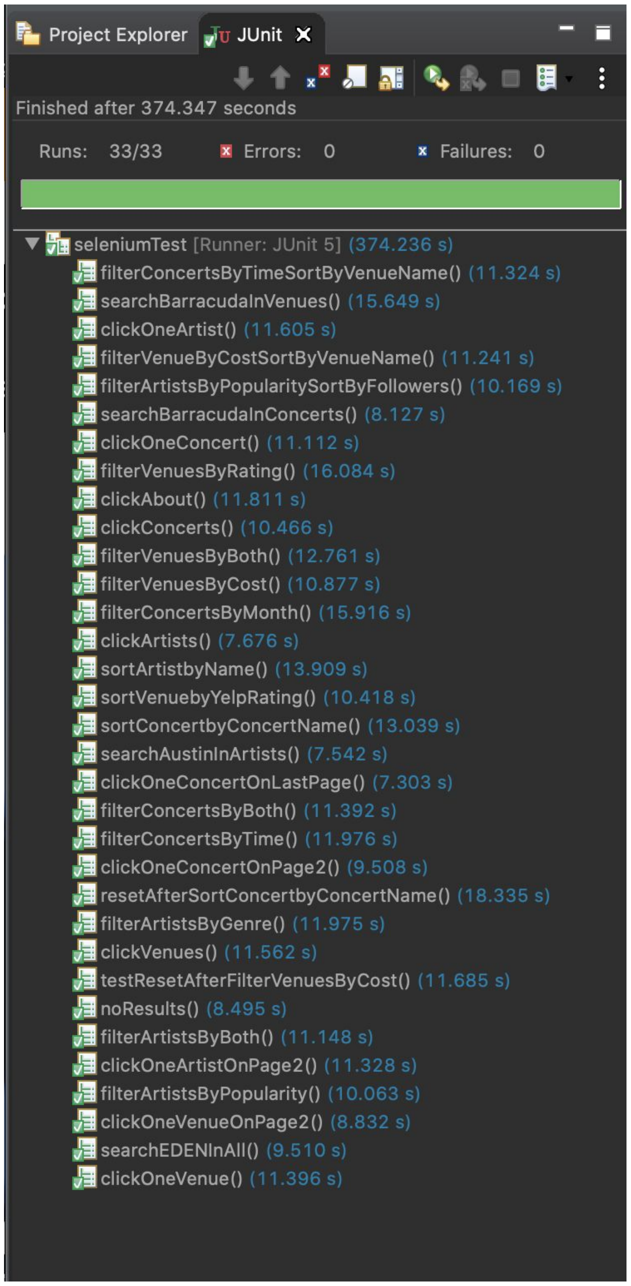
Testing

We utilized Selenium to test the navigation of the website. By traversing through the website and acquiring adequate node coverage, we can ensure that the website would remain functional through most if not all regular utilization of the website. This includes instances of searching for all models and searching, sorting, and filtering for each of our Concert, Venue, and Artist models.

We also used Django Unit Tests to cover our Python code that handles the backend. This involves requesting data from APIs, creating instances for each model, and formatting and preparing the data to store in our remote database. In addition, we were able to create a client during testing to check the functionality of our web pages, redirections, and http responses from a backend perspective. More tests were added and refined during each phase. Both our model code and views code have test coverage. By testing our backend Python code and the data process for all three models, we can ensure that our data pipeline is working correctly.

Additionally, we set up the Mocha testing framework with Chai assertions to test our Javascript code. However, at this time we do not have any Javascript code to test. We might have more Javascript in the future when adding more features. So at that time, our Mocha framework will be ready to test the Javascript code.

SELENIUM TESTING VERIFICATION:

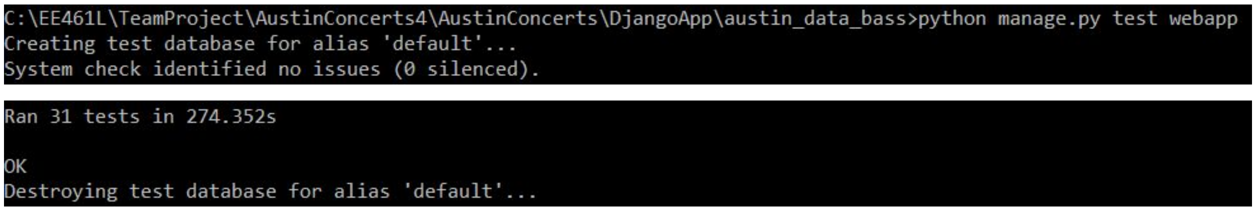


```
Finished after 374.347 seconds

Runs:   33/33      Errors:   0        Failures:   0

seleniumTest [Runner: JUnit 5] (374.236 s)
    filterConcertsByTimeSortByVenueName() (11.324 s)
    searchBarracudaInVenues() (15.649 s)
    clickOneArtist() (11.605 s)
    filterVenueByCostSortByVenueName() (11.241 s)
    filterArtistsByPopularitySortByFollowers() (10.169 s)
    searchBarracudaInConcerts() (8.127 s)
    clickOneConcert() (11.112 s)
    filterVenuesByRating() (16.084 s)
    clickAbout() (11.811 s)
    clickConcerts() (10.466 s)
    filterVenuesByBoth() (12.761 s)
    filterVenuesByCost() (10.877 s)
    filterConcertsByMonth() (15.916 s)
    clickArtists() (7.676 s)
    sortArtistbyName() (13.909 s)
    sortVenuebyYelpRating() (10.418 s)
    sortConcertbyConcertName() (13.039 s)
    searchAustinInArtists() (7.542 s)
    clickOneConcertOnLastPage() (7.303 s)
    filterConcertsByBoth() (11.392 s)
    filterConcertsByTime() (11.976 s)
    clickOneConcertOnPage2() (9.508 s)
    resetAfterSortConcertbyConcertName() (18.335 s)
    filterArtistsByGenre() (11.975 s)
    clickVenues() (11.562 s)
    testResetAfterFilterVenuesByCost() (11.685 s)
    noResults() (8.495 s)
    filterArtistsByBoth() (11.148 s)
    clickOneArtistOnPage2() (11.328 s)
    filterArtistsByPopularity() (10.063 s)
    clickOneVenueOnPage2() (8.832 s)
    searchEDENInAll() (9.510 s)
    clickOneVenue() (11.396 s)
```

DJANGO FRAMEWORK PYTHON TESTING:



```
C:\EE461L\TeamProject\AustinConcerts4\AustinConcerts\DjangoApp\austin_data_bass>python manage.py test webapp
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
```

```
Ran 31 tests in 274.352s

OK
Destroying test database for alias 'default'...
```

Models

Our three models are Artists, Venues and Concerts. These three models provide all of the information a user would need when looking for an Austin Concert. We used three main APIs that gave us the data for these models, Spotify, Yelp, and Songkick. The Artist model details information about the Artist including their bio, picture, Spotify stats and upcoming shows in Austin, with information pulled from Spotify. The Venues model provides our users with info about the Venue including the location, upcoming shows at that Venue, Yelp ratings and various information about the Venue, all of which is taken from the Yelp API. This model also includes coordinates that allow us to display a dynamic and interactive Google Map on each Venue page. The Concert model pulls together all the information about the show itself. It displays show time and date, city, concert name, artists performing, and venue. This information is pulled from the Songkick API. All of these models include pictures and links to the relevant instances of the other models.

Tools, Software, and Frameworks

In this phase we continued to utilize the Django Framework to easily use Python, HTML, CSS, Bootstrap and Javascript and facilitate database accesses and querying. We used Python to pull information from APIs and store them in a remote PostgreSQL Database hosted on Google App Engine. HTML then provided the basic template for each Instance as well as the navigation pages. Django allowed us to use Python inside of the HTML to access data from our database. We used Bootstrap for the design and overall appearance of our website, and CSS for more specific design aspects of our pages to increase consistency across the platform.

To test our project, we used Selenium to test our GUI, Django Unit Tests to test our Python code and API requests, and Mocha to test our future Javascript.

Reflection

Phase I

Our team did a really great job working together for this phase. We started on the assignment early and immediately considered ways to divvy up the work-- we decided to divide the work based on the model. We assigned two people to each of our three models. Starting early and dividing up the work allowed us to focus on our individual parts of the project and efficiently complete it. We had several meetings throughout the two week time period to update each other on progress (by way of Stand Ups) and any problems we were running into. We also stayed updated through regular use of GroupMe. With that being said, many of us are unfamiliar with the use of APIs and MongoDB to access and utilize database information about each model. This will be fundamental to our success in the future, and we will need to learn these tools on our own to be effective in phases 2, 3, and 4.

Our team learned a great deal during this phase. We all learned how to use Bootstrap as a CSS framework to stylize our website and make it reactive to window size. This really helped us make our webpage look professional and elegant. We also learned how to use JavaScript to import API data and for various other tasks like creating the Venues model search bar. The phase also gave us a chance to improve our understanding of HTML and CSS.

Phase II

Our team worked incredibly well together for Phase II. We started the assignment a week before it was due and met daily for roughly hour-long Zoom meetings. We divided up the work for this section into frontend and backend, while also maintaining our same divisions based on model. This allowed us to work most effectively based on our understanding of the Model and the API. This also provided a well-rounded experience for each of us to understand the functionality of the Python Django server, PostgreSQL database, and using a Rest API.

Our team had consistent communication and responsibilities were successfully completed by each team member in a timely fashion. We used Slack to communicate, bounce ideas off of each other, as well as debug software issues. Our daily Zoom meetings were incredibly effective and kept us on track and focused on completing the tasks required for this phase. In addition, we set up additional Zoom meetings with each other as needed to resolve problems that came up. We also screen shared over Zoom to help debug each others' code using pair programming.

Though we worked as well as we could given the current circumstances, we must admit that it was quite difficult to navigate this group project. The requirements were steep and we felt that the timing of the due date for this phase was suboptimal at best. Because of our current situation, we were also very limited in our abilities to pair program, which likely caused an increase in time spent on this phase. Acclimating to conducting meetings online took a little bit. Everyone was also very thrown off by the new changes brought by the coronavirus situation and it made the Phase feel a lot more time crunched.

Phase III

Our team worked pretty well together for Phase III. We had a lighter workload for Phase III since we put in a lot of work into Phase II and got much of the features completed during that phase. We started about two weeks before the due date which gave us plenty of time to work on parts individually and hold meetings to discuss our design. This time we divided up the work based on Python testing, Selenium testing, filtering, sorting and searching. This worked well since these were the main features we needed to implement and they were all a similar time commitment. We learned how to pass in parameters in a URL and appropriately filter and sort our dataset on the backend. We got more hands on experience with creating a test suite for our website.

We continued our strategy of regular communication through Slack and Zoom meetings. We didn't have quite as many meetings for this phase, mostly because there were less features that we planned to implement. We met every 2 to 3 days. Our communication was really strong for this phase, just as it was in the last phase.

Overall, this phase went very smoothly and we never felt very time pressured. This could mostly be to us having done a lot of extra work on Phase II. Being more adjusted to the school-from-home lifestyle and zoom meetings also helped. There weren't very many spots that we feel like we could have improved. We are really happy with the way our website is looking as of this phase.