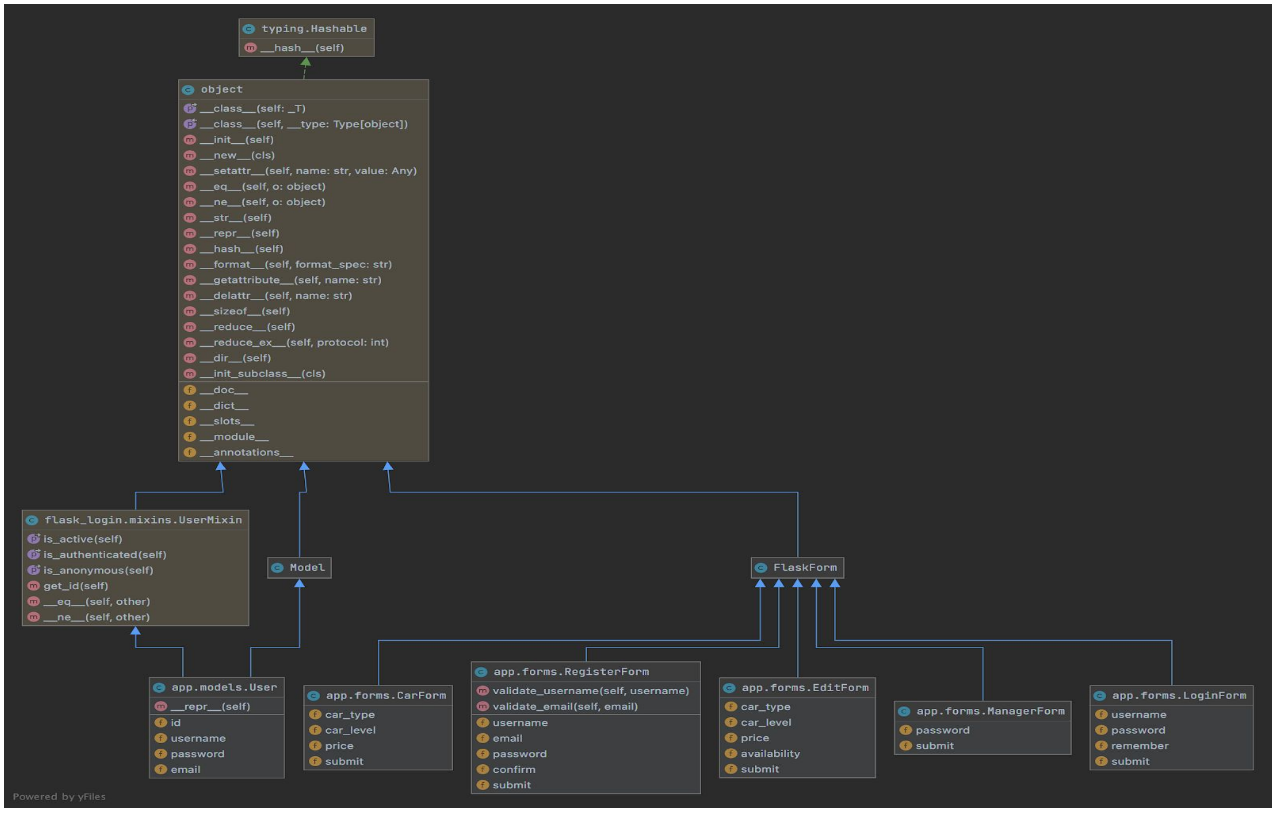
Project name: Car rental system

Team member: Fei Hu, Jie Wang

Final State of System Statement: This car rental system is finished by Flask, wtform, bootstrap... as package, Sqlalchemy as the database for users, Mysql as the database for storage, python as programming language. The rental system allows users to book the car and let the manager easily operate the storage which includes adding, editing, deleting the car. The database is connected between the customer and the manager. For example, if a manager adds a new car into the system, it will pop up when the customer checks the certain car type. We didn't implement the car class in project 6. In our original thought, we think the price of a car should vary by car type and its level. And we can apply the decorator design pattern to get the sum of car_type and car_level. However, when we do project 6, we think it's better to let the manager directly modify the price. So we remove this feature.

Final Class diagram and comparison statement:

UML class diagram



This is no pattern in this diagram since flask project used route as its main function implementation file. We use two design patterns(decorator, MVC) and one ooad concept tool(ORM: sqlalchemy).

Decorator:
All functions are decorated by @app.route. And also, because some of the page should be entered after login, we add one more decorator called @login_required. To protect data security and administrative rights, we require users to log in and enter a password before entering any page. In this way, even if the user directly input the corresponding sub-url in the url bar, also can not directly enter.

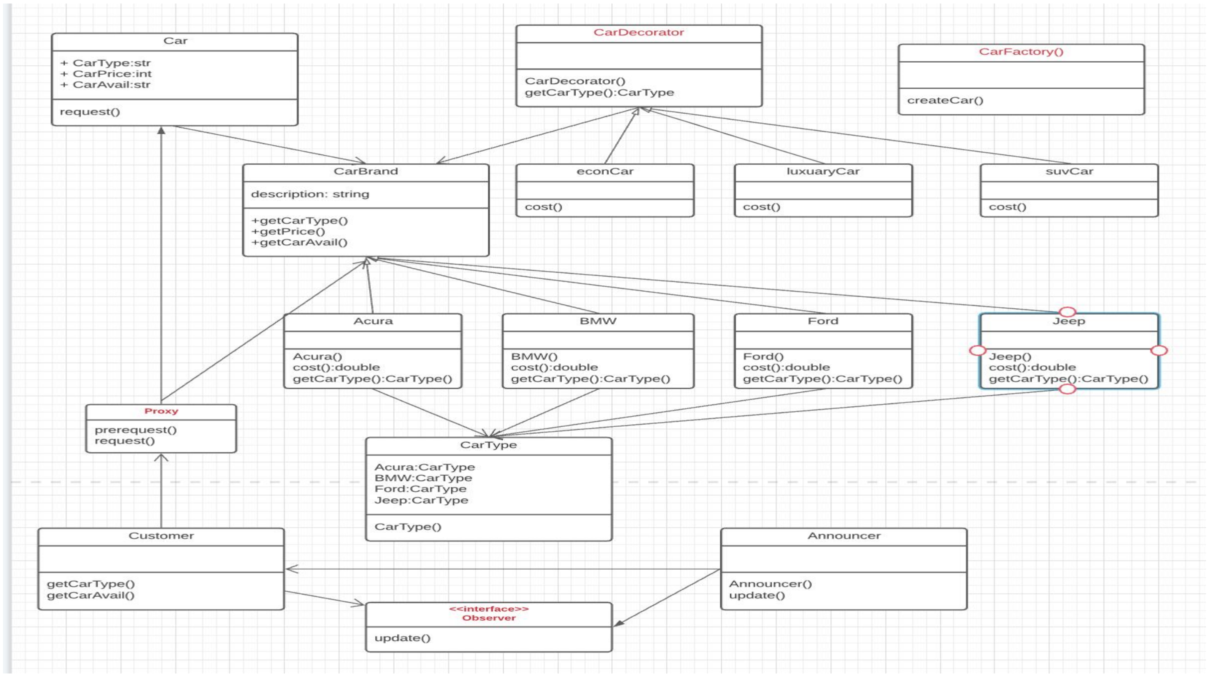
MVC:

MVC: "Model: Mysql, sqlalchemy",	
	"view: customer, manager's page",
	"controller: add, edit, and delete car(manager side), choose and confirm car(customer side)"

We use Mysql and sqlalchemy to store our user info and car info. And in the customer and manager's page, we can easily observe the data from the database and see what was changed. To operate the data, we have adding, editing, and deleting car features to update the status. This can act on both the model and view side.

ORM:
We use sqlalchemy as our user info database. By using this ORM technique, we can easily convert the data between database and programming language(python).

UML class diagram in Project 4:



We change a lot from Project 4. More simple and easier to understand. In project 4, we decided to use Java with the static type. And we switched to python with the dynamic type in project 5 and finished the remaining part in project 6.
We gave up with the decorator to calculate the price based on car_type and car_level. Instead of that, we used a more dynamic way which provides an editing page to let the manager directly modify the price. We dynamically initialized the car instead of using a factory pattern to initialize it. We used more decorator to control the access of the user. This hasn't been mentioned in Project 4.

Third-Party code vs Original code Statement
Framework: flask, flask-sqlalchemy, flask-wtf, flask-mysql
Original code: customer(), manager(), choose_car(), book_car(), successful(), statistic(), statistic_car() and all related html pages.

Third-Party code: We didn't use the same functions as others. Usually half part of these functions are our code and half are third-party code.
register(), login(), logout(), dashboard(), add_car(), edit_car(), delete_car() and all related html pages.

Third-Party resource: YouTube:
https://www.youtube.comwatchv=RWviEK1Si68&list=PLDFBYdFBxV1G4FBpG1EMyFtbsbZuJOvD
https://www.youtube.com/watch?v=addnlzdSQs4

Statement on the OOAD process for overall semester project:

Language choice: At the beginning of the project, we decided to use Java and Tomcat, and applied ooad on it. Since we were in the mid way to develop, we found Python and flask were a more convenient way to finish this car rental system. And also, in flask web development, there were also a lot of places that could be applied with OOAD. So one week before submitting project 5, we switched to Python and flask.

Design pattern using: We firstly thought about link the car price with the car_type and the car_level by decorator. However, after finishing developing the basic car system, we found it's more convenient to let the manager directly operate the car storage and change the price. In the real world, the car price usually changes with the demand. So that's why we changed part of the design pattern.

Timer: In our original thought, we wanted to design the availability based on the time period. For example, when the manager is initializing a car in the storage, they will need to enter which time period it will be available. And in the customer page, the availability will also show up. After the customer books the car, the car will be returned to the storage based on the real world time.

But we found this was really bad when demoing. During the demo, we couldn’t wait for the real time to test if this function works fine. So we decided to remove the timer in the system. And let the manager directly operate the car’s return. For example, when the customer wanted to return the car, the manager can go to the dashboard page and change the certain car’s availability from ‘no’ to ‘yes’.