

Final Project Part 3

UsedCarCentral

The Online Hub for Buying and Selling Used Cars

Section 1: Web App Architecture

1. Overview:

Model: We are using Microsoft SQL Server as our database management system to store data for our web application. The data is organized and structured into tables to minimize data redundancy and ensure efficient queries using Normalization. We are also planning to develop procedure calls for increased db security.

View: For our front-end, we are using HTML, CSS, and JavaScript to create a visually appealing and dynamic user interface. There will be several listings, dropdown menus and adding, updating, deleting the listings within the user respective workflows (without giving access to the entire data).

Controller: For the back-end development, we are using Python with the Flask framework as the controller in our MVC architecture. Flask acts as the intermediary between the front-end and the back end and manages the flow of data between the two. Flask provides various features such as URL routing, template rendering, and database integration to streamline the development process and ensure efficient and effective communication between the front-end and the back end.

2. Web Architecture:

Data Storage:

For our web application, we have chosen MSSQL as our database management system. MSSQL is a reliable and robust RDBMS that is capable of handling large amounts of data with ease. It offers advanced features such as indexing, transaction processing, and backup and recovery, which are essential for building a scalable and secure web application. The dataset we are using for our application contains approximately 550,000 records of used car listings from various online portals. The dataset contains 26 columns, including car make, model, year, condition, odometer reading, price, and location. We will be using this data to create a database for our web application. To ensure that the data is structured and organized in an efficient manner, we will be normalizing the data and splitting it into tables as needed. This will allow us to reduce data redundancy and minimize data anomalies, which can improve the performance and reliability of our application. Additionally, by normalizing the data, we can ensure that our queries are more efficient and effective.

Backend:

For our back-end development, we have chosen Python with the Flask framework. Python is a versatile programming language with a wide range of libraries and frameworks available to help us build our application. Flask is a micro web framework that is well-suited for small to medium-sized web applications. Using these in stack we are trying to build useful features such as URL routing and database integration in an easy manner. By using Python with the Flask framework, we are confident that we can build a robust and scalable web application that meets our needs.

Data Base Access/ Connections/ Security:

To integrate our Microsoft SQL Server db with Flask, we are using the Flask-SQLAlchemy extension. Flask-SQLAlchemy provides a set of easy-to-use SQLAlchemy helpers for Flask applications. It simplifies the process of connecting to the database and performing common database operations, such as creating tables, inserting data, and querying data.

In Flask application, we can configure the database by setting the **SQLALCHEMY_DATABASE_URI** configuration variable to your database connection string. Like below:

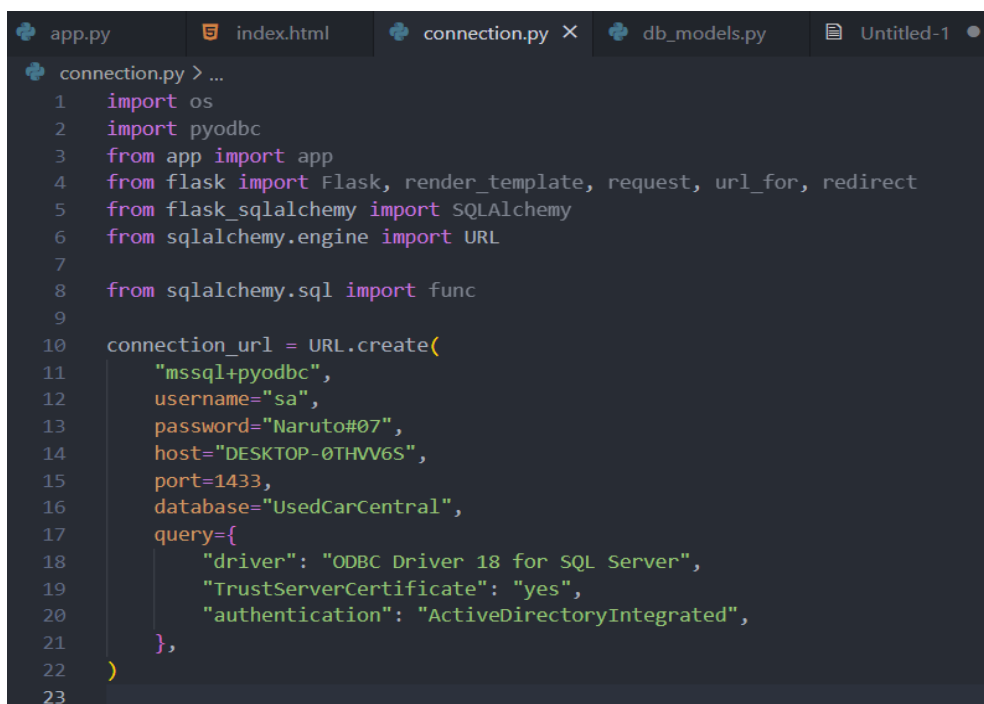
```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'mssql+pyodbc://user:password@server/database'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
```

Once you have configured your Flask application with Flask-SQLAlchemy, you can define your database models using SQLAlchemy's declarative syntax.

Our code for Database access:

'connections.py':

- The **os** module is imported for access to environment variables.
- The **pyodbc** module is imported for connecting to the Microsoft SQL Server database.
- The **Flask** module is imported to access the **app** instance.
- The **SQLAlchemy** module is imported for database interaction.
- A **URL** object is created using the **SQLAlchemy URL** module, which specifies the connection parameters for the Microsoft SQL Server database.
- The **connection_url** object is then used to configure the **SQLAlchemy** database connection.



```
app.py  index.html  connection.py X  db_models.py  Untitled-1 ●
connection.py > ...
1  import os
2  import pyodbc
3  from app import app
4  from flask import Flask, render_template, request, url_for, redirect
5  from flask_sqlalchemy import SQLAlchemy
6  from sqlalchemy.engine import URL
7
8  from sqlalchemy.sql import func
9
10 connection_url = URL.create(
11     "mssql+pyodbc",
12     username="sa",
13     password="Naruto#07",
14     host="DESKTOP-0THV6S",
15     port=1433,
16     database="UsedCarCentral",
17     query={
18         "driver": "ODBC Driver 18 for SQL Server",
19         "TrustServerCertificate": "yes",
20         "authentication": "ActiveDirectoryIntegrated",
21     },
22 )
23
```

'db_models.py':

- The **os** module is imported for access to environment variables.
- The **app** module is imported to access the **app** instance.
- The **connection_url** object is imported from the **connections.py** file to specify the database connection URL.
- The **SQLAlchemy** module is imported for database interaction.
- The **Column**, **Integer**, and **String** classes are imported from **SQLAlchemy** to define the schema of the **CarsMasterData** table.
- A **db** object is created using the **SQLAlchemy** module and the **app** instance to initialize the database.
- The **CarsMasterData** class is defined, which is a subclass of the **db.Model** class provided by **SQLAlchemy**. It defines the schema of the **CarsMasterData** table in the database. Each **Column** corresponds to a column in the **CarsMasterData** table. The **primary_key=True** argument specifies that the **CarID** column is the primary key.

```
models > db_models.py > CarsMasterData
1  import os
2  from app import app
3  from connection import connection_url
4  from flask import Flask, render_template, request, url_for, redirect
5  from flask_sqlalchemy import SQLAlchemy
6  from sqlalchemy import Column, Integer, String
7  from sqlalchemy.sql import func
8
9  app.config['SQLALCHEMY_DATABASE_URI'] = connection_url
10 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
11
12 db = SQLAlchemy(app)
13
14 class CarsMasterData(db.Model):
15     CarID = Column(Integer, primary_key=True)
16     MasterID = Column(Integer)
17     Manufacturer = Column(String)
18     ModelYear = Column(Integer)
19     CylinderCount = Column(String)
20     FuelType = Column(String)
21     TransmissionType = Column(String)
22     CarSize = Column(String)
23     CarBodyType = Column(String)
24     CarColor = Column(String)
25     VehicleIdentificationNum = Column(String)
26     DriveType = Column(String)
```

Regarding **security**, Flask itself doesn't provide any built-in security features, but there are many third-party libraries and extensions available that can help us secure our web application, such as Flask-Security, Flask-Login, and Flask-JWT.

To prevent SQL injection attacks, we are planning to use stored procedure calls. On top of that, we are also planning to restrict access to database using access control and protect our connection.py file which has the database connection string URL in it. That is the reason why we separated the connection.py file, so that we can just import the connection URL and use it rather than exposing the database connection string.

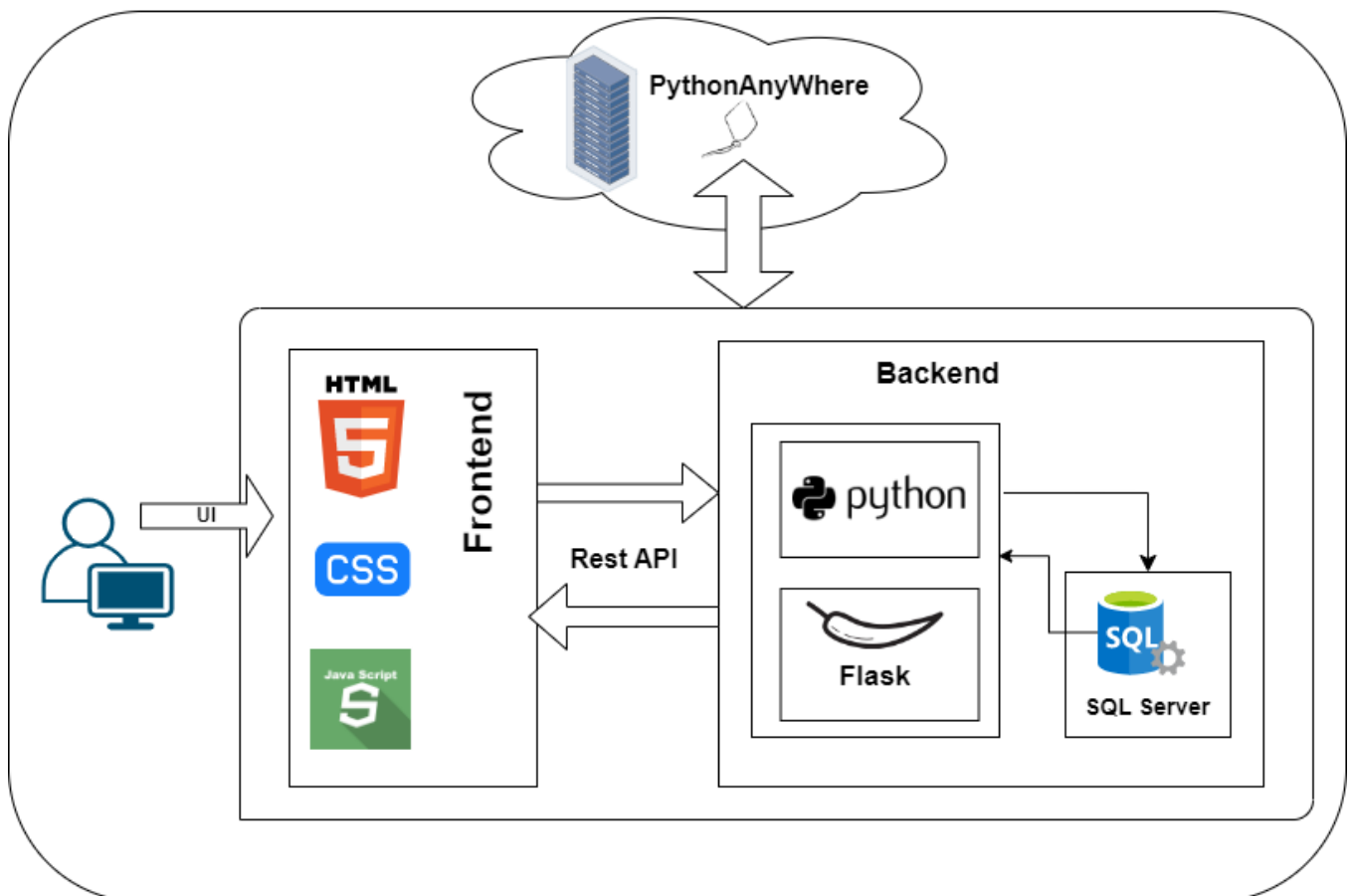
Frontend:

We are using HTML, CSS, and JavaScript for our front-end development. These languages work together to create a dynamic and responsive web interface. HTML provides the structure, CSS adds visual elements, and JavaScript brings interactivity to our web application. Using these languages allows us to create an engaging user experience. We will also be adding navigation to different tabs and the ability to add and update lists from the front-end. This will provide users with a intuitive interface that allows them to easily switch between different sections of the application and add or modify information as needed (within the restricted access to the data).

App Deployment:

We have selected PythonAnywhere as the server for our web application. PythonAnywhere is a suitable choice for small to medium-sized applications as it provides an easy-to-use platform for deploying and managing the application. This platform offers various features such as a web-based console, a file editor, and a database interface.

Architecture:



Section 2: Web App Layout

Initial layout:

The landing page of our website consists of our company's mission statement along with a navigation bar on the top and our contact information in the bottom. The company name is present on the left side of the navigation bar, and we have two options on the right. One to login to the application and the other to view all the available listings.

Menu panel:

The menu panel is present on the navigation bar on the top of the screen. The options present on this bar depends on the login. If the user has logged in, we can see three options. 1. My Listings, 2. View Listings, 3. Logout. If the user has not logged in, we can see two options. 1. View Listings, 2. Login.

Pages and tabs:

Mostly, we will be having the following pages in our application.

- Landing Page
- Login Page
- Register Page
- Listings Page
- My Listings Page

Color schema:

We are using a modern dark pallet with black top and bottom bars upon which we have used vibrant red and white fonts. We have also placed a blurred cars image as the background of the webpage.

Display Description:

- Landing Page: It consists of our company's mission statement along with a navigation bar on the top and our contact information in the bottom. This is the first page a user can see in our application.
- Login Page: We can go to the login page by clicking on the login link on the top right side of the landing page on the navigation bar. It has email and password input fields in it. We also have a register link in the bottom.
- Register Page: Once a user clicks on the register link on the login page, we shall redirect him to the Register page. User info such as name, email and password are collected here to create a new account in our system.
- Listings Page: On the landing page, we have listings link right beside the login link on the navigation bar. This will take us to the listings page in which we can find all the cars listed by sellers. It has all the details of the car such as make, model, year, etc.
- My Listings Page: Once a user logs in, we can find a new "My Listings" link beside the actual listings link on the top navigation bar. This link takes us to a page where a user can view all the cars listed by him. There will be an option for the user to update or delete an existing listing and create a new listing.

*In addition to these pages, we might have one more page which pops up when a user clicks on a specific listing. This consists of more details regarding the selected listing.

Available Functionalities and User workflow:

Functionalities such as adding a new listing, updating, or deleting an existing listing, viewing all the cars listed by other users, filtering these listings, etc are provided to the users of our application.

Sample screenshots:

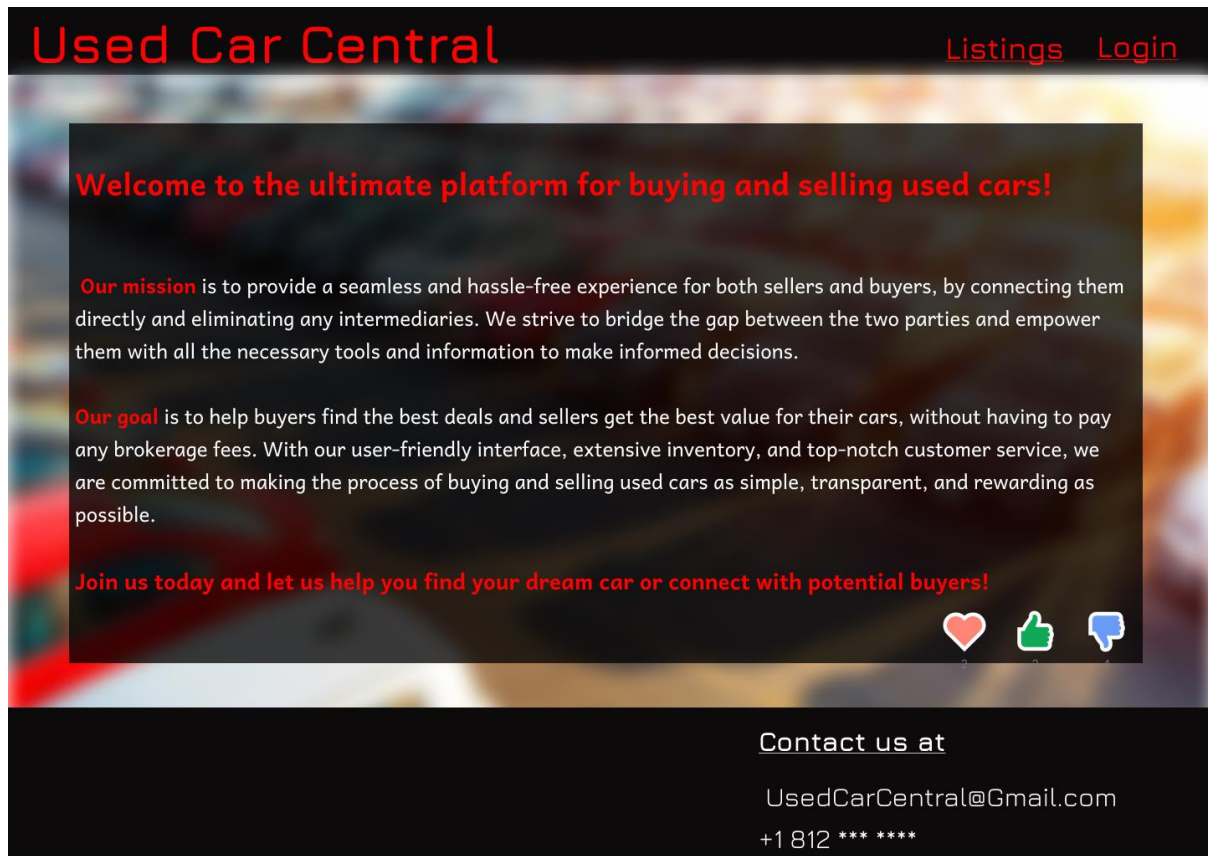


Fig 1. Landing Page

Used Car Central

REGISTER

Name:

Mobile:

Email :

Password:

Register

Contact us at
UsedCarCentral@Gmail.com
+1 812 *** ****

Fig 2. Register Page

Used Car Central

LOGIN

Email :

Password:

Login

Contact us at
UsedCarCentral@Gmail.com
+1 812 *** ****

Fig 3. Login Page

Section 3: Individual and Team Work Assessment

Name: Ram Kiran Devireddy (radevir)	
Criteria	Score (1-10)
Task completion	10
Teamwork	10
Time Commitment	09
What could be done better	We could've spent more time on the security part of the database. The credentials for connecting to database are hard coded in the connection_url variable. This is not good practice and can compromise the security. It's better to store the credentials in a secure location, such as an environment variable or a configuration file, and load them dynamically.

Name: Syam Prajwal Kammula (skammul)	
Criteria	Score (1-10)
Task completion	10
Teamwork	09
Time Commitment	09
What could be done better	The code does not perform any input validation, which can leave it vulnerable to SQL injection attacks. It's important to validate user input and sanitize it before passing it to the database. For this, we're planning to use stored procedure calls (sp calls)

Name: Revanth Posina (rposina)	
Criteria	Score (1-10)
Task completion	9
Teamwork	10
Time Commitment	9
What could be done better	The code does not have any logging, which can make it difficult to diagnose issues and monitor performance. It's important to log events and errors in database to facilitate troubleshooting and performance monitoring.