Explore And Give More

Technical Report: Phase 2

Katherine Eisen, Rik Ghosh, Mariana Medina, Daimu Iwata, Jarrod Brown

Motivation and General Overview

Our motivation behind designing Explore And Give More was to provide resources and information regarding cities, the charities within them, and the attractions the city has to offer, to prospective travelers and tourists hoping to learn more about a certain city before they visit. We plan to develop a dynamic website that contains detailed information about several cities within the United States and connect them with the attractions and tourist spots, and the charities that support the citizens within those cities. Our "*Humanitarian City Guide*" emphasizes local charities to foster benevolence and altruism, in the hopes of supporting the citizens.

In the current phase of the website, we implemented the backend API and added our data to the database, which our API uses to serve requests via endpoints with JSON format. There was some additional difficulty when it came to using the third-party APIs, as some of the data was not as rich as we were hoping it to be. Once again, our concept-to-reality propagation rate was handicapped by the limited nature of some of the APIs. The aim is to swap out some of the stale data with richer data during Phase III in order to improve the overall quality of the service the API and the website is providing.

App Details

This website displays information about three different disparate models: **Cities**, **Attractions**, and **Charities**. Each model has its dedicated route and page, which displays the different instances our website supports. These instances are tiled in a grid or a tabular fashion and showcase important attributes. These attributes, in future stages of the website, can be used to filter or sort the instances, to narrow down the specifications as per the website user's demand. Each tile links to its instance page, which has its route, accessed by the unique id of each instance within a model. Within each instance page, the website displays rich information, including visuals, that provide more detail regarding that specific instance.

As of the current phase, these pages are being rendered with dynamic data, by consuming JSON from our backend API. The API exposes several endpoints (2 per model currently) to request data in a list format (contains minimum information) or in a verbose fashion (detailed information is accessed via the instance ID). In subsequent phases, these endpoints will come equipped with filtering/sorting/searching capabilities.

How it works

Our website is designed to be easy to navigate. The user can select a model card that is identifiable through an ID to navigate to its instance page. The ID is unique to that instance within the model. Within the instance page, the user is rewarded with rich data

in multiple formats: text, images, maps, links, and website embeds. Each instance page has connections to the instance pages from other models that are related to it, making the models interconnected. This allows users to easily swap from one model to another.

Customer Stories

Here are some things customers would like to see implemented in the future!

Phase 1

- **Information about volunteer and service opportunities in cities:**

  Customer:

  "*Hi developer, in addition to donating, I would like to give back to the cities I am in through volunteering and service. There is only so much I can donate, so having opportunities to help out through volunteering would be great. I would like an option to search for charity organizations based on volunteering opportunities.*"

  Resolution:

  I think this is a great idea, as it provides citizens the opportunity to actively get involved with the organizations. As of right now, all the charities listed on our website seem to prefer donations, but this is something we are happy to look into. In future phases, we are hoping to include filters that allow a visitor to easily choose between charities that require volunteering and charities that prefer monetary donations. We will take into account your suggestions when designing this specific feature in one of the future phases.

- **Information about attraction ratings:**

  Customer:

  "I like the feature where I can sort by the type of landmark/attraction… I would prefer being able to see average user ratings of the attractions but am not too particular on where they are from"

  Resolution:

  I believe we have the infrastructure in place to show the star ratings for attractions. The current API does not seem to provide any data regarding ratings for users, but we are happy to look into alternative sources to potentially satisfy your request. In future phases, we are hoping to include filters that would enable you to filter via City, (and hopefully ratings). We will take into account your suggestions when designing this specific feature in one of the future phases.

- **Information about finding charity org by names:**

  Customer:

  “I would like to be able to find locations of specific charity organizations… I would like to search for the cities that have locations or events for this organization.”

  Resolution:

  I believe our current phase supports this exact behavior, although the specific charity you specified is not on our list for this current phase. All you have to do is find your charity, and visit the instance page. All information, including the city/cities where the charity is active.

- **Information about sorting cities by the number of charities:**

  Customer:

  “I would like to see which cities in the US have the most opportunities… I would like to have the option to sort cities by the number of charity organizations and/or charity events located in that city”

  Resolution:

  I think this is an interesting idea, as it provides a pseudo metric to gauge how charitable a certain city is. As of right now, this is something our website doesn't support. In future phases, we are hoping to include sorting features that allow a visitor to easily arrange cities (ascending or descending) based on a few attributes, hopefully, including the one you have suggested. We will take into account your suggestions when designing this specific feature in one of the future phases.

- **Minimum walk score filter:**

  Customer:

  “Developer, I am a user who is mainly concerned with the walkability of cities. In addition to sorting cities by walk score, I would like to have an option to filter cities by a minimum walk score. For example, I may want to just see cities with a walk score of 70 or greater.”

  Resolution:

  I think this is a great idea, as it provides a good metric to gauge the walkability of a certain city, and effectively filter out the cities that don't meet the specified threshold. As of right now, the walkability scores are only displayed on the

instance pages, but this is something we think should be doable in future phases. When filtering is introduced, we will include several attributes you can use in your filtration and sorting, hopefully, the one you specified here as well. We will take into account your suggestions when designing this specific feature in one of the future phases.

Phase 2

- **API call with ID vs. Model**

  Customer:

  “Hello developer, it would be great if when we use an ID when calling the API we get more data than if we just call for a list of data from each model.”

  Resolution:

  We have already implemented this functionality into our API. When you call for each model instance by ID, you will receive back information about some of the other models it's connected to and more information about the individual instance than if you ask for just a list.

- **Pagination of the API**

  Customer:

  “Hello developer, I think it would be useful to paginate the API with the URL. That way each call to the API will have a more manageable number of responses and as a user, we can access the data uniquely by page number rather than be given all of the data.”

  Resolution:

  We have implemented this functionality into our API. You can specify in your query which page number you want to pull data from with the page parameter and the number of instances per page with the per_page parameter. So, you will be able to access the data by page number and limit the number of responses to make it more manageable.

- **Score/Rating System on Cards**

  Customer:

  “It seems that every score on each model has a different scoring system. On the Cities model page, the budget score is out of 8 stars, while the walk score is out of 5 stars. On the attractions model, the popularity is out of 3 stars, and on the Charities model it is out of 4 stars. It is a little bit confusing when 3 stars are great

for some scores but terrible for others. I think scaling the data to go from 1 to 5 stars would be easier for any user to just glance at each card and get a good understanding of the score."

Resolution:

We have scaled all of these different ratings to go from 1 to 5 stars, so it will be less confusing for you and other customers when trying to interpret them.

- **Ranking system on instance pages**

  Customer:

  "As a user, it would be helpful when on an instance page, we are able to see the other related instances (i.e. on a cities page have all of the charities in that city show up ranked in the order of their ratings)."

  Resolution:

  We have implemented this functionality on the model instance pages. On each instance page, you will receive a sampling of the top N instances of each model it's connected to. These instances will be ordered by their overall rating.

- **Showing Distance between Attractions and Charities**

  Customer:

  "Hello Developer, I think it would be really useful to see where in relation to the attraction charities are. In other words, I would love to see either on a map how far apart each charity is from the attraction or by just being given the distance (i.e. Coalition to Abolish Slavery and Trafficking is 2.5 miles away from the Hollywood Walk of Fame)."

  Resolution:

  We won't have time to implement this for this phase of the project, but we will implement this functionality in the next phase. We already pull the longitude and latitude for both the attractions and cities, so this will take us a few hours to complete.

Our Customer Stories for our Developer Team

Here are some things we would like to see implemented in the future!

Phase 1

- **Embed Live Stream**

"*I love looking at the night sky but I don't have a telescope! I would love to be able to easily view different parts of space from a live stream, so I don't have to purchase a telescope myself. Adding some videos or live streams of different moons or asteroids that you're displaying would really help out an amateur stargazer like me. I particularly like watching the videos linked here: https://hubblesite.org/contents/media/videos/2022/047/01GE3AM1KCFF93ZWZ5BKV7TWES !*"

- **Looking deeper into Space**

  "Hello,

  I am an avid space explorer and spend nights looking out into the night sky. This really cool website will help me learn more about the different planets, satellites, and asteroids. I did notice that the planets this website is choosing to showcase are primarily the solar system planets. I would love to learn a little bit more about exo-planets, the ones far away from our solar system. It would be cool if this website could house information about those planets, and maybe even their satellites.

  Additionally, it would be really helpful if (in the future), your website can help me filter out exoplanets and the solar system planets, as it would make finding them on your website super convenient!"

- **Eclipse**

  "Hello, I like the project design and it sounds so interesting to be able to see the information about things in space. I often enjoy seeing solar or lunar eclipses, so I would like to be able to see the information about eclipses too! It would be nice to be able to know when we can see the next eclipse and where we can see it (maybe from only cities or areas in the US). So, the page can have some counter that shows how many days away from the next eclipse. It would be also nice if the page can give us the live streaming of eclipses so that we can see it without actually visiting cities far from home."

- **Adding filter by Composition**

  "Hello,

  I personally am only interested in planets with similar compositions to Earth. I would like to be able to exclude planets that are not the same. So, I would like you to add the functionality to filter the planets by composition."

- **Adding discovery label/attributes**

“One thing that amazes me about space is how massive it is and how different they are compared to Earth. I also find it very interesting that new discoveries about planets and our solar systems are being made constantly as technology advances. I think it would be very cool to see when planets, asteroids and/or moons were discovered.”

Phase 2

- **Sort Planets by number of moons**

  “Hello,

  I am very interested in the planets and their different moons. I would like to order the planets by the number of moons. I also want to compare the planets by their number of moons. So, can you add the functionality to sort the planets’ instances by the number of moons they have?

  I think it should take about three days to implement this functionality.”

- **Filter asteroids by location**

  “Hello,

  I am very interested in the asteroids that are in the main asteroid belt. I only wanted to see asteroids from that area, so I can specifically focus on learning more about those asteroids. So, I think it would be helpful if you could add the functionality to filter out asteroids that are not part of the main belt.

  I think this should take about two days to implement this functionality.”

- **Orbital Period on Instance Pages**

  “Hello,

  I was looking at the planet instance pages and I loved the Neptune page. I thought it was interesting that Neptune had a 165 day orbital period. I wanted to find that information on the other instance pages, but it was not there. So, I would like it if you all could add the orbital period of the planets to the instance pages.

  I think it should take about 1 day to complete this task.”

- **Add links to Moon Models on Planet instance pages**

  “Hello,

I was looking at the instance pages for the planets and started wondering about those planets' moons. I think it would be helpful if there was an easy way to navigate to a planet's moon's instance page from the planet's instance page. I would appreciate it if you all added links to a planet's moons' instance pages from the planet instance page.

I think this should take about 1 day to implement."

- **Add links to Planet on Moon Instance pages**

  "Hello,

  I was looking at the moon instance pages and I wanted to learn more about the planet that these moons orbit. There was no way to find the planet instance page that these moons were connected to without looking for the name of the planet on the planet grid page. I think it would be more convenient if the moon instance pages contained a link to the instance page of the planet that they orbit.

  I think this should take about 1 day to implement."

RESTful API

- We are using Postman to design and document our API, which can be linked to our documentation: here
  - (https://documenter.getpostman.com/view/14067869/2s83Ycg2LW).
- We are using 5 APIs for our model pages where we use 2 GET requests per model page; one to get our list of model data and the other to get the models by IDs.
  - RoadGoat API is used to retrieve city-data.
    - This API was scraped using a Python Script (primarily relying on the `requests` module). The script required the API key and the associated city name. The names were stored in a text file, which was then read from and fed into a loop. The generated JSON was stored in `cities.json` file, which can be found in `backend/json_source/` directory.
  - Charity Navigator API is used to retrieve charity data.
    - This API was scraped using a Python Script (primarily relying on the `requests` module). The script required the API key and the associated city latitude and longitude information. The coordinate information was stored in a text file, which was then read from and fed into a loop. The generated JSON was stored in `charities.json` file, which can also be found in the `backend/json_source/` directory.
  - OpenTripMap API is used to retrieve attraction data.

        - This API was also scraped using a pair of Python Scripts (primarily relying on the `requests` module). The first script was used to get a list of attractions (which contained limited data about the attractions themselves). Using the unique IDs in that list, the second script was get the detailed information for each attraction. The first script required the coordinate information regarding the city, and a radius of search. The second script used the unique IDs as a query parameter. The generated JSON was combined with data from a companion API (see below) and stored in `attractions.json` file, which can be found in `backend/json_source/` directory.
    - MediaWiki API is used to retrieve the description of each city.
        - This API is used to fill in the descriptions for some of the model instances that were missing any form of rich information. The API was also queried using a Python script, which took the wikipedia page name as input. This API was applicable only when the model instance in question had a dedicated Wikipedia page to pull the data from. The data from this API is combined with the general data collected for the model instance.
    - Google Places API is used to retrieve additional attraction data to supplement OpenTripMap.
        - This API was also scraped using a pair of Python Scripts (primarily relying on the `requests` module). The first script was used to get a list of attractions (which contained limited data about the attractions themselves). Using the unique IDs in that list, the second script was get the detailed information for each attraction. The first script required the coordinate information regarding the city, and a radius of search. The second script used the unique IDs as a query parameter. The generated JSON was is combined with data from a companion API (see above) and stored in `attractions.json` file, which can be found in `backend/json_source/` directory.
- Our API provides the following 6 endpoints:
    - GET list of cities - Returns a JSON object of cities based on the query criterion.
    - GET city by id - Returns a JSON object of a single city indexed by the unique ID, based on the query criteria.
    - GET list of attractions - Returns a JSON object of attractions based on query criterion.
    - GET attraction by id - Returns a JSON object of a single attraction indexed by the unique ID, based on the query criterion.
    - GET list of charities - Returns a JSON object of charities based on the query criterion.
    - GET charity by id - Returns a JSON object of a single charity indexed by the unique ID, based on the query criterion.

Models

Our models are connected primarily by location. Our website is a city guide that highlights not only information about a city, but also the charities and attractions present in that city. We want to make it easy for visitors to make the most of their time in a city, so that they can contribute to charities near local attractions, or explore attractions that are near the charity that they are volunteering at. Thus, our city models contain references to the charities and attractions that are present within them (1 to many). Our charity models contain a reference to the city (1 to 1) that they are in as well as nearby attractions (1 to many). And our attraction models have a reference to both their city (1 to 1) and charities (1 to many) in the surrounding area.

Cities (within the USA)

- **Sortable/Filterable attributes:**
  - State
  - Population Size
  - Budget Score
  - Walk Score
  - “Known-For” tags
- **Searchable attributes:**
  - Name
  - State
  - Bike score
  - Time-zone
  - Cost of living
- **Media:**
  - City image
  - City description
  - Embedded map of the city

Attractions (within the USA)

- **Sortable/Filterable attributes:**
  - City
  - State
  - Popularity
  - Cultural Heritage recognized
  - “Attribute” tags
- **Searchable attributes:**
  - Year of Establishment
  - Nearby Charities
  - Religious Affiliations (if any)

    - Hours of Operation (if applicable)
    - Phone number
- **Media:**
    - Attraction image
    - Description of Attraction
    - Embedded website of the attraction
    - Embedded map

Charitable Organizations (within the USA)

- **Sortable/Filterable attributes:**
    - Cause Area
    - Star Rating
    - City
    - State
    - Donation deductibility
- **Searchable attributes:**
    - IRS Subsection
    - IRS Organization Classification
    - Financial Rating
    - Accountability rating
    - charity EIN
- **Media**:
    - Image of charity logo
    - charity mission statement
    - embedded website of the charity

Tools

- **GitLab:** An online code storage platform with heavily used capabilities of issue tracking and CI/CD.
- **React Typescript:** A library that we used to create reusable components for an interactive UI in our web application.
- **Material UI:** A free library with UI tools and components. We used it in nearly every aspect of designing our frontend to make it visually pleasing and interactive.
- **AWS Amplify:** Used to host our frontend application.
- **Namecheap:** Used to register a free domain name.
- **Docker:** Used to ensure our libraries and dependencies were consistent throughout our environments.
- **Postman:** Used to test and document our APIs using a graphical user interface.
- **Black:** A code formatter that we used to format our backend files.

- **Flask:** A Python micro web framework used in our backend to gather data from our APIs
- **SQLAlchemy:** A Python SQL toolkit and object relational mapper used to interact with our database.
- **Flask SQLAlchemy:** A Flask extension with SQLAlchemy support that we used to define our database tables and models.
- **PostgreSQL:** An object-relational database system used to store and query our data into the database for our web application.
- **Marshmallow:** A Python object relational mapping serializer and deserializer that we used to convert complex data types to and from Python data types.
- **Flask Marshmallow:** A Flask extension with Marshmallow support that we used to work with Marshmallow and to add additional features to Marshmallow.
- **Google Cloud Run:** A managed computing platform used to host our backend.
- **Amazon RDS:** A relational database hosting service used to create and connect our database instance.
- **Jest:** A JavaScript framework used for Frontend tests.
- **Selenium:** A testing framework used for Frontend acceptance tests.

APIs

- Road Goat
  - https://www.roadgoat.com/business/cities-api
- Charity Navigator
  - https://www.charitynavigator.org/index.cfm?bay=content.view&cpid=1397
- OpenTripMap API
  - https://opentripmap.io/product
- MediaWiki API
  - https://www.mediawiki.org/wiki/API:Main_page
- Google Places API
  - https://developers.google.com/maps/documentation/places/web-service/overview

Hosting

Frontend

We are using AWS Amplify to build and host *Explore & Give More*, and we obtained a free domain name (exploreandgivemore.me) via Namecheap. AWS Amplify is connected directly to our GitLab repository so that it redeploys any time changes are made to our `main` branch.

Backend

We are using Google Cloud Run and Namecheap to host our backend, and we are using Amazon RDS to store our database. We have to create a new Cloud Run deployment every time we make a change to our backend to keep API requests to our site up to date. We added a subdomain (api.exploreandgivemore.me) to our previously acquired domain name through Namecheap to point to our hosted backend.

Phase 2 Features

Database

We're using Amazon RDS to host our database, which we're managing with PostgreSQL and pgAdmin. Our RDS instance is under the free tier and it's set to allocate as little storage as possible to avoid any fees. We modified the database's permissions settings so that we could log into it locally, through pgAdmin. To populate the database, we used Flask and SQLAlchemy, which we connected to the database through the database's URI. We then created and filled models using JSON files that we created from the APIs that we used, and we add and commit those model instances in `populate_db.py`, which needs to be rerun anytime we make changes to our database.

Pagination

We've implemented pagination by creating a React component that makes use of `react-router-dom`'s `useSearchParams`. Our pagination component features buttons and drop-down lists for selecting page and page size. These two fields each have an event listener that updates `useSearchParams` based on the selection. We use the React Hook `useEffect()` to run whenever `searchParams` gets updated, where we use `searchParams` to modify the search parameters in the API queries that get sent to the backend to obtain model instances fitting the specified page size and page number for the model grid and row pages.