UberCat - *CS2021 Final Project*

Link | **GitHub**

Authors - Rishabh Ravindra and Ananya Nijhawan

Introduction

Overview

UberCat is an online Uber fare estimator for students at UC. The idea was inspired by Uber's fare estimator available online. Our project is a UC specific version to help students estimate fares before they go out for the night. Our implementation specifically focuses on students living in the UC residence halls. It is a Python/Flask application powered by UBER's and Google Maps' Python APIs. The project has been deployed on the cloud hosting platform, Heroku.

Sample Rundown:

- Enter your starting address
- Select your dorm
- Get price estimates for all types of Ubers

API/Framework Selection Process

The first choice we made was to use Uber's Python API to get price estimates. This was a no-brainer as Uber officially supported the Python API.

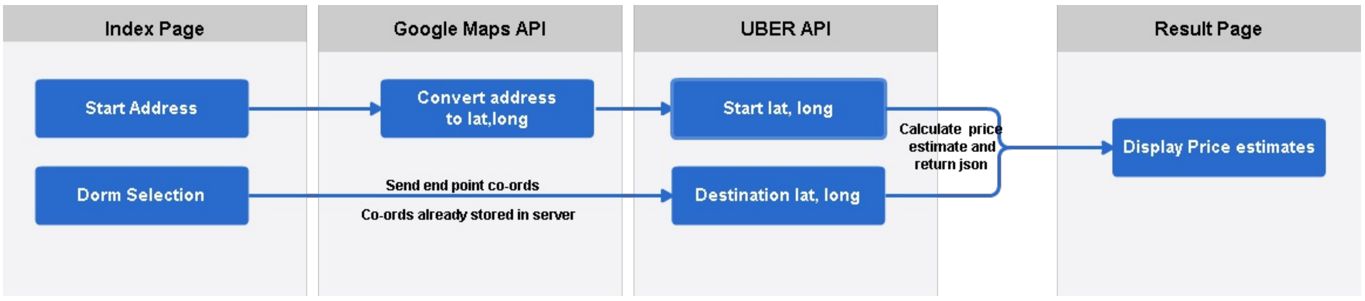
The next part was to decide where to get the co-ordinates from, since the Uber API uses co-ordinates to define its starting and ending points. Google Maps API's python client was used as both team members had prior experience with the API.

The last major decision was to choose what web framework to use to get information: Django or Flask? We settled with Flask because of its simplicity and unopinionated nature.

Process breakdown

The user is welcomed with a form that takes the starting address as the first input, and the dorm name as the secondary one. A **POST** request is sent to the server when the user clicks on the submit button. The Maps API geocodes the address given to co-ordinates. There is a dictionary defined in the app.py file which has co-ordinates of all dormitories. The server matches the user input with the dictionaries. The co-ordinates of the user starting point and dorm are then sent to the Uber API.

The API calculates price estimates of all types of Uber's available and returns a JSON file with the details. The json is stored in a variable and is sent to render the results page. In the render HTML page, there is a loop set up to render an info card with trip details about each Uber type.



Sample flowchart created during ideation

Project Results

Thoughts and challenges

The program consisted of three main parts:

- requesting data from Uber and Google Maps,
- getting user input and rendering output, and
- deploying on a cloud based server.

The first part was handled by using the respective APIs and getting back data in JSON format. The second part was handled using Flask which sent data back and forth to the server and rendered the form and output pages. The last part was handled by Heroku, a cloud hosting platform. Gunicorn, a Python HTTP server, was installed to run a python app instance on Heroku.

During the whole program design, we had to make sure that the program code was good at handling different APIs and avoiding any conflicts. As can be seen from the flowchart, a lot of processing goes between the two html pages rendered. We used a synchronous platform to accomplish the middle tasks.

For example, we had to make sure the staring address string was first converted to co-ordinates by the Maps API before we sent the co-ordinates to the Uber API. This all had to be done in a sequential pattern, else the UBER API would not be able to calculate the price estimate - a feature integral to the program.

Also, we improvised on the front-end. We had a basic html form input and a basic output render page. However, we did not like how the app looked. So we used UIKit, a HTML and CSS library with built-in web components for use.

Future improvements

One improvement that immediately comes into mind is integrating Lyft’s API to compare the prices. This feature would actually yield more value to the application as users always have to switch between the two apps to get the price estimates. We did not implement this feature in the current project because Lyft does not have an official Python client currently and does not provide ride estimates through its API.

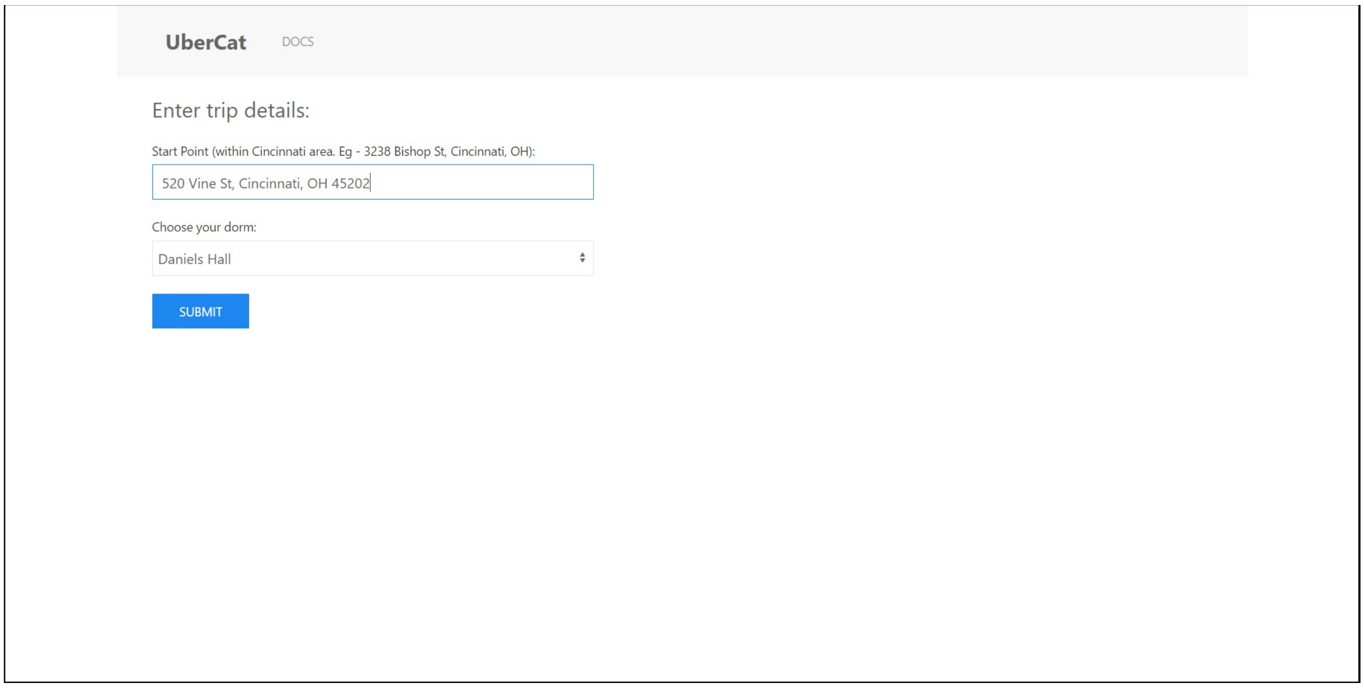
To improve the user experience, we can also embed a Google map to the results page which shows the route from the starting point to the destination. This would give the user a visualization of their trip and allow them to update their trips, if needed. This brings us to another future feature which is booking the ride from the app itself. Using

Oauth2, we can authenticate the user and let the user ride a cab right after knowing the price estimates.

Another improvement is to update the file structure. Currently, the files are all in one folder. Since the app involves a Model-View-Controller structure, the folders should reflect it. This would help any future developers working on it understand the code better.

Screenshots

Get input from user



Enter trip details:

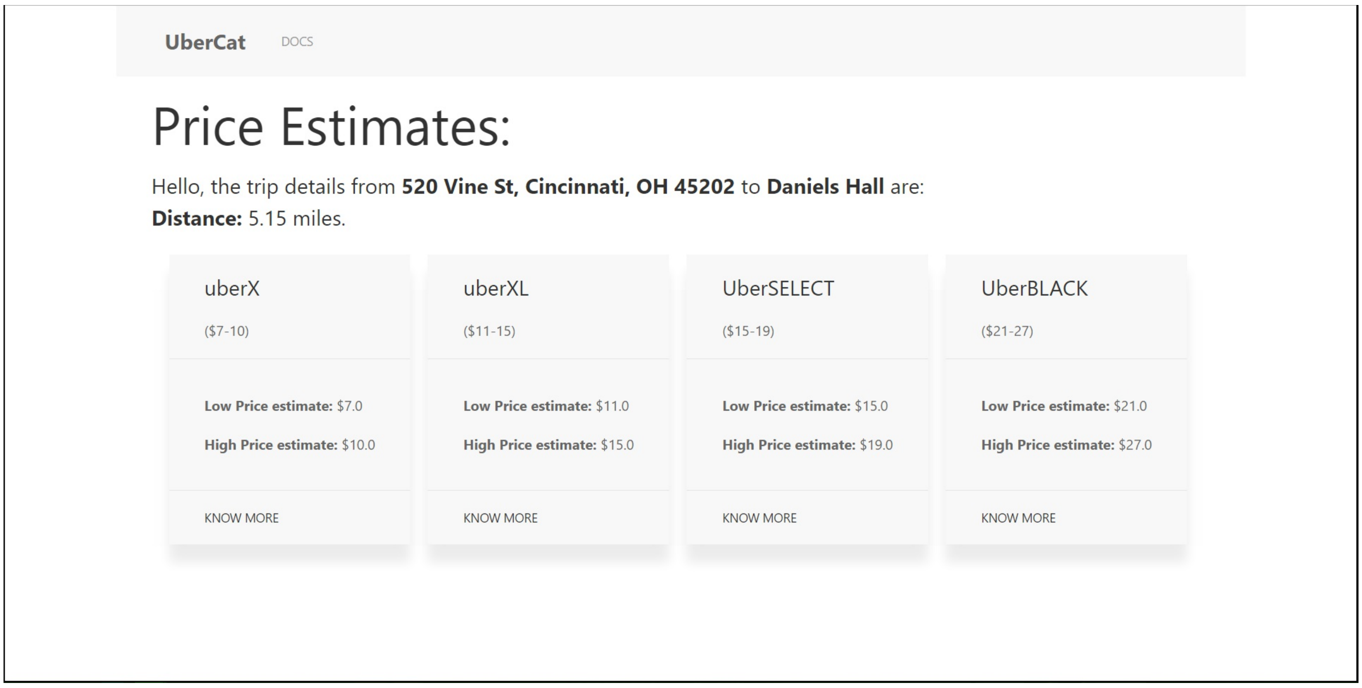
Start Point (within Cincinnati area. Eg - 3238 Bishop St, Cincinnati, OH):

520 Vine St, Cincinnati, OH 45202

Choose your dorm:

Daniels Hall

Results page with available fare rates for all types of UBERs running in Cincinnati


UberCat DOCS

Price Estimates:

Hello, the trip details from **520 Vine St, Cincinnati, OH 45202** to **Daniels Hall** are:
Distance: 5.15 miles.

uberX
($7-10)
Low Price estimate: $7.0
High Price estimate: $10.0
KNOW MORE

uberXL
($11-15)
Low Price estimate: $11.0
High Price estimate: $15.0
KNOW MORE

UberSELECT
($15-19)
Low Price estimate: $15.0
High Price estimate: $19.0
KNOW MORE

UberBLACK
($21-27)
Low Price estimate: $21.0
High Price estimate: $27.0
KNOW MORE

Division of Group Work

- **Ananya Nijhawan**

*My primary contribution was in setting up the Flask interface for taking in the input from user and converting the input into a form that could be accessed by Uber's API. I also handled the routing for the web app. So from creating a **GET** request to display the form to the **POST** request which passed on the information to the results page, I managed it. I found the Flask documentation very useful and there was a lot of support by other developers on online platforms such as Stack Overflow.*

Another feature that I worked on was geocoding. Initially, our input form took the starting point latitudes and longitudes to set the initial co-ordinates. I felt that this was not user-friendly at all. So to improve the user experience, I suggested we include the python client for

Google Maps and use geocoding to convert the address string to a pair of co-ordinates. This improved the user experience and now I feel more confident in trying new technologies - something I was always skeptical of.

This project, just as much it was about python, was about going to uncharted territories. I had previously worked briefly with the Google Maps Javascript API but had no other experience with working with 3-rd party APIS. Also, I had not touched Flask before nor did I know about html forms and routing data. After the project, though, I feel more confident in my skills in Flask and Maps API skills, and more importantly, trying new technologies.

- **Rishabh Ravindra**

The project was a great learning experience and a good opportunity to delve into the world of Python. I worked on creating a client for Uber's API and sending and getting information from it. My teammate, Ananya, used Google Maps API to geocode (convert addres to co-ordinates) the starting address provided by the user. I then took that trip data and called the '''get_price''' method to get details.
The API responded back with a JSON object which had all the details of the trip. My next task was to extract the required data (the Uber type and prices) and send it to Flask to render it in the results.

Another major responsibility I undertook was to deploy the python app to the cloud hosting platform, Heroku. Doing so gave the project a near-completion status as not only did we get price estimates but also *set up an online presence for others to use it. I had to read Heroku's documentation closely to understand how apps are deployed on its platform and how we could set our python app up. I installed Gunicorn that started a Python HTTP server. Extra configuration files were added to authorize the app and run proper commands to make the app working online.*

One key thing that I learned through the project, more than Python itself, was reading the documentation of a 3rd party API/framework. Heroku had a really confusing set of documented instructions, and I had to comb through a lot of lines of code to get to what I was looking for. While using another organization's code, it is important to be well versed with the restrictions and how-to's of an API. And that is why it is important to know how to read an APIs documentation and make good use of it.

Bibliography

A number of online resources and articles were instrumental in bringing the idea into fruition:

- Handle a POST Request In Flask
- Uber Rides Python API
- Python Client for Google Maps Services
- Deploying a simple Flask app to the cloud via Heroku

Links to API/Frameworks/Libraries

used

UberCat was possible because of the generous open-source/public libraries on the web.

API/Framework/Library	Description	README
Flask	Web framework for Python	http://flask.pocoo.org/
Google Maps API python	Python client for Google Maps API	https://github.com/googlema maps-services-python
Gunicorn	HTTP server for Python	http://gunicorn.org/
Uber Rides API	Uber rides SDK for Python	https://github.com/uber/rides sdk
UIkit	Front-end framework to quickly create web components	https://getuikit.com/docs/intr

Code Appendix

GitHub

Website