Source Code Document

CIS

https://github.com/COMP-4882-CIS

Backend:

https://github.com/COMP-4882-CIS/CIS

The backend is created using NestJS, a combination of TypeScript and javascript. And node package module for packages. We install these packages by running npm i. We must specify the CENSUS_API and the NODE_ENV= in a .env file we will need to create when running locally. The backend allows the frontend to request data for population, zipcodes, tracts, etc. The backend runs on port 3000, which the frontend requests from as an API. The backend is hosted by Heroku and is used as a source for the frontend (hosted on github pages) to request data from when clicking on a resource.

Important files and App Structure in the **CIS** folder:

CIS/package.json

This file consists of information like the node version, license, scripts, and dependencies. This file will update/generate on its own when adding dependencies via npm i.

CIS/package-lock.json

This file consists of the list of packages used, and will also auto update upon running npm i, or npm i and will automatically add any packages used within the project.

CIS/app.json

This file contains the project name, the link to the repository, and includes addons such as Heroku, and Heroku/nodejs. This file will need to be changed when changes are made to the host of the backend.

CIS/procfile

This file is needed to run the project with Heroku, can be left alone unless you want to change the host of the backend.

CIS/tsconfig.json

This file contains options for how the compiler will work with js and ts.

CIS/.env

This file is not contained on github and will need to be created if you want to run this locally, create a new file, and add CENSUS_API = API_KEY, and NODE_ENV = production or development, and REDIS_URL = redis_url if you are using REDIS caching.

API key can be requested here, https://api.census.gov/data/key_signup.html

Now let's look at **CIS/src** folder and how data is retrieved and exported from the backend to the frontend:

CIS/src/main.ts

This is the main file for the backend app, here we can specify the port of the backend, currently on 3000, this is the port the frontend will retrieve api data from.

CIS/src/module.ts

This is a module file used to import all of the data modules we will use in the backend, and also import the export module for exporting data to the frontend.

Next, we will look at how the CENSUS data is retrieved, this is the population data, poverty data, zip code data, and tract data.

The **CIS/src/census/controllers** folder is used to specify the data you want to retrieve via the CENSUS API, we have the requests we want like age ranges FifteenTo19, and UnderpovertyLevel.

The **CIS/src/census/enum** folder is used to contain the census variables we need equaling the census request code.

The **CIS/src/census/responses** folder is used to specify the type of requests we are using, so under-18, and provides the census data by tract and zipcode.

The **CIS/src/census/services** folder is used to specify the date we are requesting the census data from, currently is set to 2019, and we also specify the county we are requesting from, Shelby County, and to request each of the variables we specified in the controllers and enum.

The **CIS/src/census/types** folder creates the requests into variables.

The **CIS/src/census/census.module.ts** file contains are modules which allow the census data to be requested upon selecting a feature.

Next, we will look at how the Memphis-Datahub data is retrieved, this is the data for libraries, community centers, and parks.

The **CIS/src/mem-datahub** folder is used via api links that are available at each page of the data.

The **CIS/src/mem-datahub/controllers/** folder is used to specify what data we want to request.

The **CIS/src/mem-datahub/responses/** folder is used to choose what we are requesting.

The **CIS/src/mem-datahub/services/** folder is used to contain requests for each of the things we are wanting,.

The **CIS/src/mem-datahub/types/** folder is used to convert each request into a variable.

Next, we will look at how the Schools data is retrieved, this is the data for schools locations, contact info, and grades.

The **CIS/src/schools** folder is used to request data via tn.gov school data link.

The **CIS/src/schools/Controller** folder is used to create api controller requests for schools' data

The **CIS/src/schools/data** folder is used to read schools data from excel sheet.

The **CIS/src/schools/services** folder is used to request each school in the excel sheet, and output as file for cache data.

The **CIS/src/schools/types** folder is used to convert school data into variables.

Next, we will look at how the data is exported to the front end, this is the data for libraries, community centers, and parks.

The **CIS/src/export** folder contains files that are used to export the data from the backend to the front end.

The **CIS/src/export/controllers** folder contains the requests that can be made from the frontend.

The **CIS/src/export/requests** folder contains the files that perform the request for the data.

The **CIS/src/export/schemas** folder contains the files data breakdowns we the data that we request from the backend.

The **CIS/src/export/services** folder contains the functions used to export the data.

Frontend:

https://github.com/COMP-4882-CIS/CIS-FRONTEND

Now let’s look at the frontend. Our frontend is created as an **Angular app**, and we use **node modules** for packages. One important package that we use is **leaflet,** this allows us to implement the map, its nodes, the search bar, and the layer tree. We install these packages by running **npm I –force**. The application will run locally at localhost:4200 after running **npm run start**, or the application is live at https://comp-4882-cis.github.io/CIS-FRONTEND/ with the front end being hosted via github **pages**, under the settings for your frontend repository github page.

Important files and App Structure for CIS-FRONTEND:

CIS-FRONTEND/angular.json

This file specifies our angular app and builders CIS:build:production, and CIS:build:development. Also contains our default project, which is CIS.

CIS-FRONTEND/package.json

This file consists of information like the node version, license, scripts, and dependencies. This file will update/generate on its own when adding dependencies via npm i.

CIS-FRONTEND/package-lock.json

This file consists of the list of packages used, and will also auto update upon running npm i, or npm i and will automatically add any packages used within the project.

CIS-FRONTEND/SRC folder:

CIS-FRONTEND/SRC/main.ts

This file is our main file for cis-frontend, it enables our app module and enables production mode.

CIS-FRONTEND/SRC/assets

This folder contains files for all of our data, and what are icons look like. Our data files are in a .GeoJSON format that contains coordinates for each row that allow us to place our points of data on the map. *When we want to add a new data, we would add a new .geojson file with the data and coordinates of each point.*

CIS-FRONTEND/src/environments

This folder contains files for our production links, which use our api url, https://cis-901.herokuapp.com/ this allows us to take data from the backend. And our help video is also located here. We also have a file that contains our http://localhost:3000 that allows us to take data from backend where running locally.

CIS-FRONTEND/src/app folder:

This folder contains the bulk of our app, we have a backend folder, components folder, and helpers folder which are really important to us.

CIS-FRONTEND/src/app/backend folder:

CIS-FRONTEND/src/app/backend

This folder contains our requests, response, services, and types folders. These folders are used to request data, and then convert this data in nodes on the map, and allow us to specify data variables in the side bars for each resource we click on.

Our **Types** folder will contain the specific variables for each resource that we use, types/geo/features/layer contains the zipcodes, tracts, and district features. types/geo/features/points contain all of our other data features. *To add more data to the map, we will need to add a new point and specify its features here.*

Our **services** folder will contain our export service, our geo-tract service, and our landmark service, which allows us to call data within the project.

Our **response** folder will contain our responses from the data.

Our **request** folder will allow us to specify the data we want.

CIS-FRONTEND/src/app/components folder:

CIS-FRONTEND/src/app/components

This folder contains all of our static frontend code. Each folder in the component is for a specific visual feature we see on the application. Each folder in the components will consist of a TypeScript file that we specify the functions of each feature, and a .html file that consists of our angular code to build our website, and a .css file that allows us to further style the features.

CIS-FRONTEND/src/app/components/breakdown-chart

This folder contains files that will affect how the population breakdown-chart in the sidebar will display and function when a zip code is selected.

CIS-FRONTEND/src/app/components/breakdown-summary

This folder contains files that will affect how the summary of the zipcodes will display and function when a zip code is selected, and the map sidebar displays.

CIS-FRONTEND/src/app/components/crime-breakdown-summary

This folder contains files that will affect how the summary of crimes in the map sidebar are displayed and function when you click on a crime feature.

CIS-FRONTEND/src/app/components/map

This folder contains the files that will affect how the map and its nodes will display and function.

> The map.component.ts file will be used to add a new point to the data, it contains packages like leaflet and its plugins- **leaflet-search, leaflet-tree, and leaflet-cluster**. Leaflet (https://leafletjs.com/) and its plugins are a package installed via node modules, this package allows us to display our map, and specify the maps constraints and display/add the nodes for each feature. We also have variables for each layer (eg. Zipcodes) and point (eg. Libraries) and *if you want to add additional data, this is where we would add another point or layer variable.*
>
> **leaflet search**, is a leaflet plugin, which was not created with support for typescript so the leaflet-search we install from npm I will not work on its own, but we use a typescript file called index.d.ts found in the /src folder, which gives us type features that allow us to call leaflet search features. We can make visual changes to this feature in the .css file
>
> **Leaflet control layers tree**, is a leaflet plugin that was created with typescript support, and it implements the checkboxes for each layer and point on the map. We have a base tree that contains all of the points, and a second tree that contains the layers. We can make visual changes to this feature in the .css file
>
> **Leaflet marker cluster**, is a leaflet plugin that was also created with typescript support, it allows us to cluster the many points of our crime data so we don't have to load the all at once. We can make visual changes to this feature in the .css file

CIS-FRONTEND/src/app/components/map-box

This folder contains the files to edit the functions and display of the work cited button and poverty-detail button in the bottom right, it also displays the app title and app version.

CIS-FRONTEND/src/app/components/map-sidebar

This folder contains the files to affect the functionality and visualization of the map sidebar that opens when you click on a feature. We are able to effect how data is displayed for health and crime when clicking on a zipcode.

The **health data** is implemented in a unique way, the data is entered in the .ts page, and and if statement is called whenever the specified zipcode is selected.

CIS-FRONTEND/src/app/components/point-feature-summary

This folder contains the files that involve the functionality and visuals for each specific feature or point on the map, (eg. not zipcodes). Here we will affect what displays when we click on (for example libraries) and its popup sidebar opens.

EX. Child-Care is given if statements that check if it contains a specific variable in its backed/…/points folder, if it does, it will display, and since whenever a child-care point is selected it will always contain that variable, it allows us to create specific point features for each type of point we have.

CIS-FRONTEND/src/app/components/poverty-breakdown-summary

This folder contains files that involve what displays when we select the Poverty tab in the zipcode feature sidebar.

CIS-FRONTEND/src/app/components/poverty-chart

This folder contains files that are used to display the poverty-chart and its functions when we click on the Poverty tab in the Zipcde feature sidebar.

CIS-FRONTEND/src/app/components/school-feature-summary

This folder contains files that are used to display the school sidebar features. The school features contain more data than most and are implemented with an Angular dropdown.

CIS-FRONTEND/src/app/helpers folder:

CIS-FRONTEND/src/app/helpers

These helpers are TypeScript files that allow us to add additional visual changes to things like charts, features, layers, popups, sidebars, and tool tips.

CIS-FRONTEND/src/app/helpers/chart-data.helper.ts

This file is used to generate the charts for poverty, school, and population data. We can assign variables to the charts and labels. These charts are then imported and called in the correct component folder.

CIS-FRONTEND/src/app/helpers/feature.helper.ts

This file allows us to change how features will appear on the map. Here we can assign icons to features and attach tool tips to each feature.

CIS-FRONTEND/src/app/helpers/layer.helper.ts

This file allows us to style layers like zipcodes and districts.

CIS-FRONTEND/src/app/helpers/sidebar-data.helper.ts

This file is used to change how the sidebar will act when clicking on features.

CIS-FRONTEND/src/app/pages folder:

CIS-FRONTEND/src/app/pages/poverty-level

This folder contains the poverty info page.

CIS-FRONTEND/src/app/pages/works-cited

This folder contains the works cited page.

Other Important information:

- packages are installed via **npm I** for both frontend and backend. With the frontend, you will probably need to run **npm I –force** to get all the packages installed correctly, and just **npm I** for the backend.
- Packages like certain leaflet plugins maynot have typescript support and will need to find a workaround to get them working, for ex. Leaflet search, we had to find a workaround to get that working for typescript since it does not natively come with typescript support.
- Adding data is fairly easy, should just add it to backend, and map components, and add the icons in helpers
- NodeJS and Angular will need to be installed to run both the frontend and backend, we do not have to have redis caching working when running it locally, but you do need a census_API key in a .env file that you will create and store in CIS folder
- Data like childcare, and school testing data, and health data are data that does not have an API, and will need to be added manually. Health data is added in map-sidebar folder in components, and school testing data is added using pandas in a jupyter-notebook, we just added new columns and assigned each data by its zipcode. For childcare, we

trimmed the childcare data excel we were given, added the variables in backend and frontend like normal.

- As this is being written, the website is up and running with the final code, everything should run like normal if you have the required software installed. First clone and cd into the backend, **npm I**, and then **npm run start:dev** everything should start up if we have our .env file included, it will not be obvious that the backend is up and running, it will display connections and then nothing, if that happens then everything should be good. Next we can clone our frontend, cd into it, run **npm I –force,** then **npm run start** and then we should see localhost:4200 that we can view our locally running website at.