Explore And Give More

Technical Report: Phase 1

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 constructed a static skeleton for the general structure of the website. Our concept-to-reality propagation rate was handicapped by the limited nature of some of the APIs, so we weren't able to parse through enough responses to be able to display all the attributes specified in detail for the **Models** section. The challenges encountered while working on this website can be found in more detail in the **Challenges** section of this report.

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 currently supports. These instances are tiled in a grid 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 static data, but in future phases, the website will expose several endpoints from its API that will be used to dynamically render each of these pages.

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.

User Stories

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

- **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.

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 4 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.
    - Charity Navigator API is used to retrieve charity data.
    - OpenTripMap API is used to retrieve attraction data.
    - MediaWiki API is used to retrieve the description of each city.

Models

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
- React Typescript
- Material UI
- Amazon Web Services
- AWS Amplify
- Namecheap
- Docker
- Postman

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

Hosting

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.

Challenges we ran into and how we overcame them

There were several challenges along the way, especially concerning the lack of experience or foundational information regarding some of the tools in our toolchain. To detect type errors at “compile time” we decided to work with React Typescript, which did help in the long run, but slowed us down, as we had to get familiar with the type system and the extended features such as interfaces and classes.

There was also an initial learning curve getting used to AWS Amplify, querying data from the APIs, and setting up the GitLab API to produce the required counts for commits and issues. Despite these challenges concerning the toolchain, we are certain that this initial hurdle will help us produce frontend components in a more scalable and dynamic fashion.

One other challenge we ran into was a stack overflow exception that occurred when we initialized our **City** and **Charity** models. We were initially storing Charity objects within City objects and City objects within Charity objects, but the initialization functions for each ended up calling each other, causing unbounded recursion. We corrected this issue by instead storing the IDs for each model within the opposite models. Then, when rendering the Charity page, for example, we performed a lookup of that Charity’s City ID so that we could render a City Model Card on the Charity instance page.