The Auto Shop

**R1**

The Auto Shop is going to be a web-based service which allows users to both post their cars for sale and browse through cars for purchase. The sample dataset for this project will be created by hand using common knowledge and a few sample descriptions and images. The real dataset will be created through a program (will be accessible through the Git) using either a generative text API or some form of random but controlled generation. The administrators will be the Technology Operations team at The Auto Shop (just Araad Shams in this case), and the users will be any individual who has a car to sell or who needs to buy a car.

In specifics, this application will provide a user-to-user car selling service. Before making the purchase, the buyer will be able to see a description about the car, the company which made the car, the model name of the car, the make year of the car, the price of the car, contact information for the seller, and the odometer reading of the car. Additionally, the user will be able to see other cars made by the same company, other cars made in the same year, other cars near the same price range, and ratings/reviews of the seller. Additionally, the buyer will be able to see cars within a certain radius of his location. The seller will also be able to edit and delete their posts. The buyer will be able to review the seller provided that they have purchased from the seller.

**R2**

The Auto Shop will be built a text-based interface. This is to make it so that **other people** who would like to make online ecommerce platforms can use our service, almost as an API, that they can query to see cars and perform various operations. A graphical interface was becoming a little too overwhelming for this project, so a text-based version with all the same features and components will be made. Created locally using Python. There will also be a back-end server management system with NodeJS running locally, which maintains the queries and allows us to use simple GET and POST methods.

**R3**

For the sample data, it will be manually hand entered and the plan of the attack for the real data that will populate the database is to create a program that, either through some Generative Text processing or through some randomization algorithm created, gives us appropriate entities and tuples that can work together and populate the database. They will be created according to the schema shown in the next section.

**R4**

For the production database, I will be using the following recommended dataset (<https://www.kaggle.com/datasets/austinreese/craigslist-carstrucks-data>). The CSV file has been firstly, significantly reduced as the original file size would have slowed everything down. Next, we have kept the following rows.

**Table

Description automatically generated**

This is for the cars table. Around 1700 entries from the given dataset are loaded in. For the other tables, some random values were generated which **correlate** properly to the cars table. They can all be found in the excel file attached to this submission. At the end, all of these lines were mapped to INSERT statements. (All visible in the excel file)

**R5**

**Diagram

Description automatically generated**

**Additional Constraints:**

* Every review with a bid + sid pair must have a corresponding bid + sid pair in Sale

**Diagram, schematic

Description automatically generated**

**R6a – Interface**

This feature will be a basic search filter. For instance, if a person who is looking for a car has certain requirements on the type of car he is looking for where it may be made in a certain year, have less than a certain number on the odometer, be made by a certain company, etc. In the main page, the user will have certain check boxes and text inputs below the search bar that let them choose the filters, and then upon hitting an apply button, the query will run and the page will update to show cars that meet their requirements.

**R6b – SQL Template**

CALL AutoShopDB.get\_filtered\_cars(2012, 10000000, "Atlas");

This uses the stored procedure defined as below:

PROCEDURE `get\_filtered\_cars`(IN make\_year int, IN odometer\_max int, IN make\_wanted varchar(45))

BEGIN

SELECT \* from AutoShopDB.Car WHERE (make\_year = 0 OR year >= make\_year) AND (odometer\_max = 0 OR odometer <= odometer\_max) AND (make\_wanted = "" OR make = make\_wanted);

END

**R6b – Generated Output**



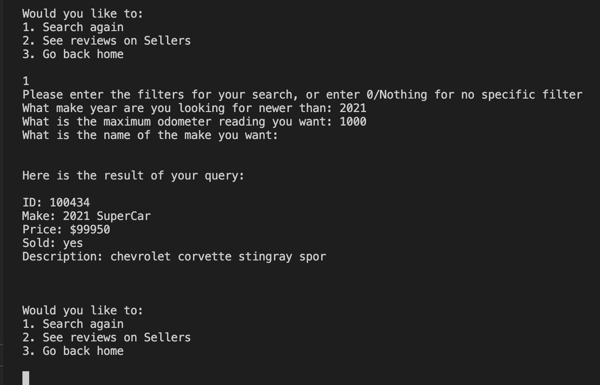
**R6b – Generated Output Text**

# cid desc img\_file price odometer sold sid mid make year

2 slow :( img.png $15.99 100 yes 1 2 Atlas 2012

**R6d – Testing with Production Database**

**Text

Description automatically generated**

**R6c – Implementation and Query Testing**

Graphical user interface, table

Description automatically generated with medium confidence

**Text

Description automatically generated**

A screenshot of a computer

Description automatically generated with medium confidence

The implementation in the UI can be seen in the *src* folder, in the main python file!

**R7a – Interface**

This feature will be a basic reviews selection on the seller. Based on the car that the person is looking at, on the same page, reviews about the seller will be queried and shown with date and timestamps at the bottom of the page. The user’s rating will also be shown next to their profile name. This will allow the buyer to make a more informed purchase and they will know how trustworthy the person they are dealing with is

**R7b – SQL Template**

CALL AutoShopDB.get\_reviews(1);

This uses the stored procedure defined as below:

CREATE PROCEDURE `get\_reviews` (IN car\_id int)

BEGIN

SELECT \* FROM Reviews WHERE sid = (SELECT sid FROM Car WHERE Car.cid = car\_id);

END

**R7b – Generated Output**

Table

Description automatically generated

**R7b – Generated Output Text**

# rid review rating date bid sid

1 My favourite person in the world! 5 2015-12-10 00:00:00 1 1

2 What an amazing selllerrrr 5 2023-02-28 03:19:09 1 1

**R7c - Implementation and Query Testing**

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

**R7d – Implementation and Testing**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

The implementation in the UI can be seen in the *src* folder, in the main python file!

**R8a – Interface**

This feature will allow a buyer to buy a car from a seller. Once the car has been bought, the buyer will then be allowed to comment and write a review about the seller by navigating to the seller’s profile page and, once recognized that a valid transaction has been made between this buyer and seller, the buyer will be able to leave a review about the seller.

**R8b – SQL Template**

CALL AutoShopDB.add\_comment("What an amazing selllerrrr", 4.9, 1, 1);

This uses the stored procedure defined as below:

CREATE PROCEDURE `add\_comment` (IN review varchar(3000), IN rating decimal, IN buyer\_id int, IN seller\_id INT)

BEGIN

INSERT INTO AutoShopDB.Reviews VALUES (NULL, review, rating, NOW(), buyer\_id, seller\_id);

END

**R8b – Generated Output**

N/A, Insert Statement (will show some confirmation on the website UI though)

**R8b – Generated Output Text**

N/A, Insert Statement (will show some confirmation on the website UI though)

Graphical user interface, text, application

Description automatically generated

**R8c – Implementation and Query Testing**

****

**R8d – Implementation and Testing**

**Text

Description automatically generated**

**Text

Description automatically generated**

**R9a – Interface**

This feature will allow users to only show sellers who are in a certain radius of their location. Using Long, Lat coordinates (determined at time of sign up), the buyer will be able to, on the main page, define a radius in which they want to search in, and then the SQL Query will remove all results that are outside of said radius.

**R9b – SQL Template**

CALL AutoShopDB.show\_sellers\_within(100, 1);

This uses the stored procedure defined as below:

CREATE PROCEDURE `show\_sellers\_within` (IN distance int, IN buyer\_id int)

BEGIN

SELECT \* FROM Car WHERE ABS((SELECT location FROM Buyer WHERE buyer\_id = Buyer.bid) - (SELECT location FROM Seller WHERE Seller.sid = Car.sid)) <= distance;

END

**R9b – Generated Output**



**R9b – Generated Output Text**

# cid desc img\_file price odometer sold sid mid make year

1 A FAST CAR img.png $15.99 100 not 1 1 Sienna 2022

2 slow :( img.png $15.99 100 yes 1 2 Atlas 2012

**R10a – Interface**

This feature will allow users to modify comments that they have written for cars that they have purchased. Comments that they have written can be modified at any time and the comment will be flagged as edited.

**R10b – SQL Template**

CREATE PROCEDURE `modify\_comment`(IN commentId int, IN commentStr varchar(3000))

BEGIN

UPDATE AutoShopDB.Reviews

SET review = commentStr

WHERE rid = commentId;

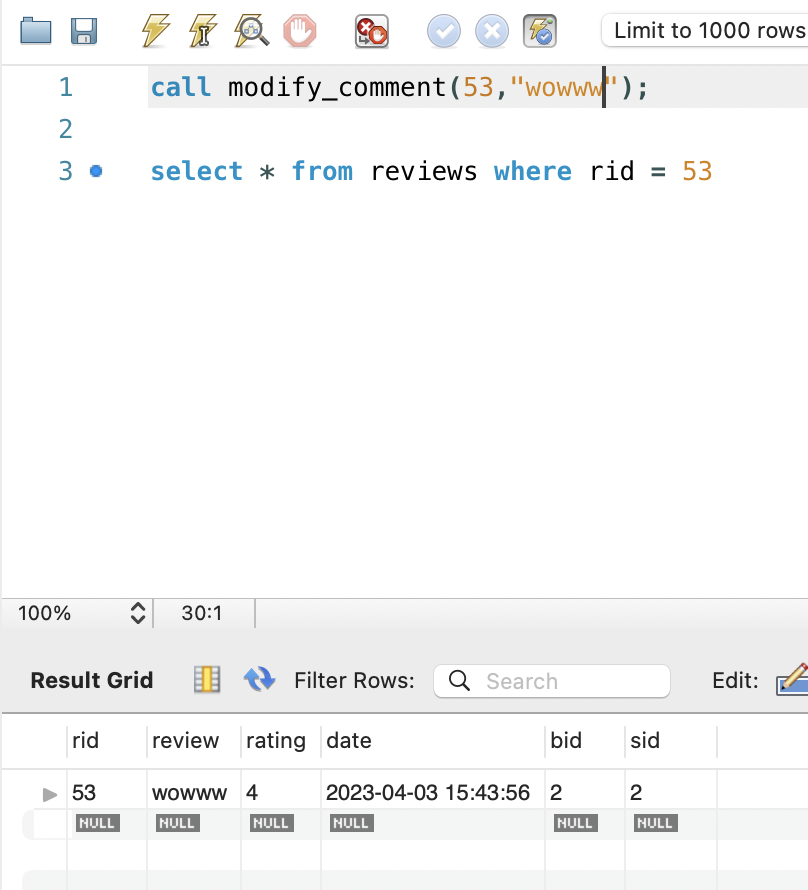
END

**R10b – Generated Output**

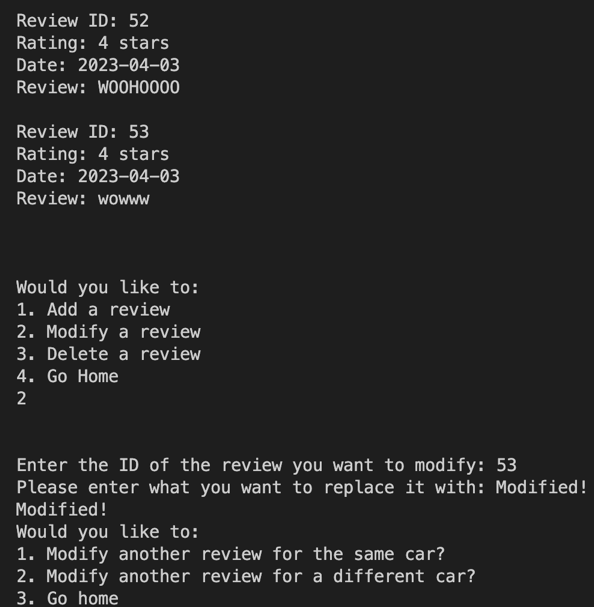
Graphical user interface, text, application

Description automatically generated

**R10c – Implementation and Query Testing**

****

**R10d – Implementation and Testing**

****

**Text

Description automatically generated**

**R11a – Interface**

This feature will allow users delete comments that they have written in the past. These comments will simply be deleted and no record of them will be kept anywhere in the database. A user can see their comments for a specific car and delete them when they would like to.

**R11b – SQL Template**

CREATE PROCEDURE `delete\_comment` (IN commentId int)

BEGIN

DELETE FROM AutoShopDB.Reviews

WHERE rid = commentId;

END

**R11b – Generated Output**

**Graphical user interface, text, application

Description automatically generated**

**R11c - Implementation and Query Testing**

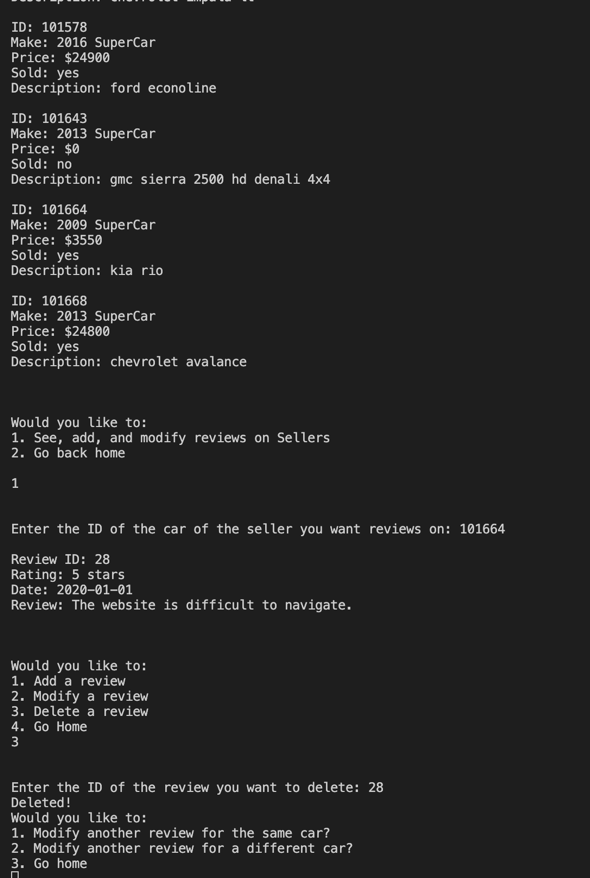
**Table

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

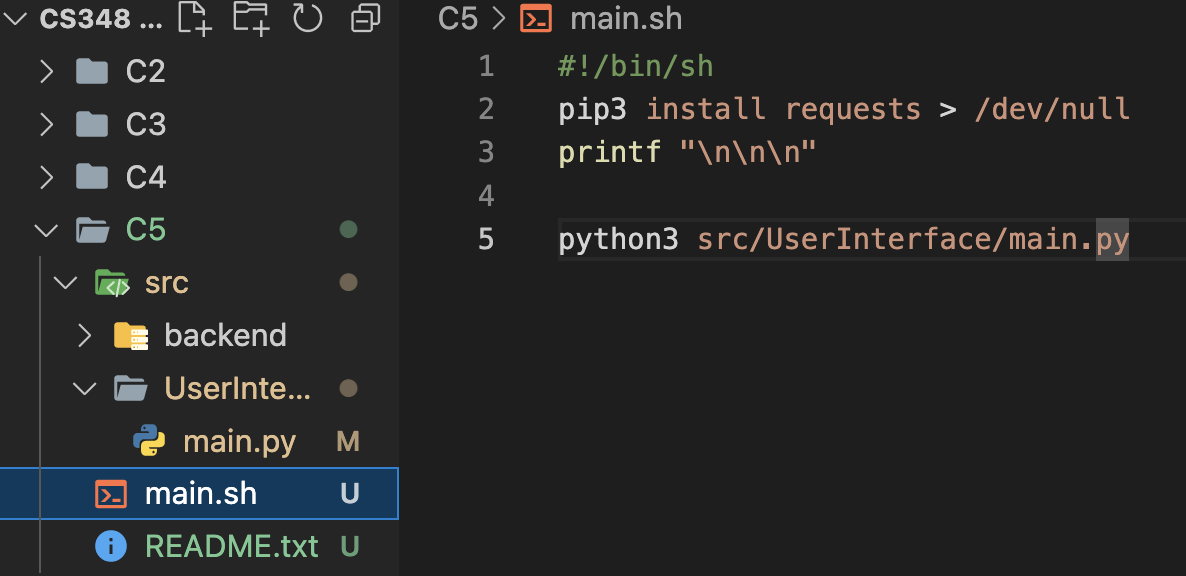
**R11d – Implementation and Testing**

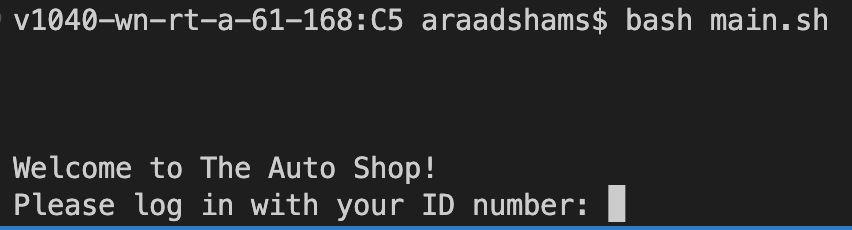
****

The implementation in the UI can be seen in the *src* folder, in the main python file!

**R12 – Ready to Go**

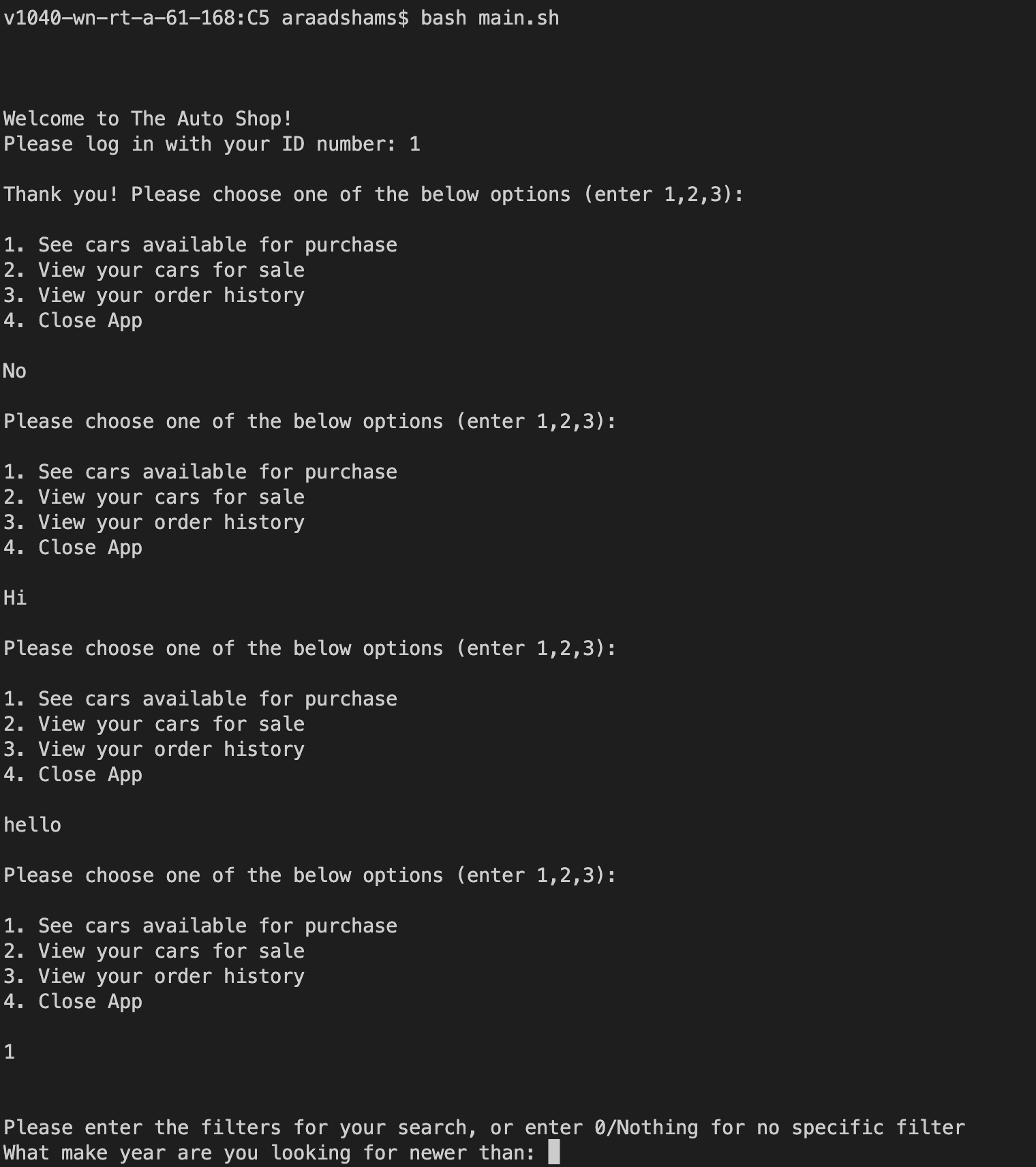
The first “fancy feature” implemented is the single-line run. There are many components to this project which much be run so that the user can use this service. But I’ve actually created a **bash script** which runs all requirements and then runs the project as well. This script installs all of the dependencies, and then proceeds to start the python script which starts the backend database and runs everything. The messy outputs are also routed to /dev/null so that the user is not bothered with any messages they don’t need. This is done to make it easy on the user or anyone implementing this program so that running the program is as simple as a single command.



****

**R13 – Invalid Input Filter**

The second, seemingly trivial but necessary feature, is to reject any type of invalid input at every stage. This allows the user to be worry-free that the program will ever crash or get stuck somewhere while they are in the middle of something. It creates a closed-loop so that no matter what the user inputs, they cannot break out of the loop.

****

**R14 – Concurrent User Access**

This can clearly become a problem for this database. As a result, it has been created in a way that each python script is directly always interacting with a live version of the database. This is not ideal for scaling, but it will ensure atomicity for each transaction and data integrity. Ideally in the future for scaling, this database will follow common concepts and practices which will allow for it to be better at maintaining its atomicity and integrity while also being efficient

**Graphical user interface, text, application

Description automatically generated**

**R15 – Flow of UI**

The flow in the UI is designed with a flowchart idea in mind so that the user can only see the commands that are applicable to them after executing a certain command, and so that especially in a text-based UI, they are not overwhelmed with information. This way, the user will be able to access all the functionalities of the app without being over-confused.

Text

Description automatically generated

**Text

Description automatically generated**

**Text

Description automatically generated**

**R16 – UI Appearance**

While the UI is text-based which is not ideal, it has been created in such a manner with tracked spacings so that the user will find a lot of comfort in using it. They will always feel a good flow when working and when reading in information, they should not get confused. All parts of the UI are well and equally spaced and text is displayed in nice, readable structures. Input prompts are very clearly indicated and easy to understand.

Text

Description automatically generated

Text

Description automatically generated

**R17**

This milestone wraps up all of the content. I made sure to implement almost everything required and touched up the text-based UI. This milestone has the final product which is functional and ready to be used. The last features were implemented and tested with PROD data, and added to the report. Fancy features were added, pointed out, and emphasized. Project is complete.