Abraham Abdulkarim, Ali Almuthafar, Alexis Whisnant
CIS 476: Software Architecture and Design Patterns

DriveShare Project Report

1. Introduction

Project Title: DriveShare – Peer-to-Peer Car Rental Platform

Overview:
DriveShare is a local application designed to connect car owners with individuals looking for short-term car rentals. Inspired by Turo.com, the platform facilitates vehicle listing, searching, booking, messaging, and payment simulation, while ensuring secure user authentication and password recovery.

Objective:
The project aims to provide a functional prototype that demonstrates the application of several design patterns in a real-world scenario. It meets the following requirements:

- User Registration and Authentication (with three security questions)
- Car Listing and Management
- Search and Booking
- Messaging and Communication
- Payment Processing Simulation
- Rental History and Reviews

---

2. Functional Requirements

User Registration and Authentication

- **Registration:** Users register with an email, password, full name, and answer three security questions.
- **User Roles:** There are two roles:

    - **Host (Owner):** Can list and update vehicles.
    - **Guest (Renter):** Can only book vehicles and leave reviews.
- **Authentication:** Users must log in to access their respective features.
- **Password Recovery:** Implemented using the Chain of Responsibility pattern (with three security questions).

Car Listing and Management

- **Listing Vehicles:** Hosts (owners) can list predefined vehicles by selecting from a list.
- **Management:** Hosts can update the listing (e.g., price, availability).
- **Availability:** Each listed vehicle has a set of upcoming dates (generated dynamically).

Search and Booking

- **Search:** Renters can search for available vehicles by filtering based on location (e.g., "Romulus, Michigan").
- **Booking:** Renters book a vehicle for a specified number of days, subject to availability.
- **Double Booking Prevention:** The system removes booked dates from a vehicle’s availability to prevent overlapping bookings.

Messaging and Communication

- **Notifications:** The system sends message notifications when:
    - A booking is confirmed (to both the renter and the vehicle owner).
    - Payment is processed.
    - Other important booking events occur.

Payment

- **Payment Simulation:** A Payment Proxy simulates payment transactions by deducting funds from the renter's balance and adding them to the owner's balance. Actual payment processing is not implemented; instead, it is simulated and confirmed via console messages.

Rental History and Reviews (Optional)

- **Rental History:** Both hosts and guests can view their rental history.
- **Reviews:** Guests can leave reviews for hosts. Hosts can view reviews left for them.

---

3. Implementation and Design Patterns

The application is designed using multiple design patterns. Below are the patterns used and their roles in the system:

3.1 Singleton Pattern

- **Purpose:** Ensure that only one instance of the session manager exists.
- **Implementation:**
  - **Class:** `SessionManager` (in `app/patterns/singleton.py`)
  - **Usage:** Manages the currently logged-in user across the application.

3.2 Observer Pattern

- **Purpose:** Notify users about booking confirmations.
- **Implementation:**
  - **Classes:** `BookingObserver` and `MessageService` (in `app/services/booking_service.py`)

    - **Usage:** Observers are registered to receive updates when a booking is confirmed or checkout is completed.

3.3 Mediator Pattern

- **Purpose:** Centralize communication between different UI components (or services) to decouple their interactions.
- **Implementation:**
    - **Class:** `MainMenuMediator` (in your main file, e.g., `Driveshare.py`)
    - **Usage:** Coordinates registration, login, vehicle listing, booking, and other UI actions based on user roles.

3.4 Builder Pattern

- **Purpose:** Provide flexibility in constructing complex `Car` objects.
- **Implementation:**
    - **Classes:** `CarBuilder` and `CarDirector` (in `app/patterns/builder.py`)
    - **Usage:** Simplifies the creation of car listings by chaining method calls to set attributes (e.g., model, year, availability).

3.5 Proxy Pattern

- **Purpose:** Securely simulate communication with a payment system.
- **Implementation:**
    - **Class:** `PaymentProxy` (in `app/services/booking_service.py`)
    - **Usage:** Processes payment transactions by updating the balances of the renter and the owner, while sending notifications.

3.6 Chain of Responsibility Pattern

- **Purpose:** Create a secure process for password recovery using multiple security questions.
- **Implementation:**
  - **Classes:** `RecoveryHandler` and `SecurityQuestionHandler` (in `app/services/auth_service.py`)
  - **Usage:** Each security question is processed in sequence until the user successfully verifies their identity or fails the chain.

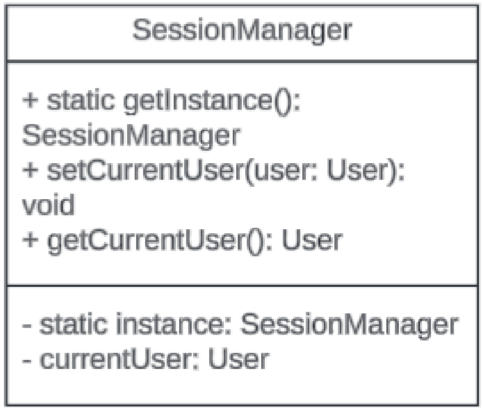
---

4. Class Diagrams and Design Mapping

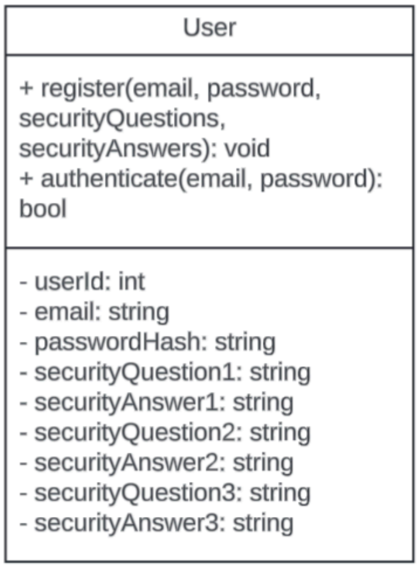
Example Class Diagram (Text Description)

While a visual UML diagram is ideal, here is a textual mapping of key classes:

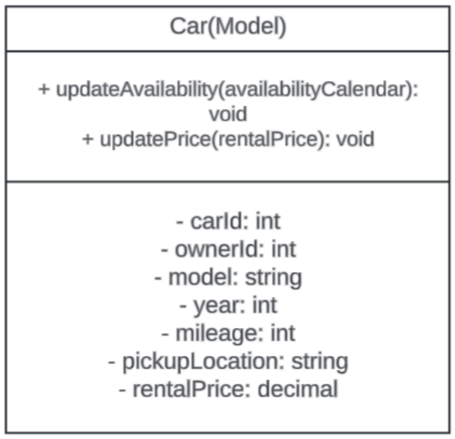
- **SessionManager (Singleton Pattern)**
  - Ensures a single active instance for managing the current user.


SessionManager
+ static getInstance(): SessionManager
+ setCurrentUser(user: User): void
+ getCurrentUser(): User
- static instance: SessionManager
- currentUser: User

- **User (Model)**
  - Attributes: email, password_hash, name, security_answers, role, balance, rental_history, reviews.

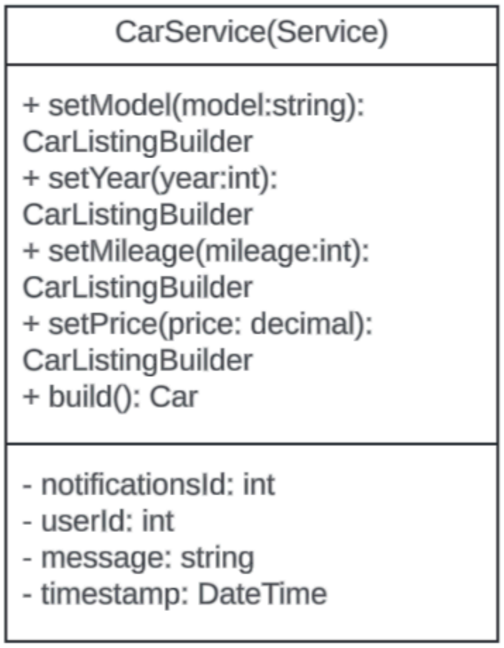

User
+ register(email, password, securityQuestions, securityAnswers): void
+ authenticate(email, password): bool
- userId: int
- email: string
- passwordHash: string
- securityQuestion1: string
- securityAnswer1: string
- securityQuestion2: string
- securityAnswer2: string
- securityQuestion3: string
- securityAnswer3: string

- **Car (Model)**
    - Attributes: owner, model, year, mileage, price_per_day, location, availability.



- **CarBuilder / CarDirector (Builder Pattern)**
    - Used to create `Car` objects with a fluent interface.

- **AuthService (Service)**
    - Handles registration, login, logout, and password recovery (using Chain of Responsibility).
- **CarService (Service)**
    - Manages vehicle listings (adding, updating, and listing cars).

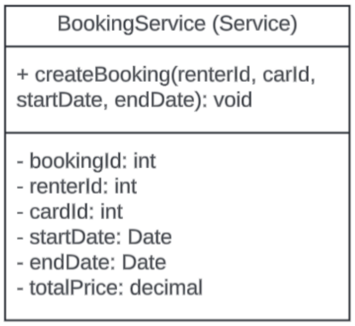


CarService(Service)

+ setModel(model:string): CarListingBuilder
+ setYear(year:int): CarListingBuilder
+ setMileage(mileage:int): CarListingBuilder
+ setPrice(price: decimal): CarListingBuilder
+ build(): Car

- notificationsId: int
- userId: int
- message: string
- timestamp: DateTime

- **BookingService (Service)**
    - Handles booking, checkout, and payment (using Proxy and Observer patterns).

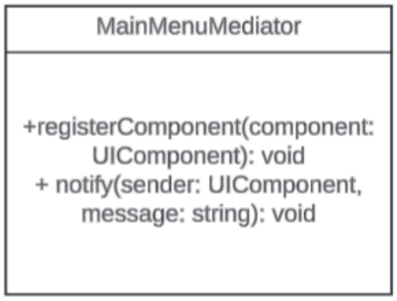


BookingService (Service)

+ createBooking(renterId, carId, startDate, endDate): void

- bookingId: int
- renterId: int
- cardId: int
- startDate: Date
- endDate: Date
- totalPrice: decimal

- **MainMenuMediator (Mediator Pattern)**
    - Acts as the central controller for user interactions and routes actions based on user roles.


MainMenuMediator
+registerComponent(component: UIComponent): void
+ notify(sender: UIComponent, message: string): void

---

5. Database Schema

Since this project uses in-memory storage:

- **Users:** Stored in an in-memory list within `AuthService.users`.
- **Cars:** Stored in an in-memory list within `CarService.car_list`.
- **Bookings:** Managed in a dictionary within `BookingService.active_bookings`.

If you were to implement a database, tables might include:

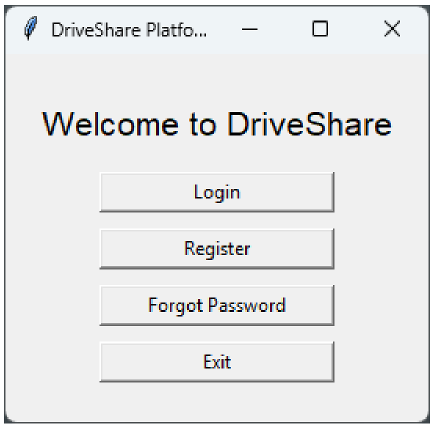
- **Users Table:** (id, email, password, name, role, balance, etc.)
- **Cars Table:** (id, owner_email, model, year, mileage, price_per_day, location, availability, etc.)
- **Bookings Table:** (id, user_email, car_id, booking_dates, total, etc.)
- **Reviews Table:** (id, reviewer_email, reviewee_email, rating, comment, date, etc.)

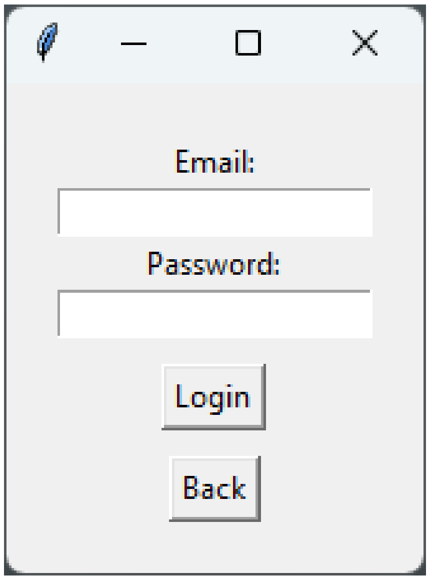
---

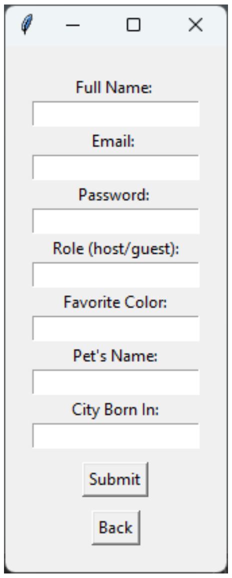
6. User Interface and Screenshots

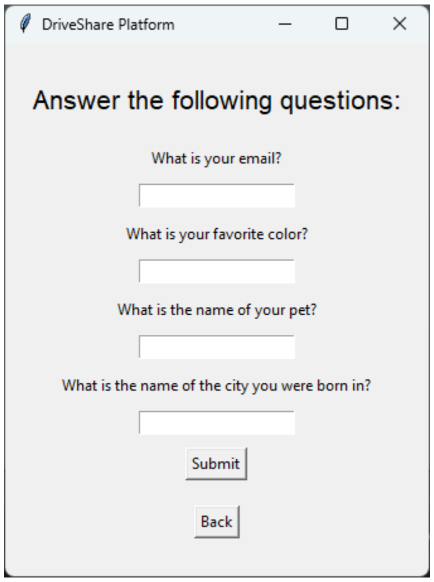
GUI-Based UI Flow:

1. **Main Menu:**
   Displays options for registration, login, password recovery, and exit



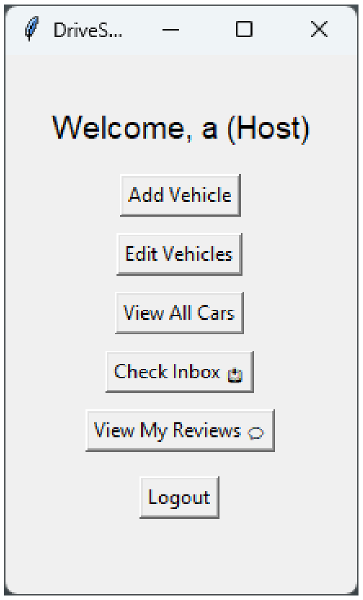


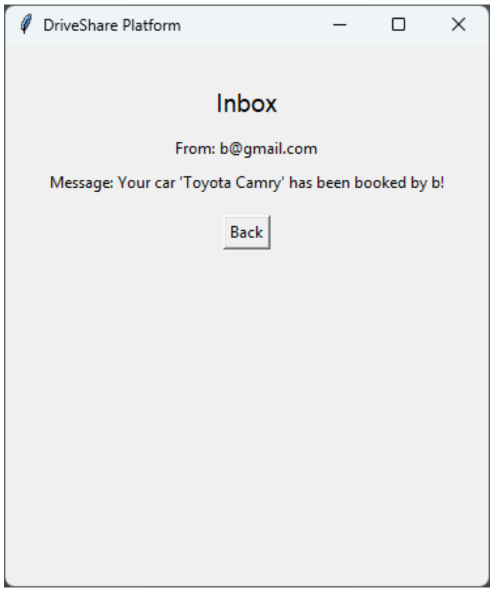



DriveShare Platform
Answer the following questions:
What is your email?
What is your favorite color?
What is the name of your pet?
What is the name of the city you were born in?
Submit
Back

2. **Host Workflow:**

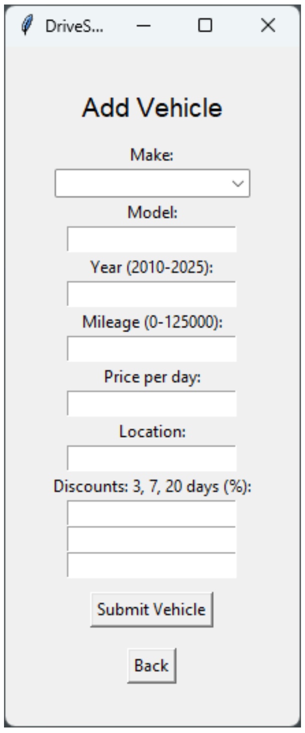
   - Hosts can list a vehicle from a predefined set.
   - Hosts can update their listings and view reviews left by guests.

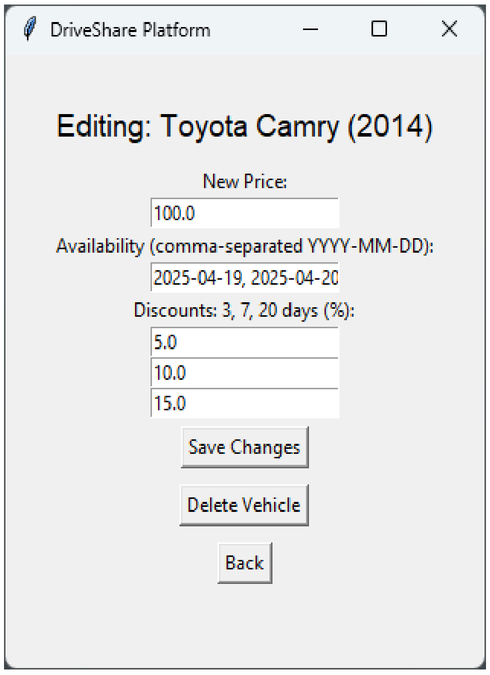




From: b@gmail.com

Message: Your car 'Toyota Camry' has been booked by b!

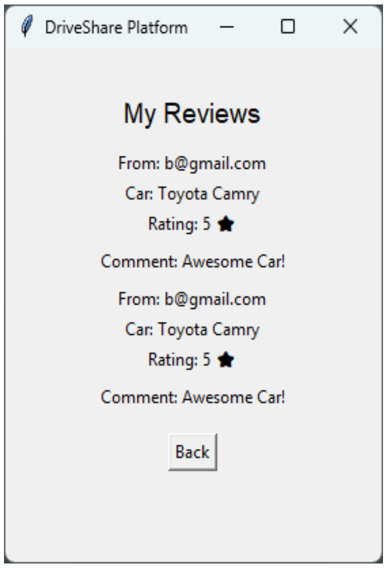




New Price: 100.0

Availability (comma-separated YYYY-MM-DD): 2025-04-19, 2025-04-20

Discounts: 3, 7, 20 days (%): 5.0 10.0 15.0



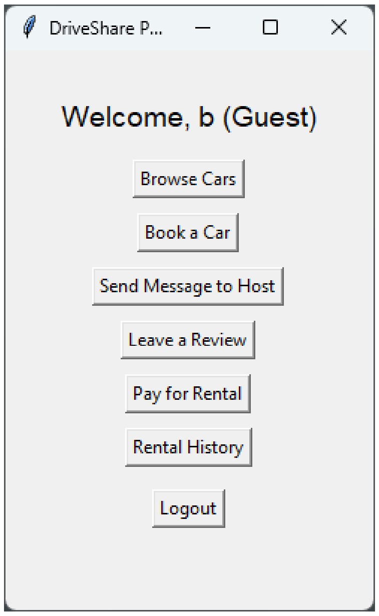
From: b@gmail.com
Car: Toyota Camry
Rating: 5 ★
Comment: Awesome Car!

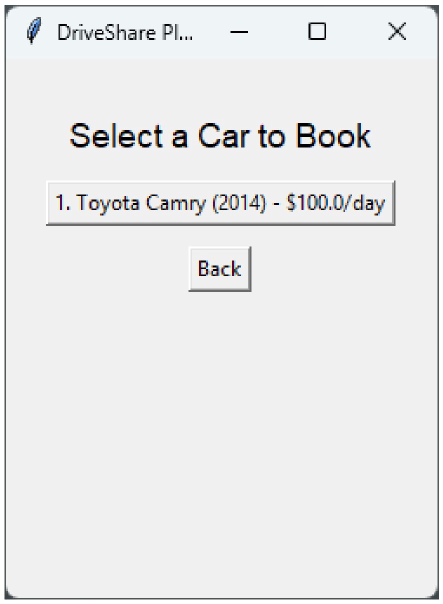
From: b@gmail.com
Car: Toyota Camry
Rating: 5 ★
Comment: Awesome Car!

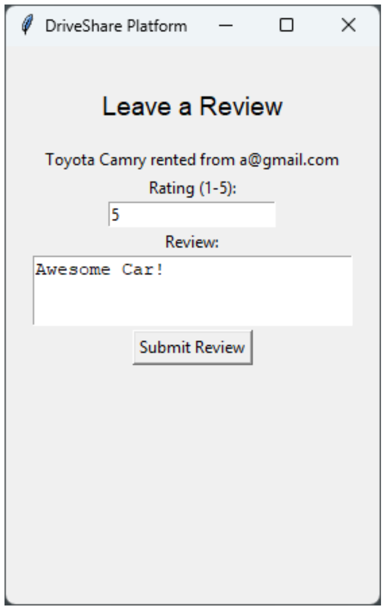
3. **Guest Workflow:**

    ◦ Guests can search for available vehicles (e.g., in Romulus, Michigan).

    ◦ Guests can book a car, proceed to checkout, view their rental history, and leave reviews for hosts.



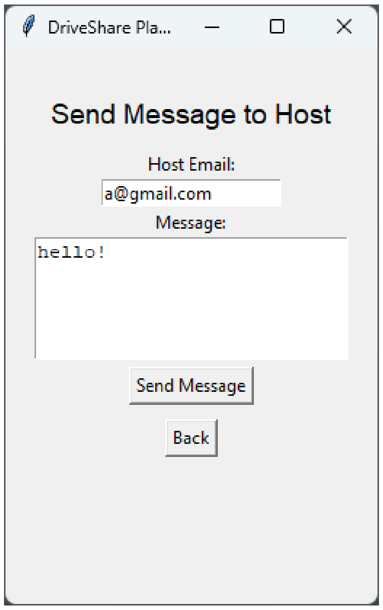




Toyota Camry rented from a@gmail.com

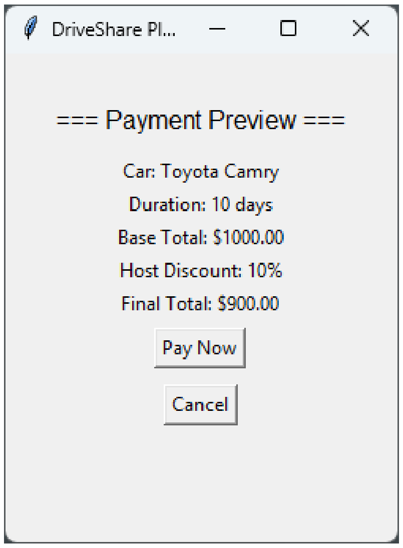
Rating (1-5): 5

Review: Awesome Car!



Host Email: a@gmail.com

Message: hello!



=== Payment Preview ===

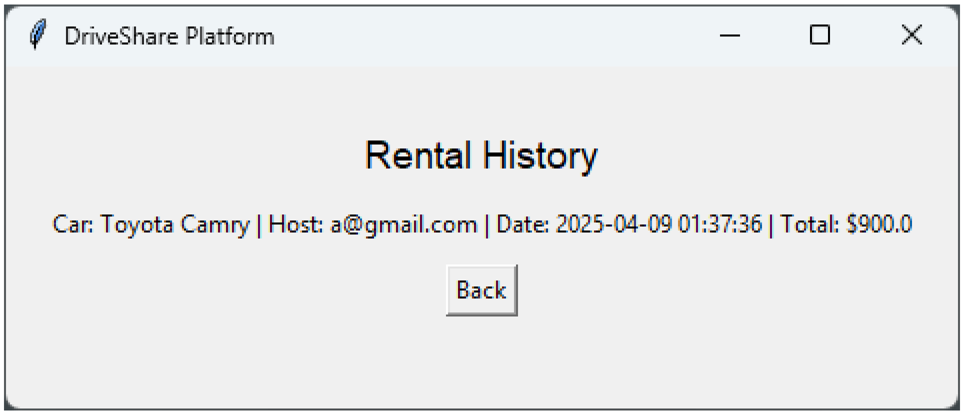
Car: Toyota Camry

Duration: 10 days

Base Total: $1000.00

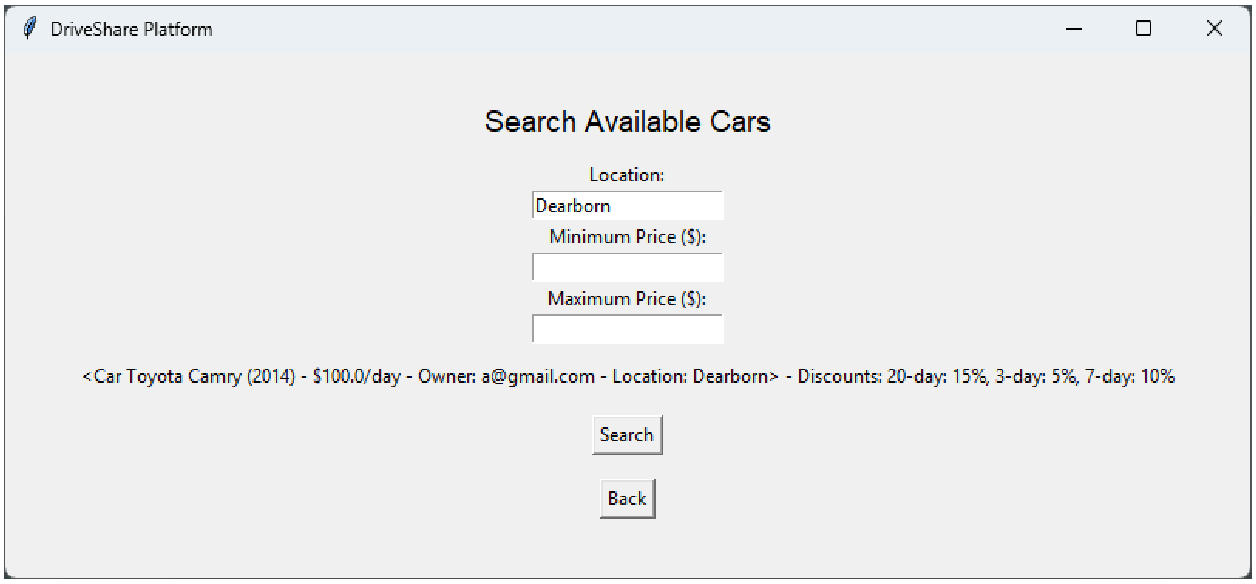
Host Discount: 10%

Final Total: $900.00



Rental History

Car: Toyota Camry | Host: a@gmail.com | Date: 2025-04-09 01:37:36 | Total: $900.0



Search Available Cars

Location: Dearborn

Minimum Price ($):

Maximum Price ($):

<Car Toyota Camry (2014) - $100.0/day - Owner: a@gmail.com - Location: Dearborn> - Discounts: 20-day: 15%, 3-day: 5%, 7-day: 10%

7. References

- **Design Patterns:**
  - Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- **Turo.com:**
  Used as inspiration for the DriveShare platform.
- **Python Documentation:**
  For language-specific implementation details.
- **Visual Studio 2022 Documentation:**
  For project setup and management.

8. Conclusion

The DriveShare project successfully implements a local peer-to-peer car rental platform using several design patterns. The application supports user registration with role-based functionality, vehicle listing and management, booking and payment processing, and additional features like rental history and reviews.