CS373 Technical Report for RelievePoverty.me

Table of Contents

Authors:

Albert Luu – Full-Stack Developer

Colin Frick – Full-Stack Developer

Daniel Chruscielski – Full-Stack Developer

Evan Weiss – APIs, Backend, and Databases

Uri Kugelmass – Frontend Developer

Yoav Ilan – Frontend Developer

Motivation

The motivation behind our site was the fact that poverty is becoming a bigger issue within the United States. The cost of living in many cities is exponentially increasing while minimum wage requirements remain almost constant. There are typically negative stereotypes concerning those who are living under the wage gap, and many are reluctant to help them.

Our hope is that by bringing awareness to the issues concerning poverty and recommending related charities that we can encourage others to help contribute to this cause.

User Stories

1. Phase 1: Get charity links

Customer Description: As a user, I would like to be able to get links to the websites of charities. This would facilitate donating and allow me to read more about the charities I care about. The link received from the API should be fully clickable/able to be pasted into a web browser.

Estimation: We initially estimated that it would take about 5 minutes to use Charitynavigator's API to get data on 3 instances.

Refinement: After enlarging the scope to account for the time to display the data on our site and add the functionality to our Postman API, we changed our estimation to 45 minutes.

Actual Time: about 1 hour.

2. Phase 1: Get state poverty rankings

Customer Description: As a user, I want to be able to get a ranked list of all 50 states from the highest percentage of the population in poverty to the lowest. This could easily be done by allowing users to get the rank of a

given state. This information is useful in determining which areas need the most help.

Estimation: After assuming that we only obtained the ranks for the 3 instances we were using, we decided that it would take about 5 minutes to scrape the information.

Refinement: After enlarging the scope to account for the time to display the data in sorted order and add the functionality to our Postman API, we changed our estimation to 30 minutes.

Actual Time: about 30 minutes.

3. Phase 1: Get state poverty percentage

Customer Description: As a user, I would like to be able to get the poverty percentage of my state. Given a state, the API should return the percentage of its population that live in poverty. This information is useful in determining which areas need the most help.

Estimation: After assuming that we only obtained the percentages for the 3 instances we were using, we decided that it would take about 5 minutes to scrape the information.

Refinement: After enlarging the scope to account for displaying the data on our website and adding the functionality to our Postman API, we changed our estimation to 30 minutes.

Actual Time: about 15 minutes.

4. Phase 1: Get state poverty line

Customer Description: As a user, I would like to be able to query the average poverty line for each state via the API. Given a state, the API should return the quantitative poverty line (in USD) in that state. This is valuable information that can be used to inform the salary decisions made by companies.

Estimation: After assuming that we would only be obtaining the averages for the 3 instances we were using, we decided that it would take about 5 minutes to scrape the information.

Refinement: After enlarging the scope to account for displaying the data on our website and adding the functionality to our Postman API, we changed our estimation to 30 minutes.

Actual Time: about 15 minutes.

5. Phase 1: Get news

Customer Description: As a user, I would like to be able to retrieve news on poverty that the site has to offer. It would be convenient to give the API a range of dates and get all the articles between them. This could be linked to the articles, or the articles themselves.

Estimation: We estimated that it would take at most 10 minutes as the only requirement was to add a parameter to a single API call on Postman.

Refinement: Instead of adding just 1 example to filter by date, we decided to also have an example to filter by both date and state and changed the estimation to 15 minutes.

Actual Time: about 10 minutes.

6. Phase 2: Icons

Customer Description: As a user, it would add to the already professional looking appearance of the site if you used an icon and unique title name in the browser tab. Right now. it's the default "React Application" and react icon. We'd like to see those changed. If you can manage it, an icon on the site itself (next to the main "Relieve Poverty" in the navbar) would be even better.

Estimation: We estimated that it would take at most 5 minutes as the only requirements were to add an image and add a document title.

Refinement: We had no refinement.

Actual Time: about 5 minutes.

7. Phase 2: Make Splash Page Images Clickable

Customer Description: As a user, I would like to be able to get to each of the models from the splash page. The map on the front page leads to states, but it would be convenient to be able to get to Charities and News from their respective pictures as well.

Estimation: We estimated that it would take at most 10 minutes as the only requirement was to link images to other parts of our website

Refinement: We had no refinement.

Actual Time: about 5 minutes.

8. Phase 2: Customize heading of model grid pages

Customer Description: As a user, it's nice to see the consistency that exists across the design of the various model grid pages. However, I think that the grey area that currently says "Learn more about poverty in the U.S." could be put to good use if it was unique for each model grid page. The titles could be more indicative of the current information displayed, or there could be a picture.

Estimation: We estimated that it would take 5 minutes because the only requirement was to change the wording of the Jumbotron.

Refinement: We had no refinement.

Actual Time: about 10 minutes.

9. Phase 2: Colors

Customer Description: We like the look of your UI, but we think it would benefit from a unified color scheme. Right now, the dominant colors on the site are blue, white, grey, and orange. Some pages look little lifeless when there's a lot of empty space of gray. We'll leave the specifics up to you guys' preferences, but we'd like to see more color.

Estimation: We estimated that it would take 10 minutes to change the color scheme as many of the category attributes are connected.

Refinement: After changing the variety of colors to include different shades, we changed the estimation to 15 minutes.

Actual Time: about 15 minutes.

10. Phase 2: Make all images consistent sizes

Customer Description: The grid (model) pages look great, but we would like for the images in the cards to be the consistently sized. It's a little distracting that the images are different ratios. It would be cool if they were standardized.

Estimation: We estimated that it would take at most 5 minutes because changing the image sizes just requires a universal CSS edit.

Refinement: Instead of making a universal change, we decided to individually change images per category to be more comprehensive and flexible and changed the estimation to 15 minutes.

Actual Time: about 20 minutes.

11. Phase 3: Sort States by Median Income

Customer Description: Median income is an interesting statistic that's already presented on the website. As a user, I'd like to be able to sort the

states by this value. That way I can compare them in another way besides their level of poverty.

Estimation: We estimated that it would take 15 minutes to complete as we only needed to make a separate API call to get the sorted data.

Refinement: We decided to generalize the function to call the API to allow for more flexible handling of different API calls. We also needed to build selection boxes for the user to interact with.

Actual Time: about 1 hour.

12. Phase 3: Filter Charities by State

Customer Description: As a user, I'd be interested in viewing all the charities in a group of states that I'm interested in. To that end, I'd like to be able to filter the list of charities by several states. This way I can view all the charities in a large area that's relevant to me.

Estimation: We estimated that it would take 5 minutes to complete as we only needed to add a filter selection box and tweak our existing filters.

Refinement: A distinct selection is needed for all fifty states, which proved more demanding than originally anticipated.

Actual Time: about 20 minutes.

13. Phase 3: Filter Articles by State

Customer Description: As a user, I'd like to be able to filter news articles by state. Perhaps with a drop down menu or something that lets me pick a state or group of states that I'd like to see the news about on the News model page. This way I can see the news most relevant to my location on the article pages.
Estimation: We estimated that it would take 5 minutes to complete as we only needed to add a filter selection box and use our existing states filter
Refinement: Formatting the filter boxes required extra time as each model page has different possible filters.
Actual Time: about 10 minutes.

14. Phase 3: Search and Sort by Article Title

Customer Description: As a user, I'd like to be able to quickly find an article that I'm looking for. To that end, I think it would be useful for the search to include article names, and to sort news results by the alphabetical article titles. This way I can browse articles quickly.
Estimation: We estimated that it would take 10 minutes to add a sort selection field; searches already return cards with the article title
Refinement: We allowed for sorting ascending and descending by title and by other properties.
Actual Time: about 25 minutes.

15. Phase 3: Filter Articles by Date

Customer Description: As a user, I'd like to be able to filter articles based on the date they were published. Perhaps there would be several options for filtering, such as "Past Week", "Past Month", and "Past Year" that will allow me to exclude articles that are older than the given time frame. This will allow me to filter out articles I deem to be too old to be relevant to my search.

Estimation: We estimated that it would take 5 minutes to add another filter based on date.

Refinement: Some styling of the filter box on the page was necessary.

Actual Time: about 10 minutes.

RESTful API

Background:

Our API will be hosted at https://api.relievepoverty.me and will be both designed and tested on Postman. Our current design and documentation are available here. Our approach was to have two API calls for each model; one to get a list of all instances with a small amount of information, then another to get all of the information about a single instance. For example, getting charity links first calls for all the charities and then finds a specific charity. The process is the same for poverty rankings, poverty percentages, poverty line, and getting news. We felt this approach was a good representation of our web-flow where a user wouldn’t need as much

information to select a charity but would probably like more after viewing the instance itself.

Models

News Articles

As the name suggests, each instance of this model is an article scraped through news sources such as the NYTimes. The model page has a list of all articles with a preview of the headline, a small summary, and a photo from the article. After clicking on an article, the user is then sent to an instance of it that has the above information in addition to the names of the news organization, the author, the date published, a link to the actual article, the state the article is about, and a list of related charities. The article instances also have internal links to their respective state and related charities. Our attributes for the news articles include the title, a summary of the article, the source it is from, the state the article refers to, who wrote the article, when it was published, the URL of the article, and an image of the article, and related charities to the article.

Charities

Each instance of this model is a charity scraped through the Charitynavigator API. The model page has a list of all charities with the charity name, mission statement, and photo. After clicking on a charity, the user is sent to an instance of it that has the above information in addition to the charities' affiliation, tax classification, state, mailing address, website link, map, and list of related charities. The charity instances also have internal links to their respective state and related articles. Our attributes for the charities include the charity name, the charity's logo, cause, affiliation, address, state location, rating, tax classification, related articles to the charity, and a Google Map of the charity's location.

States

Each instance of this model is a state within the United States. The model page has a list of states with their names, flags, and poverty rankings. After selecting a state, the user is then sent to an instance of it that has the above information in addition to the number and percentage of citizens below the poverty level across three different age distributions. The instance also has a list of counties with the highest poverty rate, and internal links to related articles and charities. Our attributes for States include the state name, poverty ranking, percentage of citizens below poverty rate, poverty under age 18, median income, poorest county, a Google Map to the poorest county, related articles to the state, and charities located in the state.

Tools

GitLab

For version control and continuous integration our group uses GitLab. We currently have two branches: master and development. Our repository is located at https://gitlab.com/urielkugelmass/relievepoverty.

Postman

Currently we are using Postman to design our API's. Our current documentation is located at https://documenter.getpostman.com/view/5460449/RWgjY1qy.

AWS

We are currently using AWS to host our web app with EC2, RDS, and Cloudfront.

Bootstrap

We're using Bootstrap, an open source CSS toolkit, for most of the designs on our

ReactJS

ReactJS is our front-end JavaScript framework. It focuses on components makes it easier for distributed, collaborative work. It also renders out the site from API data.

Flask

Flask is a web framework used on the site.

Mocha

Mocha is a JavaScript testing framework.

Selenium

Selenium is a framework that allows browser navigation automation.

Unittest

Python framework that allows the creation of unit tests.

PlantUML

A program that allows you to see the relationship between classes and their attributes.

Grammarly

Grammarly is a plugin that checks for grammar mistakes.

Hosting

We're currently hosting our web app on AWS. Within AWS we have two EC2 instances: one for the frontend, and a second for the backend. These instances are linked to static IP's and to load balancers. The load balancers are then distributed through Cloudfront on AWS which provides our domains with SSL Certificates. The only downside is that Cloudfront's cache has to be manually invalidated to display new versions of our buckets.

Pagination

Pagination allows a normal number of instances per page (12). It is important that we do not list everything in one page because it hinders the user experience, requiring the user to inefficiently scroll and search for information. We used react-paginate because it was a straightforward package we could download and customize quickly. Our biggest work was to write a handlePageClick handler method that would take in the page we clicked on and make an API call to get the instances on the given page.

DB

A database was needed to create a dynamic website to display more information than can manually be added. We scraped our API sources and stored the info into a MySQL database. While most of the information was able to be scraped, some select few charities or states were available in the

API sources, and so they were manually added. Each charity and news object contains a state which links it to a state object, based on the location of the given article or charity. The news and charity objects are related to each other through this state object, where a charity and news article are deemed to be linked if they share the same location.

Testing

We test because it is important that we build bug-free Software for our customers, and the only way to find bugs is by having a robust testing framework that makes sure all our units and system works as expected. For front-end unit testing, we used Jest, which allowed us to test the different React components we built along the way easily and check for expected states. For acceptance testing, we used Selenium and Mocha, which allowed us to navigate through our website, like a regular user would do, and check the presence or absence of specific elements on our website. For back-end unit testing we used unittest, including unittest.mock, and nose2. This allowed us to test our scrapers. We also wrote tests for Postman to test our backend and the responses from our APIs.

Filtering

The work of filtering is mostly done in the backend. When a filter is passed in, the call first retrieves all entries. If a filter is detected in the request URL, it applies Python's filter() to the results to only accept the entries matching the given filter. On the frontend, the user is presented a selection box with preformed filters from which they can select. Upon selecting one, a call to a React function handleFilter() for that specific filter reforms the API request URL and reloads the data accordingly.

Searching

Searching is done by entering one or more terms in either the model specific search bar or the global search bar on the navbar. This passes the space-separated terms in an API call to the backend. For model specific queries, a SQL query is performed once for every query term to match each query term individually. The results are merged into a set to ensure no duplicates are listed, and the results are sent back to the frontend. For a global search, a search is done for each of the three model pages. The results are passed back in a search page that includes three different sections with results from each of the three models, each with their own separate pagination.

Sorting

On each page with model data, the user is given a selection between sorting the results ascending or descending, and their selection will be included in the API parameters. When the backend receives a request for data, it checks to see if there is a sort parameter given. If so, the list returned by SQLAlchemy's SQL query is run through Python's sort with the necessary comparison function passed to it. If no sort order is specified, the results are returned unordered.