Divya Bendigeri, Rishab Shah, Soundarya Sivakumar, Sharon Newton

CS 411 Final Report

Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).

Our application allows rental businesses and customers to login using their GarageId or CustomerId, respectively, and interact with different screens based on their role. For example, the rental business owner can track the availability of all vehicles in their garage and manage ongoing or future rentals. On the other hand, customers can enter the city where they want to pick up the rental car and view a list of garages and available cars to choose from. After selecting a car, the customer can proceed with creating a rental by entering information such as start and end times, and later complete a customer satisfaction survey on a separate page.

The functionality of our app mostly aligns with our original proposal and matches the UI mockup we designed during the first stage. However, there were some changes and simplifications that were made during the web development process. Originally, we intended for customers to be able to filter vehicles by additional attributes like make, model, and hourly rate. In the final version, filtering is now limited to a city-based search.

Discuss what you think your application achieved or failed to achieve regarding its usefulness.

In terms of usefulness, RentHubDB successfully met many of the key goals we outlined in Stage 1. The platform provides a functional and user-friendly interface, enabling customers to browse available vehicles in their desired location, create rentals, and leave feedback on their experiences. The ability to track rental history and update vehicle availability in real-time significantly enhances the app's practicality for both customers and businesses. Additionally, the separate login pages for customers and businesses ensure a more personalized, role-specific user experience. For businesses, RentHubDB offers essential functionality, such as the ability to monitor vehicle availability, view active rentals, and manage operations more efficiently. Furthermore, we successfully implemented a customer profile page, allowing users to update their personal details, including age, email, name, and phone number.

However, there are a few areas where the application did not fully align with our initial vision. While we had planned for customers to filter vehicles by multiple attributes, such as make, model, and hourly rate, we were only able to implement city-based filtering in the final product. Another feature we initially proposed was an interactive heat map that would display areas with high rental demand based on real-time booking data. This would have helped businesses identify peak demand regions and adjust rental rates accordingly. Unfortunately, due to technical complexity and time constraints, the heat map feature was not developed.

Additionally, some features for the business login page were not implemented as originally planned. Businesses are currently unable to track vehicle availability, monitor vehicle mileage, or optimize the distribution of cars across multiple locations. This was a setback in terms of usefulness, as we had hoped to allow businesses to borrow vehicles from one another based on customer demand. While this feature was not implemented, we were still able to provide businesses with the ability to track rental data by viewing rentals at specific garages in a certain city.

Discuss if you changed the schema or source of the data for your application

We stayed consistent with the schema of the data for our application but there were many changes we had to make to the actual data as we traversed through this project. First of all, in our CarData source we were given the HourlyRate of many cars, but not all, so we predicted and generated the missing hourly rate values of the cars based on their GarageId, Make, Model, and Year features using a Random Forest Regressor. Additionally since all of the tables did not already have a uniquely identifiable ID number (other than GarageLocation since that table did not need an ID), we created unique ID numbers by assigning each table a 10000 value range - e.g. CarId: 10000-15181, Garages: 20000-25852, etc. This way, we are able to uniquely differentiate between entities' categories. Also for some unknown reason, there was a missing EndTime value in Rentals that we had to ultimately drop due to the errors it was causing us in uploading our table data.

Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?

In the final UML diagram, several improvements were made from the original design. By implementing these changes, we were able to enhance the database, enforce stronger relationships, and improve clarity. First, we cleaned up some foreign key connections to better match the logical flow of the data. In the original diagram, some relationships were missing primary and foreign key indications. In the final UML diagram, for the relationship between Rentals and CustomerSatisfaction, we made sure to show clearly that RentalId serves as the primary key for feedback. This guarantees that the customer satisfaction feedback is tied uniquely to one rental. Second, we simplified attribute structures in some tables. Originally, the Rentals table had a garageId and CarId directly inside it. However, in the final version, cars are parked in garages, and garages are located in a city. This separation better supports the idea of garages serving multiple cars.

We also removed redundant attributes in our final UML diagram. For example, the original Garages table included latitude and longitude, but now these are properly moved to the GarageLocation table. Garages only contain financial and operational attributes now, which keeps the location data separate. This normalization reduces data duplication and redundancy. Additionally, we enforced stronger cardinality constraints. In the final diagram, it's clearer that each rental must be associated with exactly one car (1..1) and that each customer can have zero or more rentals (0..*). Similarly, cars are now properly shown as being parked at a garage, and garages can have multiple cars parked there.

Discuss what functionalities you added or removed. Why?

Throughout the development of our application, we made several adjustments to the functionalities that were originally planned. On the customer side, we successfully implemented the ability to search for available vehicles by city, create rentals, submit customer satisfaction surveys, and have customers update their profile. However, we did not complete all features due to complexity and time constraints. One function that we specifically removed was the ability for customers to filter available vehicles by made, model, and hourly rate. We made this choice because filtering by make, model, and hourly rate was not essential or necessary, because customers could easily search for that information after viewing the list of available cars for the city of choice. Limiting the search filter to only cities helped simplify the application and make it easier for users to choose a vehicle. One functionality that we added to our original proposal was having the customers also make changes to their profile, including name, age, email, and phone number.

On the business and administrative side, we implemented several features that enhanced operational management for rental companies. The businesses have the ability to select a city and view all the garages in that city. They can also monitor the rental activity in real time, since they are able to see the number of rentals and the RentalId of the most profitable rental in a garage. However, there were many functionalities that we could not implement because of time constraints. We originally said that administrators could track vehicle availability, monitor mileage, and optimize the distribution of cars throughout multiple locations. However, these functionalities were not implemented in the final application.

Explain how you think your advanced database programs complement your application.

In the earlier stages of development of our application, we all designed advanced SQL queries to support the functionality for both customers and administrators. These queries included identifying the most profitable garages, displaying the total revenue for each car, calculating total spending by customer, and showing the average mileage of all trips for one customer. The advanced queries organized and presented important and relevant data in ways that directly supported both customers and businesses.

Later in the project, we added a stored procedure, constraints, and triggers to the database to further enhance the application's backend functionality. The stored procedure, GetCustomerRentalSummaryById, generates a comprehensive summary for a given customer, including their total spending on rentals, average rental rating, and their rank among all customers based on how much they have spent. By embedding this logic into our database, we were able to reduce the need for complex calculations. We were also able to improve performance when retrieving important customer data. The stored procedure complements the system by enabling fast and organized access to business data, which supports customer management as well as business strategies. In addition to the stored procedure, we also implemented constraints and triggers. Constraints, such as enforcing valid mileage and non-null rental dates, help maintain the accuracy and reliability of the data. Triggers can automatically update related fields, ensuring that business rules are consistently applied without requiring manual input by an administrator.

Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.

Divya

One significant technical challenge that our team faced was working with MySQL Workbench, particularly when managing the advanced queries and indexing operations. We frequently encountered persistent issues with this interface, including unstable server connections. Sometimes, MySQL Workbench would unexpectedly crash or disconnect, especially when creating or modifying indexes. These technical interruptions significantly slowed our progress and made troubleshooting very difficult. For future teams that are starting a similar project, we recommend exploring alternative platforms for working with MySQL databases. Other options might offer more stability and flexibility, especially for collaborative workflow.

Soundarya
Another technical challenge that we faced is while importing the data from the csv file for Customer Satisfaction to MySQL Workbench. When we selected all the rows from the table, there were only two rows in the output. However, when we checked the data after exporting it to a csv, we could see all the records. Eventually, we figured out that the issue was due to the presence of commas in the Comments field. While reading the data, MySQL probably thought the commas within the comments indicated a new row so this caused the issue. Once we figured this out, we replaced the commas in the text to hyphens so that there would be an issue. This fixed the problem and SELECT * FROM CustomerSatisfaction returned the right number of rows.

Rishab
Another key technical challenge our team encountered was enabling external access to our Node.js server running on the GCP VM. After deploying the server (using `node server.js`) and reserving a static external IP, our expectation was that the application would be accessible through a browser using that IP address. But, all attempts to externally open that site failed. So after investigating the issue further, we found that the issue stemmed from Google Cloud's default firewall settings, which block incoming traffic on the default HTTP port (port 80) unless it is explicitly allowed. So to resolve this, we created a new firewall rule in GCP that allowed traffic on port 80 from any IP address (0.0.0.0/0). This made the application successfully accessible, but this also introduced a very major and serious security risk; by allowing this open access from any IP address, we were potentially exposing our server to unauthorized accesses. A serious recommendation for future teams in a case like ours where their application does not require global access is to completely avoid exposing their VMs to all IP addresses. Instead, they should configure the firewall to only allow their trusted IP addresses or those that are being used and, for even safer development tactics like we did, run and test the application locally on your machine (localhost) instead of on a public server.

Sharon
We encountered another significant technical challenge when importing the data for the Rentals table. This table includes the StartTime and EndTime attributes that store the data and time of when the rental started and ended. Originally, our csv stored these values in the following format which is the default Excel datetime format: 1/6/2016 5:15:00 PM. When importing the csv into MySQL, all the StartTime and EndTime values were null because the system could not parse the data format. After doing some research, we discovered that MySQL uses the following format to store datetime values: YYYY-MM-DD HH:MM:SS. To address this issue, we had to create a python script that parsed the csv with our original data and wrote a new csv containing the datetime values in the correct MySQL format. Using the script to convert the csv allowed us to successfully import the data into Workbench. We would advise future teams to make sure

that the original datasets are properly formatted in the MySQL format. Checking the format of each attribute thoroughly will save time and streamline the data importing process.

Are there other things that changed comparing the final application with the original proposal?

Other than everything mentioned in the above sections, the final application aligned very well with the original proposal. No major changes were made to the proposal that we submitted in Stage 1, except for those highlighted earlier in this report. We hoped to add more functionality to the business side of the application, but this was not possible due to time constraints.

Describe future work that you think, other than the interface, that the application can improve on

In the future, RentHubDB could be expanded by adding more advanced features to improve both the customer and business experience. For customers, we can introduce a loyalty rewards program as an incentive for regular customers. This way, some customers will be able to get discounts or free upgrades based on their usage and how many rentals they reserve. Additionally, businesses could benefit from having dynamic pricing based on current demand. This is similar to airline or hotel pricing models, and it will help maximize revenue. Incorporating predictive analytics into the application would also allow for businesses to forecast rental demand based on holidays or local events in the location of interest. Ultimately, all of these improvements could help RentHubDB evolve into a more intelligent and secure platform for small car rental businesses.

Describe the final division of labor and how well you managed teamwork.

All 4 of us contributed with the writing portions of the initial idea proposal, and the database design portions in Stages 1 and 2. Rishab handled the data sourcing from Kaggle and data generation for missing values or additional features using Python. During Stage 3, all 4 of us split up the work pretty evenly. We each created one MySQL query and utilized MySQL workbench to capture the resulting data. We also each created two indexing designs using MySQL Workbench. Due to Python installation issues, Divya was not able to work on the backend part of this project, which was the majority of the application development. In order to somewhat make up for this loss, she contributed by writing almost all of the written report and making the slide deck and bullet points for the application pitch.

For Stage 3, Sharon and Soundarya were responsible for creating the tables in MySQL and importing the data correctly into the database. They and Rishab worked together to debug any issues encountered with data formatting and importing. For Stage 4, Sharon was responsible for implementing almost all parts of the website. She coded the login, city selection, rent form, after rental, and business home pages. She created advanced queries for the transactions, created a trigger, and implemented constraints during table creation. She also tested the website numerous times and debugged errors in both the frontend and backend files. Throughout the semester, Sharon was responsible for submitting all releases. Soundarya was responsible for connecting the html pages and ensuring that data is passed from one page to the next in a sequential manner. She made sure that the CustomerId and CarId is stored and tracked throughout the navigation of the website by modifying the url and how the parameters are passed in the backend for each route. In addition, she developed the customer home page, where there are buttons that redirect to the customer profile page or starting a rental. In addition, she created 2 stored procedures and called them so that the results are displayed on the home page. Rishab was responsible for handling the stored procedures along with helping with the connection between front end and back end. He made sure that the stored procedures were accurate and displayed correctly. Being one of the last prevalent issues in this project, he worked to debug this issue on both the backend and frontend fronts until it worked and we as a team were able to successfully deploy a fully-functioning application!