Requirements

User

1. The user should be able to sign-in to the system. Given the user’s email and a password, the user can login to the system and use any of the system functionalities. 2. The user should be able to sign up to the system. The user should provide his username, email and password. The system should check if the username or the email is registered before, if they are not registered before then the signup process should complete successfully, if not, the system will show an error to the user

3. The user should be able to search for any service in the system. The user can type the service name and the system will return all services that match the user query.

4. The user can pay for any service in the system. The system should prompt the user to the payment form when the user asks to pay for any service. The default way is to pay via credit card. The system should allow the user to consume from the wallet (check Req. 6) for this payment. If the service that should to receive the payment accepts cache on delivery, then this option should be visible too.

5. The user can ask for a refund for any complete transaction to any given service. The refund request will be issued by the user and sent to the admin. If the admin approves the refund then the refund process should complete successfully.

6. The system maintain a wallet balance for each user. The user should be able to add any funds to the wallet. Adding funds to the wallet should be done via credit card.

7. The user should be able to check any discount for any service in the system. Discounts could be added by the admin (this will be discussed later).

Admin

1. The admin should be able to add any new service provider to the system. Provider consists of a form to be sent to the user and a handler for this form. So for example Vodafone Cash provider consists of a form (Mobile number, amount) and a handler for this form (given the user answers to this form then provider will handle the user request). The form could consist of any number of field. The type of fields that are supported are text field and drop down field

2. The admin should be able to add discounts to the system. There are two types of discounts.

a. Overall discounts. For example the user should have 10% discount for the first transaction (regardless the service)

b. Specific discount.. For example the admin can apply 20% discount for all mobile recharge services.

For any given service. All overall discounts and specific discounts for this service should apply.

3. The admin should be able to list all user transactions. The transactions types are

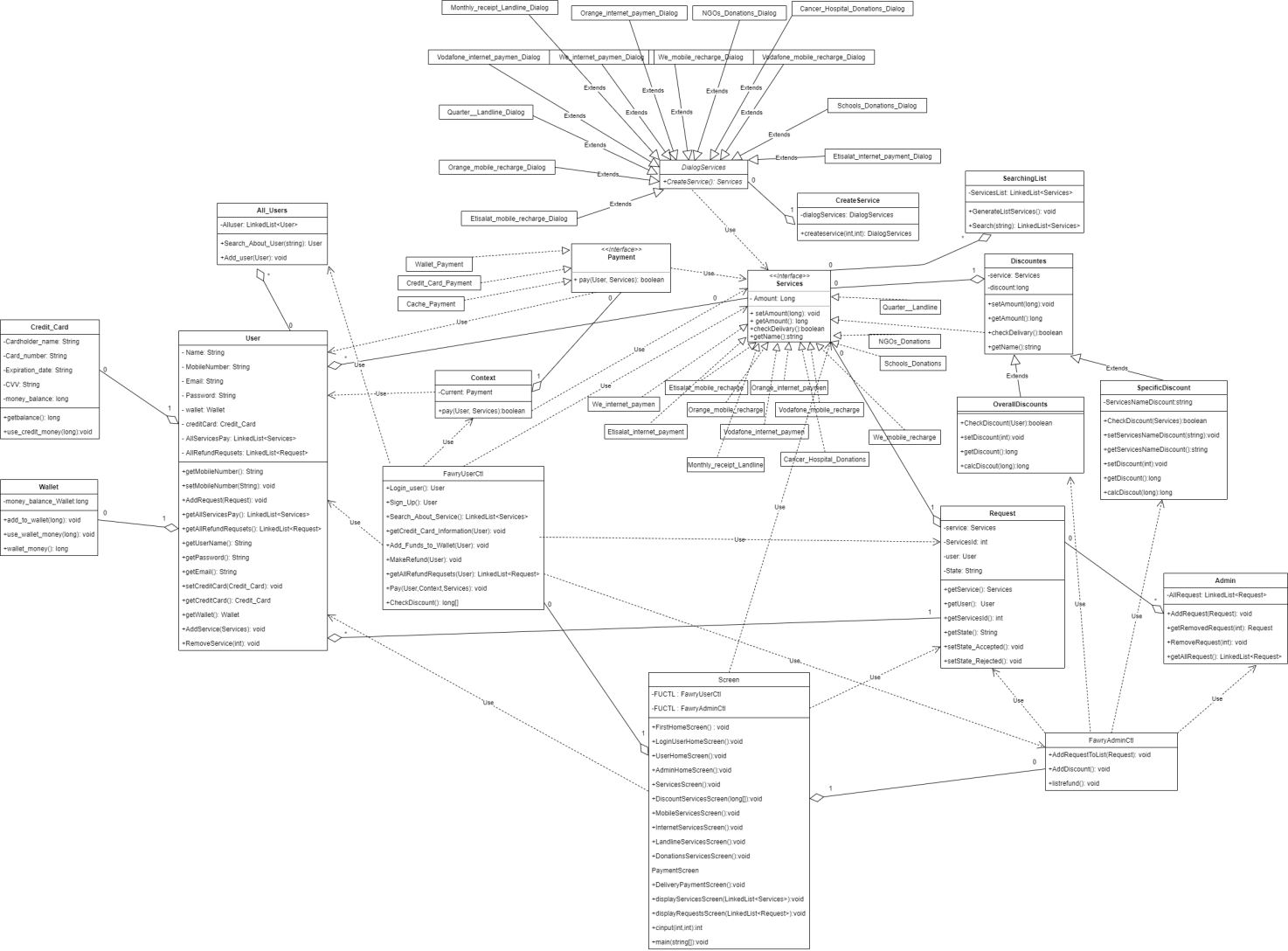
a. Payment transaction.

b. Add to wallet transaction.

c. Refund transaction.

4. The admin should be able to list all refund requests. Each refund request should contain the related service and the amount to be refunded. The admin should be able to accept or reject any refund request and if any refund request got accepted a refund transaction should be processed.

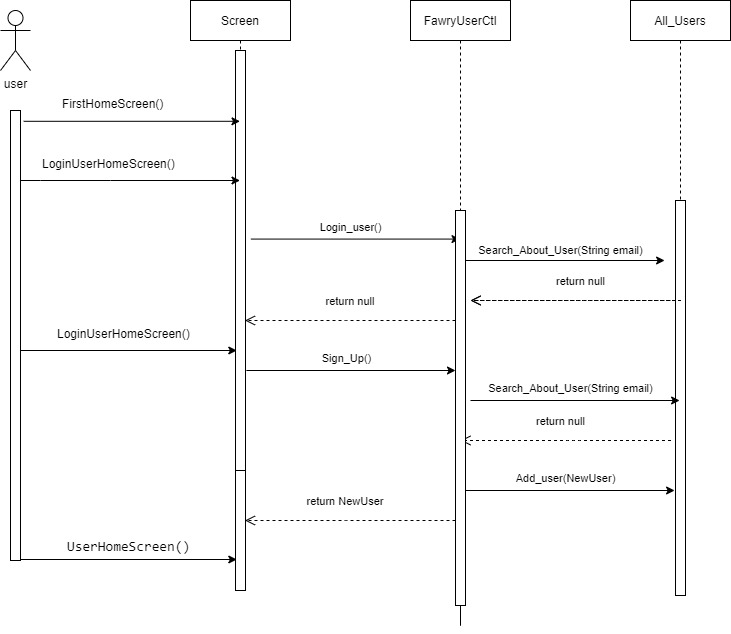
# Class diagram design

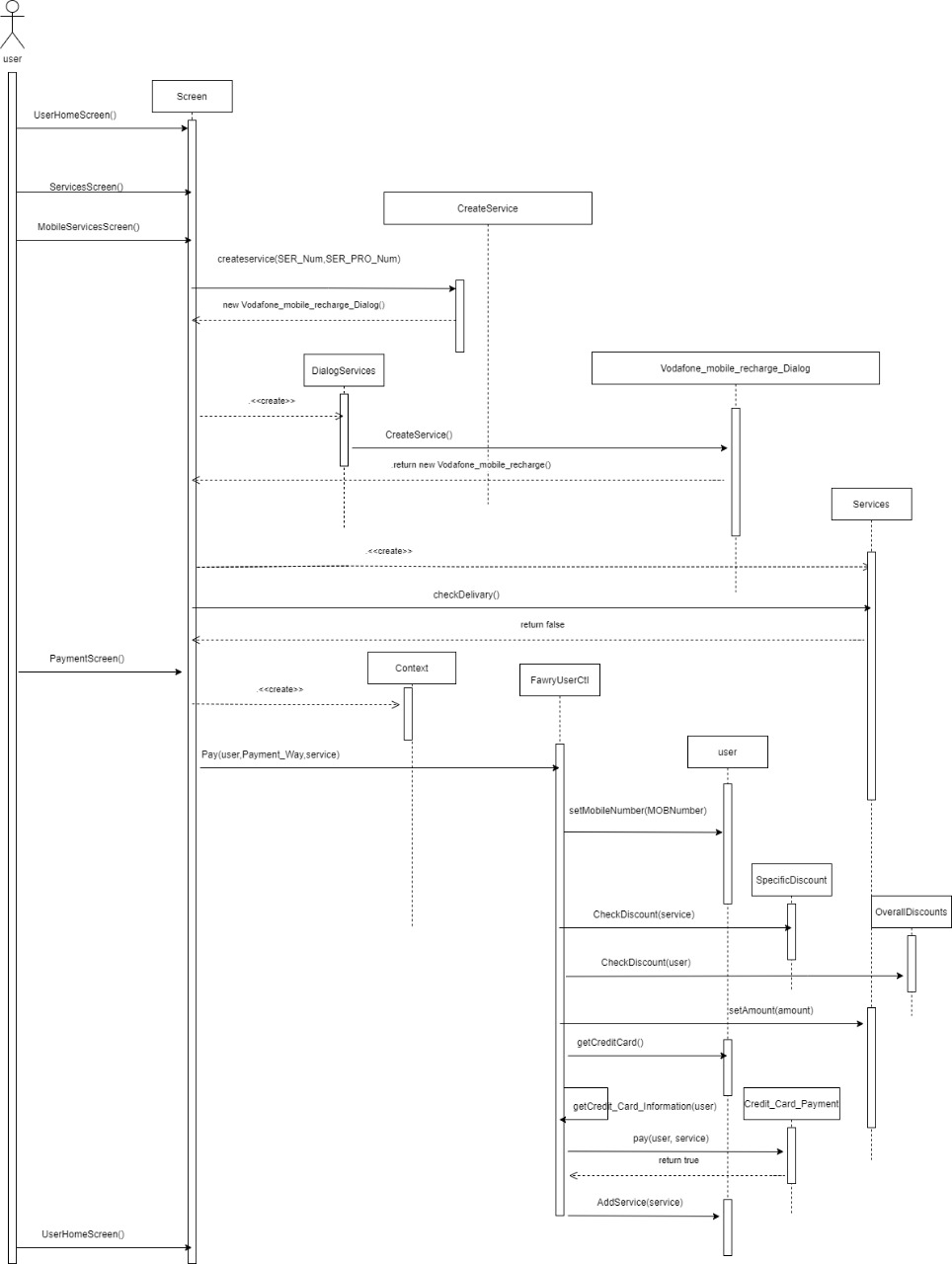
****

# Class diagram Explanation

* **Factory Method : we have a problem that we don’t know beforehand the exact types and dependencies of the objects that user should work with so we use Factory Method which is a creational design pattern that provides an interface for creating objects in a concrete classes, but allows concrete classes to alter the type of objects that will be created and the classes we used in this design (CreateService,** **Services,** **DialogServices,** **Cancer\_Hospital\_Donations\_Dialog,** **Etisalat\_internet\_payment\_Dialog,** **Etisalat\_mobile\_recharge\_Dialog,** **Monthly\_receipt\_Landline\_Dialog,** **NGOs\_Donations\_Dialog,** **Orange\_internet\_paymen\_Dialog,** **Orange\_mobile\_recharge\_Dialog,** **Quarter\_\_Landline\_Dialog,** **Schools\_Donations\_Dialog,** **Vodafone\_internet\_paymen\_Dialog** **Vodafone\_mobile\_recharge\_Dialog,** **We\_internet\_paymen\_Dialog,** **We\_mobile\_recharge\_Dialog, Cancer\_Hospital\_Donations, Etisalat\_internet\_payment, Etisalat\_mobile\_recharge, Monthly\_receipt\_Landline, NGOs\_Donations, Orange\_internet\_paymen, Orange\_mobile\_recharge, Quarter\_\_Landline, Schools\_Donations, Vodafone\_internet\_paymen, Vodafone\_mobile\_recharge, We\_internet\_paymen, We\_mobile\_recharge)**
* **Decorator: we use the Decorator pattern because we need to be able to assign extra behaviors to objects at runtime without breaking the code that uses these objects so when we want to add discount to the payment or more than one discount we will use payment as base and decorate it with discounts and the classes we used in this design(Services ,Discounts , OverallDiscounts , SpecificDiscount ,** **Cancer\_Hospital\_Donations,** **Etisalat\_internet\_payment,** **Etisalat\_mobile\_recharge,** **Monthly\_receipt\_Landline,** **NGOs\_Donations,** **Orange\_internet\_paymen,** **Orange\_mobile\_recharge,** **Quarter\_\_Landline,** **Schools\_Donations,** **Vodafone\_internet\_paymen,** **Vodafone\_mobile\_recharge,** **We\_internet\_paymen,** **We\_mobile\_recharge)**
* **Strategy: we use it because we have more than way of payment and each one has it’s own strategy so we make interface of all payment and context to choose which one we will use and the classes we used in this design are ( Context , Payment , Credit\_Card\_Payment , Wallet\_Payment , Cache\_Payment)**

# Sequence diagram design



****

