MetroPulse-NYC: Phase I

IDB Group 6

- Alex Cabrera (gac2827)
- Kamil Kalowski (ktk582)
- Kyston Brown (kmb6273)
- Thomas Moody (tjm4482)

Motivation

The city of New York has recently published concerning statistics about the rise of hypertension problems in the city. In particular, the disease is disproportionately affecting black communities, with the prevalence of hypertension in the community being almost double (44%) that of the white community (23%). More concerning is the fact that of those in the community suffering from the disease, only 30% report having controlled blood pressure. With a number so low, we aim to help this community to more easily keep their blood pressure in check by finding local blood pressure testing centers, and in case of emergency find a facility able to help them stabilize their condition. The development of our website focused on answering the following questions (TODO?):

1. How can black NYC residents suffering from hypertension control their blood pressure consistently?
2. Where can black NYC residents find urgent medical care when experiencing hypertension-related blood pressure issues?
3. What facilities geared at aiding their high blood pressure can be found in their district?

User Stories

Implement endpoint routing to link pages

- When we received this user story, our group had already discussed using react-router to link to all the separate pages in our project. We couldn't start yet since most of our pages were half-finished and independent of one another. But finally, a couple of days before the project was due, we were able to set up routes to all pages successfully.

It would be nice to input my current location and see what facilities are closest to me

- We considered marking this issue as a duplicate, given that this functionality is partially covered by our neighborhood model, which provides a small enough area to browse for nearby facilities if you know your location. However, it might be useful to automatically transform a user's location to a neighborhood instance they can browse. We would need to be careful with this feature to not intrude into each user's privacy needs.

It would be nice if I could see a credibility metric for the different facilities

- This is information we hadn't considered adding to the website, which would be really useful for any client looking for a decent place of treatment. However, we decided that the credibility metric would need to change to a rating system, to avoid giving medical advice on the website. We've considered using the Yelp API for this in Phase 2.

As a user, it would be nice to also have information about the doctors at the medical facilities

- Our group initially considered adding this information to our medical instances. However, we were unable to find a public API which provided this information for all medical facilities in NYC. Thus, we considered it out of scope for the project for the time being, and set it as a potential goal for Phase 1.

Design endpoint for getting all district instances and specific district instance

- Our group worked through designing these specific endpoints using Postman today. We drew inspiration from other pre-existing api's we had looked through on NYC Open Data. For example, our medical facilities' page's link is https://www.metropulse.link/medical/ so we made the api for getting all medical instances https://api.metropulse.link/medical?borough=Name&nta=Name&zip=Number with borough being a filterable parameter, nta (neighborhood) being a filterable parameter, and zip (zip code) being the last filterable parameter. For getting a specific medical facility, the api is https://api.metropulse.link/medical/:id, with id being a path variable for the internal id of a medical facility that the client would feed in.

RESTful API

https://documenter.getpostman.com/view/29785582/2s9YJZ3jGy

Models

	Filterable/Sortable Features	Media
Test Centers	1. Name 2. Neighborhood 3. Borough 4. Zip Code 5. Council District	1. Map 2. Center Photo
Medical Facilities	1. Name 2. Neighborhood 3. Borough 4. Zip Code 5. Council District	1. Map 2. Facility Photo
Neighborhoods	1. Name 2. Borough 3. Population 4. Area Codes 5. Number of Test Centers 6. Number of Medical Facilities	1. Map 2. Area Pictures

Tools

- **React:** A javascript library for designing websites.
- **React-bootstrap:** A framework providing components from the bootstrap project reworked for react development.
- **React-router:** A react package providing a navigation system in-between different pages of the project.
- **Zoom:** Video conferencing platform used for group meetings.

- **Ed Discussion:** Chat/Forum hybrid for posting questions and communicating with other team members.
- **Postman:** A platform/app for building, testing, and deploying APIs.

Design

The frontend of our website is laid out in the standard manner for a React project, all inside the frontend folder of the repository. The entire code for each component we developed is inside the src folder, each file separated into their own independent folders inside based on their functionality.

We have five main categories of functionality:

- **Instances:** Three separate js files each representing an instance for a particular model, all using bootstrap components for better-looking functionality. All three files are laid out very similarly to one another. First there is an *Info object which is in charge of displaying all the information of each instance, which is passed in as an argument. This *Info object is then used as a component of an *Instance object, which fetches information from a json file, passes a portion to *Info, and renders images next to the information. All instances also use some CSS to style themselves as cards, imported from the Instance.css file.

- **Models:** Three separate js files for each model page in frontend/src/models: Facilities.js, Hoods.js, TestCenter.js. The model pages are accessed from the navigation bar on the homepage. Each page lists the name of the model and the number of instances. Each instance is displayed using a bootstrap card grid displaying the name and attributes for each. The data for each instance is imported from the corresponding .json file in frontend/src/shared/data. The data for each attribute is accessed using (name of instance).(name of attribute) e.g. facility.name. Each card links to the corresponding instance using Link from react-router-dom. The path for each instance page is added to the routes in the App() function in frontend/App.js.

- **Homepage:** The homepage gives an overview of hypertension and how it affects the African-American community in New York City. We give a quick statistic and provide a link to a well cited clinic (also a clinic that we use for one of the instances of our medical facilities) that gives a more in depth overview on Hypertensions causes, symptoms, and dangers. Using inline css and defining a styles object for a container, the header, and title, we return a div with a header with name of our application, another header explaining exactly what hypertension is, and a paragraph giving a simple statistic about the disproportionate number of people in the African American community suffering

from hypertension, and finally a footer asking for support. We also instantiate a constant the stores the URL link that we use to redirect users to the clinical information page when a user clicks learn more on our webpage We export this HomePage component at the bottom which is later used as a route. This component is focused on engaging users and providing contextual information.

- **About:** This category is responsible for the About page, It corresponds to a JSON file with data about project members, a CSS formatting file for the page as well, and the main About.js file that forms the page itself. The JSON file has as many members as decided upon and contains the data to fill out the Bio, username, email, etc. If the data on the Bio cards needs to change only the JSON file needs to be updated, if more data is added to the cards, the JSON must be expanded. The About page itself contains two functions dedicated to pulling from our Gitlab API to get commits and issues and the main app function to manifest the page. The page itself is mostly a couple of text block sections and two card grids, of resource cards (resCard) and team member cards (team card).

- **App(.js):** This is the file that ties all our work together. Here, we set up a navbar that is shown on every page of the website. More importantly, we set up routing for each of our components to be able to link and navigate to/from them from every other page (specially the homepage). For this, we use the react-router package, and create “Route” objects for every page in the website. Setting up routes for all the instance pages was tricky; the final implementation uses a for loop to first store all instance information in an array, and then we call map to iterate and create independent links for all of them. To ensure they are all independent, each instance has an “id” field in their respective json data files which we use to build the links. So, a link to an instance is of the form https://{url}/{model}/{id}.

Hosting

We set up an account for the AWS Console. We navigated the Amplify service, clicked connect app and connected our GitLab repository. We then edited the amplify.yml and build settings to ensure our app was building properly. We then set up a domain (https://www.metropulse.link) with Route 53, navigated to hosted zones, selected the public hosted zone, and clicked create. We navigated back to Amplify and went to domain management, and added the domain we just created in Route 53. The record sets were luckily set up for us, including the CNAME record ANAME record. We then just had to wait for the domain changes to propagate across the internet, which took

approximately 20 minutes which was great as it can sometimes take up to 48 hours. SSL was enabled naturally by Amplify. It's also worth mentioning that Amplify uses a AWS Cloudfront distribution behind the scenes to ensure the application works with low latency for anyone who wants to use it across the world. A little bit on how CloudFront achieves this: CloudFront caches content in multiple data centers around the world, persists a pool of persistent connections to origin servers, and continuously monitors the health of the network and broader internet.

Challenges

We ran into these challenges:

1. **Project RFP:** It took our group a long time to understand the project prompt and find an appropriate topic to build upon. Once our project idea was approved, we had less than a week left to build the website, so we had to more strictly partition development time into our schedules.
2. **AWS Deployment:** We struggled with properly deploying our website through AWS amplify. It did not give any errors during the build process, but it would not appear in either our domain name or AWS's own address. We ended up trying to transfer our domain to AWS Route53, but that was taking too long, so we ended up setting up a new domain with AWS Route53 itself which proved to be seamless. All the correct host names were automatically set up and we didn't have to worry about any DNS probing.
3. **JavaScript/React:** None of us were very familiar with JS, React, APIs, or Postman before this project. So, it was challenging to figure out how our tools worked while we were actively using them. However, we overcame the difficulties before the project was due.