

Semantic Visit Aware Recommendation of Hotels

FINAL REPORT

Team 34

Client: Goce Trajcevski

Advisor: Goce Trajcevski

Thomas Frohwein

Zachary Garwood

Dylan Hampton

Kevin Knack

Nathan Schenck

Britney Yu

Joseph Zuber

sdmay23-34@iastate.edu

<https://sdmay23-34.sd.ece.iastate.edu/>

TABLE OF CONTENTS

Table of Contents	2
Figures	4
Problem Statement	5
Intended Users and Uses	5
Functional Requirements	6
Non-Functional Requirements	6
Standards	6
IEEE Std 1012, Standard for Software Verification and Validation.....	6
IEEE Std 1219, Standard for Software Maintenance.....	7
IEEE/ISO/IEC 26512-2017, Requirements for acquirers and suppliers of information for users.....	7
IEEE Std 982.1, Standard Dictionary of Measures to Produce Reliable Software.....	7
IEEE/ISO/IEC 15288-2015, System life cycle processes.....	7
ACM 1.3.....	7
Engineering Constraints	7
Security and Countermeasures	7
Design Evolution since 491	7
Frontend Design Changes.....	7
Figure 1: Improved Recommended Hotel List.....	8
Figure 2: Distance Metric Selector.....	9
Figure 3: Number of Origins Input.....	9
Backend Design Changes.....	10
Implementation Details	10
Goals.....	10
Tools Used.....	10
Frontend.....	10
Figure 4: Example Generation of Recommended Hotels.....	12
Figure 5: Example of Generated Route for a Recommended Hotel.....	13
Backend.....	13
Figure 6: Implementation Architecture.....	14
Testing Process	14
Testing Results	15
Related Products and Literature	15
Appendix I - Operation Manual	17
Introduction.....	17
Download Project.....	17
Backend Setup.....	17
Frontend Setup.....	19
Demoing the Project.....	19
Testing the Project.....	20
Appendix II - Alternative Versions of the Design	21

Frontend Design Alternatives.....	21
Figure 7: First Alternative Frontend Design.....	21
Figure 8: Second Alternative Frontend Design.....	22
References.....	23

List of figures/tables/symbols/definitions

FIGURES

Figure 1: Improved Recommended Hotel List	8
Figure 2: Distance Metric Selector	9
Figure 3: Number of Origins Input	9
Figure 4: Example Generation of Recommended Hotels	12
Figure 5: Example of Generated Route for a Recommended Hotel	12
Figure 6: Implementation Architecture	14
Figure 7: First Alternative Frontend Design	21
Figure 8: Second Alternative Frontend Design	21

PROBLEM STATEMENT

Our client has developed an algorithm to efficiently determine the optimal semantically diverse PoIs as well as the optimal route between them given user constraints like maximum travel distance, preferred PoI categories, etc. Furthermore, our client has compiled a dataset of New York City that contains categorized PoIs and the road network that contains them.

The main objective of this project is to create a web application that creates a visualization for the algorithm mentioned above for travelers to display the optimal PoIs and the path between them as there are no current means to visualize the algorithms. Additionally, visualizations of other algorithms that determine paths between PoIs will be created to compare the PoI diversity along the routes to demonstrate the algorithm's effectiveness. Users will be able to input constraints from which the algorithm will compute the route to semantically diverse PoIs.

INTENDED USERS AND USES

Users:

- Tourists
 - Key characteristics:
 - They care about the locations they visit while on a trip.
 - They want to have new experiences.
 - Needs:
 - Easy to use interface.
 - Quick loading times.
 - Best route through their chosen interests.
 - A diverse route through different interests.
 - How they benefit:
 - Our application will give them the best opportunity to easily find the best route that fits their needs.
- Educator
 - Key characteristics:
 - They care about the efficiency of the application.
 - They care about the clear visibility of the algorithm's work.
 - Needs:
 - Need to be able to easily switch between routing algorithms.
 - Need an easy to use but professional interface.
 - Need the application to not slow down the algorithms for accurate comparisons between them.
 - How they benefit:

- Our application will allow educators to directly compare multiple semantically diverse path-finding algorithms through points of interest. This will help them visually compare the algorithms for testing which is best.
- Business Professional
 - Key characteristics:
 - They care about the distance and time between points of interest.
 - Needs:
 - Easy to use interface.
 - Quick loading times.
 - How they benefit:
 - Our application will give them the best opportunity to visit points of interest within their schedule.

FUNCTIONAL REQUIREMENTS

- The application should be able to display the determined route that fulfills the input of the user.
 - In case there are multiple routes, the application should clearly indicate them.
- The user should be able to easily input, view, and modify the current choices for points of interests and starting locations.
- Stretch goal: Incorporate other attributes in ranking the selections i.e., rank hotels by price or stars; rank museums by entry fee, etc.
- The application should allow the user to specify which algorithm they want their route to be determined by.
- The application should be scalable i.e., allow multiple users to use it.

NON-FUNCTIONAL REQUIREMENTS

- Design of the code should be extendable for easy addition of more areas/cities.
- Application should be able to reliably perform the task of input and creating routes.
- The user experience should be intuitive and clean.
- The code implementing every functionality should be designed with maintainability and extensibility in mind. For example,
 - UI components should be added over time
 - Data for additional cities should be easily adoptable

STANDARDS

IEEE Std 1012, Standard for Software Verification and Validation

- This standard will help this project to meet and or exceed the expectations and requirements of users and their needs.

IEEE Std 1219, Standard for Software Maintenance

- This standard will help this project to allow future developers to be able to easily maintain this software.

IEEE/ISO/IEC 26512-2017, Requirements for acquirers and suppliers of information for users

- This standard will help this project to improve users quality and availability of information about the software.

IEEE Std 982.1, Standard Dictionary of Measures to Produce Reliable Software

- This standard will help this project to create reliable and dependable software.

IEEE/ISO/IEC 15288-2015, System life cycle processes

- This standard will aid us with definitions for the entire life cycle of the system, i.e., conception, development, production, utilization, support, and retirement.

ACM 1.3

- This standard will aid this project developing software more ethically.

ENGINEERING CONSTRAINTS

- Route should be fully displayed within seconds after input submission.
- Route visualization should be accurate.
- Total cost of implementation should not exceed \$300.
- Time: 2 semesters

SECURITY AND COUNTERMEASURES

Security for the application is outside of the scope of the project. The project is not intended to be a fully fledged commercial application, the application is intended solely to be a demonstration application for our client. Additionally, the application does not handle any user specific data such as usernames or passwords.

DESIGN EVOLUTION SINCE 491

Frontend Design Changes

1. *Changing our planned Generated Route window to be incorporated into the Hotels list.* When we arrived at this stage in the frontend implementation, what we had thought our design would look like had, for the most part, stayed the same. However, after working on the application and using it like a user would, the frontend team decided it would be

best to move away from making the user switch between windows to interact with the hotels. Thus, we implemented the textual route visualization into our list of hotels instead of displaying it separately. This allows the user to easily click between hotels and routes without having to switch between two windows. The change added complexity, but in the end we believe it makes the flow of the application smoother.

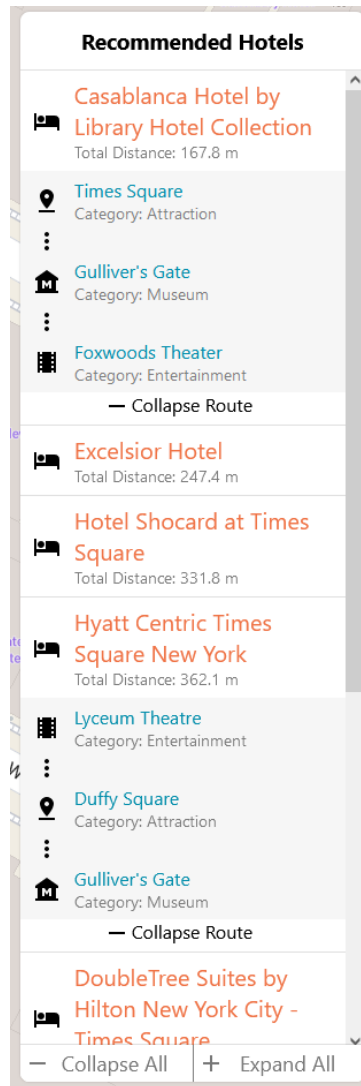
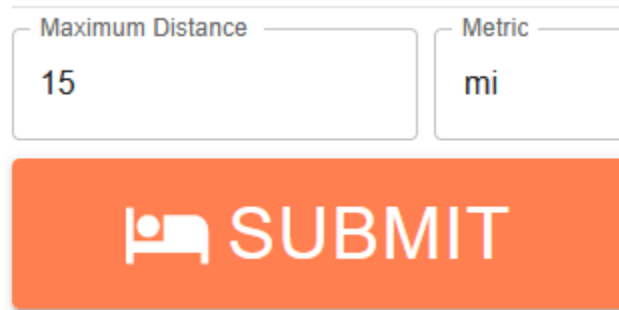


Figure 1: Improved Recommended Hotel List

2. Adding the ability to choose distance measurement.

During a meeting with our client, while we were showing off our frontend user interface, we showed off the distance input field where it specified meters. Our client had the idea of letting the user switch between different units of measurement to fit the individual user better. So, now the frontend converts the user's input into meters before we send the

processing request to the backend. This was a really good idea and a nice feature, so we implemented the change.

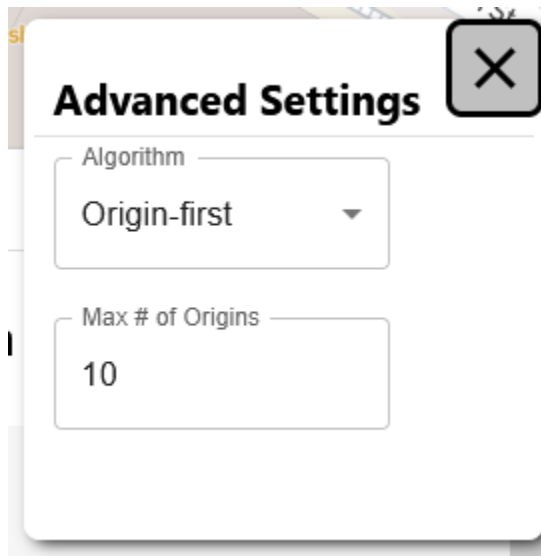


The image shows a form with two input fields. The first field is labeled "Maximum Distance" and contains the value "15". The second field is labeled "Metric" and contains the value "mi". Below these fields is a large orange button with a white icon of a person at a computer and the text "SUBMIT".

Figure 2: Distance Metric Selector

3. *Adding the ability to specify the desired number of origins.*

While working on our project, the backend team discovered that the maximum number of route origins was an input to our client's algorithm. With that discovery, we decided instead of making that value static, we should allow the user to specify how many origins they want generated. Then, the frontend team changed our design to incorporate this new field.



The image shows a dialog box titled "Advanced Settings" with a close button (X) in the top right corner. Inside the dialog, there are two input fields. The first is a dropdown menu labeled "Algorithm" with the value "Origin-first" selected. The second is a text input field labeled "Max # of Origins" with the value "10" entered.

Figure 3: Number of Origins Input

Backend Design Changes

1. *Migrating to Flask.*

When we had originally designed our backend implementation, we had planned on using Spring Boot since we were all familiar with Java and Spring Boot from classes like ComS 309. Our solution to implementing the algorithms would be to use a Python driver that our Java code called, however this proved to be quite a difficult task. It would introduce a lot of latency and compatibility issues, as well as take a lot of time to implement. These factors, combined with the backend's knowledge of Python, led us to consider using Flask as an alternative. We started with a simple implementation, and Flask proved to be easy enough for us to use, so we made the full switch.

IMPLEMENTATION DETAILS

Goals

For our project we wanted to be able to create an application that is fast and efficient, easily used, and can be easily expanded upon for future city datasets.

Tools Used

- Frontend
 - TypeScript
 - React
 - Redux
 - MapBox
- Backend
 - Python
 - Flask
 - Apache HTTP Web Server

Frontend

For our implementation of the frontend, we decided to use TypeScript as the main language, as some of the members in our group already knew how to use it, as it's very similar to JavaScript. We used React as the frontend framework to design reusable components that we can use throughout the application. We believe that our choice of TypeScript and React helps to make our application fulfill its needs of code readability, maintainability, and extensibility.

We used Redux with React for the state management in our application. Multiple states are held within the application. The larger state store being the one for all of the user input for the

application. The other being the map, its functions and its data, so different components of the application can control the map and the map can control them. Having Redux for our state management has helped make it easier to modify, access, and store our application's state which helps our application maintain readability and extensibility.

Finally, we decided to use MapBox as our visualization tool to show routes generated from our routing algorithm on the backend. MapBox has an extremely robust API and is very customizable, so we were able to fit it to our application's needs. MapBox also has many out-of-the-box features that fit the usability and visuals we wanted our application to include. The use of MapBox allowed us to create the application which fulfilled our user's needs for controllability and map readability.

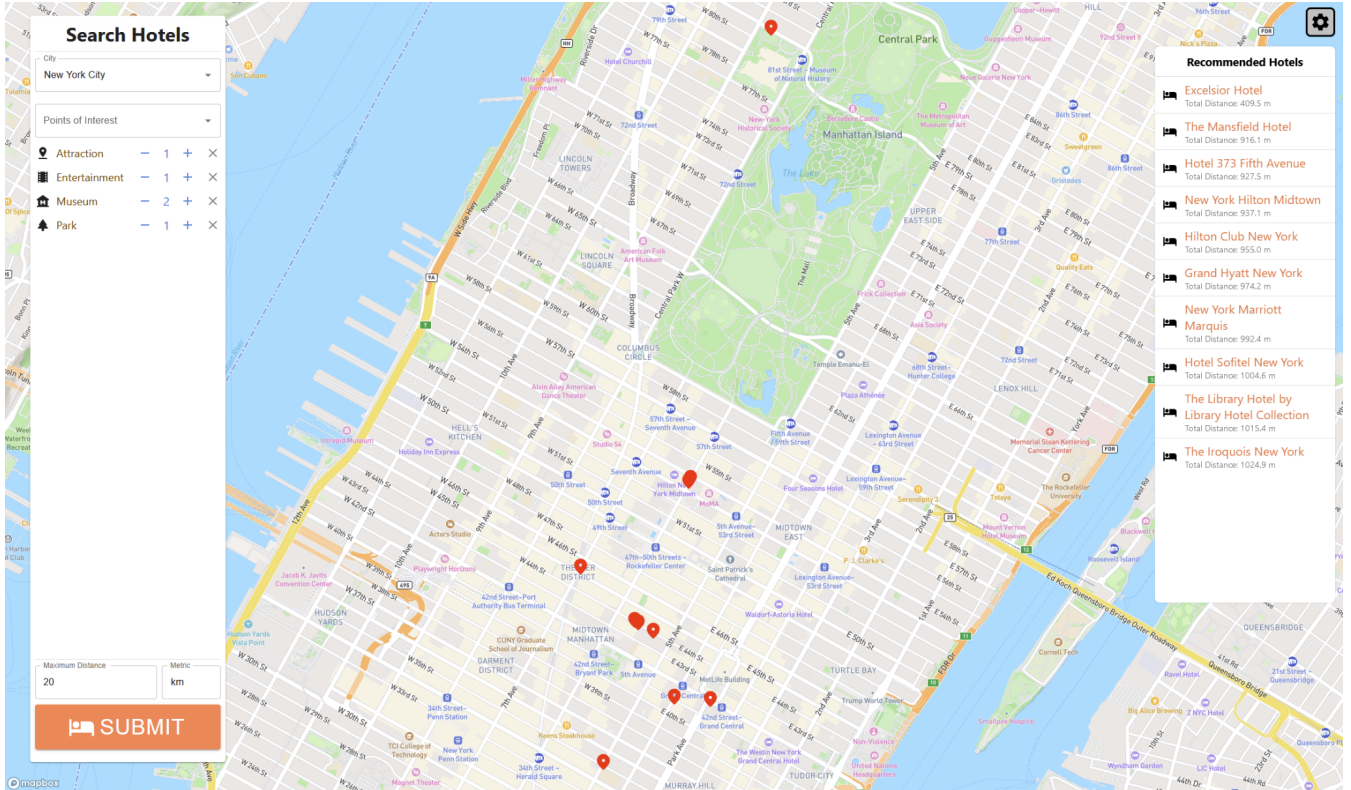


Figure 4: Example Generation of Recommended Hotels

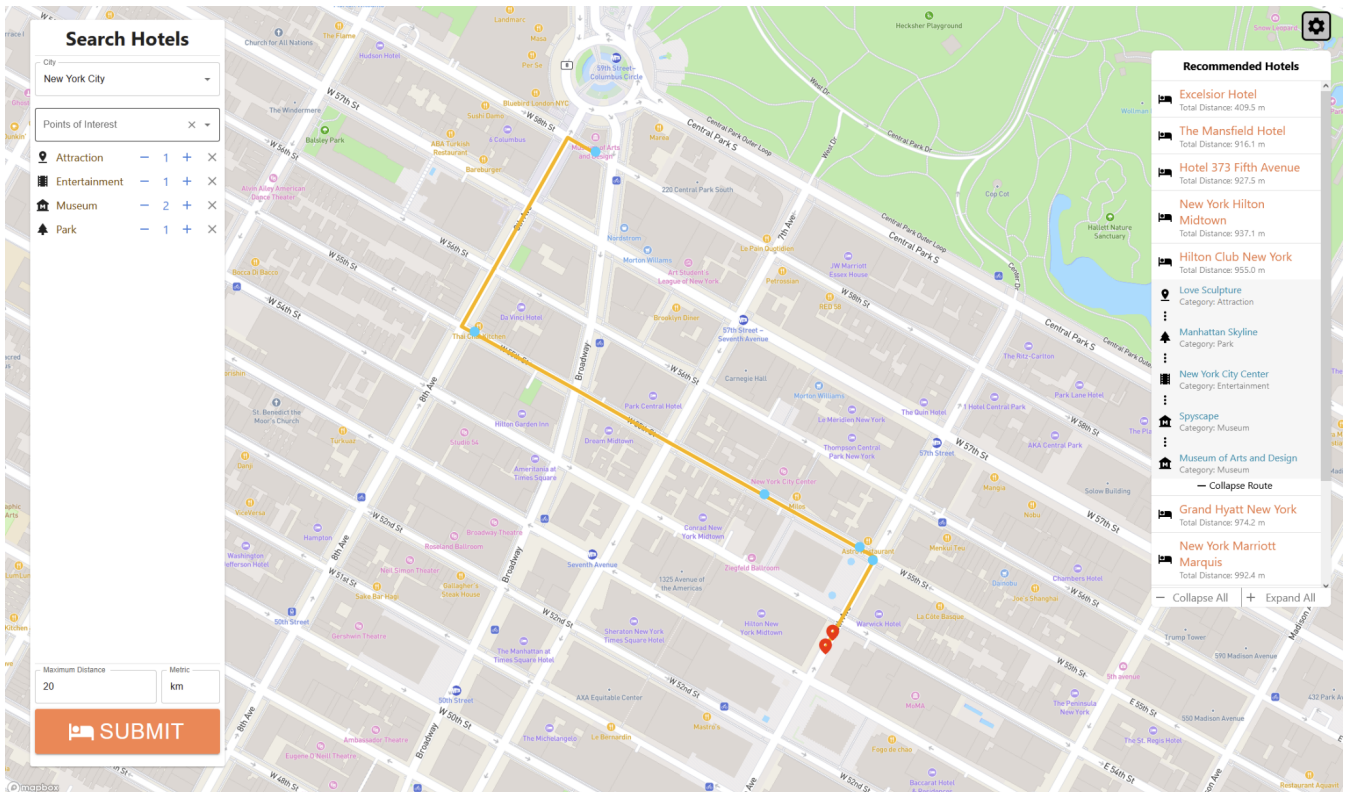


Figure 5: Example of Generated Route for a Recommended Hotel

Backend

For our implementation of the backend, we decided to use Python as our primary programming language. We evaluated various programming languages and frameworks, such as Java and Spring Boot, but ultimately went with Python because our client's route generation code was already implemented in Python. By utilizing the same language as the client's code, it allowed for a seamless integration between our code and theirs.

In order to develop our REST API, we chose Flask as our web framework. Flask is a lightweight and flexible framework that allows developers to create web applications quickly and easily. Unlike other frameworks, like Django, Flask is less opinionated, which means developers have more control over their application's design and architecture. Additionally, Flask is highly customizable and provides the necessary features needed for our project, making it ideal.

Finally, to deploy our application, we utilized Apache HTTP Server on our virtual machine. Apache HTTP Server is a widely used web server that provides a robust and scalable platform for hosting web applications. To enable Apache to interact with Flask, we set up WSGI (Web Server Gateway Interface) as our middleware. By setting up WSGI as our middleware, we enabled Apache to communicate with Flask and forward requests to it.

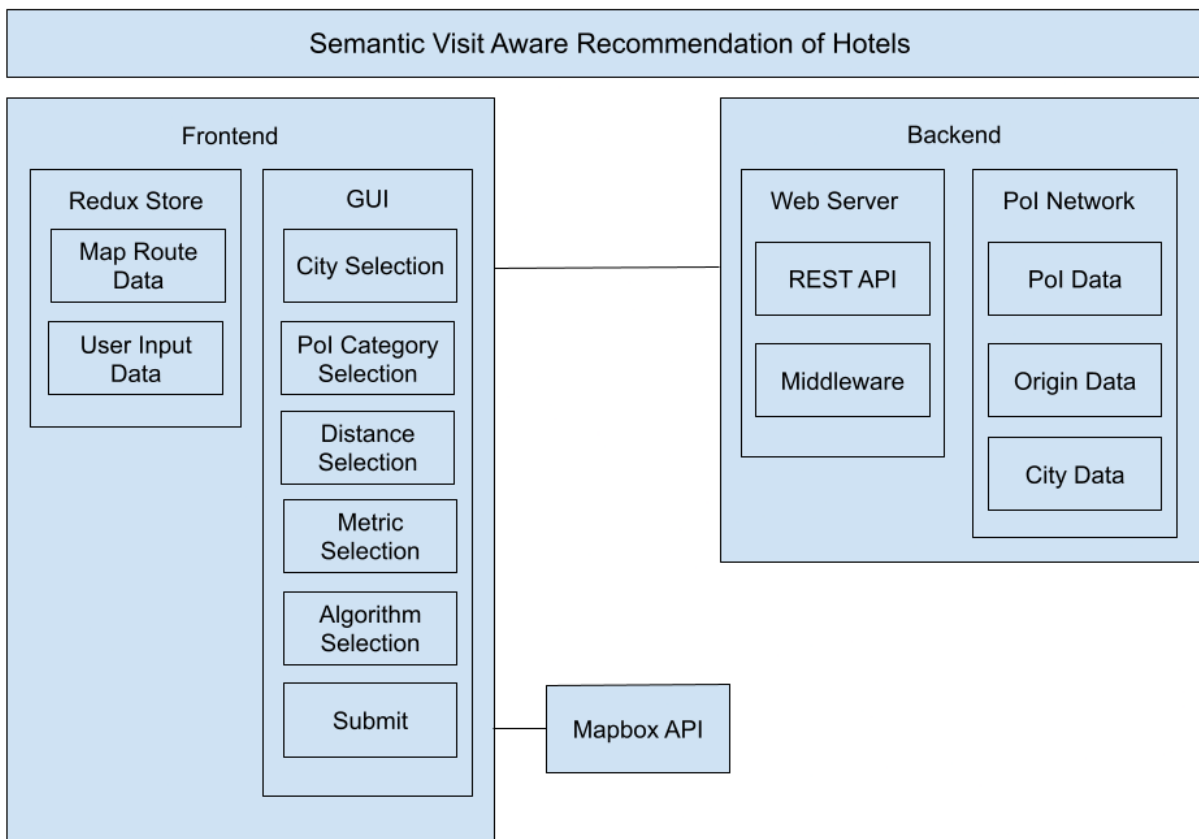


Figure 6: Implementation Architecture

TESTING PROCESS

Our main testing strategy was to test early and often and to also ensure that throughout the process of adding new code to our project that our application was still fully functioning correctly and running as it should. To achieve this we implemented CI/CD into our git repository very early in the process when we had very few features implemented in the application. This allowed every change that a person made to our project to be tested and make sure that change didn't break previously functional features of the application. As the frontend and backend continued to add more code to the application, tests were added continuously to cover as much of that code as possible using Jest for unit testing on the frontend and Pytest for unit testing on the backend. To truly make sure as much of our code was being tested as we could, we looked at code coverage reports which told us what percentage of our code was being covered by our tests, allowing us to write more tests to reach a higher percentage and test our application fully.

The non-functional requirements of our project were primarily tested through demonstrations of the implementation to our client throughout the development process. In these meetings we got feedback from our client regarding his requirements and any improvements to our implementation.

TESTING RESULTS

Our testing results were acceptable throughout the process of writing our application, when changes were made that either broke our application or didn't produce the results we wanted them to, these issues were easily addressed right away because we were testing throughout the entire process. Also, because we made sure to test our application so thoroughly by continuously adding tests and looking at code coverage, our application has become reliable and runs smoothly most, if not all of the time.

RELATED PRODUCTS AND LITERATURE

With the rise of smartphones over the last couple decades, an increasing number of users have begun to use keyword-based search for content that is often related to their geographic location. This has sparked research focused on integrating location into keyword-based querying of geo-textual content (Chen et al., 2021). An example of such research is from a paper titled, KORS: Keyword-aware Optimal Route Search System, which outlines an algorithm to answer queries that find routes such that they cover a set of user-specified keywords, a specified budget constraint is satisfied, and an objective score of the routes is optimized (Cao et al.). The shortcoming of this approach is that it generates a route that brings the user to the specified keywords, but does not take into account the diversity of these keywords/PoIs.

There are many products on the market today that visualize routes between destinations and provide nearby PoIs for people to visit. Some of the more notable products in this area include Google Maps and Roadtrippers. Google Maps is known for its efficient and detailed route planning that allows users to see things like travel time, different routes, and nearby PoIs that the user may like based on their data. Roadtrippers is well known for their travel guides where they provide PoIs that are within a given range along the route the user is taking as well as hotels for the user to stay at along the way.

Neither of the two solutions in the previous paragraph take into account PoI diversity when generating routes for the users to follow. This may lead to routes or travel guides that do not allow users to really see what the area they are visiting has to offer. Our solution fixes this problem by creating routes that maximize the PoI diversity based on constraints given by the user. In addition to this, our solution has the following pros and cons as compared to the other products:

Pros

- Route maximizes PoI diversity based on given constraints
- Specify certain PoI categories to visit
- Does not store user information

Cons

- Limited to routes in NYC, Chicago
- Limited to starting from a known location

While our solution had initial cons going into the project, we were able to surpass and create solutions to them. Initially, our route dataset only included NYC however our team was able to gather new data to allow us to add Chicago to our available cities for route generation. Our team is confident that this process can be readily extended to include many other cities. In addition, the datasets can also be updated to account for including more known locations when data of known locations are accessible.

APPENDIX I - OPERATION MANUAL

Introduction

The following information provides step-by-step instructions on how to set up, demo, and test our system. The commands shown are for a Linux system and may need to be modified for other operating systems. The requirements needed to begin the setup are to have Git, Python, and Node installed. If you do not have them installed, you can do so with the following links:

- [Git Download](#)
- [Python Download](#)
- [Node Download](#)

You can verify that you have them installed with the commands.

```
$ git --version
$ python3 --version
$ node --version
```

Download Project

To begin, you need to clone the repository or download a zip of the project to your local machine. You can clone the repository with the following command, or you can download a zip of the project [here](#). If you downloaded the zip file, you will need to unzip it to get the project contents. This will create a directory called sdmay23-34 containing our project.

```
$ git clone https://git.ece.iastate.edu/sd/sdmay23-34.git
```

Backend Setup

We will start with setting up the backend. This will include installing Python 3.7 and creating a virtual environment that will contain all the needed dependencies.

First, change to the directory where you want your virtual environment to be stored. In my case, it will be in the Documents/environments directory for my user.

```
$ cd /home/user/Documents/environments
```

Next, install virtualenv which is used to create a virtual environment.

```
$ pip3 install virtualenv
```

Then you will install Python 3.7. You will need to enter ‘y’ and hit ‘Enter’ at times to continue the installation. (Note: the project does seem to work fine with newer Python versions, but to create the environment we are running on our VM, do the following)

```
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt install python3.7
```

With Python 3.7 and virtualenv installed, we can create a virtual environment that will be using Python 3.7. The following will create a virtual environment called python37.

```
$ python3 -m virtualenv --python="/usr/bin/python3.7" python37
```

Now we can activate the virtual environment to use it, and verify that the Python version is 3.7.x.

```
$ source python37/bin/activate
(python37) $ python3 --version
```

Once completed, you will see the virtual environment name in your console. This virtual environment will allow us to install all the needed packages for the project in an isolated manner, which we will do now.

Change your directory to the Backend folder of the project you cloned earlier.

```
(python37) $ cd /home/user/Documents/sdmay23-34/Backend
```

Install all the needed requirements with pip3.

```
(python37) $ pip3 install -r requirements.txt
```

If you run into errors with gdal, rtree, or another package, try the following commands and then rerun the previous command.

```
(python37) $ sudo apt install libspatialindex-dev
(python37) $ sudo apt install libpython3.7-dev
```

Once all the packages have been installed correctly, you are ready to start up the backend! The following command will start a development server that can be accessed at <http://localhost:5000> and will be used later when running the frontend.

```
(python37) $ python3 app.py
```

You should see text like “* Running on http://127.0.0.1:5000” in your console if things are working correctly. You may see warnings, but those can be ignored as they are telling you that what is being run is a development server, which is fine for demos. Congratulations, you have the backend running!

If you want to deactivate your virtual environment, you can run the following command.

```
(python37) $ deactivate
```

Frontend Setup

Before we begin, make sure you have the backend running by following the steps in “Backend Setup”. The frontend will need the development server running to function properly. To make things easier, you can have one terminal running the backend, and another running the frontend. With the backend running, we can now begin!

First, change your directory to the Frontend/svarh directory in the project you cloned earlier, and install the node modules.

```
$ cd /home/user/Documents/sdmay23-34/Frontend/svarh  
$ npm install
```

With the node modules installed, we can now start up the frontend. Running the following command will open a browser window with our project.

```
$ npm run start
```

If everything is running correctly, you should be able to interact with our application and see the recommended hotels, based on generated routes, that bring the user to the requested points of interest!

Demoing the Project

If you followed the “Backend Setup” and “Frontend Setup”, you will have reached a point where you could use/demo the application. If you have not gone through those setup sections please do so as this section will walk you through how to use the application.

On the left side of the application, you will see a city selection dropdown, point of interest dropdown, distance constraint, and submit button. The city selection is for the user to decide which city they want to be recommended hotels for. The point of interest dropdown allows the user to select how many PoIs in each category they want to visit. The distance constraint allows the user to specify the max distance they are willing to travel. Once all of these sections have been populated, the user can click the submit button to see the recommended hotels and routes from these hotels to the requested PoIs.

The hotels will be marked by pin drops on the map, and the PoIs will be marked by cyan circles. Once a hotel is clicked, an orange route will be displayed which brings the user to their requested PoIs within the distance specified. Information on the hotels, PoIs, and routes such as names, categories, and distance can be found in the “Recommended Hotel” panel on the right.

For the more advanced users, there is a settings cog in the top right that allows the user to specify the algorithm used to generate the routes as well as the number of hotels to be visualized. The defaults for these are Origin-first for the algorithm, and 10 for the number of hotels to be visualized.

Testing the Project

If you have not done the “Backend Setup” and “Frontend Setup” sections, please do so before starting this section.

To run the backend test cases, you will need to change your directory to the Backend folder of the cloned project, activate your virtual environment, and then run a command to start the tests.

```
$ cd /home/user/Documents/sdmay23-34/Backend
$ source /home/user/Documents/environments/python37/bin/activate
(python37) $ python3 -m pytest -W ignore::DeprecationWarning
```

You should see that 22 test cases passed.

To run the frontend test cases, you will need to change your directory to the svarh folder within the Frontend directory. After changing your directory to svarh, simply run `npm test` to run the Frontend tests.

```
$ cd /home/user/Documents/sdmay23-34/Frontend/svarh
$ npm test
```

APPENDIX II - ALTERNATIVE VERSIONS OF THE DESIGN

Frontend Design Alternatives

Version 1:

Semantic Visit Aware Recommendation of Hotels

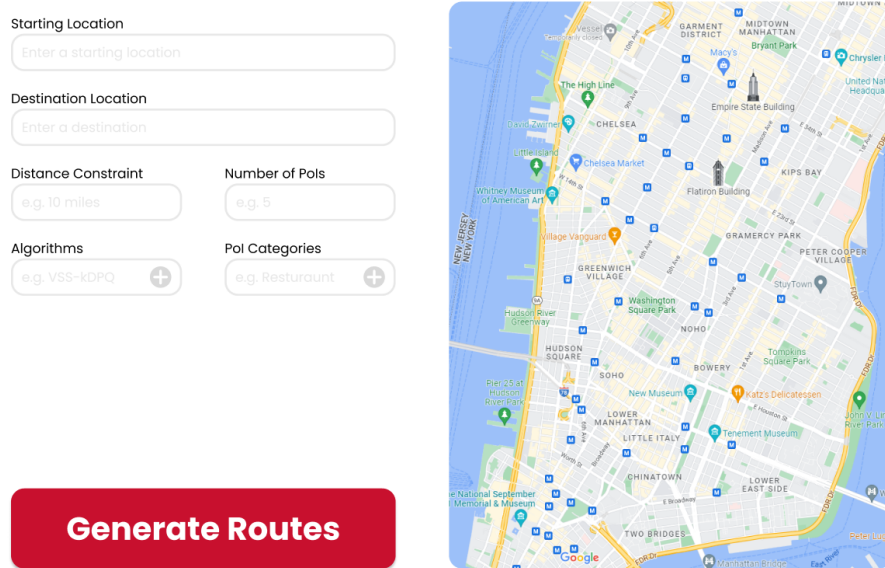


Figure 7: First Alternative Frontend Design

This version of the frontend was designed to get the required functionality and inputs onto the screen. It was important for us to get the base user inputs and outputs settled before moving onto a more complex design.

Version 2 (on Figma):

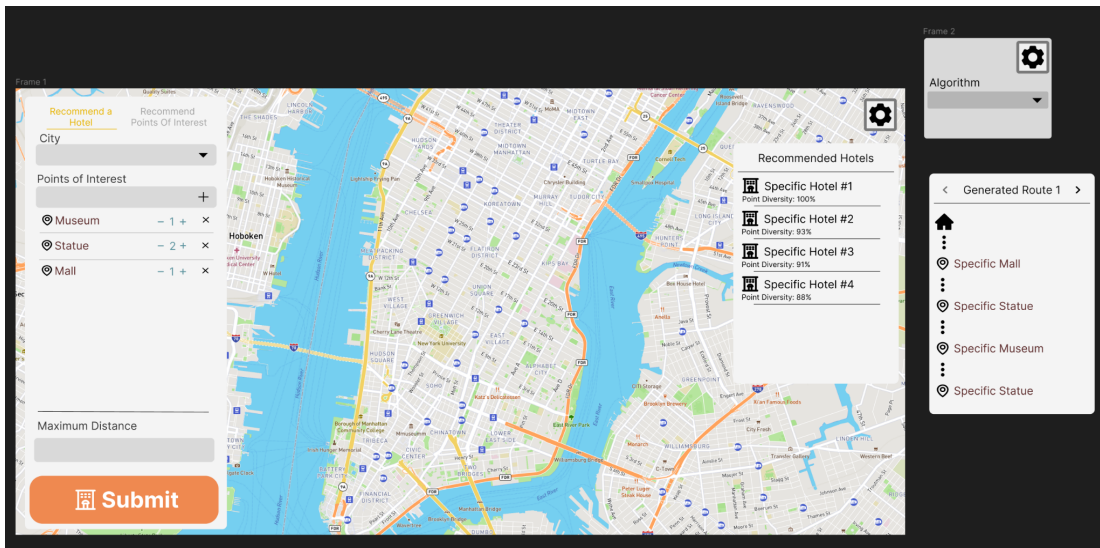


Figure 8: Second Alternative Frontend Design

This version of the frontend was used to add more complex features to the design to improve the user experience. The design tries to implement a path for the user from the left side to the right side of the screen. The user inputs their needs on the left, views the route in the middle and can see extra information/controls on the right side of the screen. While this was the base of the final design, as needs and ideas evolved, the design changed.

REFERENCES

- Teng, Trajcevski, & Züfle. (2021). Searching Semantically Diverse Paths.
ACM Code of Ethics and Professional Conduct. (n.d.). Association for Computing Machinery.
<https://www.acm.org/code-of-ethics>
- Chen, Z., Chen, L., Cong, G. et al. Location- and keyword-based querying of geo-textual data: a survey. *The VLDB Journal* 30, 603–640 (2021).
<https://doi.org/10.1007/s00778-021-00661-w>
- X. Cao, L. Chen, G. Cong, J. Guan, N. -T. Phan and X. Xiao, "KORS: Keyword-aware Optimal Route Search System," 2013 IEEE 29th International Conference on Data Engineering (ICDE), 2013, pp. 1340-1343, doi: 10.1109/ICDE.2013.6544939.
- Code of ethics: IEEE Computer Society. Code of Ethics | IEEE Computer Society. (n.d.). Retrieved November 28, 2022, from <https://www.computer.org/education/code-of-ethics>