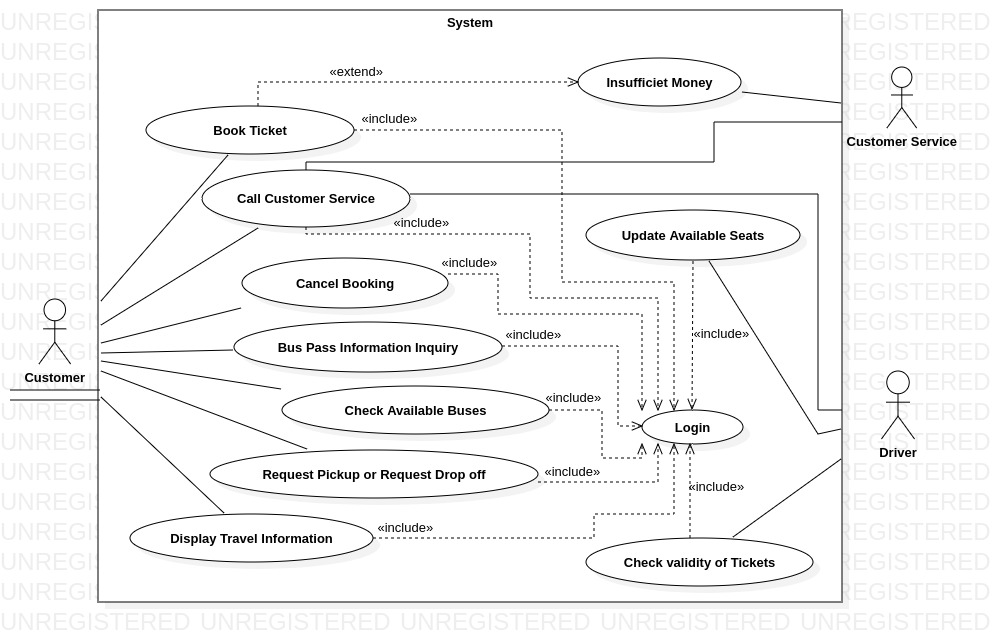
**USE CASE Diagram**



The system includes three actors:

1. **User** : Any person who wants to travel from one place to another.
2. **Driver** : Person who facilitates the ride from source to Destination.
3. **Customer Service** : Intermediary for both Driver and User for issues or queries.

**Driver**

**Use Case** :

Update Available Seats : *For every new PickUp / DropOff, Driver needs to update the Available seats for the current occupancies.*

Call Customer Service : *Allows to clarify/post any queries / complaints or enquire anything related to ride.*

Check Validity of Tickets : *Driver is responsible for the verifying the integrity of ticket by inspecting the QR-Scanner indication for the e-Ticket provided towards it.*

* Every Driver needs to Login to the system to get the ride details. And hence, each of the Use Case that is directly connected to Driver includes *Use Case – Login.*

**Customer**

**Use Case** :

Book Ticket : *Allows user to book ticket.*

Call Customer Service : *Allows to clarify/post any queries / complaints or enquire anything related to ride.*

Cancel Booking : *Allows to Cancel booking and also adds Cancellation charge based on the bus location.*

Bus Pass Information Inquiry : *Allows user to buy, use, renew Bus Pass for daily Commuters.*

Check Available Buses : *Provides the list of all buses for the given Source and Destination.*

Request PickUp / DropOff : *Allows to book ride by providing Source and Destination.*

Display Travel Information : *Provide ride details like Estimated time, estimated fare, etc.*

* Customer is provided with all the functionality of booking a ride.
* Every Customer needs to Login to the system to use its functionality and enjoy its features. And hence, each of the Use Case that is directly connected to Customer includes *Use Case – Login.*
* *Use Case – Call Customer Service* is also a Use Case for Customer Service, in that way the communication is made end-to-end between them.

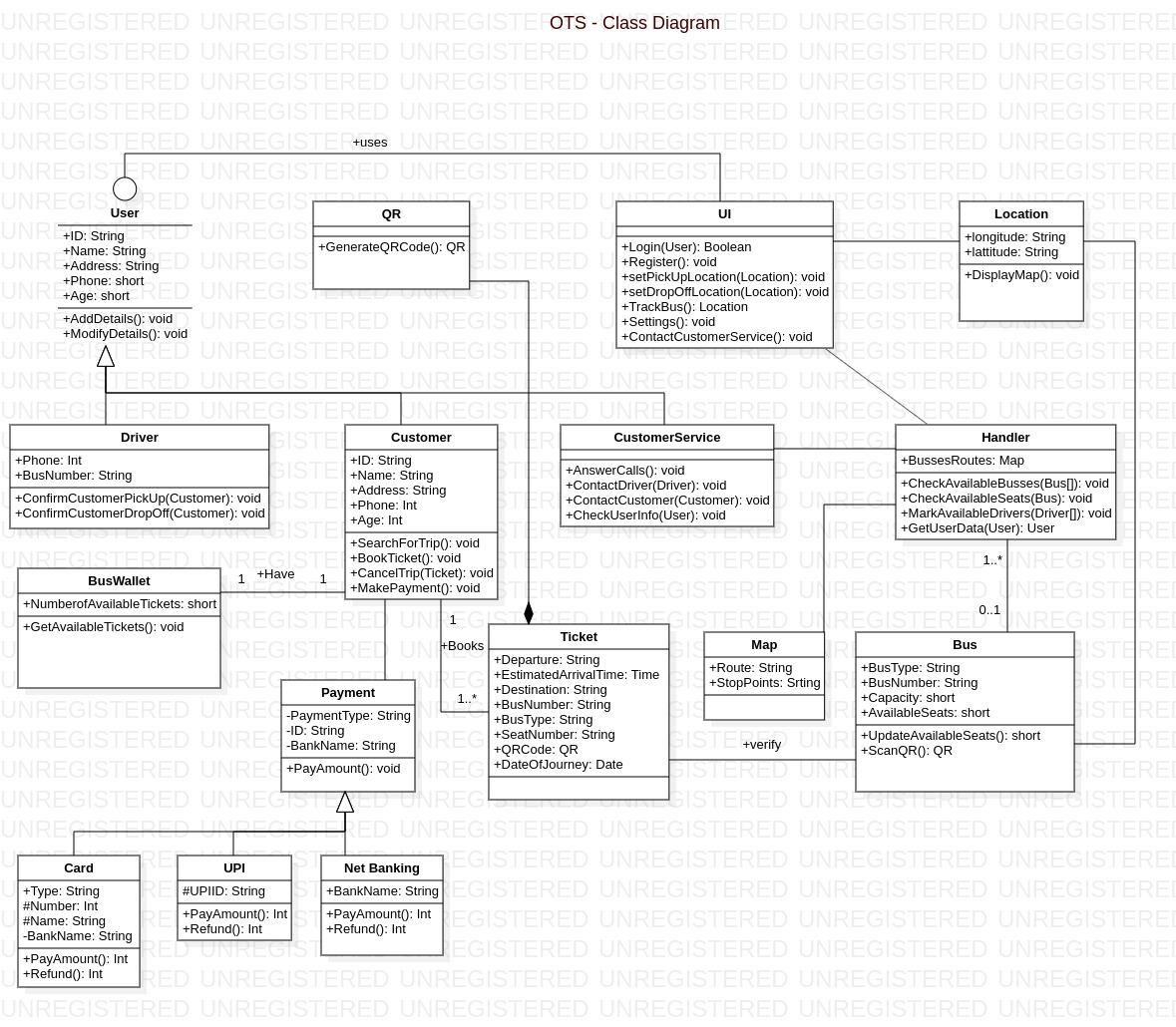
**Customer Service**

**Use Case** :

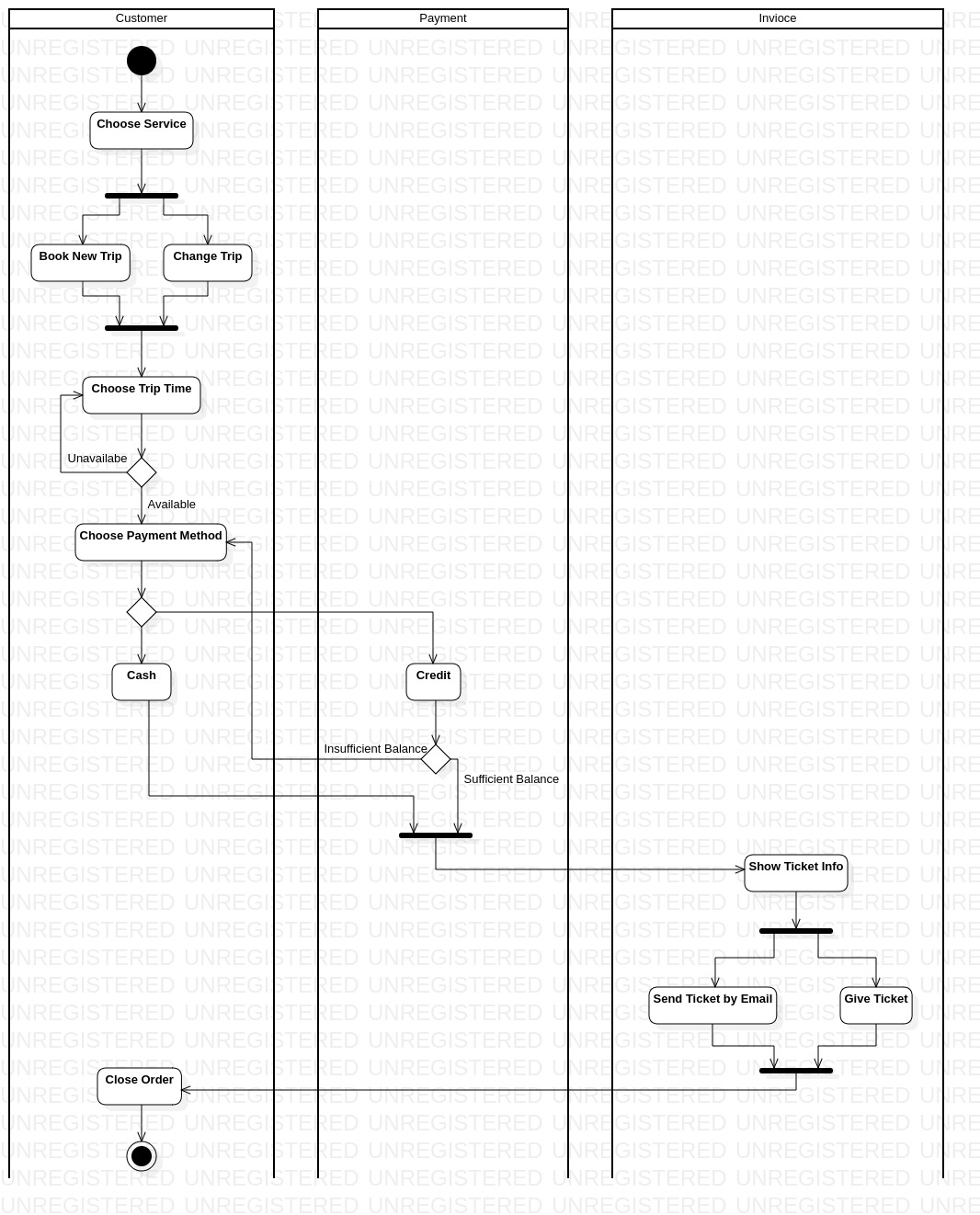
Call Customer Service : *Acts as intermediary between User, Driver & Service and provide help to all.*

Insufficient Money : *Handles the payment related issues while booking.*

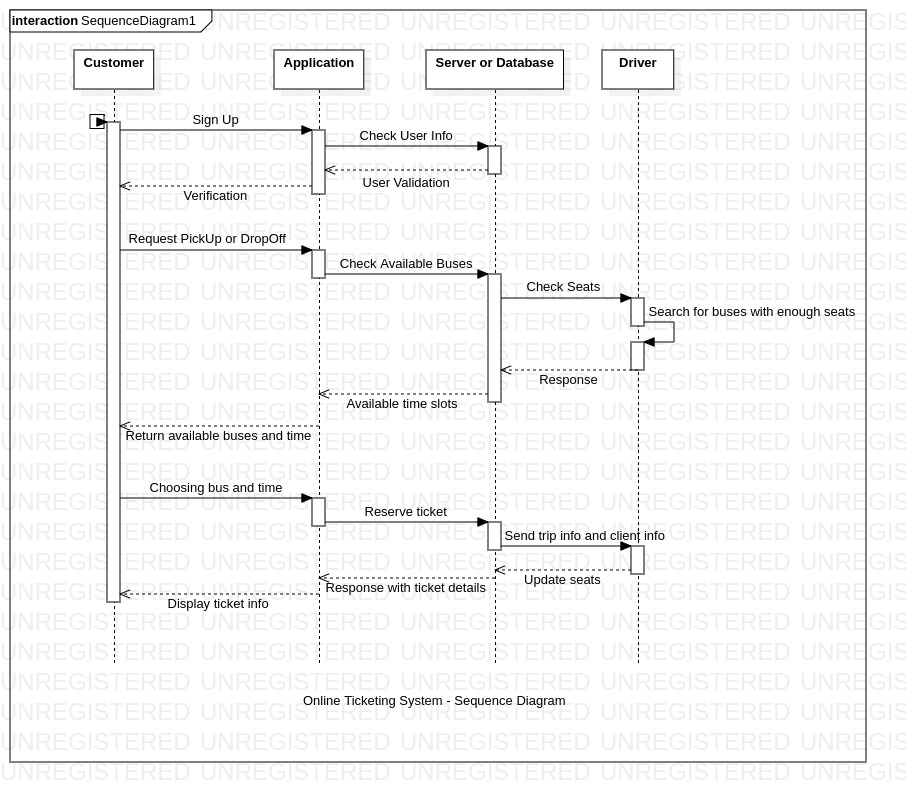
**CLASS DIAGRAM**



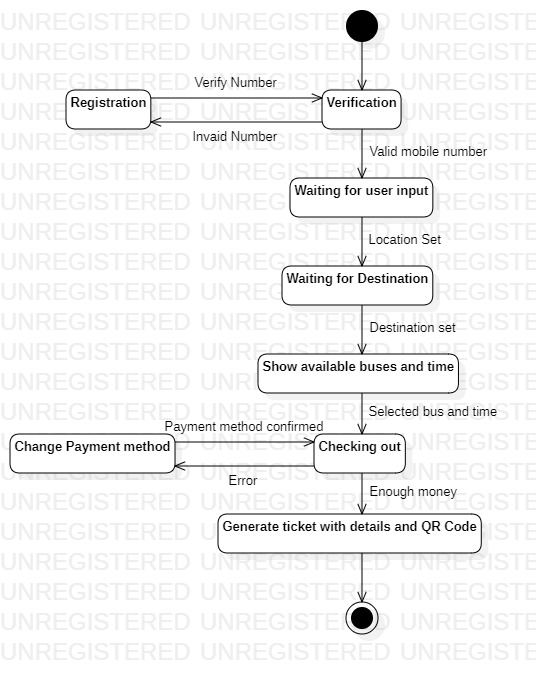
**ACTIVITY DIAGRAM**



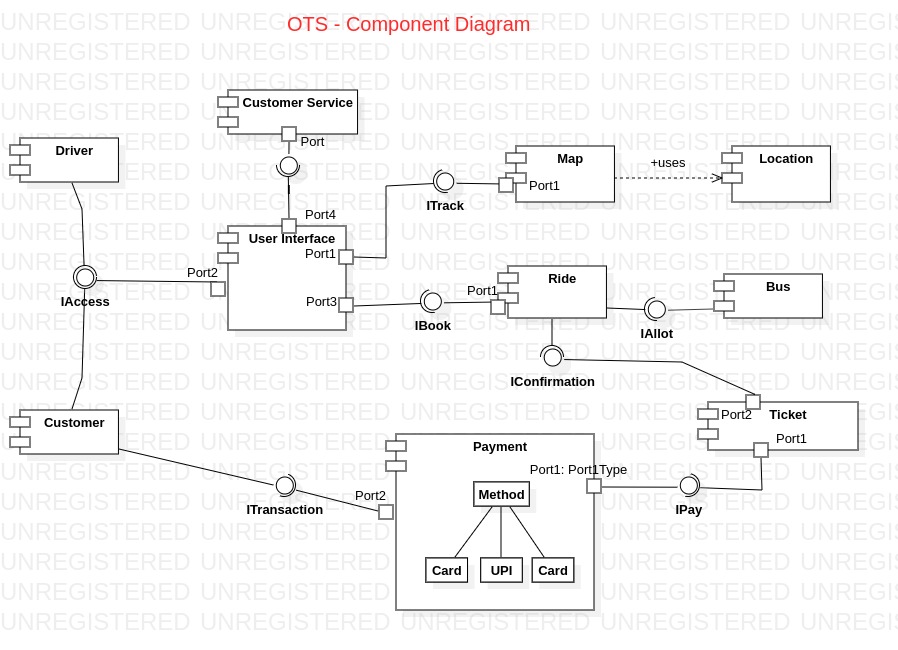
**SEQUENCE DIAGRAM**



**STATE DIAGRAM**



**COMPONENT DIAGRAM**



**DESIGN PRINCIPLES**

Single Responsibility Principle

*“A class should have single responsibility that denotes it should be considered having one reason to change.”*

Our system abides by the Principle of Single Responsibility.

Open-Close Principle

*“Software entities like classes, modules, and functions should be open for extension and closed for modification.”*

The system abides by Open-Closed Principle.

Liskov’s Substitution Principle

*“Derived Types must be completely substitutable for base types”, its an extension of the Open-Closed Principle.*

Violated because there are different representation of Child Classes [Driver and Customer].

Interface Segregation Principle

*“Client should not be forced to depend upon the interfaces they do not use.”*

The system violates this principle because driver, customer and CustomerService provider has a single interface to login.

Dependency Inversion Principle

*“High-Level modules should not depend on low-level modules. Both should depend on abstractions.”*

It abides by the Dependency Inversion Principle as interface is introduced between High-Level Modules and Low-Level Modules.

**DESIGN PRINCIPLES**

Information Expert

This principle is not violated as responsibilities are assigned to classes keeping in mind the information they possess.

Creator

This principle is not violated as the Customer is responsible for instantiating the Ticket object.

LowCoupling

This principle is not violated as the dependency of any Module B on any Module A is minimized. Ex: Dependency of Customer and Ticket.

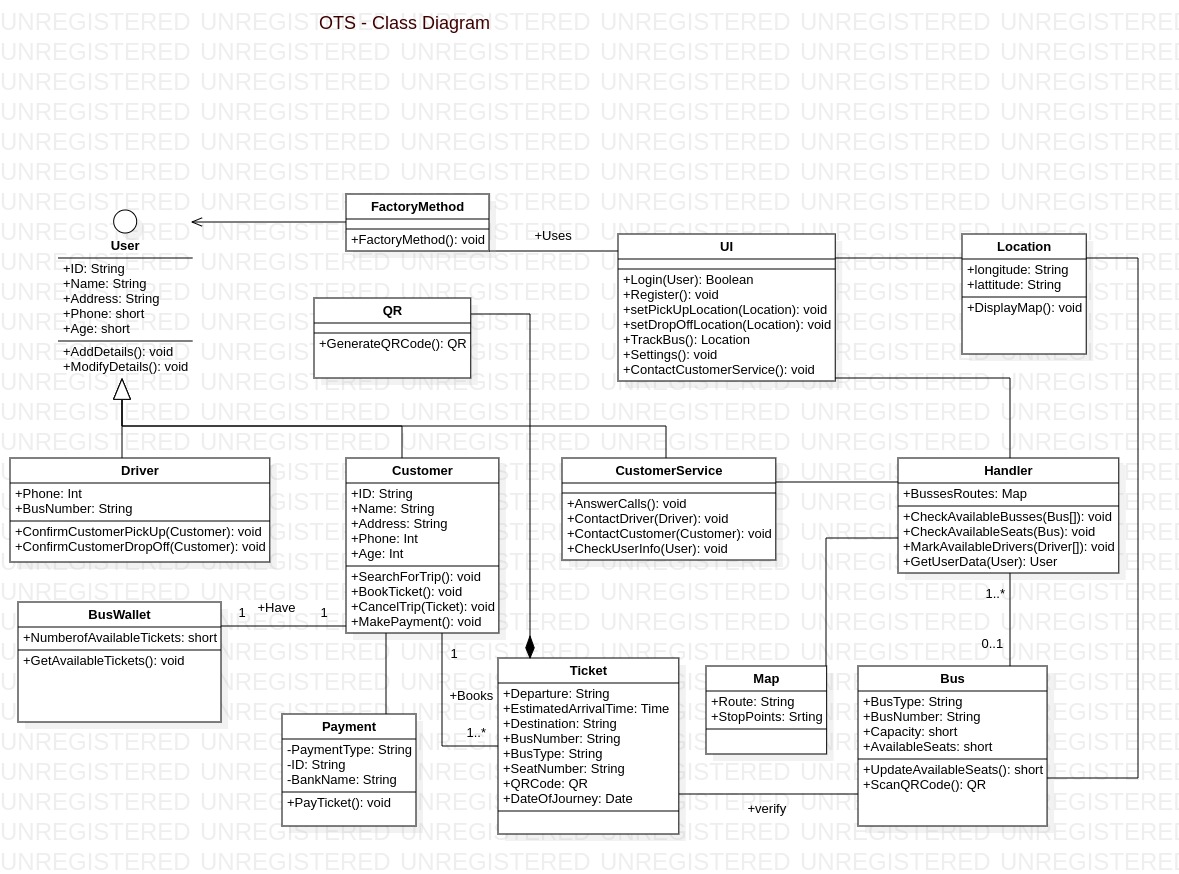
Controller

This principle is violated as there is no distinguishing factor between UI and domain object models.

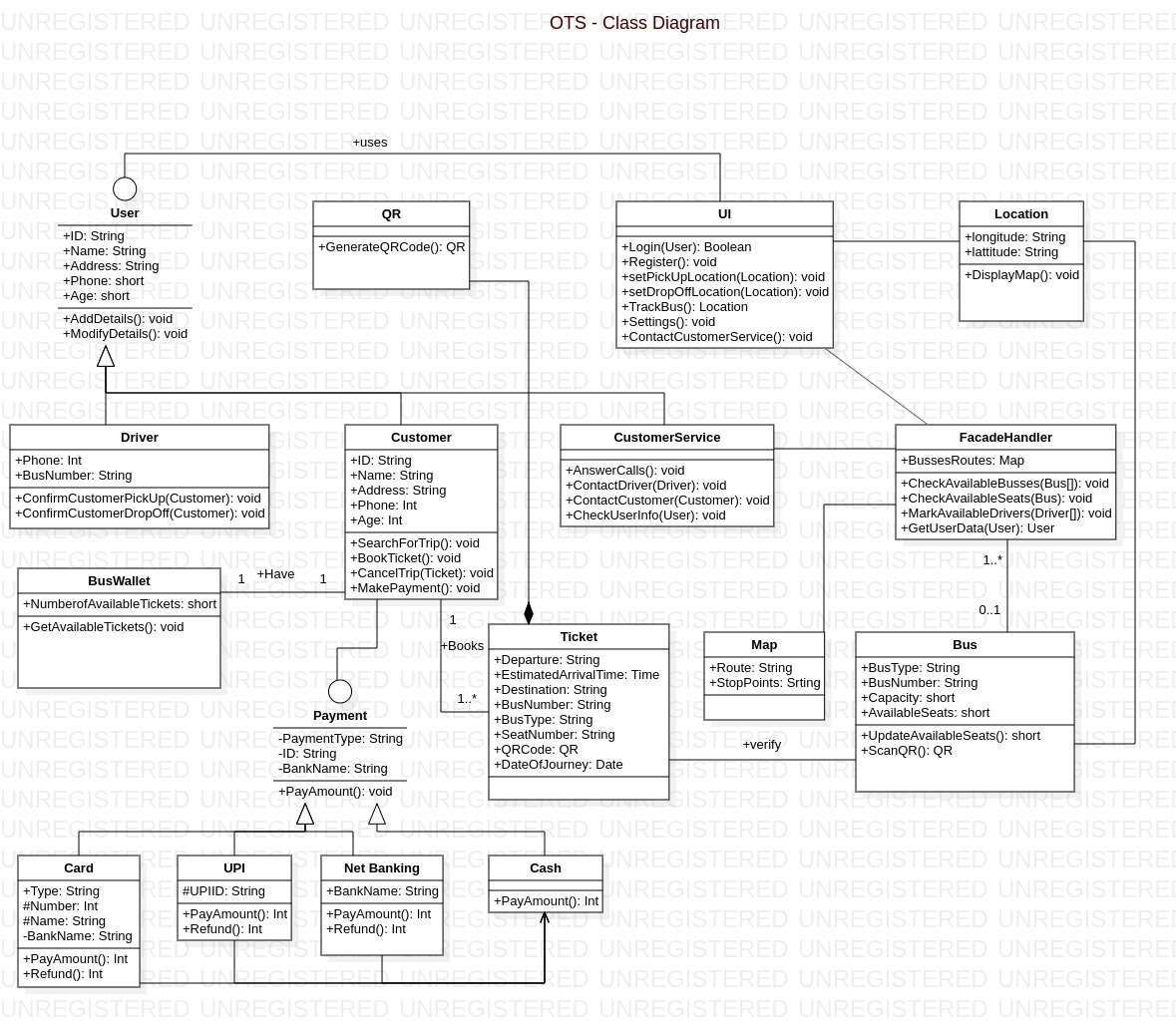
HighCohesion

This principle is violated in our class diagram. Ex: Customer module.

**DESIGN PATTERNS - Creational Pattern**



**DESIGN PATTERNS - Structural Pattern**



**DESIGN PATTERNS - Behavioral Pattern**

