# Transport Management System

"Swift Move: Simplified Transport Operations"

## Vaishnavi Patel

**1. Introduction**

**1.1 Objective**

The TMS is designed to make the management of transport operations easier and much more organized. The user can quite easily register and log in to make a transport schedule booking with no hassle, but at the same time, administrators may use the system in managing vehicles, creating or updating schedules, so everything will be running fine and smooth. With its simple design and easy-to-use features, TMS helps reduce mistakes as well as it is time saving and ensures that everything is well managed for both users and administrators.

**1.2 Purpose**

This project overcomes the problems that arise when managing transport manually. The project offers a user-friendly and efficient platform, making things simpler for the users and also for the administrators. Instead of time-consuming and error-prone methods of manual management, the system serves as a modern solution, as it provides easy booking of transport schedules and proper vehicle management with schedule management on the part of the administrators. Replacing these traditional processes with the user-friendly system saves time and reduces errors, thus making it easy to experience improved results for anyone using them.

**2. System Overview**

The TMS streamlines to simplify and enhance the management of transport operations. It has a variety of rich features made to service both users and administrators alike for an experience that is seamless and efficient. Let's take a closer look at what this system has in store:

**2.1 Features**

1. **User Registration and Login**
   * Users can easily create an account and log in to access the platform's features.
   * The system ensures security by using password hashing to protect user information.
2. **Admin Functionalities**
   * Administrators have powerful tools to manage the transport system:
     + Add New Vehicles: Easily register vehicles into the system, specifying details like type, capacity, and status.
     + Create and Manage Schedules: Admins can plan routes, set departure and arrival times, and assign vehicles, making schedule management hassle-free.
3. **User Functionalities**
   * Designed with simplicity in mind, users can:
     + View Available Schedules: Browse through a list of transport options to find the one that best suits their needs.
     + Book Schedules: Securely book a spot on their preferred schedule with just a few clicks, ensuring convenience and ease.
4. **Database Integration**
   * At the heart of the system is MongoDB, a dynamic and efficient database solution.
   * MongoDB enables real-time data storage and retrieval, ensuring smooth operations and quick responses for both users and admins.

TMS combines functionality, security, and usability in one platform. This line distinguishes tradition transport management from modern streamlined services. Whether you're a user booking a journey or an admin managing operations, the system has been designed to help you along in your tasks, making them easier and enjoyable..

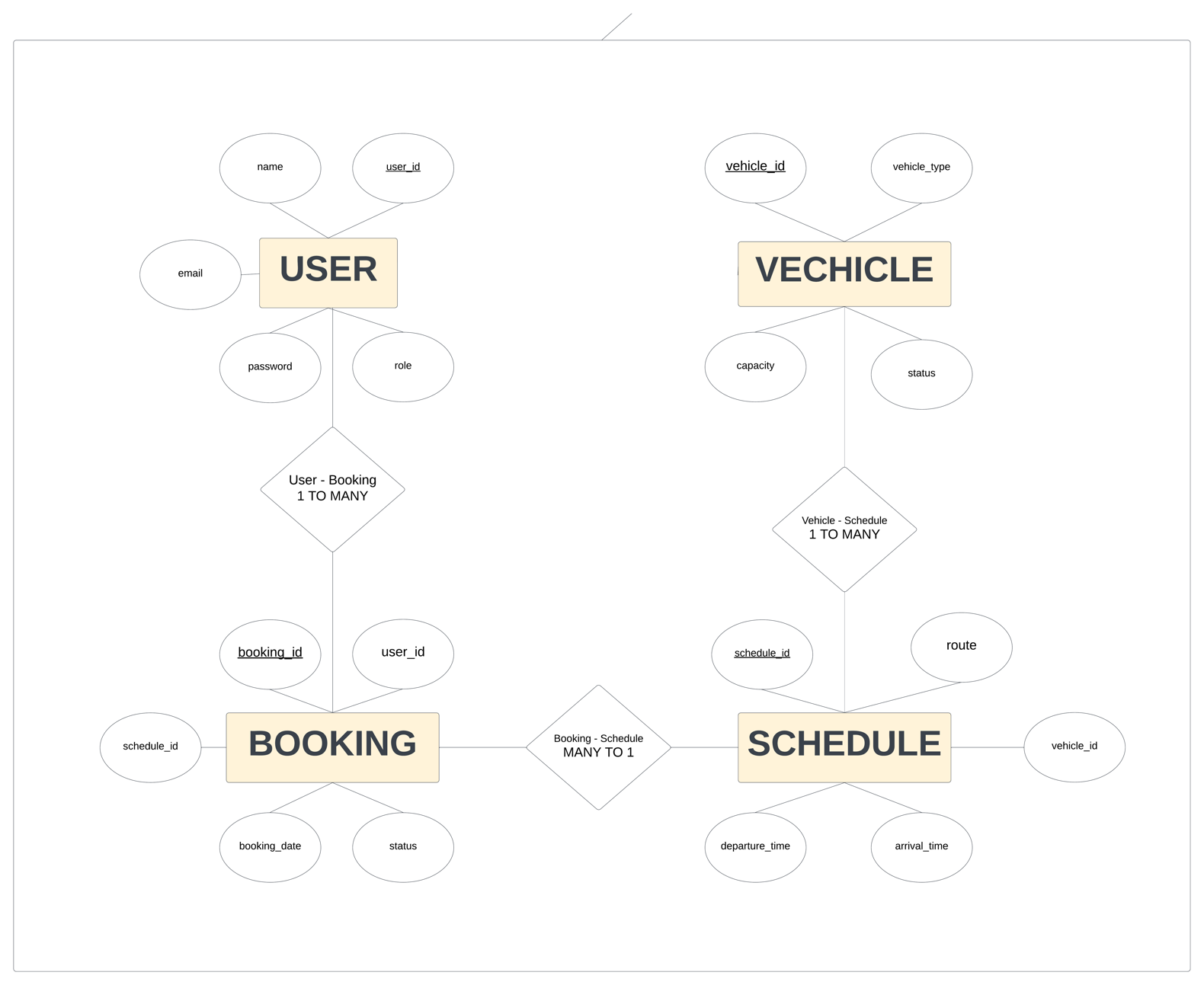
**3. Technologies Used**

* The TMS is built from a robust and modern tech stack that boasts very powerful tools and libraries that result in a very secure, efficient, and user-friendly platform. All technologies selected were based on enhancement in functionality and an effortless experience for both users and administrators. Here's what powers the system:
* **Backend: Python**
  + Python forms the backbone of TMS, offering simplicity, flexibility, and reliability for handling business logic and data processing.
* **Frontend: Streamlit**
  + The user interface is designed with Streamlit, making it interactive and intuitive. Streamlit provides a clean, web-based platform that simplifies user interactions, whether booking a schedule or managing operations.
* **Database: MongoDB**
  + At the core of the system is MongoDB, a NoSQL database that efficiently handles dynamic data. It ensures that all user data, schedules, and vehicle information are securely stored and instantly accessible when needed.
* **Security: Password Hashing with Werkzeug**
  + To protect user information, especially passwords, the system uses Werkzeug, which provides robust password hashing techniques. This ensures that user accounts remain safe and secure from unauthorized access.
* **Libraries:**
  + pymongo: This library facilitates smooth interaction with the MongoDB database, enabling the system to query, update, and manage data effortlessly.
  + bson: Used for handling ObjectId operations, ensuring accurate identification and management of database records.

The combination of such technologies ensures that TMS does not only provide a guarantee of efficiency and security but also ease of use and maintenance. Starting from the simplest yet powerful Python backend to an engaging Streamlit frontend, every single bit works in great harmony to deliver reliability and joy. Whether it's a ride booking or schedule management, TMS leverages such technologies to make it happen in the silken glove.

**4. System Design**

The Transport Management System is designed with a lot of thought, ensuring that users and administrators use it effortlessly. Its general structure looks simple, modular, and highly efficient; hence, all the interactions will be smooth and free of errors. Here's the overview of this system design:



**4.1 Architecture**

* **User Interface:**  
  The Streamlit web application powers the front-facing side of the system. This makes it easy to navigate, be it to book schedules or manage operations with its clean and interactive design.
* **Business Logic:**  
  The core functionality is encapsulated in modular Python functions, thereby making the system well-structured and easy to maintain. Updates and improvements come in quite handy since they enhance the modularity.
* **Database:**  
  This data is then efficiently managed by the system using MongoDB, a NoSQL database, by putting it into collections for:  
   o Users: Storing user details and credentials.  
   o Vehicles: Keeping track of all registered transport options.  
   o Schedules: Maintaining route, timing, and vehicle details.  
   o Bookings: Secure recording of user bookings.
* **4.2 Workflow**

The workflow of the system is designed to be logical, user-friendly, and efficient:

1. **User Registration and Login:**
   * New users can create an account and securely log in to access the system's features.
   * Returning users can log in to continue booking or managing schedules.
2. **Admin Functions:**
   * Administrators can add and manage vehicles, including their type, capacity, and availability.
   * Schedules can be created and updated, specifying routes, departure times, and the assigned vehicle.
3. **User Booking:**
   * Logged-in users can view available schedules and book their preferred transport options.
   * The booking process is quick and straightforward, ensuring a hassle-free experience.
4. **Data Storage:**
   * All bookings, user details, schedules, and vehicle information are securely stored in the MongoDB database, ensuring data consistency and reliability.

The system design focuses on making every interaction simple, fast, and secure. Whether you're an admin managing operations or a user looking to book a ride, the TMS architecture and workflow work together to ensure that everything is handled smoothly and efficiently.

**4.3 Database Schema**

* **Users**:

json

{

"name": "string",

"email": "string",

"password": "hashed\_string",

"role": "string"

}

* **Vehicles**:

json

{

"type": "string",

"capacity": "integer",

"status": "string"

}

* **Schedules**:

json

{

"route": "string",

"departure\_time": "datetime",

"arrival\_time": "datetime",

"vehicle\_id": "ObjectId"

}

* **Bookings**:

json

{

"user\_id": "ObjectId",

"schedule\_id": "ObjectId",

"booking\_date": "datetime",

"status": "string"

}

**4.3 Database Implementation**

The **Transport Management System (TMS)** integrates a robust and flexible database solution using **MongoDB**, a NoSQL database that excels at handling dynamic and complex data structures. This database is the backbone of the system, ensuring that all information is securely stored, easily retrievable, and efficiently managed. Here's how we implemented MongoDB in the project:

**Database Setup**

1. **Connection to MongoDB**:
   * We connected the project to a local MongoDB server using the **pymongo** library. The connection ensures that the application communicates seamlessly with the database.

python

from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")

db = client['MYDB1']

The database is named MYDB1, where all collections are stored to manage users, vehicles, schedules, and bookings.

1. **Database Structure**:
   * The database consists of four collections, each tailored to specific functionalities:
     + **Users**: Stores user details such as name, email, hashed passwords, and roles (user or admin).
     + **Vehicles**: Contains information about the type of vehicle, capacity, and status (available or unavailable).
     + **Schedules**: Manages transport schedules, including routes, departure times, arrival times, and the vehicle assigned.
     + **Bookings**: Tracks bookings made by users, linking them to the corresponding schedule and user ID.

Example:

python

users = db['users']

vehicles = db['vehicles']

schedules = db['schedules']

bookings = db['bookings']

**Database Integration in Features**

1. **User Management**:
   * During registration, user data (name, email, and hashed password) is inserted into the users collection. Passwords are securely hashed using the **Werkzeug** library to ensure data protection.
   * For login, the system queries the database to verify user credentials and retrieve their role (admin or user).
2. **Vehicle and Schedule Management**:
   * Admins can add vehicles, specifying their type, capacity, and availability status, which are stored in the vehicles collection.
   * Schedules are added by associating routes, timings, and vehicle IDs, and this data is stored in the schedules collection.
3. **Booking Process**:
   * When a user books a schedule, the booking information, including the user ID, schedule ID, and booking date, is stored in the bookings collection.
   * The **bson.ObjectId** is used to handle unique identifiers for schedules and users, ensuring data integrity.
4. **Real-Time Updates**:
   * The MongoDB integration allows the system to dynamically update or query collections, ensuring that users and admins always have access to the latest data.

**5. Implementation Details**

**5.1 Key Functions**

* **User Registration**  
  This function allows new users to create an account. During the registration process, the system securely hashes their password using generate\_password\_hash to ensure their credentials are protected. The registration flow is designed to be simple and user-friendly, ensuring a smooth onboarding experience.
* **User Login**  
  The login function authenticates users by verifying their entered credentials (username and password) against the records in the system. If the credentials are valid, the user is granted access to their account. The system ensures that login attempts are secure and reliable, with appropriate error messages if authentication fails.
* **Add Vehicle (Admin Only)**  
  This feature is exclusive to admin users and allows them to add new vehicles to the transport system. Each vehicle entry includes essential details such as type, capacity, and operational status, helping the system maintain accurate and up-to-date information about available transport resources.
* **Add Schedule (Admin Only)**  
  Admin users can create transport schedules that are linked to specific vehicles. This function enables admins to define routes, departure times, and frequencies, ensuring that the transport system runs efficiently and users can access up-to-date schedule information. It allows admins to maintain a smooth and organized timetable for users to follow.
* **Book Schedule (For Users)**  
  This function allows users to browse available schedules and book their preferred transport options. After selecting a schedule, users can confirm their reservation, which is then processed and confirmed in the system. It offers an intuitive booking experience, ensuring that users can easily secure their desired ride without complications.

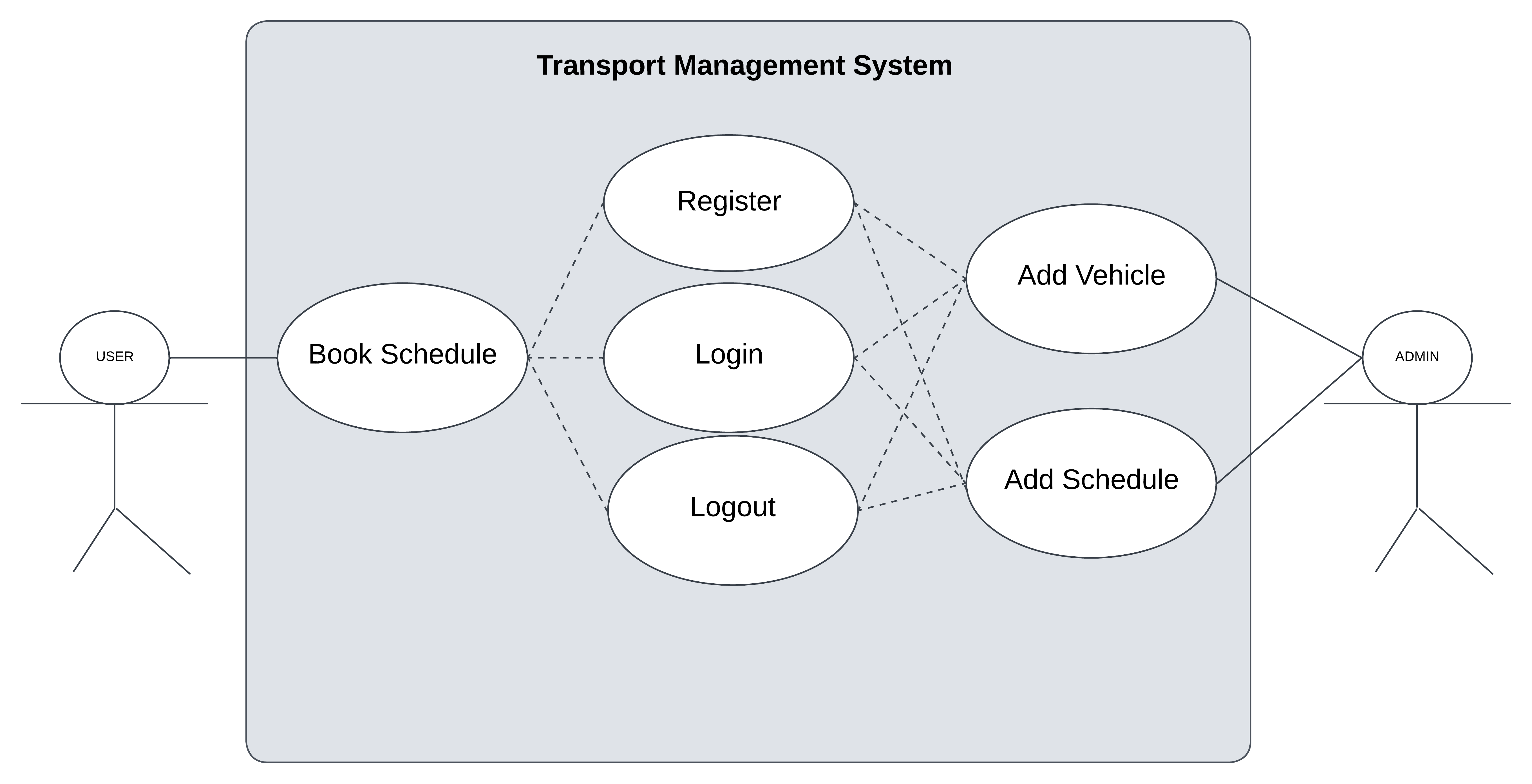
**6. Challenges and Solutions**

**6.1 Challenges**

1. **Database Connectivity**  
   *Challenge:* One of the primary challenges we encountered was ensuring seamless communication between the application and MongoDB. Frequent database connection drops or failures could disrupt the entire system, making it unreliable for users.  
   *Solution:* To address this, we implemented robust error handling mechanisms and utilized connection pooling. Connection pooling ensures that a pool of connections is maintained, which minimizes the overhead of opening new connections every time one is needed. This improves the overall efficiency and reliability of our system, making database interactions smoother and faster.
2. **Schedule ID Conversion**  
   *Challenge:* A key issue arose when handling the conversion of MongoDB's ObjectId to appropriate formats during schedule bookings. The ObjectId, being a unique identifier in MongoDB, sometimes caused issues when mapping it correctly for schedule retrieval and bookings. Incorrect handling of these conversions could lead to errors in fetching or processing schedule data.  
   *Solution:* To resolve this, we implemented precise type checks and utilized bson for proper ObjectId handling. This ensured that ObjectIds were correctly converted and validated before being processed. The use of bson helped to avoid errors during data retrieval, allowing for accurate and consistent schedule bookings.

**7. Future Scope**

1. **Real-time Notifications**  
   Looking ahead, we plan to enhance the user experience by integrating real-time notifications. This will allow users to stay updated with important information such as schedule changes, booking confirmations, or any other relevant updates in real time. Notifications will be sent via in-app alerts or push notifications, ensuring users are always in the loop without having to manually check for updates.
2. **Advanced User Roles**  
   To improve the system’s flexibility and management capabilities, we envision introducing advanced user roles, such as sub-admins or supervisors. These roles will enable a hierarchical structure, allowing specific users to manage certain aspects of the platform (like adding vehicles or editing schedules) while maintaining overall security and control for admins. This will empower organizations to better delegate responsibilities and streamline operations.
3. **Data Analytics**  
   To further enhance decision-making and improve service delivery, we plan to integrate data analytics into the system. By analyzing patterns in bookings, schedule usage, and vehicle performance, we will provide valuable insights to admins. This could help in optimizing routes, improving vehicle maintenance schedules, and even predicting demand trends to better serve users.
4. **Payment Gateway Integration**  
   In the near future, we aim to add a payment gateway to facilitate online payments for bookings. This integration will allow users to pay securely for their transport reservations directly through the platform. Whether it's through credit/debit cards or digital wallets, offering a seamless payment experience will simplify the booking process and increase overall convenience for users.



**8. Code**

**8.1 Backend Code**

python

from pymongo import MongoClient

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from datetime import datetime

import bson

client = MongoClient("mongodb://localhost:27017/")

db = client['MYDB1']

users = db['users']

vehicles = db['vehicles']

schedules = db['schedules']

bookings = db['bookings']

def register\_user(name, email, password, role='user'):

    if users.find\_one({"email": email}):

        return "Email already exists."

    hashed\_password = generate\_password\_hash(password)

    users.insert\_one({"name": name, "email": email, "password": hashed\_password, "role": role})

    return "Registration successful!"

def login\_user(email, password):

    user = users.find\_one({"email": email})

    if user and check\_password\_hash(user['password'], password):

        return user

    return None

def add\_vehicle(vehicle\_type, capacity, status):

    vehicles.insert\_one({"type": vehicle\_type, "capacity": capacity, "status": status})

    return "Vehicle added successfully."

def add\_schedule(route, departure\_time, arrival\_time, vehicle\_id):

    schedules.insert\_one({

        "route": route,

        "departure\_time": departure\_time,

        "arrival\_time": arrival\_time,

        "vehicle\_id": vehicle\_id

    })

    return "Schedule added successfully."

def book\_schedule(user\_id, schedule\_id):

    booking\_date = datetime.now().strftime('%Y-%m-%d')

    bookings.insert\_one({

        "user\_id": user\_id,

        "schedule\_id": bson.ObjectId(schedule\_id),

        "booking\_date": booking\_date,

        "status": "confirmed"

    })

    return "Booking confirmed."

**Modular breakdown for the app.py**

**1. User Management Module**

* **Purpose**: Handles user registration and login to manage access to the system.
* **Functions**:
  + **register\_user(name, email, password, role)**:
    - Registers a new user by taking their name, email, password, and role. The password is hashed for security purposes before storing it in the users collection.
    - Checks if the email already exists to avoid duplicate registrations.
  + **login\_user(email, password)**:
    - Authenticates an existing user by checking their email and verifying the password hash. Returns user data if login is successful, otherwise returns None.

**2. Vehicle Management Module**

* **Purpose**: Manages the details of vehicles available in the transport system.
* **Functions**:
  + **add\_vehicle(vehicle\_type, capacity, status)**:
    - Adds a new vehicle to the vehicles collection, including details like vehicle type, capacity, and operational status (e.g., "Available" or "Unavailable").
    - Primarily used by admins to ensure the fleet is up-to-date.

**3. Schedule Management Module**

* **Purpose**: Handles creation and management of transport schedules for each route.
* **Functions**:
  + **add\_schedule(route, departure\_time, arrival\_time, vehicle\_id)**:
    - Adds a schedule to the schedules collection, linking it to a vehicle via vehicle\_id. This function records route information, departure and arrival times, and the vehicle associated with that schedule.

**4. Booking Management Module**

* **Purpose**: Manages the booking process for users to reserve transport on scheduled routes.
* **Functions**:
  + **book\_schedule(user\_id, schedule\_id)**:
    - Books a transport schedule for a user by inserting a booking record in the bookings collection.
    - This function associates a user with a schedule by linking user ID and schedule ID and includes the booking date and status ("confirmed").
    - Ensures that users can confirm bookings and view them in the booking system.

**8.2 Frontend Code**

python

import streamlit as st

from app import register\_user, login\_user, add\_vehicle, add\_schedule, book\_schedule

from app import vehicles, schedules, bookings

from bson.objectid import ObjectId

from bson.errors import Inv

alidId

st.title("Transport Management System")

st.sidebar.title("Navigation")

options = ["Home", "Register", "Login", "Add Vehicle (Admin)", "Add Schedule (Admin)", "Book Schedule"]

choice = st.sidebar.selectbox("Choose Action", options)

if 'user' not in st.session\_state:

    st.session\_state['user'] = None

if choice == "Home":

    st.write("Welcome to the Transport Management System!")

    if st.session\_state['user']:

        st.write(f"Logged in as {st.session\_state['user']['name']}")

    else:

        st.write("Please log in or register to continue.")

elif choice == "Register":

    st.subheader("User Registration")

    name = st.text\_input("Name")

    email = st.text\_input("Email")

    password = st.text\_input("Password", type="password")

    role = st.selectbox("Role", ["user", "admin"])

    if st.button("Register"):

        message = register\_user(name, email, password, role)

        st.success(message)

elif choice == "Login":

    st.subheader("User Login")

    email = st.text\_input("Email")

    password = st.text\_input("Password", type="password")

    if st.button("Login"):

        user = login\_user(email, password)

        if user:

            st.session\_state['user'] = user

            st.success(f"Logged in as {user['name']}")

        else:

            st.error("Invalid email or password.")

elif choice == "Add Vehicle (Admin)" and st.session\_state['user'] and st.session\_state['user']['role'] == 'admin':

    st.subheader("Add New Vehicle")

    vehicle\_type = st.text\_input("Vehicle Type")

    capacity = st.number\_input("Capacity", min\_value=1)

    status = st.selectbox("Status", ["Available", "Unavailable"])

    if st.button("Add Vehicle"):

        message = add\_vehicle(vehicle\_type, capacity, status)

        st.success(message)

elif choice == "Add Schedule (Admin)" and st.session\_state['user'] and st.session\_state['user']['role'] == 'admin':

    st.subheader("Add New Schedule")

    route = st.text\_input("Route")

    departure\_time = st.text\_input("Departure Time (YYYY-MM-DD HH:MM)")

    arrival\_time = st.text\_input("Arrival Time (YYYY-MM-DD HH:MM)")

    vehicle\_id = st.selectbox("Vehicle", [(str(v["\_id"]), v["type"]) for v in vehicles.find()])

    if st.button("Add Schedule"):

        message = add\_schedule(route, departure\_time, arrival\_time, vehicle\_id)

        st.success(message)

elif choice == "Book Schedule" and st.session\_state['user']:

    st.subheader("Book a Schedule")

    schedule\_options = [(str(s["\_id"]), s["route"]) for s in schedules.find()]

    selected\_schedule = st.selectbox("Schedule", schedule\_options)

    if st.button("Book"):

        schedule\_id = selected\_schedule[0]

        try:

            st.write(f"Selected Schedule ID: {schedule\_id}, Type: {type(schedule\_id)}")

            schedule\_id\_obj = ObjectId(schedule\_id)

            message = book\_schedule(st.session\_state['user']['\_id'], schedule\_id\_obj)

            st.success(message)

        except InvalidId:

            st.error("Invalid schedule ID. Please select a valid schedule.")

        except TypeError as e:

            st.error(f"TypeError: {e}")

if st.sidebar.button("Logout"):

    st.session\_state['user'] = None

    st.success("Logged out successfully.")

**Modular breakdown for the Streamlit Frontend Interface**

**1. App Setup and Navigation Module**

* **Purpose**: Sets up the title, sidebar, and navigation options for the app.
* **Key Elements**:
  + **App Title**: Displays the app's title at the top.
  + **Sidebar**: Provides navigation options, such as "Home," "Register," "Login," "Add Vehicle (Admin)," "Add Schedule (Admin)," and "Book Schedule."
  + **Session State**: Manages logged-in user state using st.session\_state.

**2. Home Page Module**

* **Purpose**: Acts as a welcome page that displays a message depending on the user’s login status.
* **Key Elements**:
  + **Logged-in Check**: If a user is logged in, it shows a welcome message with the user's name.
  + **Prompt to Login/Register**: If no user is logged in, it prompts the user to either register or log in to proceed.

**3. User Registration Module**

* **Purpose**: Allows new users to register with details like name, email, password, and role.
* **Key Elements**:
  + **Fields**: Collects the user's name, email, password, and role (user/admin).
  + **Registration Action**: When the user clicks "Register," the data is sent to the backend register\_user function.
  + **Feedback**: Provides feedback on registration success or failure.

**4. User Login Module**

* **Purpose**: Handles user login, allowing access to different features based on their role.
* **Key Elements**:
  + **Fields**: Collects email and password.
  + **Login Action**: Sends the credentials to the login\_user function to verify the user.
  + **Session Update**: If login is successful, the user’s details are stored in st.session\_state['user'].
  + **Error Handling**: Displays an error message if login fails.

**5. Vehicle Management Module (Admin Only)**

* **Purpose**: Allows an admin to add a new vehicle to the system.
* **Key Elements**:
  + **Fields**: Collects vehicle type, capacity, and status.
  + **Action**: When the admin clicks "Add Vehicle," the data is passed to add\_vehicle for storage.
  + **Admin Check**: Only available if the logged-in user is an admin.

**6. Schedule Management Module (Admin Only)**

* **Purpose**: Allows admins to add a new transport schedule, specifying route details and assigning vehicles.
* **Key Elements**:
  + **Fields**: Collects route, departure time, arrival time, and associated vehicle ID.
  + **Vehicle Selection**: Lists available vehicles from the database to assign to a route.
  + **Admin Check**: Only accessible if the logged-in user is an admin.
  + **Action**: On "Add Schedule," calls the add\_schedule function to save the schedule to the database.

**7. Booking Management Module (User Only)**

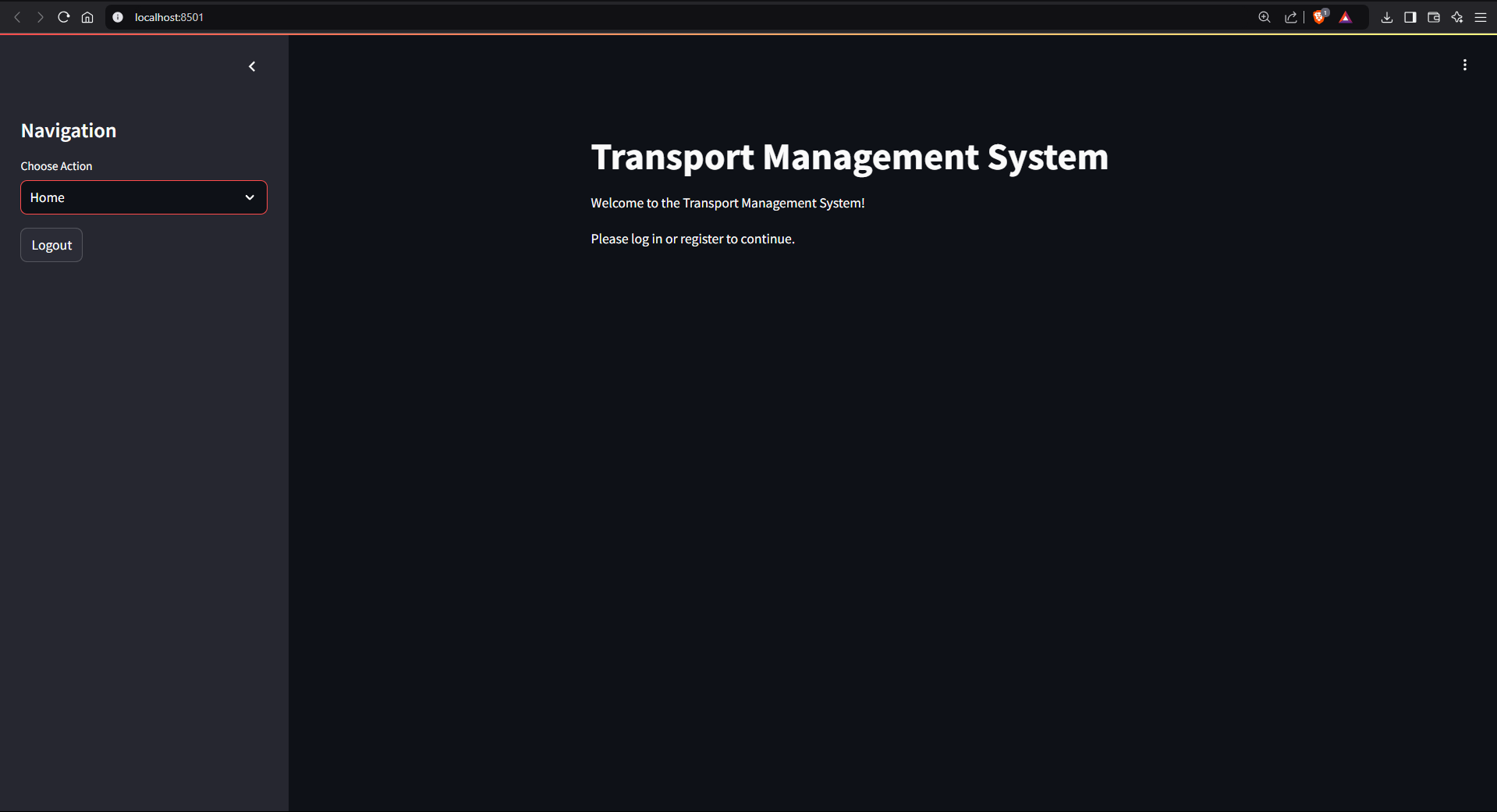
* **Purpose**: Allows a logged-in user to book a specific transport schedule.
* **Key Elements**:
  + **Schedule Selection**: Provides a dropdown list of available schedules for the user to choose from.
  + **Action**: On clicking "Book," the selected schedule is linked to the user through the book\_schedule function.
  + **User Check**: Only accessible if the user is logged in.

**8. Logout Module**

* **Purpose**: Allows a user to log out of the system, clearing session information.
* **Key Elements**:
  + **Logout Button**: Clears st.session\_state['user'] when clicked, effectively logging out the user.
  + **Feedback**: Displays a "Logged out successfully" message after logout

**9. Output**

The **Transport Management System (TMS)** provides a seamless and user-friendly interface, delivering dynamic outputs based on user actions. Here’s an overview of the outputs generated by the system:



**Navigation Menu**



**9.1 User Outputs**

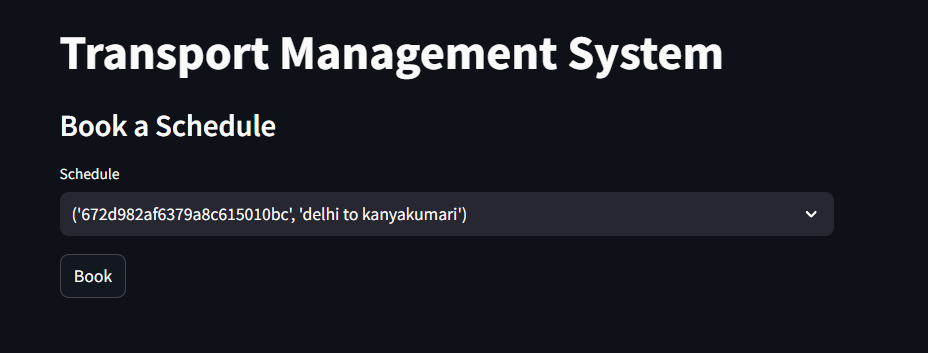
1. **Registration Success**:
   * When a user successfully registers, a confirmation message is displayed:  
     "Registration successful!"
   * If the email already exists, the system prompts:  
     "Email already exists."



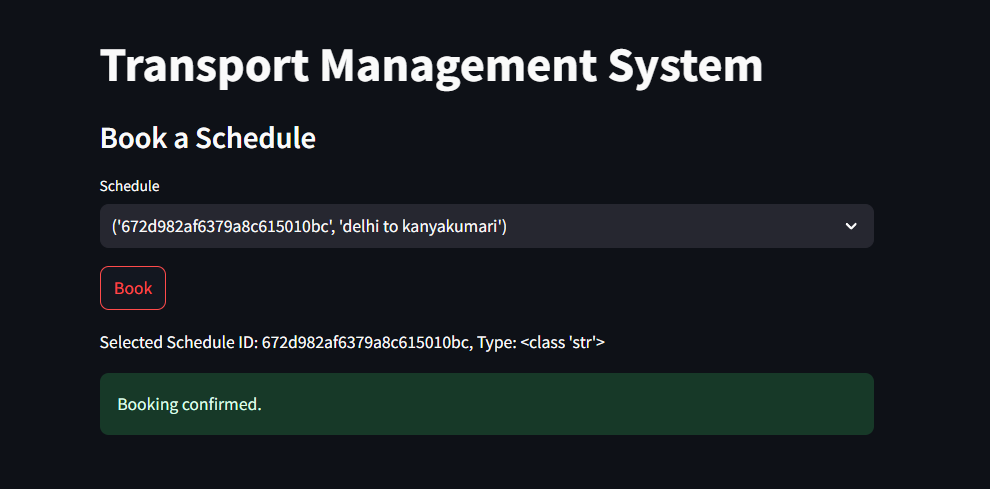
1. **Login Status**:
   * A successful login displays a personalized message:  
     "Logged in as [User Name]."
   * For invalid credentials, the system returns:  
     "Invalid email or password."



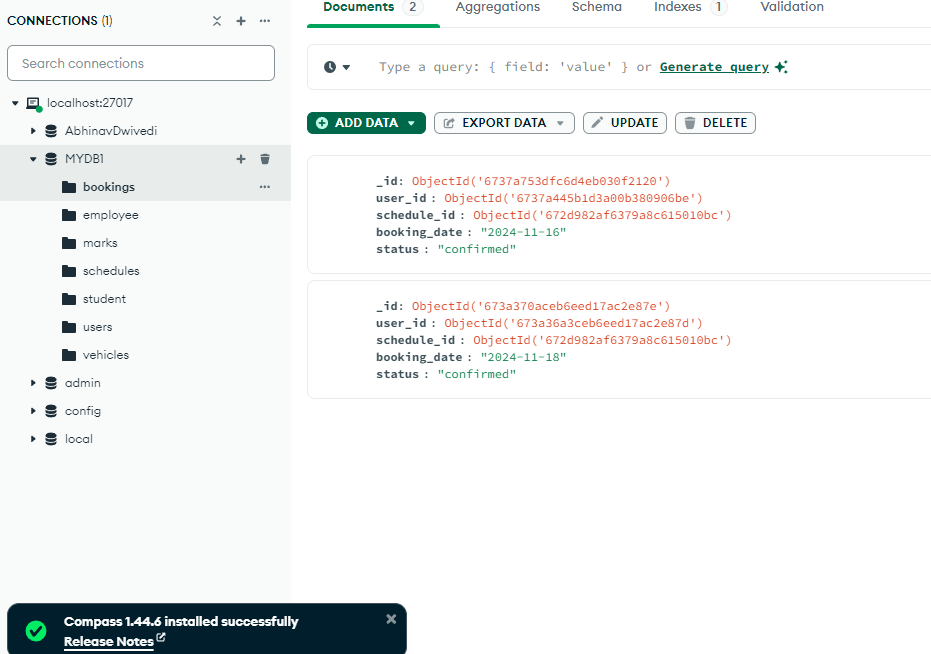
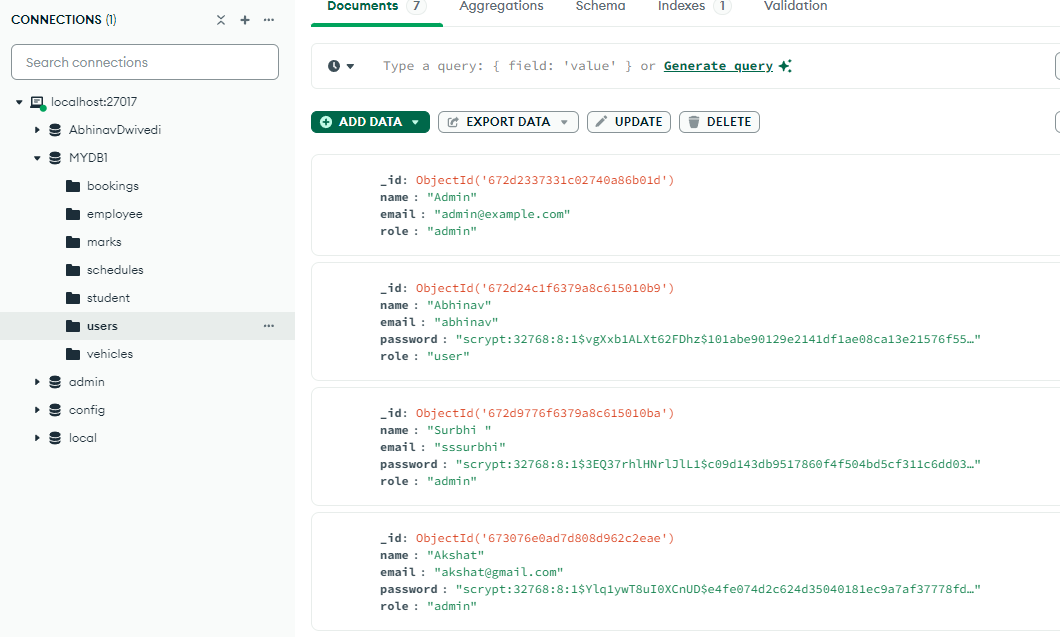
1. **Available Schedules**:
   * Users can view a list of available schedules, showing the route, departure time, arrival time, and vehicle details.

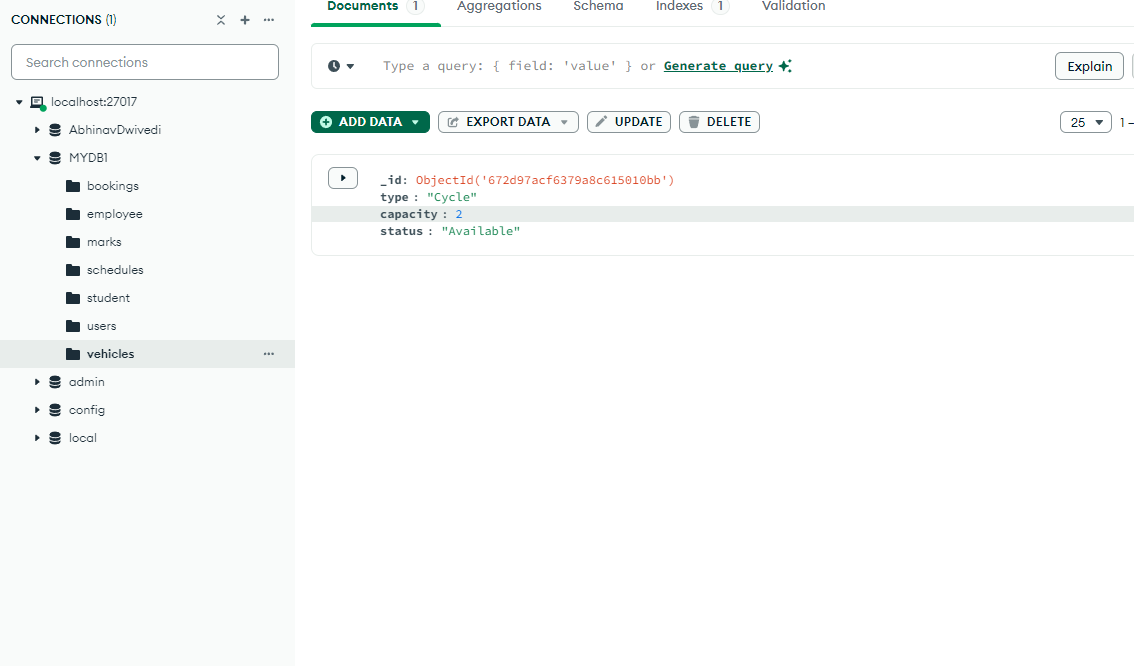


1. **Booking Confirmation**:
   * After booking a schedule, users receive the message:  
     "Booking confirmed."



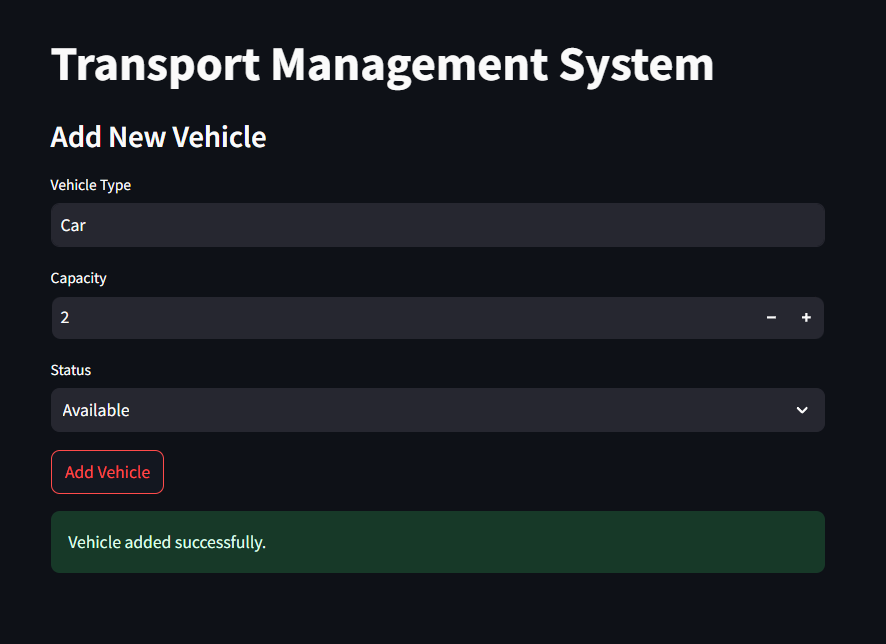
* + Bookings are securely recorded in the database and visible for reference in the admin or user logs.



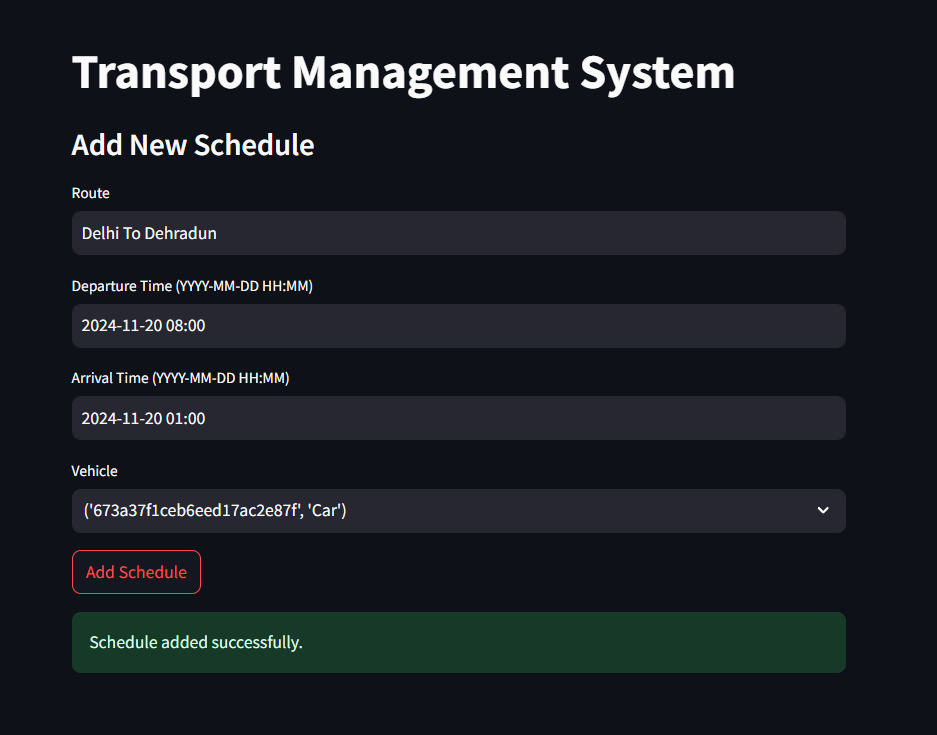


**9.2 Admin Outputs**

1. **Vehicle Management**:
   * When a new vehicle is added, the system confirms:  
     "Vehicle added successfully."



1. **Schedule Management**:
   * Upon creating a new schedule, the admin receives feedback:  
     "Schedule added successfully."



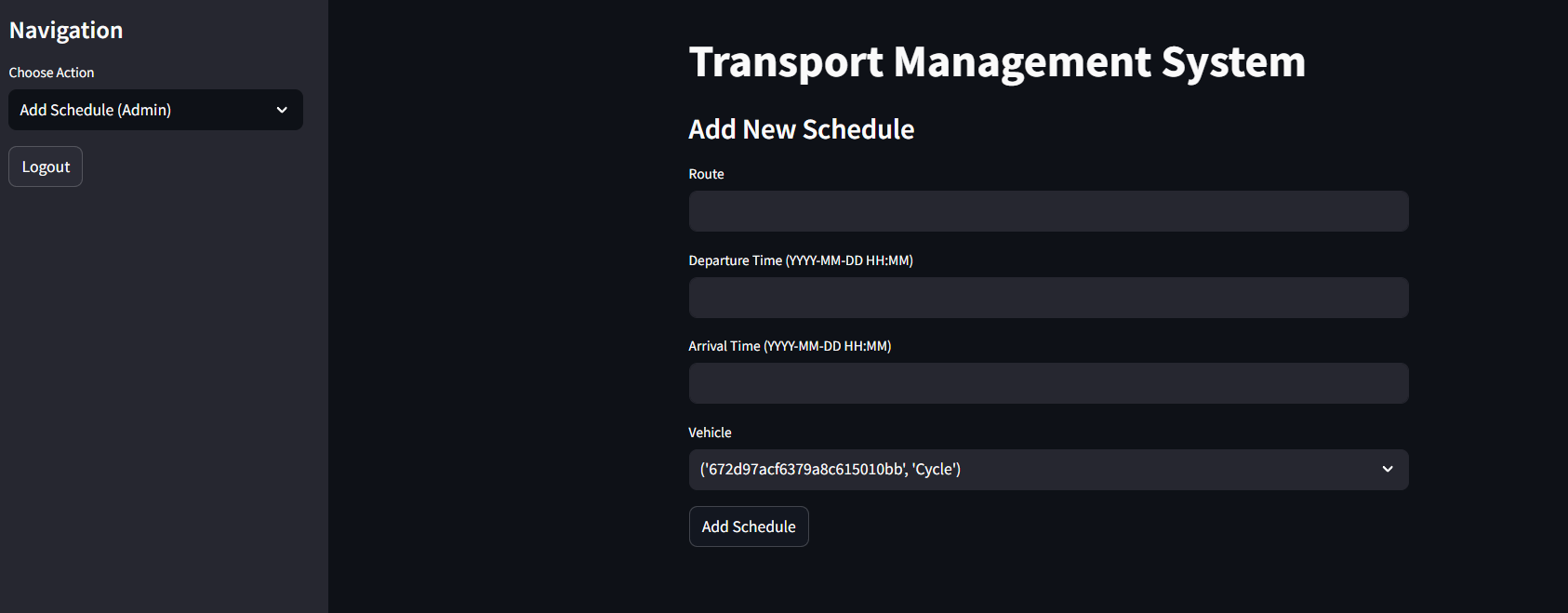
**9.3 Visual Interface Outputs**

* **Streamlit Interface**:
  + The **Home Page** displays a welcome message tailored to the user's login status.
  + Admins see options for adding vehicles and schedules, while users can view and book schedules.

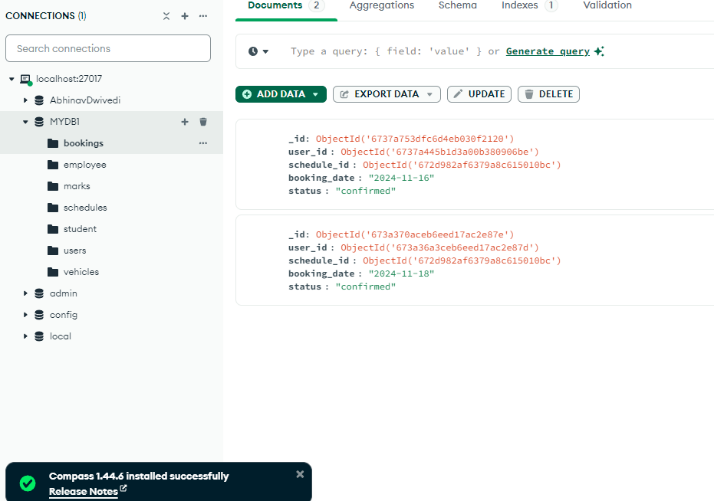


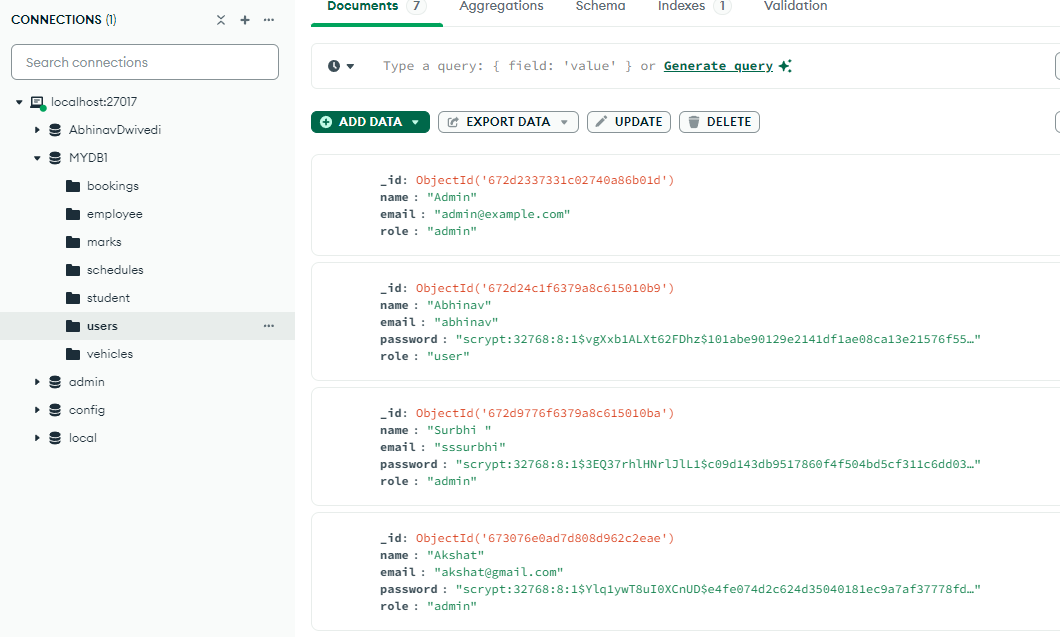
* **Interactive Dropdowns and Tables**:
  + Admins and users interact with dropdown menus (e.g., selecting vehicles or schedules) and see real-time data tables for available vehicles, schedules, or bookings.





**9.4 Database Outputs**

1. **Dynamic Data Updates**:
   * Every user action (e.g., booking a schedule, adding a vehicle) updates the MongoDB database in real-time.
   * The system retrieves and displays updated data, ensuring users and admins have the most accurate information.



1. **Error Handling**:
   * Invalid inputs or actions prompt helpful error messages, such as:  
     "Invalid schedule ID. Please select a valid schedule."

The system outputs are designed to ensure clarity and provide instant feedback, making the platform intuitive and efficient for both users and administrators.

**9. Conclusion**

The Transport Management System makes it easy to manage transport operations by bringing everything online. It helps users register, book schedules, and stay updated while allowing admins to manage vehicles and schedules effortlessly. The system is secure, reliable, and built to handle growth, making transport management smoother and more efficient for everyone involved.