Cairo University Faculty of Computers and Artificial Intelligence



**Advanced SE**

**Software design specification document**

**2022**

**Project Team**

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Email** |
| 20200520 | Mariam Ashraf Amin | mariamashrafamin@gmail.com |
| 20200521 | Mariam Saeid Shawky | mariamsaeid142@gmail.com |
| 20200186 | Reem Ayman Abdel-Fattah | reem.ayman52002@gmail.com |

Contents

[Instructions[To be removed] 2](#_Toc120811426)

[Class diagram design 3](#_Toc120811427)

[Class diagram Explanation 4](#_Toc120811428)

[Sequence diagram design 6](#_Toc120811429)

[Github repository link 13](#_Toc120811430)

# Class diagram design

**Diagram, schematic

Description automatically generatedThere is picture with more clear quality in folder**

# Class diagram Explanation

**Command pattern**:

* **Classes:**

Command as command class,

RegistrationCommand, UserCommand and AdminCommand as concrete command class,

RegistrationMenu, UserMenu and AdminMenu as receiver class.

* Using command pattern to turn the request itself into an object. This object can be stored and passed around like other objects.
* We can see that AdminCommand acts as controller which lets user interact with InitiateData and AdminMenu so passing reply on fund request,

RegistrationCommand acts as controller which lets user interact with InitiateData and RegistrationMenu so let user signup or sign in.

UserCommand acts as controller which lets user interact with InitiateData and UserMenu so let user choose options, services and it’s provider, handling payment and refund.

**Factory method pattern:**

* **Classes:**

Services as product class,

MobileRechargeServices, InternetPaymentServices, LandLiveServices and Donation as concrete product class,

ServiceProvider as creator class,

Vodafone, Etisalat, Orange and We as concrete creator class.

* Using factory method pattern to let sub-classes to choose the type of objects to create.
* We can see that Donation can be done by alternative ways (Vodafone-We-Etisalat-Orange)

**Decorator pattern:**

* Classes:

Services as component class,

MobileRechargeServices, InternetPaymentServices, LandLiveServices and Donation as concrete component class,

Discounts as base decorator class,

OverAllDiscount and SpecificDiscount as concrete decorator class.

* Using decorator pattern to let inserting new behaviors to objects by placing these objects inside one or some wrapper objects that contain the behaviors.
* We can see that in OverAllDiscount and SpecificDiscount we can use them to add new behavior in services’ subclasses, one of them or mix of two decorator.

# Sequence diagram design

1) admin list all refund requsets Sequence

Diagram

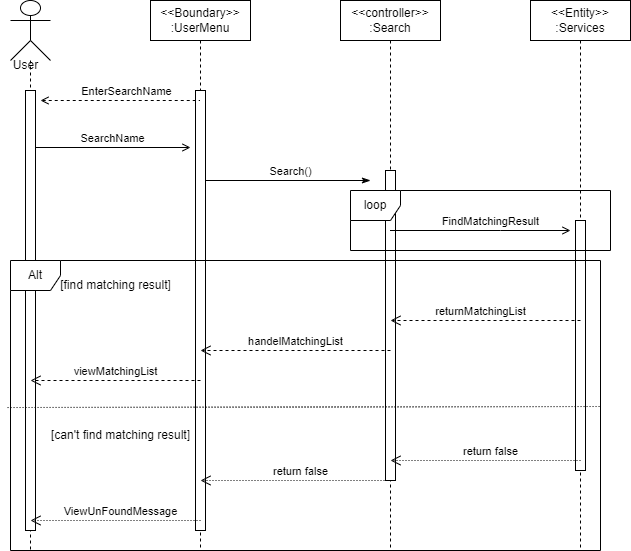
Description automatically generated

2) signIn – siginUp Sequence

**Diagram

Description automatically generated**

3) search Sequence



Diagram, engineering drawing

Description automatically generated4) choose service and service provider Sequence

Diagram, schematic

Description automatically generated5) pay Sequence

Diagram

Description automatically generated6) ask for refund Sequence

Diagram

Description automatically generated7) add funds Sequence

# Github repository link

* **https://github.com/Mariam-Ashraf-510/AdvancedSoftwareProject**