Phase III: Project Final Report | DATA MANAGEMENT

OntarioTech
UNIVERSITY

Members: Nisarg Bharucha (100820217), Shiv Patel (100818727), Joshua Cardoz (100827231), Fernando Chan Qui (100844946)

CRN: 43511

GitHub Repository Link:
https://github.com/nisargbharucha/FinalProjectDataManagement

Table of Contents

Abstract

This project aims to develop a website application for buying and selling pre-owned cars. It involves creating a comprehensive database that includes information on cars, user accounts, messages, reviews, etc. Real-time updates are crucial, ensuring users can see edits instantly. The chosen technologies are Node.js, Express.js, React.js, and mySQL, working collaboratively for different aspects of the website. User interface abstraction is emphasized to simplify the user experience. Users can access features such as viewing detailed listings, buying cars, listing vehicles with comprehensive information, messaging for negotiations, and leaving reviews. The goal is to seamlessly integrate these features using a mySQL database with appropriate keys, entities, and attributes. A solid understanding of SQL databases is essential to design, query, and display information on the website's front-end.

Introduction

This project aims to create a web-application using relevant industry technologies and a culmination of theoretical knowledge and practical skills acquired during the Data Management Systems course. To apply these skills, we created a web-application that gives users the ability to buy and sell pre-owned vehicles. In response to the complexities inherent in managing diverse datasets and user interaction, our project sought to leverage the principles of database design and management learned in the course to create a robust platform. This report outlines the goals, design, implementation, challenges, and overall analysis of our web application. It also aims to highlight key features that address user authentication, dynamic filtering, detailed car listings, selling functionality, and more.

The primary goal of this project was to design a user-friendly web application that allows for the buying and selling of pre-owned vehicles. The key-goals include user authentication, dynamic car filtering, comprehensive car listings, communication features, and seller reviews. This report will also cover the goal of the project, relationship to the "Data Management Systems" course, the design and implementation choices that were made, challenges that were faced, and the results and analysis in relation to the project.

Goal of the Project

The goal of this project is to create a website application that can be used to buy and sell previously-owned cars whilst employing database management techniques covered in the topics of the "data management systems" course. To accomplish this, we must create a database that includes information about the cars listed on the site, users that have an account, messages that are being sent to each other, reviews, and more. Each of these elements inside the database must also be edited and the users must be able to see the edits that are happening in real time (example: if a user lists a car on the site, it should also show up on other users' screens who want to buy the car). To implement this into a website application, we want to use Node.js, Express.js, React.js, and mySQL as our technologies. All of these technologies are used for a different component of the website and must work in unison to make the website function. We also want a level of abstraction from the user, as they are not interested in specifics of the backend and the database itself. As such, the user interface should give the user access to every intended feature that is on the website.

Users on this website should have access to the following features:

- The ability to view a listing which should include the description, selling price, and more specific information about a car that is listed in the database
- The ability to buy a car that has been listed and is in the database by contacting the user that listed it
- The ability to list a pre-owned vehicle on the site, which includes specific information about the condition, price, and more about the car. These listings should have enough information about the listed car to give an idea to a potential buyer of what they're buying
- The ability to see every listing with the descriptions mentioned above
- The ability to message other users on the website that might be selling a vehicle to negotiate or find a common location to trade the car
- The ability to leave a review on another user's profile to indicate to other users

All of these features should be included in the final product of this website and should work with no issues for the user.

To accomplish all of these features, we must use a mySQL database with appropriate keys, entities, and attributes to hold information. We must also be able to utilize appropriate queries to that database to get certain information based on criteria from the user. To do this, we will require a good understanding of an SQL database, design a good database, create queries for this database, and display information from this database to the front-end of the website.

Relation to the Course

This project directly aligns with the core concepts covered in the "Data Management Systems" course. Before the design of the application itself, we created relational schemas, brainstormed potential views that might be required for this project to function as intended, and created an entity-relationship diagram to get a better understanding of the relationships between entities in the database. We did this to get a better understanding of what we wanted in the database, and how these tables/entities would interact with one another. This leverages knowledge learned in this course as these topics were directly covered in the first three chapters of the course.

Leveraging knowledge related to relational database models, SQL, and database management strategies, we designed a robust schema encompassing tables for cars, listings, images, users, messages, and reviews. We employed the use of relational models within MySQL to manage the data relating to pre-owned cars and the users on the website. We also used the SQL that was learned in the class, including commands to create tables, insert data into those tables, update records, and retrieve information using queries. SQL is crucial for managing and manipulating data within a relational database. We also used JavaScript to interact with the database, as Node.js allows for server-side JavaScript and React.js allows for client-side. The combination of these two allows the client-side to make asynchronous requests to the server, which will in turn access information in the database, facilitating data retrieval and record updates. This project also employs some of the methodologies behind handling large data objects and compression. Overall, this project employs many relevant topics taught within this course.

Design and Implementation

The design process of this web application started with theorizing the tables that would be required to accomplish the required tasks and included information for the features that we intended to add to the system. Once we figured out all the entities that apply to this system, we then decided to split them further, in order to make the database scalable for larger amounts of data. Finally, we finalized the entities that were going to be used in the system; users, listings, images, car, message, and reviews, and created a relational database diagram for our database using these tables (figure 4.1), which also gave us a chance to get the primary and foreign keys for each table.

The next step was to determine the way that these entities and their attributes would relate to one another. The best way to illustrate this was with an Entity-Relation diagram (ER-Diagram), as shown in figure 4.2, which would help us determine which entities relate with what, and other smaller specific information about their relationships. Some of the things that we kept in mind while creating this was the specific relationships between the entities, which include the one-to-many, one-to-one, and many-to-many

relationships. Using these relations, as well as those created by us (e.x. A user creates a listing, or a listing pulls specific car information from the ‘images’ entity), we then connected all the entities and attributes with the relations, the entity-relationship diagram was complete, and we had a better idea on how each table worked with one another.

Now that we had an idea of the database structure, we had to create an SQL database and create a local server for it. For this, we used MySQL, and the MySQL workbench to create a local server where we could hold the database and the tuples that go into them. Once this was done, it was time to create routes to get information from the database, post to it, and just make overall updates to it.

Next, we had to choose the technologies that would be used for this project, and to this, there were multiple factors that we had to think about. Some of these included the group members’ proficiency in certain technologies (how well we know how to use certain technologies) and how these technologies would work to satisfy our needs (e.x. Need something that would work well to search for, delete, and update an SQL database). Eventually, we chose Node.js, Express.js, React.js, Tailwind.css, and the MySQL database we created earlier. Technologies that members were not completely familiar with were understood to a level that was sufficient for the purpose of this project.

Before we started developing the backend server routes and frontend, we decided to design a quick run-down of the frontend and a general run-down of what basic functionalities are available to the user. These diagrams are very simple and are just meant to be a basic illustration of what the user would see. The screenshots included here are design ideas for the car select page, the listings, and the car details page. This was done for all pages on the website to get an idea of what we should design. Using this quick design, we were also able to get an idea of what information from the database was required for each page, for example, the ‘car information’ page requires all information about the car, including its name, description, images, and more. Using this, we could get started on the server routes of our backend and pull the specific information from the database. From this point on, we could start the development of the frontend and backend, which was done concurrently by splitting up roles.

The roles were split so that the frontend development and the backend development was done at the same time. To start, the frontend was created to somewhat match the quick designs that we created above. To do this, we used Tailwind.css for some basic styling. While this was being developed, the backend server routes to get information from the SQL database was being developed as well. An

example of these two sections working together can be seen in the development of the Sell page, where the user is meant to enter specific information about the car in a form. While this user interface was being developed, the server route to save this information into the corresponding tables in the database was being made. Once these two sections were finished separately, time was spent to merge the two and changes were made as required to input information specified by the user into the SQL database tables. Throughout this development, both teams worked together to figure out what information they needed and how it would be inputted into the database tables.

The server routes did a multitude of tasks, both to post and get information to and from the database, respectively. An example of a route that posts is the logic credential, as it sends the inputted information into the database and authenticates with existing information in the database, if it is authenticated correctly, it sends the user back a unique token that can be used throughout the session. An example of a get route in the backend is the route for getting all car information for a specific car from the database based on a unique identifier (carId in our database). These were developed to create or own api, and these routes can be thought of as api endpoints that can be used by the frontend to fetch or post information depending on the task at hand. Some of the views that we used in this code was getting all the information from the car table from the car type, and also getting the images, description, fileName, and imageURL of the **(figure 4.3)**, a view for returning all relevant information from the listings, car, and images table, as well as the information about the seller's first name, last name, and contact information **(figure 4.4)**, and another view that returns all the reviews for a specific listing, as well as the reviewers' first and last name **(figure 4.5)**.

Challenges Faced

While working on this project, we faced many challenges, both technical and otherwise. One of the challenges that we first faced was finding a way to collaborate on the code together, which can be done through GitHub. However, not all of us were proficient in using all the features of GitHub, and setting up the remote repository on everyone's local computers was quite difficult as we ran into some problems along the way. Eventually we got the repositories on everyone's machine locally so we could commit and pull changes as required.

Another challenge that we faced was implementing the ability to message other users on the website. After some research, we found that this feature requires the implementation of socket.io in Node.js, and this was something that was completely new to us. This feature would also require much larger amounts of data, as we wanted to also hold a record of messages that were sent between users.

Another smaller challenge that we faced was that not everyone in the group was completely familiar with all the technologies that we used for this project. Because of this, some time had to be spent learning technologies by going through the specific documentation.

Images also proved to be a very difficult challenge that we faced, as the data-type that we had to use for this feature was 'longblob' in the table. However, although we have used it in the past, it was extremely difficult to implement and for it to run in a reliable manner. A problem that we have previously was that the size of the .png file mattered a lot, and the file was limited to a very small size, one that is not realistic for images that are taken today. Because of this, we had to downscale the image to the point where the image did not look anything like it was meant to. Rather than going with this approach, we then decided it would be more beneficial for the user to input a link to an image of their car on a web-based

server, and pull information from there, rather than save it locally in the database. This method proved to work with no issues.

Overall, we ran into some challenges during every development step of this project, however we were able to find a solution or a fix to most of them, leaving us with a website that includes most of the features that we intended to include from the beginning.

Results and Analysis

This website greets a user with the dashboard screen, which looks something like this **(figure 6.1).**
On the top right, the user can login to their account to see the inventory and create their own listing. If they click on that bell, and they are not already logged on (this is checked by saving their userID in the storage of the browser, if it is still null, that means they're not logged in), they are prompted for their username and password, which is then checked in the database to see if it's an existing user. Alternatively, they can choose to register an account using a username and password of their choosing **(figure 6.2, figure 6.3)**

Once an account has been made and/or they have logged in, they will be returned to the dashboard, where they can select to create a new car listing of their own or buy a listed car. If they choose to buy a car, they will be able to filter their search by the type of car that they want to buy (includes the choices SUV, Coupe, Sedan, or Trucks). This can help the user narrow down their search for the car that they want to buy **(figure 6.4)**

They can select one of the types of cars that they want, and see the listings that are available on the website. From here, they can see pictures of the listings and a quick description of the cars that exist on the database already. However, it does not give all the information about the car, and they would have to click on a specific listing to see specifics about it **(figure 6.5)**.

Once they see a car of their liking, they can select one of the listings in particular, which will then direct them to a page where they can see specific information about the car. This specific information includes the name of it, a description provided by the seller, the odometer, the asking price, the overall condition, and an image of the car so they can get a better understanding of the car. All of this information is held within the SQL database and is unique for every listing. This page also includes a chatting function for the potential buyer to ask the seller to either ask about its availability, ask for more information about the car, negotiate the price, or even to just ask about where they can meet to exchange the car. Finally, there is a section for reviews for the car, and these would be filled out by people that have already gone in-person and have seen the car **(figure 6.6).**

Another key feature in this website is the ability to sell one of your own cars, and users can do that through the dashboard, which gives them an option to sell their own car. This will redirect them to a page where they are given the ability to input information about their cars, including the condition, odometer, description, image, price, and more. This is done using a form, and once that form is submitted, that information is sent to the database in the respective tables **(figure 6.7 and figure 6.8)**.

Future Steps

Although we have decided and implemented many of the features that we had intended, the user experience would be greatly improved with some other functional features. Some of these features include

the addition of saving and using other payment methods securely, and the ability to hold a list of the cars that the user has sent (builds upon the reviews table).

Rather than forcing users to meet up and find a location to do a trade of the car, it would be much easier to make something where the users can do transactions before the car is exchanged. However, this would impose some security in the sense that a buyer could complete the transaction and the seller would not respond and take the money. To combat this in some way, the way we would implement it would be for the website itself to be involved in the transaction, so if something like this were to happen and there was enough proof to show that it did, it would charge back the seller and the buyer would be reimbursed. Another thing that this would allow for would be to create a bidding system on some cars. Some other websites have functionality for users (usually for higher-end items) to put up an item in the form of an auction, where interested buyers would compete to buy the item, and the item would be sold to the buyer with the highest price. This would also bring in a bigger market for people trying to sell higher-end collector-level cars, or just even trying to see how much their car is worth.

Another feature that would be nice to have to further give the buyer an idea of who they're buying from would include a table of previous listings that a user has sold. In addition to the reviews, seeing if they'd bought or sold a car before would help them identify whether or not they have ever used the website, and to further give an idea of the credibility of them. However, although it is very useful, it was not in the scope of this project.

Related Works

Our project draws inspiration from the dynamic world of e-commerce platforms, with a particular focus on revolutionizing the car buying, selling, and rental experience. Taking cues from established websites like cargurus, kijiji, Facebook Marketplace, we aim to take the best features and functionalities to create a seamless and user-friendly platform designed specifically for the automotive industry. Our project strives to enhance the overall experience for both buyers and sellers in the automotive marketplace.

Cargurus was the main inspiration we had for this project, while cargurus it also aims solely on the automotive industry. This website does not provide a proper chat feature inside where user can chat with each other and also they do not provide an online payment method similar to eBay. That is what differentiates us from cargurus.

We also based our project on well known websites such as Kijiji or Facebook Marketplace. However, both kijiji and Facebook Marketplace aim more on the broad market while ours focuses directly on the automotive sector. We took specific features from this website such as, being able to users put a review on sellers and sellers adding reviews about the buyer, users can easily communicate with each other through a chat implemented inside the website.

Conclusion

Our project successfully achieved its primary goal of creating a user-friendly web application for buying and selling pre-owned vehicles. Focusing on the knowledge acquired from the Data Management Systems course, we designed a database using MySQL and implemented it using Node.js, Express.js, and

React.js. The project made use of concepts learned throughout the course such as relational schemas, entity-relationship diagrams, and SQL queries to manage the data related to cars, listings, images, users, messages, and reviews.

The design and implementation process involved accurate consideration of database structure, entity relationships, and technologies. Challenges, such as collaborating using GitHub, implementing messaging features with socket.io, and handling image data types, were managed through research. Regardless of these challenges, we were able to implement essential features to our website, including user identification, dynamic filtering, detailed car listings, messaging functionality, and website payment.

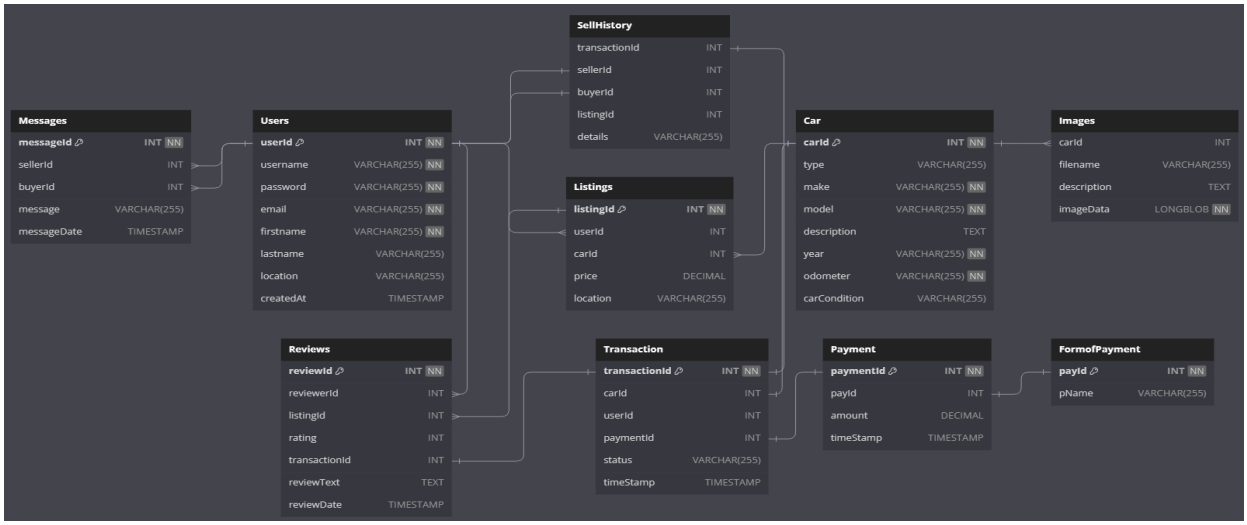
Looking forward, the project leaves potential opportunities for future improvements, such as transaction security method, bidding system, and sellers history. The related works shows inspiration from established e-commerce platforms like cargurus, kijiji, and Facebook Marketplace, emphasizing the project's unique features, such as a built-in chat system and online payment methods, setting it apart from existing platforms.

In conclusion, our project not only achieved its initial goal but also laid the groundwork for potential future improvements. The integration of theoretical knowledge from the course with practical skills in web development resulted in a comprehensive and functional web application for the buying and selling of pre-owned vehicles.

Schematics

Relational Schema (figure 4.1)

This diagram shows all the tables that are required for this project, as well as the relationship between each of them. Each of these tables has its own unique identifier (primary key), as well as foreign keys that link the tables with each other. Using these keys, we can create queries to get specific information from the database with certain constraints. Each of these tables also includes the datatype for each value.

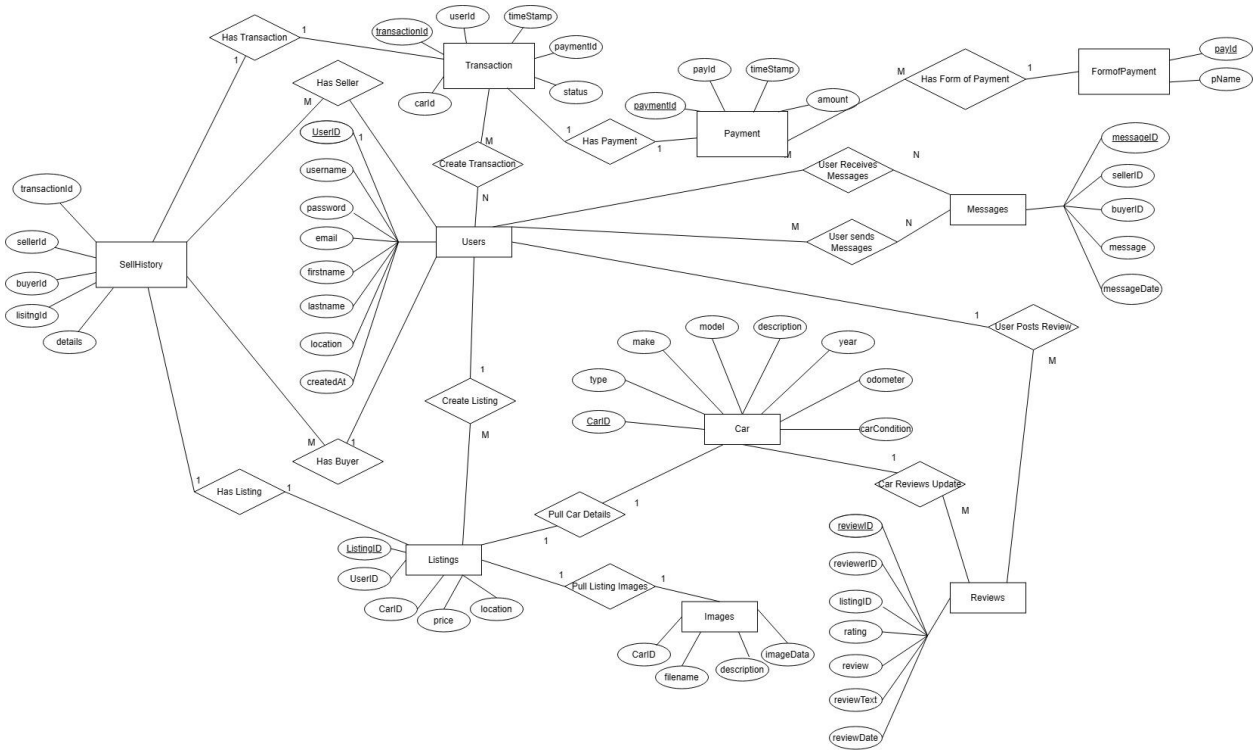


Entity-Relationship Diagram (figure 4.2)

The entity-relationship diagram is often used to create or debug a relationship database. In this diagram, each table is represented by an entity (rectangle), which has its own attributes (ovals

connecting to each of the entities). These entities can be used to describe each of the entities in many ways, and include a unique identifier as well. These unique attributes are underlined in the entity-relationship diagram. The E-R diagram is most useful when trying to get a better idea of the relationship between each entity, which is represented by the diamond. The other way we can show the relationship between entities is by using the one-to-many, one-to-one, or many-to-many relationships to get a better idea of what each entity is used for.

All of these symbols are used to illustrate a general image of the database and how each entity in it relies on each other.



Design Diagrams

```
const [cars] = await connection.query(`
  SELECT car.*, images.*
  FROM car
  JOIN images ON car.carId = images.carId
  WHERE car.type = ?`, [carType]);
```

Figure 4.3: View for getting car information from three tables based on the carID

```
const [results] = await connection.query(`
  SELECT listings.*, car.*, images.*, users.firstname, users.lastname, users.email
  FROM listings
  JOIN car ON listings.carId = car.carId
  JOIN images ON car.carId = images.carId
  JOIN users ON listings.userId = users.userId
  WHERE car.carId = ?`, [carId]);
```

Figure 4.4: View that returns all relevant information from the listings, car, and images table, as well as the information about the seller's first name, last name, and contact information.

```
const [reviews] = await connection.query(`
  SELECT reviews.*, users.firstname, users.lastname
  FROM reviews
  JOIN users ON reviews.reviewerId = users.userId
  WHERE reviews.listingId = ?
`, [listingId]);
```

Figure 4.5: View that returns all the reviews for a specific listing, as well as the reviewers' first and last name.

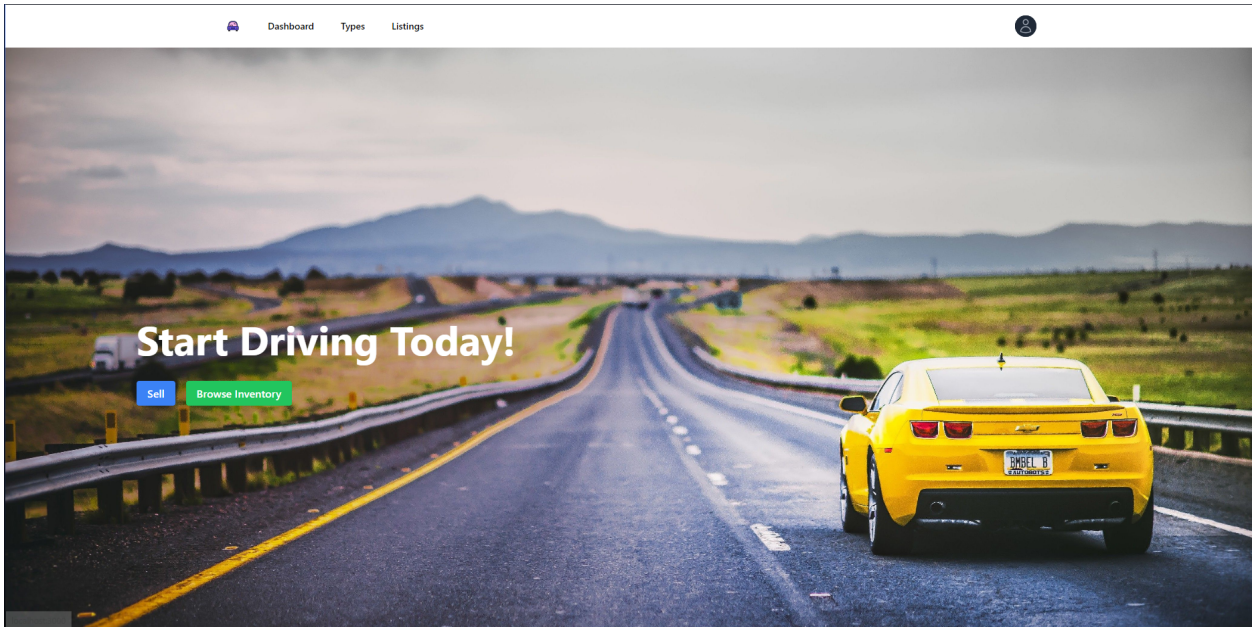


Figure 6.1: main dashboard for user

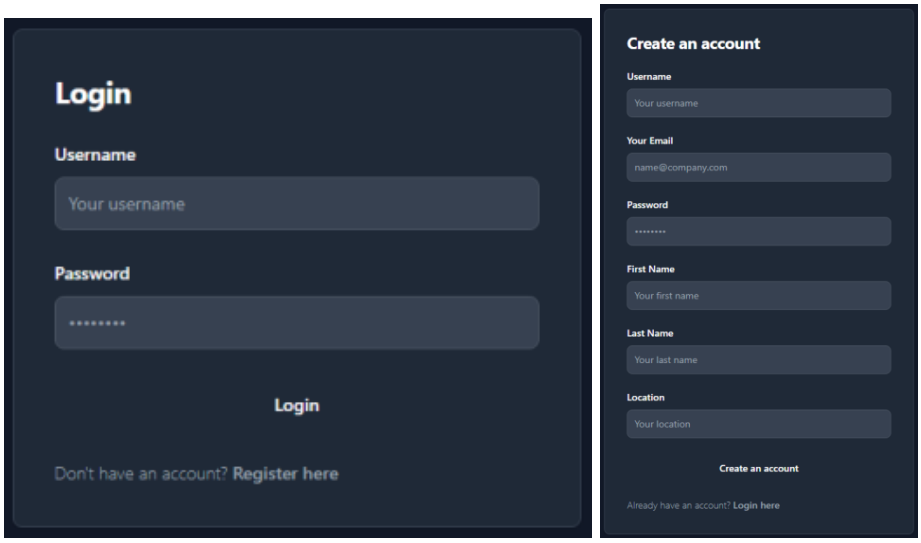


Figure 6.2 and 6.2: User login and registration page

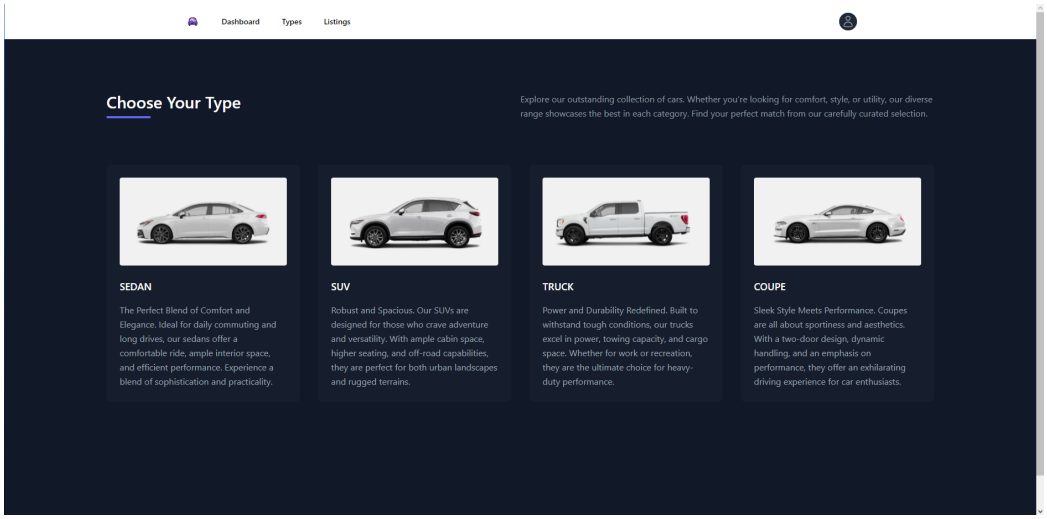


Figure 6.4: Page for buyer to view car based on filters

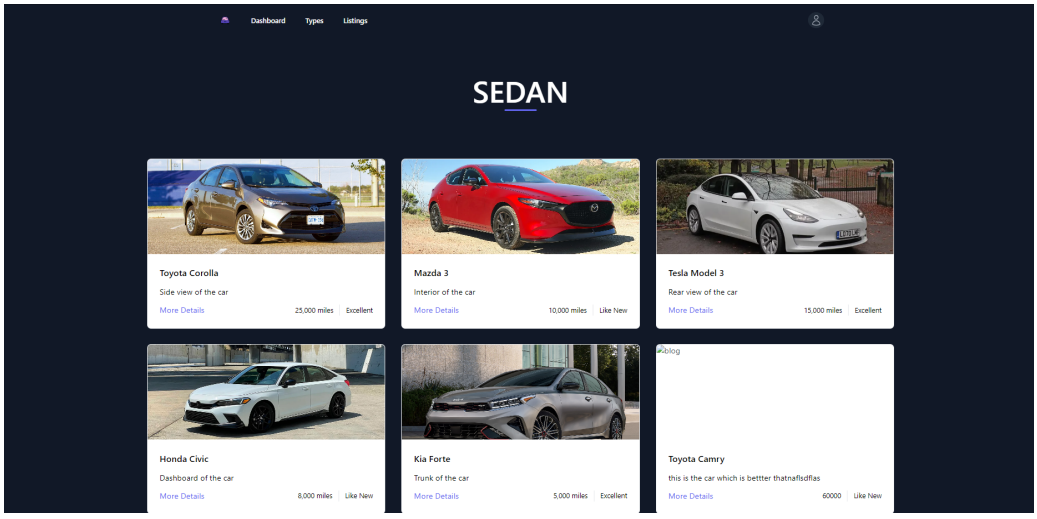


Figure 6.5: Image of all listings after clicking on the "Sedan"

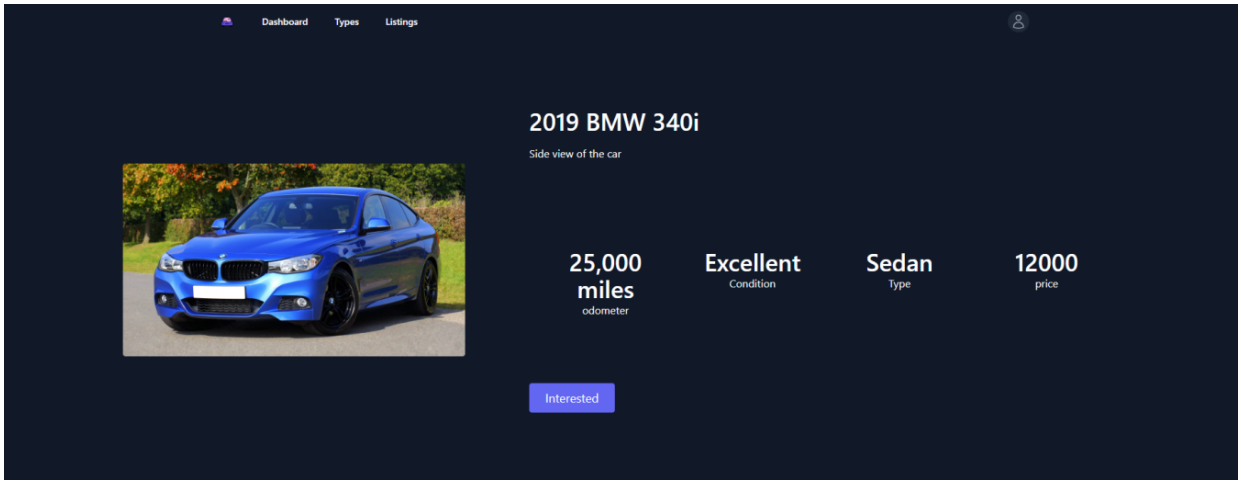


Figure 6.6: Image of the page that includes detailed information about a selected listing

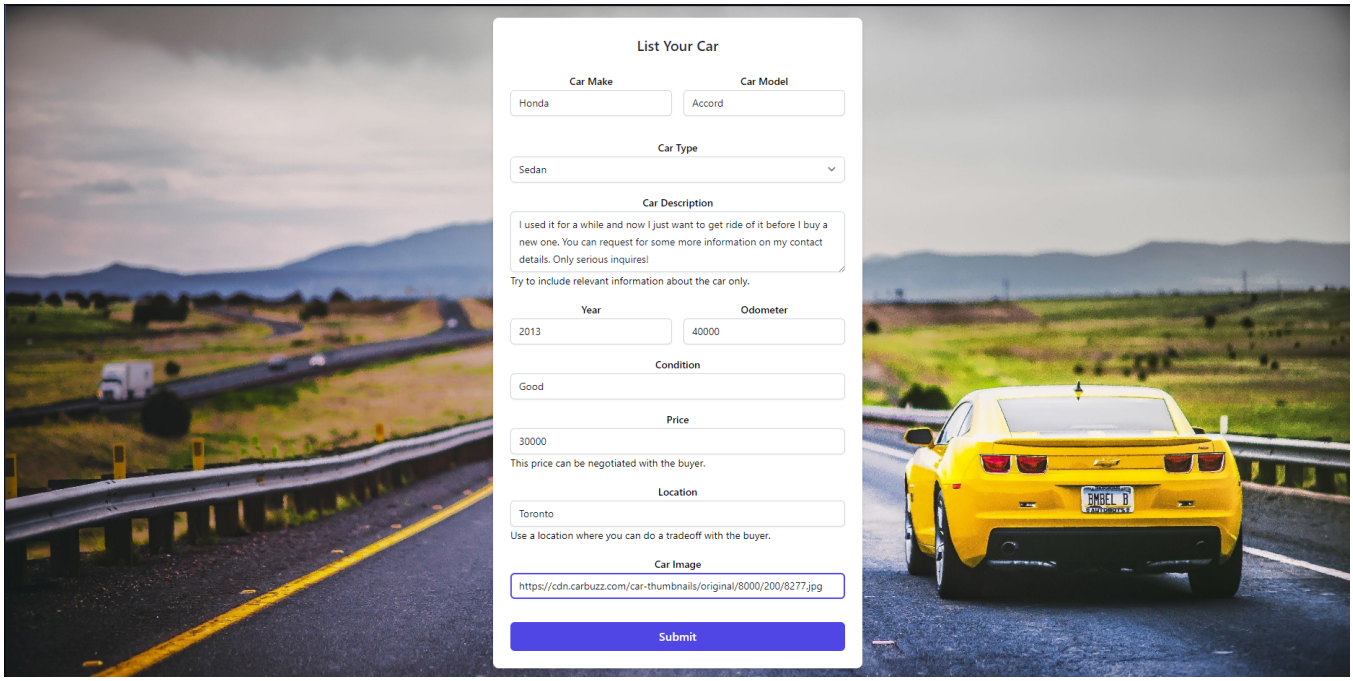


Figure 6.7: Image of the selling form with specific information in it

9	Sedan	Honda	Accord	I used it for a while and now I just want to …	2013	40000		Good

Figure 6.8: Image of the SQL database tables including the information from before

Work Distribution

- Nisarg - Frontend, backend, report
- Fernando - Frontend, backend, report
- Joshua - Frontend, backend, report
- Shiv - Frontend, backend, report

References

Kilic, S. (2021). Socket.io with Node.js + Express. Retrieved from https://medium.com/kocfinanstech/socket-io-with-node-js-express-5cc75aa67cae

Nalimov, C. (2020). Diagram maker for developers. Retrieved from https://www.gleek.io/blog/relational-schema

What is an entity relationship diagram (ERD) (2020). Retrieved from https://www.lucidchart.com/pages/er-diagrams