CS-373 IDB Technical Report: Group 1

Motivation

The motivation behind the project is to allow users to find jobs based on geographical location and/or price and simplify the housing search for those looking to move, especially if they have a budget or preferred location.

User stories

Phase 1

User Story: Search for cities by job

"One feature that would be very useful would be the ability to filter/search cities by a specific job title(s). This way, a user could find all the cities they could work in and pick one to live in based on information on each city. Right now, you can search for jobs which contain information on what city they are located in, but it's extremely likely that multiple jobs will be located in one city, so it isn't very useful for finding what cities you could potentially work/live in."

This is a useful idea! It should be doable because we can get location data for each job posting and we can search for job titles in the API. We will implement this in Phase 3.

User Story: View City Job Demographics

"I'm a recent graduate who is looking to move somewhere new, but don't know what kind of job I want. Currently I can search for individual jobs in a city, but ideally I want to be able to view the top jobs in each city. Having the ability to view the top X jobs/industries in a city by quantity/pay/rating would allow me to get a better sense of where I want to live."

This idea seems to be challenging but worthwhile. We should be able to get information on the number of jobs within a certain industry in a city, as well as information about pay within each job and industry, which we can then either display on the city page or on the jobs page. We will have this implemented by Phase 3, once we have implemented sorting and searching.

User Story: City budget search

“Hi, I already have a remote job and I need to know what cities fit my budget. Could there be some sort of price search which will return the closest cities and apartments as well as similar ones? Maybe a range or average cost?”

This idea seems redundant with the ability to sort cities by budget score, then sort apartments by price within that city, which will already be implemented in Phase 3.

User Story: Implement job search by apartment

“*I'm a very materialistic person, so I've decided to search for my dream apartment first. I'll need to come up with the money somehow, so I'll need to get a nearby job. Implement the reverse of your proposal: given an apartment, show me all the possible jobs in the city that I'd have to get in order to financially support my dream of living in this apartment.”*

This idea results in the list of jobs being sent back to the user being the same as that of the list of highest paying jobs, up until the point at which the income is not enough for the apartment. However, we will try to implement a reverse search in Phase 3.

User Story: Filter jobs by several categories

“I'm a recent college graduate looking for a job and I would like to be able to quickly find the best jobs that I am suited for. Specifically, being able to filter by experience requirements to find something entry level would be a huge save on time and on top of that filtering by location and salary would be great as well.”

This is a great idea! Using multiple filters would absolutely be useful when searching for jobs that fit a specific skill set and situation. This will be implemented in Phase 3, when sorting and searching are implemented.

Phase 2

User Story: Given a job and city instance, describe my commute

“I have used your website to first find a job and then an apartment within the city. I would like to know what my daily commute would look like. Could you give some sort of indication of what I should expect when traveling to my job? It could be something like directions, duration of travel, recommended travel method (like walking, transit, car), or distance.”

I think a lot of this is dependent on the data we're collecting, so we'll look into that once we establish how we're filtering things in Phase 3. The most likely is looking into distance.

User Story: Explore Nearby Cities

"I am someone who is living in a state where major cities are very close and I can theoretically travel to any. It would be beneficial to sort by distance from my current city, and get both apartment and job listings within a certain range. If this is too generalized being able to see the listings in the closest city or two would be helpful."

This kind of goes along with the filtering we want to get done in Phase 3, so we'll try it out then. It also goes along with the earlier issue from Brian where he wanted to see similar cities. We'll look into implementing this!

User Story: Display walk scores on the instance page

"Is there any chance you could put the walk, transit, and bike scores of a city on the instance page itself? I think people would have a better user experience if they didn't have to leave the site to check those ratings. You can still keep the link to walkscore.com, since it provides some additional information, but I just want to quickly be able to find a city's walk score by opening its instance page."

We'll look into collecting the walk score data and displaying it within this next phase.

User Story: Similar Cities

"I'm not sure if this is something you are filtering on already, but it would be cool if on the city instance pages, there were suggestions for similar cities with overlapping things that they're known for. This would be really nice if there was some reason you couldn't live in a certain city. then you could find alternatives quickly."

This kind of goes along with the sorting/filtering we want to do in Phase 3, so we'll work on this then.

User Story: Buttons for changing between models

"Right now on the model pages, there is blue, hyperlinked text to switch between various models. I think it would look a lot better if these hyperlinked text bullets were instead changed into some sort of GUI button that users could click. Right now, it looks fairly ugly, and changing it would help make the webpage look better."

We just fixed this! Thanks for your input!

RESTful API

https://documenter.getpostman.com/view/23607622/2s83tFHBCK

Models

City - This model provides all around information about living in a given city, including: population, rating, budget score, safety score, ability to get around on foot, bike, or public transit, images, and various tags on what the city is known for.

Apartment - This model provides information about apartments in a given city, including: location, rating, number of reviews, phone number, and images.

Job - Provides information about any job given a job title or industry, including: company, expected salary range, location, job description.

Tools

General

Discord: online communication platform
GitLab: git software that allows for managing version control and continuous integration/ continuous deployment
Visual Studio Code: code editor
Zoom: online communication platform

Frontend

AWS Amplify: platform for deploying and hosting web applications, supports continuous
Axios: HTTP client library, used to access APIs from React application
Docker: Used to create environment to be able to use tools, Flask and SQLAlchemy
Material UI: provides React components that implement Google's Material Design standard
NameCheap: domain name registrar, allows purchasing and managing domain names
integration and continuous deployment
Node Package Manager (NPM): manages installation of node packages
React: JavaScript library for building user interfaces, powers the front-end
React-Bootstrap: CSS framework built on React, simplifies integration of the frameworks
React Router: library for routing in React, enable navigation between pages

Backend

AWS Certificate Manager: Used to provision, manage, and deploy public and private SSL/TLS certificates for use with AWS service
AWS Elastic Beanstalk: An orchestration service offered by Amazon Web Services for deploying applications which orchestrates various AWS services, including EC2

AWS Relational Database Service: Supports an array of database engines to store and organize data and helps with relational database management tasks, such as data migration, backup, recovery and patching
Flask: Web framework used for implementing the API
Jest: Testing framework for JavaScript codebase, used for frontend testing
MySQL: Tool used to manage databases and organizing data within them
Marshmallow: Python library used for converting complex data types to python types
PlantUML: Open source tool for creating diagrams from plaintext
Postman: API platform for documenting, building, and using APIs
Selenium: Testing tool used to validate functionality of web applications
SQLAlchemy: Toolkit that allows Python programs to interact with databases
Unittest: Library for testing backend Python code and used in conjunction with selenium to test the front end
Waitress: A Web Server Gateway Interface used as calling convention for web servers to forward requests to web applications or frameworks written in the Python programming language

Data Sources

RoadGoat: get information about cities
Google Maps Embed: get maps showing where apartments and jobs are located
Realty Mole: get information about apartments
Adzuna: get information about jobs
GitLab: version control and issue tracking, gets information about commits and issues

Frontend Hosting

First, we registered the domain *geojobs.me* on NameCheap. For hosting, we used a tool called AWS Amplify, which is part of Amazon Web Services. We had to set up SSL/TLS with NameCheap and Amplify. This allowed us to connect different branches of our GitLab repository to different deployments, which we could then associate with different CNAME records on NameCheap. We set dev.geojobs.me to point to the development branch and geojobs.me (and www.geojobs.me) to point to the main branch.

Backend Hosting

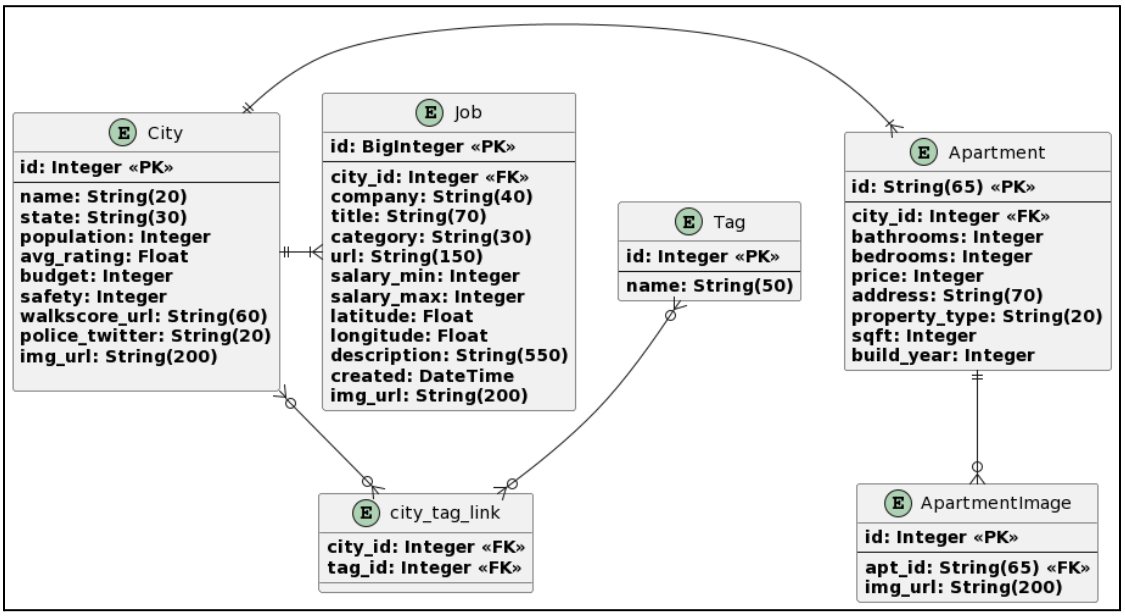
We created a MySQL database and used AWS Relational Database Service to host it. We used Waitress as a production-grade WSGI server for our Flask application that runs the API, and then hosted that on AWS Elastic Beanstalk using Docker on Amazon Linux. To allow HTTPS connections, we obtained an SSL certificate from AWS Certificate Manager, and then set up a listener for the application load balancer on port 443 that used the previously acquired SSL certificate and forwarded HTTPS traffic to the Flask application running on the Waitress server.

After configuring the security groups for RDS and Elastic Beanstalk to accept incoming traffic on their respective ports, we were able to access these services programmatically.

Database Implementation

For our database implementation, we first created models for the different types of data we would be storing in our database and set up relation tables in Flask-SQLAlchemy. We then requested data from various APIs and stored them in JSON files. For additional information that we could not obtain from the APIs, we used Selenium to scrape the web and store this information in supplementary JSON files. To populate the database, we fed these files into a Flask-SQLAlchemy application that created the tables and added the instances in the JSON files for each model. In total, we had 6 tables in the database (as shown below): 5 for the models and 1 to hold the many-to-many relation between cities and tags.

UML Diagram for Database



Pagination

Pagination was implemented in the backend, where the front end would pass a page number and a limit per page as arguments in the request to the backend, then the backend would only return a number of instances up to the limit and return a set of instances that corresponds to the page number.

On the front-end, react-bootstrap pagination components were added to the top of the Jobs and Apartments pages.

Challenges

It was hard for us to get started with React and Bootstrap, so there was a learning curve before we could really do much on the front-end of the site. We had difficulty finding apartment APIs that gave us the information we wanted: for example, we tried Zillow, but it wasn't quite working for what we wanted to do. At some point, the AWS Amplify project was not redeploying when our main branch was updated, so we had to manually redeploy instead of it happening automatically; identifying this issue took a lot of time though, as we were not familiar with the intricacies of AWS Amplify.