MIDDLE EAST TECHNICAL UNIVERSITY

NORTHERN CYPRUS CAMPUS

DEPARTMENTOF

COMPUTER ENGINEERING

CNG 352

Auto Maintenance & Management Platform (AMMP)

Hasan Eren Yarar

245702

Mehmet Fatih Çelik

2385286

Muntaser Maruf Bhuiyan

2413532

PROJECT DESCRIPTION

The Auto Maintenance & Management Platform (AMMP) is a web application system designed to help vehicle owners keep track of all their electronic and mechanical maintenance, as well as schedule appointments with repair shops for regular maintenance and repair work. With AMMP, users can create user accounts, add their vehicles, and input maintenance data such as oil changes,tire rotations, and brake replacements. The platform provides reminders to users when the next maintenance is due based on their vehicle's mileage or time elapsed since the last maintenance. Additionally, users can schedule appointments with repair shops for regular maintenance and fixmechanical problems. The system will send notifications to users to remind them of upcoming appointments and provide information on the status of their vehicle's repairs. AMMP is user- friendly and efficient in terms of keeping a record of all maintenance-related activities and appointments, making it easier for vehicle owners to keep track of their vehicle's maintenance history and ensure that their vehicle is always in good condition. The system will be implementedas a web application that can be accessed through a browser, and will be named Auto Maintenance & Management Platform (AMMP) to reflect its key features and capabilities.

Data requirements

User: Each user will need to register their account through their name, email address, password and phone number. Creating an account will allow them to access the web application and use it accordingly. Using the account on the platform they can manage their vehicles and each vehicles maintenance records. Each user may have one or more vehicles associated with their account. They will be able to make appointments to repair shops according to their maintenance requirement through this app on the schedule appointed for each shop.

Vehicle: Each vehicle has a unique Vehicle ID, Make, Model, Year, and VIN (Vehicle Identification Number). The Vehicle entity represents a vehicle that a user has added to their account on the platform. A user can have multiple cars added under their name. Each vehicle will also have their maintenance records stored. These data are input by the users.

Maintenance: Each maintenance record has a unique Record ID, Date, Type of Maintenance, and Cost. The Maintenance Record entity represents a record of maintenance activities performed on a particular vehicle. The maintenance type is divided into two types: Mechanical and regular. Mechanical is when the vehicle needs major change in specific car parts where the vehicle will be left to the shop. The duration attribute stores the duration time the car might be left in the shop for fixing, which is input by the mechanic of shop. This duration attribute is also used to notify the user of the job being done. Meanwhile, the regular maintenance are the types which needs to be changed after specific period, such as oil change, brake fluid change. The frequency attribute stores the time interval after which the maintenance needs to be done and notifies the used based on the frequency data. The maintenance information is inserted into the system by the specific shop personnel where the maintenance is done. Maintenance data can be accessed by the shop after the appointment to that shop is done by user so that the shop can suggest the user what sort of maintenance will be better for the vehicle.

Repair shop: Each repair shop has a unique shop ID, Name, Address, and Phone Numbers. This information is put in by a shop personnel and registers into the system as shop where the users can make appointments to schedule maintenance or get suggestion on the maintenance to be done on their registered vehicle in the system. Each repair shop can register more than one phone numbers. These repair shop can access and view their appointments available to them for the day and week to them via their schedule.

Appointment: Each appointment has a unique Appointment ID, Date of the appointment, userID of the user making the appointment ,shop ID of shop where appointment is to be done, maintenanceID which records the maintenance details performed during the appointment and scheduleID which checks if the relevant time slot or period is available for appointment. The status of the appointment will be decided by repair shop and notified to the user.

Schedule: It contains uniquely generated scheduleID, starting time, and end time. The schedule entity requires a way to store the start and end times of a scheduled appointment, as well as a unique identifier for each of it. These attributes can be used to retrieve and manage scheduled timeslots which can be used to determine if the user can make an appointment in the shop during that timeslot.

Transaction requirements

Data entry

- Enter the details of a new user (such as Dan Evans, with email danevans@gmail.com, having password 48+9lo, with phone number 05338579896)
- Enter the details of a user's new vehicle (such as sports car VO1, Audi R8, 2007 model)
- Enter the details of new repair shops (such as ProTech Auto Repair, with PTArepair@gmail.com, located in Kalkanli, Guzelyurt, can be contacted via 05789678546)
- Enter shop name and, date and time to make appointment (ProTech Auto Repair on 20th March 2023, 14:00)
- Enter maintenance record performed on the vehicle (such as Changing brake fluids on 20th March 2023, for 3300 TL, which needs to be changed regularly at an 8-month interval)
- Enter the scheduled timeslot for the appointment (scheduled appointment starts on May 1st, 2023, at 10:00 AM and ends at 12:00 PM on the same day)

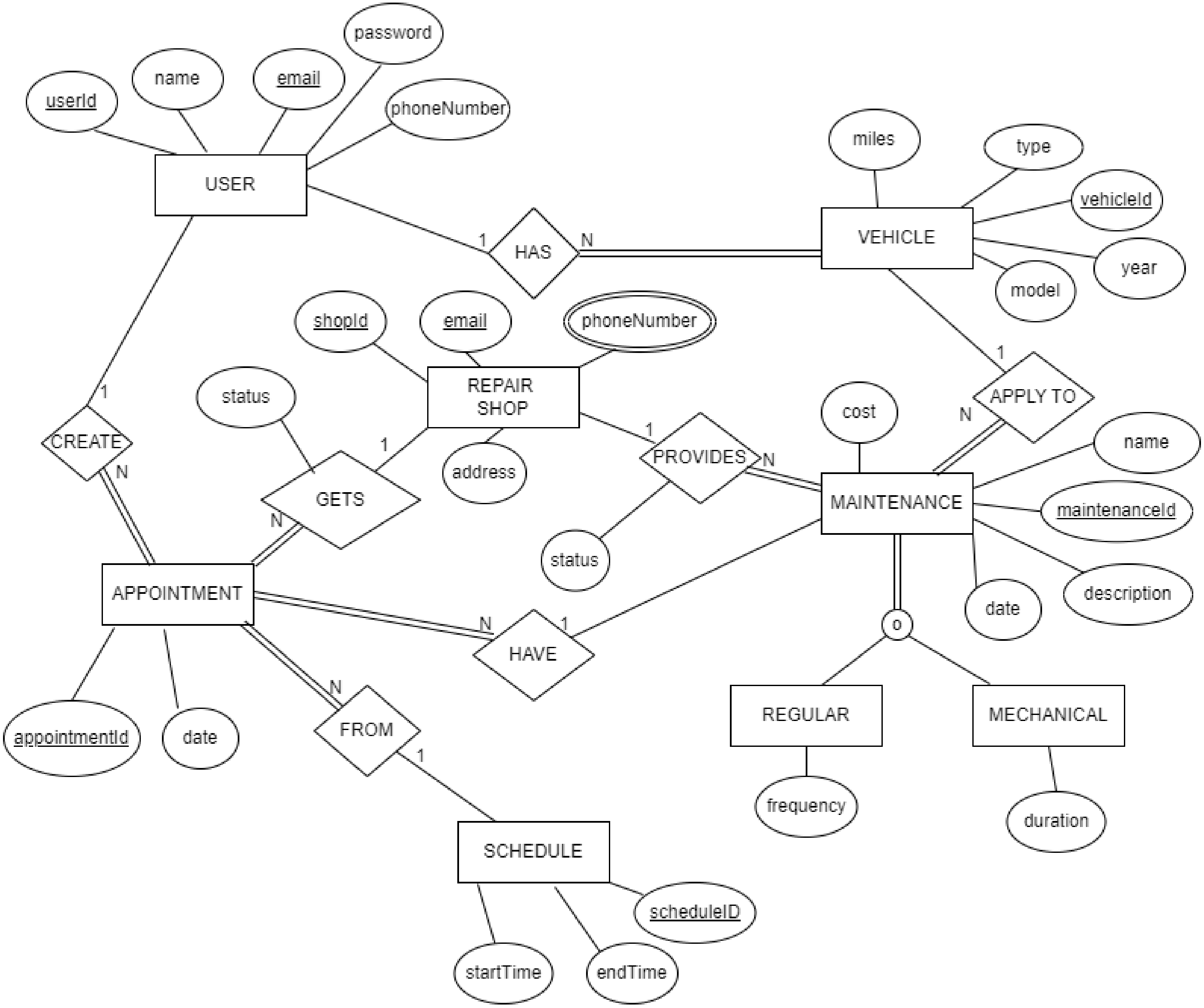
Data update/delete

- Update/delete the details of the vehicle
- Update/delete the details of the user
- Update/delete the details of the repair shops
- Update/delete the details of the appointment
- Update/delete the details of the scheduled timeslots

Data queries

- List all the details of the vehicles owned by the user
- List all the appointments of the repair shops each day in the repair shop view
- List all the maintenance records of each of the user vehicle per year
- Identify the most visited repair shop
- List the name of the vehicles according to their model year.
- Identify the total cost maintenances based on monthly and annually.
- Identify the vehicles that are under maintenance to the user
- List the details of the shop based on the name of the shop
- Identify the vehicle details based on the name of the vehicle
- List the vehicle's details according to the closest maintenance date
- List the available time slot schedule suitable for appointment to user for the specific shop
- Identify the regular maintenance that needs to be done within the next period of time
- List all the appointment for a specific type of maintenance record
- Count the maintenance record performed in each specific shop for user
- List and sort the vehicles based on the total number of maintenances done to them for the shop

ER DIAGRAM:



ASSUMPTIONS

- We assume that there is just one phone number for each user in the system.

Table Before the Normalization:

USER

UserID (PK)	Email(PK)	Name	Password	phoneNumber

APPOINTMENT

AppointmentID (PK)	date	Status

REPAIR SHOP

ShopID (PK)	email	address	phoneNumber

VEHICLE

VehicleID (PK)	Type	Model	Year	Miles

SCHEDULE

ScheduleID (PK)	StartTime	EndTime

MAINTENANCE

MaintenanceID (PK)	Name	Cost	Description	Date

REGULAR

MaintenanceID (PK)	Frequency

MECHANICAL

MaintenanceID (PK)	Duration

Functional Dependencies

FD1: {UserID} -> {Email, Name, Password, PhoneNumber}
FD2: {AppointmentID} -> {Date, UserID, ShopID, ScheduleID, status}
FD3: {ShopID} -> {ShopEmail, Address, ShopPhoneID }
FD4: {VehicleID} -> {Type, Model, Year, Miles, UserID}
FD5: {ScheduleID} -> {StartTime, EndTime}
FD6: {MaintenanceID} -> {Name, Cost, Description, Date, ShopID, Frequency, Duration }

FD7: {UserID,Email,Password} -> {Name, PhoneNumber}
FD8: {MaintenanceID, ShopID, VehicleID } -> { UserID, Name, Cost, Description, Date, Frequency, Duration }
FD9: {VehicleID,UserID} -> { Type, Model, Year, Miles }
FD10: { AppointmentID, ShopID, ScheduleID } -> { UserID, Date, status }
FD11: { AppointmentID, UserID } -> { UserName, Date, ShopID, ScheduleID, status }
FD12: {Email} -> {UserID, Name, Password, PhoneNumber}

Normalisations

1NF

The REPAIR SHOP has a multivalued attribute called phoneNumber, which violates the 1NF. Therefore, we split the REPAIR SHOP table into two separate tables: SHOP and SHOP_PHONE

Repair_SHOP_PHONE

R_ShopPhoneID	PhoneNumber

2NF

In the 2NF schema, we have ensured that every non-key attribute is dependent on the whole primary key, and there are no partial dependencies. This means that the schema is in 2NF.

3NF

In this case, each table in the 3NF schema has no transitive dependencies. Each non-key attribute depends only on the primary key, and there are no dependencies between non-key attributes. Therefore, the schema is in 3NF.

BCNF:

All these tables meet the BCNF condition because every determinant is a super key

After the Normalization steps:

USER

UserID	Email	Name	Password

APPOINTMENT

Appointment ID	date	A_UserID [FK:USER :UserID]	A_ShopID[FK: Repair Shop:ShopID]	A_MaintenanceID[FK: Maintenance: MaintenanceID]	A_ ScheduleID [FK: SCHEDULE: ScheduleID]	Status

REPAIR SHOP

ShopID	email	address

Repair_SHOP_PHONE

R_ShopPhoneID [FK:Repair Shop:ShopID]	*PhoneNumber*

VEHICLE

VehicleID	Type	Model	Year	Miles	V_UserID[FK:USER :UserID]

SCHEDULE

ScheduleID	StartTime	EndTime

MAINTENANCE

MaintenanceID	Name	Cost	Description	Date	M_ VehicleID [FK: VEHİCLE:VehicleID]	M_ShopID[FK:Repair Shop:ShopID]

REGULAR

MaintenanceID[FK: Maintenance: MaintenanceID]	Frequency

MECHANICAL

MaintenanceID[FK: Maintenance: MaintenanceID]	Duration

Database Definition Queries

```
import mysql.connector

mydb = mysql.connector.connect(
        host = "localhost",
        user = "root",
        passwd = "123456.",
        port = "3307",
        database = "aamp"
        )

#Creating Cursor Instance
myCursor = mydb.cursor()

# Create the User table
myCursor.execute("""
    CREATE TABLE User (
        userId INTEGER AUTO_INCREMENT PRIMARY KEY,
        email VARCHAR(100) NOT NULL,
        name VARCHAR(100) NOT NULL,
        password VARCHAR(100) NOT NULL
    )
""")


# create the RepairShop table
myCursor.execute("""
    CREATE TABLE RepairShop (
        shopId INTEGER AUTO_INCREMENT PRIMARY KEY,
        email VARCHAR(100) NOT NULL,
        address VARCHAR(255) NOT NULL
    )
""")


# create the Schedule table
myCursor.execute("""
    CREATE TABLE Schedule (
        scheduleId INTEGER AUTO_INCREMENT PRIMARY KEY,
        startTime TIMESTAMP NOT NULL,
        endTime TIMESTAMP NOT NULL
    )
""")
```

```
# create the Vehicle table
myCursor.execute("""
    CREATE TABLE Vehicle (
        vehicleId INTEGER AUTO_INCREMENT PRIMARY KEY,
        type VARCHAR(100) NOT NULL,
        model VARCHAR(100),
        year YEAR(4),
        miles INTEGER,
        userId INTEGER NOT NULL,
        FOREIGN KEY(userId) REFERENCES User(userId)
    )
""")

# create the Maintenance table
myCursor.execute("""
    CREATE TABLE Maintenance (
        maintenanceId INTEGER AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(100) NOT NULL,
        cost DECIMAL(10,2) UNSIGNED NOT NULL,
        description VARCHAR(255),
        maintenanceDate DATETIME NOT NULL,
        vehicleId INTEGER NOT NULL,
        shopId INTEGER NOT NULL,
        FOREIGN KEY(vehicleId) REFERENCES Vehicle(vehicleId),
        FOREIGN KEY(shopId) REFERENCES RepairShop(shopId)
    )
""")

# create the Appointment table
myCursor.execute("""
    CREATE TABLE Appointment (
        appointmentId INTEGER AUTO_INCREMENT PRIMARY KEY,
        userId INTEGER NOT NULL,
        shopId INTEGER NOT NULL,
        maintenanceId INTEGER NOT NULL,
        scheduleId INTEGER NOT NULL,
        FOREIGN KEY(userId) REFERENCES User(userId),
        FOREIGN KEY(shopId) REFERENCES RepairShop(shopId),
        FOREIGN KEY(maintenanceId) REFERENCES Maintenance(maintenanceId),
        FOREIGN KEY(scheduleId) REFERENCES Schedule(scheduleId)
    )
""")

# create the RegularMaintenance table
myCursor.execute("""
    CREATE TABLE RegularMaintenance (
        maintenanceId INTEGER NOT NULL,
        frequency VARCHAR(100) NOT NULL,
        PRIMARY KEY (maintenanceId),
        FOREIGN KEY (maintenanceId) REFERENCES Maintenance(maintenanceId)
    )
""")

# create the MechanicalMaintenance table
myCursor.execute("""
    CREATE TABLE MechanicalMaintenance (
        maintenanceId INTEGER NOT NULL,
        duration VARCHAR(100) NOT NULL,
        PRIMARY KEY (maintenanceId),
        FOREIGN KEY (maintenanceId) REFERENCES Maintenance(maintenanceId)
    )
""")

# create the RepairShopPhones table
myCursor.execute("""
    CREATE TABLE RepairShopPhones (
        shopId INTEGER NOT NULL,
        phoneNumber VARCHAR(20) NOT NULL,
        PRIMARY KEY (shopId, phoneNumber),
        FOREIGN KEY (shopId) REFERENCES RepairShop(shopId)
    )
""")
mydb.commit()
```

Database Manipulation Queries

# **1-)** (USER)Find the top 10 user who has spent the most money on maintenance. We created this query to offer these users special promotions or discounts on future maintenance services, or to analyze their maintenance patterns and identify potential areas for cost savings.

```
myCursor.execute("""SELECT User.userId, User.name, SUM(Maintenance.cost) AS total_cost
                                  FROM User
                                  JOIN Vehicle ON User.userId = Vehicle.userId
                                  JOIN Maintenance ON Vehicle.vehicleId = Maintenance.vehicleId
                                  GROUP BY User.userId
                                  ORDER BY total_cost DESC
                                  LIMIT 10
""")
result = myCursor.fetchall()
for row in result:
        print(row)
```

# **2-)** (REPAIR SHOP)List the adress and phone numbers of all repair shops that have performed maintenance on a vehicle that has over 50,000 miles with the maintenance name. We created this query to identify repair

shops that have experience performing maintenance on vehicles with high mileage. This information could be used by users to find repair shops that are experienced in working on vehicles with similar mileage and maintenance needs.

```
myCursor.execute("""SELECT RepairShop.email, RepairShop.address, MIN(RepairShopPhones.phoneNumber)
as phoneNumber, Maintenance.name
                                        FROM RepairShop
                                        JOIN Maintenance ON RepairShop.shopId = Maintenance.shopId
                                        JOIN Vehicle ON Maintenance.vehicleId = Vehicle.vehicleId
                                        JOIN      RepairShopPhones      ON      RepairShop.shopId      =
RepairShopPhones.shopId
                                        WHERE Vehicle.miles > 50000
                                        GROUP BY RepairShop.address, Maintenance.name;
""")
result = myCursor.fetchall()
for row in result:
        print(row)
```

# **3-)** (REPAIR SHOP)Get the number of appointments and total cost of maintenance for each repair shop. We created this query to give an overview of maintenance costs and frequency for each vehicle model. It can help users make informed decisions about vehicle purchases and manufacturers improve their vehicles.

```
myCursor.execute("""SELECT      RepairShop.address,      COUNT(Appointment.appointmentId)      as
numAppointments, SUM(Maintenance.cost) as totalCost
                                        FROM RepairShop
                                        JOIN Appointment ON RepairShop.shopId = Appointment.shopId
                                        JOIN      Maintenance      ON      Appointment.maintenanceId      =
Maintenance.maintenanceId
                                        GROUP BY RepairShop.address;
""")
result = myCursor.fetchall()
for row in result:
        print(row)
```

# **4-)** (VEHICLE)Get the total cost of maintenance and the number of appointments for each vehicle model. We created this query to provide an overview of maintenance costs and frequency for each vehicle model. It can help users make informed decisions about vehicle purchases and manufacturers improve their vehicles.

```
myCursor.execute("""SELECT  Vehicle.model,  COUNT(Appointment.appointmentId)  as  numAppointments,
SUM(Maintenance.cost) as totalCost
                                        FROM Vehicle
                                        JOIN User ON Vehicle.userId = User.userId
                                        JOIN Appointment ON User.userId = Appointment.userId
                                        JOIN      Maintenance      ON      Appointment.maintenanceId      =
Maintenance.maintenanceId
                                        GROUP BY Vehicle.model;
""")
result = myCursor.fetchall()
for row in result:
        print(row)

# 5-) (VEHICLE)Get the most recent maintenance performed on each vehicle along with the name of the repair shop that performed it. We created this query to provide information on the most recent maintenance performed on each vehicle and the repair shop that performed it. It can help users keep track of their vehicle's maintenance history and make informed decisions about future maintenance.

myCursor.execute("""SELECT Vehicle.vehicleId, Vehicle.model, Maintenance.name, RepairShop.address, Maintenance.maintenanceDate
                                        FROM Vehicle
                                        JOIN Maintenance ON Vehicle.vehicleId = Maintenance.vehicleId
                                        JOIN RepairShop ON Maintenance.shopId = RepairShop.shopId
                                        WHERE Maintenance.maintenanceDate = (
                                                SELECT MAX(maintenanceDate)
                                                FROM Maintenance
                                                WHERE Maintenance.vehicleId = Vehicle.vehicleId
                                        )
                                        ORDER BY Maintenance.maintenanceDate DESC;
""")
result = myCursor.fetchall()
for row in result:
        print(row)

# Inserting a new user, their vehicle, and the vehicle's maintenance history

# Defining Insert commands
user_query = "INSERT INTO User (email, name, password, phoneNumber) VALUES (%s, %s, %s, %s)"
vehicle_query = "INSERT INTO Vehicle (type, model, year, miles, userId) VALUES (%s, %s, %s, %s, %s)"
maintenance_query = "INSERT INTO Maintenance (name, cost, description, maintenanceDate, vehicleId, shopId) VALUES (%s, %s, %s, %s, %s, %s)"

# Inputting the data
user_data = ('yyeliz@metu.edu.tr', 'Yeliz Yesilada', 'password123', "05369945787")
vehicle_data = ('Sedan', 'Toyota Corolla', 2021, 10500, 1)
maintenance_data = [
  ('Oil Change', 599.99, 'Replace engine oil and filter', '2023-06-11 10:30:00', 2, 1),
  ('Tire Rotation', 399.99, 'Rotate tires for even wear', '2022-05-21 11:30:00', 2, 1),
  ('Brake Inspection', 1000.00, 'Check brake pads', '2021-03-08 17:50:00', 2, 1)
]

# Executing user query
myCursor.execute(user_query, user_data)
```

```
# Get last inserted user ID
user_id = myCursor.lastrowid

# Update vehicle data with last inserted user ID
vehicle_data = (*vehicle_data[:4], user_id)

# Execute vehicle query
myCursor.execute(vehicle_query, vehicle_data)

# Get last inserted vehicle ID
vehicle_id = myCursor.lastrowid

# Update maintenance data with last inserted vehicle ID
maintenance_data = [(name, cost, description, maintenanceDate, vehicle_id, shopId) for name, cost, description, maintenanceDate, vehicleId, shopId in maintenance_data]

# Execute maintenance query for each maintenance item
for data in maintenance_data:
  myCursor.execute(maintenance_query, data)

mydb.commit()
```

#UPDATE QUERIES

# **1-)**Update the phone number for a specific repair shop:

```
# define the SQL statement to update RepairShopPhones table's phoneNumber attribute according to specific shopId
sql = """UPDATE RepairShopPhones
                SET phoneNumber = '536 994 57 87'
                WHERE shopId =
                (SELECT shopId
                 FROM RepairShop
                 WHERE shopId = 30);"""

# execute the SQL statement
myCursor.execute(sql)

mydb.commit()
```

# **2-)**Update the maintenance frequency for all regular maintenance tasks associated with a specific vehicle model

```
sql = """UPDATE RegularMaintenance
SET frequency = 'Every 6 months'
```

```
WHERE maintenanceId IN
    (SELECT maintenanceId
     FROM Maintenance
     WHERE vehicleId IN
         (SELECT vehicleId
          FROM Vehicle
          WHERE model = 'Audi A4'));"""

# execute the SQL statement
myCursor.execute(sql)

mydb.commit()

# 3-)Update the duration of all mechanical maintenance performed on a vehicle within a certain mileage range

sql = """UPDATE MechanicalMaintenance
            SET duration = "2 hour 30 minutes"
            WHERE maintenanceId IN (
            SELECT maintenanceId
            FROM Maintenance
            WHERE vehicleId IN (
                        SELECT vehicleId
                        FROM Vehicle
                        WHERE miles >= 65000 AND miles <= 90000
            )
            )"""

# execute the SQL statement
myCursor.execute(sql)

mydb.commit()

#DELETE QUERIES

# 1-)Delete all regular maintenance records that are associated with vehicles with less than 10,000 miles

sql = """DELETE FROM RegularMaintenance
WHERE maintenanceId IN (
    SELECT maintenanceId FROM Maintenance
    JOIN Vehicle ON Maintenance.vehicleId = Vehicle.vehicleId
    WHERE Vehicle.miles < 10000
);
            """

# execute the SQL statement
myCursor.execute(sql)
```

```
mydb.commit()

# 2-)Delete all mechanical maintenance records for vehicles that have more than 100,000 miles

sql = """DELETE FROM MechanicalMaintenance
WHERE maintenanceId IN (
    SELECT maintenanceId
    FROM Maintenance
    WHERE vehicleId IN (
        SELECT vehicleId
        FROM Vehicle
        WHERE miles > 90000
    )
);
    """

# execute the SQL statement
myCursor.execute(sql)

mydb.commit()

# 3-)Delete all appointments that were scheduled outside of business hours

DELETE FROM Appointment
WHERE scheduleId IN (
  SELECT scheduleId
  FROM Schedule
  WHERE HOUR(startTime) < 9 OR HOUR(endTime) > 17
);
```

NOTE: Since we have 100 tuples for each table, we didn’t include data population queries in the report but we submitted them in the zip file.

DISCUSSION

Our application has a total of nine tables, namely User, Schedule, Vehicle, Maintenance, Appointment, RepairShop, RegularMaintenance, MechanicalMaintenance, and RepairShopPhones. The workload of our application heavily depends on the number of users and repair shops registered in the system, and the frequency of the queries and updates associated with them. If there are many appointments and maintenance requests being made at the same time, the Schedule table may experience a high workload. Similarly, if there are many different types of vehicles and maintenance services offered by repair shops, the Maintenance and MechanicalMaintenance, RegularMaintenance tables may have a higher workload.

Out of all the tables, Appointment, Maintenance, and Schedule are expected to be updated most frequently, while User, RepairShop, and Vehicle tables will have a lower frequency of updates. The number of records in each table will also vary. Tables like User, RepairShop, Vehicle will have fewer records compared to tables like Maintenance, Appointment, Schedule.

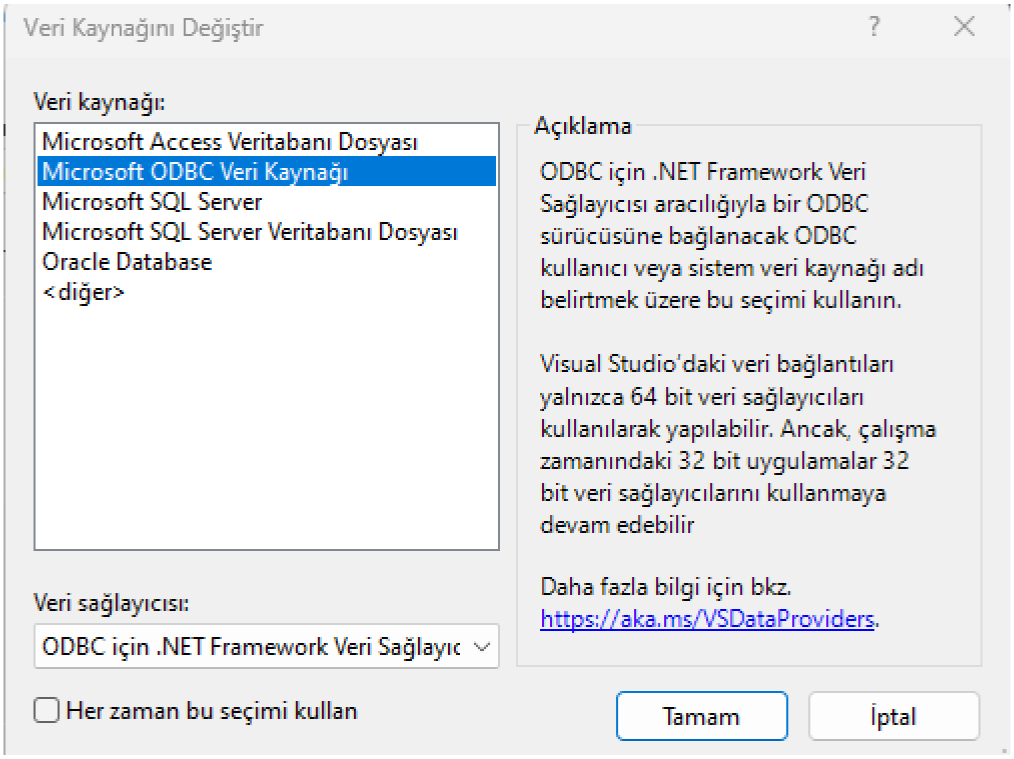
Since this application is Cyprus-based, the number of tuples in each table is not expected to be very high, which will not put a significant strain on the application. we need to consider which columns are frequently used with WHERE, JOIN, GROUP BY, ORDER BY operations to optimize query performance.

For example, since the Vehicle table has a foreign key reference to the User table, creating an index on the userId column in the Vehicle table could improve query performance for joins between the two tables.

Similarly, the Maintenance and Appointment tables both have foreign key references to the Vehicle, RepairShop, and User tables, so creating indexes on the corresponding columns in each table could improve performance for queries involving joins between these tables.

We chose not to use indexing in our Cyprus-based application due to specific requirements and the nature of the application. The quantity and quality of data did not warrant the development of indexes, and we modified our questions without using them. Moreover, a stable user base and data access pattern enabled us to achieve efficient data recovery without relying on indexes.

Modifications/changes on our database


Veri Kaynağını Değiştir
?
Veri kaynağı:
Microsoft Access Veritabanı Dosyası
Microsoft ODBC Veri Kaynağı
Microsoft SQL Server
Microsoft SQL Server Veritabanı Dosyası
Oracle Database
<diğer>
Açıklama
Tamam
İptal

ODBC için .NET Framework Veri Sağlayıcısı aracılığıyla bir ODBC sürücüsüne bağlanacak ODBC kullanıcı veya sistem veri kaynağı adı belirtmek üzere bu seçimi kullanın.

Visual Studio'daki veri bağlantıları yalnızca 64 bit veri sağlayıcıları kullanılarak yapılabilir. Ancak, çalışma zamanındaki 32 bit uygulamalar 32 bit veri sağlayıcılarını kullanmaya devam edebilir

Daha fazla bilgi için bkz. https://aka.ms/VSDataProviders.

Veri sağlayıcısı: ODBC için .NET Framework Veri Sağlayıc

☐ Her zaman bu seçimi kullan

As you can see, ASPNET gives permission to only Oracle, and Microsoft SQL Server. Since Microsoft SQL Server is not allowed, we decided to move our database to Oracle. We just changed our database sever, the remaining tables, columns remained the same.

Implementation Details

1. Language: C#
2. Framework: ASP.NET
3. Integrated Development Environment: Visual Studio
4. Database System: Oracle
5. Data Access: DataTableAdapters
6. Frontend Framework: Bootstrap

In developing our application, we decided to use ASP.NET and Visual Studio as our primary tools. We chose C# as our programming language for backend development. Visual Studio was our integrated development environment (IDE).

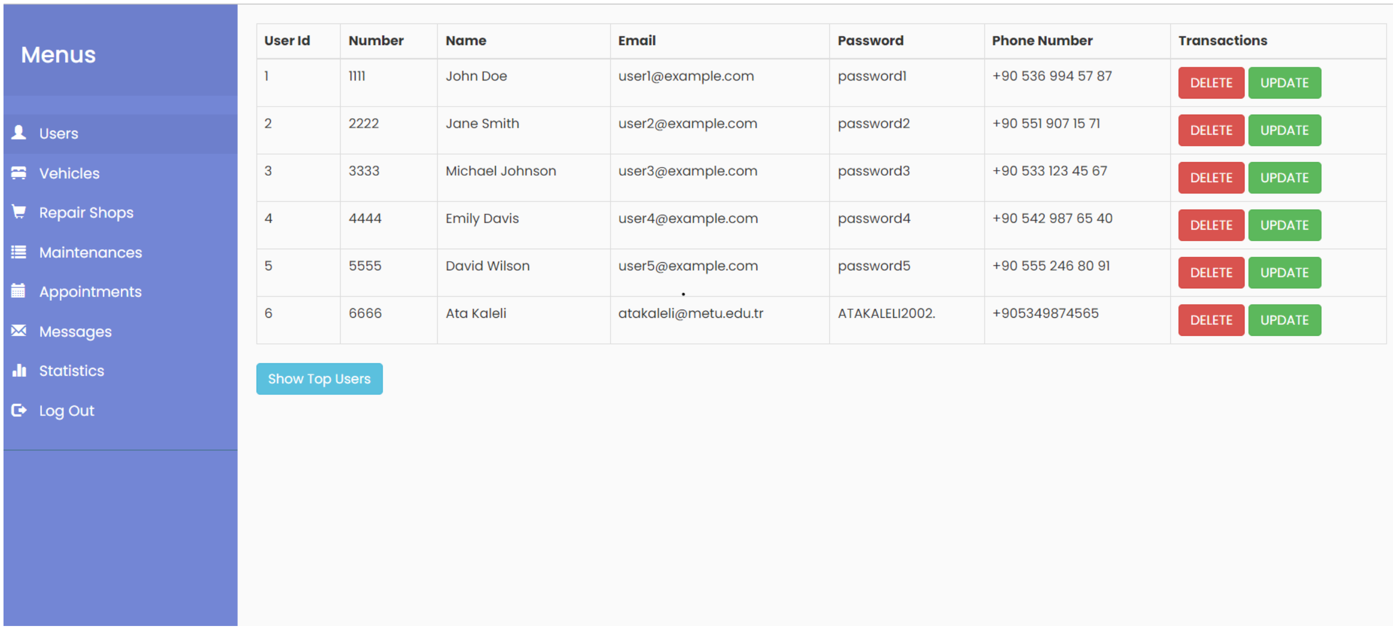
We chose Oracle as our database system to meet our data storage and recovery needs. For interaction with the database, we used DataTableAdapters, which are important parts of the ADO.NET framework in ASP.NET. These DataTableAdapters played an important role in executing SQL queries and loading data from the database. To ensure a flexible approach to data manipulation, we used strongly typed DataSets that provided type safety and flexible data manipulation.

At the front end, we have Bootstrap framework. It has collection of pre-built CSS and JavaScript components, we were able to create a visually appealing and functional user interface. This allowed us to deliver a unique user experience across devices and screen sizes.

By integrating ASP.NET, Visual Studio, Oracle, DataTableAdapters, and Bootstrap, we built user-friendly application that met our project goals.

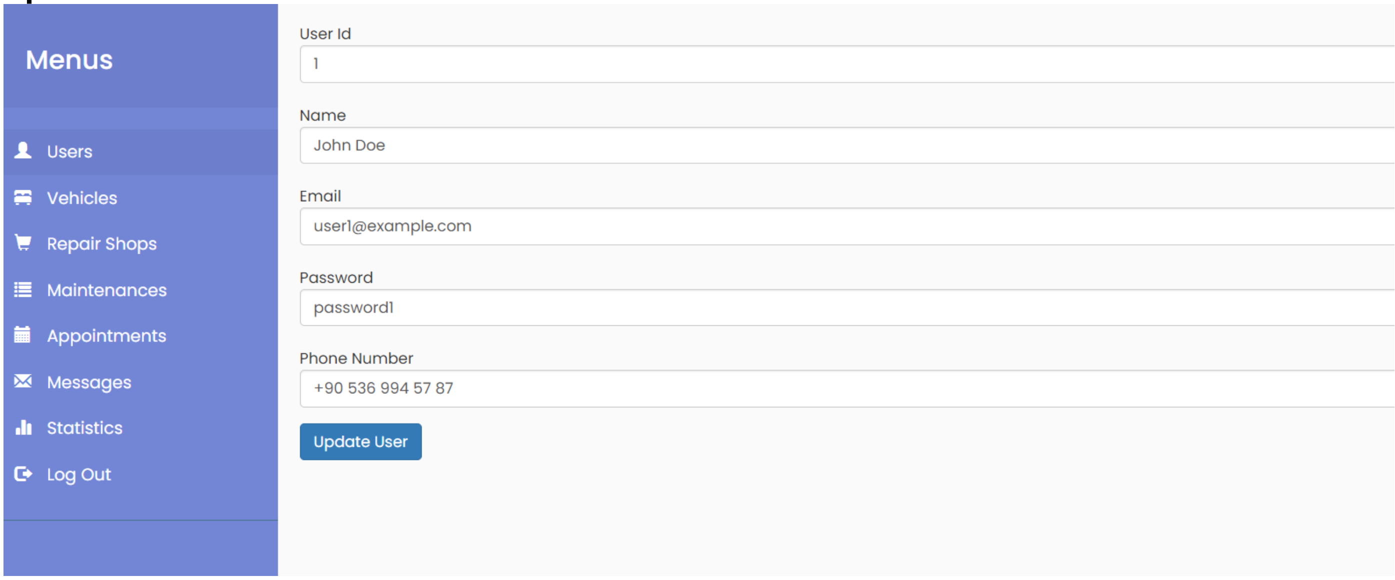
Screenshots

User List



User Id	Number	Name	Email	Password	Phone Number	Transactions
1	1111	John Doe	user1@example.com	password1	+90 536 994 57 87	DELETE UPDATE
2	2222	Jane Smith	user2@example.com	password2	+90 551 907 15 71	DELETE UPDATE
3	3333	Michael Johnson	user3@example.com	password3	+90 533 123 45 67	DELETE UPDATE
4	4444	Emily Davis	user4@example.com	password4	+90 542 987 65 40	DELETE UPDATE
5	5555	David Wilson	user5@example.com	password5	+90 555 246 80 91	DELETE UPDATE
6	6666	Ata Kaleli	atakaleli@metu.edu.tr	ATAKALELI2002.	+905349874565	DELETE UPDATE

Update User



User Id: 1

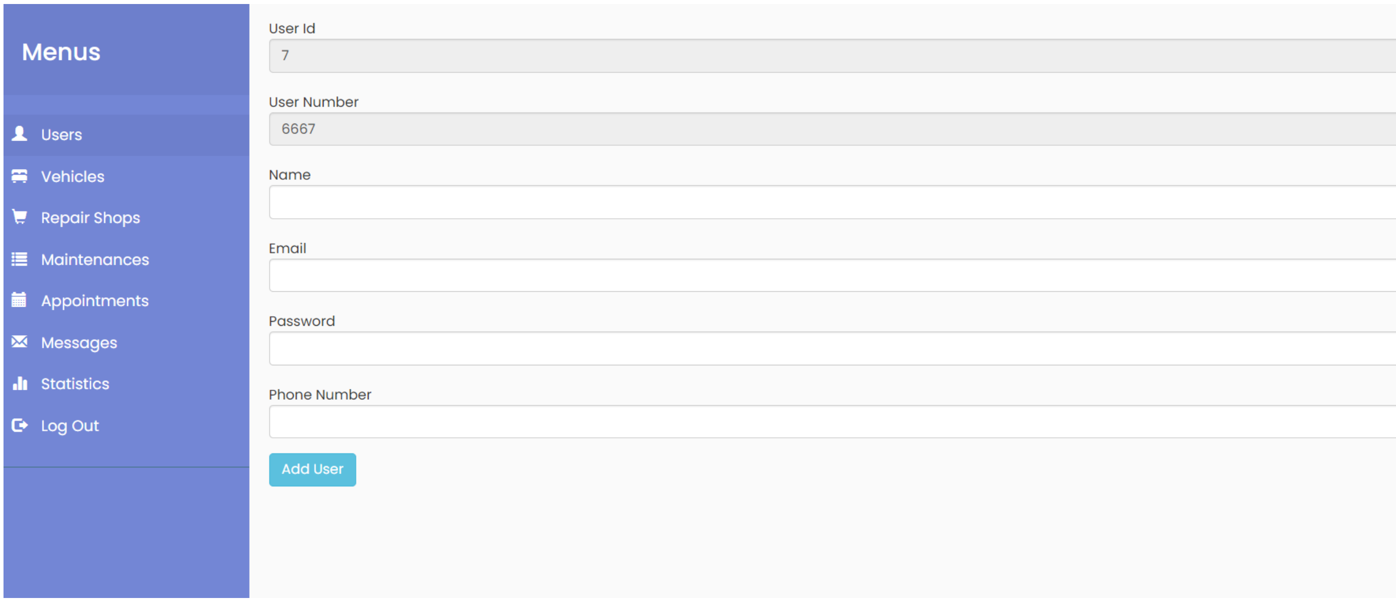
Name: John Doe

Email: user1@example.com

Password: password1

Phone Number: +90 536 994 57 87

Add User



User Id
7

User Number
6667

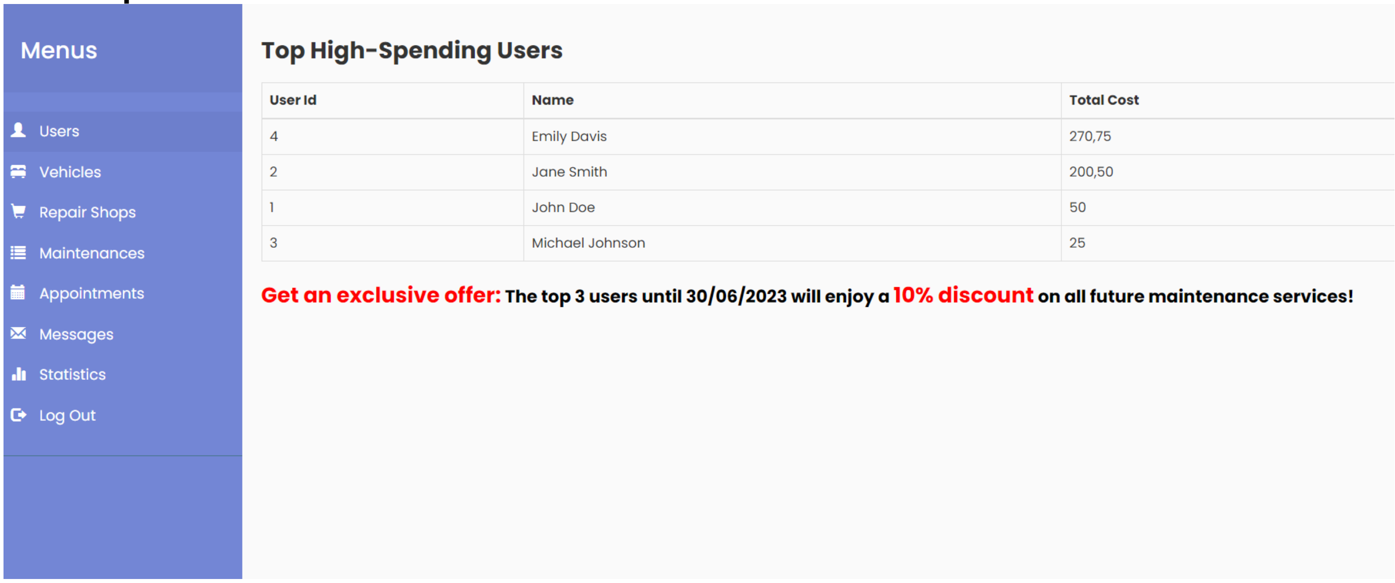
Name

Email

Password

Phone Number

Show Top Users

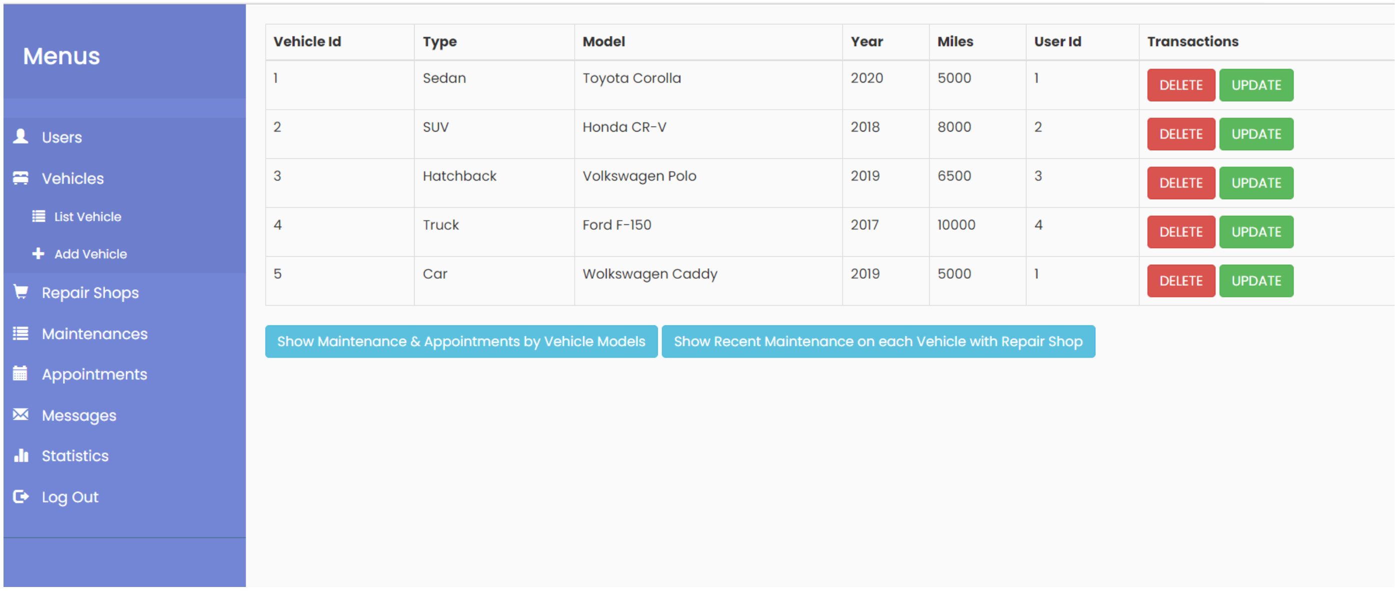


Top High-Spending Users

User Id	Name	Total Cost
4	Emily Davis	270,75
2	Jane Smith	200,50
1	John Doe	50
3	Michael Johnson	25

Get an exclusive offer: **The top 3 users until 30/06/2023 will enjoy a** **10% discount** **on all future maintenance services!**

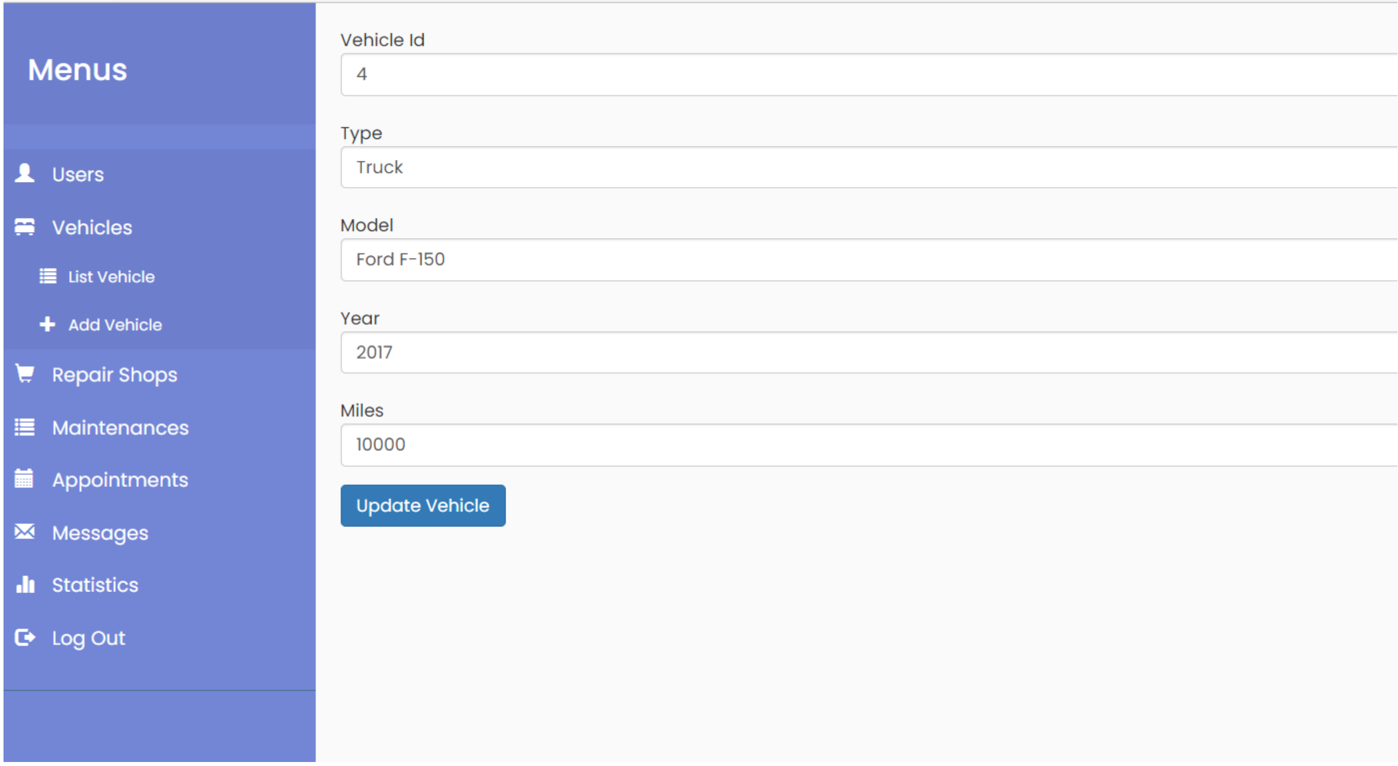
Vehicle List


Menus
Users
Vehicles
List Vehicle
Add Vehicle
Repair Shops
Maintenances
Appointments
Messages
Statistics
Log Out

Vehicle Id	Type	Model	Year	Miles	User Id	Transactions
1	Sedan	Toyota Corolla	2020	5000	1	DELETE UPDATE
2	SUV	Honda CR-V	2018	8000	2	DELETE UPDATE
3	Hatchback	Volkswagen Polo	2019	6500	3	DELETE UPDATE
4	Truck	Ford F-150	2017	10000	4	DELETE UPDATE
5	Car	Wolkswagen Caddy	2019	5000	1	DELETE UPDATE

Show Maintenance & Appointments by Vehicle Models
Show Recent Maintenance on each Vehicle with Repair Shop

Update Vehicle



Vehicle Id
4

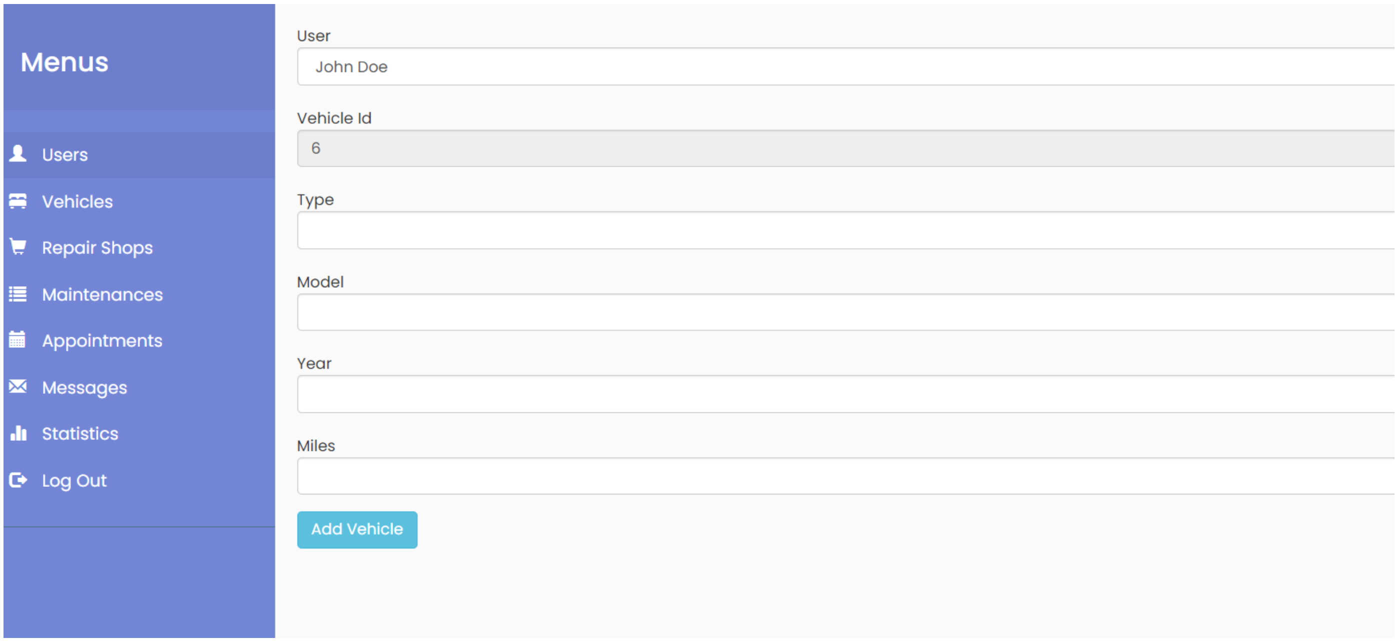
Type
Truck

Model
Ford F-150

Year
2017

Miles
10000

Add Vehicle



User

John Doe

Vehicle Id

6

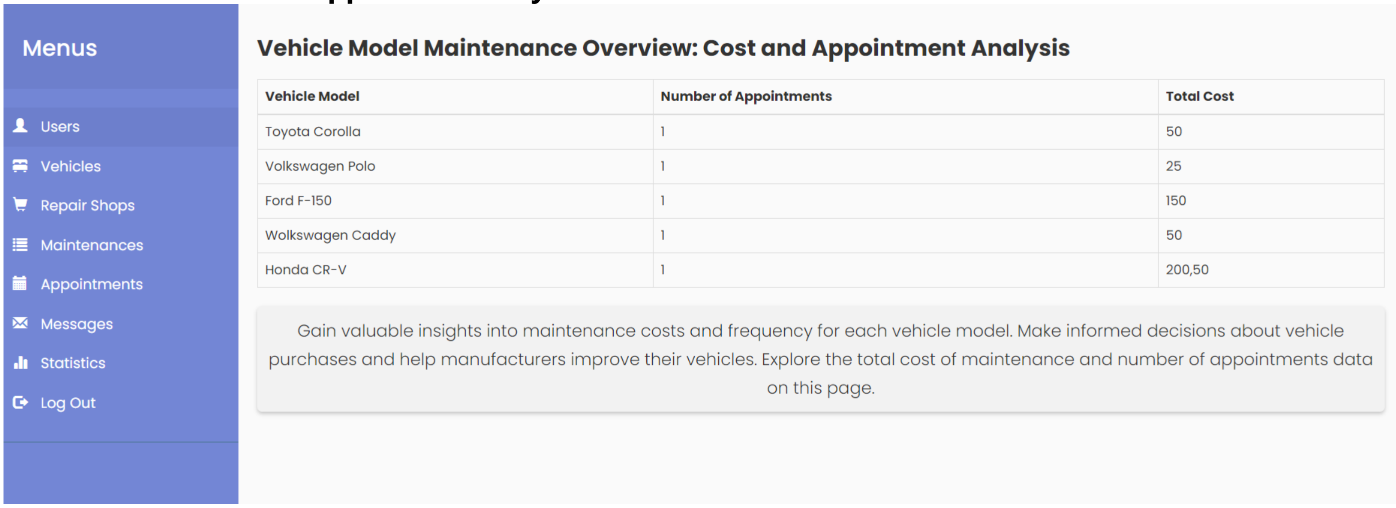
Type

Model

Year

Miles

Show Maintenance & Appointments By Vehicle Models

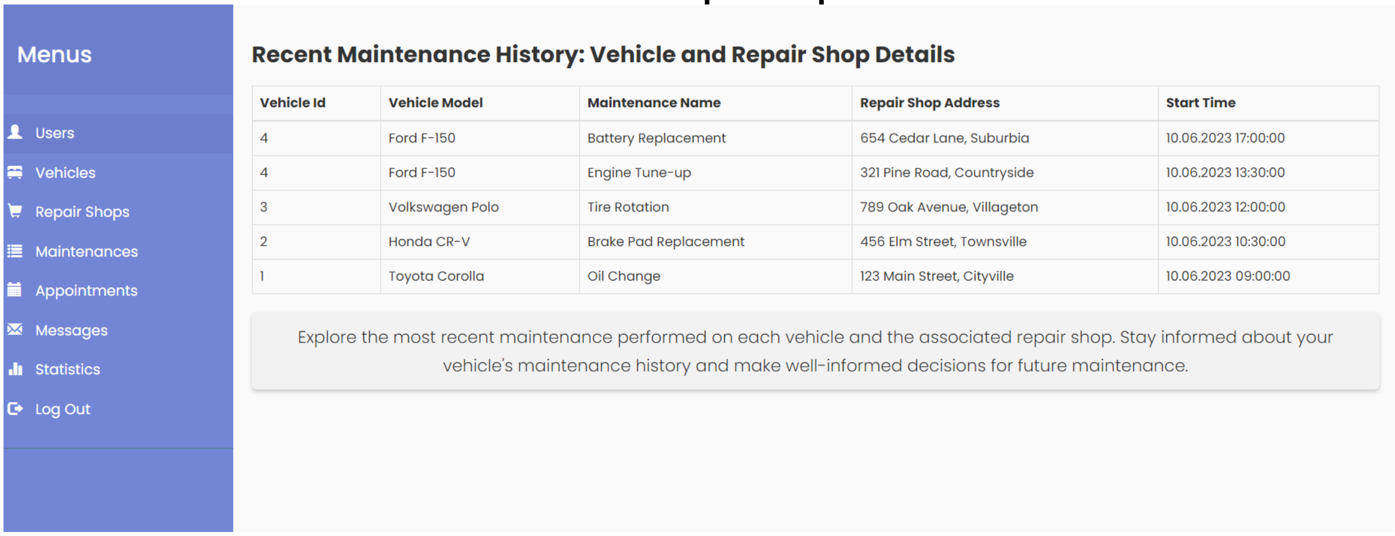


Vehicle Model Maintenance Overview: Cost and Appointment Analysis

Vehicle Model	Number of Appointments	Total Cost
Toyota Corolla	1	50
Volkswagen Polo	1	25
Ford F-150	1	150
Wolkswagen Caddy	1	50
Honda CR-V	1	200,50

Gain valuable insights into maintenance costs and frequency for each vehicle model. Make informed decisions about vehicle purchases and help manufacturers improve their vehicles. Explore the total cost of maintenance and number of appointments data on this page.

Show Recent Maintenance on each Vehicle with Repair Shop

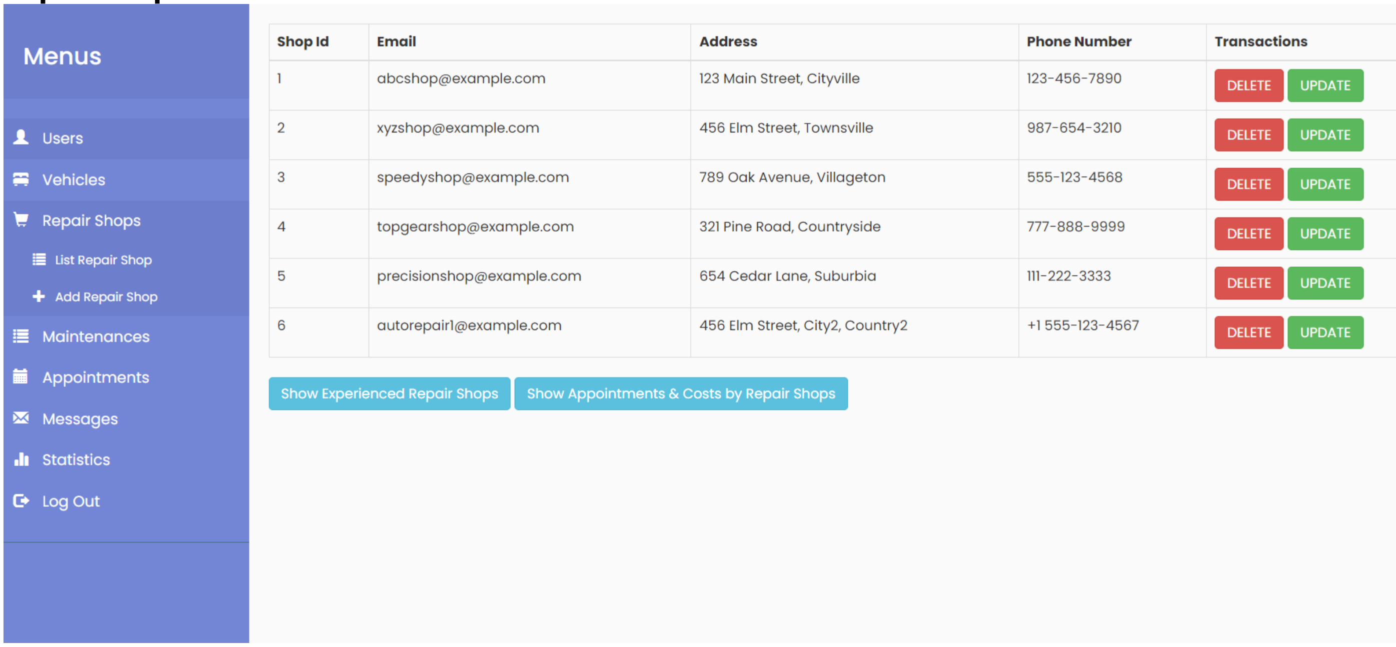


Recent Maintenance History: Vehicle and Repair Shop Details

Vehicle Id	Vehicle Model	Maintenance Name	Repair Shop Address	Start Time
4	Ford F-150	Battery Replacement	654 Cedar Lane, Suburbia	10.06.2023 17:00:00
4	Ford F-150	Engine Tune-up	321 Pine Road, Countryside	10.06.2023 13:30:00
3	Volkswagen Polo	Tire Rotation	789 Oak Avenue, Villageton	10.06.2023 12:00:00
2	Honda CR-V	Brake Pad Replacement	456 Elm Street, Townsville	10.06.2023 10:30:00
1	Toyota Corolla	Oil Change	123 Main Street, Cityville	10.06.2023 09:00:00

Explore the most recent maintenance performed on each vehicle and the associated repair shop. Stay informed about your vehicle's maintenance history and make well-informed decisions for future maintenance.

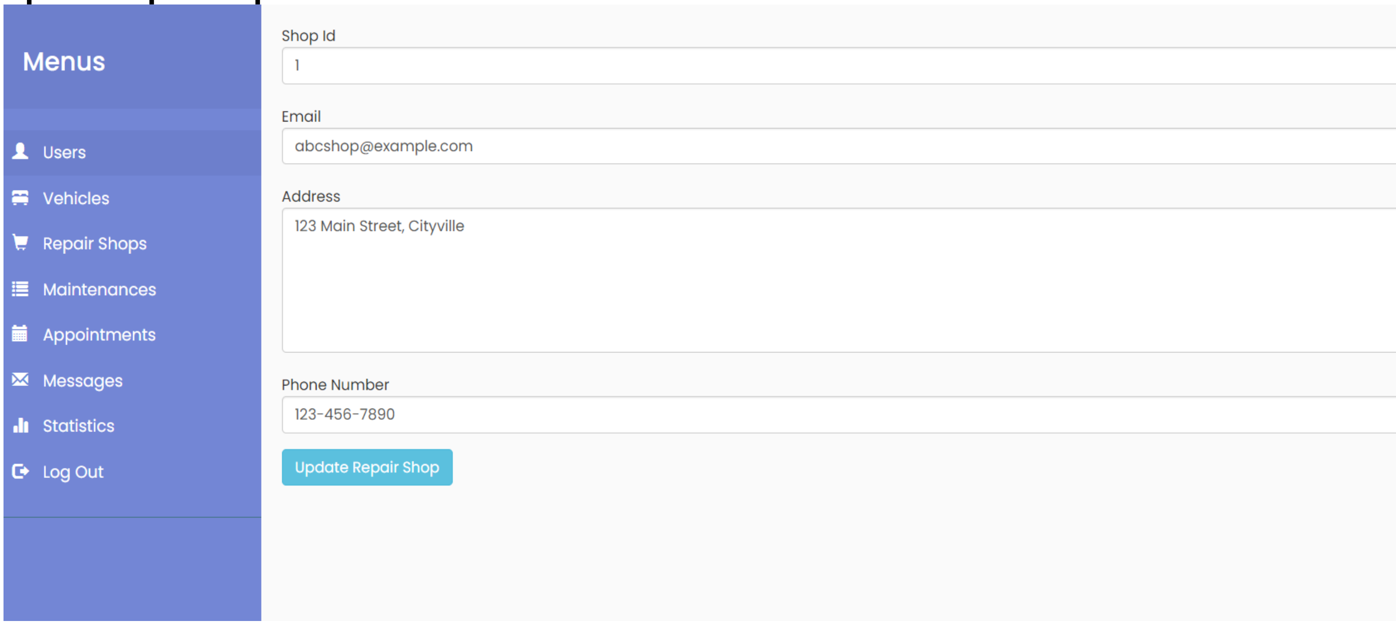
Repair Shop List



Shop Id	Email	Address	Phone Number	Transactions
1	abcshop@example.com	123 Main Street, Cityville	123-456-7890	DELETE UPDATE
2	xyzshop@example.com	456 Elm Street, Townsville	987-654-3210	DELETE UPDATE
3	speedyshop@example.com	789 Oak Avenue, Villageton	555-123-4568	DELETE UPDATE
4	topgearshop@example.com	321 Pine Road, Countryside	777-888-9999	DELETE UPDATE
5	precisionshop@example.com	654 Cedar Lane, Suburbia	111-222-3333	DELETE UPDATE
6	autorepair1@example.com	456 Elm Street, City2, Country2	+1 555-123-4567	DELETE UPDATE

Show Experienced Repair Shops
Show Appointments & Costs by Repair Shops

Update RepairShop



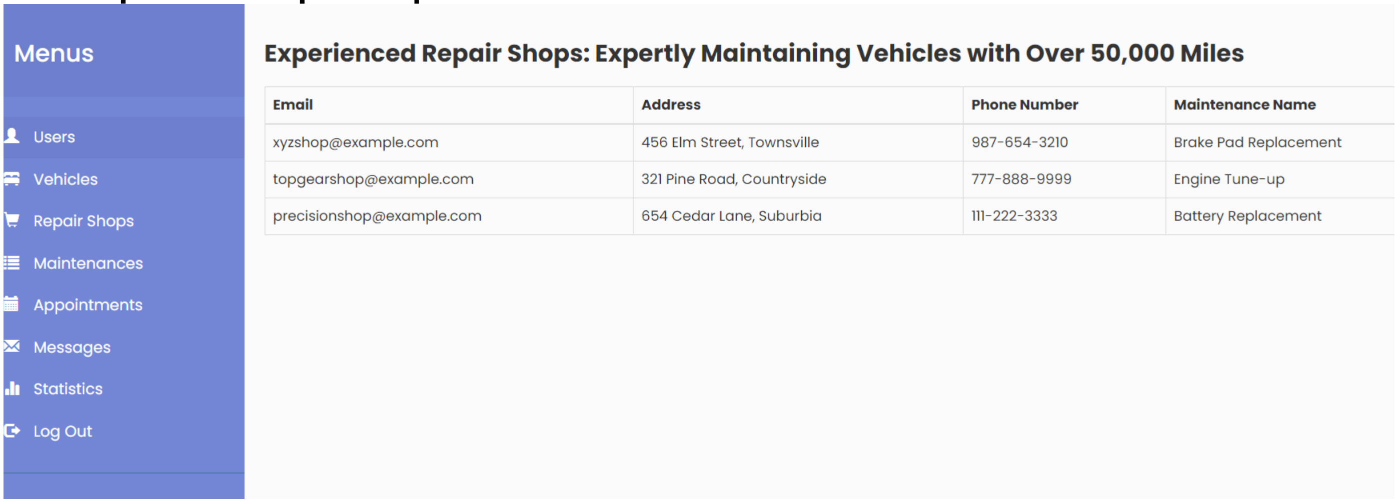
Shop Id
1

Email
abcshop@example.com

Address
123 Main Street, Cityville

Phone Number
123-456-7890

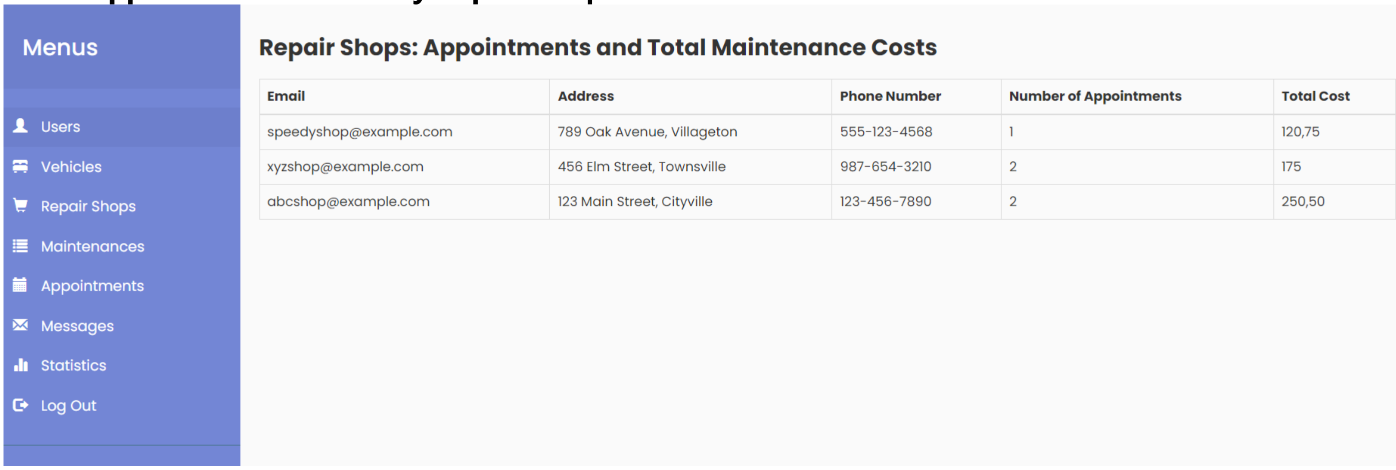
Show Experienced Repair Shops



Experienced Repair Shops: Expertly Maintaining Vehicles with Over 50,000 Miles

Email	Address	Phone Number	Maintenance Name
xyzshop@example.com	456 Elm Street, Townsville	987-654-3210	Brake Pad Replacement
topgearshop@example.com	321 Pine Road, Countryside	777-888-9999	Engine Tune-up
precisionshop@example.com	654 Cedar Lane, Suburbia	111-222-3333	Battery Replacement

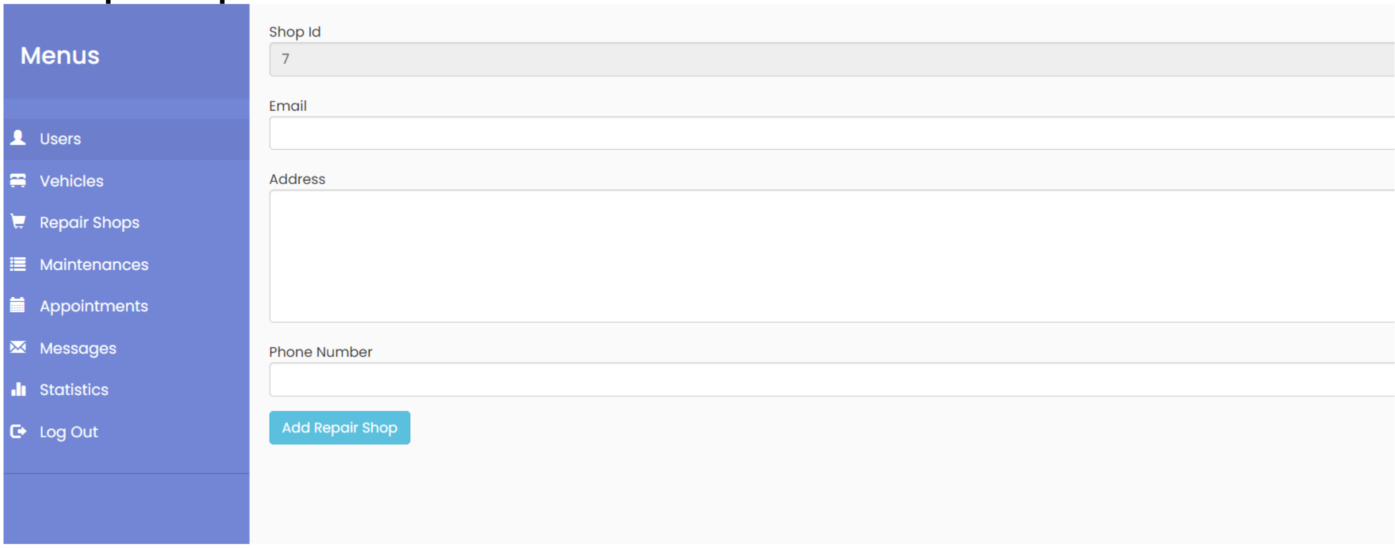
Show Appointments & Cost by Repair Shops



Repair Shops: Appointments and Total Maintenance Costs

Email	Address	Phone Number	Number of Appointments	Total Cost
speedyshop@example.com	789 Oak Avenue, Villageton	555-123-4568	1	120,75
xyzshop@example.com	456 Elm Street, Townsville	987-654-3210	2	175
abcshop@example.com	123 Main Street, Cityville	123-456-7890	2	250,50

Add Repair Shops



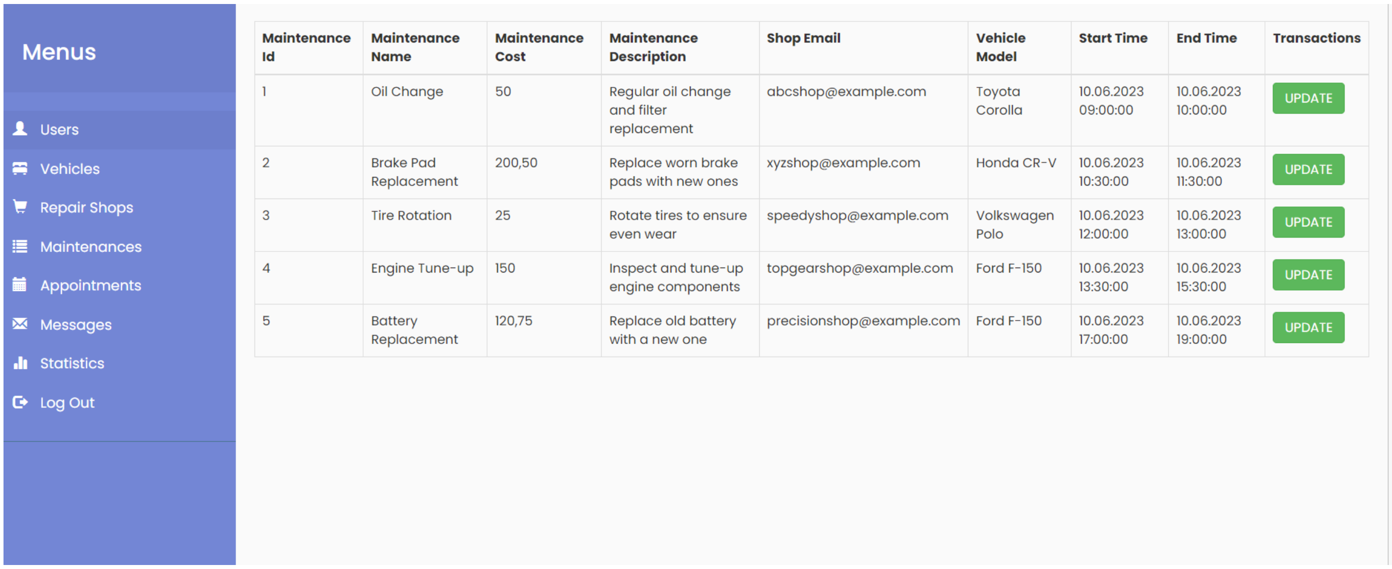
Shop Id: 7

Email

Address

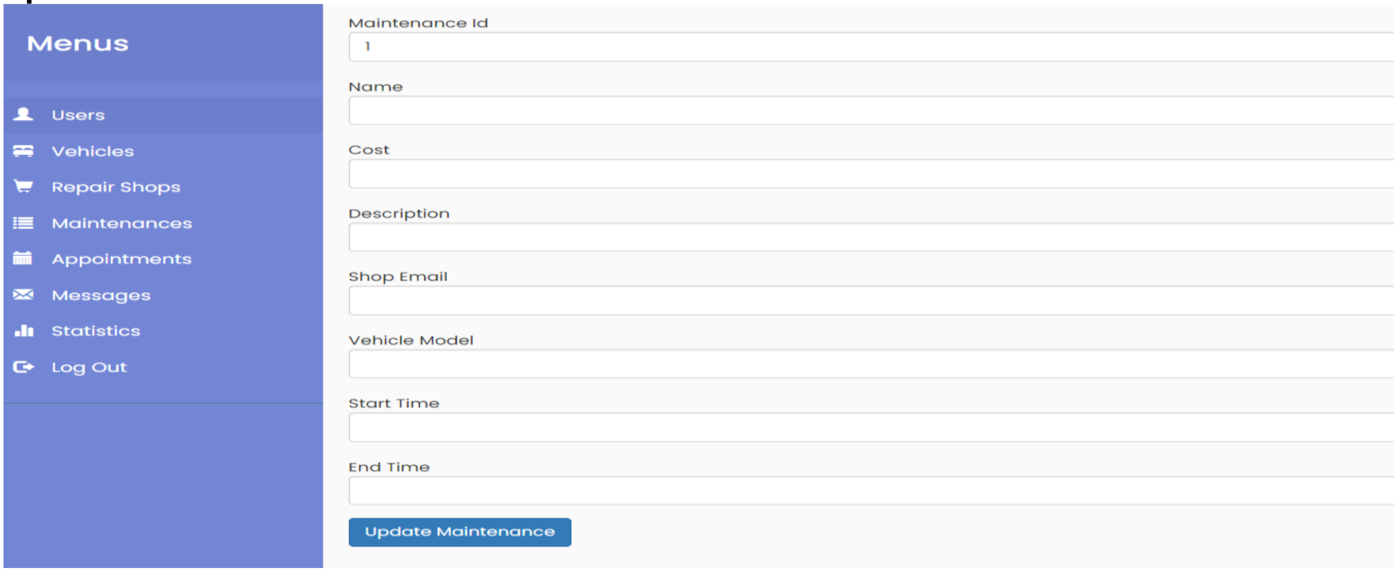
Phone Number

Maintenance List



Maintenance Id	Maintenance Name	Maintenance Cost	Maintenance Description	Shop Email	Vehicle Model	Start Time	End Time	Transactions
1	Oil Change	50	Regular oil change and filter replacement	abcshop@example.com	Toyota Corolla	10.06.2023 09:00:00	10.06.2023 10:00:00	UPDATE
2	Brake Pad Replacement	200,50	Replace worn brake pads with new ones	xyzshop@example.com	Honda CR-V	10.06.2023 10:30:00	10.06.2023 11:30:00	UPDATE
3	Tire Rotation	25	Rotate tires to ensure even wear	speedyshop@example.com	Volkswagen Polo	10.06.2023 12:00:00	10.06.2023 13:00:00	UPDATE
4	Engine Tune-up	150	Inspect and tune-up engine components	topgearshop@example.com	Ford F-150	10.06.2023 13:30:00	10.06.2023 15:30:00	UPDATE
5	Battery Replacement	120,75	Replace old battery with a new one	precisionshop@example.com	Ford F-150	10.06.2023 17:00:00	10.06.2023 19:00:00	UPDATE

Update Maintenance



Maintenance Id: 1

Name:

Cost:

Description:

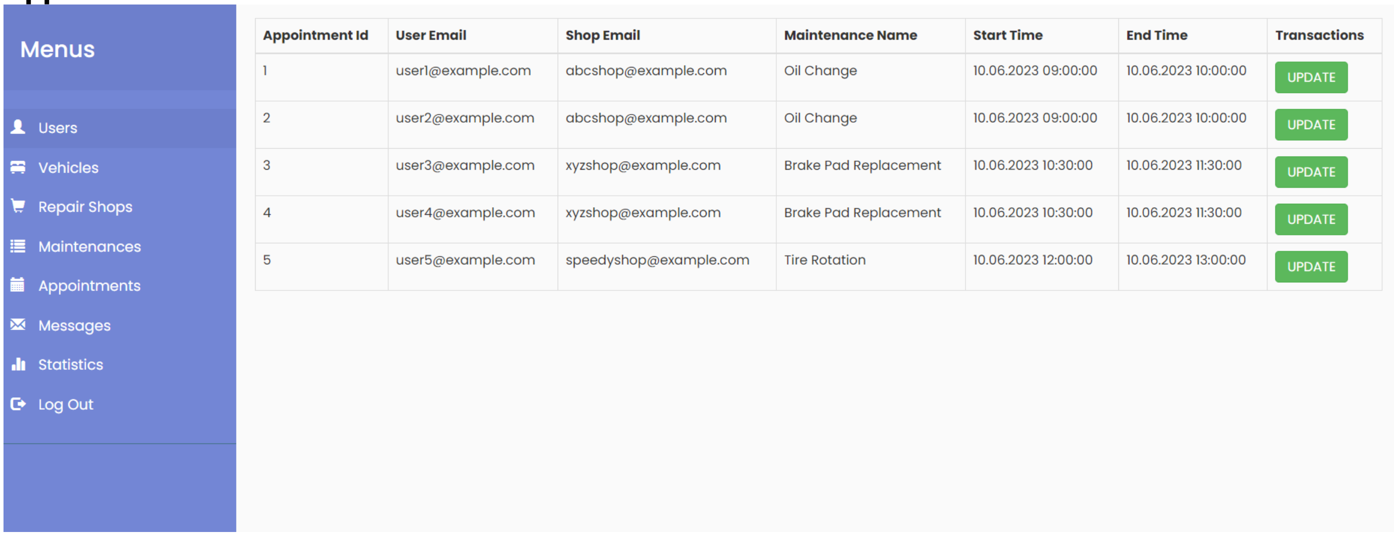
Shop Email:

Vehicle Model:

Start Time:

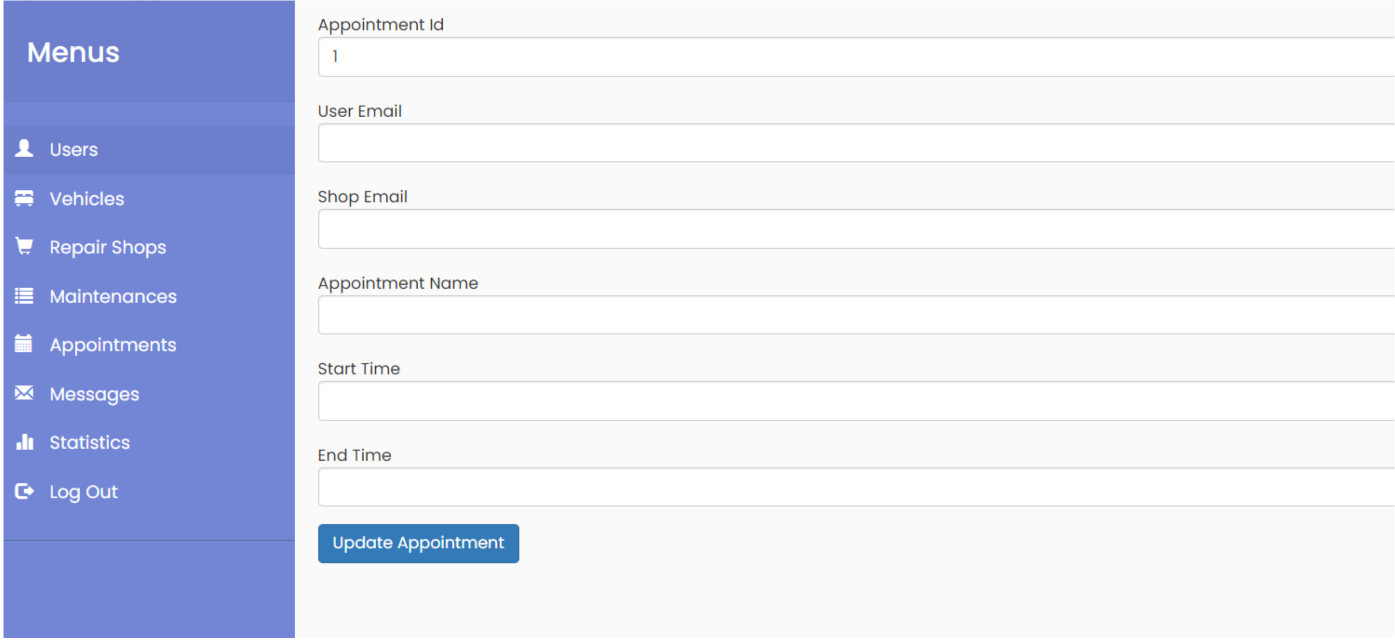
End Time:

Appointment List



Appointment Id	User Email	Shop Email	Maintenance Name	Start Time	End Time	Transactions
1	user1@example.com	abcshop@example.com	Oil Change	10.06.2023 09:00:00	10.06.2023 10:00:00	UPDATE
2	user2@example.com	abcshop@example.com	Oil Change	10.06.2023 09:00:00	10.06.2023 10:00:00	UPDATE
3	user3@example.com	xyzshop@example.com	Brake Pad Replacement	10.06.2023 10:30:00	10.06.2023 11:30:00	UPDATE
4	user4@example.com	xyzshop@example.com	Brake Pad Replacement	10.06.2023 10:30:00	10.06.2023 11:30:00	UPDATE
5	user5@example.com	speedyshop@example.com	Tire Rotation	10.06.2023 12:00:00	10.06.2023 13:00:00	UPDATE

Update Appointment



Appointment Id: 1

User Email:

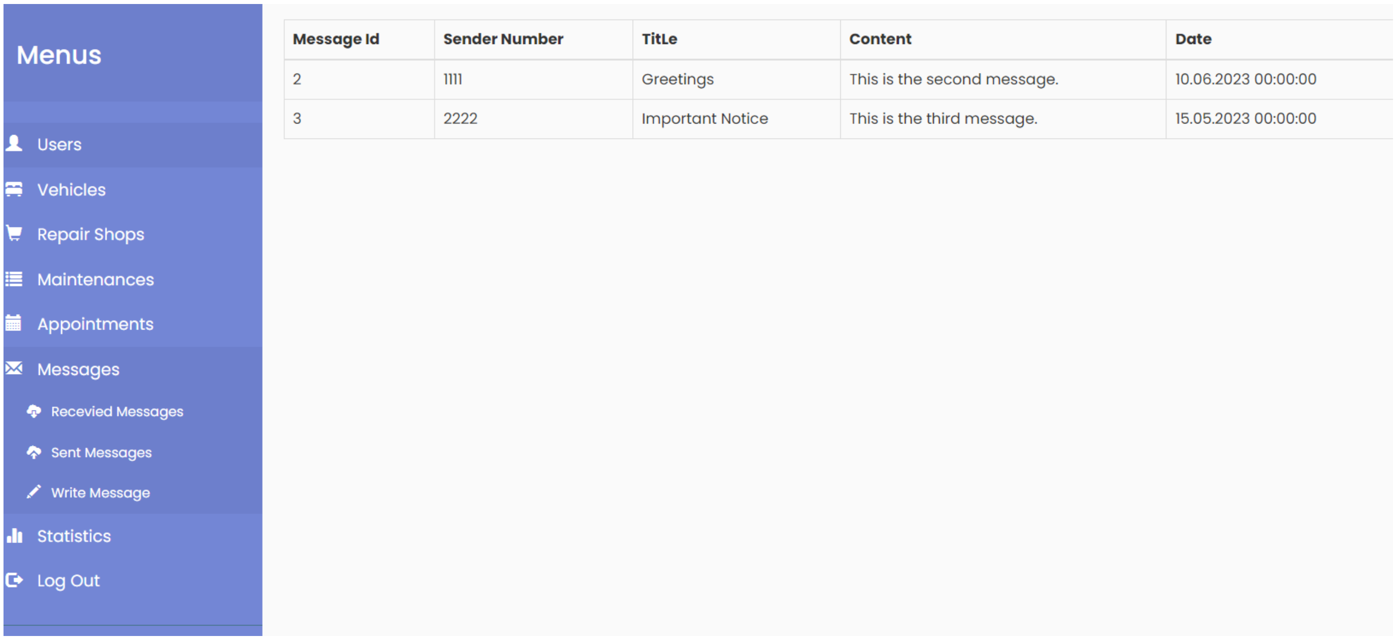
Shop Email:

Appointment Name:

Start Time:

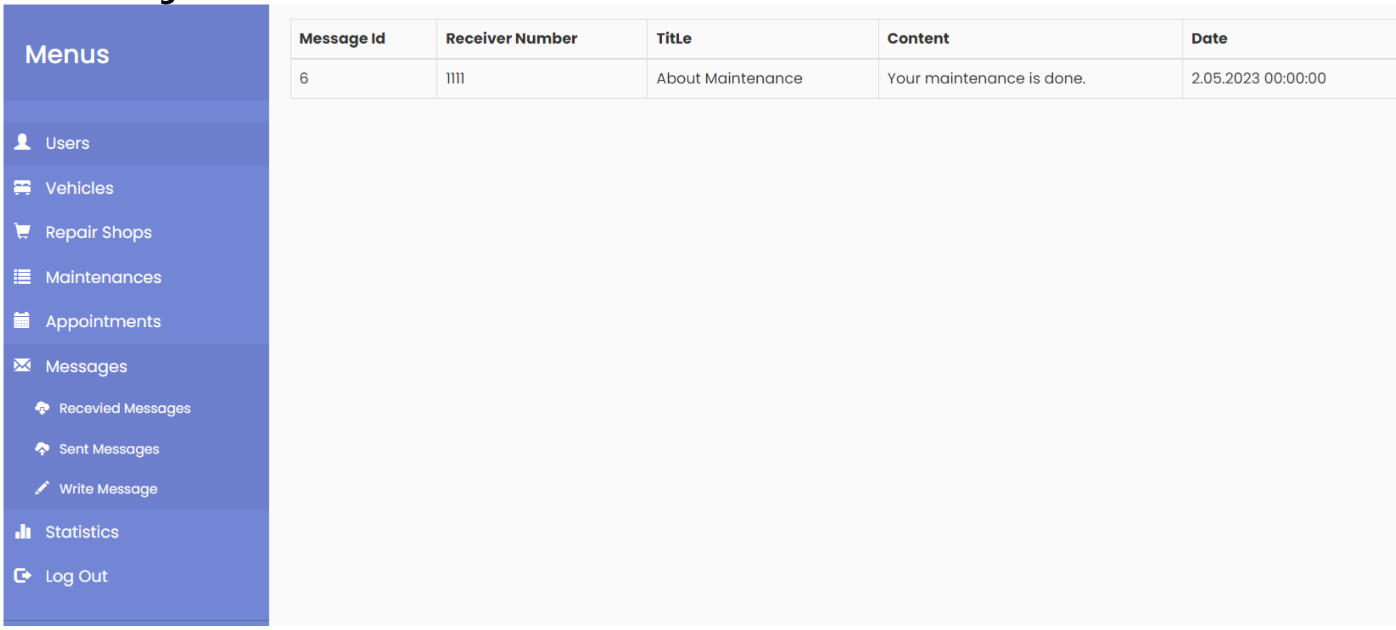
End Time:

Received Messages



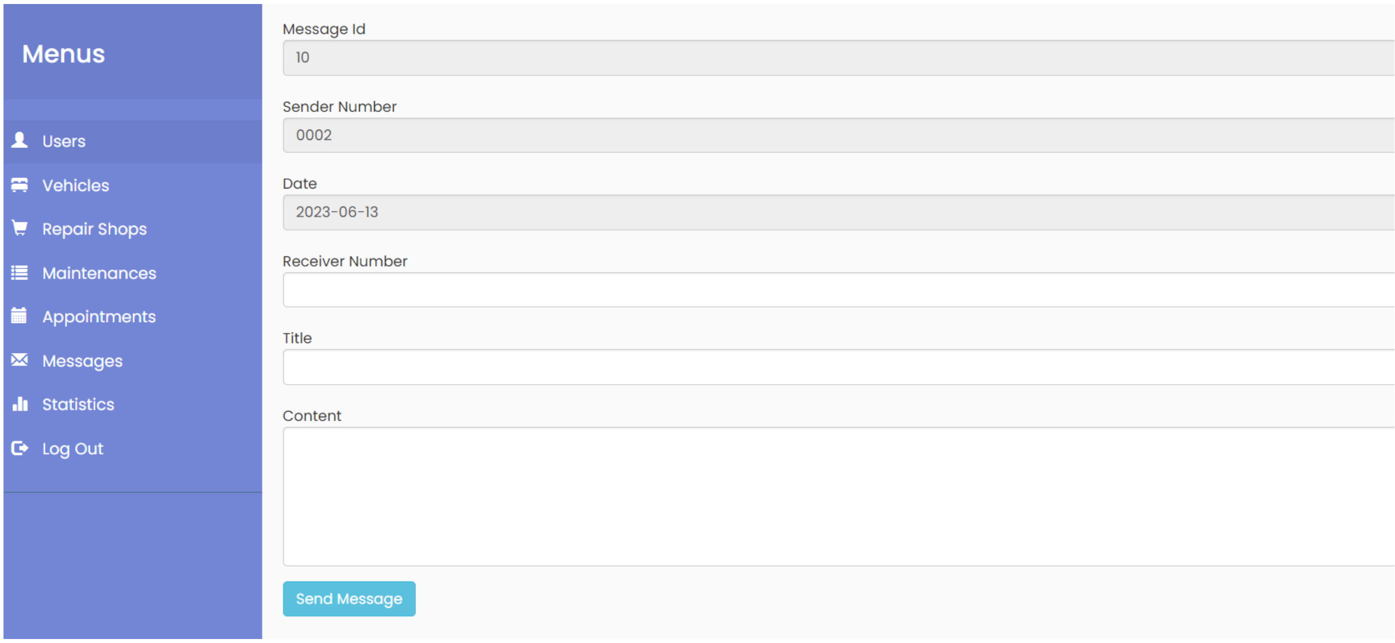
Message Id	Sender Number	TitLe	Content	Date
2	1111	Greetings	This is the second message.	10.06.2023 00:00:00
3	2222	Important Notice	This is the third message.	15.05.2023 00:00:00

Sent Messages



Message Id	Receiver Number	TitLe	Content	Date
6	1111	About Maintenance	Your maintenance is done.	2.05.2023 00:00:00

Write Message



Message Id
10

Sender Number
0002

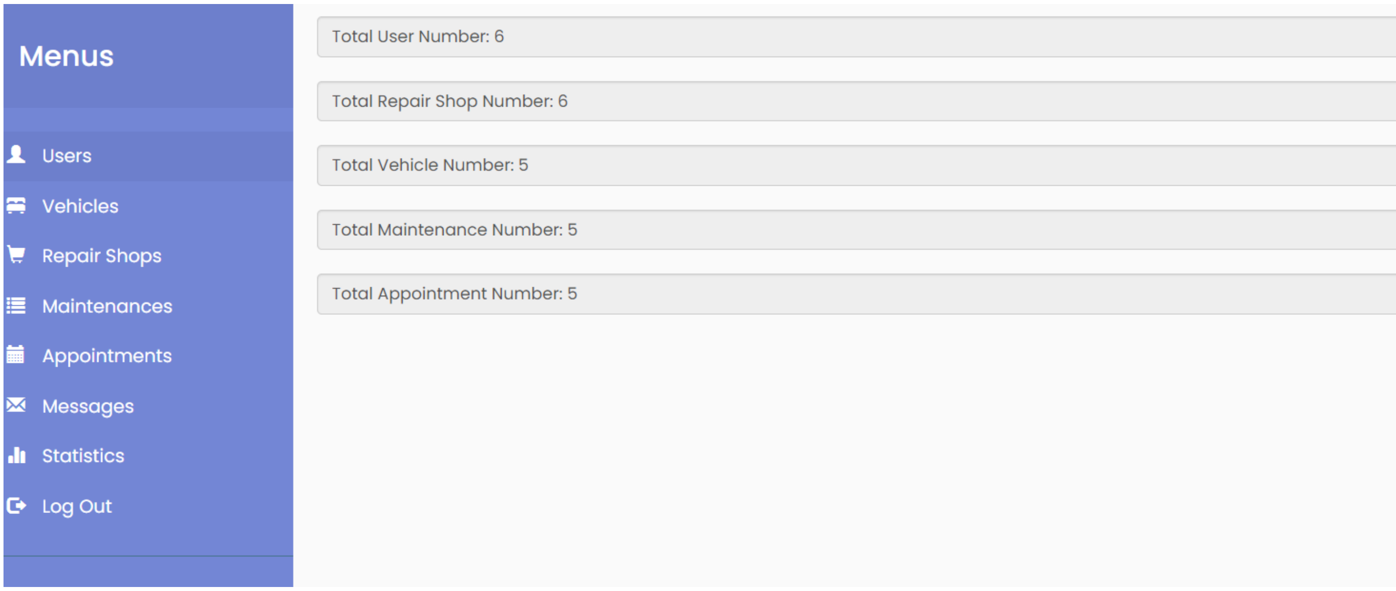
Date
2023-06-13

Receiver Number

Title

Content

Statistics



Total User Number: 6

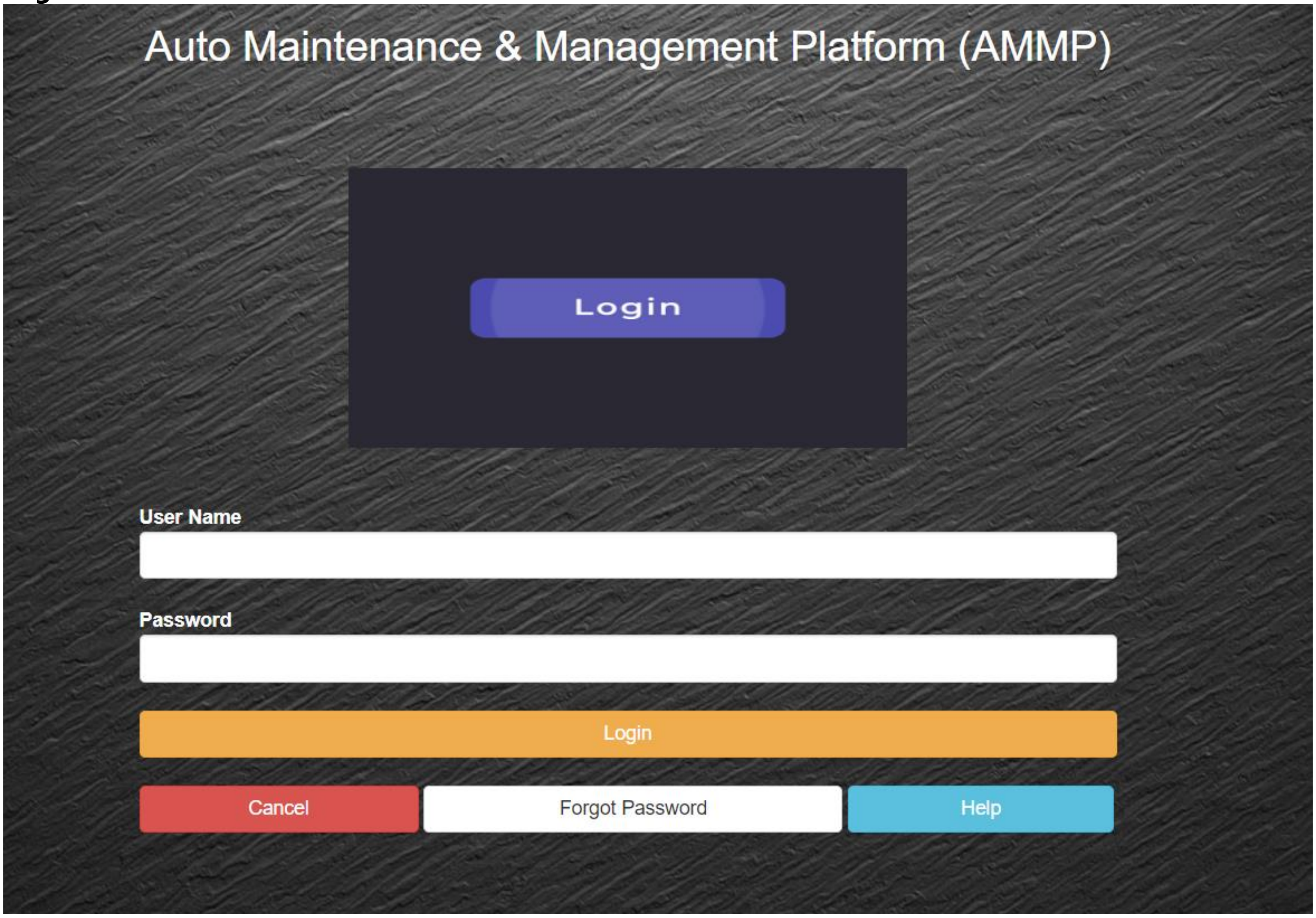
Total Repair Shop Number: 6

Total Vehicle Number: 5

Total Maintenance Number: 5

Total Appointment Number: 5

Login


Auto Maintenance & Management Platform (AMMP)
Login
User Name
Password
Login
Cancel
Forgot Password
Help