VEHICLE RENTAL MANAGEMENT SYSTEM

Tech Stack

1. Frontend

- ***HTML:*** For structuring the main content of the web application.
- ***CSS:*** For styling the interface, layout, and design elements.
- ***JavaScript:*** For handling dynamic interactions, form submissions, and DOM manipulation on the frontend.
- ***AJAX /Fetch API:*** For asynchronous requests to the server for creating bookings, checking vehicle availability, etc.

2. Backend

- ***Node.js:*** The JavaScript runtime environment for building the server-side application.
- ***Express.js:*** A lightweight web application framework for Node.js, used for handling routing, middleware, and HTTP requests.
- ***UUID:*** Used to generate unique IDs for each rental transaction (Rental_ID).

1. Database

- ***MySQL:*** The relational database system used to store data related to rentals, vehicles, renters, and owners.
- ***SQL Queries:*** For data retrieval, insertion, updating, and validation within the database.

2. Libraries & Modules

- ***body-parser:*** Middleware for parsing incoming request bodies in a middleware, making it accessible under req.body.
- ***Express-session:*** For handling sessions, enabling renter or owner-specific functionalities by tracking the user's session.
- ***MySQL Driver (mysql2):*** Used to connect and communicate with the MySQL database from the Node.js application.

3. Development Tools

- ***Postman:*** For testing API endpoints during development.
- ***Git:*** Version control for tracking code changes.
- ***VS Code:*** Text editor for writing and managing the project code.

Tables:

1. User

User (User_ID, User_Name, DOB, Email, Password, Address)

User					
User_ID	User_Name	DOB	Email	Password	Address

```
Create Table User(
User_ID char(40) primary key,
User_Name char(50),
DOB date,
Email char(50),
Pass_word char(20),
Address char(50));
describe User;
```

Field	Type	Null	Key	Default	Extra
User_ID	char(40)	NO	PRI	NULL	
User_Name	char(50)	YES		NULL	
DOB	date	YES		NULL	
Email	char(50)	YES		NULL	
Pass_word	char(20)	YES		NULL	
Address	char(50)	YES		NULL	

2. User_MobileNumber

User_MobileNumber (User_ID, Mobile_Number)

User_Mobile	
User_ID	Mobile_Number

```
Create Table User_MobileNumber(
Mobile_Number char(50),
User_ID char(40),
FOREIGN KEY (User_ID) REFERENCES User(User_ID));
describe User_MobileNumber;
```

Field	Type	Null	Key	Default	Extra
Mobile_Number	char(50)	YES		NULL	
User_ID	char(40)	YES	MUL	NULL	

3. Owner

Owner (User_ID, OwnerID)

Owner	
User_ID	OwnerID

```
Create Table Owners(
User_ID char(40),
Owner_ID char(40) primary key,
FOREIGN KEY (User_ID) REFERENCES User(User_ID));
describe Owners;
```

Field	Type	Null	Key	Default	Extra
User_ID	char(40)	YES	MUL	NULL	
Owner_ID	char(40)	NO	PRI	NULL	

4. Renter

Renter (User_ID, RenterID, DriverLicense)

Renter		
User_ID	RenterID	DriverLicense

```
Create Table Renter(
User_ID char(40),
```

```
Renter_ID char(40) primary key,
DriverLicense char(20),
FOREIGN KEY (User_ID) REFERENCES User(User_ID));
describe Renter;
```

	Field	Type	Null	Key	Default	Extra
▶	User_ID	char(40)	YES	MUL	NULL	
	Renter_ID	char(40)	NO	PRI	NULL	
	DriverLicense	char(20)	YES		NULL	

5. Rental

Rental (RentalID, RenterID, VehicleID, Rental_amount, Start_date, End_date)

Rental					
RentalID	RenterID	VehicleID	Rental_amount	Start_date	End_date

```
Create Table Rental(
Rental_ID char(40),
Renter_ID char(40),
Vehicle_ID char(40),
Rental_Amount int,
Start_date date,
End_date date,
StatusofRental char(20) DEFAULT 'pending',
Foreign key (Renter_ID) REFERENCES Renter(Renter_ID),
Foreign key (Vehicle_ID) REFERENCES Vehicle(Vehicle_ID));
```

	Field	Type	Null	Key	Default	Extra
▶	Rental_ID	char(40)	YES		NULL	
	Renter_ID	char(40)	YES	MUL	NULL	
	Vehicle_ID	char(40)	YES	MUL	NULL	
	Rental_Amount	int	YES		NULL	
	Start_date	date	YES		NULL	
	End_date	date	YES		NULL	
	StatusofRental	char(20)	YES		pending	

6. Vehicle

Vehicle (VehicleID, OwnerID, Seater, AC Type, Fuel, Fastag, Distance, Year, Variant, Brand)

Vehicle									
VehicleID	OwnerID	Seater	AC Type	Fuel	Fastag	Distance	Year	Variant	Brand

```
Create Table Vehicle(
Vehicle_ID char(40) primary key,
Owner_ID char(40),
Seater varchar(10),
AC_Type varchar(20),
Fuel varchar(10),
Fastag enum("Available", "Not Available"),
Distance int,
Yr char(4),
Variant char(10),
rand char(10),
FOREIGN KEY (Owner_ID) REFERENCES Owners(Owner_ID));
```

Field	Type	Null	Key	Default	Extra
Vehicle_ID	char(40)	NO	PRI	NULL	
Owner_ID	char(40)	YES	MUL	NULL	
Seater	varchar(10)	YES		NULL	
AC_Type	varchar(20)	YES		NULL	
Fuel	varchar(10)	YES		NULL	
Fastag	enum('Available','Not Available')	YES		NULL	
Distance	int	YES		NULL	
Yr	char(4)	YES		NULL	
Variant	char(10)	YES		NULL	
rand	char(10)	YES		NULL	

7. Vehicle_Certificates

Vehicle_Certificates (VehicleID, Certificates)

Vehicle_Certificates	
VehicleID	Certificates

```
Create Table Vehicle_Certificates
(Vehicle_ID char(40),
Vehicle_Certificates varchar(255),
Foreign key (Vehicle_ID) REFERENCES Vehicle(Vehicle_ID));
```

Field	Type	Null	Key	Default	Extra
Vehicle_ID	char(40)	YES	MUL	NULL	
Vehicle_Certificates	varchar(255)	YES		NULL	

8. Payment_Transaction

Payment_Transaction (PaymentID, Amount, Date, Method, TransactionID, RenterID, OwnerID)

Payment_Transaction						
PaymentID	Amount	Date	Method	TransactionID	RenterID	OwnerID

```
Create table Payment_Transactions(
Payment_ID char(40) primary key,
Amount int,
Payment_Date date,
Method Enum("Credit","Debit","UPI","Cash"),
Transaction_ID char(40) unique,
Renter_ID char(40),
Owner_ID char(40),
Foreign key (Renter_ID) REFERENCES Renter(Renter_ID),
FOREIGN KEY (Owner_ID) REFERENCES Owners(Owner_ID));
describe Payment_Transactions;
```

Field	Type	Null	Key	Default	Extra
Payment_ID	char(40)	NO	PRI	NULL	
Amount	int	YES		NULL	
Payment_Date	date	YES		NULL	
Method	enum('Credit','Debit','UPI','Cash')	YES		NULL	
Transaction_ID	char(40)	YES	UNI	NULL	
Renter_ID	char(40)	YES	MUL	NULL	
Owner_ID	char(40)	YES	MUL	NULL	

9. Invoice

Invoice (InvoiceID, PaymentID)

Invoice	
InvoiceID	PaymentID

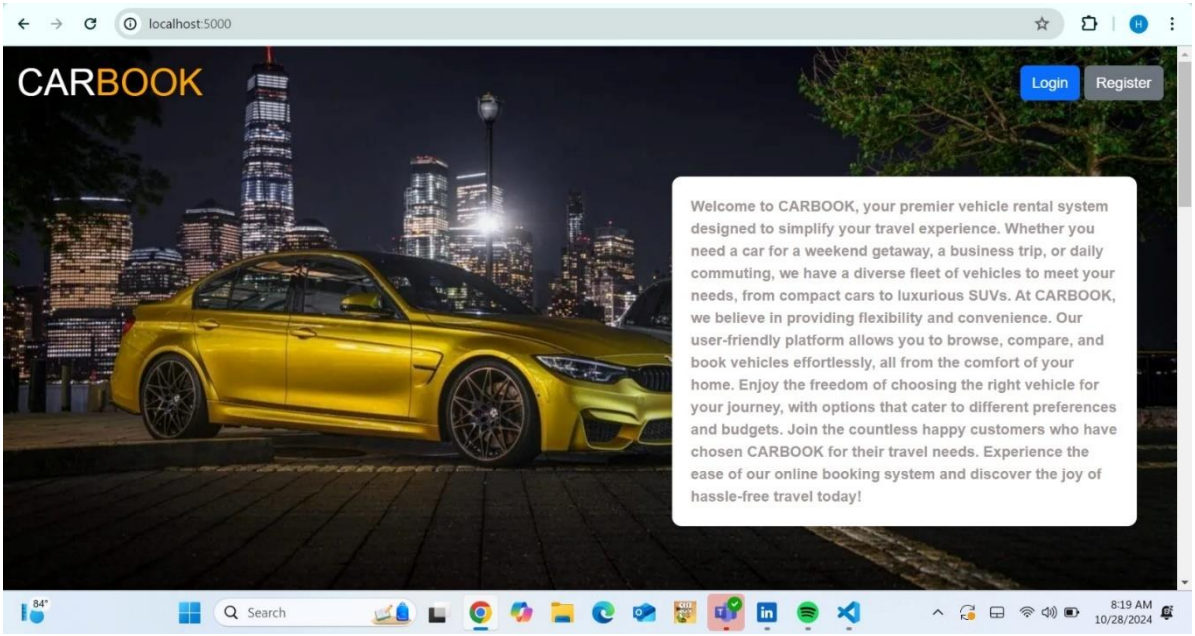
```
Create table Invoice(
Invoice_ID char(40) primary key,
Payment_ID char(40),
```

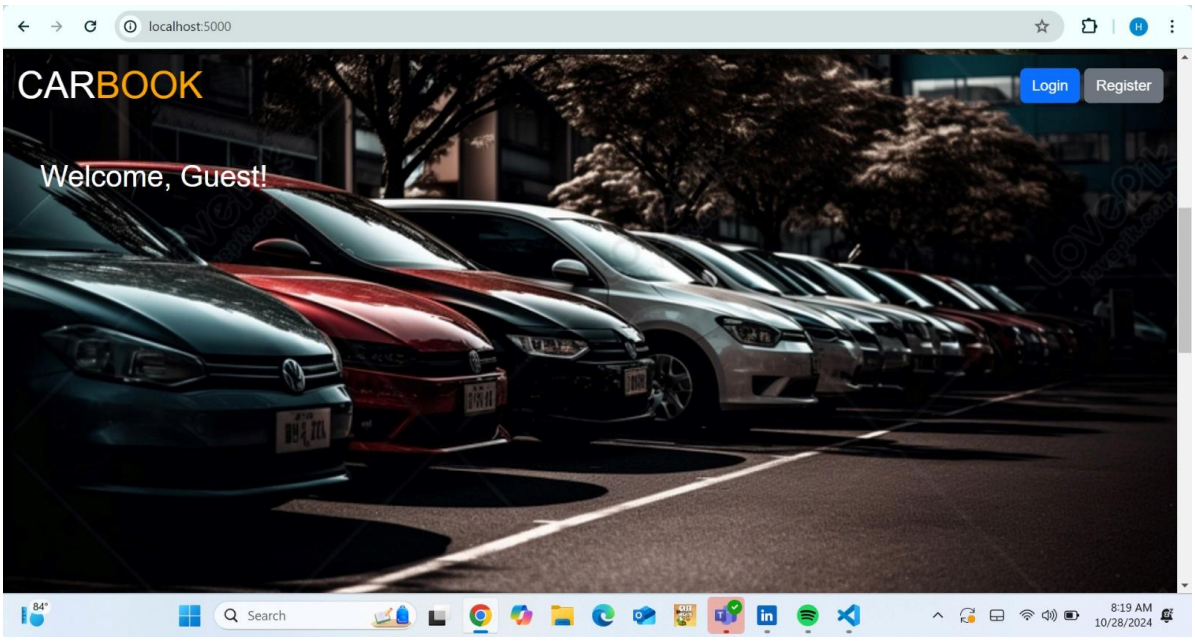
```
Foreign key (Payment_ID) references Payment_Transactions(Payment_ID));
```

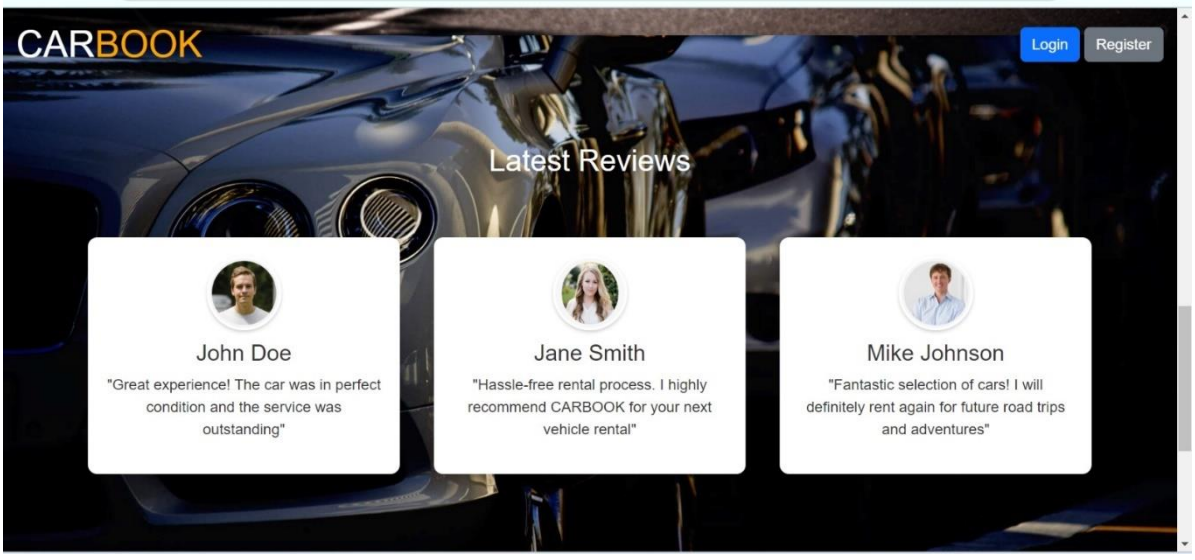
Field	Type	Null	Key	Default	Extra
Invoice_ID	char(40)	NO	PRI	NULL	
Payment_ID	char(40)	YES	MUL	NULL	

Project Frontend Screenshot

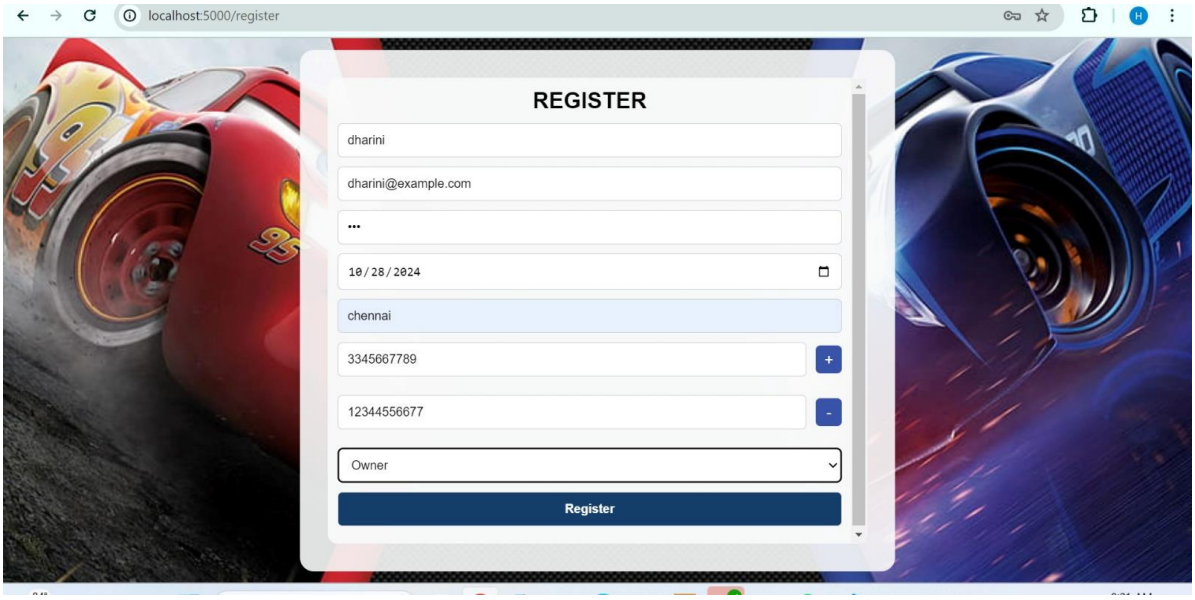
Home Page



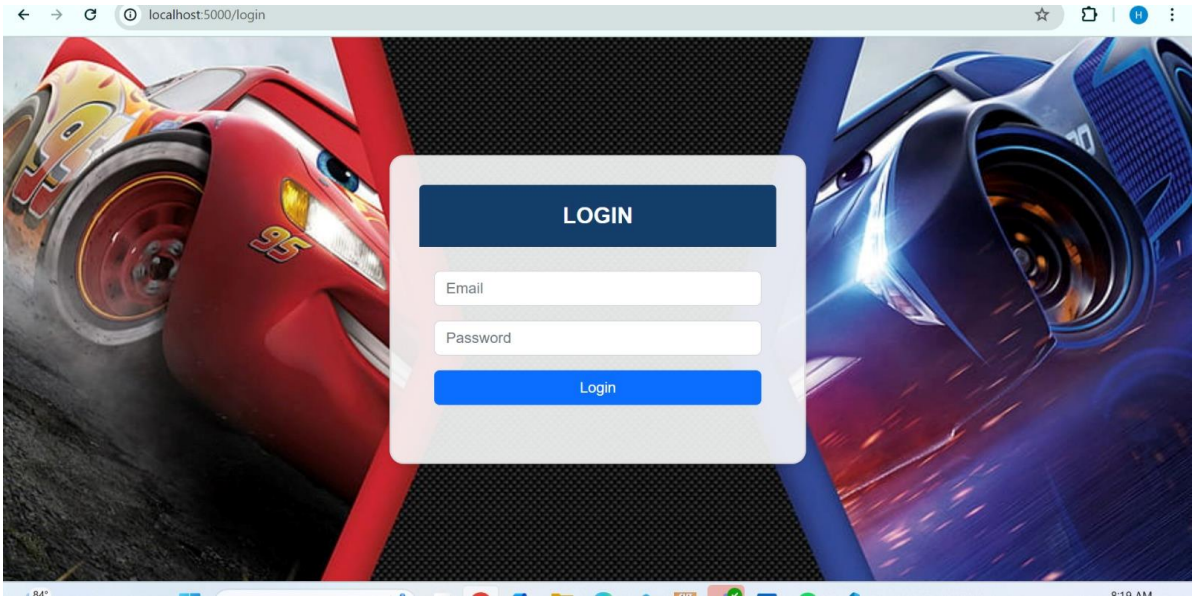

localhost:5000
CARBOOK
Login
Register
Welcome, Guest!
Search
8:19 AM
10/28/2024


CARBOOK
Login
Register
Latest Reviews
John Doe
"Great experience! The car was in perfect condition and the service was outstanding"
Jane Smith
"Hassle-free rental process. I highly recommend CARBOOK for your next vehicle rental"
Mike Johnson
"Fantastic selection of cars! I will definitely rent again for future road trips and adventures"

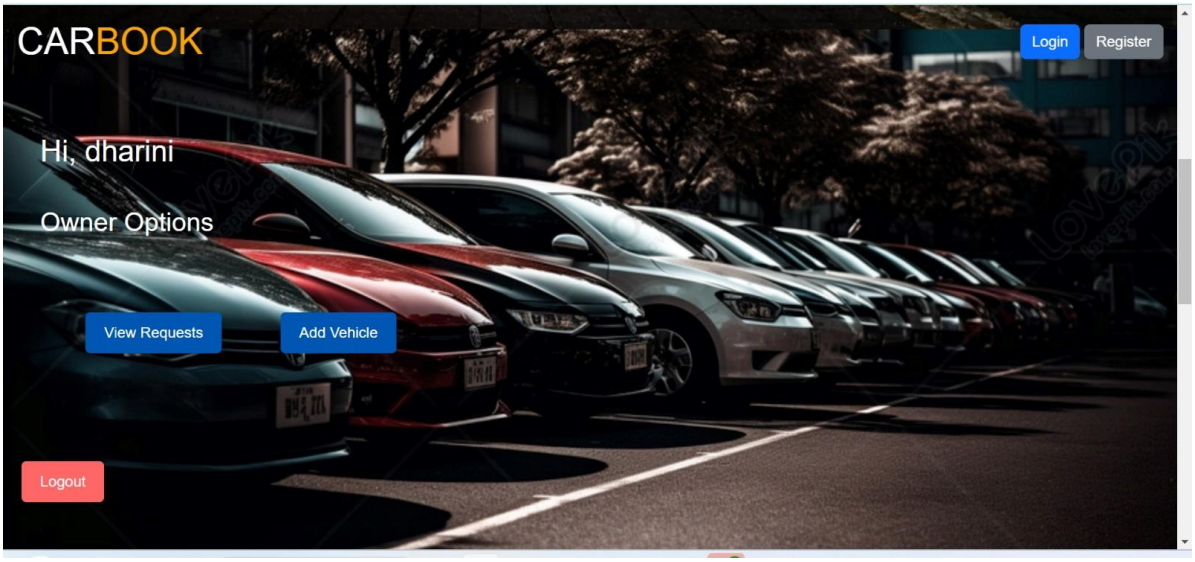
Register Page



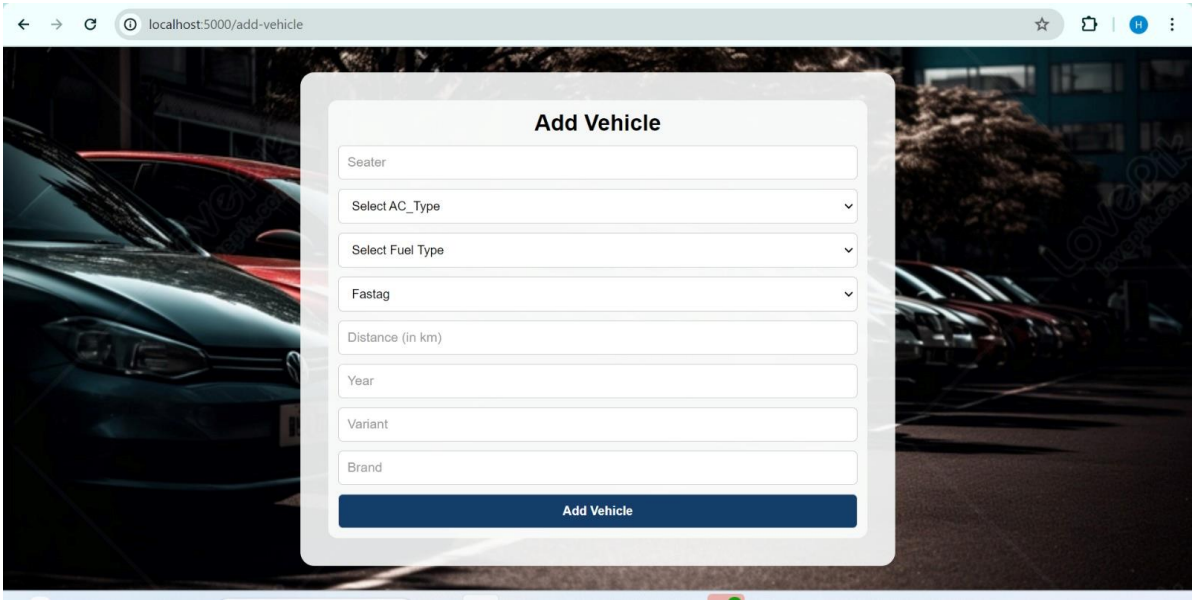
Login Page



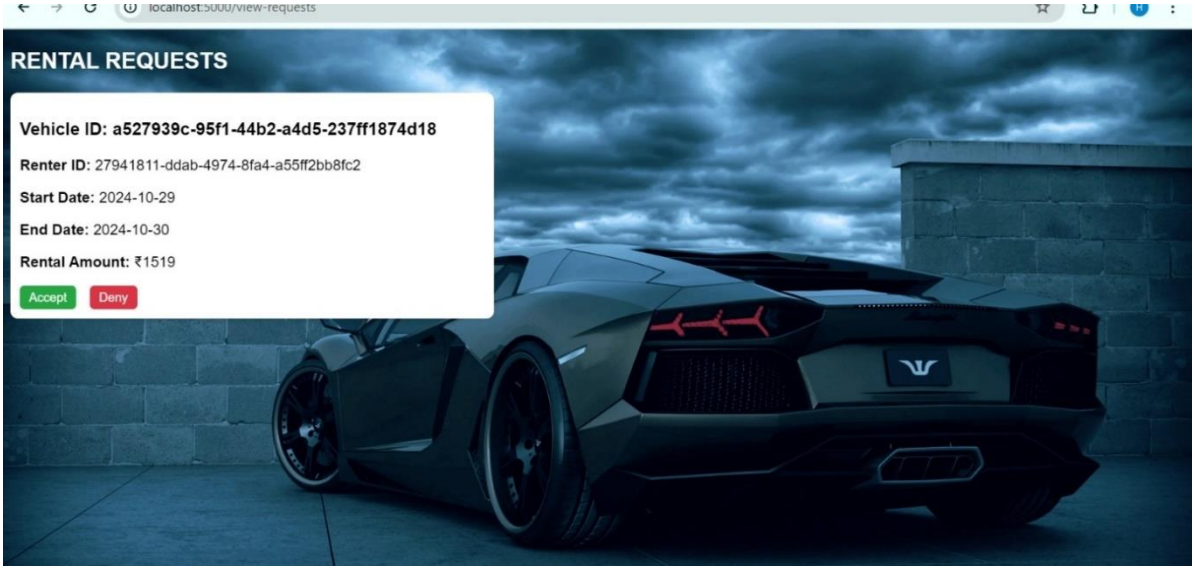
Owner Home Page



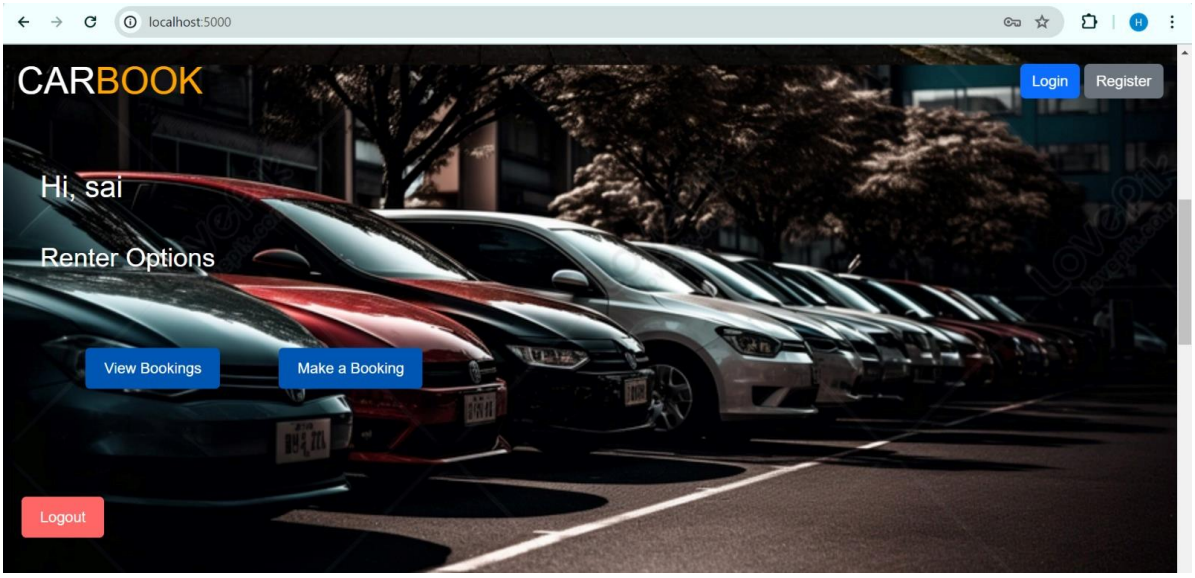
Owner Add Vehicle Page



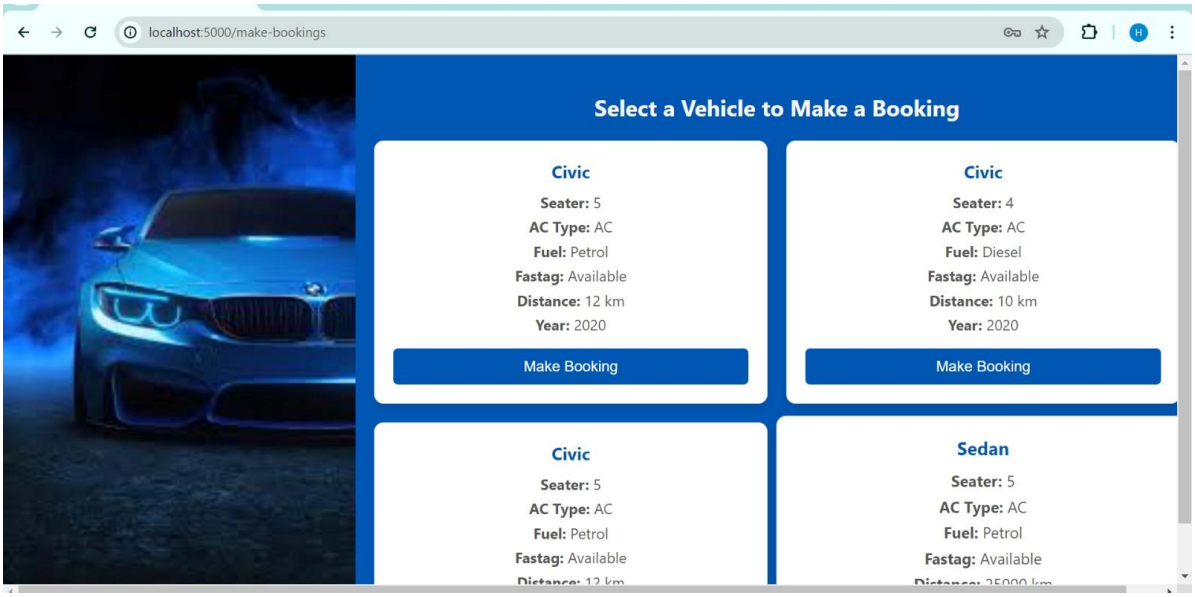
Owner Rental Request Page



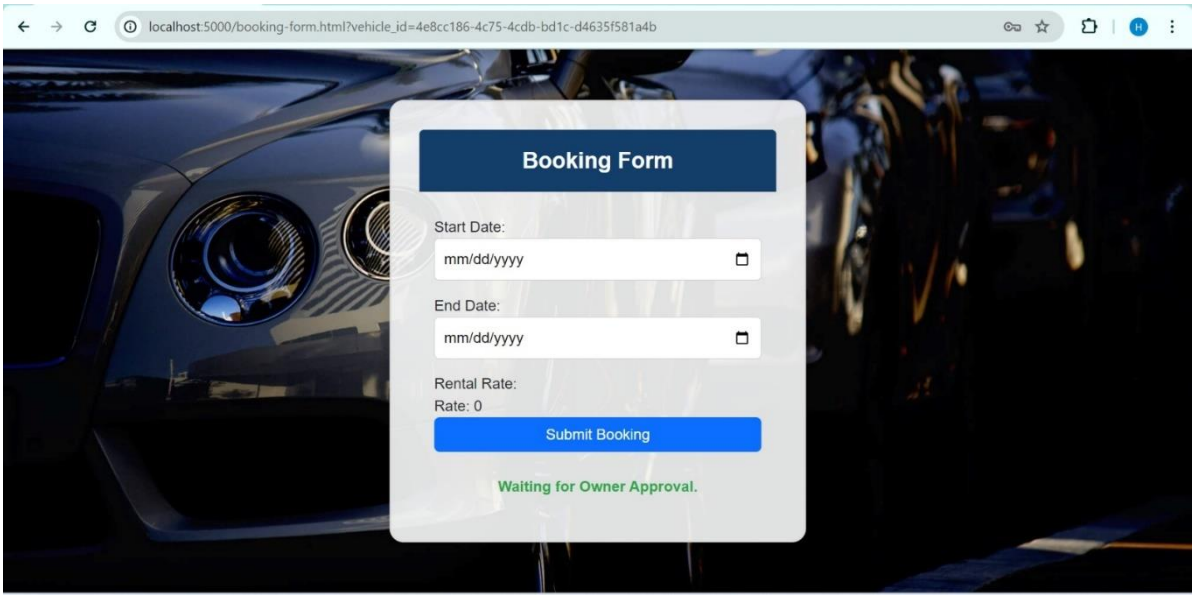
Renter Home Page



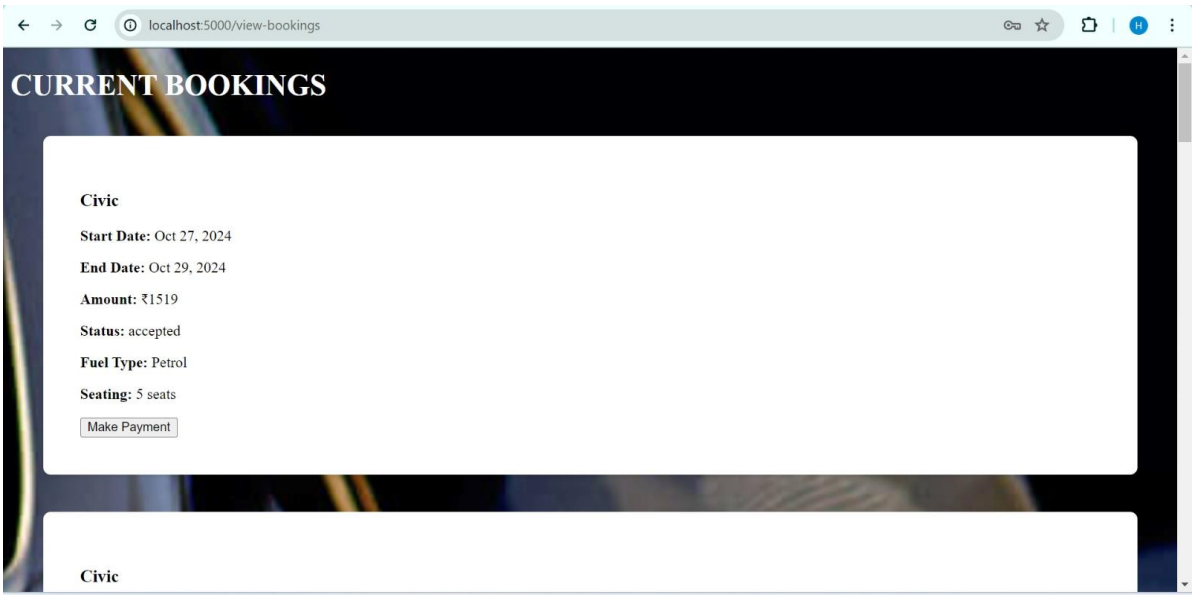
Renter Booking Page



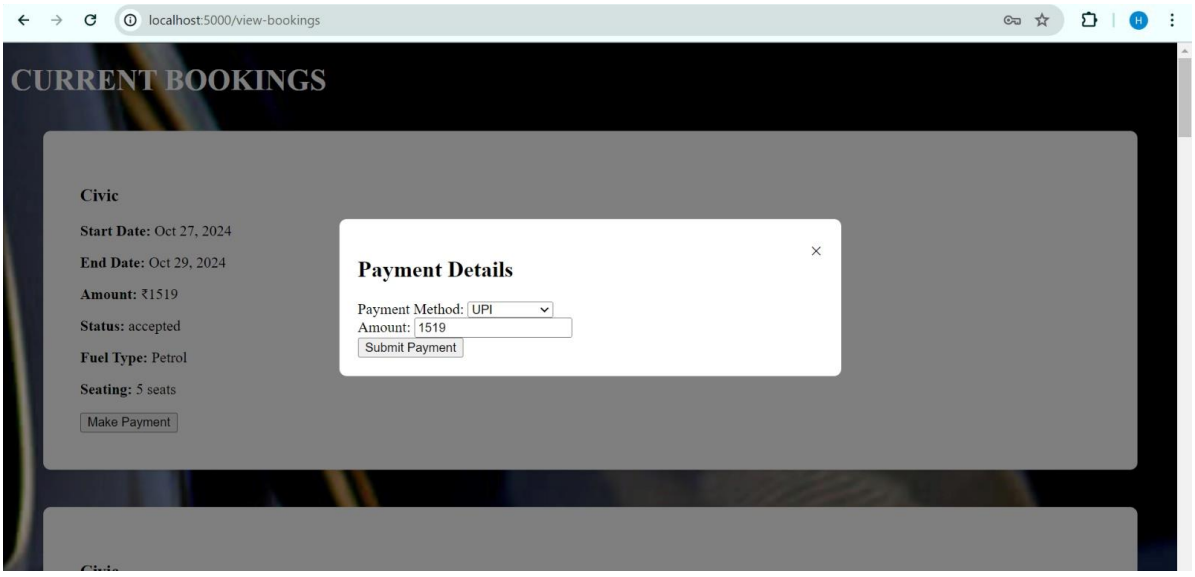
Renter Booking Form Page



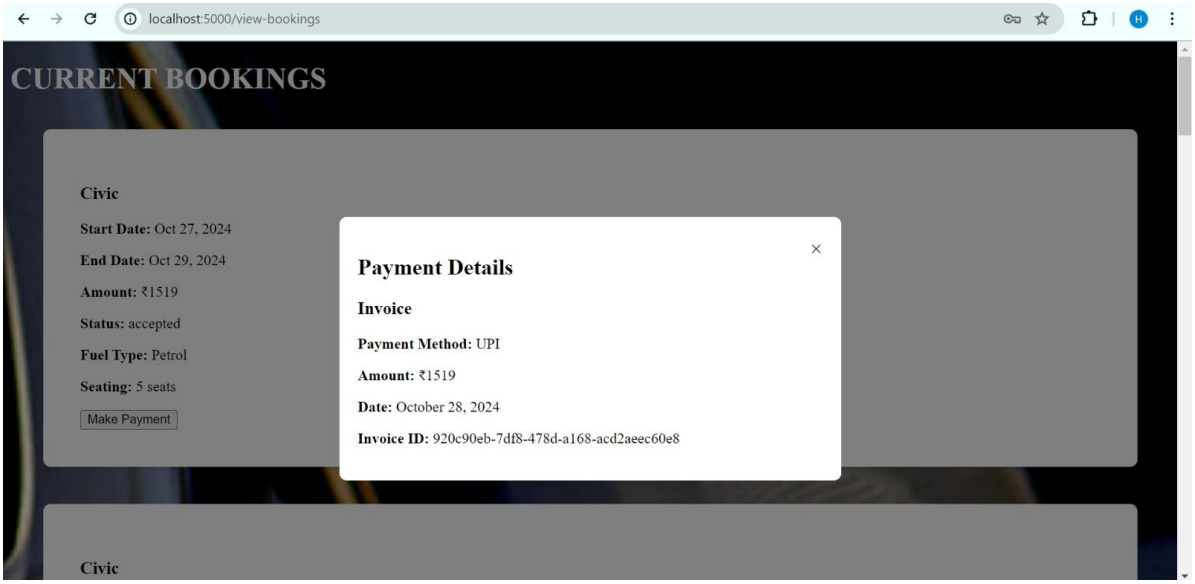
Renter Current Booking Page



Renter Payment Page

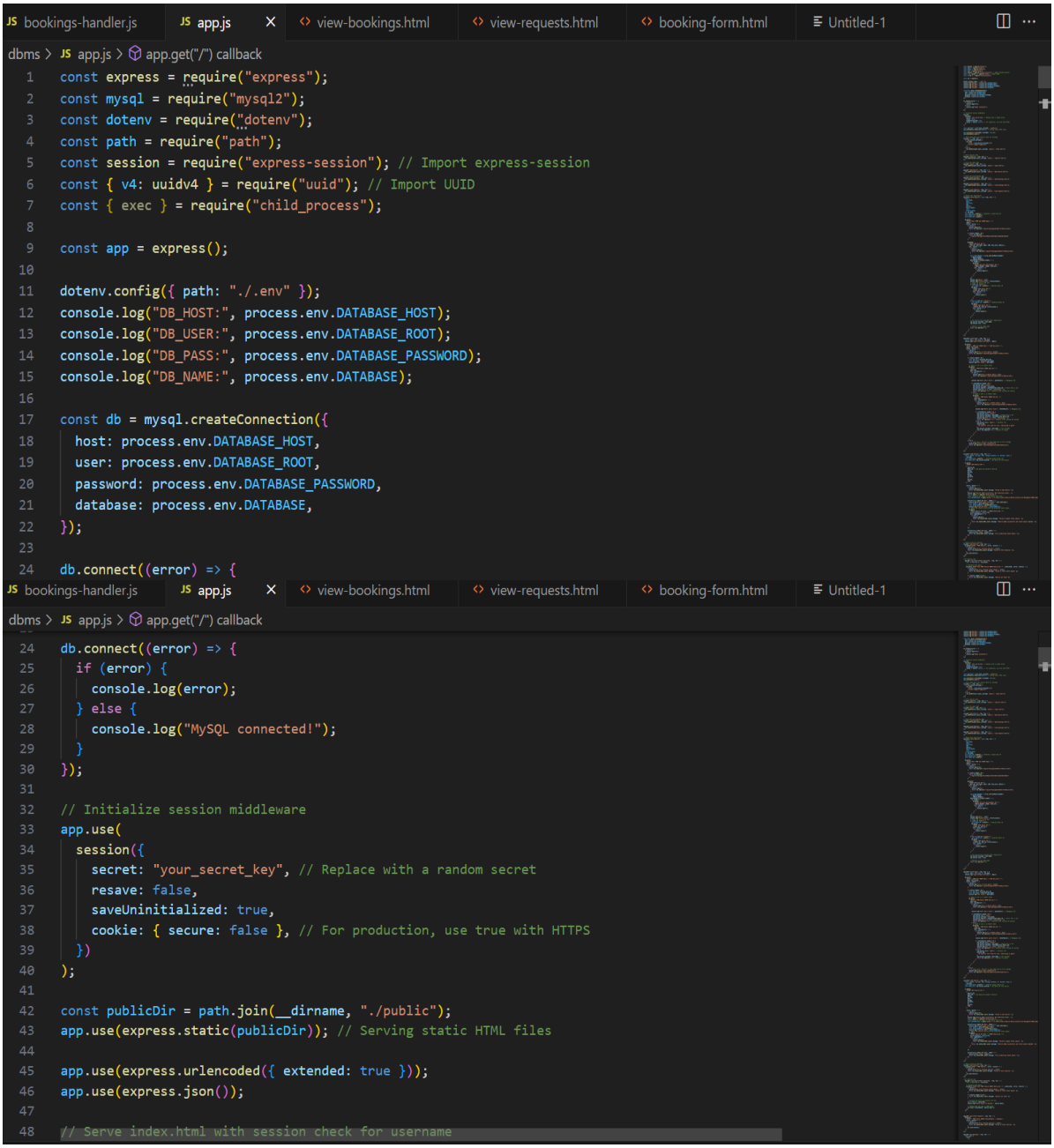


Transaction Invoice Page



BACKEND CODE

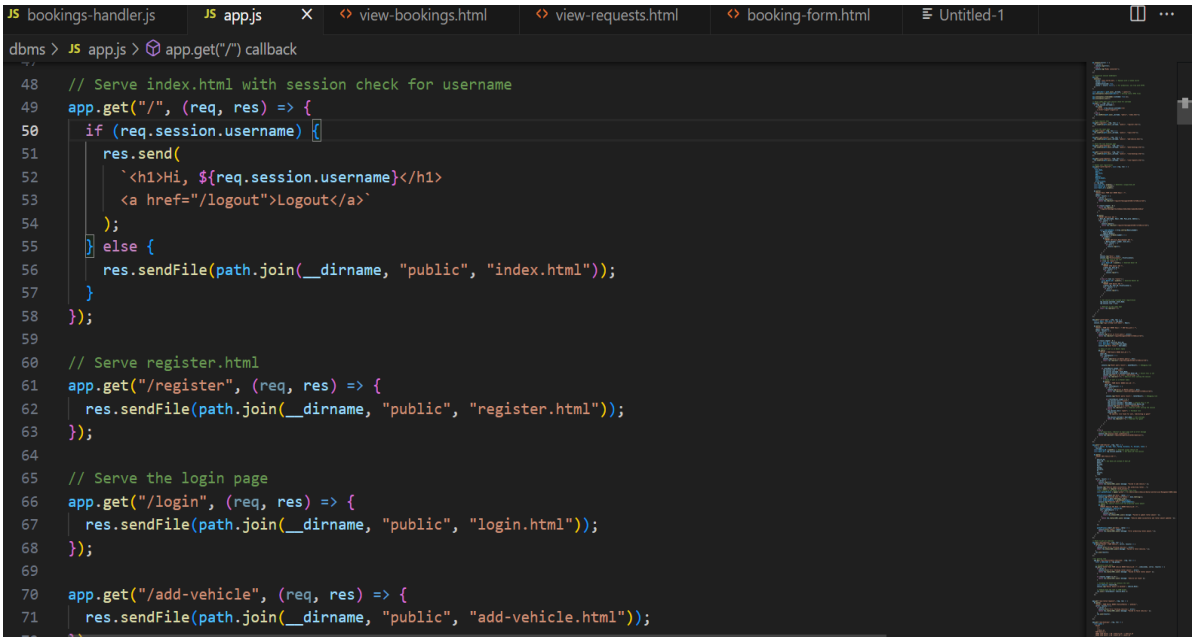
App.js



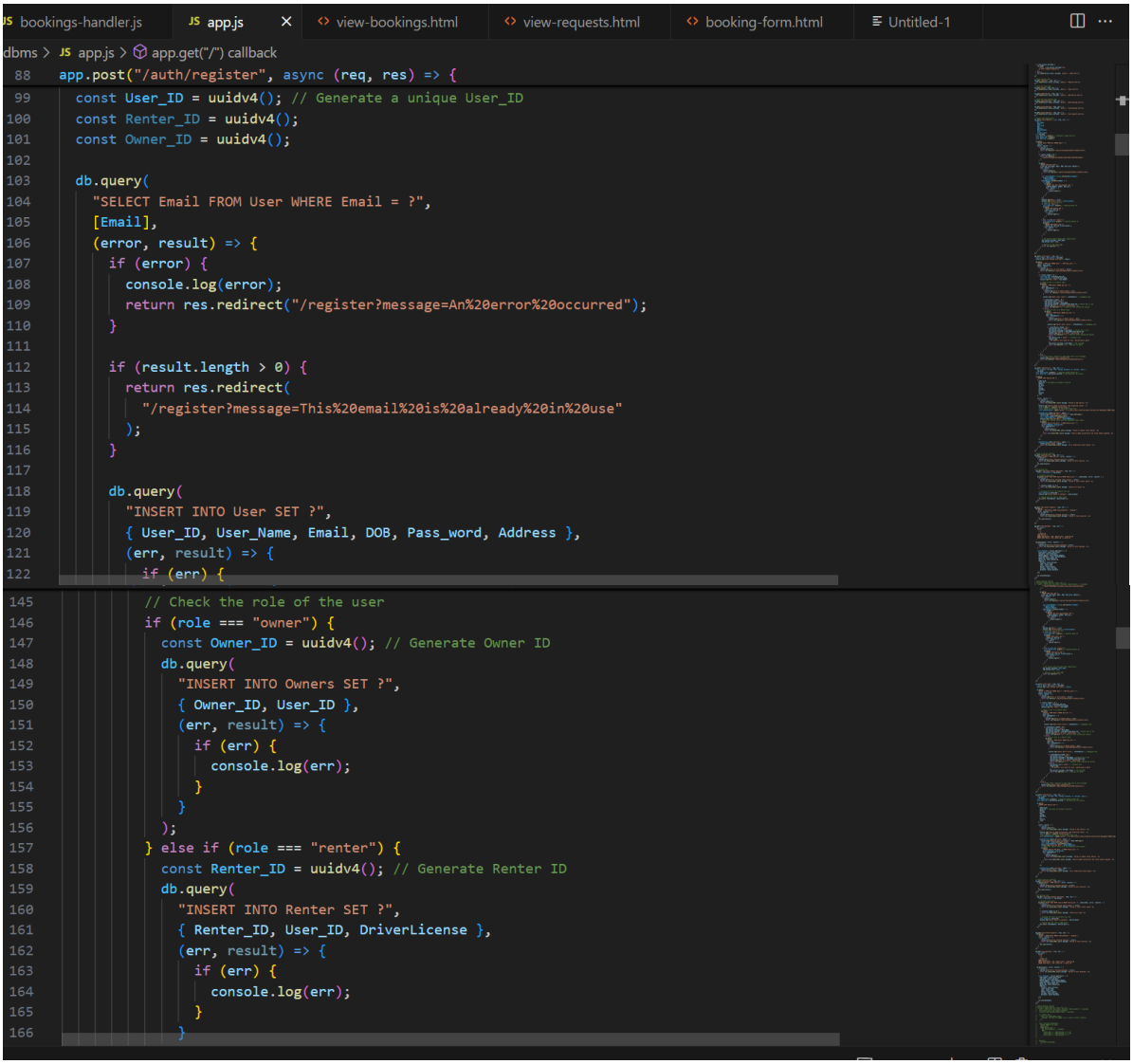
```
bookings-handler.js | app.js | view-bookings.html | view-requests.html | booking-form.html | Untitled-1
dbms > app.js > app.get("/") callback
1  const express = require("express");
2  const mysql = require("mysql2");
3  const dotenv = require("dotenv");
4  const path = require("path");
5  const session = require("express-session"); // Import express-session
6  const { v4: uuidv4 } = require("uuid"); // Import UUID
7  const { exec } = require("child_process");
8
9  const app = express();
10
11 dotenv.config({ path: "./.env" });
12 console.log("DB_HOST:", process.env.DATABASE_HOST);
13 console.log("DB_USER:", process.env.DATABASE_ROOT);
14 console.log("DB_PASS:", process.env.DATABASE_PASSWORD);
15 console.log("DB_NAME:", process.env.DATABASE);
16
17 const db = mysql.createConnection({
18   host: process.env.DATABASE_HOST,
19   user: process.env.DATABASE_ROOT,
20   password: process.env.DATABASE_PASSWORD,
21   database: process.env.DATABASE,
22 });
23
24 db.connect((error) => {
```

bookings-handler.js | app.js | view-bookings.html | view-requests.html | booking-form.html | Untitled-1
dbms > app.js > app.get("/") callback

```
24 db.connect((error) => {
25   if (error) {
26     console.log(error);
27   } else {
28     console.log("MySQL connected!");
29   }
30 });
31
32 // Initialize session middleware
33 app.use(
34   session({
35     secret: "your_secret_key", // Replace with a random secret
36     resave: false,
37     saveUninitialized: true,
38     cookie: { secure: false }, // For production, use true with HTTPS
39   })
40 );
41
42 const publicDir = path.join(__dirname, "./public");
43 app.use(express.static(publicDir)); // Serving static HTML files
44
45 app.use(express.urlencoded({ extended: true }));
46 app.use(express.json());
47
48 // Serve index.html with session check for username
```


bookings-handler.js
app.js
view-bookings.html
view-requests.html
booking-form.html
Untitled-1
dbms > app.js > app.get("/") callback
48 // Serve index.html with session check for username
49 app.get("/", (req, res) => {
50 if (req.session.username) {
51 res.send(
52 `<h1>Hi, ${req.session.username}</h1>
53 Logout`
54);
55 } else {
56 res.sendFile(path.join(__dirname, "public", "index.html"));
57 }
58 });
59
60 // Serve register.html
61 app.get("/register", (req, res) => {
62 res.sendFile(path.join(__dirname, "public", "register.html"));
63 });
64
65 // Serve the login page
66 app.get("/login", (req, res) => {
67 res.sendFile(path.join(__dirname, "public", "login.html"));
68 });
69
70 app.get("/add-vehicle", (req, res) => {
71 res.sendFile(path.join(__dirname, "public", "add-vehicle.html"));


bookings-handler.js
app.js
view-bookings.html
view-requests.html
booking-form.html
Untitled-1
dbms > app.js > app.get("/") callback
74 // Serve the view bookings page
75 app.get("/make-bookings", (req, res) => {
76 res.sendFile(path.join(__dirname, "public", "make-bookings.html"));
77 });
78
79 app.get("/view-bookings", (req, res) => {
80 res.sendFile(path.join(__dirname, "public", "view-bookings.html"));
81 });
82
83 app.get("/view-requests", (req, res) => {
84 res.sendFile(path.join(__dirname, "public", "view-requests.html"));
85 });
86
87 // Handle user registration
88 app.post("/auth/register", async (req, res) => {
89 const {
90 User_Name,
91 Email,
92 Pass_word,
93 DOB,
94 Address,
95 Mobile_Number,
96 role,
97 DriverLicense,
98 } = req.body;


bookings-handler.js
app.js
view-bookings.html
view-requests.html
booking-form.html
Untitled-1
dbms > app.js > app.get("/") callback
app.post("/auth/register", async (req, res) => {
const User_ID = uuidv4(); // Generate a unique User_ID
const Renter_ID = uuidv4();
const Owner_ID = uuidv4();
db.query(
"SELECT Email FROM User WHERE Email = ?",
[Email],
(error, result) => {
if (error) {
console.log(error);
return res.redirect("/register?message=An%20error%20occurred");
}
if (result.length > 0) {
return res.redirect(
"/register?message=This%20email%20is%20already%20in%20use"
);
}
db.query(
"INSERT INTO User SET ?",
{ User_ID, User_Name, Email, DOB, Pass_word, Address },
(err, result) => {
if (err) {
// Check the role of the user
if (role === "owner") {
const Owner_ID = uuidv4(); // Generate Owner ID
db.query(
"INSERT INTO Owners SET ?",
{ Owner_ID, User_ID },
(err, result) => {
if (err) {
console.log(err);
}
}
);
} else if (role === "renter") {
const Renter_ID = uuidv4(); // Generate Renter ID
db.query(
"INSERT INTO Renter SET ?",
{ Renter_ID, User_ID, DriverLicense },
(err, result) => {
if (err) {
console.log(err);
}
}

```
            );
          }

          // Set session with username after registration
          req.session.username = User_Name;
          req.session.role = role;

          // Redirect to the index page
          return res.redirect("/");
        }
      );
    }
  );
});

app.post("/auth/login", (req, res) => {
  const { Email, Pass_word } = req.body;
  console.log("Login attempt with email:", Email);

  db.query(
    "SELECT * FROM User WHERE Email = ? AND Pass_word = ?",
    [Email, Pass_word],
    (error, result) => {
      if (error) {
        console.log("Error in first query:", error);
        return res.redirect("/login?message=An%20error%20occurred");
      }

      if (result.length > 0) {
        const User_ID = result[0].User_ID;
        const User_Name = result[0].User_Name;
        console.log("User found:", User_Name);

        // Check if user is in Owners table
        db.query(
          "SELECT * FROM Owners WHERE User_ID = ?",
          [User_ID],
          (err, ownerResult) => {
            if (err) {
              console.log("Error in Owners query:", err);
              return res.redirect("/login?message=An%20error%20occurred");
            }

            console.log("Owner query result:", ownerResult); // Debugging line

            if (ownerResult.length > 0) {
              req.session.role = "owner";
              req.session.username = User_Name;
              req.session.ownerId = ownerResult[0].Owner_ID; // Ensure this is set
              console.log("User is an owner, redirecting...");
              return res.redirect("/"); // Redirect after setting the session
            } else {
              // Check if user is in Renter table
              db.query(
                "SELECT * FROM Renter WHERE User_ID = ?",
                [User_ID],
                (err, renterResult) => {
                  if (err) {
                    console.log("Error in Renter query:", err);
                    return res.redirect("/login?message=An%20error%20occurred");
                  }

                  console.log("Renter query result:", renterResult); // Debugging line

                  if (renterResult.length > 0) {
                    req.session.role = "renter";
                    req.session.username = User_Name; // Ensure this is set
                    req.session.renterId = renterResult[0].Renter_ID;
```

```
                    console.log("User is a renter, redirecting...");
                    return res.redirect("/"); // Redirect after setting the session
                  } else {
                    req.session.role = "guest"; // Fallback role
                    console.log(
                      "No specific role found for user, redirecting as guest"
                    );
                    req.session.username = User_Name; // Set username
                    return res.redirect("/"); // Redirect for guest
                  }
                }
              );
            }
          }
        );
      } else {
        // If login fails, redirect to login page with an error message
        console.log("Invalid login credentials");
        return res.redirect("/login?message=Invalid%20credentials");
      }
    }
  );
});

app.post("/add-vehicle", (req, res) => {
  const { Seater, AC_Type, Fuel, Fastag, Distance, Yr, Variant, rand } =
    req.body;
  const Vehicle_ID = uuidv4(); // Generate unique vehicle ID
  const Owner_ID = req.session.ownerId; // Get Owner_ID from session

  db.query(
    "INSERT INTO Vehicle SET ?",
    {
      Vehicle_ID,
      Owner_ID, // Use Owner_ID instead of User_ID
      Seater,
      AC_Type,
      Fuel,
      Fastag,
      Distance,
      Yr,
      Variant,
      rand,
    },

    (error, result) => {
      if (error) {
        console.log(error);
        return res.status(500).json({ message: "Failed to add vehicle." });
      }
      console.log("Vehicle added successfully. Now predicting rental...");
      const { spawn } = require('child_process');
      //this should be the addess of the python train file.
      const pythonProcess = spawn('python', ['C:\\Users\\dell\\Vehicle-Rental-and-Services-Management-DBMS\\dbms\\tra

      pythonProcess.stdout.on('data', (data) => {
        console.log("Received data from Python:", data.toString());
        const output = data.toString().trim();
        const predictedRental = parseFloat(output);
        console.log("Predicted Rate:", predictedRental);
        // Update the vehicle record with the predicted rental amount
        db.query(
          "UPDATE Vehicle SET Rate = ? WHERE Vehicle_ID = ?",
          [predictedRental, Vehicle_ID],
          (err, updateResult) => {
            if (err) {
              console.log(err);
              return res.status(500).json({ message: "Failed to update rental amount." });
            }
```

```
                return res.status(200).json({ message: "Vehicle added successfully and rental amount updated." });
              }
            );

          });

          pythonProcess.stderr.on('data', (data) => {
            console.error(`stderr: ${data}`);
            return res.status(500).json({ message: "Error predicting rental amount." });
          });
        }
      );
    });

    // Route to get all vehicles
    app.get("/api/vehicles", (req, res) => {
      db.query("SELECT * FROM Vehicle", (error, results) => {
        if (error) {
          console.error("Error fetching vehicles:", error);
          return res.status(500).json({ message: "Failed to fetch vehicles." });
        }
        res.json(results);
      });
    });

    //for getting rate
      app.get('/api/vehicle-Rate/:vehicleId', (req, res) => {
        const { vehicleId } = req.params;

        // Callback style query
        db.query('SELECT Rate FROM vehicle WHERE Vehicle_ID = ?', [vehicleId], (error, results) => {
          if (error) {
            console.error('Error fetching rental amount:', error);
            return res.status(500).json({ message: 'Failed to fetch rental amount' });
          }

          if (results.length === 0) {
            return res.status(404).json({ message: 'Vehicle not found' });
          }

          // Assuming the first row contains the rate
          const vehicle = results[0];
          console.log("Rental Amount in backend:", vehicle.Rate);

          res.json({ rentalAmount: vehicle.Rate });
        });
      });



    app.get("/api/rental-requests", (req, res) => {
      db.query(
        "SELECT * FROM Rental WHERE StatusofRental = 'pending'",
        (error, results) => {
          if (error) {
            console.error("Error fetching vehicles:", error);
            return res.status(500).json({ message: "Failed to fetch vehicles." });
          }
          res.json(results);
        }
      );
    });

    app.get("/api/bookings", (req, res) => {
      const query = `
        SELECT
          v.*,
```

```
const query = `
  SELECT
    v.*,
    r.*,
    o.Owner_ID
  FROM Vehicle v
  INNER JOIN Rental r ON v.Vehicle_ID = r.Vehicle_ID
  INNER JOIN Owners o ON v.Owner_ID = o.Owner_ID
`;

db.query(query, (error, results) => {
  if (error) {
    console.error("Error fetching bookings:", error);
    return res.status(500).json({ message: "Failed to fetch bookings." });
  }

  const bookings = results.map(result => ({
    Start_date: result.Start_date,
    End_date: result.End_date,
    Rental_Amount: result.Rental_Amount,
    StatusofRental: result.StatusofRental,
    Owner_ID: result.Owner_ID,
    Renter_ID: result.Renter_ID,
    Vehicle: {
            Variant: result.Variant,
            rand: result.rand,
            Fuel: result.Fuel,
            Seater: result.Seater,
            AC_Type: result.AC_Type,
            Distance: result.Distance
          }
        }));

        res.json(bookings);
      });
    });
    app.post("/book-vehicle", (req, res) => {
      const { Vehicle_ID, Start_date, End_date } = req.body;
      const Renter_ID = req.session.renterId;

      if (!Renter_ID) {
        return res.status(401).json({
          message: "You must be logged in as a renter to book a vehicle.",
        });
      }
      const availabilityCheckSql = `
        SELECT COUNT(*) AS count
        FROM Rental
        WHERE Vehicle_ID = ?
          AND StatusofRental = 'accepted'
          AND (
            (Start_date <= ? AND End_date >= ?) OR
            (Start_date <= ? AND End_date >= ?) OR
            (Start_date >= ? AND End_date <= ?)
          )
      `;

      db.query(
        availabilityCheckSql,
        [
          Vehicle_ID,
          Start_date,
          End_date,
          Start_date,
          End_date,
          Start_date,
          End_date,
        ],
        (error, results) => {
```

```
      if (error) {
        console.log("Error checking availability:", error);
        return res
          .status(500)
          .json({ message: "Error checking vehicle availability." });
      }

      const { count } = results[0];
      if (count > 0) {
        // Vehicle is already booked
        return res
          .status(400)
          .json({ message: "Vehicle not available during these dates." });
      }

      // Fetch the Rate from the Vehicle table
      const rateQuery = `SELECT Rate FROM Vehicle WHERE Vehicle_ID = ?`;
      db.query(rateQuery, [Vehicle_ID], (error, rateResult) => {
        if (error) {
          console.log("Error fetching rate:", error);
          return res
            .status(500)
            .json({ message: "Failed to retrieve vehicle rate." });
        }

        const Rental_Amount = rateResult[0].Rate; // Use the rate as Rental_Amount
        const Rental_ID = uuidv4(); // Generate unique rental ID

        // Insert rental record with the fetched rate as Rental_Amount
        db.query(
          "INSERT INTO Rental SET ?",
          {
            Rental_ID,
            Renter_ID,
            Vehicle_ID,
            Rental_Amount,
            Start_date,
            End_date,
          },
          (error, result) => {
            if (error) {
              console.log("Error inserting data:", error);
              return res.status(500).json({ message: "Failed to book vehicle." });
            }
            console.log("Data inserted successfully");
            return res
              .status(200)
              .json({ message: "Waiting for Owner Approval." });
          }
        );
      });
    }
  );
});


app.post("/api/update-rental-status/:rentalId", (req, res) => {
  const { rentalId } = req.params; // Use rentalId from the route parameter
  const { StatusofRental } = req.body;

  console.log(`Updating rental ${rentalId} with status: ${StatusofRental}`); // Log for debugging

  // SQL Query to update status
  const query = `UPDATE Rental SET StatusofRental = ? WHERE Rental_ID = ?`;

  // Check if rentalId is valid and not undefined/null
```

```
  if (!rentalId) {
    return res.status(400).send("Rental ID is required");
  }

  // Execute the query
  db.query(query, [StatusofRental, rentalId], function (err, result) {
    if (err) {
      console.error("Error updating rental status:", err); // Log the error
      return res.status(500).send("Error updating rental status");
    }

    // Check if any rows were affected/updated
    if (result.affectedRows === 0) {
      return res.status(404).send("Rental ID not found");
    }

    res.status(200).send("Status updated successfully");
  });
});

// Check session status to display username
app.get("/session-status", (req, res) => {
  if (req.session.username) {
    res.json({
      loggedIn: true,
      username: req.session.username,
      role: req.session.role,
    });
  } else {
    res.json({ loggedIn: false });
  }
});
app.post("/submit-payment", (req, res) => {
  const { amount, renterId, ownerId, method } = req.body;
  const paymentId = uuidv4(); // Generate unique Payment ID
  const invoiceId = uuidv4(); // Generate unique Invoice ID
  const paymentDate = new Date(); // Current date for payment date

  // Validate payment method
  const validMethods = ['Credit Card', 'Debit Card', 'UPI', 'Cash'];
  if (!validMethods.includes(method)) {
      return res.status(400).json({ message: "Invalid payment method" });
  }

  // Start a transaction to ensure both payment and invoice are created
  db.beginTransaction(function(err) {
      if (err) {
          return res.status(500).json({ message: "Transaction start failed" });
      }

      // Insert payment transaction
      db.query(
          "INSERT INTO Payment_Transactions SET ?",
          {
              Payment_ID: paymentId,
              Amount: amount,
              Payment_Date: paymentDate,
              Method: method,
              Renter_ID: renterId,
              Owner_ID: ownerId,
          },
          (error, result) => {
              if (error) {
                  return db.rollback(() => {
                      res.status(500).json({ message: "Payment insertion failed" });
                  });
              }

              // Insert invoice
              db.query(
                  "INSERT INTO Invoice SET ?",
```

```
                    {
                        Invoice_ID: invoiceId,
                        Payment_ID: paymentId,
                    },
                    (error, result) => {
                        if (error) {
                            return db.rollback(() => {
                                res.status(500).json({ message: "Invoice creation failed" });
                            });
                        }

                        // Commit the transaction
                        db.commit((err) => {
                            if (err) {
                                return db.rollback(() => {
                                    res.status(500).json({ message: "Transaction commit failed" });
                                });
                            }
                            res.status(200).json({
                                message: "Payment processed successfully",
                                invoiceId,
                                paymentId
                            });
                        });
                    });
                }
            );
        }
    );
  });
});
// Handle user logout
app.get("/logout", (req, res) => {
  req.session.destroy((err) => {
    if (err) {
      console.log(err);
      return res.redirect("/?message=Logout%20failed");
    }
    res.redirect("/login?message=Logged%20out%20successfully");
  });
});

app.listen(5000, () => {
```