DRIVE-SHARE WEBSITE

TEAM ALPHA

SOUAD OMAR

ZAYNAB MOURTADA

LEAH MIRCH

FIRAS ABUEIDA

TABLE OF *Contents*

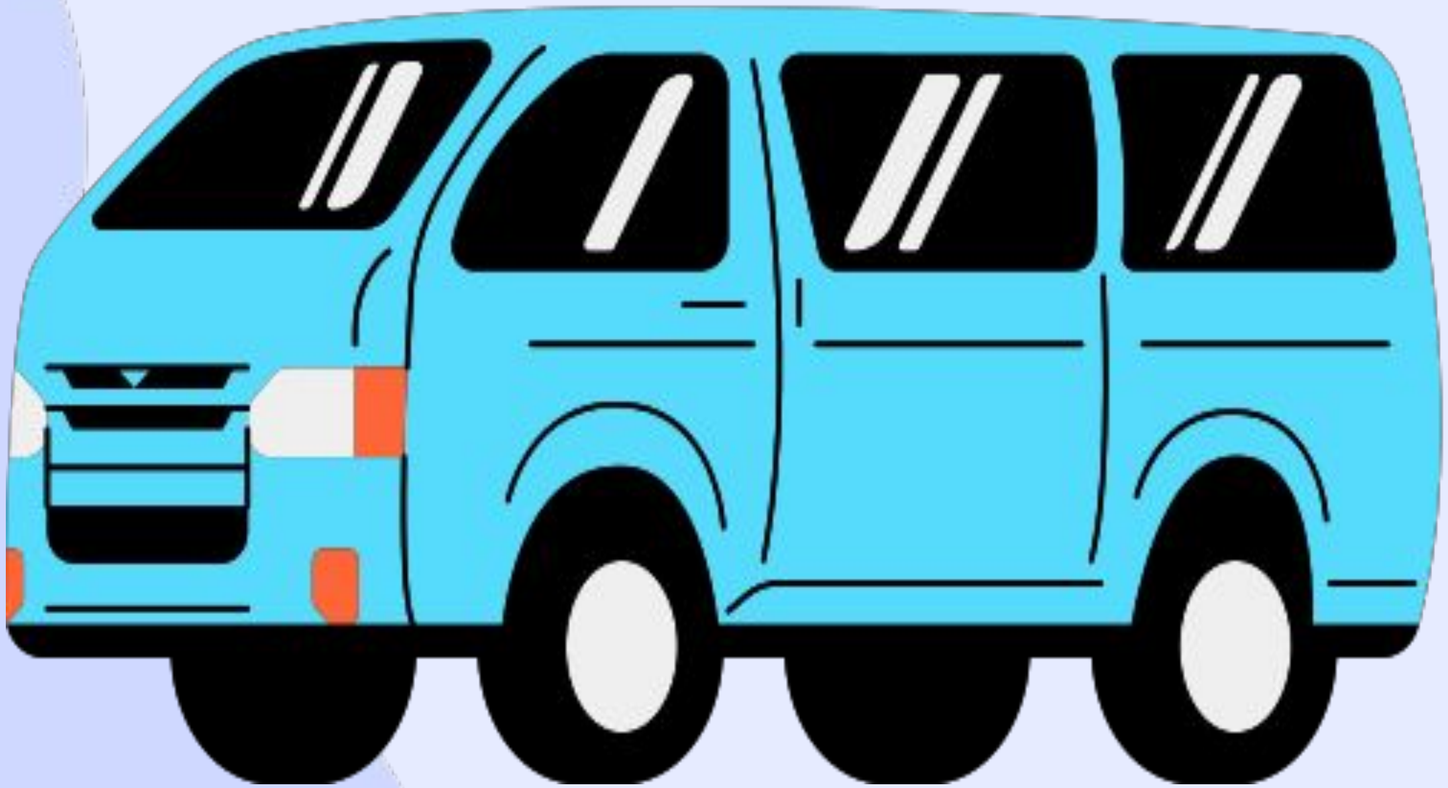
TEAM ALPHA

01. INTRODUCTION

- A peer-to-peer car rental website to connect car owners with renters.
- Designed with HTML, CSS, Javascript, Python Flask, and SQLite
- Some features include: user authentication, car listing and browsing, booking functionality, messaging, and a central dashboard

TEAM ALPHA

02. WEBSITE DEMO TIME!



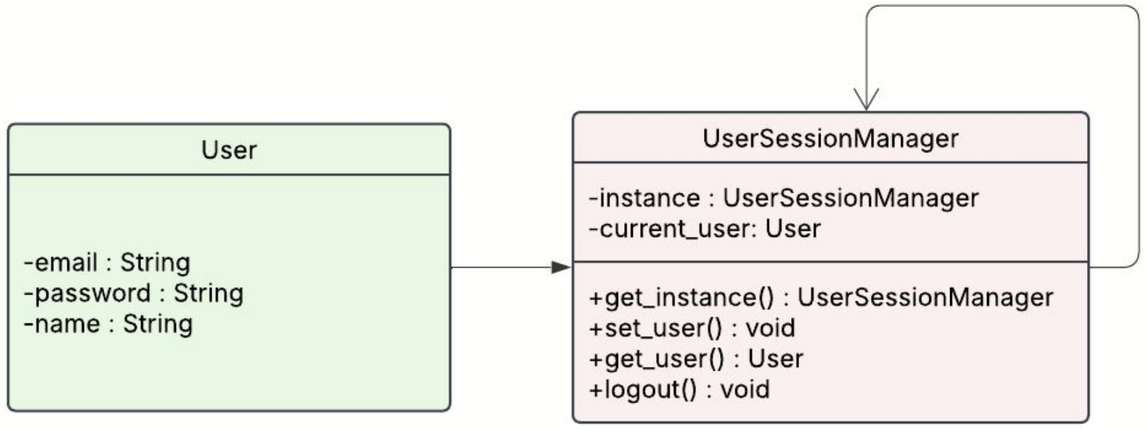
TEAM ALPHA

03. UML DIAGRAMS

TEAM ALPHA

SINGLETON FOR USER SESSION MANAGEMENT

- Singleton Pattern used via UserSessionManager to ensure a single, consistent session instance across the app.
- Manages session-related data like the currently logged-in user.
- Provides methods to set, retrieve, and clear the active user session.



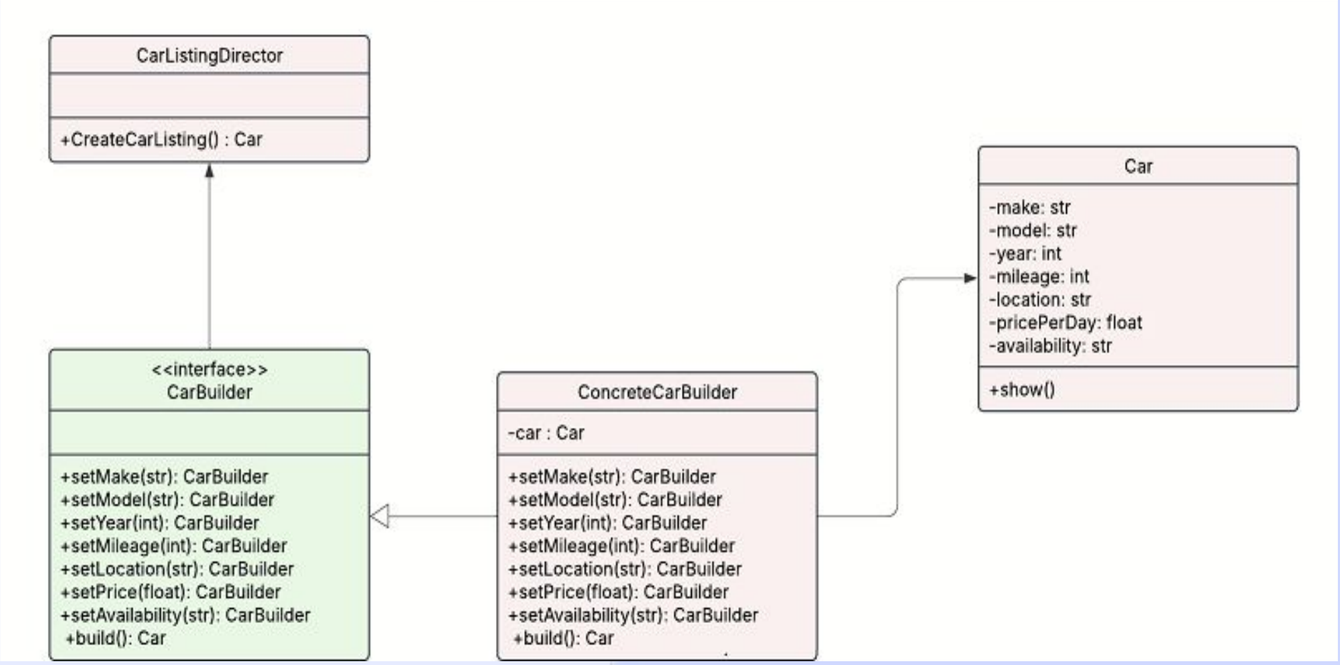
TEAM ALPHA

SINGLETON FOR USER SESSION MANAGEMENT

```python
# Singleton Pattern for Session
class UserSession:
    _instance = None

    def __init__(self):
        if UserSession._instance is not None:
            raise Exception("Singleton already exists!")
        self.user_id = None
        self.email = None
        self.role = None
        UserSession._instance = self

    @staticmethod
    def get_instance():
        if UserSession._instance is None:
            UserSession()
        return UserSession._instance

    def login(self, user_id, email, role):
        self.user_id = user_id
        self.email = email
        self.role = role

    def logout(self):
        self.user_id = None
        self.email = None
        self.role = None

    def is_authenticated(self):
        return self.user_id is not None
```

BUILDER FOR CAR CREATION

- Builder Pattern allows step-by-step creation of flexible, modular car listings.
- Avoids long constructors by setting attributes like make, model, year, price, etc., individually.
- CarBuilder defines methods to set each field, with build() returning the complete Car.
- ConcreteCarBuilder constructs the car instance internally.



TEAM ALPHA

BUILDER FOR CAR CREATION

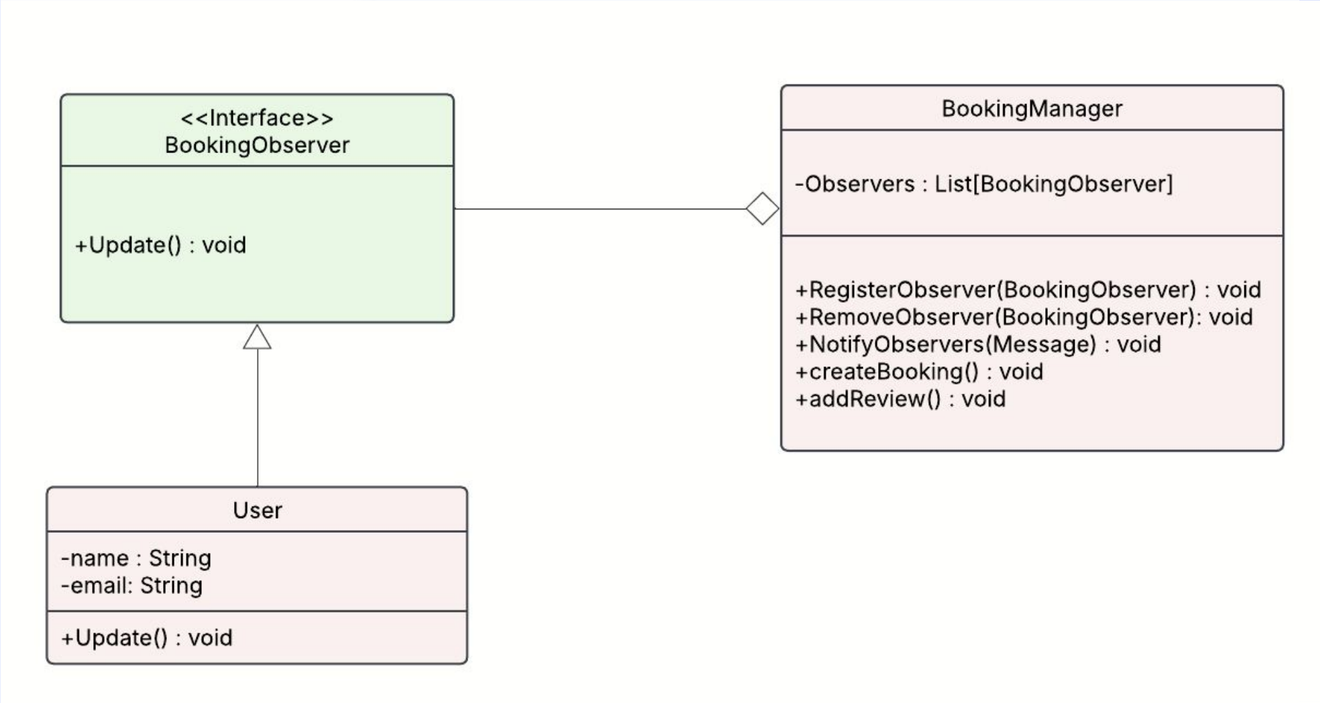
```python
# Builder Pattern for Car Creation
class Car:
    def __init__(self, owner_id, make, model, year, mileage, color, price, location, precise_location):
        self.owner_id = owner_id
        self.make = make
        self.model = model
        self.year = year
        self.mileage = mileage
        self.color = color
        self.price = price
        self.location = location
        self.precise_location = precise_location
```

```python
class CarBuilder:
    def __init__(self):
        self._car_data = {}

    def set_owner_id(self, owner_id):
        self._car_data["owner_id"] = owner_id
        return self

    def set_make(self, make):
        self._car_data["make"] = make
        return self

    def set_model(self, model):
        self._car_data["model"] = model
        return self

    def set_year(self, year):
        self._car_data["year"] = year
        return self

    def set_mileage(self, mileage):
        self._car_data["mileage"] = mileage
        return self

    def set_color(self, color):
        self._car_data["color"] = color
        return self

    def set_price(self, price):
        self._car_data["price"] = price
        return self

    def set_location(self, location):
        self._car_data["location"] = location
        return self

    def set_precise_location(self, precise_location):
        self._car_data["precise_location"] = precise_location
        return self

    def build(self):
        return Car(**self._car_data)
```

TEAM ALPHA

OBSERVER FOR NOTIFICATIONS

- BookingManager acts as the subject, managing a list of observers.
- Observers are notified via NotifyObservers() during actions like createBooking() or addReview().
- Promotes a clean separation between booking logic and notifications.

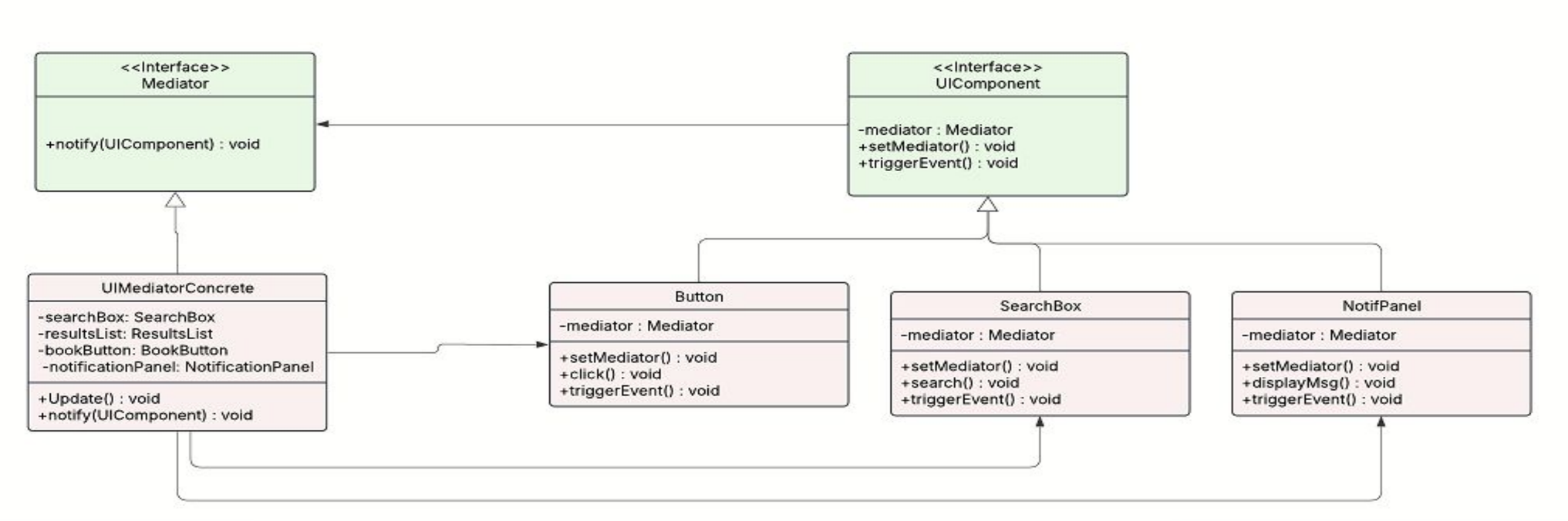


TEAM ALPHA

OBSERVER FOR NOTIFICATIONS

```python
# Observer Pattern for Booking Notifications

class BookingSubject:
    def __init__(self):
        self._observers = []

    def attach(self, observer):
        if observer not in self._observers:
            self._observers.append(observer)

    def detach(self, observer):
        self._observers.remove(observer)

    def notify(self, message, user_id):
        for observer in self._observers:
            observer.update(message, user_id)

class Observer:
    def update(self, message, user_id):
        raise NotImplementedError("Subclasses must override this method")

class InAppNotification(Observer):
    def update(self, message, user_id):
        with sqlite3.connect("database.db") as conn:
            cursor = conn.cursor()
            cursor.execute('''
                INSERT INTO notifications (user_id, message, timestamp, is_read)
                VALUES (?, ?, datetime('now'), 0)
            ''', (user_id, message))
            conn.commit()
```

MEDIATOR FOR UI COMPONENTS



- UI elements interact through a central mediator instead of directly with each other.
- UIMediatorConcrete coordinates actions between components like SearchBox, Button, and NotificationPanel.
- All components implement a shared UIComponent interface with setMediator() and triggerEvent().
- Makes the UI easier to maintain and extend without introducing tight coupling.

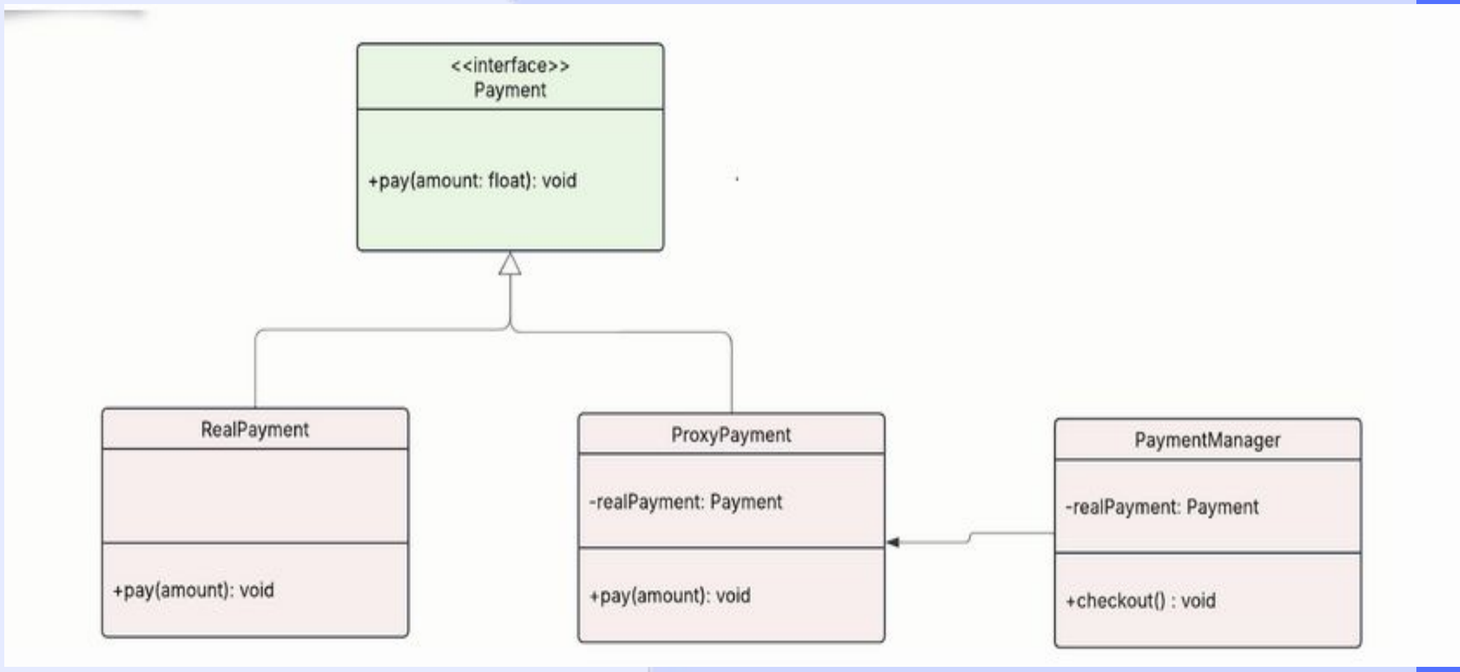
TEAM ALPHA

MEDIATOR FOR UI COMPONENTS

```
import Mediator from './mediator';

class UIMediatorConcrete extends Mediator{
    constructor(){
        super();
        this.components = {};
    }

    registerComponent(name, component){
        this.components[name] = component;
        component.setMediator(this);
    }

    notify(sender, event, data){
        if ( event === "loginSuccess"){
            this.components['NotificationPanel'].displayMsg("Welcome back!");
            this.components['Dashboard'].update(data);
            window.location.href = '/dashboard';
        } else if (event === "loginFailure") {
            this.components['NotificationPanel'].displayMsg("Login failed! Please check your credentials.");
        } else if (event === "logout"){
            this.components['NotificationPanel'].displayMsg("Logged out successfully.");
            window.location.href = "/login";
        } else if (event === 'registrationSuccess') {
            this.components['NotifcationPanel'].displayMsg("Registration successful! Please log in.");
            window.location.hred = "/login";
        } else if (event === "registrationFailure"){
            this.components["NotificationPanel"].displayMsg("Registration failed! Email already registered.");
        }
    }
}

const mediator = new UIMediatorConcrete();
export default mediator;
```

PROXY FOR PAYMENT PROCESSING

- Proxy Pattern adds a secure layer between DriveShare and the real payment system.
- ProxyPayment controls access to RealPayment, allowing for validation, logging, or simulation.
- Both implement the Payment interface with a common pay() method.
- PaymentManager uses only the interface, calling checkout() to handle transactions.



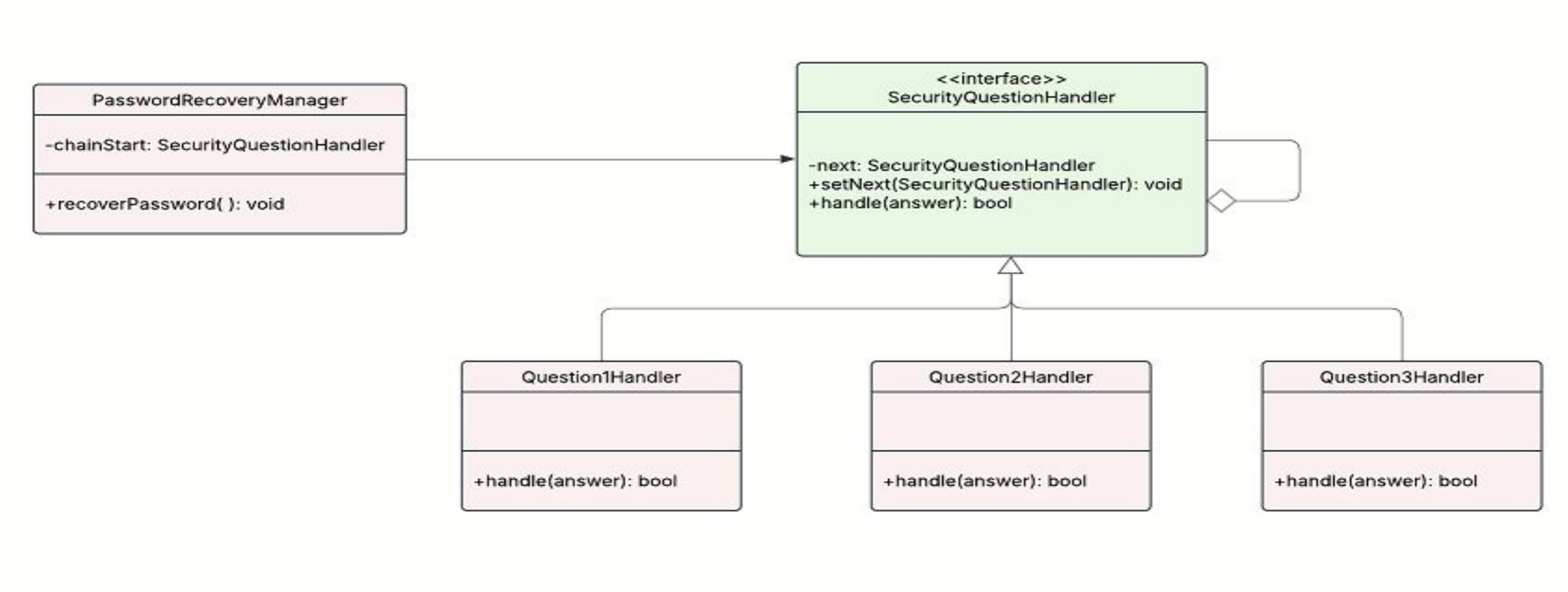
TEAM ALPHA

PROXY FOR PAYMENT PROCESSING

```python
class RealPaymentProcessor:
    def process_payment(self, payer_id, receiver_id, amount):
        # Logic to update balances in the database
        import sqlite3
        with sqlite3.connect("database.db") as conn:
            cursor = conn.cursor()

            # Deduct from renter
            cursor.execute("UPDATE users SET balance = balance - ? WHERE id = ?", (amount, payer_id))

            # Add to owner
            cursor.execute("UPDATE users SET balance = balance + ? WHERE id = ?", (amount, receiver_id))

            # Simulate notification
            print(f"NotifyObserver: ${amount} transferred from User {payer_id} to User {receiver_id}.")


class PaymentProxy:
    def __init__(self):
        self.processor = RealPaymentProcessor()

    def pay(self, payer_id, receiver_id, amount):
        print(f"Proxy: Initiating payment of ${amount} from {payer_id} to {receiver_id}")
        self.processor.process_payment(payer_id, receiver_id, amount)
```

TEAM ALPHA

CHAIN OF RESPONSIBILITY FOR PASSWORD RECOVERY



- Each security question is a handler in the chain—if answered correctly, the chain continues.
- Handlers implement SecurityQuestionHandler with handle() and setNext() methods.
- Question1Handler, Question2Handler, Question3Handler each verify specific questions.
- PasswordRecoveryManager triggers the chain via recoverPassword().

TEAM ALPHA

CHAIN OF RESPONSIBILITY FOR PASSWORD RECOVERY

```python
class SecurityQuestionHandler:
    def __init__(self):
        self.next = None

    def set_next(self, handler):
        self.next = handler
        return handler

    def handle(self, user_input, expected_answer):
        raise NotImplementedError("Must override handle method")


class Question1Handler(SecurityQuestionHandler):
    def handle(self, user_input, expected_answer):
        if user_input != expected_answer:
            return False
        if self.next:
            return self.next.handle_chain()
        return True

    def handle_chain(self):
        return self.next.handle_chain() if self.next else True


class Question2Handler(SecurityQuestionHandler):
    def handle(self, user_input, expected_answer):
        if user_input != expected_answer:
            return False
        if self.next:
            return self.next.handle_chain()
        return True

    def handle_chain(self):
        return self.next.handle_chain() if self.next else True
```

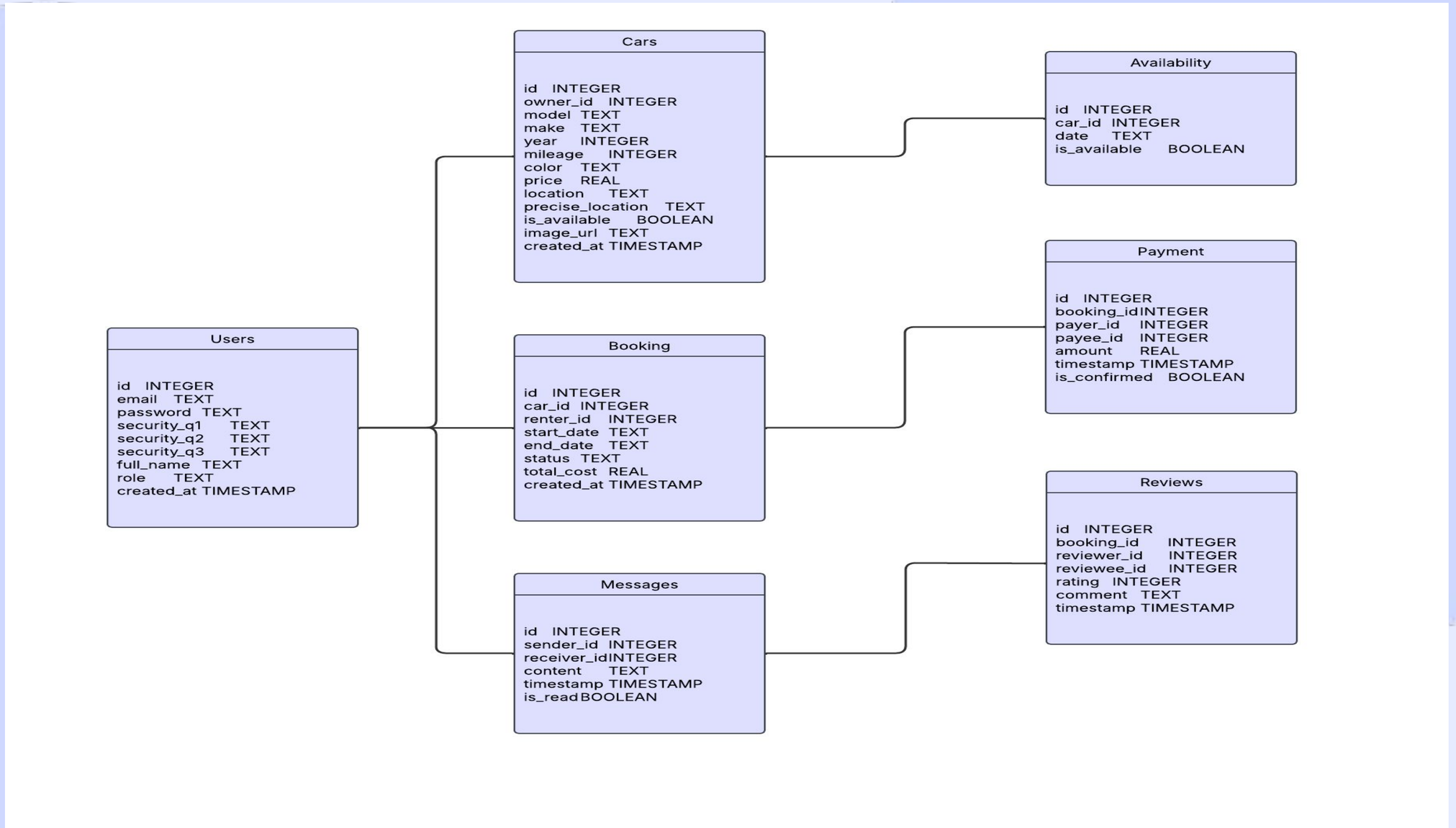
```python
class Question3Handler(SecurityQuestionHandler):
    def handle(self, user_input, expected_answer):
        return user_input == expected_answer

    def handle_chain(self):
        return True  # End of chain


class PasswordRecoveryManager:
    def __init__(self):
        self.q1 = Question1Handler()
        self.q2 = Question2Handler()
        self.q3 = Question3Handler()
        self.q1.set_next(self.q2).set_next(self.q3)

    def recover_password(self, inputs, expected):
        return (self.q1.handle(inputs[0], expected[0]) and
                self.q2.handle(inputs[1], expected[1]) and
                self.q3.handle(inputs[2], expected[2]))
```

TEAM ALPHA

04. DATABASE SCHEMA

COMPANY NAME

DATABASE SCHEMA



TEAM ALPHA

05. TEAM CONTRIBUTIONS

COMPANY NAME

Team Contributions

Leah Mirch	Database, Payment, Car listing and Management, Searching and Booking, Rental History, Payment
Zaynab Mourtada	Front End Design, Messaging, Notification system
Souad Omar	Documentation/Slides/Diagrams, User Registration & Authentication, User/Owner Reviews
Firas Abueida	Documentation, Slides,Testing, Worked with Souad via DAS

TEAM ALPHA

THANK *you*

TEAM ALPHA