Group 01 Phase 4 Technical Report

Kevin Xu, Josh Yu, Ben Yu, Bryan Lee, Tyler Kubecka

Overview

In this report, we will be detailing our BrighterBeginnings project to provide a means of navigation through the repository and understanding of the architectures. This involves explaining the structure, testing, and hosting for the front-end and back-end.

Purpose

We deem this website necessary as we believe that financial constraints should never hinder the pursuit of knowledge. BrighterBeginnings aims to empower low-income K-12 students to reach for the stars by connecting them with knowledge about scholarship opportunities and organizations providing aid, whether financial or otherwise.

Toolchains

The tools we used to create our web application are as follows:

- Development
  - Gitlab
  - Docker
  - Visual Studio Code
- Front-end Tools
  - React
  - Bootstrap
  - Node
- Back-end Tools
  - Flask
  - SQLAlchemy
  - Gunicorn
  - MySQL Workbench
- Hosting
  - AWS Amplify - Front-end
  - AWS EC2 - Back-end

    - AWS RDS - MySQL Database
    - Namecheap - Domain Name Registration
- Testing
    - Jest - Front-end
    - Selenium - Front-end
    - Unittest - Back-end
    - Postman - Back-end

Front-end Architecture

The front end is a React app, which you can build and run the app like this:

```
(in cs373-group-01/front-end)
      npm install
      npm run start
```

If running locally, it will be at `localhost:3000` by default.

Structure

The structure of the website currently is as follows:

- A home page
    - Relevant files:
        - src/pages/Home.jsx
        - src/components/cards/ExploreCard.jsx
        - src/components/AwesomeSearch.jsx
- An about page (/about)
    - Pages for each developer on click
    - Relevant files:
        - src/pages/About.jsx
        - src/pages/AboutSubPage.jsx
        - src/data/about.js
        - src/components/cards/StatsCard.jsx
        - src/components/cards/ToolsCard.jsx

- Cities page (/cities)
    - Pages for each city on click
        - Contains images and info about each city
        - Contains a map of the location of each city
        - Links to organizations with help resources in those cities
    - Relevant files:
        - src/pages/Cities.jsx
        - src/pages/CitySubPage.jsx
        - src/data/cities.js
        - src/components/cards/CitiesCard.jsx
- Organizations page (/organizations)
    - Pages for each organization on click
        - Contains images and info for each organization
        - Links to scholarships and cities that these organizations are based in
        - Contains maps of the locations of each of the organizations
    - Relevant files:
        - src/pages/Organizations.jsx
        - src/pages/OrganizationSubPage.jsx
        - src/data/organizations.js
        - src/components/cards/OrganizationsCard.jsx
- Scholarships page (/scholarships)
    - Pages for each scholarship on click
        - Contains images and info for each scholarship
        - Contains state flags of all the relevant states for each scholarship
        - Links to organizations that provide those scholarships if applicable
    - Relevant files:
        - src/pages/Scholarships.jsx
        - src/pages/ScholarshipSubPage.jsx
        - src/data/scholarships.js
        - src/components/cards/ScholarshipsCard.jsx
- Routing is done in src/App.js

Testing

Jest is used to test the rendering of React components as well as the proper text and links are rendered on those components. It allows for us to render the components in isolated environments outside of the context of the whole website.

Selenium is used for the acceptance testing by going through the actual domain. It uses a headless browser (in our case Firefox) to click through all the elements on the page to make sure they route to the correct places. It also verifies the text content of each of the pages upon loading.

Visualizations

Our visualizations were achieved through the use of Recharts, a React library built upon D3 which allows for elegant and powerful visualizations within React projects. This library allowed us to make calls to both our and our developer's API to feed into their components which led to seamless integration between data and visualization. The data chosen to visualize was based on what would make the most sense and allow for three unique visualizations.

We have 3 visualizations for our data:

1. Scatterplot of poverty rate vs unemployment rate for cities
2. Bar graph of average award value for different eligibility classes of scholarships
3. Radar chart of types of scholarships

And 3 visualizations for our provider's data:

1. Pie chart of abortion status in every state
2. Bar graph of states with certain policy exception cases
3. Scatterplot of women of reproductive age vs number of treatment centers in cities where abortion is restricted

Hosting

For our front-end, we are hosting it on AWS Amplify, in which it automatically pulls from the Gitlab repository to update. The domain is https://www.brighterbeginnings.me. To configure the domain, all that is necessary is to navigate to the domain registrar and add the ALIAS and CNAME records provided on Amplify to the advanced DNS settings of the domain.

Back-end Architecture

Structure

Within our back-end folder, we have our Flask application saved as app.py. In app.py, we import the models.py and schema.py modules, which are used to define the models of our data instances and the SQLAlchemyAutoSchema classes for them, respectively. Within models.py, this is where we establish our database connection, define the table names and model attributes, as well as create the tables. Within schema.py, we define the schema classes for our models, which serve to transform complex data types into a format that can easily be serialized into JSON form. The dependencies needed to run the Flask app are found in the requirements.txt file, which our Dockerfille uses to install all the necessary packages. Also within our backend folder, we have two folders labeled data and data collection. The former contains csv files for all of our models with data on every instance in our database, whereas the latter contains the sources and web scrapers used to collect said data from third party APIs. We used pandas to transform data from API calls into our desired format.

API Documentation

Our API is able to get any single or all data instance(s) for all of our models, including all the fields displayed on our website. To learn more about our API, you can read about it on Postman.

Testing

Within our backend folder, we have a tests.py and postman_collection.json file that are used to test our API endpoints we defined in our app.py. To ensure that every request worked as intended by making sure that our get all methods returned the right number of instances and that our get specific instance methods returned the one and only correct instance.

Hosting

For our back-end, we are hosting it on AWS EC2, where we run our Flask app in a tmux session on the EC2 instance terminal. The domain is https://api.brighterbeginnings.me. To actually run the server, we use Gunicorn, a WSGI server, and enter the command ‘gunicorn -w 2 -b 0.0.0.0:8080 ‘app:app’. This specifies two worker processes to handle incoming requests, binds all available network interfaces (‘0.0.0.0’) on port 8080, and runs our app module.

Database Architecture

Structure

For our data connections, we made use of SQLAlchemy ORM (object-relational mapper). Within our MySQL database, we have a Scholarship, Organization, City, and an association table called Scholarship_City_Association. The latter exists to create a many-to-many connection between scholarships and cities, while the organization schema has a scholarship_id and scholarships field to create a connection between organizations and scholarships. To look more into the model schema, attributes, and connections, refer to the model.py and schema.py files.

Models

We have three models for scholarships, organizations, and cities with the following attributes:

- Scholarship - id, name, awarded_by, award_amount, merit_based, need_based, essay_based, nationwide, img_src, link, cities, organizations
- Organization - id, name, email, phone, organization_type, img_src, scholarship_id, scholarship
- City - id, name, population, state, median_income, unemployment_rate, college_educated_rate, poverty_rate, img_src, scholarships

Searching, Sorting, and Filtering

Implemented through the abstracted matching and ordering libraries by SQLAlchemy. Each of these calls is implemented on the backend and callable through the API as queries where the variable can be set in the URL. Further documentation of the exact syntax is on Postman. On the front end, on a model page, when entering into the search text field or selecting sort/filter options, the API URL used in the code to dynamically get the list of components will be changed, so that the backend will now return the list of instances with the search/sort/filters applied.

Hosting

We currently host our database on AWS RDS, which is connected to an AWS EC2 instance. In terms of specs, our database runs on a MySQL Community engine, size db.t3.micro, and on the us-east-2a availability zone with an endpoint on MySQL's default port, 3306. We also have a network load balancer set up for our EC2 instance (target group) in order to route traffic.

User Stories

Our customers did not provide us with user stories for this phase.

Challenges Faced

This phase’s main challenges involved visualizations, which were the primary new feature that was required of us this phase. First, we needed to figure out we wanted to integrate D3 into our project. We initially tried to use plain D3 but integrating it into React proved to be quite difficult. Thus, we chose to use Recharts, as it is among the most popular D3 libraries, especially for React.

Once that was out of the way, our team faced the challenge of choosing which data to model and what types of visualizations this data should be fed into. Finding different sets of data which could be turned into three unique types of visualizations was a little difficult. This was especially the case for provider visualizations, as we felt there was not much visualizable data that was different enough to use distinct visualizations. We were also slightly limited by the types of graphs Recharts offers, as the library currently does not have box plots or choropleths.

Aside from visualizations, the rest of the challenges came with refining the website. There were some tricky algorithms we had to implement, like fixing our “google-like” search. We also had to optimize a few things regarding the search bars and pagination and this took quite a significant amount of effort.

Finally, the project’s deadline, combined with the need to prepare a presentation as well as the growing amount of work for other classes as finals week approaches, was a bit strenuous to work with.