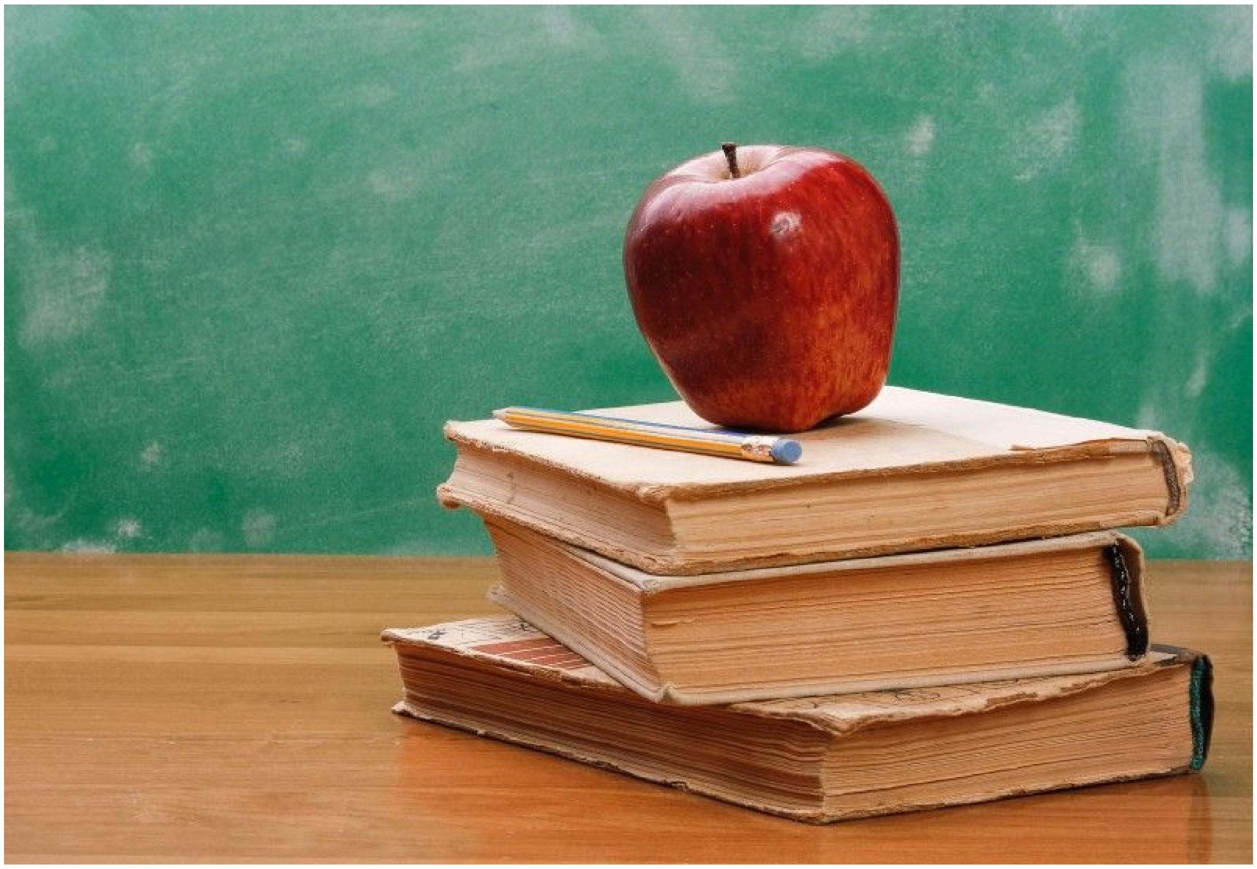
Equal Education Technical Report



CS373: Software Engineering Fall 2018

Authors:

- Kyle Sotengco
  - Front-end + Documentation
- Intae Ryoo
  - Front-end + Documentation
    - Phase 1 Leader
- Andrew Harmon
  - Back-end
- Jesus Palos
  - Front-end + API
    - Phase 2 Leader
- Prateek Kolhar
  - Back-end
- Ailyn Aguirre
  - Front-end

Motivation:

Our website is designed to help people find charitable opportunities to assist underprivileged children with their education. In today's society, a strong, reliant education is essential to living a successful life; yet there are many school districts in America that lack even basic school supplies. We wanted to showcase less fortunate areas in the US and allow users the opportunity to give their time or financial support to these communities. We hope that these contributions allow less privileged children have the same opportunities and benefits as more affluent areas.

User Stories:

Our core interest is to provide an extensive database of school districts in need of assistance. The website aggregates a large number of low-income areas and links them to charitable organizations and communities relevant to that area. Users can then either search for school districts within their area, search for specific communities in the United States to view their economic situation, or donate to charity organizations that they trust. We believe that many people would be willing to donate their resources, but they might not know how or whom to trust. We want to showcase to our users just how many different, reliable ways there are to help out the community in regards to education.

Models:

Our three models are school districts, charitable organizations, and communities.

School Districts

The school district model consists of lower income school districts in America. Each school district will have its name, location, description, population under poverty, charities involved, and communities that encompass the district. Users can sort by poverty level (using data from census.gov), and date of establishment to determine which school is the best fit. There is also an option to filter districts by state so the user can easily find opportunities within their area.

Charities

The charity model consists of several charitable organizations involved with education. Associated with each charity is its location (city, state), deductibility, ruling date, rating, their mission statement, communities supported, and schools associated. The charities can be sorted the name of the company, date of establishment, and rating (as done by CharityNavigator.org.) We want to make sure users feel comfortable in giving their donations to a reputable cause. The user can also directly search for any specific organization they have in mind.

Communities

The communities model consists of a myriad of communities associated with a specific school district. Listed for each community is the location (state, city), counties within community, a description of the community (brief summary, population details), school districts associated, and a map of the community via the Google Maps API. Communities can be sorted by name, location, and population size. We want to give our users a personal look into these communities so that they can assess how they can help out.

RESTful API:

Link: https://documenter.getpostman.com/view/5488016/RWgm41VA

The mockups for the models are built using Postman Mock server. We built 4 endpoints for each model based on i)page ii)id iii) links to the other entities:

We have three models:

1. Charities:

Sl no.	**Name**	**Type**	**Description**
1	id	int	Identifier
2	name	string	Name of the charity
3	tagline	sting	Short tagline of the Charity
4	state	string	State where the charity operates in
5	scope	string	Regional or National
6	image	string	Path to the image file
7	Category	string	Type of the charity
8	Mission	string	Elaborate mission statement
9	website	string	Url of their website

   API:
   1. {{prod_url}}/charities/?id=182 : gets all the details of a charity for the given id.
   2. {{prod_url}}/charities/?page=1 : gets the total number of pages (“num_pages”) and “grid” which contains charities with details upto 7th attribute in the above table.

2. Communities

Sl no.	**Name**	**Type**	**Description**

1	id	int	Identifier
2	community	string	Name of the community
3	image	string	Path to the image file
4	summary	string	Short summary of the community
5	state	string	State where the community is located
6	counties	string	The counties located in community
7	population	string	The population of community
8	map_url	string	Path to the map

API:

1. {{url}}/communities?page=5: get the grid page
2. {{url}}/communities?id=39: get the entity with id
3. {{url}}/school_districts?communities_id=18: get the id and the name of school_districts list
4. {{url}}/charities?communities_id=5: get the id and the name of charities list

3. School Districts

Sl no.	**Name**	**Type**	**Description**
1	id	int	Identifier
2	name	string	Name of the school
3	poverty	int	Percentage of population under poverty line
4	state	string	State where the school is located
5	community	String	Community school is based in
6	image	string	Path to the image file
7	district_code	int	code/id associated with each district (according to census.gov)

API:

- {{url}}/school_districts?page=2: gets school districts on a certain page
- {{url}}/school_district?id=451: gets school district with specific id
- {{url}}/communities?school_districts_id=54: returns communities associated with school district with specific id
- {{url}}/charities?school_districts_id=2: returns charities associated with school district with specific id

Tools:

- **Gitlab** used as our Git-repository manager
- We used **Docker** to deploy our backend
- Our project is hosted on **GCP**
- **Django** is our web framework
- Our Django application is running on **GKE**
- Our domain name is registered from **Namecheap**
- **React** JavaScript library used to build the User Interface
- **Postman** used to develop and test API
- Team communication was done primarily through **Slack**
- Front-end testing done with **Mocha, Chai, and Enzyme**
- Acceptance testing for our frontend done with **Selenium**
- Created UML diagram with **PlantUML**
- Spelling and grammar-checking with **Grammarly**
- **Bootstrap** framework used for front-end design

Hosting:

- **Google Cloud Platforming (GCP)** is used to host our website/API
- **Google Cloud Storage (GCS)** is used for storing and accessing data from database hosted by **GCP**
- Domain (EqualEducation.info) provided by **Namecheap**

Testing:

- **Unit Test - RESTful API using Postman**
- **Unit Test - JavaScript code using Mocha**
- **Acceptance Test: Java code using Selenium**

Mocha

Mocha is a JavaScript test framework running on Node.js and in the browser. We use Mocha for making asynchronous testing JavaScript.

Enzyme

Enzyme is a JavaScript testing utility for React. We use Enzyme for testing the React components by using `shallow` rendering.

Jest

We use the function `expect` from Jest. The function `expect` is used whenever you access to a number of "matchers" that let you validate different things.

Selenium

Selenium is a portable software-testing framework for web applications. We use Selenium to write automated tests to test our website.

Testing Components

We test components in isolation from the child components they render by using `shallow` rendering API from Enzyme. To install it, run:

```
npm install --save-dev enzyme enzyme-adapter-react-16
```

Or, alternatively you may use `yarn` :

```
yarn add enzyme enzyme-adapter-react-16
```

Here is the example how we test navigation bar component:

`test/unit_tests/Navbar.test.js`

```
import React from 'react';
import Enzyme from 'enzyme';
import expect from 'expect';
import { shallow } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';
Enzyme.configure({ adapter: new Adapter() });

import MyNavbar from
'../../src/components/MyNavbar/MyNavbar';

describe('MyNavbar', () => {
  it('renders without exploding', () => {
    expect(
      shallow(
        <MyNavbar />
      ).length
    ).toEqual(1);
  });
});
```

Import all the tools mentioned above including `enzyme-adapter-react-16`, and configure the `enzyme` with the `adpater`. Also, import the `.js` file containing the components that you want to test. I import **`MyNavbar.js`** as `MyNavbar` to test `MyNavbar` component. Then make an assertion to check if there is only one `<MyNavbar />` component with `.length` and `.toEqual(1)`.

To run the test, run the following command:

```
npm test
```

All the unit tests(`*.test.js`) under `test/unit_tests` directory will render without throwing, to shallow rendering and testing some of the output. With the navigation bar test, we have the following results.

```
> mocha --no-warnings --require ignore-styles --require
babel-polyfill --require babel-core --compilers
js:babel-register "test/unit_tests/*.test.js" -r
mock-local-storage

  MyNavbar
    ✓ renders without exploding

  1 passing (57ms)
```

- **Unit Test - Python code using unit tests**

Testing The Website

We use Selenium to write automated tests to test the frontend of the website. We utilize names, titles, and more to tests all of the pages as well as the functionality of each page. Our file can be found in **`test/acceptance_tests/guitests.py`**

To run selenium you must have the webdriver installed. This can be done on multiple web browsers, but our test code utilizes Selenium's Firefox webdriver. To install the Firefox Webdriver, first make sure that the path ~/.local/bin is in your execution PATH. Then:

```
$ wget
https://github.com/mozilla/geckodriver/releases/download/v0.19.1/geckodriver-v0.19.
1-linux64.tar.gz
$ tar xvfz geckodriver-v0.19.1-linux64.tar.gz
$ mv geckodriver ~/.local/bin
```

Then install Selenium:

```
$ pip install selenium
```

We test the pages of the website in the main method which calls the individual tests for each page.

```
# ----
# main
# ----

if __name__ == "__main__":
      driver = webdriver.Firefox()
    test_home(driver)
    test_school(driver)
    test_volunteer(driver)
    test_charity(driver)
    test_about(driver)
    driver.close()
```

In the code that follows we demonstrate some of the methods that each page tested on uses. These check the header, title and footer of the page, which should remain consistent throughout the pages.

```
# Check header.
def check_header(driver) :
      header = driver.find_element_by_id("page-header")
      assert header.is_displayed()

# Check title.
def check_title(driver) :
      assert "Equal Education" in driver.title

# Check footer.
def check_footer(driver) :
      footer = driver.find_element_by_xpath("/html/body/main/footer")
      assert footer.is_displayed()
```

Within the home page, we also test the elements in the carousel. We confirm that the text displayed is correct by finding the elements by the class name “carousel-option”, as well as using the xPath of them.

```
# Test carousel elements.
      a = driver.find_elements_by_class_name("carousel-caption")
      # Should enter three times.
      count = 0
      for elem in a :
            if count == 0 :
                  assert elem.text() == "School Districts"
                  xpath =
find_element_by_xpath("///*[@id="myCarousel"]/div/div[1]/div/div/h1")
                  assert xpath.text() == "School Districts"
            else if count == 1 :
                  assert elem.text() == "Charities"
                  xpath =
find_element_by_xpath("//*[@id="myCarousel"]/div/div[2]/div/div/h1")
                  assert xpath.text() == "Charities"
            else if count == 2 :
                  assert elem.text() == "Communities"
                  xpath =
find_element_by_xpath("//*[@id="myCarousel"]/div/div[3]/div/div/h1")
                  assert xpath.text() == "Communities"
            count += 1
```

If all is correct, then none of the assertions shall fail. In this way, we tests the multiple pages of EqualEducation.info to confirm that it is functional.