Homework 6: Final Project

Team Members and Project Title:

- Edwin Chiang
- Danny Nguyen
- Long Nguyen

DinoCo

Final State Features:

User Account Creation
User Login/Logout
Ability to create road trip routes by entering endpoints and vehicle information
Ability to delete road trip routes
API to display area maps
API to obtain travel distance between any two city end points in the United States
Calculator to determine cost of travelling between two cities

DinoCo is a flask based framework that allows user to create accounts, log their travel routes, and find out the cost of a trip. Users are able to create accounts on the web app, login to the service and logout as well. When logged in users are able to create new trips that allow them to see the cost of traveling from point a to point b depending on their car's mpg and current fuel costs which are calculated as soon as the user enters their information. If the user no longer wishes to view a certain trip they can navigate to that route's specific page and delete the trip. There are various API's throughout the web app that enhance the user's experience such as the ability to view their route on google maps.

Use of Design Patterns:

Design patterns played a large role in the foundation and formulation of this project and is seen in many aspects. These are a few of the important patterns showcased throughout the website.

Factory:
We did not implement a factory, but with SQLAlchemy and Flask we were able to use a built-in function called db.create_all() which created all the necessary database tables that we would be using.

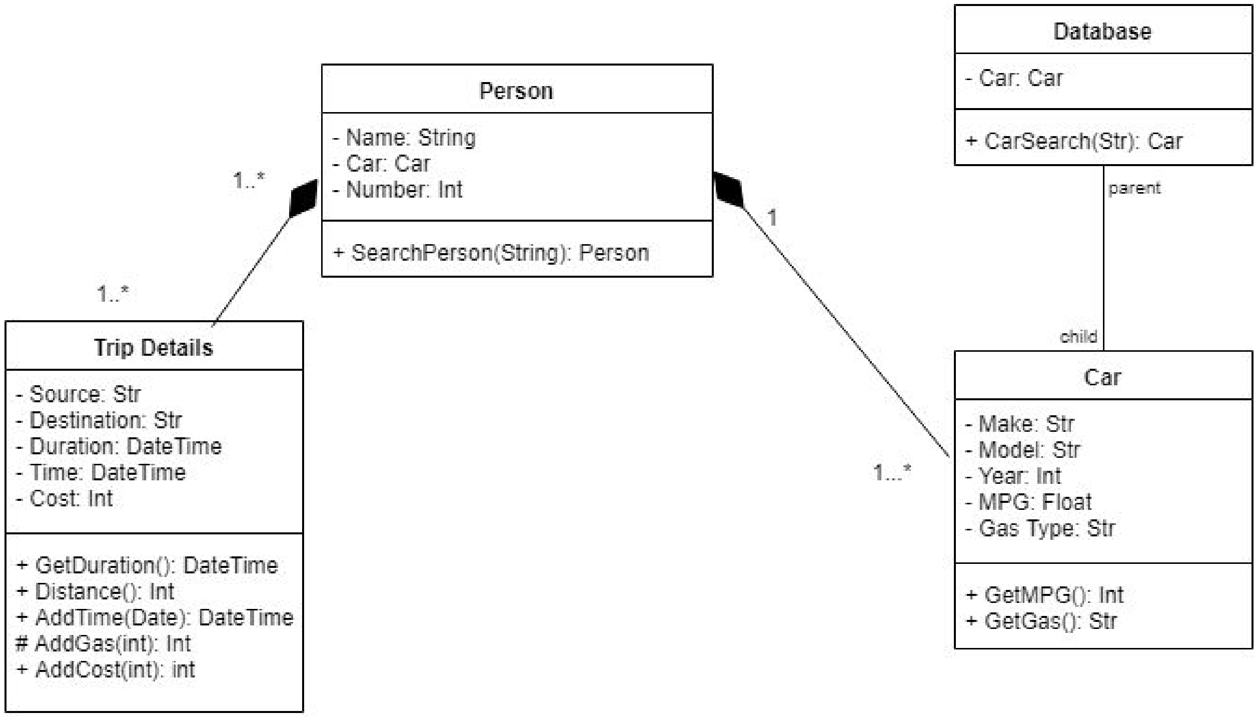
Object Pool:
Object Pool was used within this project though creating the database. The database is a large and costly object to create so all users of the web app use the same one and add their own entries to it through forms. Each user has information in the user and routes tables which combine everyone's data into a single location.

Bridge:
Bridge was used in this project through the use of multiple API's. In the web app there are two independent API's that work together through the python files. The map distance API gets locations that are entered and finds how far apart two places are and then sends it to the map api which attempts to display the end points.

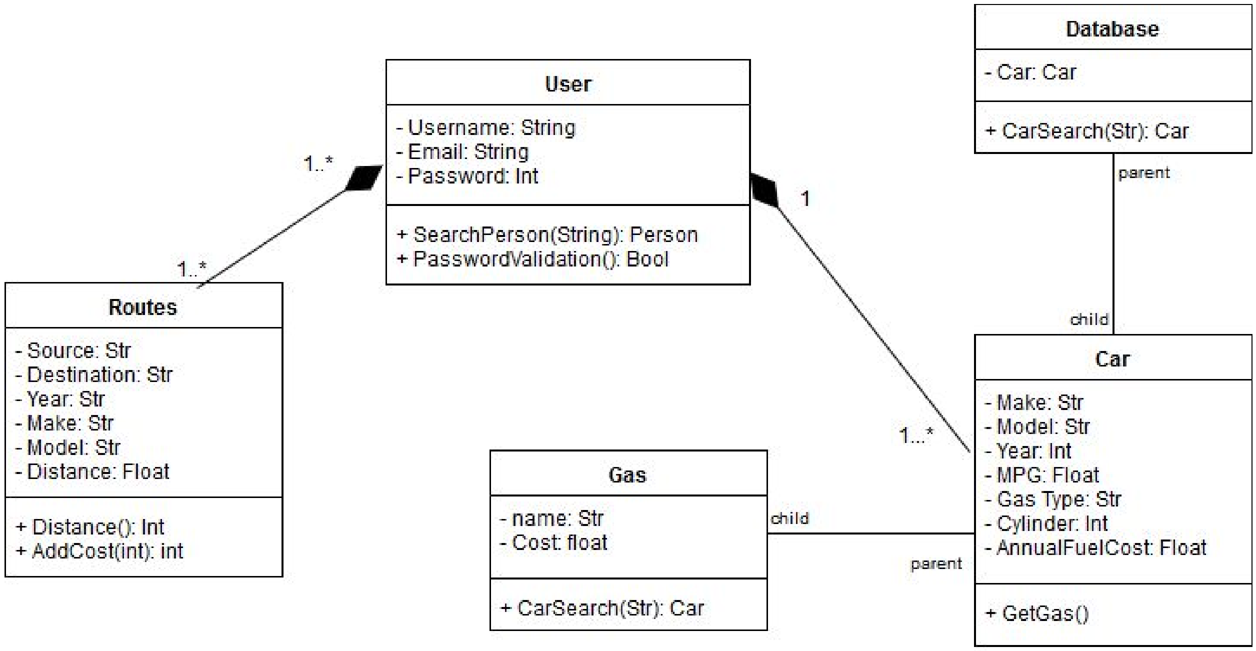
Template:
Template was used with the .html potion of the project. We have a layout.html that sets up a template and each other .html file will work off of that. We do so by using code blocks within the .html so that it pulls formating of the layout file and adds its perspective part to the front end.

Adapter:
When working with a huge dataset of car information we needed to cut it down so it is much more manageable. From Kaggle we found a file for almost all cars in existence along with all their properties like MPG, annual fuel cost, transmission, or engine types, but we do not need all this information. We then created an adapter that truncates the data down to a more specific set of data that we can use. It cut down on all the cars 10 years or older and drops all the extra information not needed in calculating the cost of a trip. This data is then entered into our database populating Car so we can query through it for information and to do validation checks. Another time we used adapter was when we were implementing API's. We used an API to check the distances between towns which resulted in json so we had to make it so that it returns usable data for ourselves. The same thing with Google maps' API.

Class Diagram:
Homework 4 Diagram:



Final Diagram:



Some changes that we made between homework 4 and now is how the cars are loaded in, what user contains, and the functionality of routes. Since we populate Cars with the data from a database many of the things we expected to create a method out of came with the data set. There was only annual fuel cost in the data set so we had to implement another database which pulls what gas type the car uses and adds the new information based on the query. Users now need a password and email to register an account. This makes it so each user has their own routes and cars that they drive. Routes now only focused on the distance it takes to get from town to town and how much gas would cost. We felt time was not of importance in this system.

What we learned:

We came up with an idea to calculate travel cost and from there we had to analyze what was needed to answer that problem. We looked at what we needed like the data and how to get it in working order. As we were creating the program if we did not keep in mind single responsibility principle it would create mess and unidentifiable code. Keeping the responsibilities of the code seperate aided us in debugging the code. This occured the most when working with the database. Instead of going through one long file of code. Having a structure of how the classes relate between each other before starting gave us a good framework. It allowed us to have a reference for review and how things should come together. Don't Repeat Yourself was stressed when we were creating the web pages since many of the pages look identical with some discrepancy. This birthed the layout file so that each remaining one focused on individual content rather than how it looks. In the program many parts required interpretation of data so it worked correctly. This was shown most vividly when working between programs like flask and sqlite.

3rd Party Tools/Sources:

https://www.youtube.com/playlist?list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH
Credit to YouTuber Corey Schafer, his playlist linked above was used to establish a file structure, setup the flask framework, and understand how flask works.

https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world
Credit to Miguel Grinberg, code snippets on his blog were adapted to help us use and understand the SQLAlchemy ORM.

https://developers.google.com/maps/documentation/javascript/tutorial
Google Maps API used to display Google's map on our route specific pages.

https://developers.google.com/maps/documentation/directions/start
Google Directions API used to take user input on locations and calculate distance traveled.

https://www.kaggle.com/epa/fuel-economy
Used the csv from here to populate our database.

The portion of code written that was ours was the API integration, anything that had to do with querying/sending data, and all of the HTML that is displayed on the corresponding pages.

For our original code elements of design we mainly used adapter and bridge to integrate the APIs and template for having a base HTML layout and building off of that for all our other pages.