INTEGRATING APPLICATIONS HOSTED ON EC2 WITH RDS USING THIRD-PARTY TOOL

AUGUST 24

Devamadhav Kattimuttathu

INTEGRATING APPLICATIONS HOSTED ON EC2 WITH RDS USING THIRD-PARTY TOOL

Problem Statement

The goal is to deploy a scalable, cloud-based solution for a car manufacturing and dealership ecosystem using Amazon Web Services (AWS). The solution involves three interconnected applications:

- App1 - Car Manufacturing: This application manages car manufacturing processes. It requires deployment on an AWS EC2 instance with a database hosted on Amazon RDS.

- App2 - Car Dealership: This application handles car dealership operations and needs to be integrated with App1 to fetch manufacturing data. It will also be deployed on an AWS EC2 instance with a database on Amazon RDS.

- App3 - Car Details: This application provides detailed information about the cars manufactured and available for sale. It connects with App2 to fetch dealership data and with App1 for manufacturing data. App3 does not require a database but will be deployed using an EC2 container on AWS.

Observations and Screenshots

APP 1

Initially, we created an RDS database named app1 and selected MySQL as the database engine, choosing the free tier. We then set up an inbound rule in the security group to allow MySQL/Aurora connections. Following that, we launched an EC2 instance running Windows.

After setting up the EC2 instance, we connected it to the RDS database. We accessed the instance using the key pair generated during its creation. Once logged in, we installed SQL Workbench and XAMPP on the EC2 instance and started the XAMPP services.

Next, we configured SQL Workbench to connect to the RDS database using the provided endpoint. We created a new schema in the database and imported the necessary file into it. Additionally, we copied the contents of the app1 folder into the htdocs directory of the EC2 instance.

When we accessed the website via localhost/app1, it prompted us for a username and password and successfully logging into the website. We added a car brand named "BMW," and it appeared correctly in the car brand list.

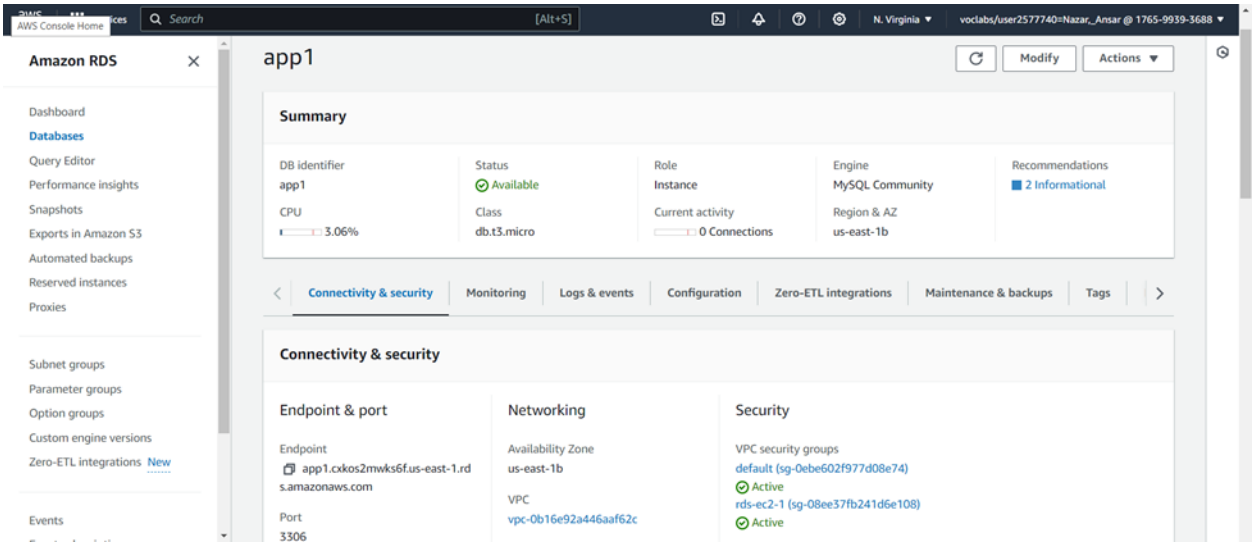

Figure 1 RDS database with name app1 created.

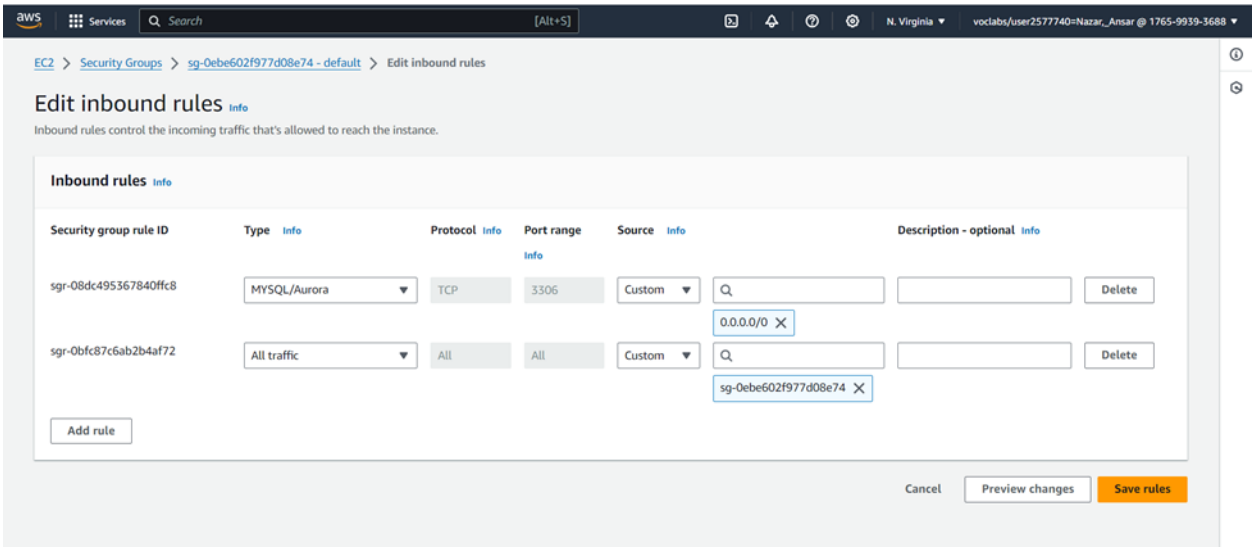

Figure 2 RDS database security group rule.

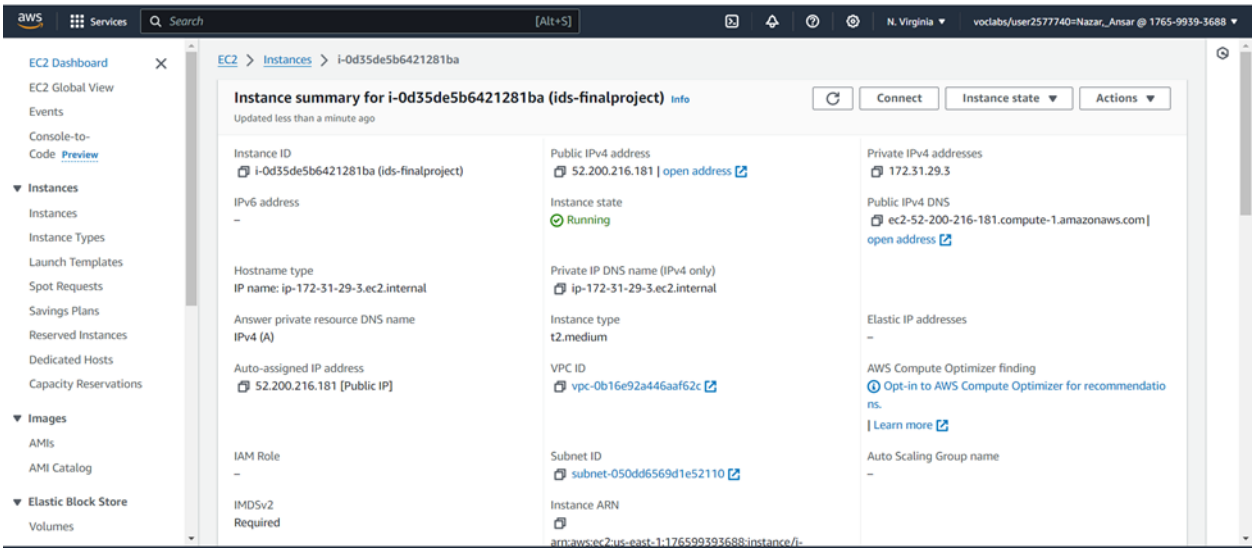

Figure 3 An EC2 instance with the name ids-finalproject created.

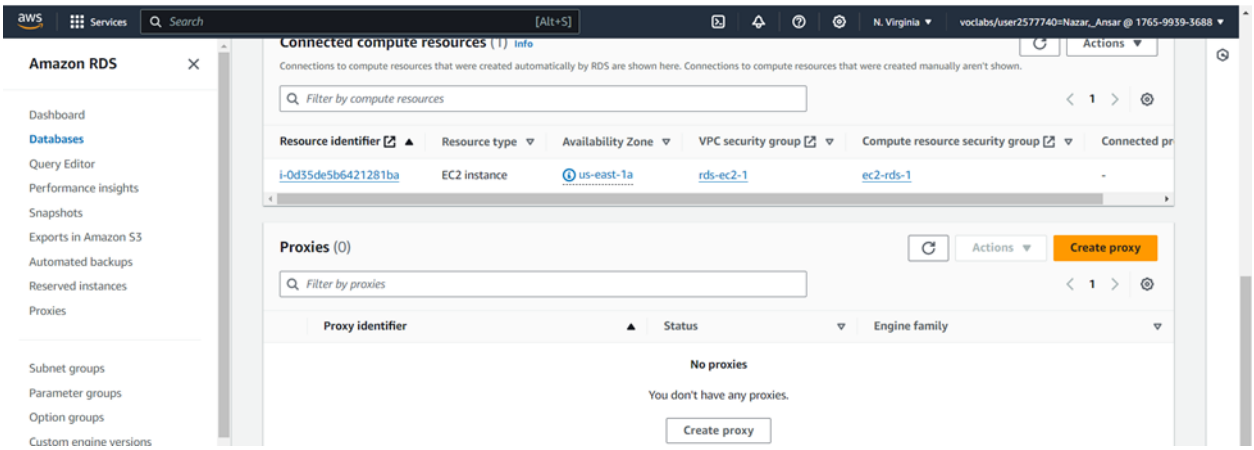

Figure 4 RDS database connected to the instance.

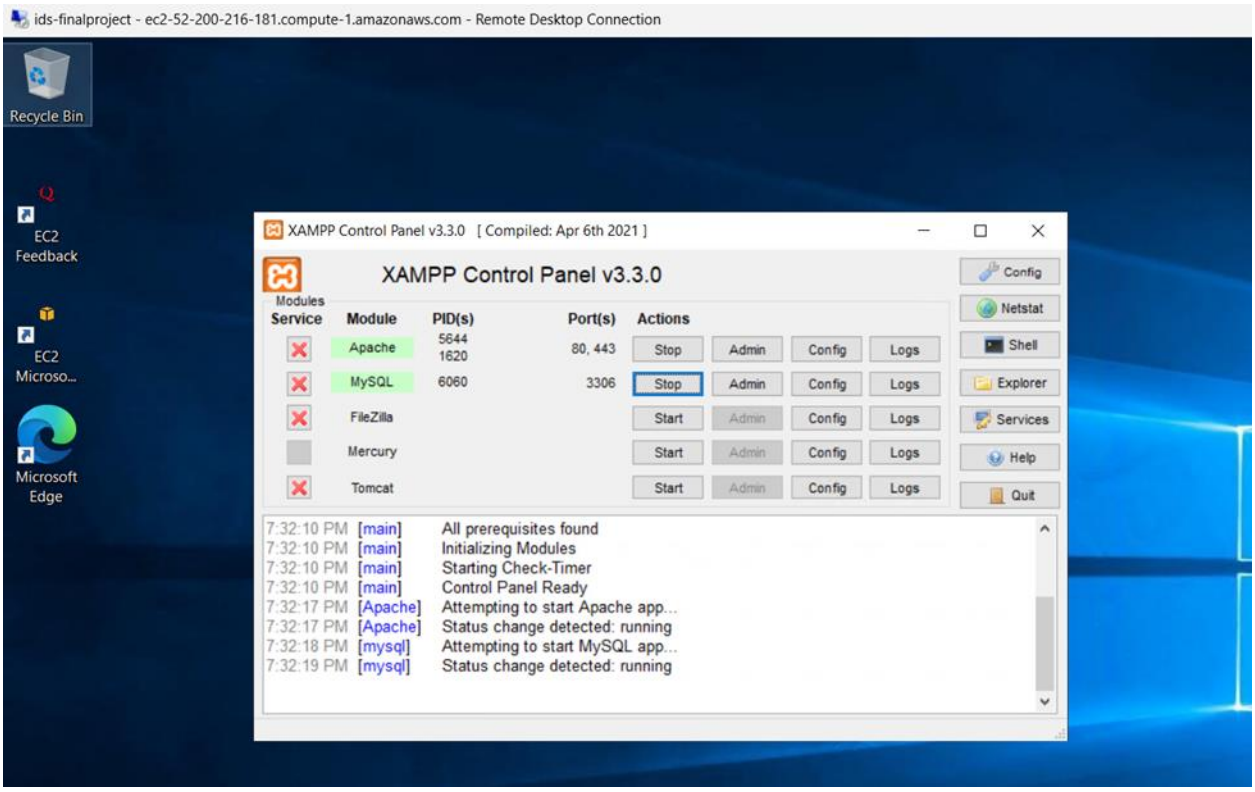

Figure 5 XAMPP installed and started the services.

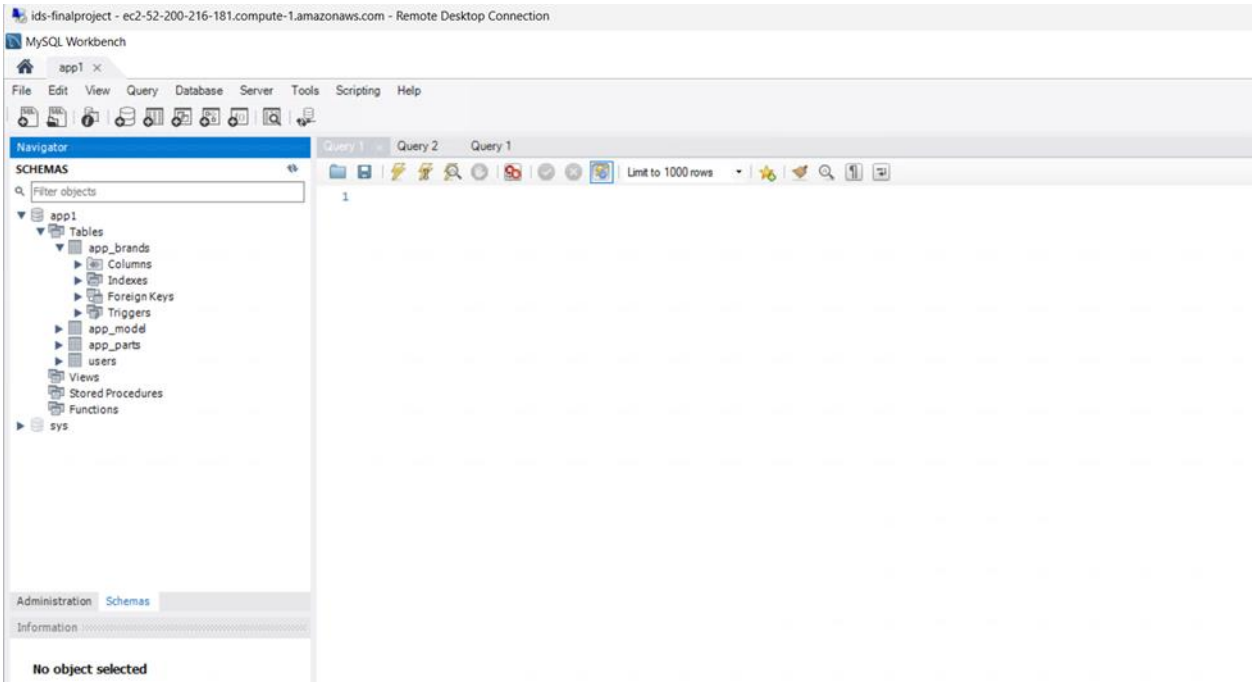

Figure 6 Workbench connected to the database and a schema named app1 created and imported the SQL.

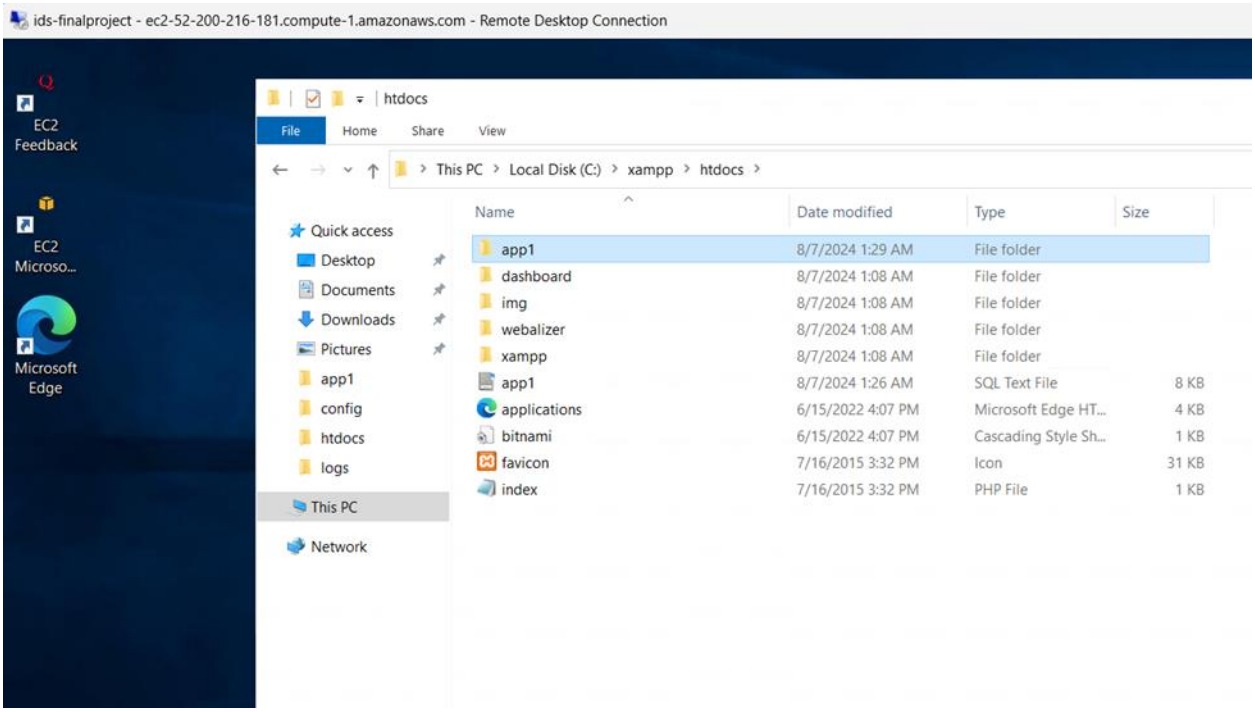

Figure 7 app1 data copied to the htdocs folder in the EC2 instance.

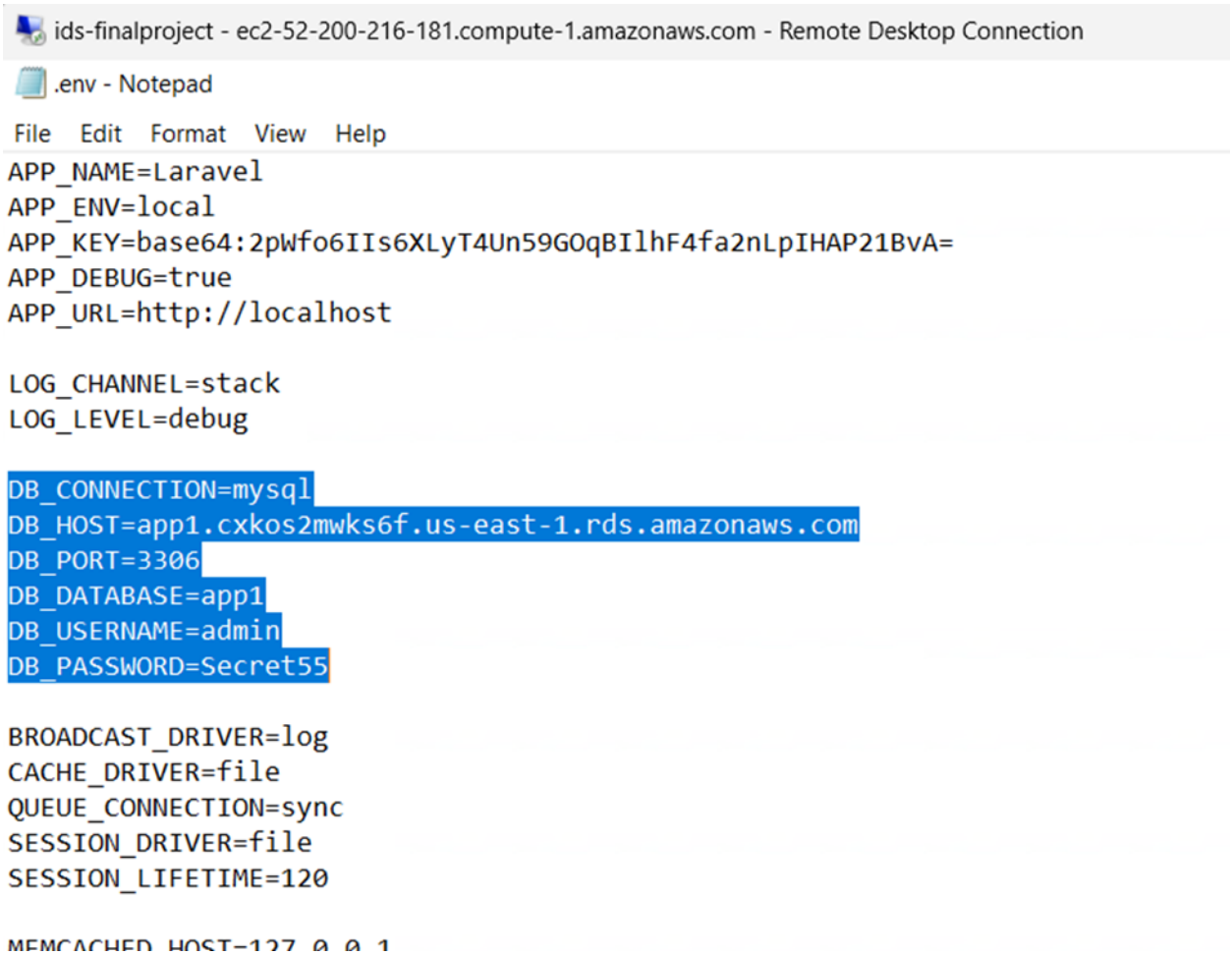

Figure 8 Configuration details of the .env file.

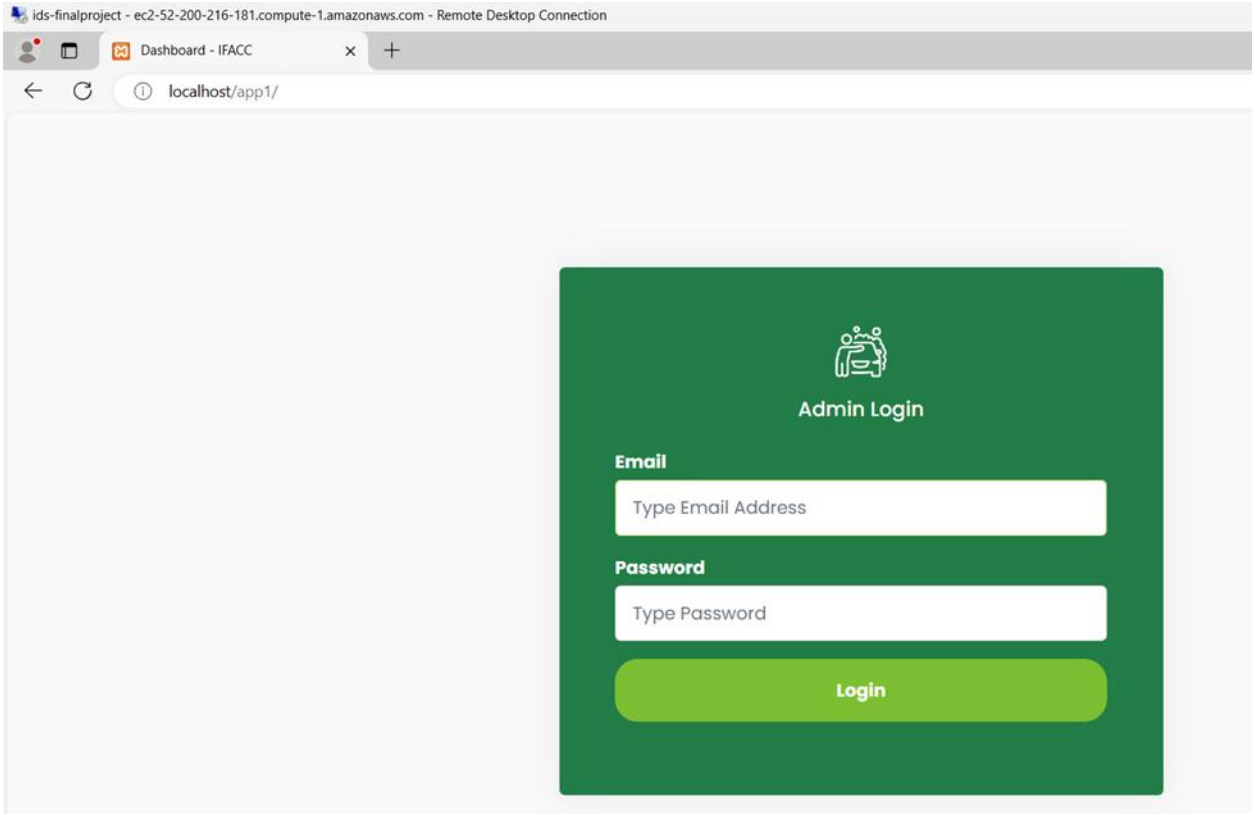

Figure 9 app1 login page accessed.

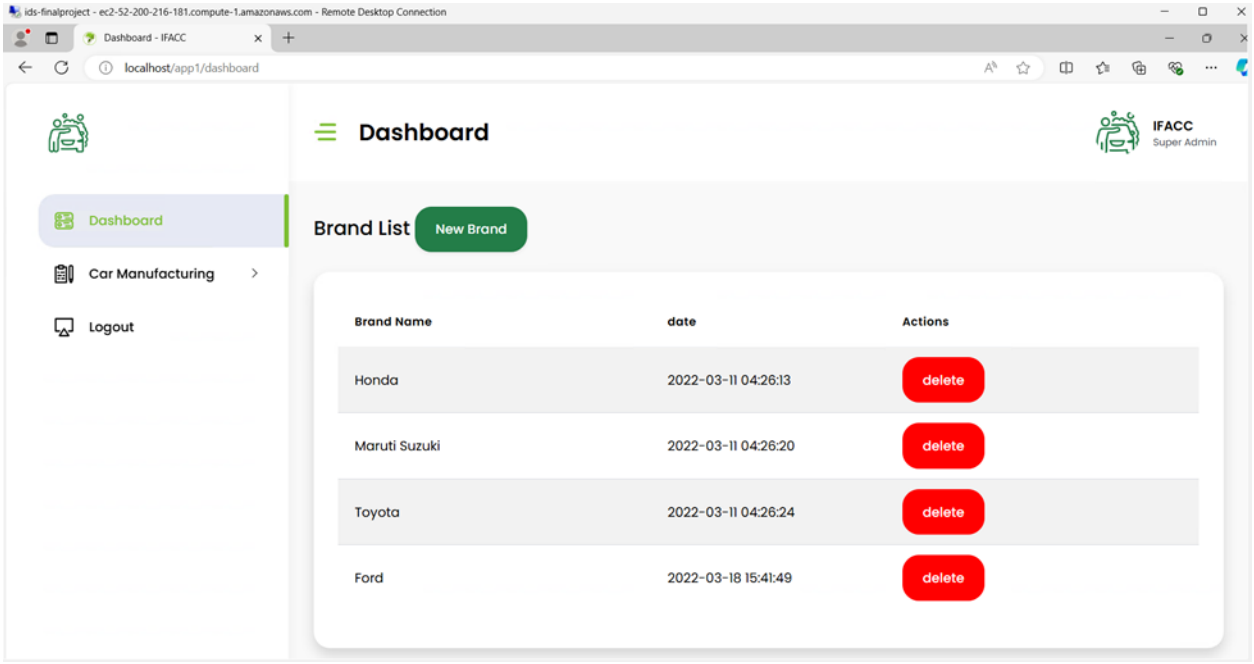

Figure 10 Successfully logged into the app1.

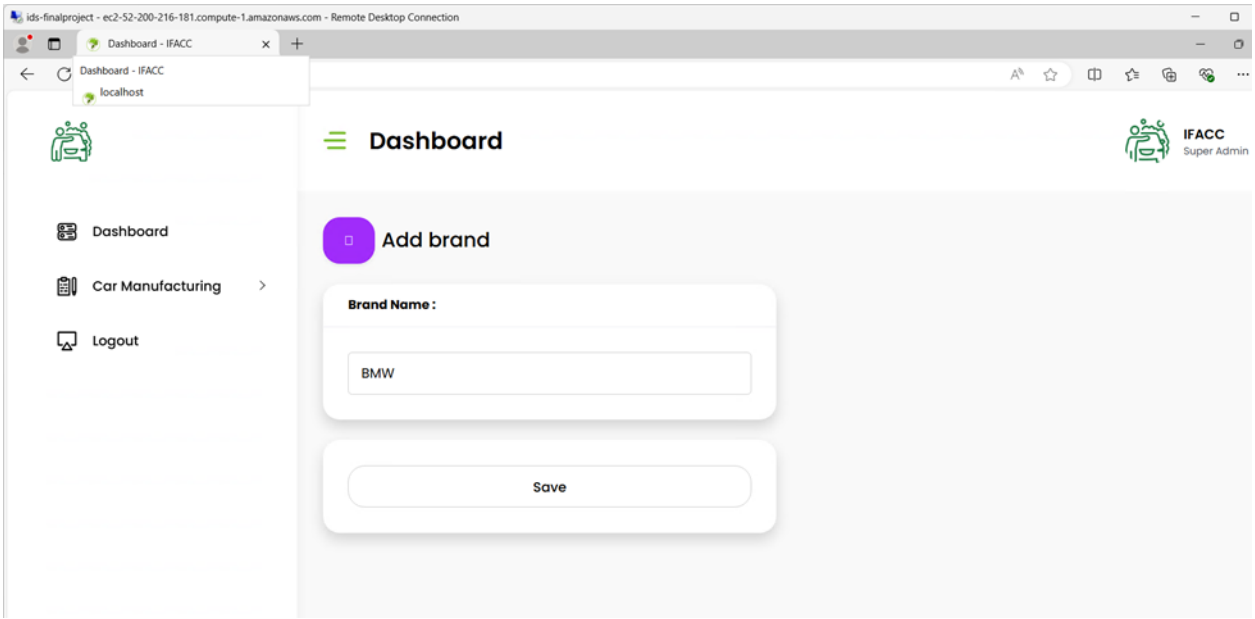

Figure 11 Adding a new car brand BMW.

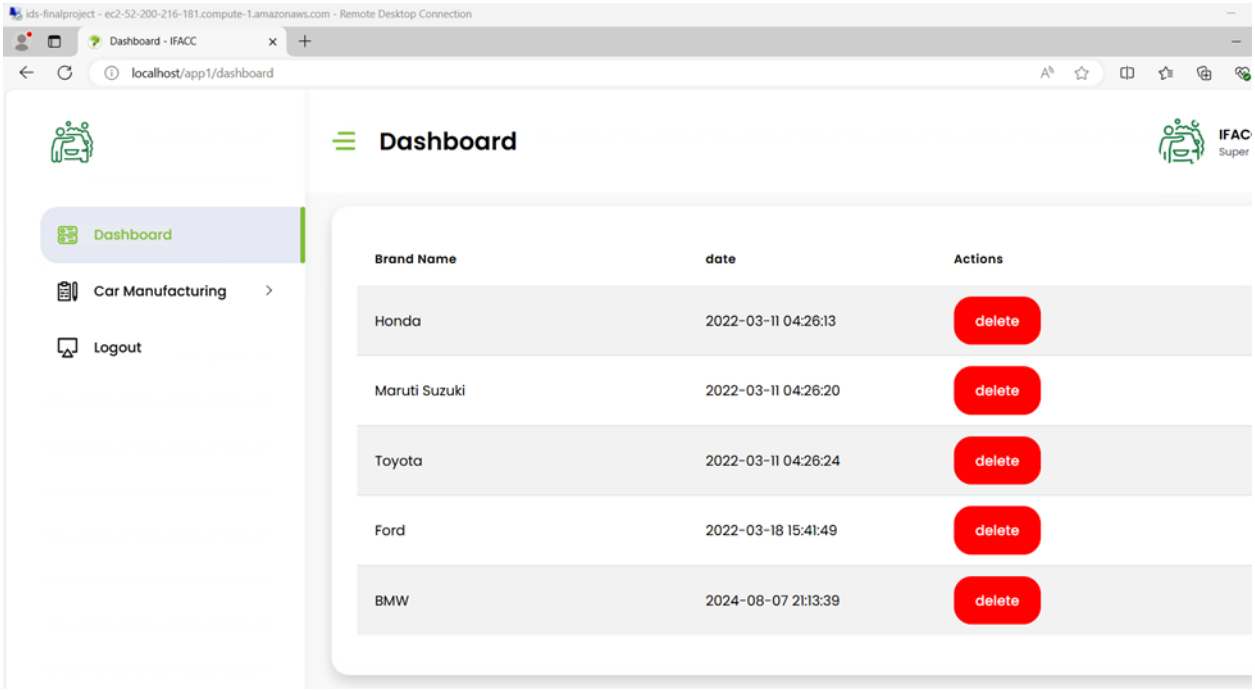

Figure 12 Added car brand is shown in the list.

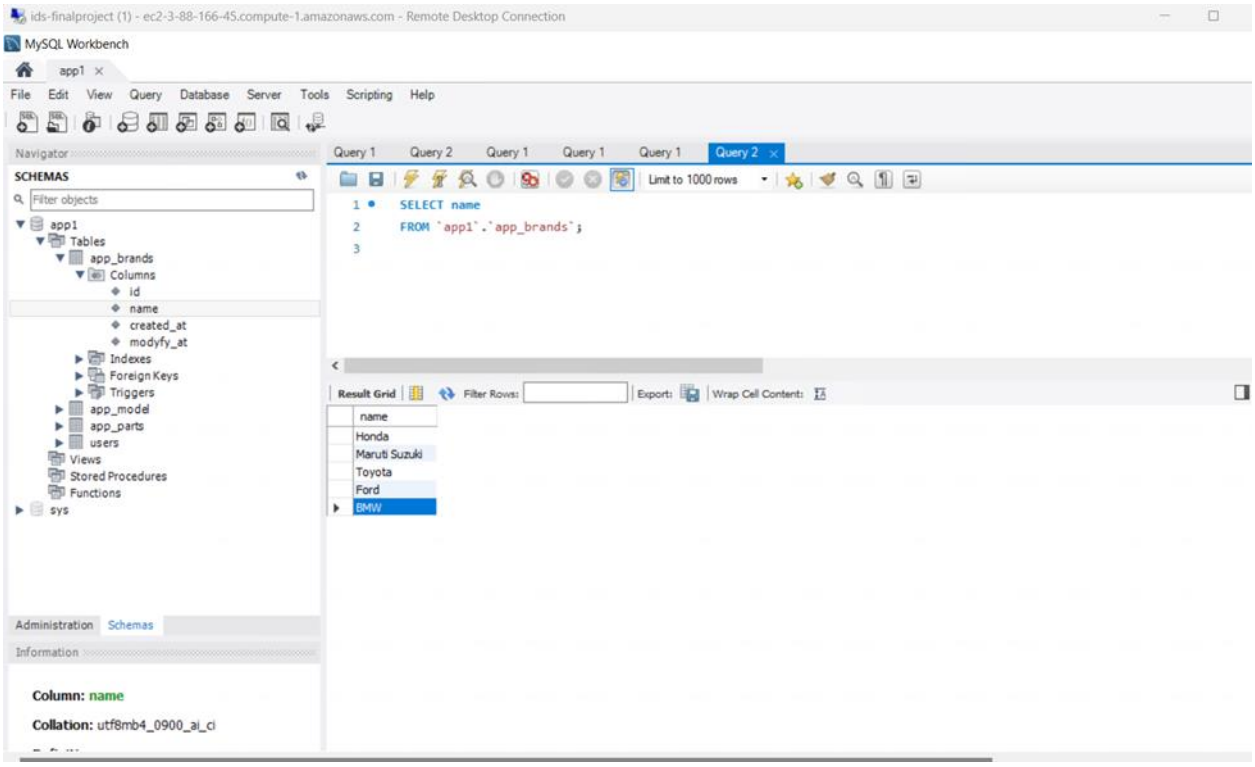

Figure 13 added brand is showing in the workbench.

APP 2

We started the lab by creating an EC2 instance for app2. We enabled HTTP, RDS, HTTP traffic for the EC2. Afterward, we created an RDS database in our AWS named app2.We configured the security groups to allow traffic from the ec2 and the mysql port and ensured the dp is publically accessible. Moreover, the ec2 instance is successfully connected to the RDS. Thereafter we accessed the ec2 instance using RDP and installed XAMPP and workbench to proceed with further configurations. XAMPP was installed, and we started the Apache and MySQL server successfully. Afterward, we created a connection with our RDS database in the workbench using the endpoint from the RDS instance we created. The connection with RDS was successfully established, and we created a new schema named app2. Afterward, we imported all the tables to the schema. The app2 file is copied to the htdocs folder in XAMPP. We then navigated to the .env file and configured the host, db_name, port, username, and password correspondingly. All the connections are configured, and we loaded the applications locally and accessed the web application successfully. The modifications made in app1 are reflected in app2. We verified the integration by adding a new brand BMW in app1, and it was listed in the brands in app2

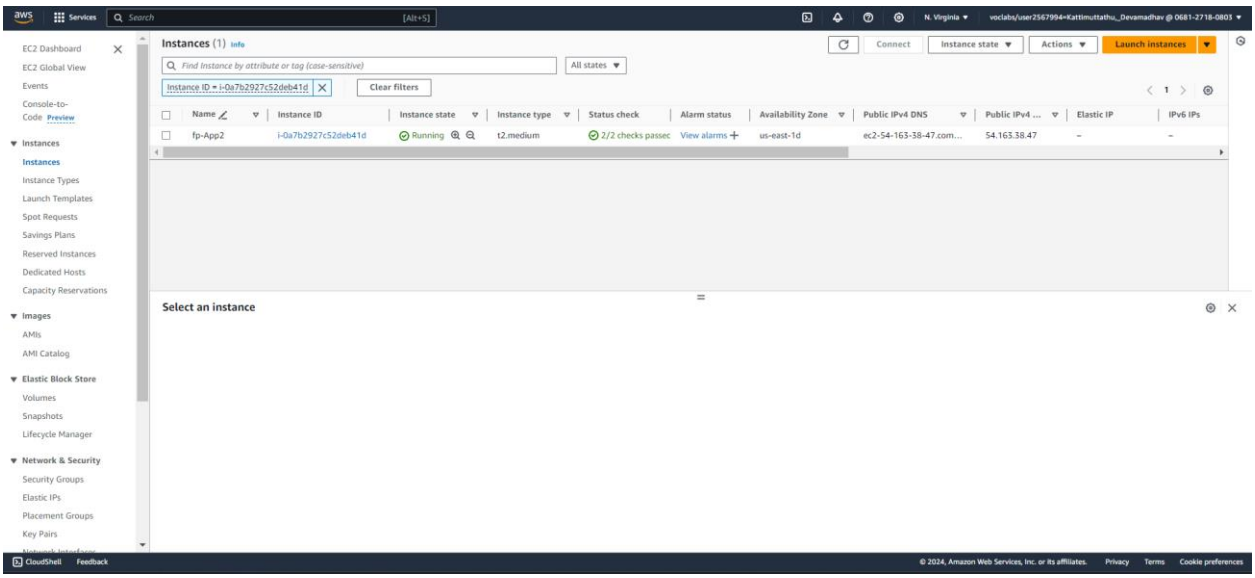

Figure 1 - Created an instance for app2

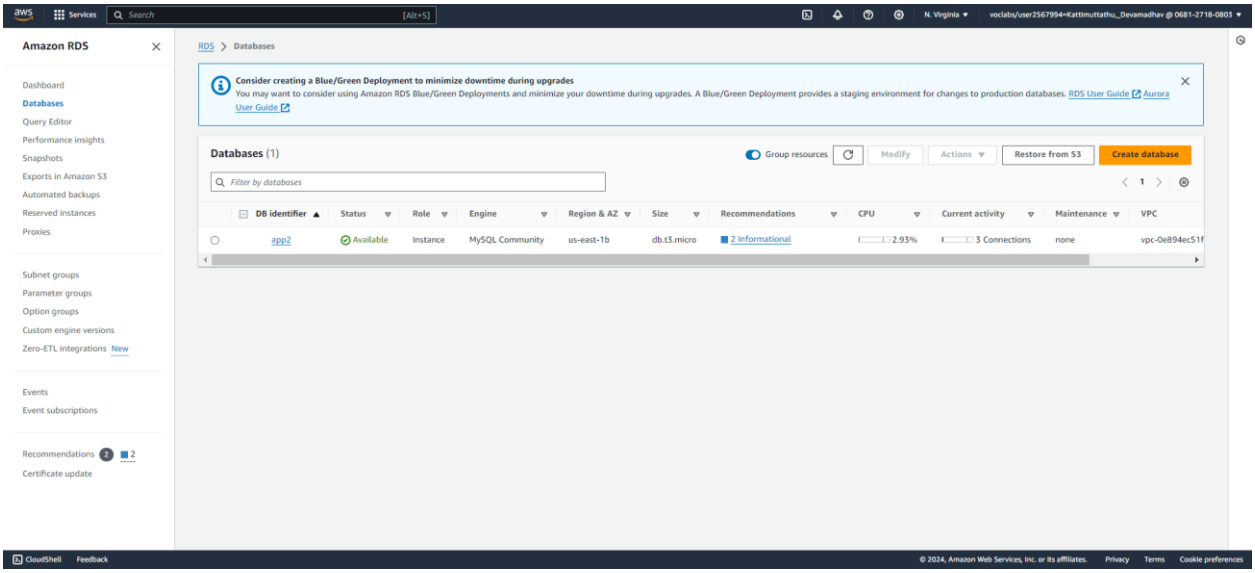

Figure 2- Created an RDS in app2

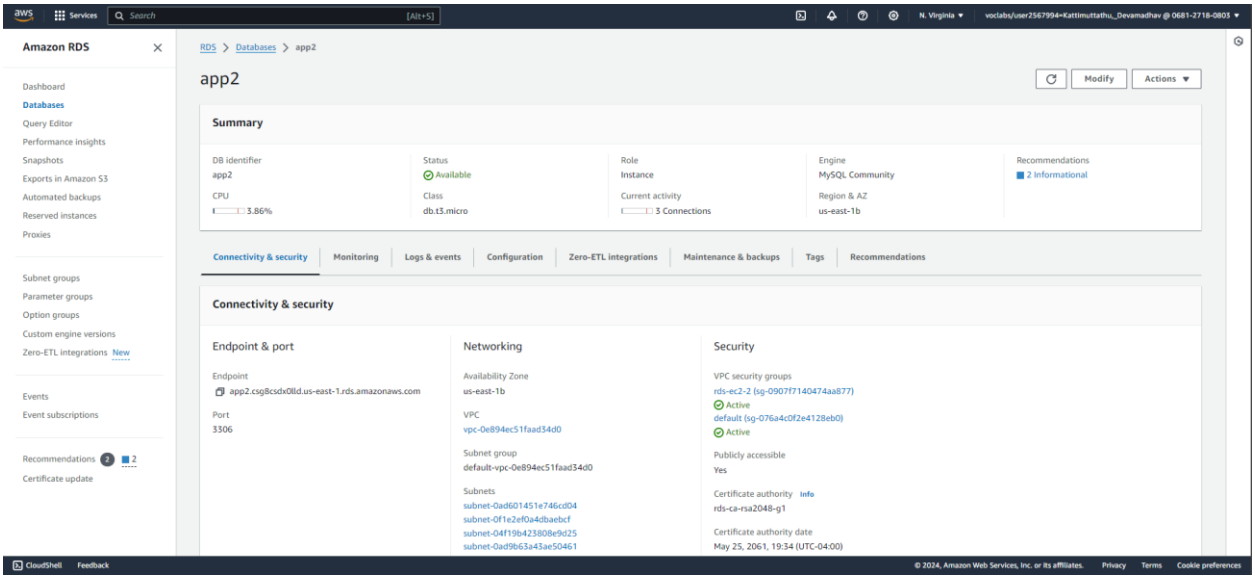

Figure 3 -Displaying the details of RDS

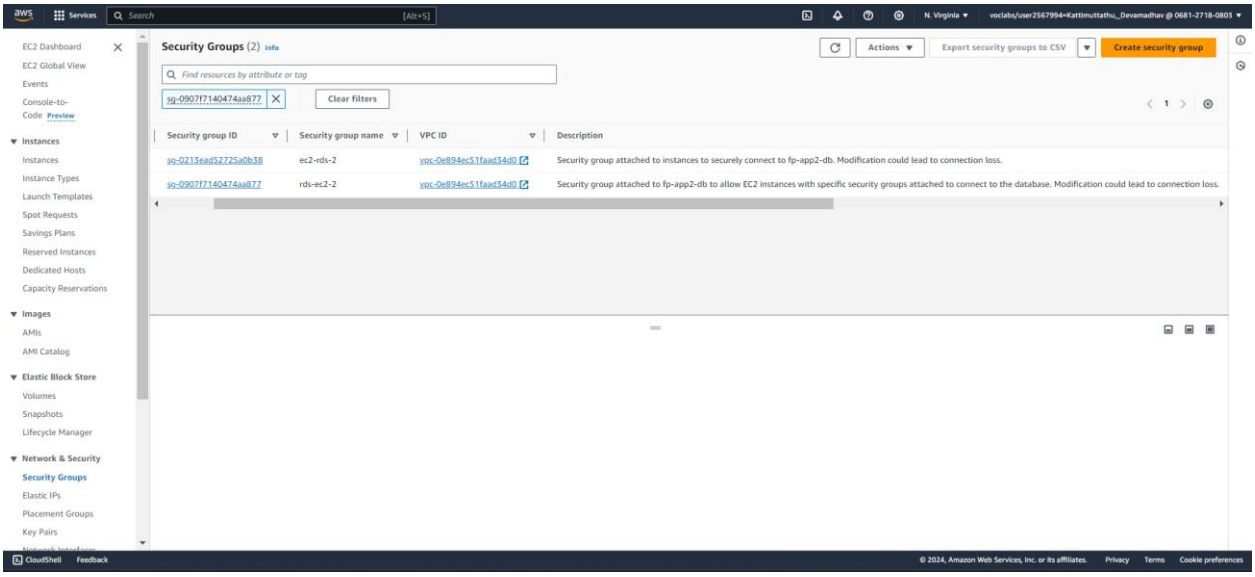

Figure 4 - Security groups of RDS

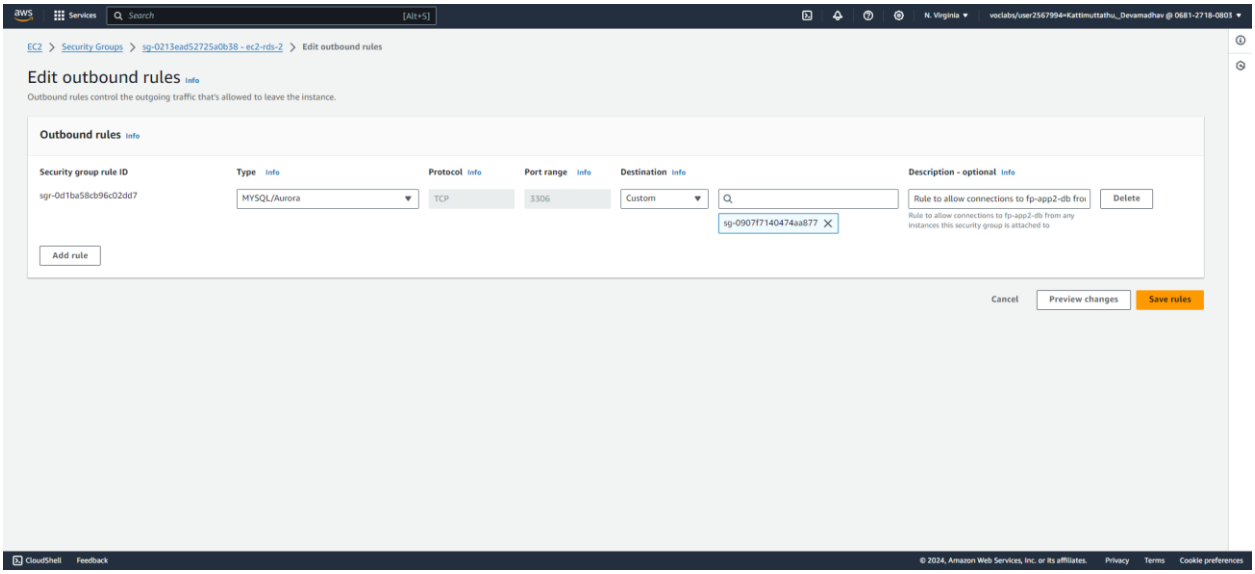

Figure 5 - Outbound rule set for the ec2 instance

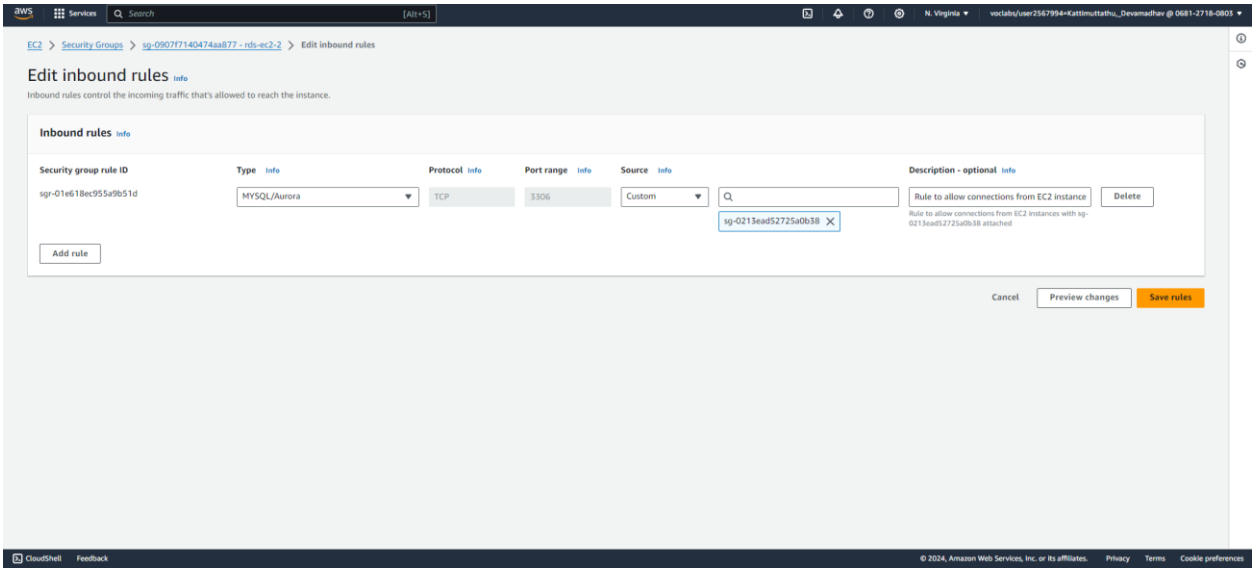

Figure 6 - Inbound set rules for RDS

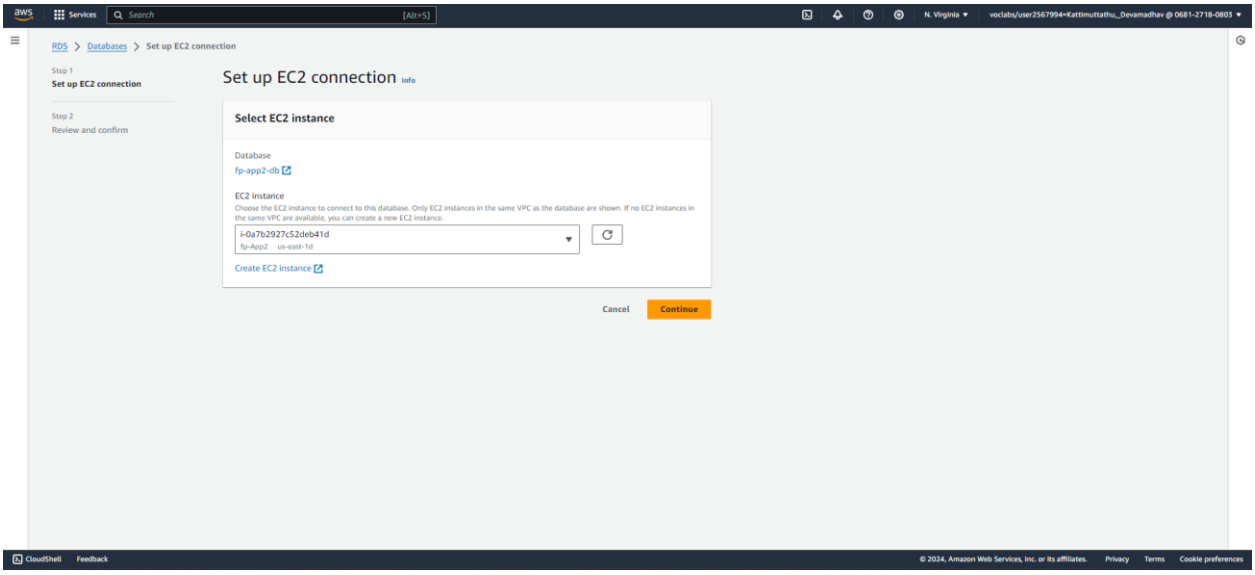

Figure 7 - Connecting Ec2 with RDS

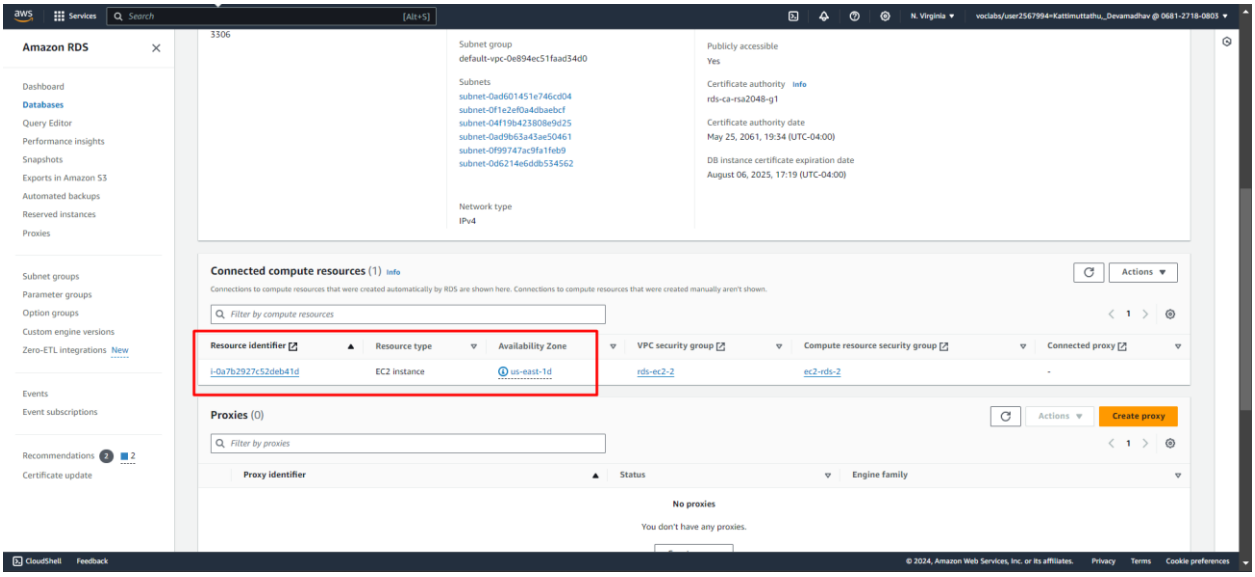

Figure 8 - Successfully connected the EC2 with RDS

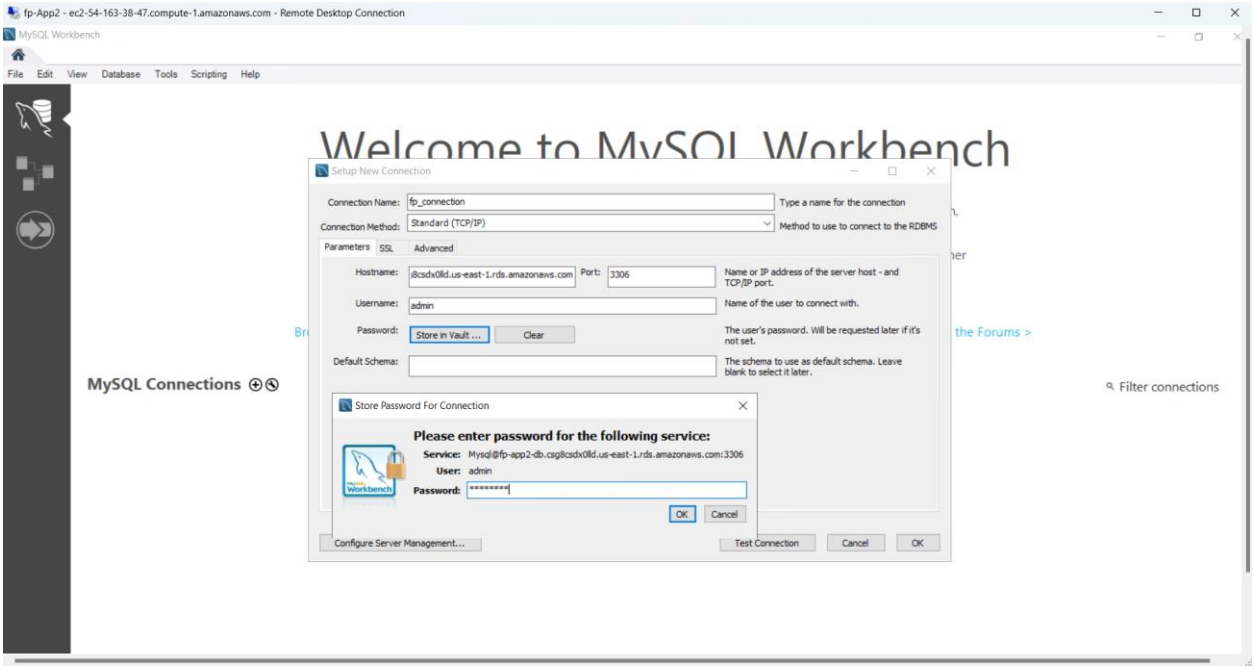

Figure 9 - Creating a connection in workbench

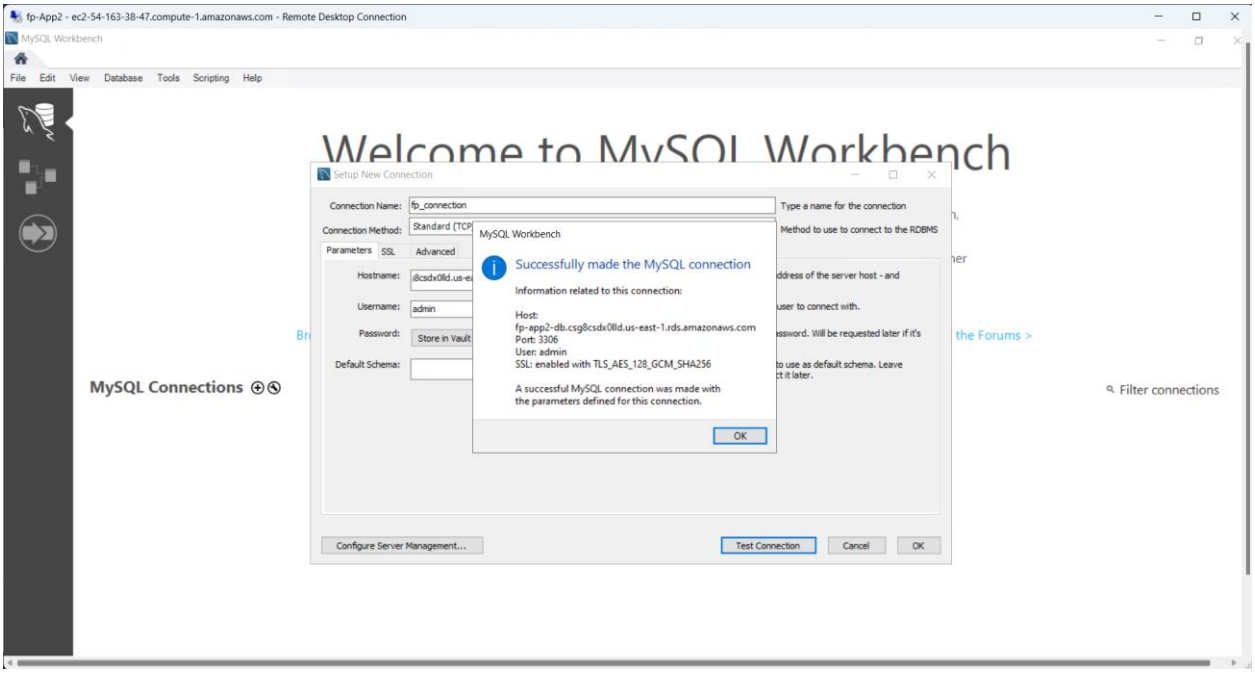

Figure 10 - Successfully connected to the RDS

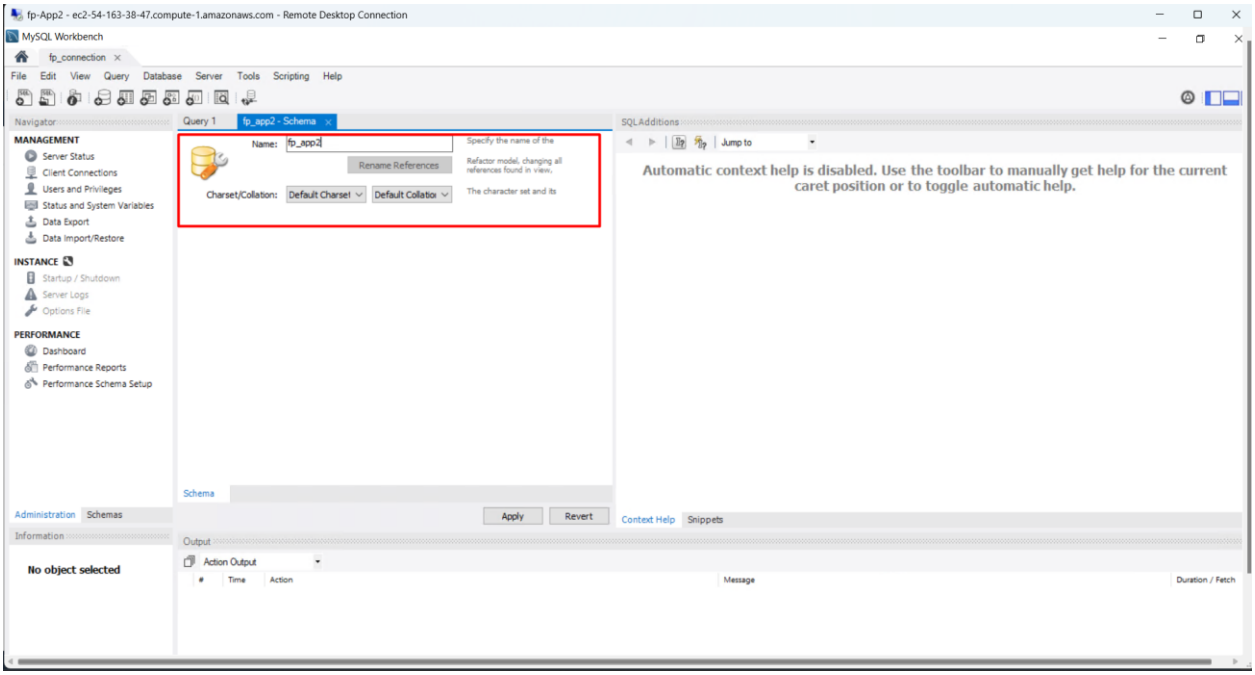

Figure 11 - Creating a schema inside the workbench

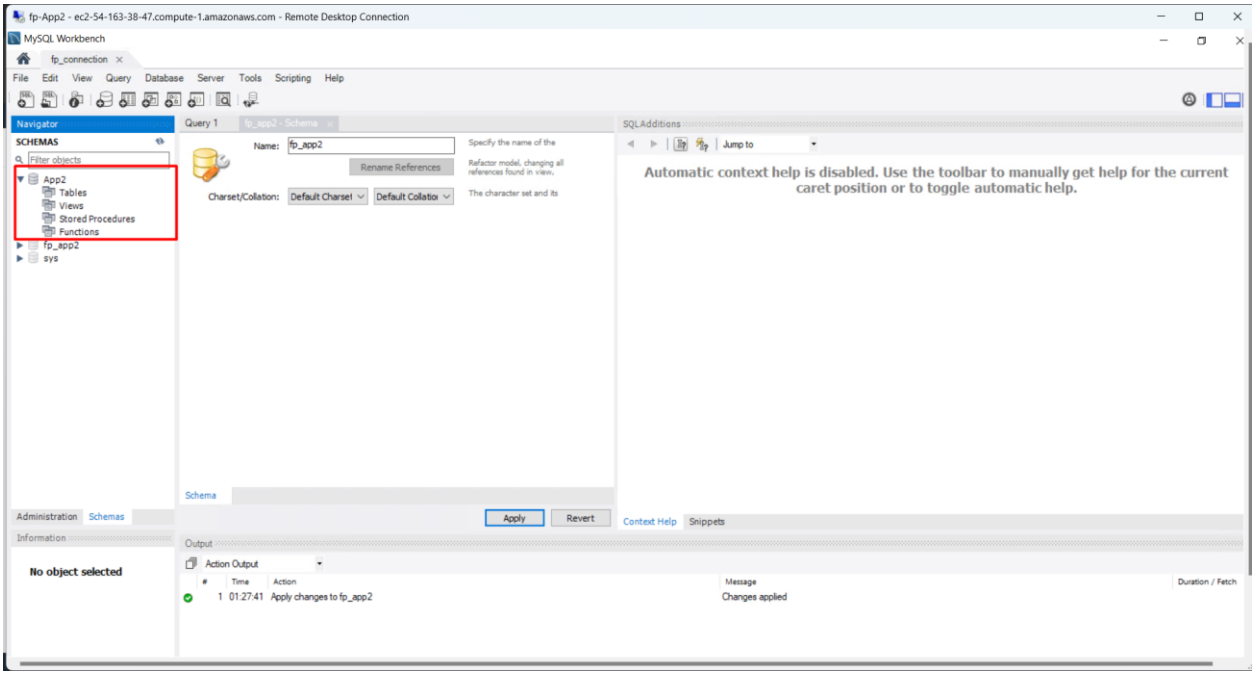

Figure 12 -Successfully created the schema

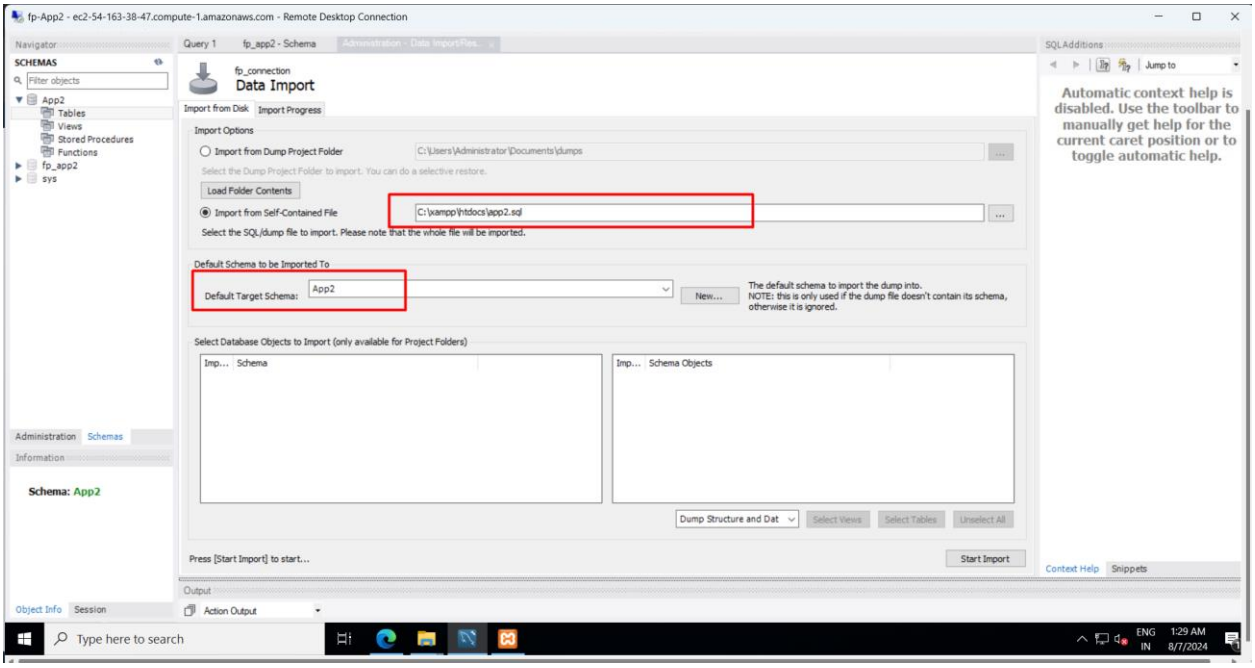

Figure 13 _ Importing the tables

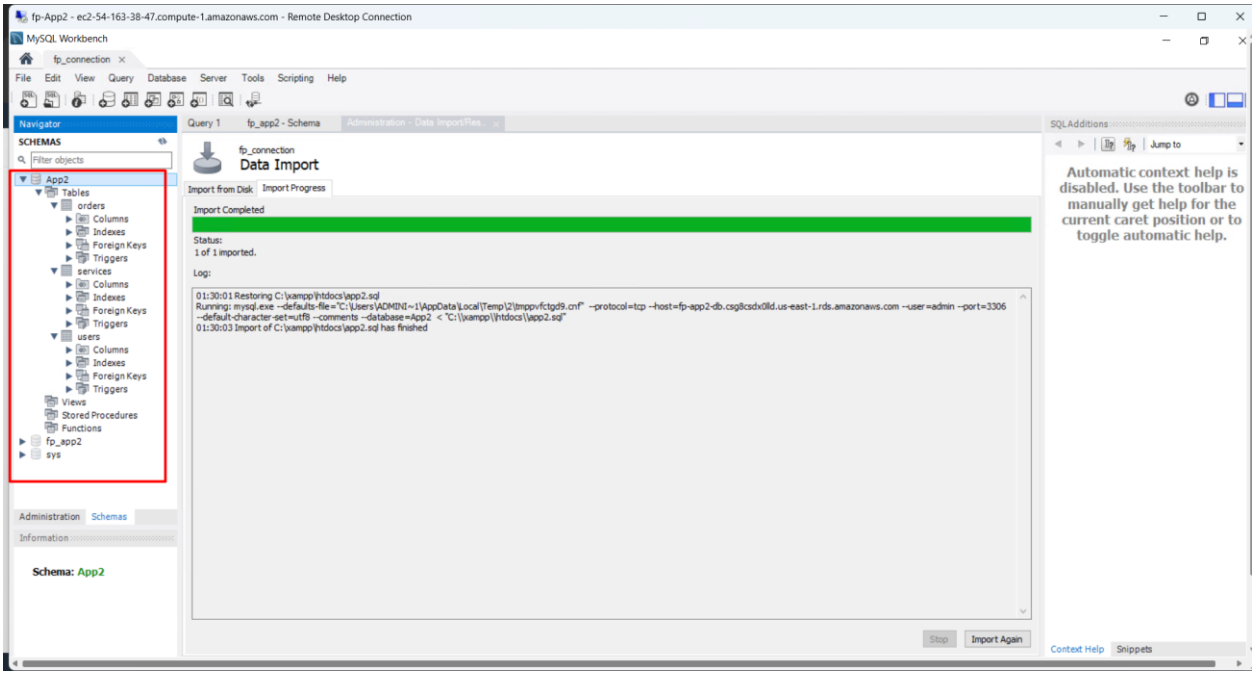

Figure 14 - Successfully imported the data

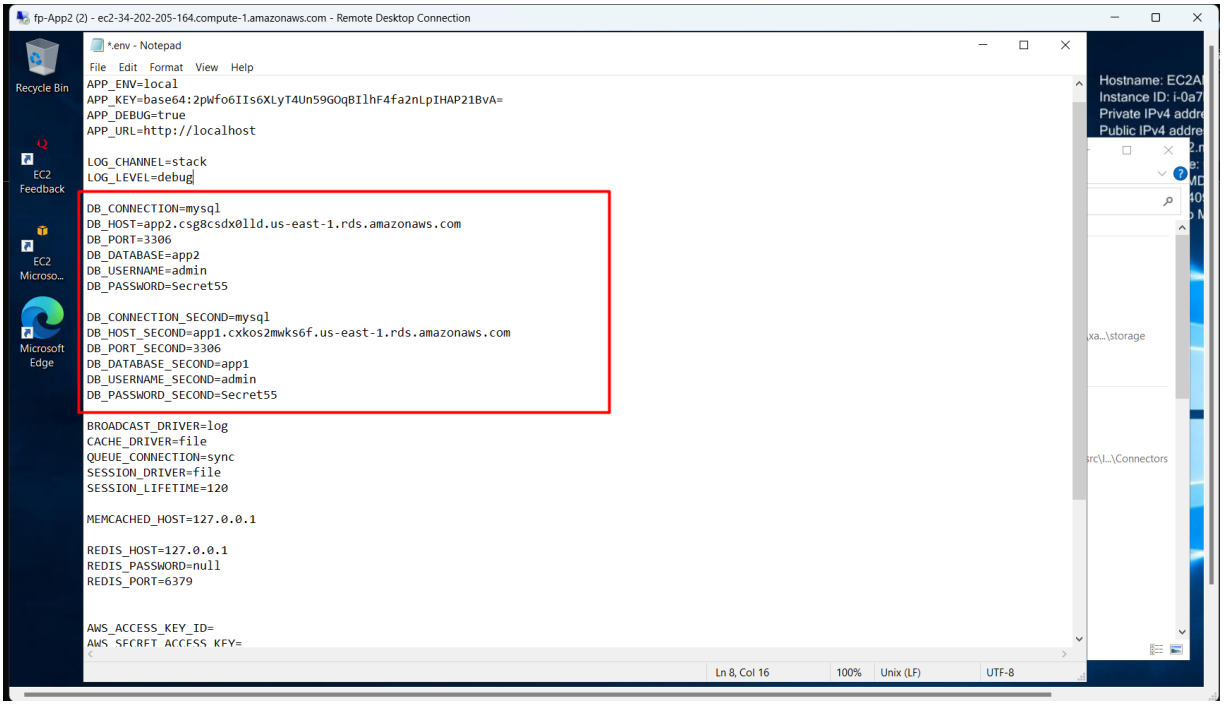

Figure 15 - Modified the .env file

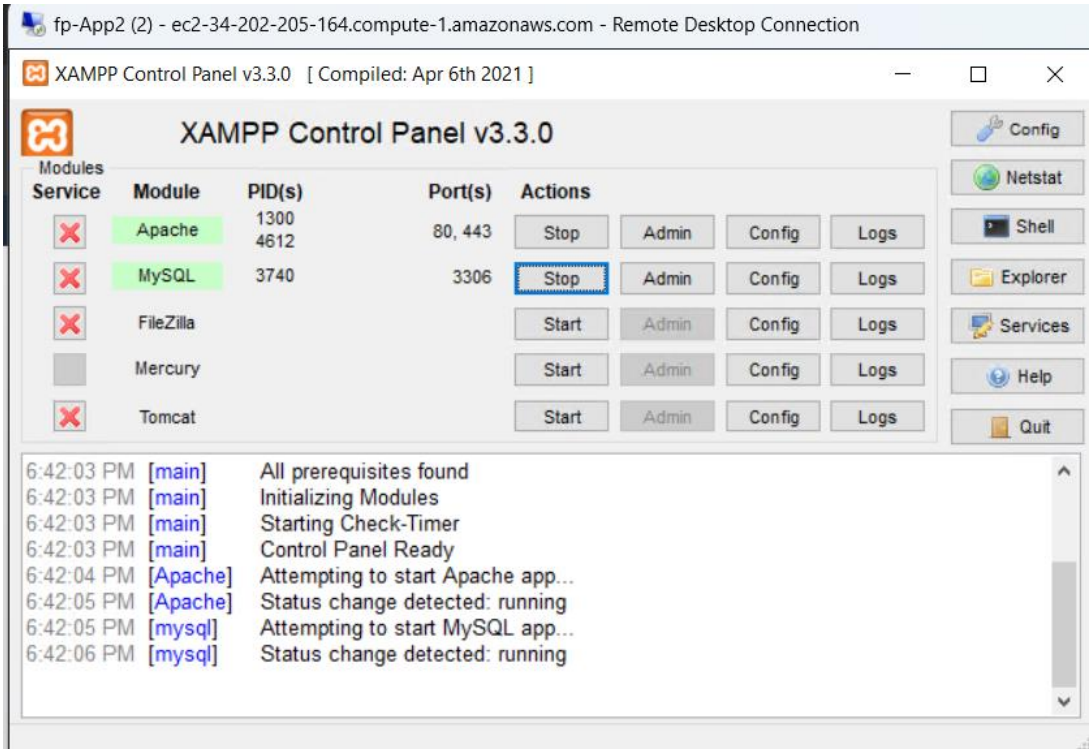

Figure 16 -Displaying the status of xampp

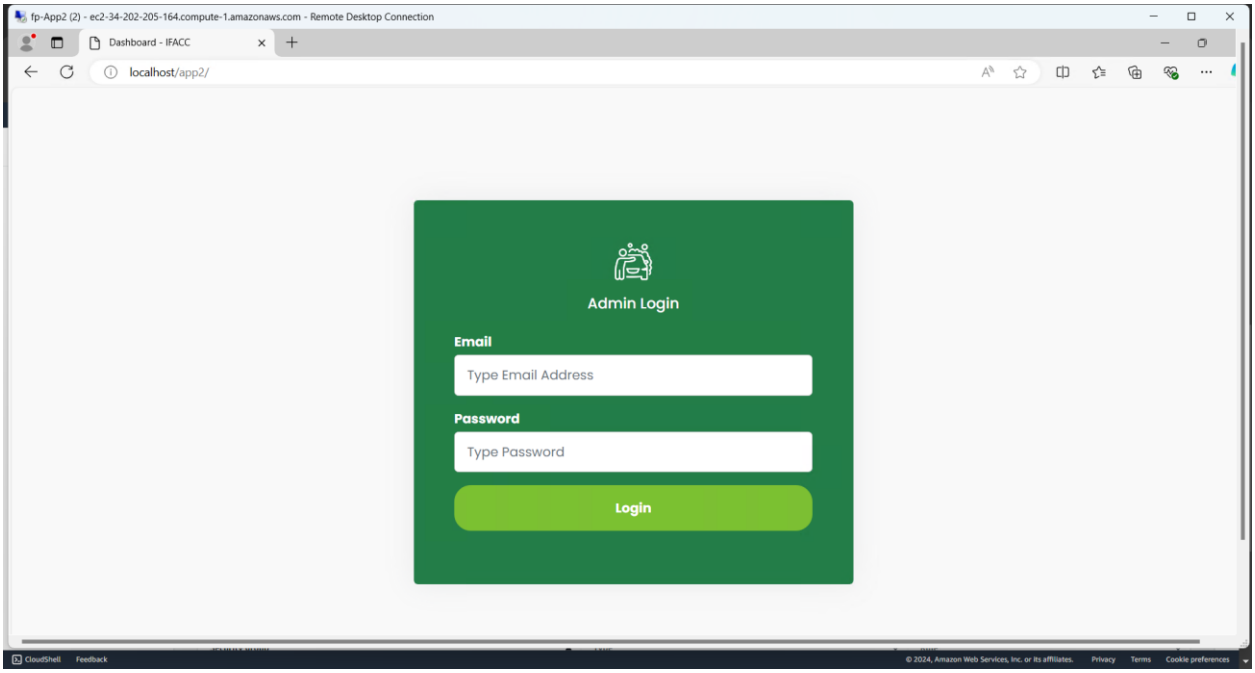

Figure 17 - Successfully accessed the homepage

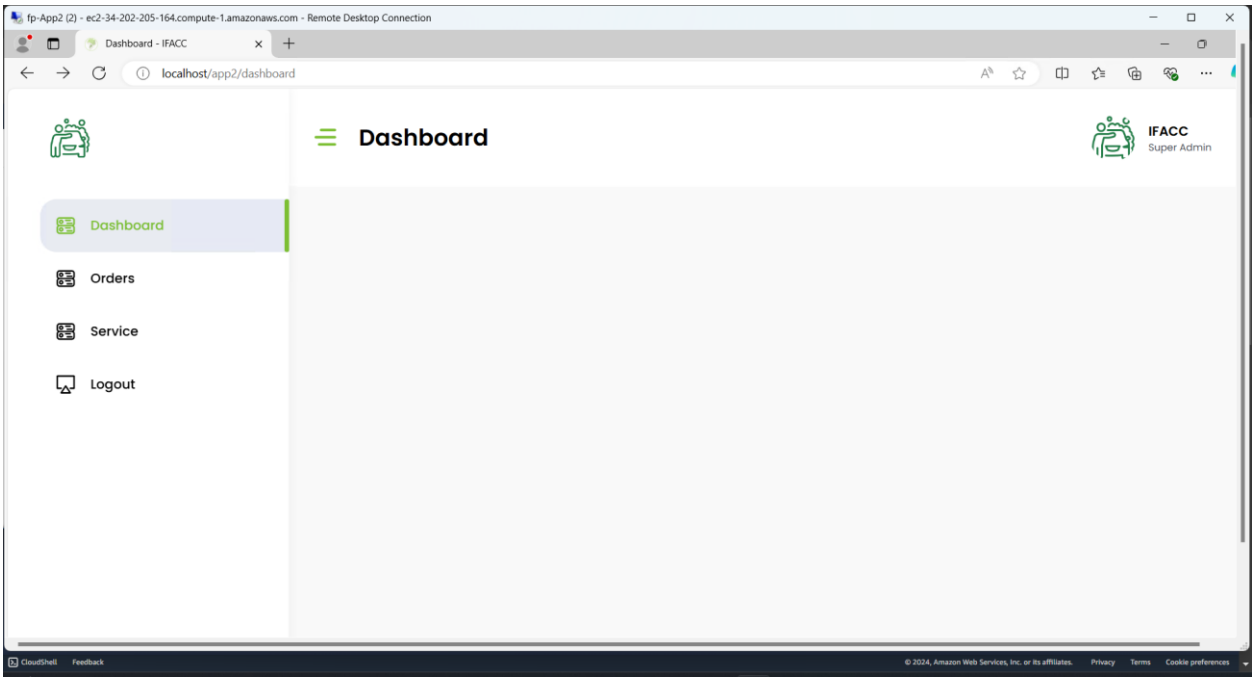

Figure 18 - Accessed the dashboard of our app2

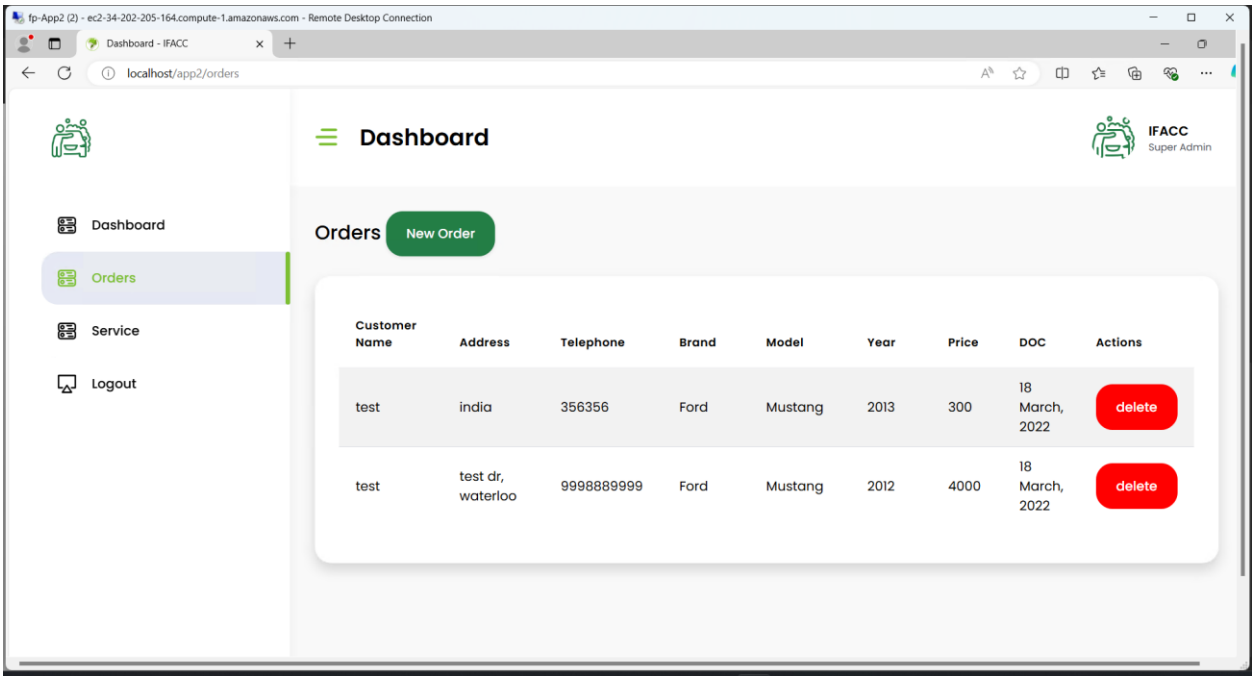

Figure 19 - Displaying the order tab

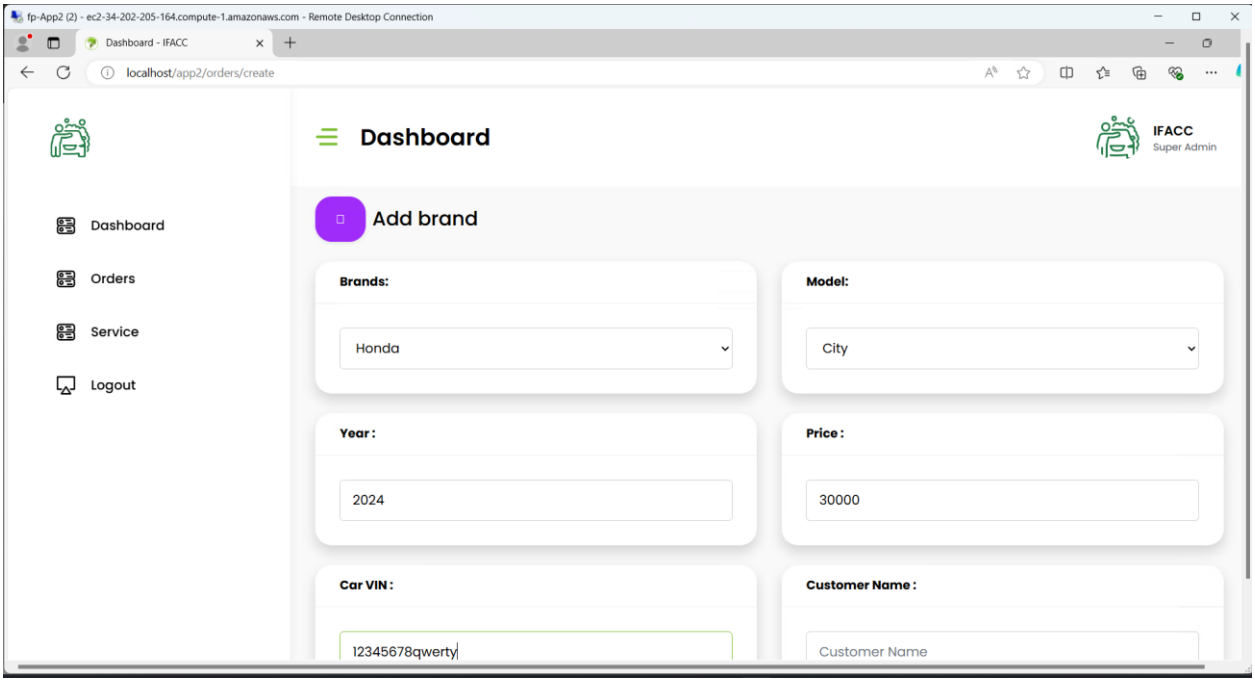

Figure 20 - Adding a new order

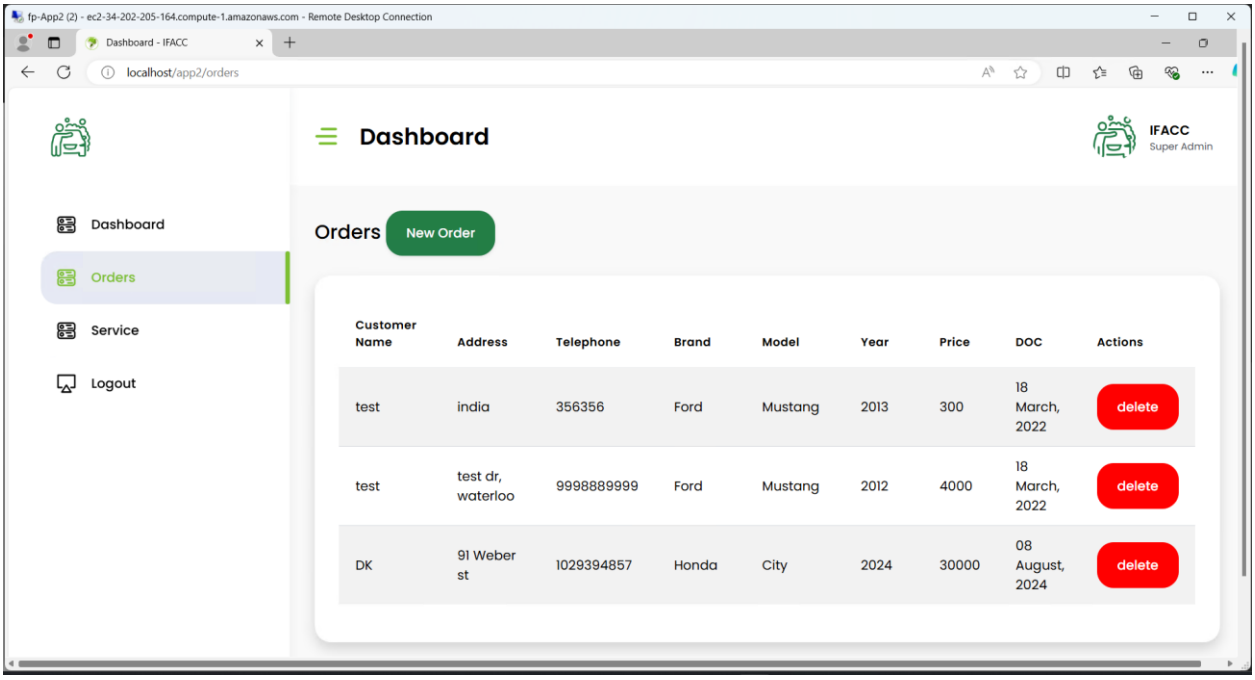

Figure 21 - Successfully added a new order

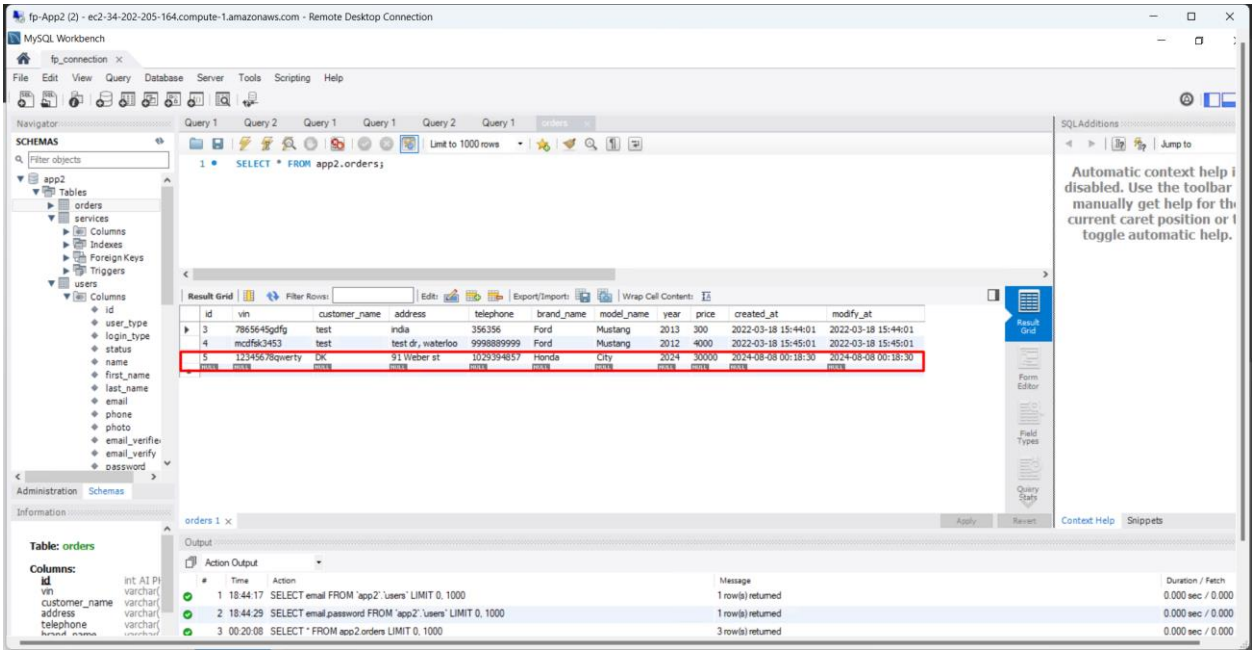

Figure 22 - Displaying the new order updated on our db

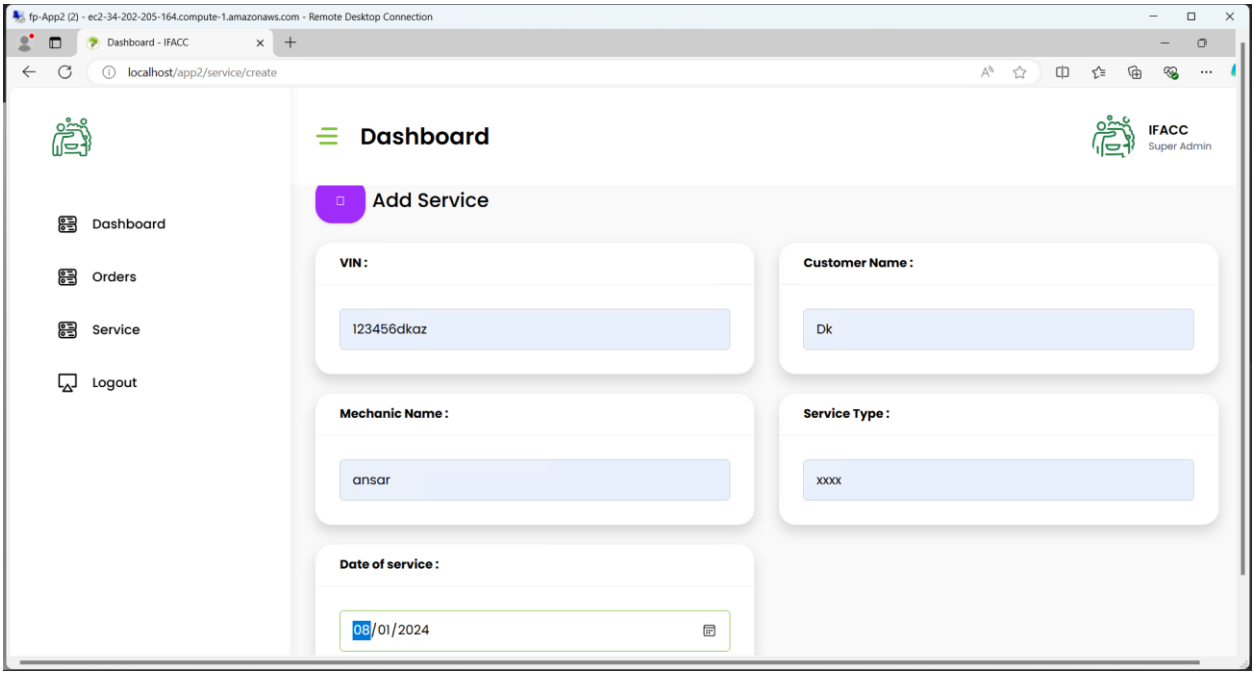

Figure 23 - Adding a new service in app2

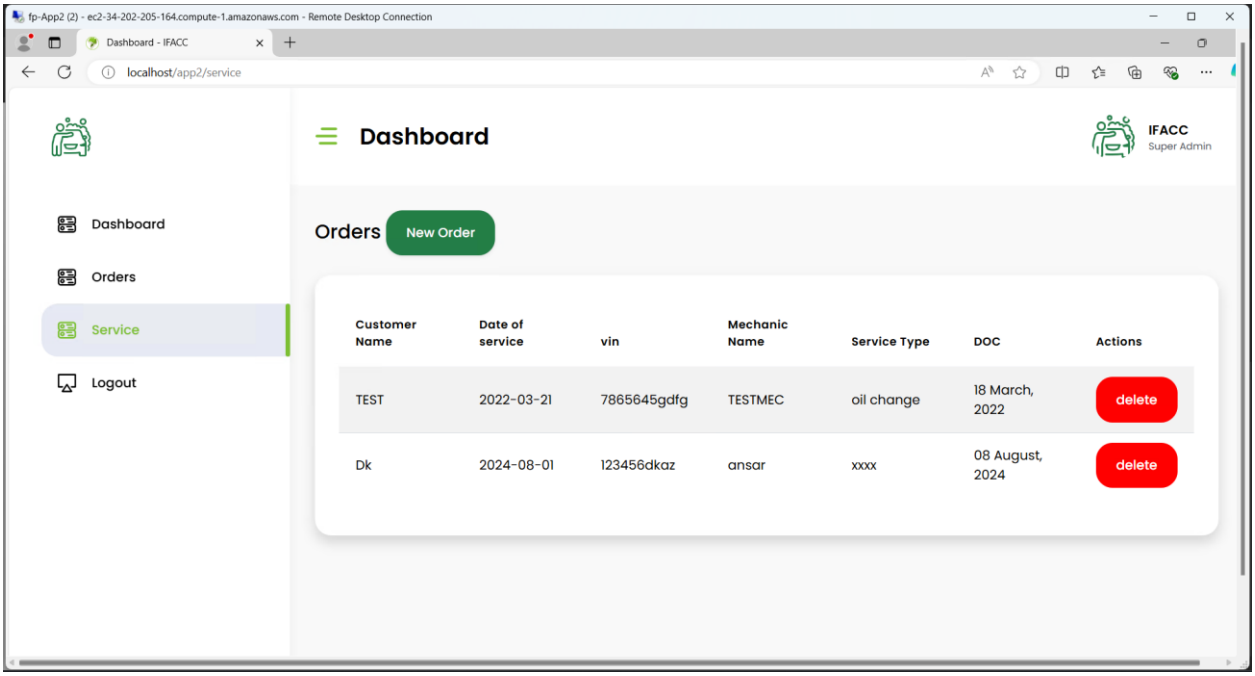

Figure 24 - Successfully added a new service log

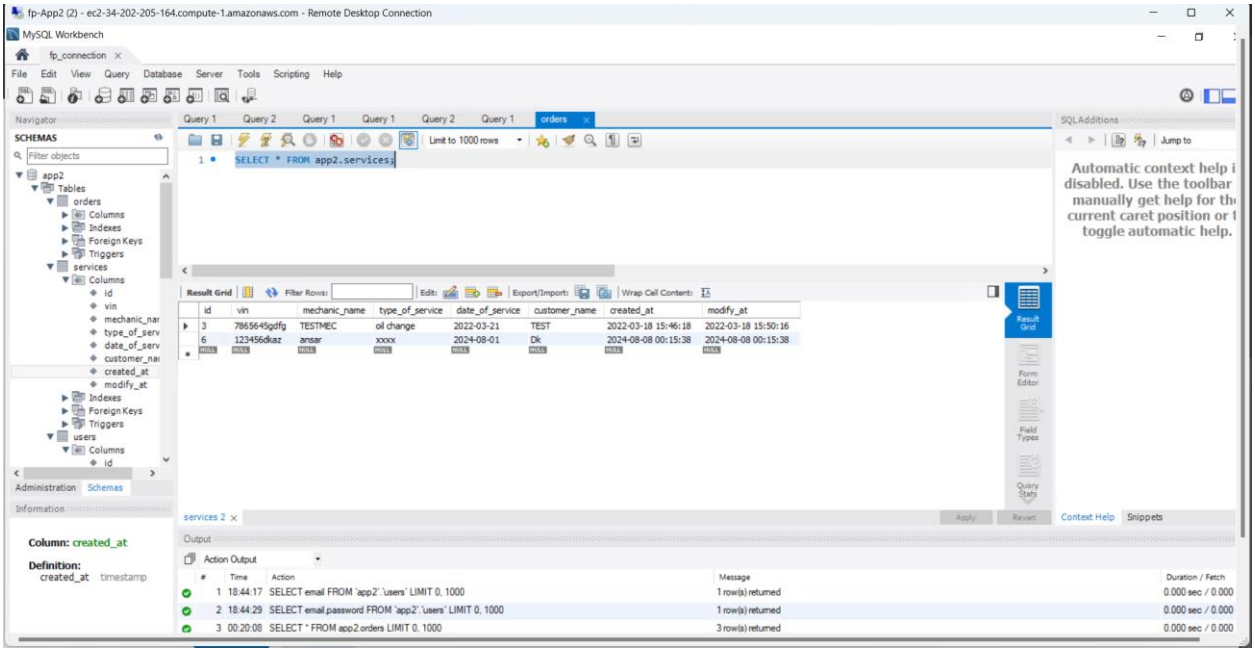

Figure 25 - The service is successfully added on the db

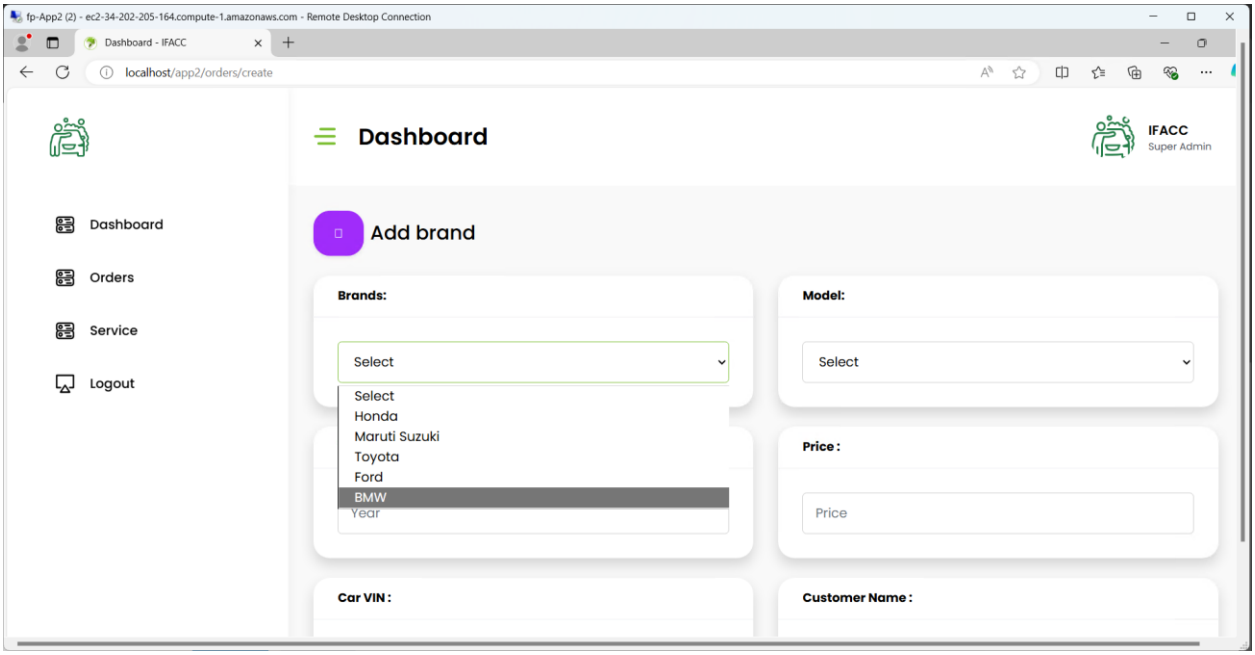

Figure 26 - A new brand BMW added on app1 is reflected on the app2

APP 3

Created the EC2 instance for app3. Added the outbound and inbound rules for EC2. Downloaded and installed XAMPP. Then we started the Apache and Mysql in XAMPP. Edited the env file added the db host details of app 2. Then launched the app3 in localhost. Successfully accessed the web application. I entered the vin ID and got the details of the corresponding ID.

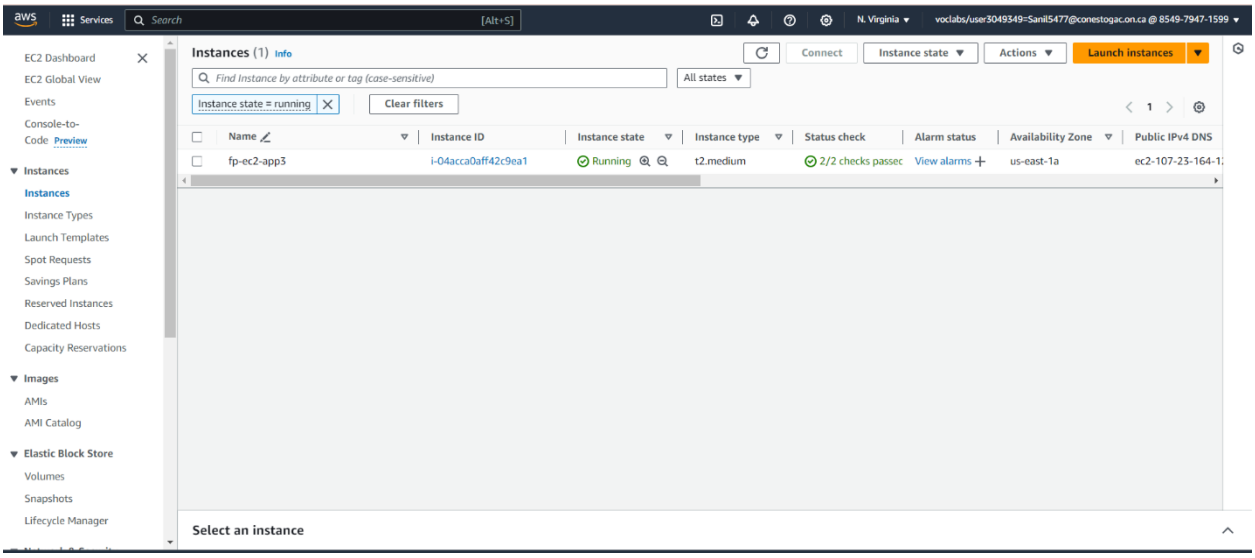

Figure 27 - Created an instance for app3

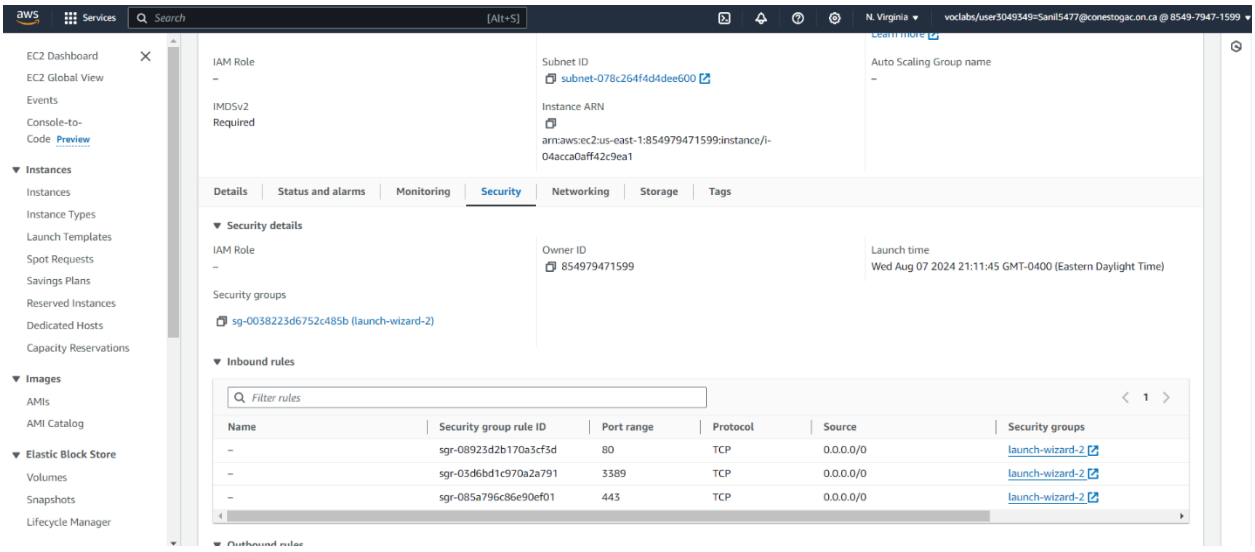

Figure 6 - Inbound rule set for the ec2 instance

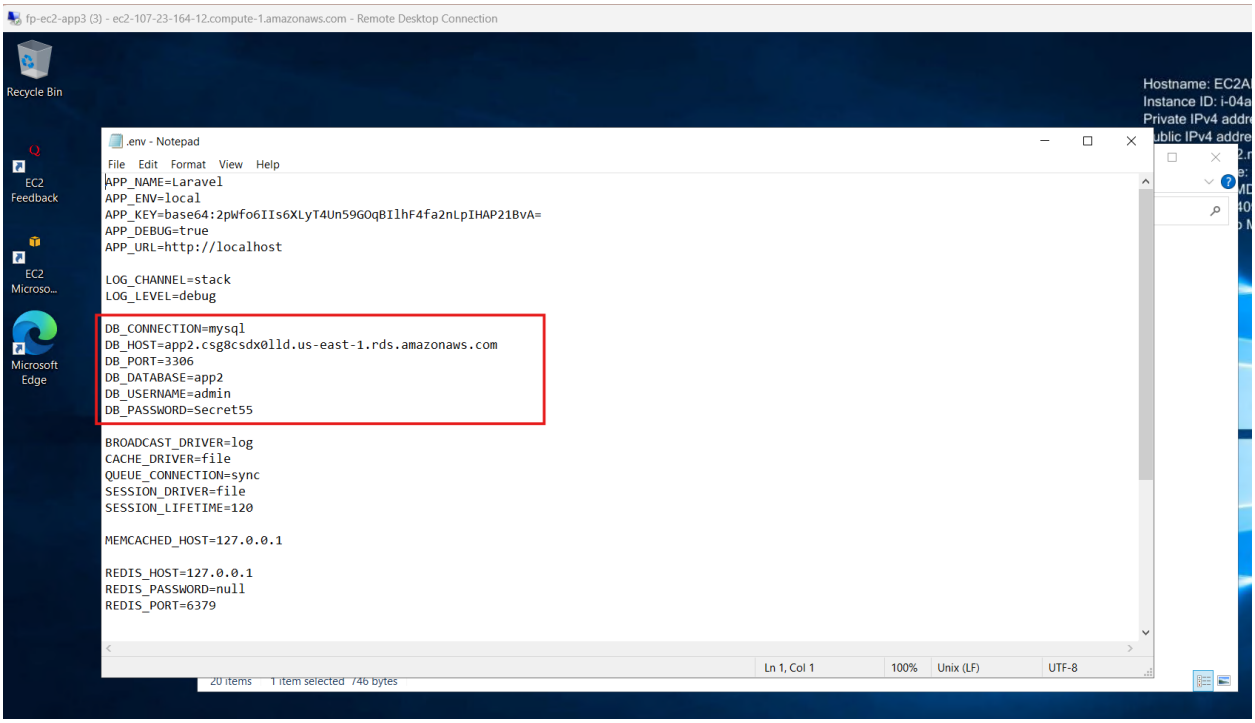

Figure 12 - Modified the .env file

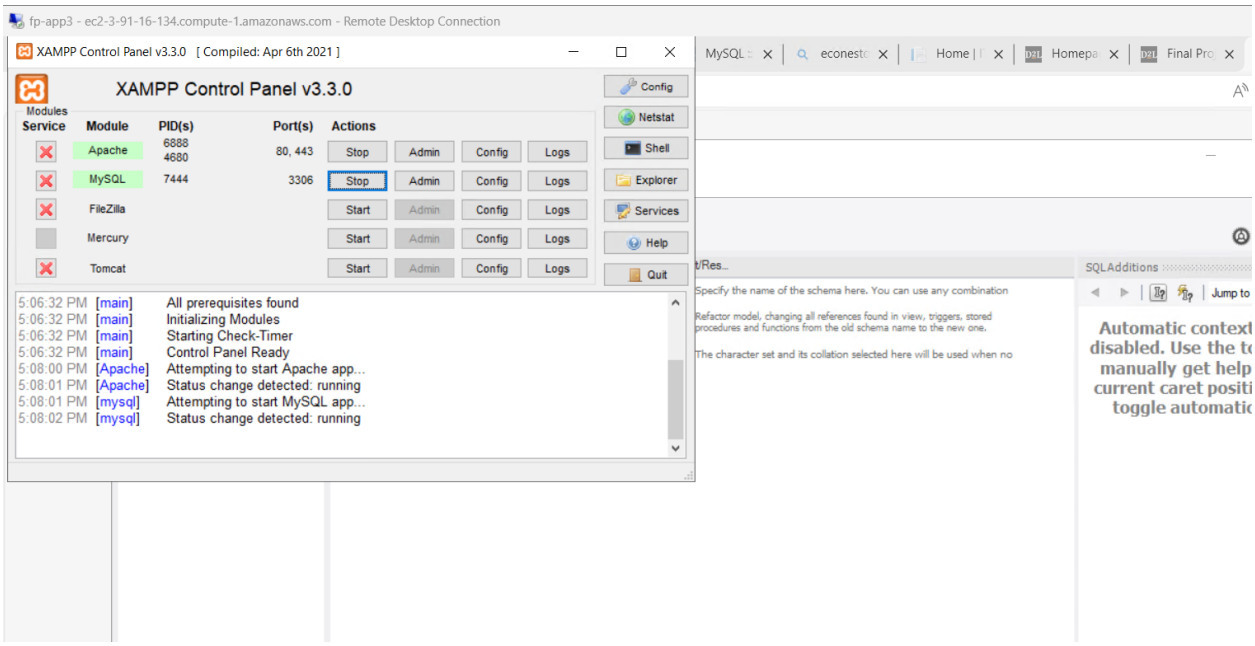

Figure 13 -Displaying the status of xampp

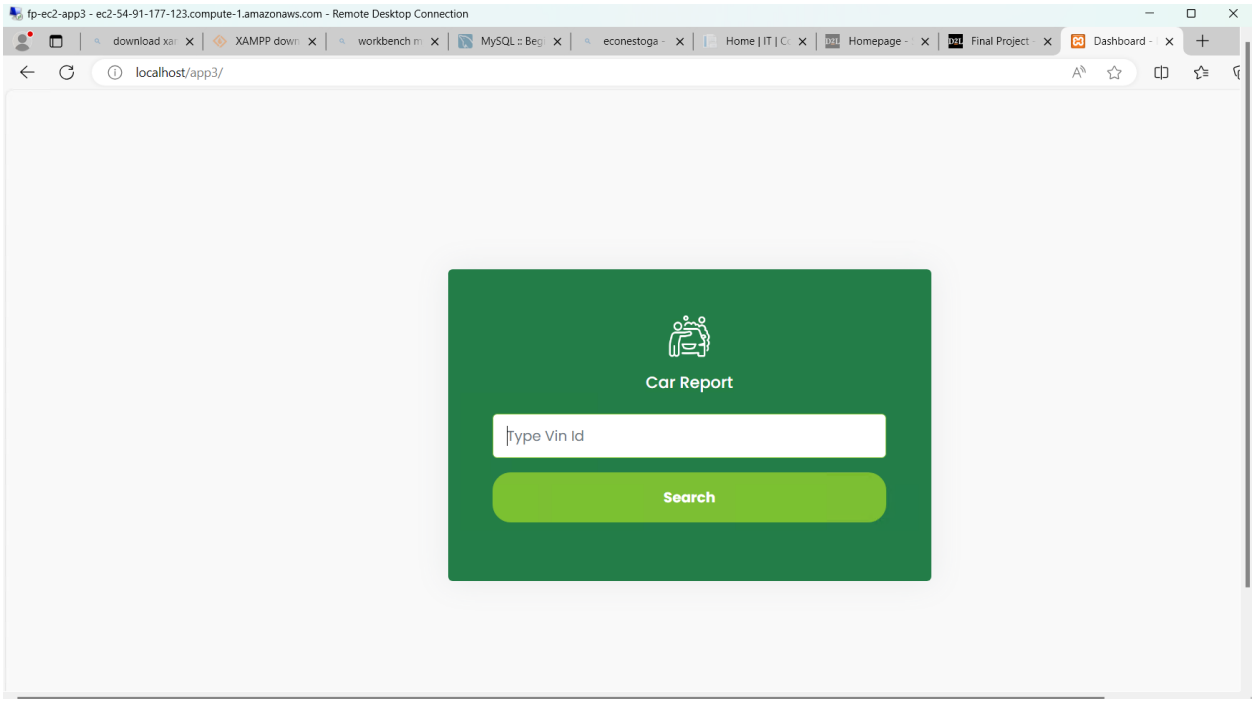

Figure 14 -Displaying the localhost/app3

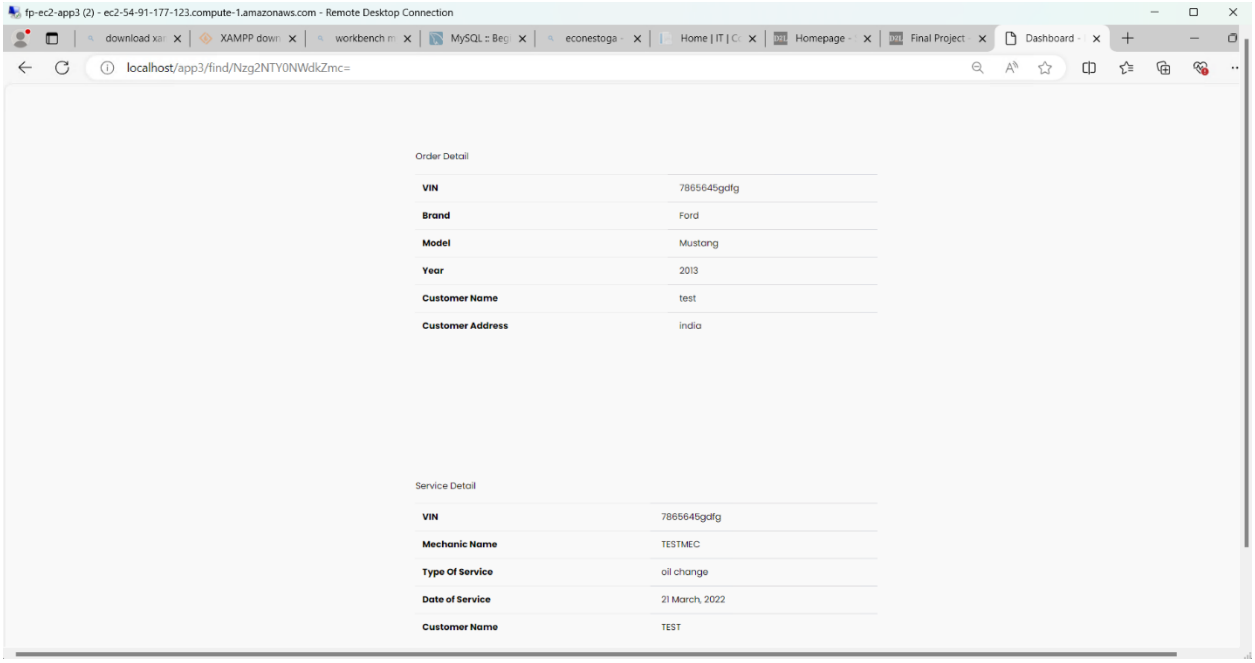

Figure 15 – Entered the vin id and got the order details.

Reflections

Based on the instructions provided, three integrated applications on the AWS are ready. This process includes creating EC2 instances, creating environments, migrating applications, and configuring RDS databases. Every application was hosted on another AWS account to have a distributed and scalable structure.

Key Achievements:

App1: We used first AWS account for implementing the App1. Car Manufacturing was established on an EC2 instance that came as a package with an RDS database, meaning that the root for storing data was created.

App2: We used second AWS account for implementing the App2. Car Dealership was launched on another EC2 instance having reference to its database from RDS and was also successfully integrated with App1 to fetch data from it.

App3: we used third account to show the integration part. App3 known as Car Details was of web application nature and was installed with Docker at an EC2 instance, intended for pulling and presenting information from both App1 and App2.

By correct arrangements of security groups, environment, and network settings, each application now communicates, which forms an extremely reliable system that mimics real-world car manufacturing, dealers, and detailed car data collection. This deployment proves the usage of AWS services in the case of the distributed applications and the understanding of the management of complex integrations in the above-mentioned cloud environment.