Cairo University Faculty of Computers and Artificial Intelligence



**Software design specification document**

**2022**

**Project Team**

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Email** |
| 20200234 | سميه محمد يوسف محمد | [sooma3mohammed@gmail.com](mailto:sooma3mohammed@gmail.com) |
| 20200501 | محمود عبدالراضي جادالرب | [mahmoudten55@gmail.com](mailto:mahmoudten55@gmail.com) |
| 20201038 | ايمان ابراهيم شعبان جاد | [emanelkaser@gmail.com](mailto:emanelkaser@gmail.com) |
| 20201061 | دينا احمد عبدالراضي | [da449393@gmail.com](mailto:da449393@gmail.com) |

Contents

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

[Class diagram design 2](#_Toc120811427)

[Class diagram Explanation 3](#_Toc120811428)

[Sequence diagram design 3](#_Toc120811429)

[Github repository link 4](#_Toc120811430)

# Instructions[To be removed]

* **IMPORTANT. Rename this document to** Phase2\_LabGroupNumber\_Phase1\_StudentID1\_StudentID2\_StudentID3\_StudentID4**\_SDS Document.docx**
* **Remove the following notes and any red notes**

**Read ME**

* To access admin functionality please use the following account: “email: admin, password: 123”
* If you want to enter to user account you can use sign up to make account or use the following account: “email: user, password: 123”

# Class diagram design

* **You should provide clean version for your class diagram design.**
* **Class diagram is a static diagram and should not represent any dynamic flow of events.**
* **Put Relationships between classes and the types of the relationships.**
* **Put multiplicity.**
* **Put relationship name (e.g. faculty "offer" course).**
* **Put attributes in the classes.**
* **Put functions &Put parameters.**
* **Put data types of each attributes and the parameters.**
* **Highly perfered: Each class has a corresponding interface**
  + **Let all objects parameters and returns be of interface type.**
* **See Shopping Cart Case Study**



# Class diagram Explanation

* Singleton, we used it to manage a shared resources and to restrict the instantiation of our classes and ensures that only one instance of our classes exists.

Classes: UserAccounts, PaymentWaysList, ServiceList, SignList, refundsRequestsModel, UserFunctionList, AdminFunctionList, TypeOfDiscounts, DonationService, InternetPaymentService, LandlineService and MobileRechargeService.

* Factory, we used it to allow the sub-classes in our system to choose the type of objects to create which enable us to initiate our classes in a separate class.

Classes: SignFactory, PaymentFctory, MakeDiscountFactory, ServiceFactory, DiscountFactory, FunctionFactory, UserFunctionFactory and AdminFunctionFactory

* Observer, to make the transaction request subscribe to the refunds request model when the user wants to send it to the admin. When rejecting or accepting the request, the request state will be updated and then the request will unsubscribe from this model.

Classes: refundsRequestsModel and TransactionRequest

* Strategy pattern: This pattern enables us to select a specific algorithm at run time, so we used it because we have multiple algorithms for specific classes in our system

Classes (the name of abstract/interface classes): FormUI, Controller, Discount, Authentication, ISearch, Check and 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.**



# Requirements Exposure as Web Service API

**Part 1: Exposed Postman Collection**

**Part 2:**

|  |  |  |
| --- | --- | --- |
| Requirement | Exposed API | Examples |
| The system should check if the username or the email is registered before. | GET /user/check  A service to check if the user exists or not. This service returns all user info if exists  Input: email and password. | GET /user/check  body:  {      "email":"user",      "password":"123"  } |
| 1- The user should be able to sign-in to the system. Given the user’s email and a password  2- The user should be able to sign up to the system. The user should provide his username, email and password. | POST /user/sign/{way}  A service to sign to the system.  {way} = sign-in or sign-up  If the way is sign-in the user can sign to the system with his account  Input: email and password.  If the way is sign-up the user can add new user to the system  Input: username, email and password. | POST /user/sign/sign-up  body:  {      "username":"ahmed",      "email":"ahmed",      "password":"123456"  }  POST /user/sign/sign-in  body:  {      "email":"admin",      "password":"123"  } |
| 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 | GET /user/search/{search}  A service to search for service in the system.  {search} will be the name of service  This service returns the name of all services that match the user query | GET /user/search/orange  no body needed |
| The user can pay for any service in the system. The default way is to pay via credit card. The system should allow the user to consume from the wallet. If the service that should to receive the payment accepts cache on delivery, then this option should be visible too. | 1- GET /user/provider/{type}/{name}/save-for-payment  A service to select service provider  {type} = landline, internet, mobile or donation  {name} = vodafone, etisalat, we, orange, ngo, school, cancer-hospital, monthly-receipt or quarter-receipt  Inputs: the parameters that the handler of this service can understand  2- POST /user/pay  A service to pay for the service using credit card (default) to complete the transaction  Input: number, password and amount (amount is required)  3- POST /user/pay/{way}  A service to pay for the service to complete the transaction  {way} = credit-card, wallet or cache  Input: needed inputs and amount (amount is required) | GET /user/provider/donation/school/save-for-payment  body:  {      "name":"school",      "address":"2 street"  }  POST /user/pay  POST /user/pay/credit-card  body:  {      "number":"123",      "password":"123",      "amount":"10"  }  POST /user/pay/wallet  body:  {      "amount":"10"  }  POST /user/pay/cache  body:  {      "address":"123",      "amount":"10"  } |
| 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. | 1- GET /user/transactions  A service to get transaction for the current user to make the user know the name of service he wants to make refund for it and its amount easily  2- POST /user/refund-request  A service to request refund for specific transaction  3- GET /user/refund-requests  A service to get all refund requests for the current user | GET /user/transactions  no body needed  POST /user/refund-request  body:  {      "service-name": "Vodafone Mobile Recharge Service",      "amount":50  }  GET /user/refund-requests  no body needed |
| The system maintains 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. | POST /user/add-funds  A service to add funds to the wallet via credit card  Input: amount and credit card information. | POST /user/add-funds  body:  {      "amount": 10,      "credit-card-info":{          "number":"123",          "password":"123"      }  } |
| The user should be able to check any discount for any service in the system. | 1- GET /user/get-discount-list  A service to check discount for all services  2- GET /user/get-discount-list/{name}  A service to check discount for specific service  {name} will be the name of service | GET /user/get-discount-list  no body needed  GET /user/get-discount-list/vodafone-internet  no body needed  GET /user/get-discount-list/monthly-quarter-landline  no body needed |
| The admin should be able to add discounts to the system. There are two types of discounts. a. Overall discounts.  b. Specific discount. | POST /admin/make-discount  A service to make discount for services  Inputs: type (overall or specific), discount-percentage and service-name for specific discount and transaction-number for overall discount | POST /admin/make-discount  body:  {      "type":"specific",      "service-name":"vodafone internet",      "discount-percentage":"10"  }  POST /admin/make-discount  body:  {      "type":"overall",      "transaction-number":"10",      "discount-percentage":"10"  } |
| 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. | GET /admin/transactions/{username}  A service to get all transactions for specific user  {username} will be the username of the user | GET /admin/transactions/user  no body needed |
| 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. | GET /admin/get-refunds  A service to get all refund requests that the user made with refund id to make the admin select one easily  GET /admin/manage-refund  A service to manage specific refund  Inputs: id of the request and its status | GET /admin/get-refunds  no body needed  GET /admin/manage-refund  body:  {      "id":1,      "status":"accepted"  } |

# Github repository link

* <https://github.com/Sooma-M/PaymentSystem>
* If the link didn’t work, please contact with anyone of us via email or in our section