Uber/Lyft Dynamic Pricing Application

Easy Cabs

Introduction

Easy Cabs assists you in obtaining dynamic Uber and Lyft cab prices. When you enter your location, Easy Cabs will take into account your latitude, longitude, and current weather forecast to estimate surge pricing for your rides on both platforms. The end user can then choose which cab is suitable for them in terms of cost and convenience.

Machine learning models are developed to compute the surge multiplier and cab fare between two different locations in Boston. The user specifies the origin and destination, as well as the cab type for Uber and Lyft, and then clicks the "Calculate Price" option.

The user will be prompted with the pricing of an Uber or Lyft vehicle in USD, as well as the estimated arrival time in minutes and distance in miles. EasyCabs considers these elements and attempts to mimic a generic form of these apps' pricing and surging microservices. As a separate microservice, it also allows you to retrain models utilizing a feedback loop.

Functional specification

Our ML-based solution with surge solves this problem by providing a one-stop experience for evaluating pricing for various sorts of cabs. By taking into account rush hours, weather fluctuations, and location, the surge calculation duplicates Uber's microservice for surge calculation. The surge, distance, taxi type source, and destination location data are used in the dynamic pricing ML model to generate the final price. Our goal is to recommend an appropriate Uber or Lyft cab type based on the user's budget and requirements.

User profile: People who want to compare and book cabs at an optimal cost.
Data sources: Data Set from Kaggle which has over 693,000 records of data for weather and cab-rides.

Use cases.

Choosing a cab from Uber and Lyft as per: -

a) cost
b) convenience

i. USER: Enters source and destination.
ii. USER: Provides similar cab types for Uber and Lyft.
iii. PROGRAM: Runs the model for surge prediction.
iv. PROGRAM: Runs the model for dynamic price calculation using the surge prediction model output.
v. OUTPUT: Outputs the price for Uber, Lyft respective cab types and ETA and distance between the source and destination.

Component specification

Component: API(getCabPrice)
Input source, destination, and type of cabs.
The component then calls the following subcomponents to get the required values:

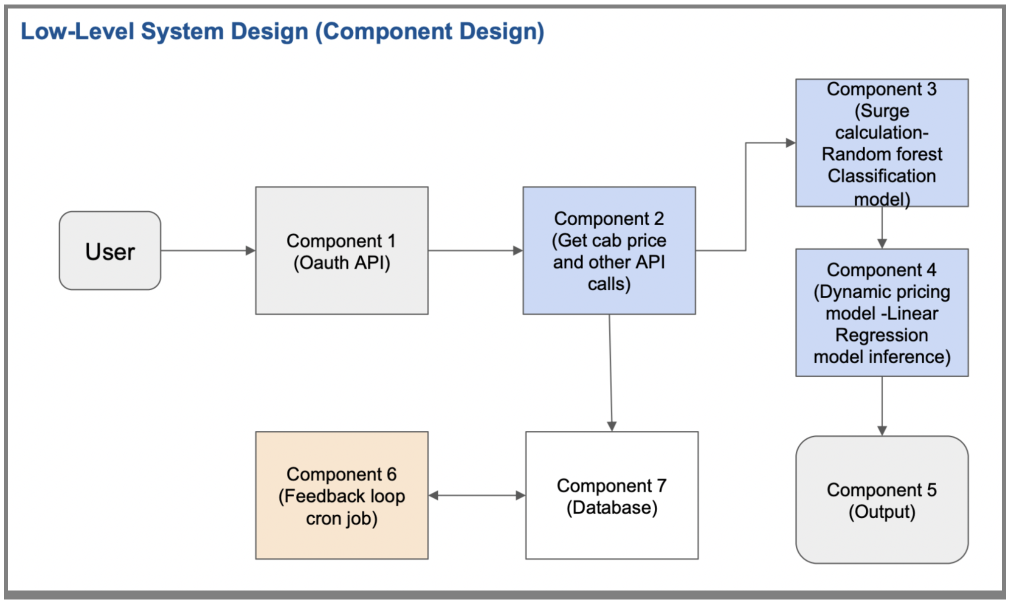
1. Geospatial Data: Input: Source and Destination; Output: Latitude and Longitude, ETA and Distance.
2. Weathermap API: Input: Source Latitude and Longitude; Output: Required weather parameters.
3. Uber API: Input: source and destination; Output: dynamic Price and surge of the required cab *.
4. Lyft API: Input: source and destination; Output: dynamic Price and surge of the required cab *

Component: class(surge_inference)

Input weather parameters obtained in getCabPrice Component. Run the classification model to calculate surge. Return (surge_price)

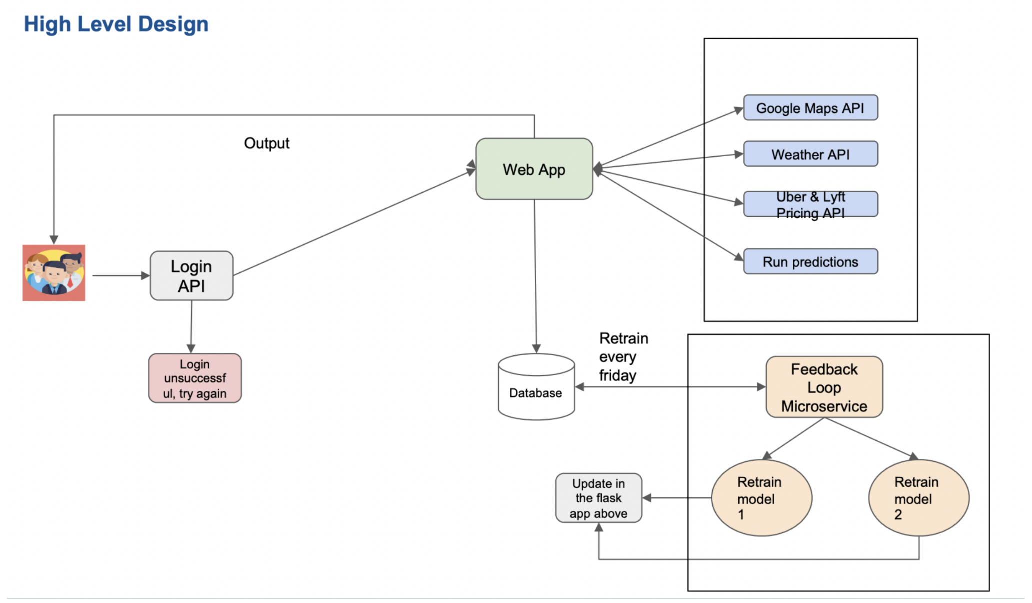
Component: class(dynamic_price_inference)

Input surge, distance, source, and destination geospatial information
Runs the data manipulation and the regression model to calculate dynamic price for Uber and Lyft individually.
Return (Price for Uber and Lyft selected cab prices)


Low-Level System Design (Component Design)
User
Component 1 (Oauth API)
Component 2 (Get cab price and other API calls)
Component 3 (Surge calculation-Random forest Classification model)
Component 4 (Dynamic pricing model -Linear Regression model inference)
Component 5 (Output)
Component 6 (Feedback loop cron job)
Component 7 (Database)

Other components used in the application but not specific to the comparison module are login, result_generator / UI component, feedback loop using cron job – feedbackapp, database.

Design decisions incorporated



The low-level design decisions included taking into consideration the following design principles:

Decision 1: Separation of concerns
The first important design decision we made was keeping a separate module for SurgePriceModel inference. The input was a single dataframe for creating the class object, which made it a deeper module with interface less complex than the abstracted implementation. This design decision helped in handling all the functions and variables required for SurgeModel inference wrapped in a class, separated from other modules in turn reducing cognitive load of the developers.

```
class SurgePriceClassifier:
    def __init__(self, data_frame):
        self.data_frame = data_frame
        self.predictive_surge_mapping = {1: 1, 2: 1.25, 3: 1.5, 4: 1.75, 5: 2}

    def get_rush_hour(self):
        #returns rush hour tag

    def surge_prediction_model(self):
            #Contains prediction logic
```

Decision 2: API Call Abstraction
The openweathermap API returns the data for the whole day. It was important to extract only the relevant information. So, we decided to keep it as a separate module in the utils folder. If any of the weather parameters need to be added in future, the developers can directly make changes to the relevant file without breaking other codes. This API wrapper works with just a single call from the aggregator and returns the required values as a dataframe which is then used by the surge model inference module stated above.

Decision 3: Separate UI component
All the UI files were put in a separate folder named templates to ease code readability and keep the backend and the frontend separate and easy to extend. Now, developers can use the boilerplate code to develop further UI in future.

Comparison of the design with an existing software

Existing software link: **https://github.com/abhishekv5055/uber-lyft-price-prediction**

Limitations of the existing/comparison software:

- Hard coded values for locations in the main.py file. Does not support code extensibility.
- Data cleaning or manipulation steps are in the main code, no separation of concerns.
- The logic used to calculate surge multiplier is brute force and would not work in case any different values are passed to the application from the prespecified ones.
- Although the different train, test and data split modules look very well separated and support scalability, the modules are shallow with very limited implementation
- From an ML point of view, it does not calculate the surge or the dynamic price, just does the static price.
- Although, the repository has a docker file and pip file but fails to mention the usage or installation steps.
- The ML model does not account for weather as a parameter for calculating the surge price.

All the above mentioned limitations are taken care of in the implementation of Easy Cabs (surge price prediction model) as described in the above sections as well.

Extensibility of the software

Software as a whole: -

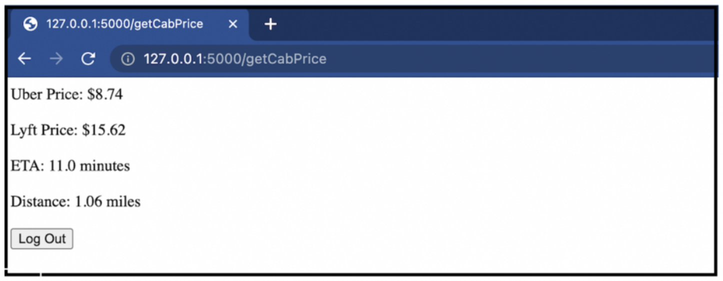
Every service, including the feedback loop, is relatively self-contained and expandable. It can be scaled horizontally and vertically. It is simple to add a database and other features. The aggregator can be scaled at any time by placing a nginx load balancer in front of it. The code has been constructed in such a way that it can be easily extended for certain classes and API calls without disrupting the original code.

Surge multiplier software: -

The current prediction model is trained for the locations within Boston. But the training of the model lies in a different class and module. Right now, we pull the weather and location data from Google APIs which has data for all the cities thus training a new model to incorporate the surge price multiplier is feasible. Data from other cities can be added to the existing training dataset in the similar format and the entire code will continue to work. However, the data for all other cities should contain similar columns/parameters used for our models and this can be a manual effort to append the data in the current training dataset. We provide this functionality with the feedback loop.

How our Models work compared to Uber and Lyft

Uber X and Lyft Lux selected which gave the following price prediction

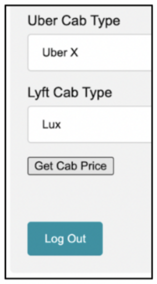


Uber Price: $8.74

Lyft Price: $15.62

ETA: 11.0 minutes

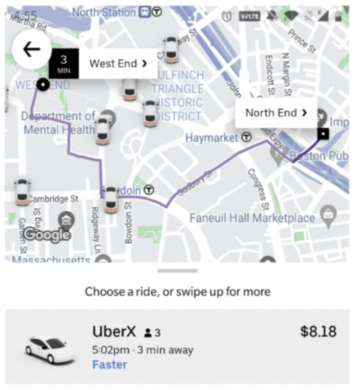
Distance: 1.06 miles



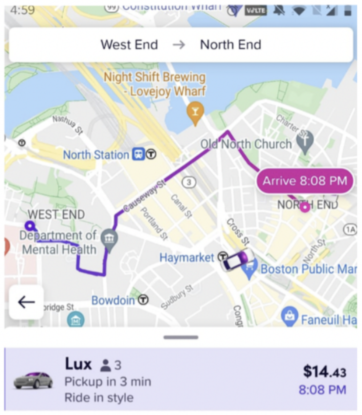
Uber Cab Type: Uber X

Lyft Cab Type: Lux

Uber X and Lyft Lux selected at the same time.



UberX 3 — $8.18
5:02pm · 3 min away
Faster



Lux 3 — $14.43
Pickup in 3 min — 8:08 PM
Ride in style

	Real App	Easy Cabs
Uber X	$8.18	$8.74
Lyft Lux	$14.43	$15.62

The above can be extensible in terms of the UI as well and a better front-end can be implemented. The requirements for the front end like logo and other images are available under the template folder in the github repo.

Software Requirements for Easy Cabs

authlib
flask-login
flask
googlemaps
geopy
requests
numpy *
openweathermap *
pandas *
pytest *
setuptools
scikit-learn *
sklearn *

*Have been specifically used for the implementation of the surge price predictor model. The comparison library used only pandas, sklearn and numpy.