VEHICLE RENTAL SYSTEM

A MINI PROJECT REPORT

Submitted by

KISHORE KAARTHIK S	**220701135**
MITESH KUMAR M	**220701164**

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM
CHENNAI-602105

2024-2025

BONAFIDE CERTIFICATE

Certified that this project report "**VEHICLE RENTAL SYSTEM**" is the bonafide work of " **KISHORE KAARTHIK S (220701135), MITESH KUMAR M(220701164)** " who carried out the project under my supervision.

Submitted for the Practical Examination held on __________________

SIGNATURE	**SIGNATURE**
Dr.R.SABITHA	**Ms.D.KALPANA**
Professor and II Year Academic Head	**Assistant Professor (SG),**
Computer Science and Engineering,	**Computer Science and Engineering,**
Rajalakshmi Engineering College	**Rajalakshmi Engineering College**
(Autonomous),	**(Autonomous),**
Thandalam, Chennai - 602 105	**Thandalam, Chennai - 602 105**

INTERNAL EXAMINER	**EXTERNAL EXAMINER**

ABSTRACT

The Vehicle Rental Management System is a robust and user-friendly application designed to streamline and automate the operations of vehicle rental services. Built with Python, MongoDB, and Streamlit, this system efficiently manages key rental operations, including vehicle management, customer registration, rental processing, and vehicle returns.

The application features an intuitive graphical user interface, developed using Streamlit, which provides seamless navigation across different functionalities. MongoDB serves as the backend database, ensuring secure and efficient data management. Users can register, log in, view available vehicles, rent, and return vehicles, while administrators can manage vehicle details and monitor rental statuses. This comprehensive system enhances operational efficiency, ensuring a smooth experience for both administrators and clients.

TABLE OF CONTENTS

CHAPTER 1

1.1 INTRODUCTION

This Streamlit-based application facilitates vehicle rental management with a MongoDB backend, providing users with a user-friendly interface to interact with the system. Through various functionalities like signing up, logging in, and browsing available vehicles, users can seamlessly rent and return vehicles. Admins have access to additional features like adding, updating, and removing vehicles, ensuring efficient management of the vehicle inventory. This comprehensive system enhances the rental experience for both clients and administrators alike.

1.2 OBJECTIVES

The vehicle rental system, built with Streamlit and MongoDB, facilitates user registration, authentication, and vehicle management. Clients can conveniently browse available vehicles, rent them, and return them as needed, while administrators oversee vehicle details and rental statuses. Real-time updates on vehicle availability enhance the user experience, ensuring efficient operations. With robust error handling and data validation mechanisms, the system guarantees reliability and security throughout the rental process. Overall, the system offers a seamless and intuitive platform for hassle-free vehicle rentals.

1.3 MODULES

- Vehicle Management Module.
- Rental Management Module.
- User Management Module.
- Admin Management Module
- Database Management Module.

CHAPTER-2

2.1 SOFTWARE DESCRIPTION

Visual studio Code:

A versatile and customizable source code editor with built-in debugging and Git integration, ideal for web and software development projects.

2.2 LANGUAGES

1. Python:

- A high-level programming language known for its simplicity and readability, widely used for web development, data analysis, artificial intelligence, and automation tasks.

2. Streamlit:

- A powerful Python library for creating interactive web applications with simple Python scripts, enabling rapid prototyping and deployment of data-driven applications with ease..

3. MongoDB:

- A scalable NoSQL database solution, offering high performance, flexibility, and seamless integration with modern applications..

CHAPTER-3

REQUIREMENT AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION:

Functional Requirements:

1. **User Authentication**

   Users should be able to sign up and log in securely to access the system functionalities.

2. **Admin Management**

   Administrators should have privileges to add, update, and remove vehicles from the system.

3. **Client Interface**

   Clients should be able to view available vehicles, rent vehicles, and return rented vehicles.

4. **Vehicle Management**

   The system must track vehicle availability, rental status, and rental prices.

5. **Checkout Process**

   Clients should be able to confirm vehicle rentals, providing necessary details for the rental transaction.

Non-Functional Requirements:

1. **Security**

User passwords must be securely hashed for storage, and only authenticated users should access sensitive functionalities.

2. **Performance**

The system should efficiently handle database operations, ensuring quick response times for user interactions.

3. **Scalability**

The system should be designed to accommodate a growing number of users, vehicles, and rental transactions over time.

4. **User Experience**

The user interface should be intuitive and user-friendly, providing clear navigation and informative feedback.

5. **Reliability**

The system must ensure data integrity and consistency, preventing issues like double bookings or data loss.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS:

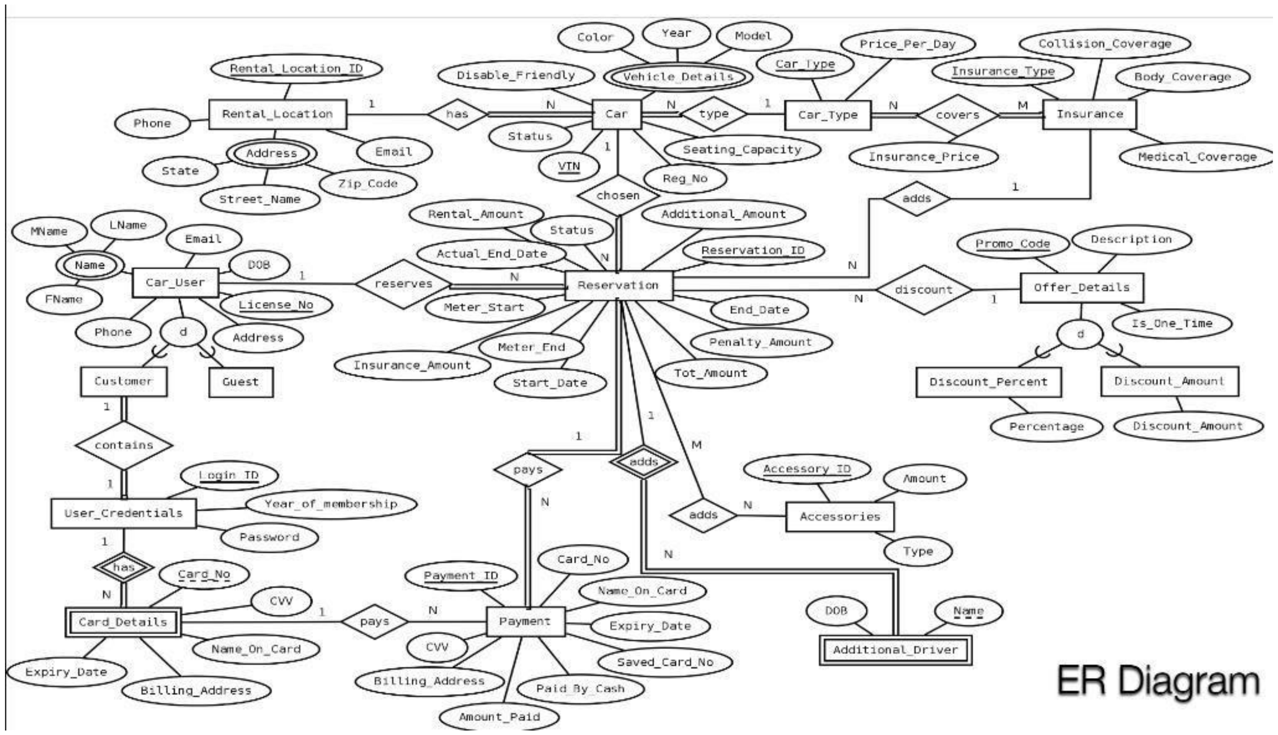
Hardware Requirements

- Processor: 1 GHz or faster processor
- RAM: 2 GB or more
- Storage: At least 500 MB of available disk space
- Display: Minimum resolution of 1024x768
- Input Devices: Keyboard and mouse
- Network: Stable internet connection

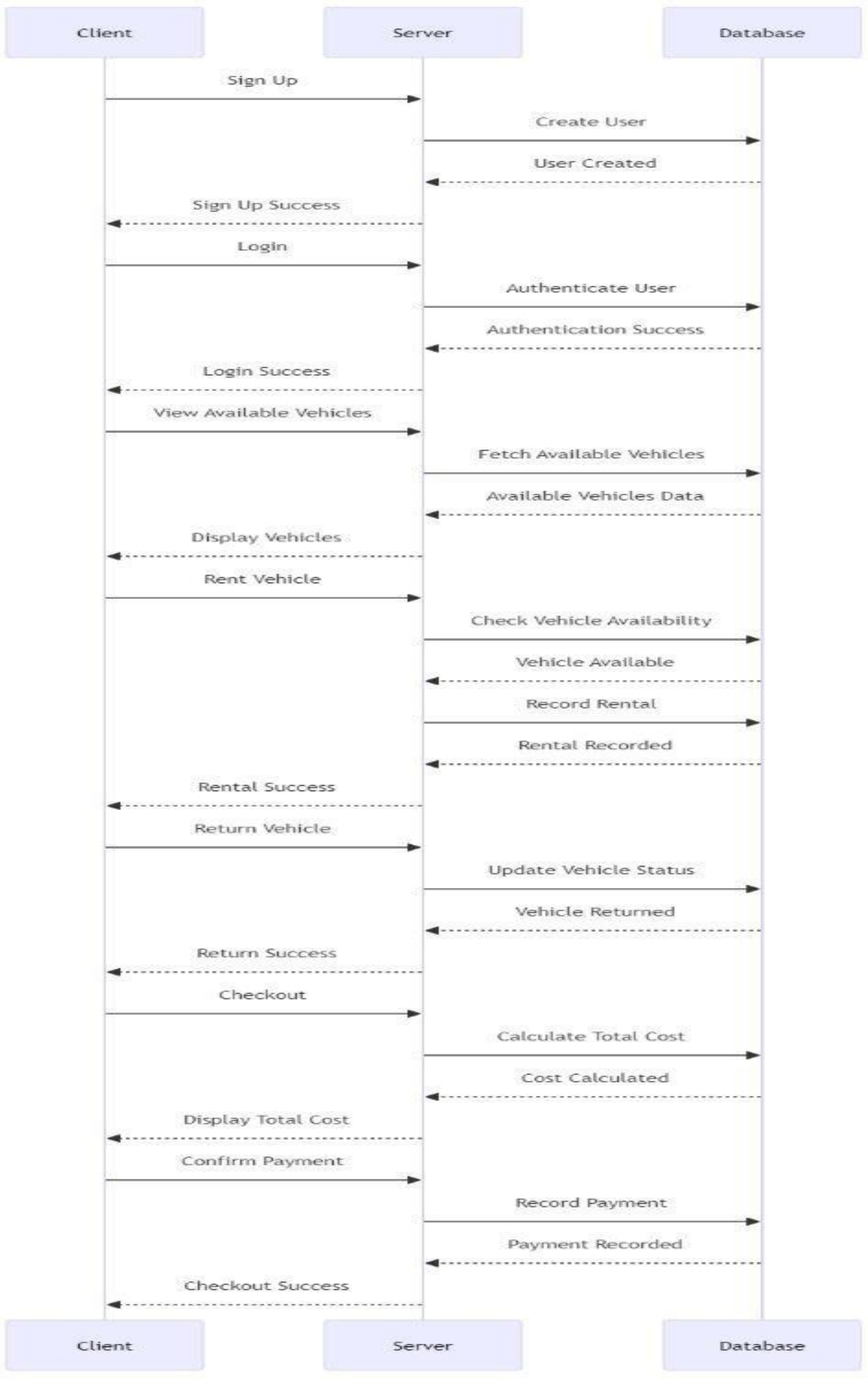
Software Requirements

- Operating System: Windows 7 or later, macOS, or Linux
- Python: Version 3.6 or higher
- MongoDB: Version 3.6 or higher
- Python Libraries:
  - ‘pymongo’ connecting to and interacting with MongoDB (use command pip install pymongo)
  - ‘Streamlit’ building the web application interface (use command pip install streamlit)
  - ‘werkzeug’ for password hashing and security (use command pip install werkzeug)
  - ‘pandas’ for data manipulation and display (use command pip install pandas)

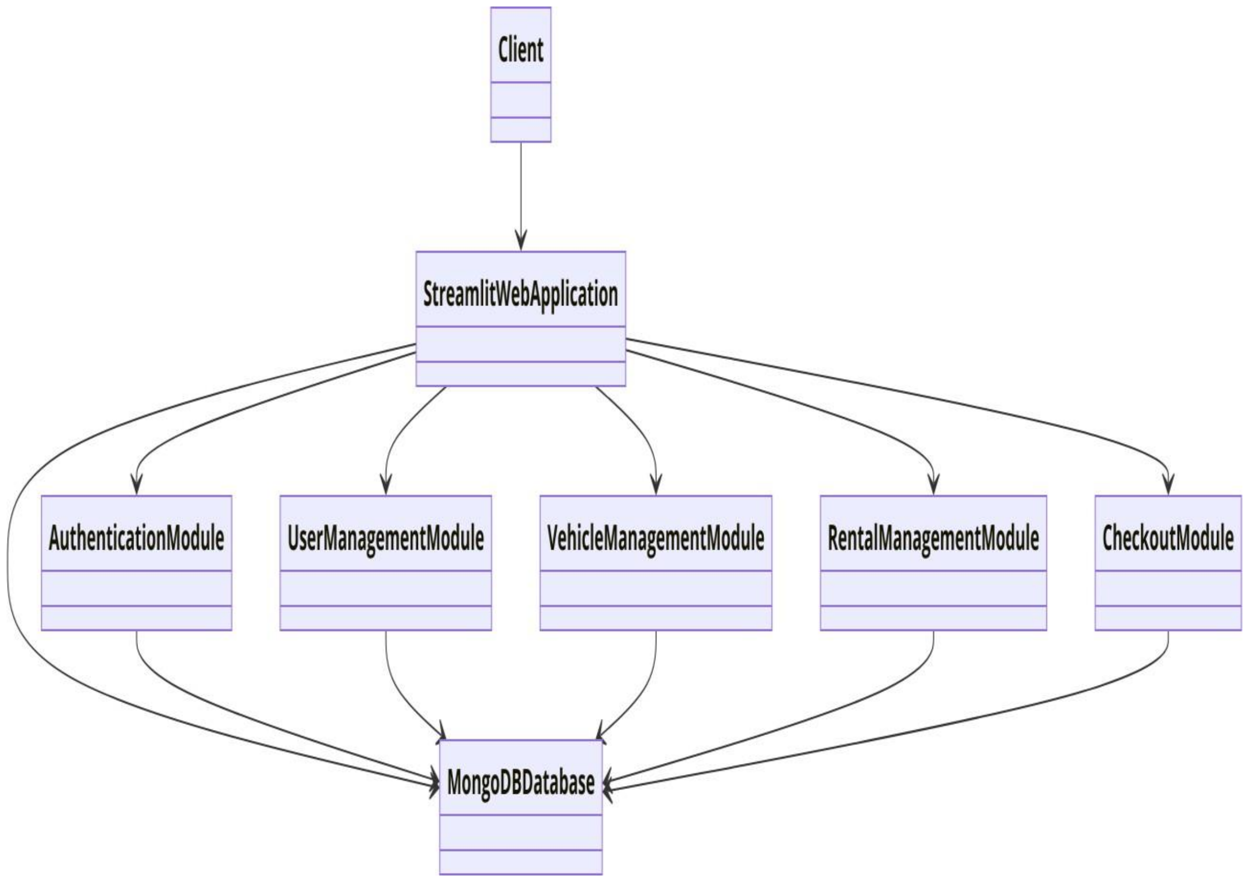
3.3 ER DIAGRAM:



3.4 SEQUENCE DIAGRAM:



3.5 ARCHITECTURE DIAGRAM:



CHAPTER-4

PROGRAM CODE

```
import streamlit as st
from pymongo import MongoClient
from werkzeug.security import generate_password_hash,
check_password_hash
import pandas as pd

# Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["vehicle_rental"]
vehicles_collection = db["vehicles"]
rentals_collection = db["rentals"]
users_collection = db["users"]

# Sidebar for navigation
st.sidebar.title("Navigation")
page = st.sidebar.radio("Go to", ["Sign Up", "Login", "Admin", "Client",
"Available Vehicles", "Checkout"])

# Define the admin key (in a real application, store this securely)
ADMIN_KEY = "admin123"
```

```
def create_user(username, password, name, address, phone, license_no):
    hashed_password = generate_password_hash(password,
method='pbkdf2:sha256')
    user = {
        "username": username,
        "password": hashed_password,
        "name": name,
        "address": address,
        "phone": phone,
        "license_no": license_no
    }
    users_collection.insert_one(user)

def authenticate_user(username, password):
    user = users_collection.find_one({"username": username})
    if user and check_password_hash(user["password"], password):
        return True
    return False

def signup_page():
    st.title("Client Signup")
    username = st.text_input("Username")
    password = st.text_input("Password", type='password')
    name = st.text_input("Name")
    address = st.text_area("Address")
    phone = st.text_input("Phone Number")
    license_no = st.text_input("License Number")
    if st.button("Sign Up"):
        if username and password and name and address and phone and
license_no:
            if users_collection.find_one({"username": username}):
                st.error("Username already exists")
            else:
                create_user(username, password, name, address, phone,
license_no)
                st.success("Signup successful! Please login.")
        else:
            st.warning("Please fill out all fields")

def login_page():
    st.title("Client Login")
    username = st.text_input("Username")
    password = st.text_input("Password", type='password')
    if st.button("Login"):
        if authenticate_user(username, password):
            st.success("Login successful!")
            st.session_state["logged_in"] = True
            st.session_state["username"] = username
            st.experimental_rerun()
        else:
            st.error("Invalid username or password")
```

```
def admin_page():
    st.title("Admin Page")

    # Add Vehicle Section
    st.header("Add a New Vehicle")
    vehicle_id = st.text_input("Vehicle ID")
    vehicle_name = st.text_input("Vehicle Name")
    rental_price = st.number_input("Dollar(s) per Hour", min_value=0.0,
step=0.1)
    if st.button("Add Vehicle"):
        if vehicle_id and vehicle_name and rental_price:
            vehicle = {
                "id": vehicle_id,
                "name": vehicle_name,
                "rental_price": rental_price,
                "rented": False
            }
            vehicles_collection.insert_one(vehicle)
            st.success(f"Vehicle {vehicle_name} added successfully!")
        else:
            st.warning("Please provide vehicle ID, name, and rental price.")

    # Update Vehicle Section
    st.header("Update a Vehicle")
    update_vehicle_id = st.text_input("Enter Vehicle ID to Update")
    new_vehicle_name = st.text_input("New Vehicle Name")
    new_rental_price = st.number_input("Dollar(s) per Hour",
min_value=0.0, step=0.1, key='update_price')
    if st.button("Update Vehicle"):
        if update_vehicle_id:
            update_fields = {}
            if new_vehicle_name:
                update_fields["name"] = new_vehicle_name
            if new_rental_price:
                update_fields["rental_price"] = new_rental_price
            if update_fields:
                vehicles_collection.update_one({"id": update_vehicle_id},
{"$set": update_fields})
                st.success(f"Vehicle {update_vehicle_id} updated
successfully!")
            else:
                st.warning("No new values provided for update.")
        else:
            st.warning("Please provide a vehicle ID to update.")

    # Remove Vehicle Section
    st.header("Remove a Vehicle")
    remove_vehicle_id = st.text_input("Enter Vehicle ID to Remove")
    if st.button("Remove Vehicle"):
        if remove_vehicle_id:
            vehicles_collection.delete_one({"id": remove_vehicle_id})
            st.success(f"Vehicle {remove_vehicle_id} removed successfully!")
        else:
            st.warning("Please provide a vehicle ID to remove.")

def client_page():
    st.title("Client Page")

    # Rent a Vehicle
    st.header("Rent a Vehicle")
    vehicle_to_rent = st.text_input("Enter Vehicle ID to Rent")
    rental_duration = st.number_input("Enter Rental Duration in Hours",
min_value=1, step=1)
    client_name = st.text_input("Client Name",
value=st.session_state.get("username", ""))
    client_address = st.text_area("Client Address")
    client_phone = st.text_input("Client Phone Number")
    client_license = st.text_input("Client License Number")
    if st.button("Rent Vehicle"):
        if vehicle_to_rent and rental_duration and client_name and
client_address and client_phone and client_license:
            vehicle = vehicles_collection.find_one({"id": vehicle_to_rent,
"rented": False})
            if vehicle:
                st.session_state["rental_details"] = {
                    "vehicle_id": vehicle_to_rent,
                    "vehicle_name": vehicle["name"],
                    "rental_price_per_hour": vehicle["rental_price"],
                    "rental_duration": rental_duration,
                    "client_name": client_name,
                    "client_address": client_address,
                    "client_phone": client_phone,
                    "client_license": client_license
                }
                st.experimental_rerun()
            else:
                st.warning("Invalid vehicle ID or vehicle already rented.")
        else:
            st.warning("Please provide all client information.")

    # Return a Vehicle
    st.header("Return a Vehicle")
    vehicle_to_return = st.text_input("Enter Vehicle ID to Return")
    review = st.slider("Rate your experience (1-5 stars)", 1, 5, 5)
    if st.button("Return Vehicle"):
        result = vehicles_collection.update_one({"id": vehicle_to_return,
"rented": True}, {"$set": {"rented": False}})
        if result.modified_count > 0:
            rentals_collection.update_one(
                {"vehicle_id": vehicle_to_return, "rental_status": "rented"},
                {"$set": {"rental_status": "returned", "review": review}}
            )
            st.success(f"Vehicle {vehicle_to_return} returned successfully
with a review of {review} stars!")
```

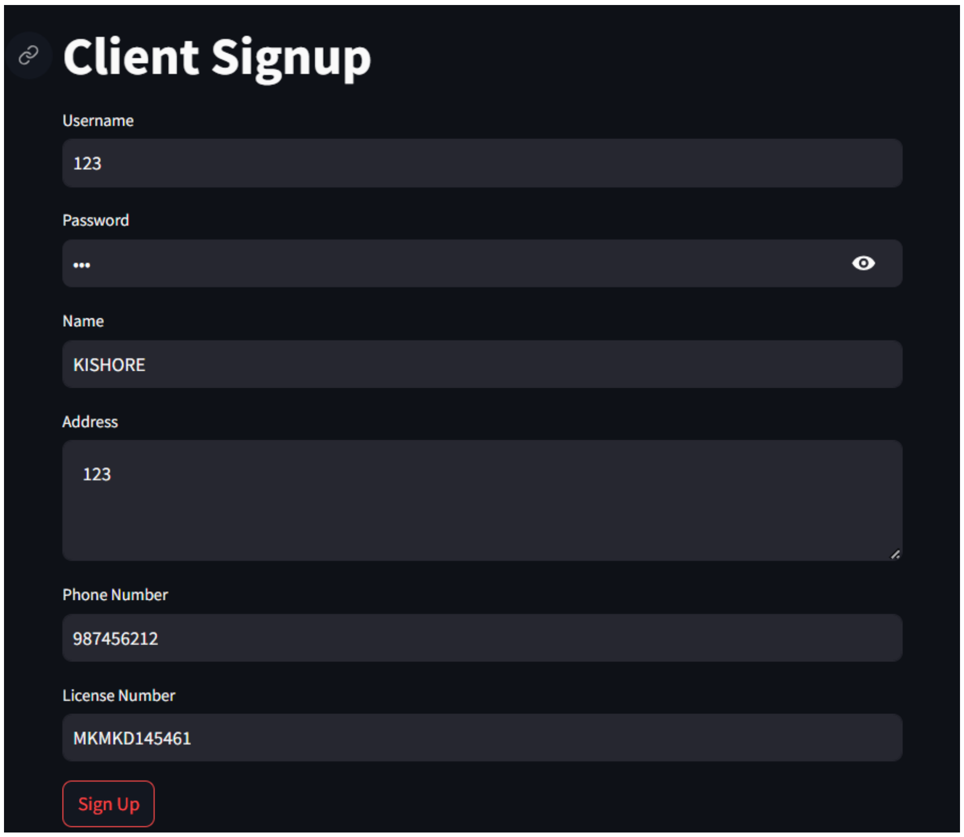
```
        else:
            st.warning("Invalid vehicle ID or vehicle not rented.")

    if st.button("Logout"):
        st.session_state["logged_in"] = False
        st.session_state["username"] = ""
        st.experimental_rerun()

def checkout_page():
    st.title("Checkout Page")

    if "rental_details" in st.session_state:
        rental_details = st.session_state["rental_details"]
        vehicle_name = rental_details["vehicle_name"]
        rental_duration = rental_details["rental_duration"]
        rental_price_per_hour = rental_details["rental_price_per_hour"]
        total_cost = rental_duration * rental_price_per_hour

        st.write(f"Vehicle ID: {rental_details['vehicle_id']}")
        st.write(f"Vehicle Name: {vehicle_name}")
        st.write(f"Rental Duration: {rental_duration} hour(s)")
        st.write(f"Rental Price per Hour: ${rental_price_per_hour:.2f}")
        st.write(f"Total Cost: ${total_cost:.2f}")

        if st.button("Confirm Rental"):
            rental = {
                "vehicle_id": rental_details["vehicle_id"],
                "client_name": rental_details["client_name"],
                "client_address": rental_details["client_address"],
                "client_phone": rental_details["client_phone"],
                "client_license": rental_details["client_license"],
                "rental_duration": rental_duration,
                "total_cost": total_cost,
                "rental_status": "rented"
            }
            rentals_collection.insert_one(rental)
            vehicles_collection.update_one({"id":
rental_details["vehicle_id"]}, {"$set": {"rented": True}})
            st.success("Rental confirmed!")
            del st.session_state["rental_details"]
            st.experimental_rerun()
    else:
        st.warning("No rental details found.")
        st.stop()

def available_vehicles_page():
    st.title("Available Vehicles")

    # Display Available Vehicles
    st.header("List of Available Vehicles")
    available_vehicles = vehicles_collection.find({"rented": False})
    vehicle_list = []
    for vehicle in available_vehicles:
        rental_price = vehicle.get("rental_price", "N/A")
        if rental_price != "N/A":
            rental_price = f"${rental_price:.2f}"
        vehicle_list.append({
            "ID": vehicle["id"],
            "Name": vehicle["name"],
            "Rental Price": rental_price
        })

    if vehicle_list:
        df = pd.DataFrame(vehicle_list)
        st.table(df)
    else:
        st.write("No available vehicles at the moment.")

# Navigation to Admin, Client, Available Vehicles, or Checkout page
if page == "Admin":
    admin_key = st.sidebar.text_input("Enter Admin Key",
type="password")
    if admin_key == ADMIN_KEY:
        admin_page()
    else:
        st.sidebar.error("Invalid Admin Key")
elif page == "Client":
    if "logged_in" in st.session_state and st.session_state["logged_in"]:
        client_page()
    else:
        st.warning("Please login to access this page.")
elif page == "Available Vehicles":
    available_vehicles_page()
elif page == "Checkout":
    checkout_page()
elif page == "Sign Up":
    signup_page()
elif page == "Login":
    login_page()
```

CHAPTER-5

RESULTS AND DISCUSSION

5.1 USER DOCUMENTATION:

USER MANAGEMENT MODULE:



Client Signup

Username: 123

Password: •••

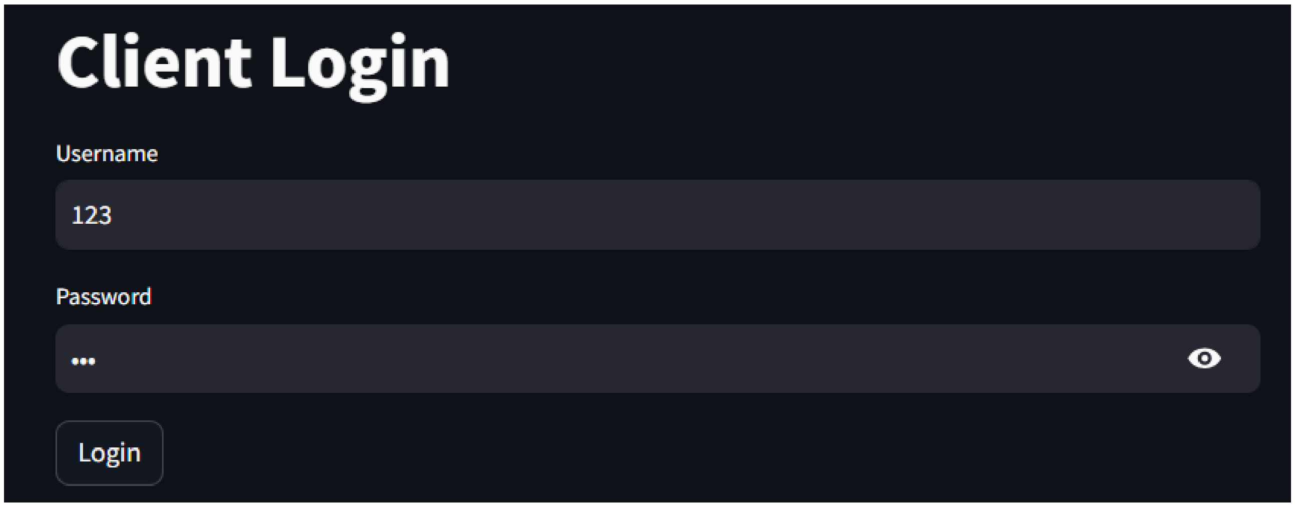
Name: KISHORE

Address: 123

Phone Number: 987456212

License Number: MKMKD145461

USER MANAGEMENT MODULE:



Client Login

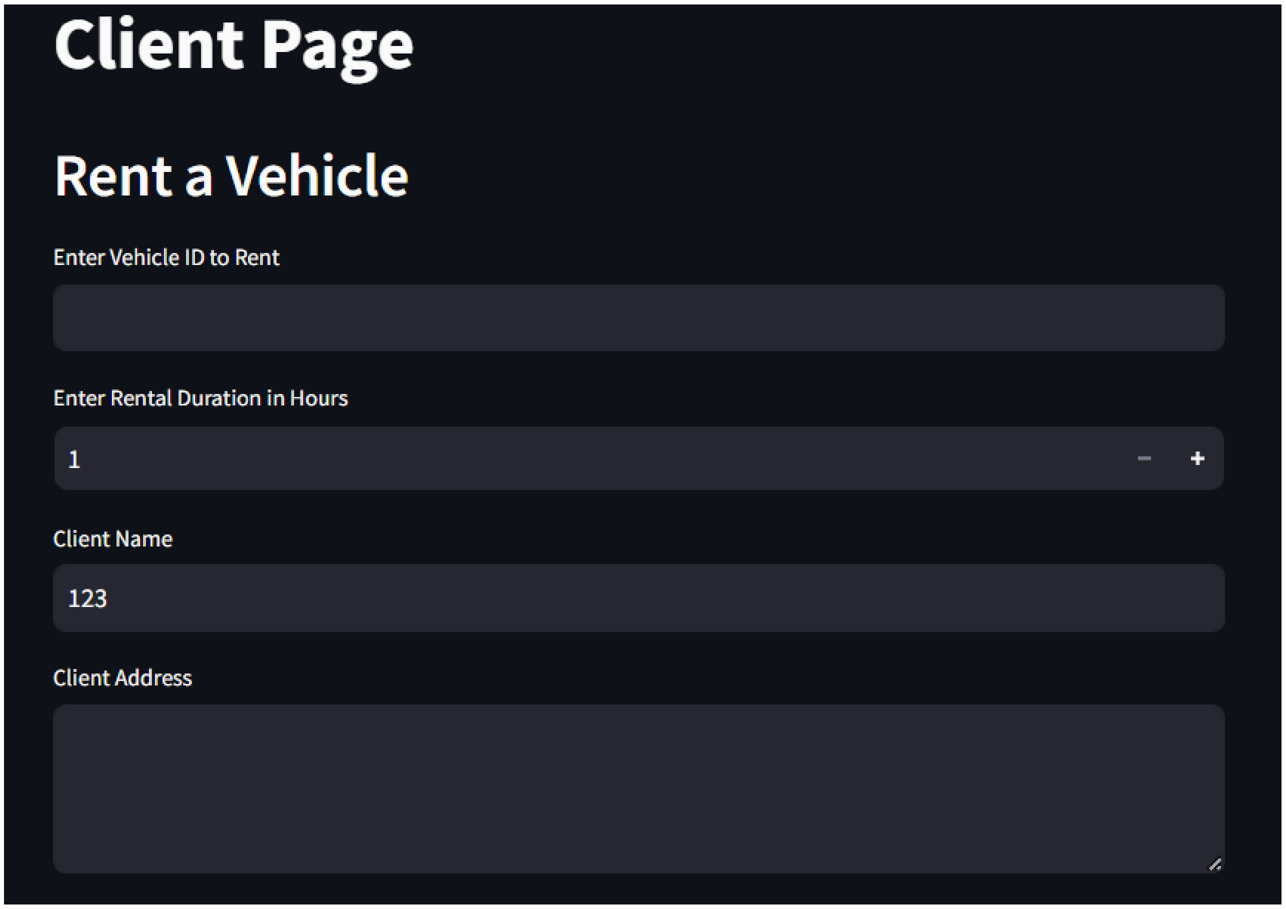
Username

123

Password

•••

Login



Client Page

Rent a Vehicle

Enter Vehicle ID to Rent

Enter Rental Duration in Hours

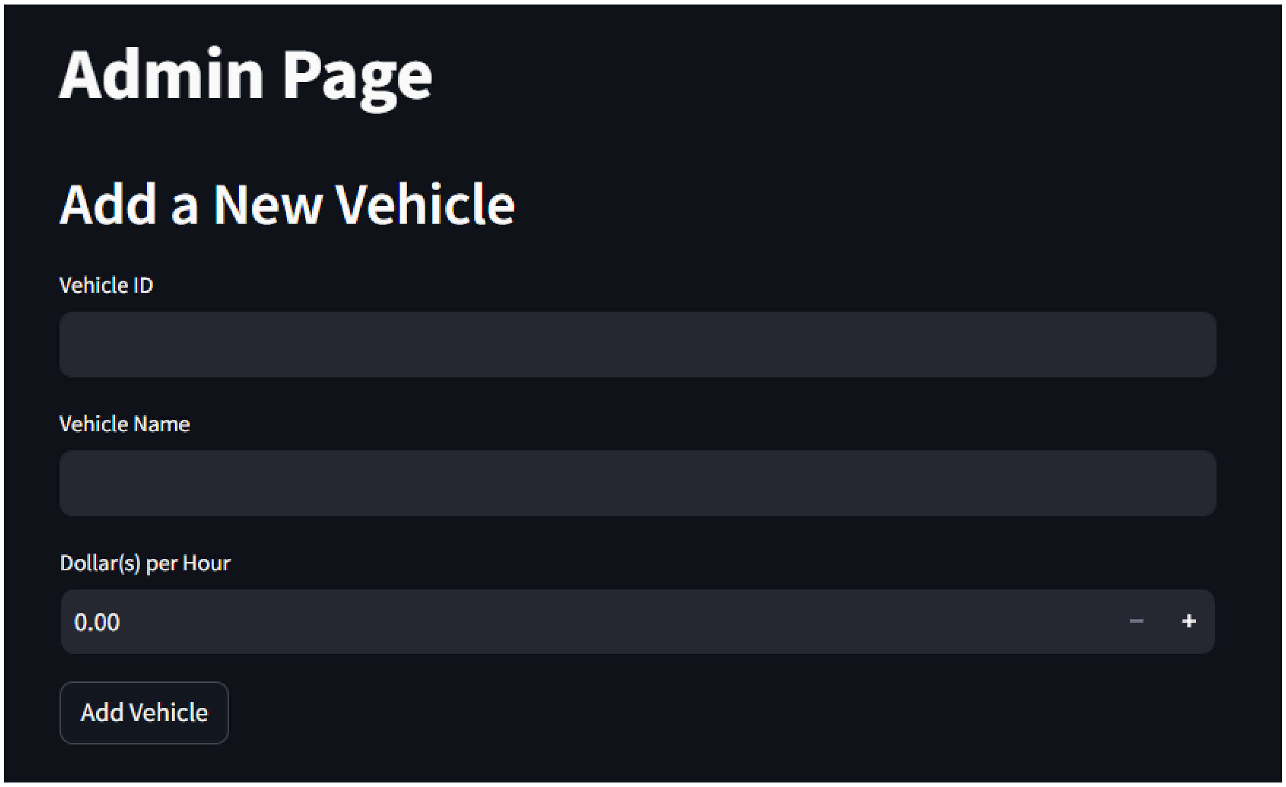
1

Client Name

123

Client Address

ADMIN MANAGEMENT MODULE:



Admin Page

Add a New Vehicle

Vehicle ID

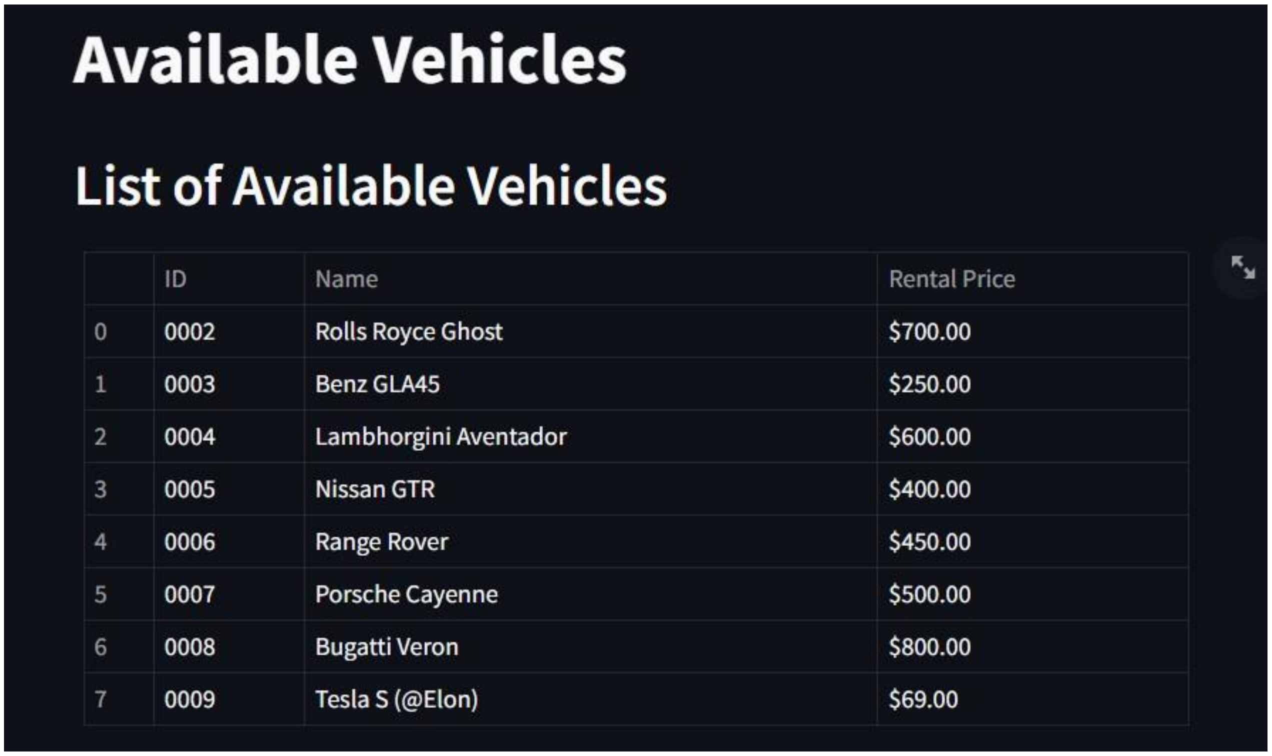
Vehicle Name

Dollar(s) per Hour

0.00

Add Vehicle

VEHICLE MANAGEMENT MODULE:

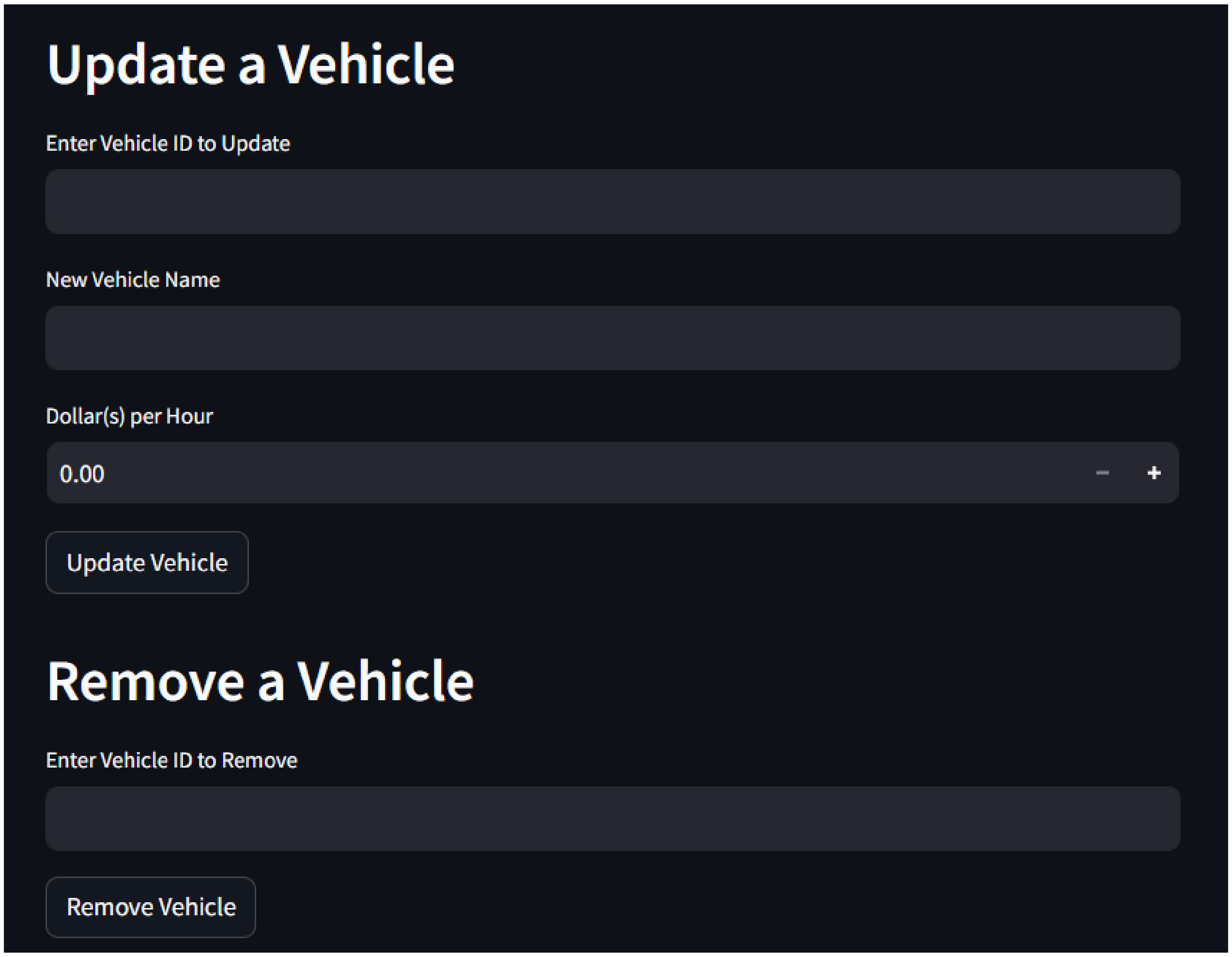


Available Vehicles

List of Available Vehicles

	ID	Name	Rental Price
0	0002	Rolls Royce Ghost	$700.00
1	0003	Benz GLA45	$250.00
2	0004	Lambhorgini Aventador	$600.00
3	0005	Nissan GTR	$400.00
4	0006	Range Rover	$450.00
5	0007	Porsche Cayenne	$500.00
6	0008	Bugatti Veron	$800.00
7	0009	Tesla S (@Elon)	$69.00

DATABASE MANAGEMENT MODULE:


Update a Vehicle

Enter Vehicle ID to Update

New Vehicle Name

Dollar(s) per Hour

0.00

Remove a Vehicle

Enter Vehicle ID to Remove

CHAPTER-6

6.1 FUTURE IMPROVEMENTS

1. Improve Security

Password Policy: Implement a stronger password policy (e.g., minimum length, special characters).

Two-Factor Authentication: Add two-factor authentication (2FA) for enhanced security.

Rate Limiting: Implement rate limiting to prevent brute-force attacks on login endpoints.

2. Enhance User Experience

Form Validation: Add client-side and server-side validation for all input forms to improve user experience and data integrity.

Real-time Updates: Use WebSockets or similar technologies to provide real-time updates for vehicle availability.

Responsive Design: Ensure the app is fully responsive and works well on different devices (mobile, tablet, desktop).

3. Advanced Features for Vehicle Management

Vehicle Maintenance: Add functionality to track and manage vehicle maintenance schedules.

Vehicle History: Maintain a history of rentals for each vehicle, including past renters, rental durations, and reviews.

Dynamic Pricing: Implement dynamic pricing based on factors like demand, time of day, or special events.

4. Improved Client Features

Rental History: Allow clients to view their past rentals, including dates, vehicle details, and reviews.

Profile Management: Enable clients to update their profile information and view their rental history.

Notifications: Implement email or SMS notifications for important events (e.g., rental confirmation, return reminders).

5. Admin Dashboard Enhancements

Analytics: Provide detailed analytics and reports on rental statistics, revenue, and vehicle utilization.

Bulk Operations: Allow admins to perform bulk operations, such as importing/exporting vehicle data or updating multiple vehicles at once.

User Management: Include functionality for managing user accounts (e.g., resetting passwords, deactivating accounts).

CHAPTER-7

7.1 CONCLUSION:

The vehicle rental system represents a significant advancement in the management of vehicle rental businesses, offering a user-friendly platform for both administrators and clients. By leveraging Streamlit for the frontend interface and MongoDB for backend data storage, the system ensures seamless navigation and robust data handling. From client signup to vehicle management and rental processing, every aspect of the rental process is streamlined for maximum efficiency and convenience.

Moreover, the system's adaptability and scalability make it suitable for businesses of all sizes, from small rental agencies to large-scale operations. Its intuitive design and comprehensive feature set empower administrators to maintain an up-to-date fleet, while clients benefit from a hassle-free rental experience. Overall, the vehicle rental system represents a significant leap forward in modernizing rental operations, enhancing customer satisfaction, and driving business growth in the competitive vehicle rental industry.

CHAPTER-8

8.1 REFERENCES:

1. Python Documentation : Python Software Foundation. (n.d.). Available at: https://docs.python.org/3/

2. Streamlit, Inc. (2021) ‘Streamlit Documentation’, Streamlit, Available at: https://docs.streamlit.io/

3. Williams, K. (2018) ‘MongoDB in Action’, Manning Publications, pp.1-375.

4. Davidson, P. (2015) ‘MongoDB Basics’, Apress, pp.1-250.

5. Greene, M. (2020) ‘Practical Python Programming: 100+ Essential Coding Exercises and Projects’, No Starch Press, pp.1-450.