Doctor Search Technical Report

Abhinav Kasamsetty, Nathan Hoang, Derrin Ngo, Sitong Li, Rohan Arya

Motivation

Currently, there aren't many good ways to search for doctors on the internet. People can search for doctors in their area on Google, but the results are not all-inclusive and only show the closest results. Because of this issue, we came up with the idea for Doctor Search, a website where users can search for all doctors in major cities and learn what specialties are being practiced in them to make an informed decision on which doctor to go to.

Tasks

1. Create a website about doctors, cities, and specialties and an about page.
2. Create Postman API documentation.
3. Get a domain and add SSL support to the website.

To resolve the first task, we used React and NodeJS and constructed a website. We manually found data from our data sources and entered information about doctors, cities, and specialties into JavaScript arrays, which were then used as data in the React application to form the components on the pages. To improve the look of the website, we used the Material-UI CSS framework. On the about page, we manually entered information about biographies and tests but used the GitLab API to dynamically generate each member's issue and commit count.

For the second task, we used the Postman desktop application to construct our schema. We wrote an API schema using the OpenAPI 3.0 standards and generated a collection and documentation from the schema. The schema can be found in `backend/api-schema.yaml`. More information about the API is in the RESTful API section below.

To complete the last task, we obtained the domain doctorsearch.me from Namecheap for free, where our website is currently at. We added SSL support by changing our DNS provider from the domain to AWS Route 53 and using AWS Amplify to add a domain to our website, which automatically handled SSL certification.

User Stories

Customer User Stories

We were tasked with the below:

1. **Rating Scale** - What are the ratings for the doctors out of? 5? 10? I think it would be more clear if you had '4.6/5' instead of '4.6'.
2. **Title** - When you look at the title (in the tab with the favicon), SEARCH is capitalized. Does search stand for something? If not, maybe change it to uppercase for consistency.
3. **Cities Zip Code** - I'm assuming the first string of digits represents the zip code for each city card. I think it would be more clear if you explicitly state that as a zip-code as well as to keep the consistency.
4. **Whitespace** - There seems to be a lot of white space when you click on additional info, maybe you can have the information and location side by side to help fill this in?
5. **Center titles on model pages** - I think it would be cleaner if you could center the titles on the model pages since your card components are already aligned in the center.

The first three user stories were easy to resolve as they were quick changes to the text in the React application. However, the fourth user story was more difficult as it required us to use the `Grid` component from Material-UI on the instance pages with one column for the information and one for the map. The last user story required changing some styling of the text in our Material-UI `Card` component for each model.

Developer User Stories

We tasked our developers with the below:

1. Get a domain from a website such as Namecheap. Eventually, the website will use this domain with https support. The domain name should match the name of the project, Costly Commute.
2. There should be a Postman API Schema with information about the API endpoints. The endpoints should be centered around the models used in Costly Commute. The technical report should contain a link to the schema documentation.
3. Host the website on the domain using various AWS services. AWS Amplify could be used for React or an S3 bucket could be used for a purely static website with only HTML/CSS. The pages on the website will be served through AWS.
4. Create the splash, model, and instance pages that will likely be only static. There should be at least 13 pages (every page except for the about page). The model pages should have the instances displayed in a grid or a table and the instance pages should have a lot of attributes and some media displayed.
5. Create the about page of the website, which contains information about the developers of the site. The page should show a picture and GitLab information about the repository used to develop the site. The GitLab API should be used to dynamically get this information.

Models

There are three models for the website.

1. Doctors
    a. **Description:** Doctors that can provide medical services. The model page contains a profile picture to identify the doctor and lists five attributes describing the doctor. Users will be able to sort this model by city, specialty, rating, name, and gender. Each instance page contains the list of attributes, a profile picture of the doctor, and a picture of the doctor's location on a map.
    b. **Attributes:**
        i. Name
        ii. City
        iii. State
        iv. Phone
        v. Specialty
        vi. Name of Practice/Affiliated Hospital
        vii. Address
        viii. Insurance plans accepted
        ix. Rating
        x. Biography
        xi. Website
2. Cities
    a. **Description:** Cities in the United States. The model page contains a picture of the city and a list of five attributes that list statistics about the city. Users will be able to sort the cities by population, state, number of doctors within the city, specialties, and name. Each instance page of this model includes a picture of the city and a map of the city.
    b. **Attributes:**
        i. Name
        ii. State
        iii. County
        iv. Zip Code
        v. Population
        vi. Mayor
        vii. Area in Square Miles
        viii. Number of Doctors in the city
        ix. Specialties
3. Specialties
    a. **Description:** Different types of medical specialties practiced by doctors. The model page contains a representative picture of the specialty and a list of five attributes that describe the specialty. Users will be able to sort the specialties by

city, number of doctors, name, and category. Each instance page of this model includes a representative picture of the specialty and a map of places in the United States where the specialty is practiced.

b. **Attributes:**
   i. Name
   ii. Category
   iii. Number of Doctors
   iv. Doctors practicing it
   v. Cities it is in
   vi. Description

RESTful API

Postman API Documentation: https://documenter.getpostman.com/view/9000368/SVtPXqcN

The Postman API documentation is split into three sections, **doctors**, **cities**, and **specialties**, which are the three models described above. Each model has two endpoints that correspond to it. Using the model **doctors** as an example, we have:

- `/doctors`
  - This endpoint allows users to get a list of doctors and filter them based on the optional parameters that are described in the documentation.
- `/doctors/{id}`
  - This endpoint allows users to get a specific doctor based on the ID that uniquely represents it.

The other two models each have two endpoints that work in the same way, which will eventually allow the frontend to dynamically pull data to display.

The documentation was generated using the Postman desktop client. We wrote an API schema using the OpenAPI 3.0 standards in YAML, which then could be converted to a collection and documentation. The schema can be found in `backend/api-schema.yaml`.

Tools

- **GitLab** - We used GitLab to store the source code to DoctorSearch and. We also used it to version control our code and track issues.
- **Postman** - We used Postman to create our API Schema and generate a collection and documentation from it.
- **Material-UI** - We used Material-UI as our CSS Framework to make sure our website is responsive. We constructed the navigation bar and grid of model cards using CSS components from the framework.

- **Slack** - We used Slack for communicating within the team.
- **React** - We used React as our frontend framework to build the website, handle the routes for the different pages, and create UI components that could be reused across pages.
- **React-Router** - We used React-Router to build routing for the website so that users could go between the home page, about page, model pages, and instance pages without doing a full page reload.

Hosting

- **Namecheap** - We used Namecheap to get our domain, doctorsearch.me.
- **AWS Route 53** - We used AWS Route 53 as our DNS resolver for our website. To achieve this, we created a new hosted zone in the AWS Route 53 console, which automatically generated nameservers. We then entered the nameservers in the Namecheap domain settings so that AWS Route 53 would handle DNS resolution instead of Namecheap.
- **AWS Amplify** - We used AWS Amplify to host our React application. We made a new project in the AWS Amplify console, which we connected to our GitLab repository. We then wrote a build script that AWS Amplify runs every time a new commit is pushed to the repository, which runs `npm run build` in the `frontend` directory to update the website. We added our AWS Route 53 managed domain to the project, which automatically handled SSL certification.