



Software design specification document 2023

Project Team

ID	Name	Email
20210428	Nour Muhammed	nourmuhammad835@gamil.com
20210387	Mariam Haitham	haithammariam36@gmail.com
20210133	Rana Essam	ranaessam03@gmail.com
20210499	Noor Eyad	nooreyadd39@gmail.com



CS352: Sprint SDS – NRM - Online Payment System

SDS document

Contents

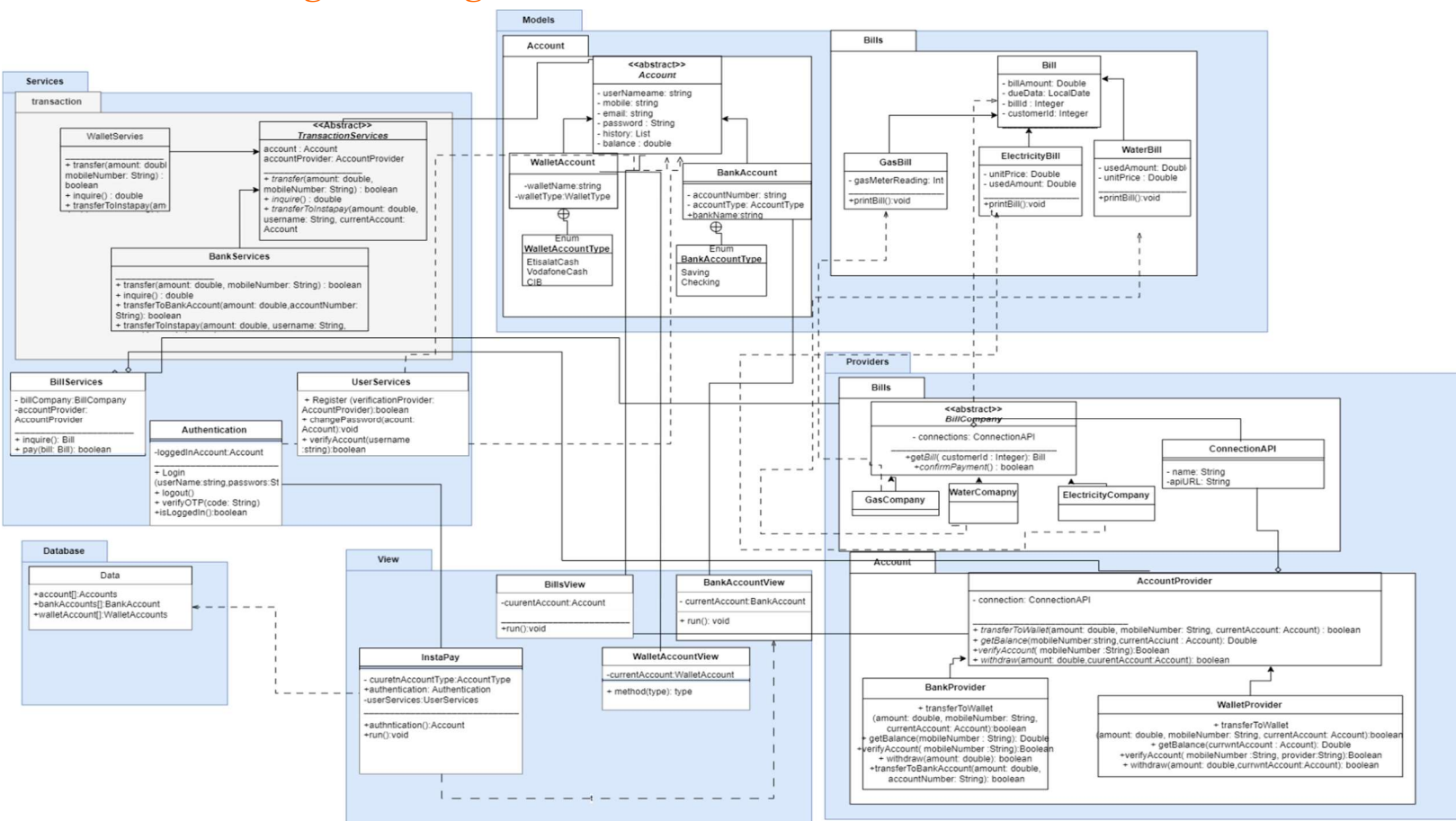
Class diagram design.....	3
Class diagram Explanation.....	4
Sequence diagram design	5
Github repository link.....	9



CS352: Sprint SDS – NRM - Online Payment System

SDS document

Class diagram design





CS352: Sprint SDS – NRM - Online Payment System

SDS document

Class diagram Explanation

1. Factory Method Design Pattern:

As we know, factory pattern aims to solve a fundamental problem in instantiation (the creation of a concrete object of a class) in OOP. In principle, creating an object directly within the class that needs or should use this object is possible, but very inflexible. It binds the class to this object and makes it impossible to change the instantiation independently of the class. This kind of code is avoided in the factory pattern approach by first defining a separate operation for creating the object – the factory method. As soon as this is called up, it generates the object instead of the class constructor already mentioned.

We can see the application of the factory pattern in our code in bills section. The factory abstract class being the BillCompany class, which has 3 different classes – GasCompany, WaterCompany, ElectricityCompany - that inherits from it to implement the getBill and ConfirmPayment functions according to the classes themselves and how they use them, all three of these classes representing the concrete factory classes. As for the abstract product class, it is represented by the Bill class, which has 3 different classes – GasBill, WaterBill, ElectricityBill – that inherits from it to create a bill according to the type the user wants and implement the printBill function that differs from a bill to another, all three of these classes represent the concrete product classes.

2. Strategy Pattern:

We're familiar with the strategy pattern and that it separates the behavior of an object from the object itself. The behavior is encapsulated into different strategies, each with its own implementation of the behavior. The context maintains a reference to a strategy object and interacts with it through a common interface. At runtime, the context can swap the current strategy with another one, effectively changing the object's behavior, which ensures flexibility and maintainability of the code.

This pattern is seen in our code in 2 main parts:

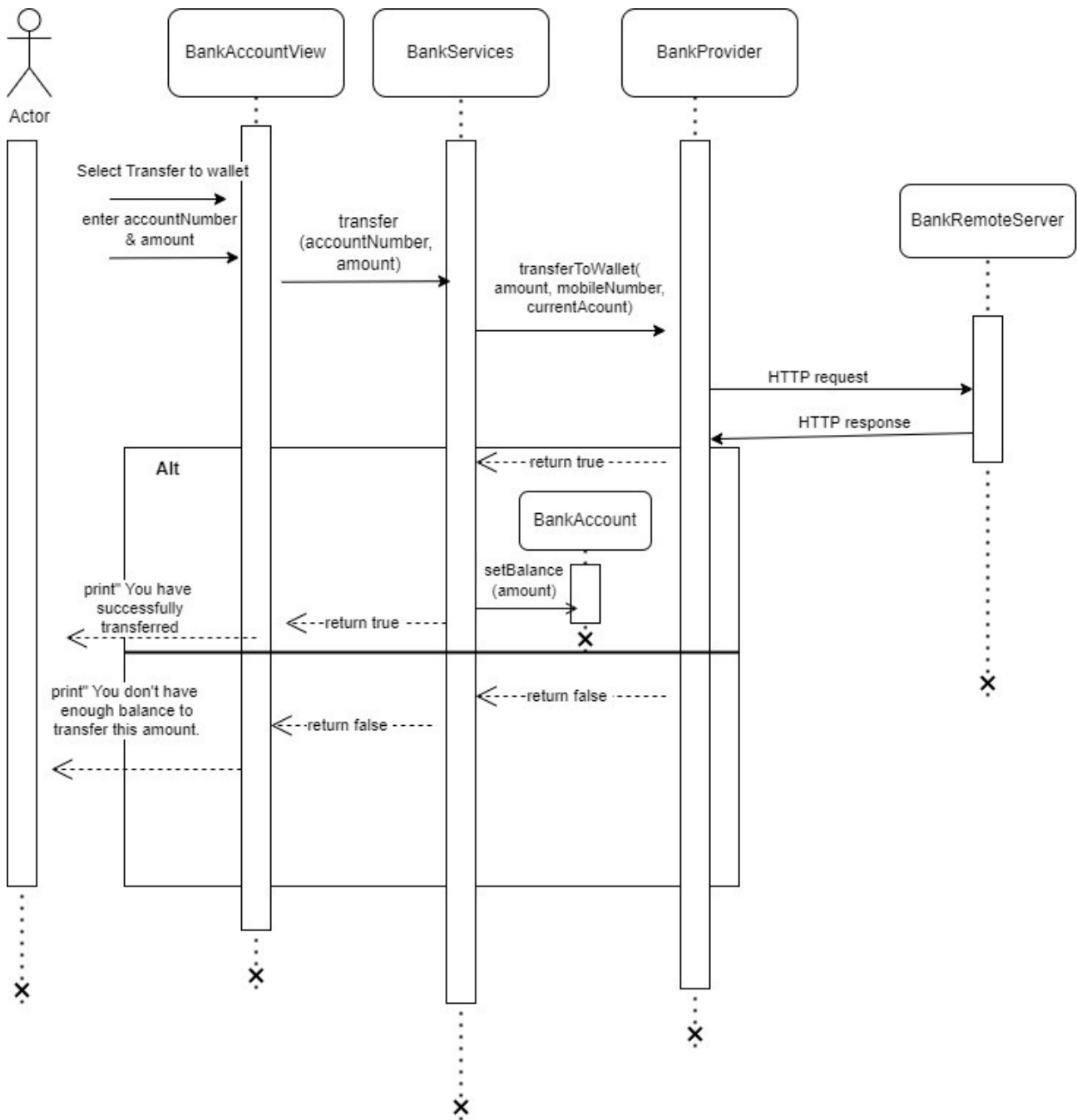
- a. The first time is in AccountProvider class, it being the strategy interface, and then the concrete strategies of it are the WalletProvider and BankProvider, then implementing verifyAccount, getBalance, withdraw and transferToWallet functions, all these functions being in both concrete classes, and an extra function – transferToBank - in the BankProvider class.
- b. The second time is in Account class, it being the strategy interface, and then the concrete strategies of it are the BankAccount and WalletAccountClass, then extending the Account class as an entity/model to make sure that all the attributes that differ between both of them are separate.

CS352: Sprint SDS – NRM - Online Payment System

SDS document

Sequence diagram design

1. Transferring from bank to wallet sequence diagram:





SDS document

```

sequenceDiagram
    actor Actor
    participant BankAccountView
    participant BillsView
    participant BillComapny
    participant BillServices
    participant Bill
    participant TransactionProvider

    Actor->>BankAccountView: 1: give a logged in user list of options
    activate BankAccountView
    BankAccountView->>BillsView: 2: user chooses option 5 : Pay bills
    deactivate BankAccountView
    activate BillsView
    BillsView->>BillComapny: 2.1: run()
    deactivate BillsView
    activate BillComapny
    BillComapny->>Actor: 2.1.1: give the user list of bill comapnies
    deactivate BillComapny
    activate Actor
    Actor->>BillsView: 3: chooses one of 3: gas, water, electricity
    deactivate Actor
    activate BillsView
    BillsView->>Actor: 4: give the user 2 options: inquire bill, pay bill
    deactivate BillsView
    activate Actor
    alt 5: user chooses to inquire bill
        Actor->>BillsView: 5.1: asks the user to enter the customer ID
        deactivate Actor
        activate BillsView
        BillsView->>Actor: 6: enters customer ID
        deactivate BillsView
        activate Actor
        Actor->>BillsView: 7: inquire()
        deactivate Actor
        activate BillsView
        BillsView->>BillComapny: 7.1: getBill(customerID)
        deactivate BillsView
        activate BillComapny
        BillComapny->>BillServices: 7.1.1: create bill
        deactivate BillComapny
        activate BillServices
        BillServices->>Bill: 7.1.1.1: printBill()
        deactivate BillServices
        activate Bill
        Bill->>Actor: 
        deactivate Bill
    else 8: user chooses to pay bill
        Actor->>BillsView: 9: asks the user to enter the customer ID
        deactivate Actor
        activate BillsView
        BillsView->>Actor: 10: enters customer ID
        deactivate BillsView
        activate Actor
        Actor->>BillsView: 10.1: pay(billComapny.getBill(customerID), currentAccount)
        deactivate Actor
        activate BillsView
        BillsView->>BillComapny: 10.1.1: getBill(customerID)
        deactivate BillsView
        activate BillComapny
        BillComapny->>BillServices: 10.1.1.1: create bill
        deactivate BillComapny
        activate BillServices
        BillServices->>TransactionProvider: 10.1.2: withdraw(amount, currentAccount)
        deactivate BillServices
        activate TransactionProvider
        TransactionProvider->>BillServices: 10.1.2.1: confirmPayment()
        deactivate TransactionProvider
        activate BillServices
        BillServices->>TransactionProvider: 10.1.2.1.1: return true
        deactivate BillServices
        activate TransactionProvider
        TransactionProvider->>BillsView: 10.1.2.1.1.1: return true
        deactivate TransactionProvider
        activate BillsView
        BillsView->>Actor: 10.1.2.1.1.1.1: print message that the payment was successful
        deactivate BillsView
    else 10.1.2.1.1.1.1: return false, insufficient amount
        BillComapny->>BillServices: 10.1.2.1.2: create bill
        deactivate BillComapny
        activate BillServices
        BillServices->>TransactionProvider: 10.1.2.1.1.2: withdraw(amount, currentAccount)
        deactivate BillServices
        activate TransactionProvider
        TransactionProvider->>BillServices: 10.1.2.1.1.2.1: return false, insufficient amount
        deactivate TransactionProvider
        activate BillServices
        BillServices->>BillsView: 10.1.2.1.1.2.1.1: return false, insufficient amount
        deactivate BillServices
        activate BillsView
        BillsView->>Actor: 11: print message that the payment was unsuccessful
        deactivate BillsView
    end
    deactivate Actor
  
```

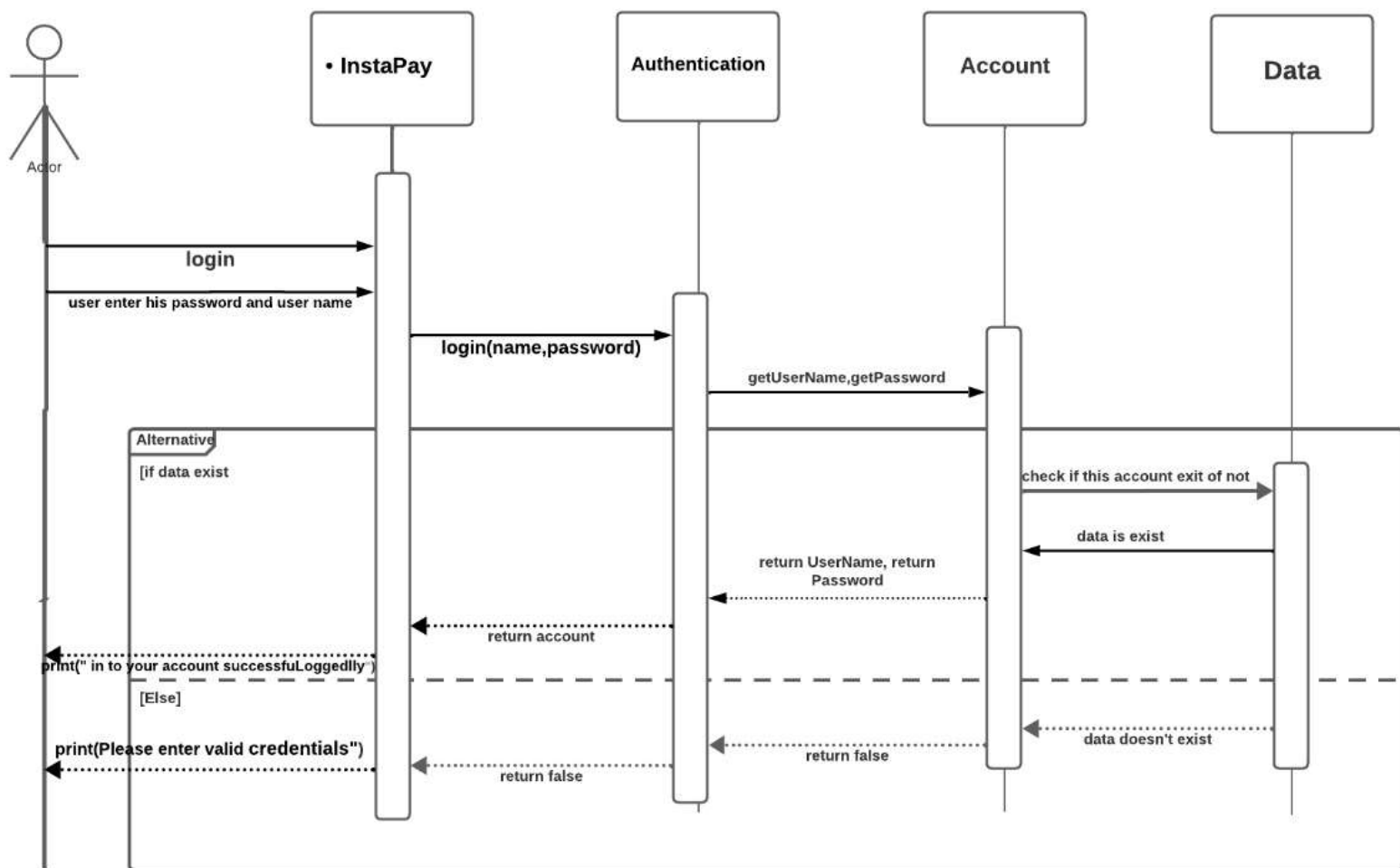
UML sequence diagram illustrating the bill payment process:

- Participants:** Actor, BankAccountView, BillsView, BillComapny, BillServices, Bill, TransactionProvider.
- Sequence of Messages:**
 - Actor to BankAccountView: 1: give a logged in user list of options
 - BankAccountView to BillsView: 2: user chooses option 5 : Pay bills
 - BillsView to BillComapny: 2.1: run()
 - BillComapny to Actor: 2.1.1: give the user list of bill comapnies
 - Actor to BillsView: 3: chooses one of 3: gas, water, electricity
 - BillsView to Actor: 4: give the user 2 options: inquire bill, pay bill
 - Alt Block (User chooses to inquire bill):**
 - Actor to BillsView: 5.1: asks the user to enter the customer ID
 - Actor to BillsView: 6: enters customer ID
 - Actor to BillsView: 7: inquire()
 - BillsView to BillComapny: 7.1: getBill(customerID)
 - BillComapny to BillServices: 7.1.1: create bill
 - BillServices to Bill: 7.1.1.1: printBill()
 - Alt Block (User chooses to pay bill):**
 - Actor to BillsView: 9: asks the user to enter the customer ID
 - Actor to BillsView: 10: enters customer ID
 - Actor to BillsView: 10.1: pay(billComapny.getBill(customerID), currentAccount)
 - BillsView to BillComapny: 10.1.1: getBill(customerID)
 - BillComapny to BillServices: 10.1.1.1: create bill
 - BillServices to TransactionProvider: 10.1.2: withdraw(amount, currentAccount)
 - TransactionProvider to BillServices: 10.1.2.1: confirmPayment()
 - BillServices to TransactionProvider: 10.1.2.1.1: return true
 - TransactionProvider to BillsView: 10.1.2.1.1.1: return true
 - BillsView to Actor: 10.1.2.1.1.1.1: print message that the payment was successful
 - Alt Block (Insufficient amount):**
 - BillComapny to BillServices: 10.1.2.1.2: create bill
 - BillServices to TransactionProvider: 10.1.2.1.1.2: withdraw(amount, currentAccount)
 - TransactionProvider to BillServices: 10.1.2.1.1.2.1: return false, insufficient amount
 - BillServices to BillsView: 10.1.2.1.1.2.1.1: return false, insufficient amount
 - BillsView to Actor: 11: print message that the payment was unsuccessful

CS352: Sprint SDS – NRM - Online Payment System

SDS document

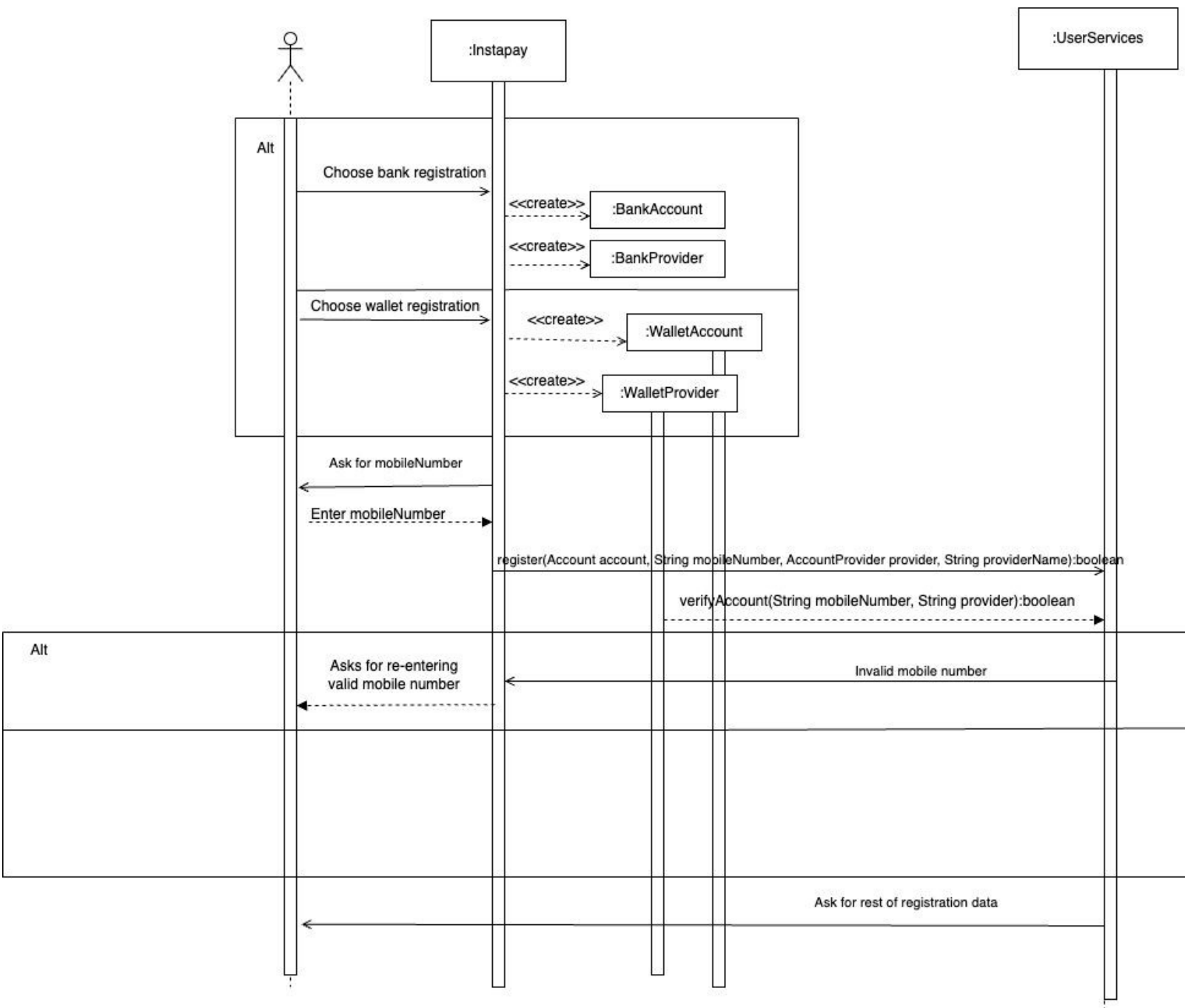
3. Log-in:



CS352: Sprint SDS – NRM - Online Payment System

SDS document

4. Registration





CS352: Sprint SDS – NRM - Online Payment System

SDS document

GitHub repository link

<https://github.com/RanaEssam03/Online-Payment-System>