Online Reservation System's Requirements

#	User Authentication & Registration
1	Allow users to create an account.
2	Require a valid email address during registration.
3	Verify email address for account activation.
4	Support social media authentication.
5	Enable users to securely log in and reset passwords.

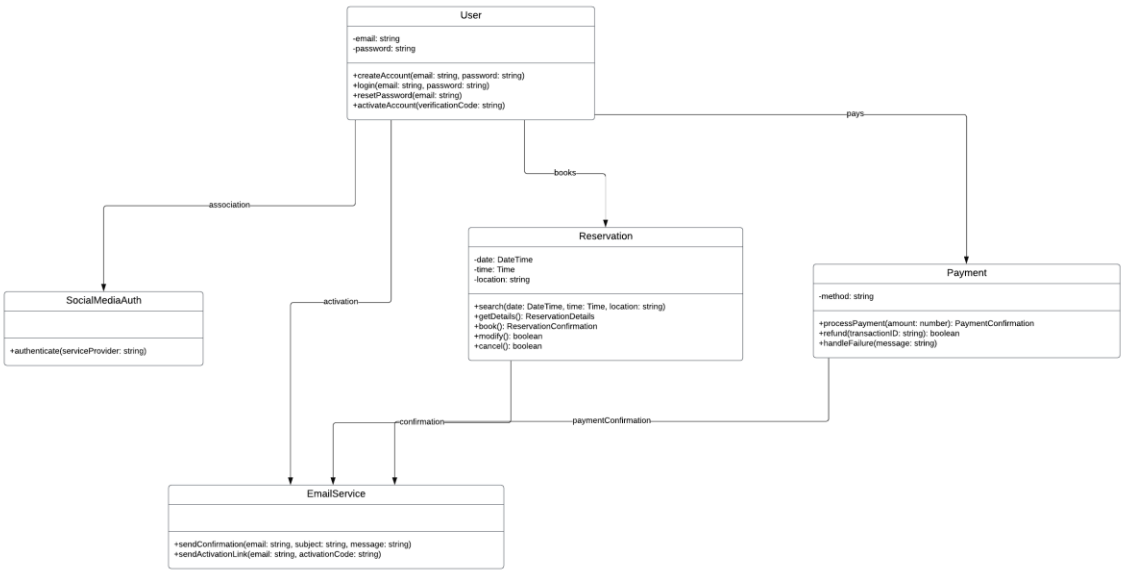
Following are some of the requirements related to user reservation and management:

#	Reservation and Management
1	Enable users to search for available reservations by date, time, and location.
2	Allow users to view detailed information about available reservations before confirmation.
3	Support booking of multiple reservations in a single session.
4	Provide confirmation of reservation bookings via email.
5	Allow modification or cancellation of existing reservations.

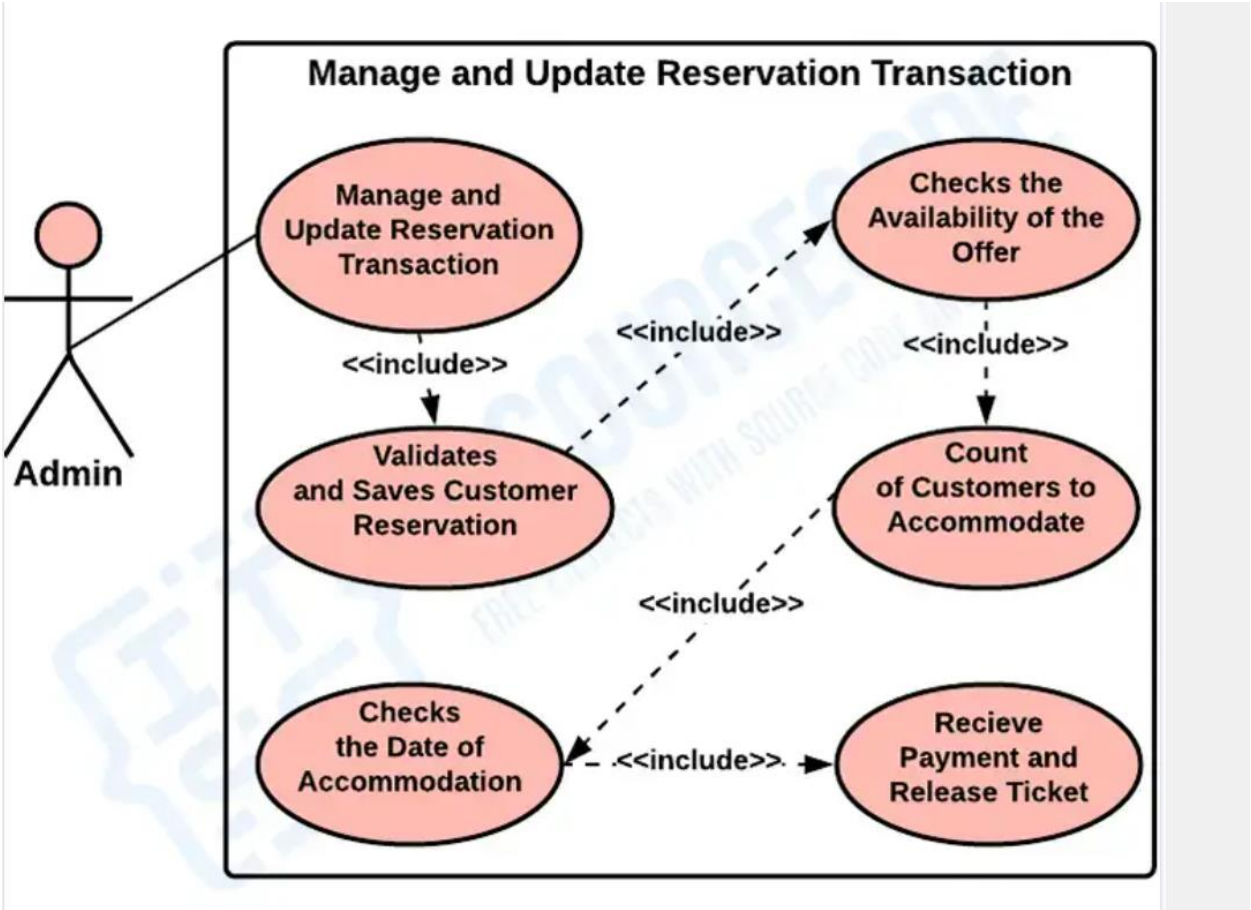
Following are some of the requirements related to user payment processing:

#	Payment Processing
1	Process payment transactions securely in real-time.
2	Support multiple payment methods such as credit/debit cards, PayPal, etc.
3	Send immediate confirmation of successful payment transactions.
4	Handle payment failures and provide clear error messages.
5	Automatically process refunds for canceled reservations.

CLASS DIAGRAM



USECASE DIAGRAM



Singleton Pattern

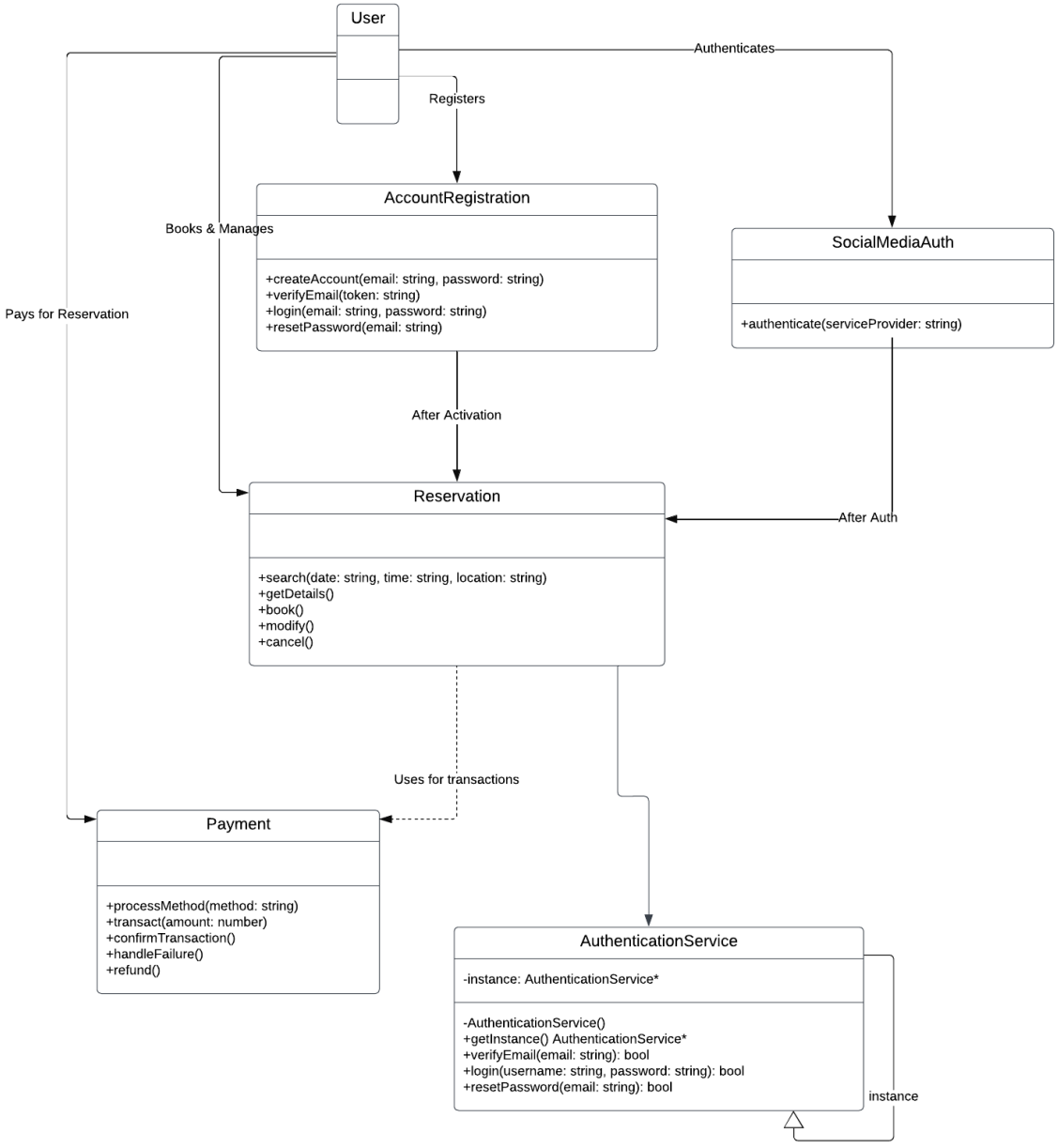
Code(Skeleton):

```
#include <iostream>

using namespace std;
```

```
class AuthenticationService
{
        private:
                static AuthenticationService* instance;
                AuthenticationService() {}

        public:
                 static AuthenticationService* getInstance()
                 {
                         if (!instance)
                         {
                         instance = new AuthenticationService();
                         }
                         return instance;
                }

                bool verifyEmail(const std::string& email)
                 {
                         // Implementation for verifying email address
                         cout << "Verifying email: " << email << endl;
                         return true;
                }

                bool login(const std::string& username, const std::string& password)
                 {
                      cout << "Logging in user: " << username << endl;
                      return true;
                  }

                bool resetPassword(const std::string& email)
                 {
                         cout << "Resetting password for email: " << email <<endl;
                          return true; // Placeholder implementation
```

```
                        }
};

// Initialize static member
AuthenticationService* AuthenticationService::instance = nullptr;

int main()
{
        AuthenticationService* authService1 = AuthenticationService::getInstance();
        AuthenticationService* authService2 = AuthenticationService::getInstance();

        cout << "Is authService1 == authService2? " << (authService1 == authService2 ? "Yes" : "No") << endl;

        authService1->verifyEmail("example@example.com");
        authService1->login("username", "password");
        authService1->resetPassword("example@example.com");

        delete authService1; // Ensure cleanup
        return 0;
}
```

Class Diagram:


User
Authenticates
Registers
AccountRegistration
+createAccount(email: string, password: string)
+verifyEmail(token: string)
+login(email: string, password: string)
+resetPassword(email: string)
Books & Manages
SocialMediaAuth
+authenticate(serviceProvider: string)
Pays for Reservation
After Activation
Reservation
+search(date: string, time: string, location: string)
+getDetails()
+book()
+modify()
+cancel()
After Auth
Uses for transactions
Payment
+processMethod(method: string)
+transact(amount: number)
+confirmTransaction()
+handleFailure()
+refund()
AuthenticationService
-instance: AuthenticationService*
-AuthenticationService()
+getInstance() AuthenticationService*
+verifyEmail(email: string): bool
+login(username: string, password: string): bool
+resetPassword(email: string): bool
instance

Abstract Factory Pattern

Code(Skeleton):

```
#include <iostream>
#include <string>
using namespace std;

class Reservation
{
	public:
		virtual void book() = 0;
};
//Concrete Classes
class HotelRoomReservation : public Reservation
{
	public:
	void book() override
	{
		cout << "Booking hotel room reservation..." << endl;
	}
};

class EventReservation : public Reservation
{
	public:
		void book() override
		{
			cout << "Booking event reservation..." << endl;
		}
};

// Reservation Factory interface
```

```
class ReservationFactory
{
	public:
		virtual Reservation* createReservation() = 0;
};

class HotelRoomReservationFactory : public ReservationFactory
{
	public:
		Reservation* createReservation() override
		{
			return new HotelRoomReservation();
		}
};

class EventReservationFactory : public ReservationFactory
{
	public:
		Reservation* createReservation() override
		{
			return new EventReservation();
		}
};

int main()
{
	ReservationFactory* hotelRoomFactory = new HotelRoomReservationFactory();
	Reservation* hotelRoomReservation = hotelRoomFactory->createReservation();
	hotelRoomReservation->book();

	ReservationFactory* eventFactory = new EventReservationFactory();
	Reservation* eventReservation = eventFactory->createReservation();
	eventReservation->book();
```

```
    delete hotelRoomFactory;
    delete hotelRoomReservation;
    delete eventFactory;
    delete eventReservation;

    return 0;
}
```

Class Diagram: