Final Project Part 4 – Technical Report
UsedCarCentral

The Online Hub for Buying and Selling Used Cars

Section 1: Web App URL and GitHub

1. Full URL of our deployed UsedCarCentral Website.

   http://34.125.254.29:5000/

2. Full URL of the GitHub Repository we used for storing our project code.

   https://github.com/ramkiran55/UsedCarCentral.git

   We utilized the repository above as a centralized location for our codebase and version control management. Using this repository allowed us to easily collaborate and share updates with one another, making the development process much more efficient. Additionally, having all of our code in one place made it simple to keep track of changes and revert to previous versions if necessary.

Team

Ram Kiran Devireddy (radevir)

Syam Prajwal Kammula (skammul)

Revanth Posina (rposina)

Section 2: Project Purpose and Functionality

Purpose

Our project aims to develop a user-friendly web application that facilitates the buying and selling of used cars using Kaggle's Used Cars Dataset. Sellers can post or edit their existing used car listings, while buyers can browse, choose cars of their interest, and purchase the used cars they are interested in. Users can register on the website to access the available car listings, or they can hop on to "my listings" page and add car postings or ads they want to sell.

- Create a user-friendly web application for buying and selling used cars.
- Allow users to easily find all the available car listings or ads and select the car they are looking for based on their preferences (SELECT statements).
- Provide sellers with a platform to easily list and manage their used car listings (Insert, delete and Update statements).

Future Scope

After receiving valuable feedback during our peer review, we have decided to enhance our project with the following functionalities.

1. First, we plan to integrate a chat feature that will allow buyers and sellers to communicate directly on our platform.

2. Second, we aim to incorporate a search filter that will enable buyers to refine their search criteria based on location, price, and other relevant factors.

3. Finally, we are looking to implement a map feature that will display the location of each car listing using the latitude and longitude attributes provided in our dataset.

In addition to these improvements, we are planning to leveraging React's material UI to enhance the user interface and provide a more seamless and responsive user experience. We believe that these enhancements will make our platform even more user-friendly and efficient, and we are excited to move forward with these changes.

Usefulness

Our database web application is designed to cater to the needs of people who want to buy or sell used cars. With our platform, buyers can easily browse through listings and find the car that meets their preferences at the best price. Similarly, sellers can easily list their used cars, manage their listings, and connect with potential buyers on our platform. Our interactive and user-friendly interface makes the process of buying and selling used cars efficient and hassle-free.

Communication and Sharing

We have used two GitHub repositories for sharing, managing and controlling the versions of our code.

1. https://github.com/ramkiran55/UsedCarCentral.git

   The above is our main code repository which we have used for sharing our code between ourselves and version control.

2. https://github.com/ramkiran55/UsedCarCentral_Team_SRR.git

   We have used the above repository in the initial phases of our project building like project proposal.

   On top of that, we used Microsoft Teams for collaboration.

Building the Project: A Comprehensive Overview of the Development Process

Our project was developed using the MVC (Model-View-Controller) architecture.

Model

We opted to use Microsoft SQL Server as our database management system for our web application, employing a table-based structure to minimize data redundancy and enable efficient queries through the principles of normalization. Additionally, to enhance both performance and security, we developed procedure calls for all of the CRUD applications we employed in our web app.

Our web application implements CRUD (Create, Read, Update, Delete) operations using stored procedures in Microsoft SQL Server. These procedures provide a secure and efficient way to interact with the database, as well as help minimize errors and ensure consistency in data manipulation.

These Stored Procedures are:

For creating and adding car listings:

- *real.CreateUsedCarsMasterData*
- *real.CreateCarsMasterData*
- *real.CreateCarDetails*
- *real.CreateLocations*
- *real.CreateListings*
- *real.CreateUserCarListings*

For reading and selecting car listings:

- *real.ReadUserCarListing*
- *real.ReadUserCarListings*

- *real.ReadCarListings*
- *real.GetUserByUserEmail*

For deleting car listings:

- *real.DeleteCarListing*

For updating and editing car listings:

- *real.UpdateCarListing*
- *real.UpdateCarPrice*

These Stored Procedures have been utilized in the 'Listings' and 'MyListings' pages of our website, to enable efficient and secure data transactions.

View

To create a visually appealing and dynamic user interface, we utilized HTML, CSS, and JavaScript for our front-end development. Our UI includes multiple listings, dropdown menus, and streamlined workflows for adding, updating, and deleting listings while ensuring user access is restricted to their respective data only.

In flask, all the view or html files must be stored under the templates folder.

Below is the workflow of our project:

- *landing.html*
- *login.html*
- *register.html*
- *prelisting.html*

The above HTML pages serve as the initial interface for our website. The *landing.html* page is the first point of contact for users, providing basic information about our website along with links to the *login* and *registration* pages. The *login* and *register.html* pages allow users to create an account or sign into their existing account. Once logged in, users are directed to the *prelisting.html* page, which offers guidance on how to navigate our website, including links to *mylisting.html* and *listing.html* pages which contain the actual functionality of our web application.

- *listing.html*
- *mylisting.html*

The *mylisting.html* and *listing.html* pages contain the actual functionality of our website. The *listing.html* implements the Read functionality from the database and displays all the car listings whereas the *mylisting.html* displays only the car listings added by the user who logged in, such that they will be able to add, delete or edit their car listings.

- *AddCarDetails.html*
- *AddCarMasterData.html*
- *AddCarLocationDetails.html*
- *FilterForUpdates.html*

Finally, In the *mylisting.html*, users will be able to edit, delete their car listings and there will be link to direct the users to add new car listings. The above html pages, *AddCarDetails.html, AddCarMasterData.html* and *AddCarLocationDetails.html* implements the Create or Add Car listing features which calls the respective stored procedures to create a car listing, while the *FilterForUpdates.html* page is used to update a car listing.

Controller

Finally, to power our application's back end, we utilized the Flask framework of Python, which acted as the controller in our MVC architecture. It enabled us to manage the data flow between the front-end and back-end through URL routing, template rendering, and database integration features. We ensured seamless communication between the two parts of the application by employing various Flask features, allowing for efficient and effective development of the project.

- *UsedCarCentral.py*

The brain or the master of our web application is the above *UsedCarCentral.py* file. This file contains the code to implement all the functionalities of our application, while also performing URL redirections and rendering html pages.

- *connection.py*
- *db_models.py*

To implement the procedure calls to the database and secure our database calls we have used the *db_models.py* file. Whereas the *connection.py* file contains the connection string of our database application. We simply called the *connection.py* API whenever we needed to connect to the database to execute procedure calls.

- *User.py*

Finally, the *User.py* file contains the logic for user related activity, such as login and registration.

Tools and Technologies

We have used.

1. **Microsoft SQL Server** as our main Database technology.

2. **Python** and **Flask** and the Controller logic to implement RESTful APIs and business logic.

3. **HTML, CSS** and **JavaScript** to build our view or the UI of our application.

For writing the procedures, database creation scripts etc., we have used the **SSMS (SQL Server Management Studio)** and for coding the rest of the application, we used **VisualStudioCode**.Other than these, We used **Git** for Version Control and Teams for collaboration.

Technologies

Frontend: HTML, CSS and JavaScript.

Backend: Python and Flask.

Database: MS SQL Server.

Tools:

Visual Studio Code (VS Code), SQL Server Management Studio (SSMS), Git and Microsoft Teams.

Deployment/Hosting

For the deployment of our web application, we have used Google Cloud Platform. The web app is deployed in a Virtual Machine using compute engine provide by the **Google Cloud Platform**.

We have hosted our Microsoft SQL Sever database in a cloud SQL Server provided by **Microsoft Azure**.

Technologies for hosting: Google Cloud Platform and MS Azure

Data: Dataset and Database Design

Dataset

The Kaggle dataset we used 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 used this data to create a database for our web application and normalize the data and split it into tables as needed. We also supplement this data with additional information from other sources to provide more comprehensive listings.

The dataset was collected by Austin Reese and published on Kaggle in 2019. The purpose of the dataset is to provide a comprehensive collection of used car listings for research and analysis.

Dataset: https://www.kaggle.com/datasets/austinreese/craigslist-carstrucks-data

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.

Our project utilizes two schemas, namely the "Test Schema" and "Real Schema". The Test Schema was created to facilitate data loading, table creation, and testing of various table creation methods. This helped to bring greater clarity to the data and enabled us to create meaningful data divisions. Once we had finalized the data structure, we moved the code and data to the Real Schema, which serves as the main schema for our project.

Data Base Constraints:

A **primary key** (**PK**) is a field or set of fields in a table that uniquely identifies each record. It is used to enforce integrity constraints and ensure that each record is distinct.

A **foreign key (FK)** is a field or set of fields in one table that references the primary key of another table. It is used to establish relationships between tables and ensure that data is consistent across them. FKs help maintain data integrity by preventing actions that would leave orphaned records or violate referential integrity.

The relationship between the tables can be described as follows:

- The UsedCarsMasterData table has a one-to-many relationship with the CarsMasterData table, as one used car can have multiple car models.

- The CarsMasterData table has a one-to-many relationship with the CarDetails table, as one car can have multiple details such as odometer reading and car condition.

- The UsedCarsMasterData table has a one-to-many relationship with the Locations table, as one used car can be available in multiple locations.

- The UsedCarsMasterData table has a one-to-many relationship with the CarListings table, as one used car can have multiple listings or posts.
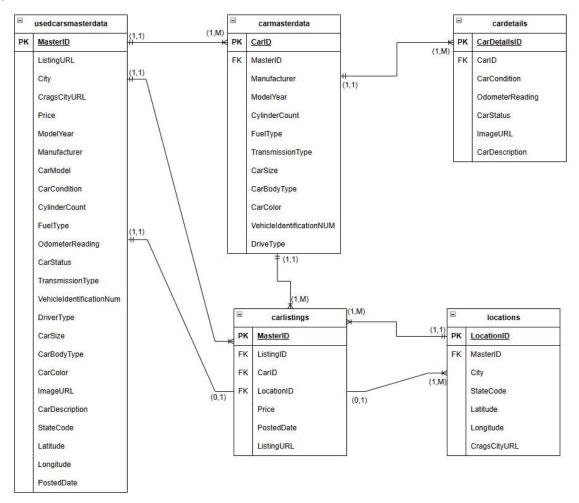
Table 1: UsedCarsMasterData

- Primary key: MasterID
- The MasterID column ensures that each record in the table is unique.

Table 2: CarsMasterData

- Primary key: CarID
- Foreign key: MasterID (from UsedCarsMasterData)
- The CarID column ensures that each record in the table is unique. The MasterID foreign key links each record in this table to a specific record in the UsedCarsMasterData table.

Table 3: CarDetails

- Primary key: CarDetailsID
- Foreign key: CarID (from CarsMasterData)
- The CarDetailsID column ensures that each record in the table is unique. The CarID foreign key links each record in this table to a specific record in the CarsMasterData table.

Table 4: Locations

- Primary key: LocationID

- Foreign key: MasterID (from UsedCarsMasterData)
- The LocationID column ensures that each record in the table is unique. The MasterID foreign key links each record in this table to a specific record in the UsedCarsMasterData table.

Table 5: CarListings

- Primary key: ListingID
- Foreign keys: MasterID (from UsedCarsMasterData), CarID (from CarsMasterData), LocationID (from Locations)
- The ListingID column ensures that each record in the table is unique. The foreign keys link each record in this table to specific records in the other tables. The MasterID foreign key links each record in this table to a specific record in the UsedCarsMasterData table. The CarID foreign key links each record to a specific record in the CarsMasterData table. The LocationID foreign key links each record to a specific record in the Locations table.

Functionalities

Our web application offers a comprehensive set of functionalities to facilitate the buying and selling of used cars. These include:

1. User Authentication: Our web application offers a secure login and registration functionality that allows users to access their accounts and personalize their experience.
2. Browse All Listings: Users can browse through all available used car listings on our platform without the need to create an account.
3. My Listings: Users who have logged in can view their own car listings and manage them from a centralized dashboard.
4. Add New Listings: Our platform allows users to create and add new car listings by providing all the necessary details.
5. Update Existing Listings: Users can update or edit the details of their existing car listings, allowing them to keep their listings current and accurate.
6. Delete Listings: Users also have the ability to delete their existing car listings, providing them with full control over the content on our platform.

Our web application provided a comprehensive platform for buying and selling used cars. To ensure that users had a smooth and seamless experience, we included a range of functionalities that catered to all their needs.

Users were able to register and log in to the platform, which gave them access to all the available used car listings. The platform provided two separate views for the listings - one for all the available listings, and another for the listings added by the users themselves.

Users could add new car listings by providing all the relevant details, such as the make, model, year, and price. They could also update the details of the car listings they added and delete them if required. All these
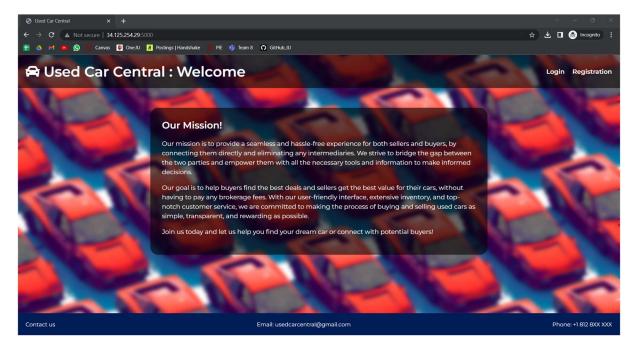
functionalities were implemented using stored procedures, which ensured efficient and secure access to the database.

To ensure a responsive and dynamic user interface, we used HTML, CSS, and JavaScript in the front-end development. The Flask framework with Python was used as the controller in our MVC architecture, which managed the flow of data between the front-end and back-end.
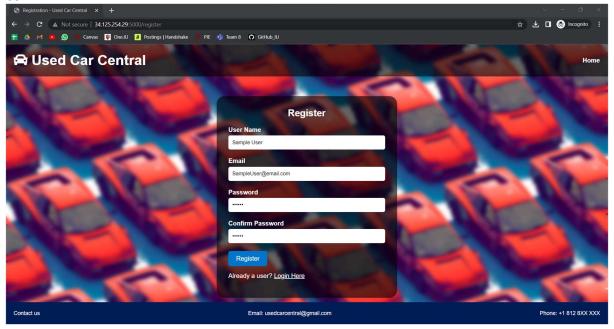
Overall, our web application provided a user-friendly and efficient platform for buying and selling used cars, with all the necessary functionalities for a seamless experience.
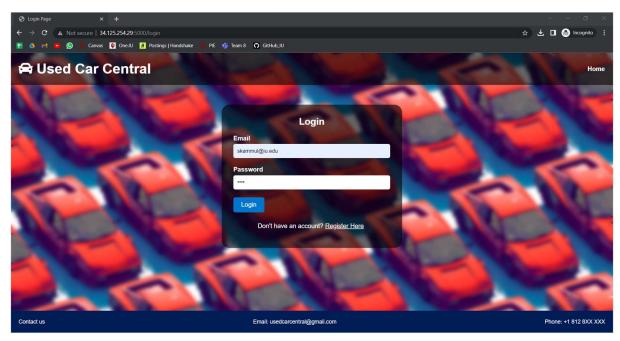
Workflow

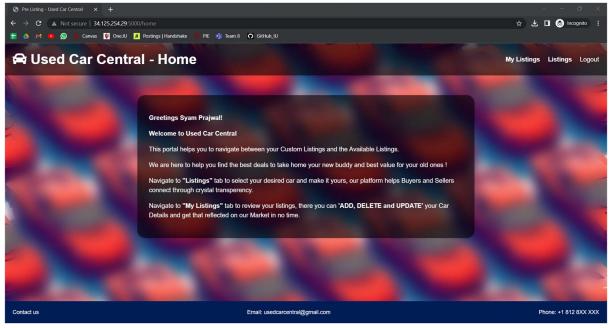Below are screenshots showing the workflow of our web application.
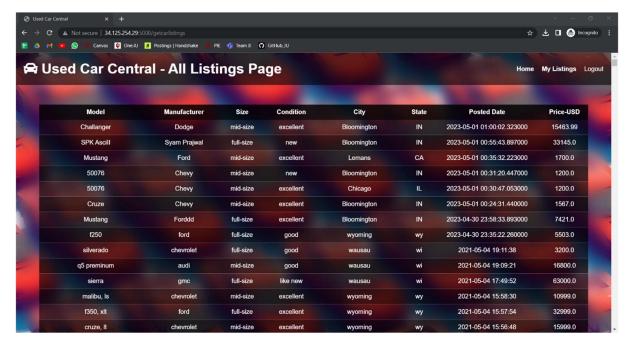

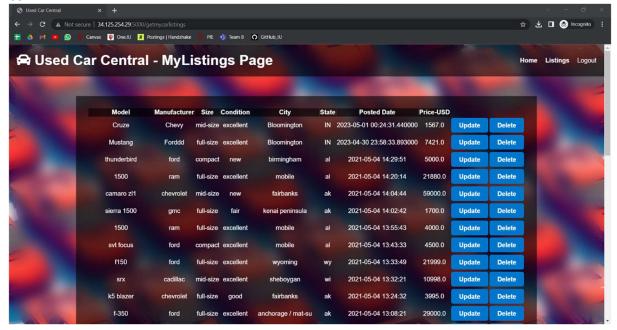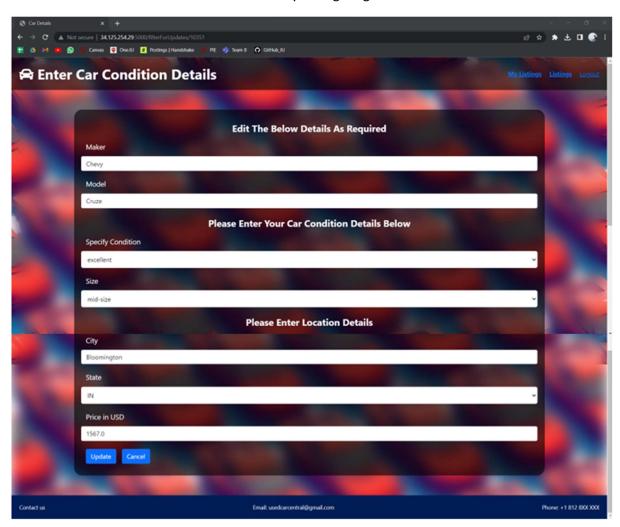Landing Page

Registration Page
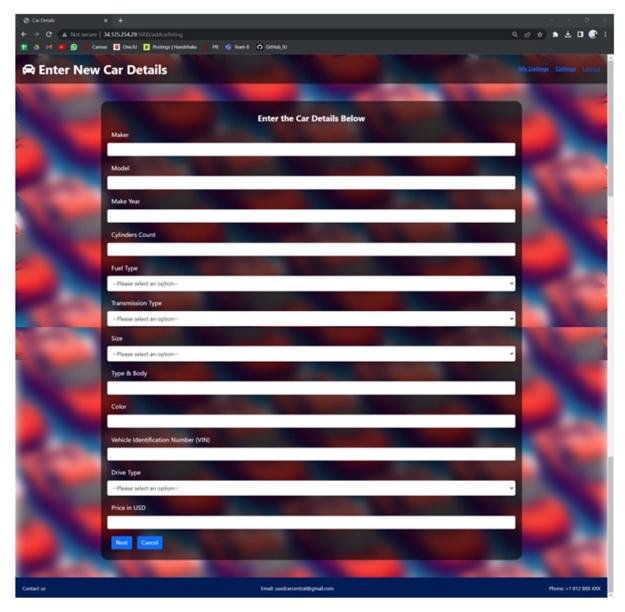

Login Page

User Home Page

View All Listings Page

View My Listings Page

Update a Listing Page

Add a New Listing Page

Section 3: Teamwork Evaluation

Name: Ram Kiran Devireddy (radevir)	
Criteria	Score (1-10)
Task completion	10
Teamwork	09
Time Commitment	10
What could be improved	We could've added more features to our web application like searching, filtering etc. I also wished that we could've designed an even more appealing frontend by using React and Material UI. We are planning to include these in our web app going forward.
What went well	The web-app design and overall architecture design has been fantastic which enabled us to develop more efficiently without deviating from our end goals. On top of that, the implementation of CRUD operations through Procedure calls was fantastic. The collaboration and teamwork had been great, which allowed each of us to be on the same page while designing, developing and documenting. The learning from this project had been top class and the team has done particularly well in completing their tasks in time.
What is my contribution	I have led the backend development of the project by planning the database design and development and business logic of the application in flask. On top of that I have also developed the stored procedure calls and performed Normalization. I have also led the task assignment for our project and also helped our in the documentation.

Name: Syam Prajwal Kammula (skammul)	
Criteria	Score (1-10)
Task completion	10
Teamwork	10
Time Commitment	09
What could be improved	We could've included the chat functionality in our application and also worked more on the security part of our application. We also could've included different views by including an admin user, who approves the newly created car listing's identity before displaying it on the website.

What went well	The database design has been perfect, which helped us to move the same architecture to the front end. This simplified the data organization and code organization.
What is my contribution	I led the front-end development of the application by designing our website architecture. I have also helped designing the frontend and backend integration of our web app. I have also deployed our web app and database.

Name: Revanth Posina (rposina)	
Criteria	Score (1-10)
Task completion	09
Teamwork	10
Time Commitment	10
What could be improved	I thought that we could've organized the html pages in a even more better way which would've helped me in designing the styling more efficiently. Other than that, I also think that we could've used a better frontend technology like Angular or React to build an even better UI.
What went well	The reason that this project was built seamlessly was because of our collaboration and teamwork. Even though we have divided the tasks, we worked together enabling us to know the whole overview on the design of our web app. I particularly like the planning of our application by the teammates which allowed us to meet deadlines and design and excellent web application.
What is my contribution	I developed the front-end styling and behavior while also helping in the database design. I have led the documentation of our project, while also helping out in the database creation.