Cairo University Faculty of Computers and Artificial Intelligence



**Advanced SE**

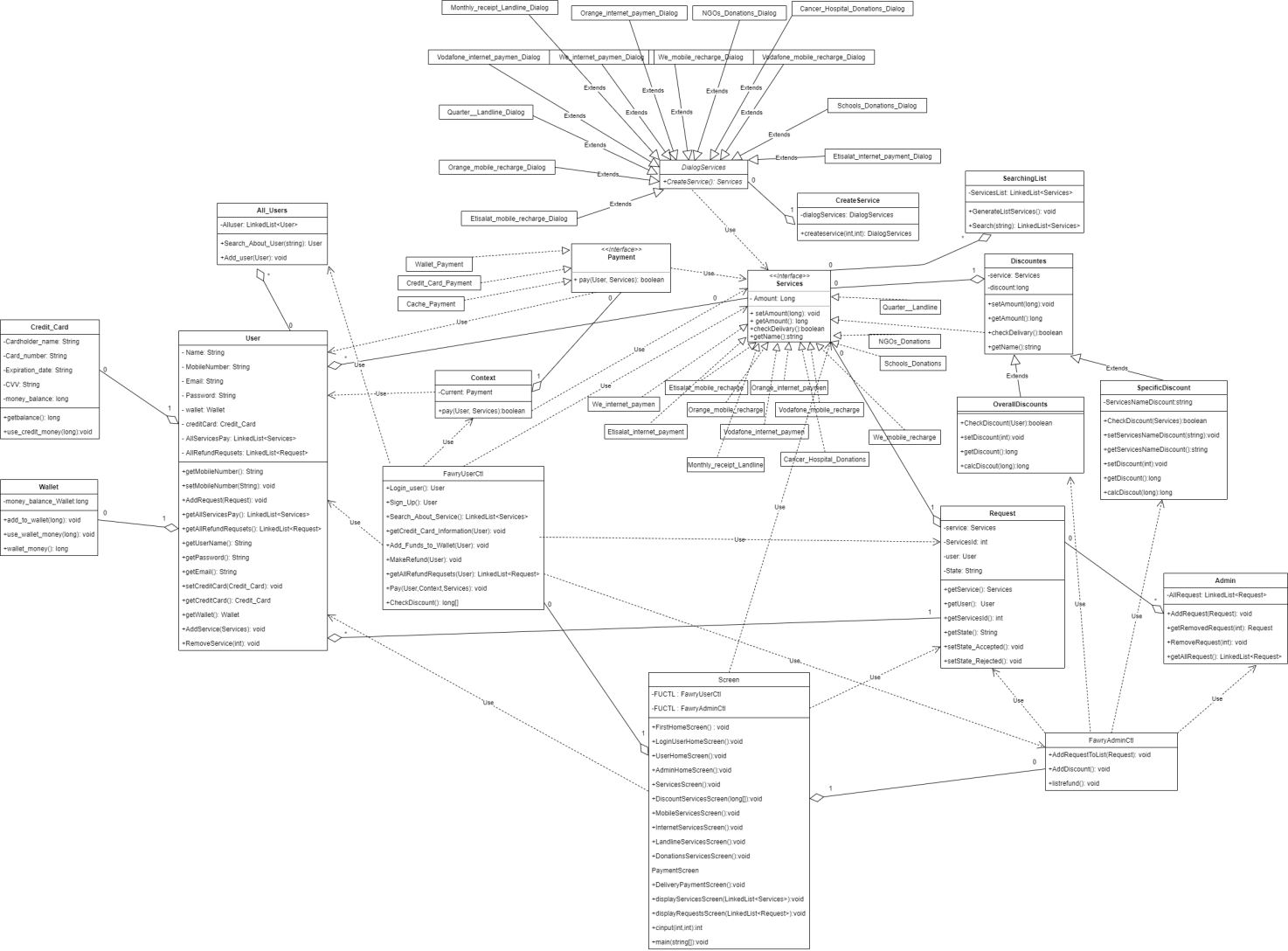
**Software design specification document**

**2022**

**Project Team**

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Email** |
| 20200500 | Mahmoud Adel Mamdouh | [mahmoudadel5556@gmail.com](mailto:mahmoudadel5556@gmail.com) |
| 20200794 | Mohamed Tarek Fathi | [mohamed21tarek@gmail.com](mailto:mohamed21tarek@gmail.com) |
| 20200140 | Habiba Ayman eltahary | [Pandatommo70@gmail.com](mailto:Pandatommo70@gmail.com) |
| 20200173 | Dina Othman emam | [Dinaosman581@gmail.com](mailto:Dinaosman581@gmail.com) |

# 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

* **List Sequence diagrams for the most important user story (according to your opinion).**
* **Make sure that each object in the sequence diagram has a corresponding class in the class description table above. If not, it will be REJECTED.**
* **Put actual function calls with proper parameters and return types corresponding to class diagrams.**
* **Following are couple of examples for small / meduim examples. We expect such diagrams, however there is a missing thing in them. Most of calls don’t have parameters. Please always specify the parameters in the call, matching the class diagram.**



# Github repository link

* <https://github.com/Mahmoudadel17/Software_project.git>