Cairo University

Faculty of Computers and Artificial Intelligence

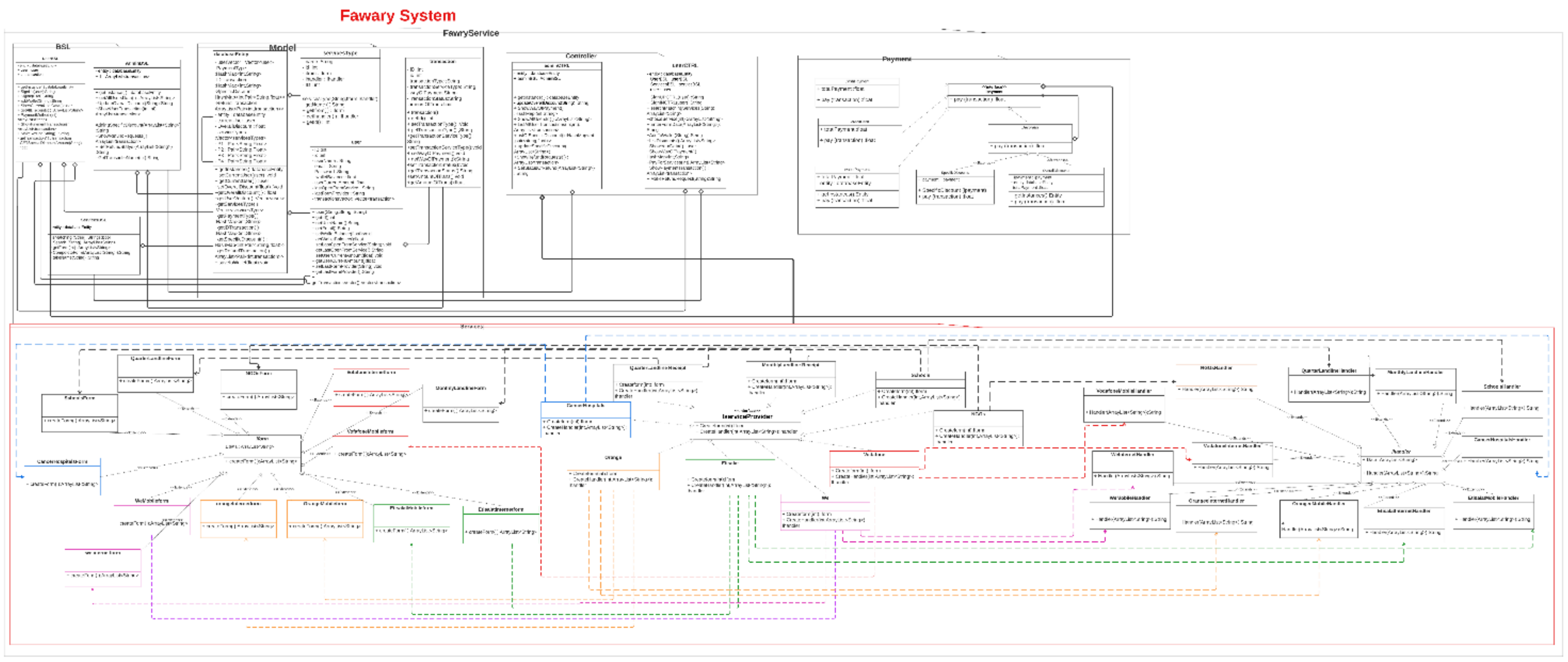
Software design specification document

2022

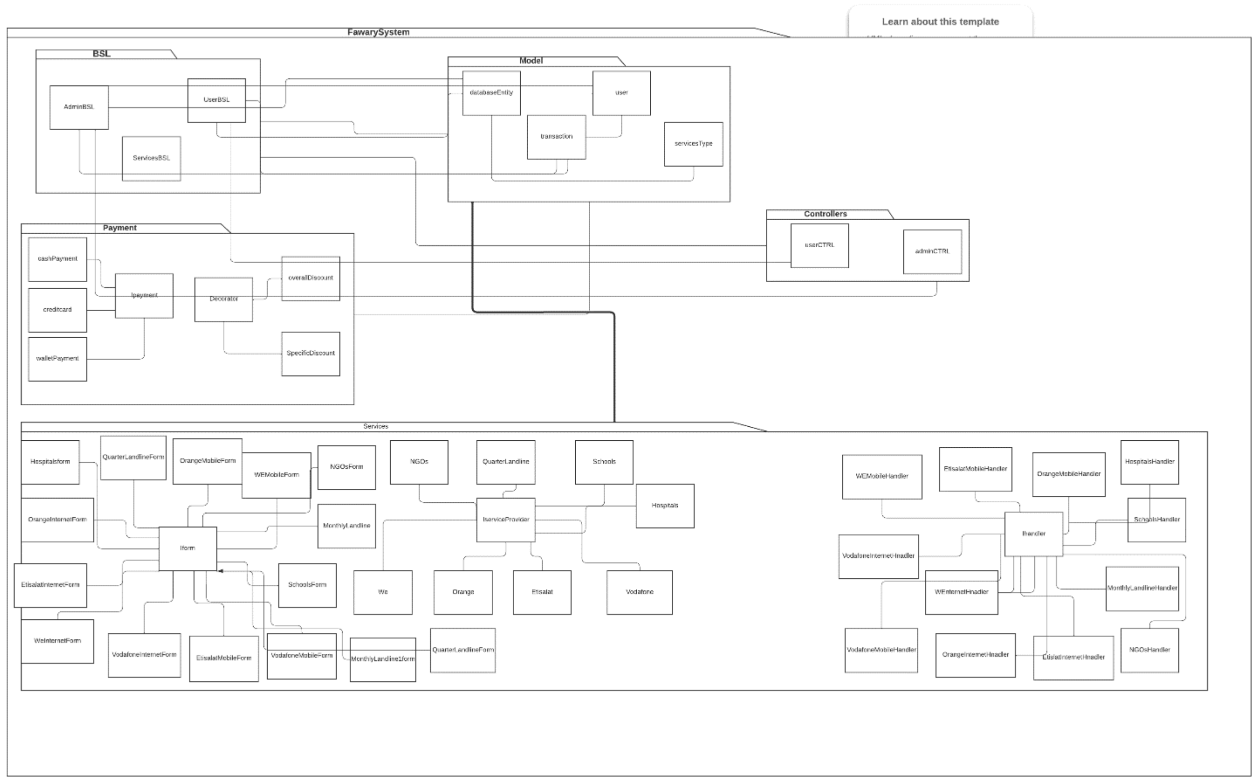
Contents

Class diagram design

SDS document



SDS document



Class diagram Explanation

1)Abstract Factory Method Design Pattern:

Justification: we used this pattern while designing the part of service and service providers where we create a form and create handler for each service our factories are(service providers)and the products are the form and handler for each service provider ,we used this pattern in order to:

1) avoid tight coupling between concrete products and client code.
2) Achieve *Single Responsibility Principle*
3) Achieve *Open/Closed Principle.*

Participating classes:

- class CancerHospitals

- **class** CancerHospitalDonationForm

- **class** CancerHospitalDonationHandler

- **class** Etislat

- **class** EtislatInternetForm

- **class** EtislatInternetHandler

- **class** EtislatMobileForm

- **class** EtislatMobileHandler

- **class** Vodafone

SDS document

- **class** VodafoneInternetForm
- **class** VodafoneInternetHandler
- **class** VodafoneMobileForm
- **class** VodafoneMobileHandler
- **class** Orange
- **class** OrangeMobileForm
- **class** OrangeMobileHandler
- **class** OrangeInternetForm
- **class** OrangeInternetHandler
- **class** We
- **class** WeMobileForm
- **class** WeMobileHandler
- **class** WeInternetForm
- **class** WeInternetHandler
- **class** School
- **class** SchoolDonationForm
- **class** SchoolDonationHandler
- **class** NGOs
- **class** NGOsDonationForm
- **class** NGOsDonationHandler

SDS document

- **class** MonthlyRecipt

- **class** MonthlyLandlineForm

- **class** MonthlyLandlineHandler

- **class** QuarterRecipt

- **class** QuarterLandlineForm

- **class** QuarterLandlineHandler

- **class** Iform

- **class** Ihandler

-interface IservicesProvider

2)Decorator Design Pattern

Justification: we used this pattern while designing the part of adding discounts via the admin as the admin should add two types of discounts which are(overall discount and special discount)with the capability of adding more types of discounts so that's why we choose the decorator pattern as we wanted to decorate the payment that is done by the user by adding discounts (that is how we place the object which was the payment inside a special wrapper objects that contain the behaviour,this pattern also helped us in the point of adding new decorators(new types of discounts)without violating any solid principle and it helped us in

1) add or remove responsibilities from an object at runtime.
2) can combine several behaviors by wrapping an object into multiple decorators.
3) Achieve the *Single Responsibility Principle*

Participating class:

-class Decorator

-class OverallDiscounts

-class SpecificDiscounts

-class CashPayment

-class creditCard

-class WalletPayment

3)Strategy Design Pattern:

Justification: we used this design pattern in creating the three different ways of payment which are (pay by credit card, pay by wallet, pay via cash)

We used this pattern in order to define a family of algorithms by putting them into a separate class and make their objects interchangeable

We choose this pattern specifically in this functionality as we were just having a diff types of payments but at last they are all payments

This design pattern helped us in :

1) swapping algorithms used inside an object at runtime.

2) isolate the implementation details of an algorithm from the code that uses it.

3) replace inheritance with composition.

4)achieve the *Open/Closed Principle*

Participating classes:

-Interface(iPayment)

-class CashPayment

-class creditCard

-class WalletPayment

5)Singleton Design Pattern:

Justification: we used this pattern in our database where it helps us in ensuring that a class only have one instance while providing a global access point to this instance

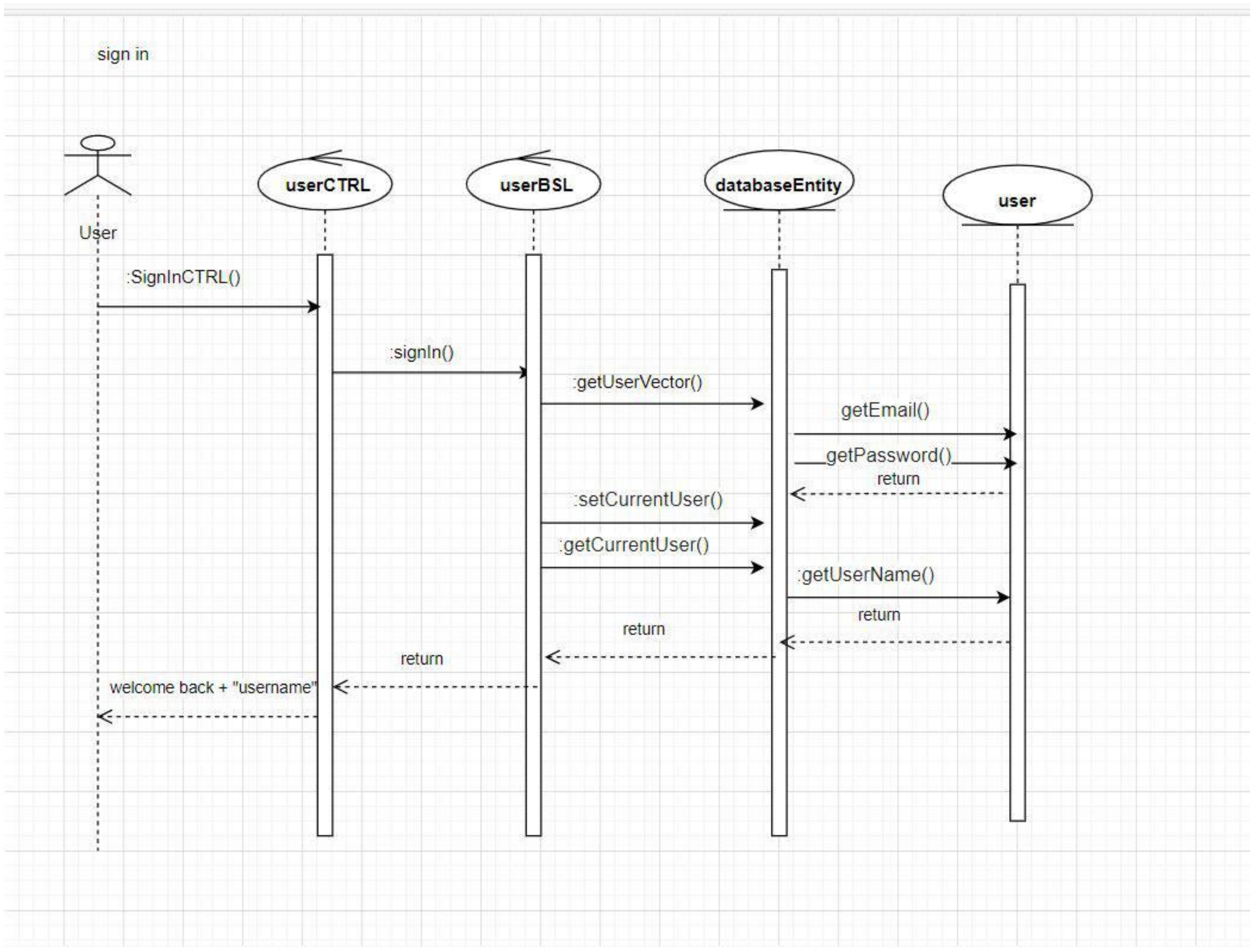
Participating Classes:

-class DatabaseEntity

SDS document

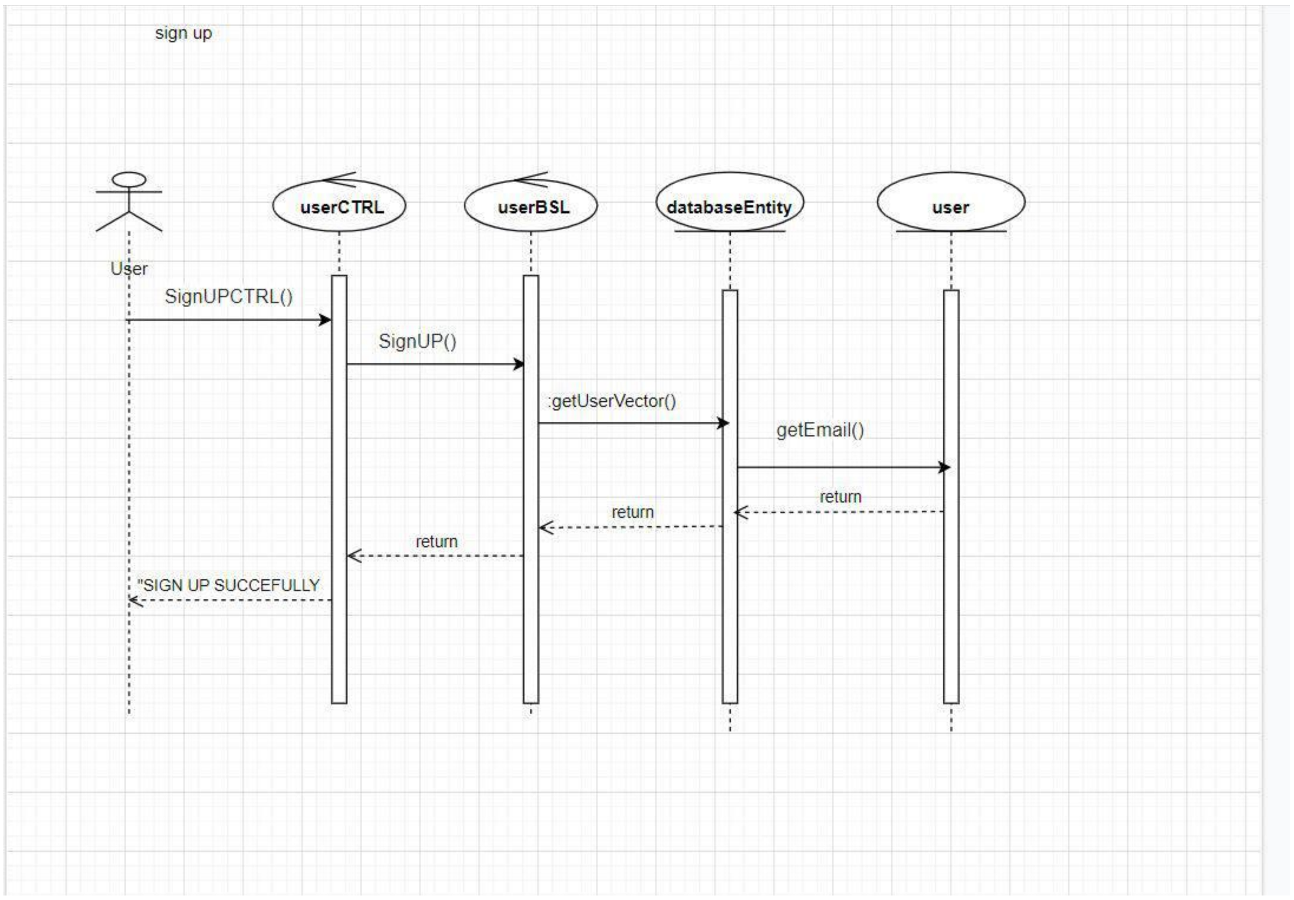
Sequence diagram design

Sign in sequence

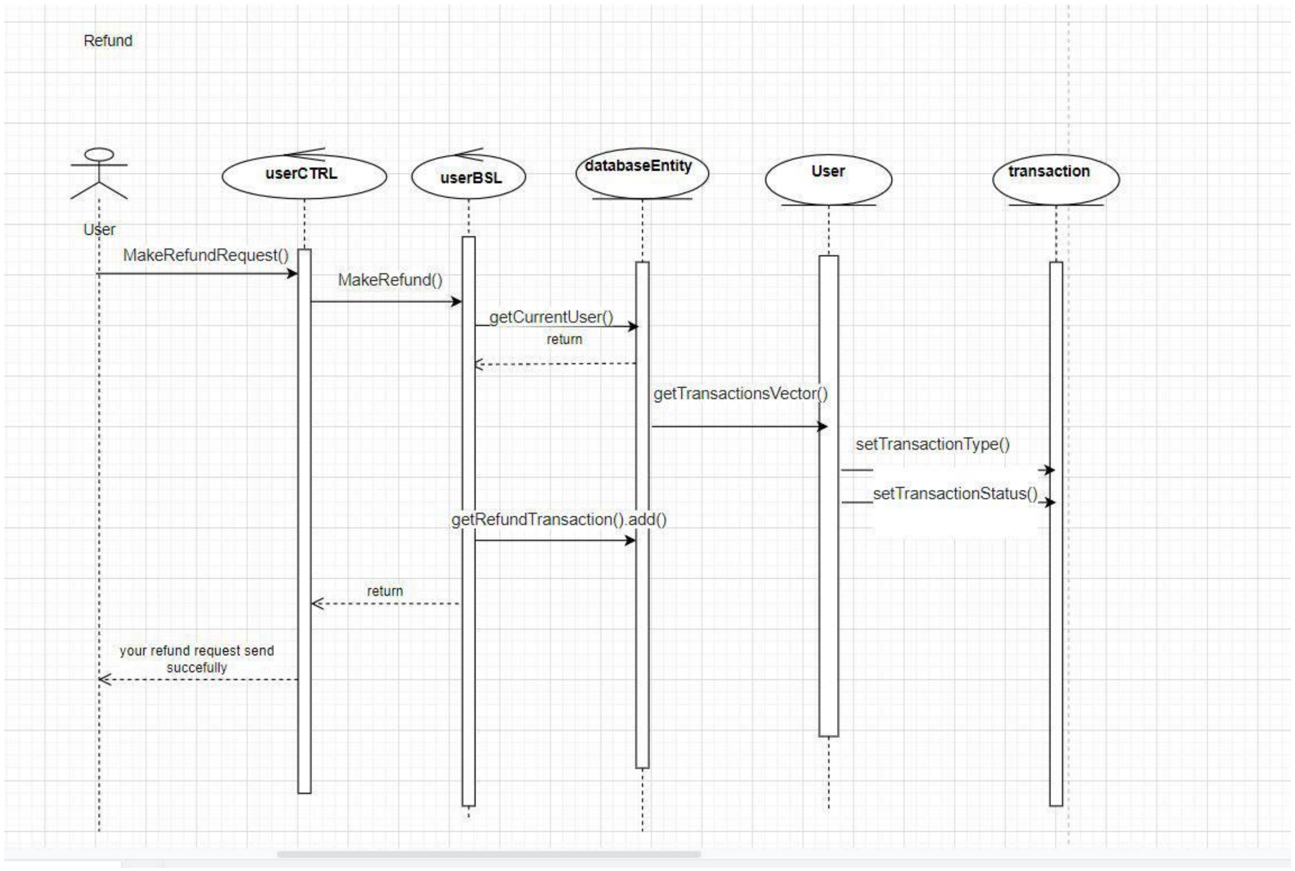


SDS document

Sign up

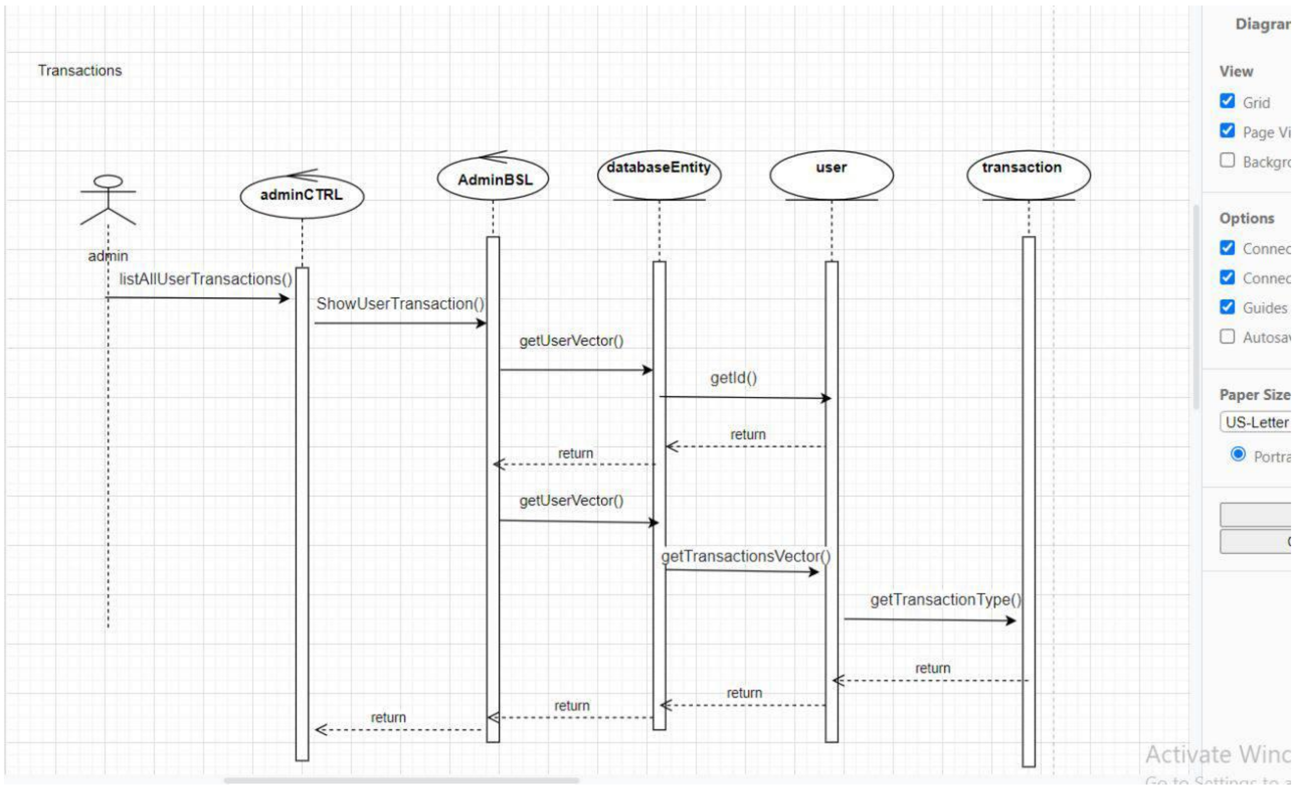


REFUND


Refund
User
userCTRL
userBSL
databaseEntity
User
transaction
MakeRefundRequest()
MakeRefund()
getCurrentUser()
return
getTransactionsVector()
setTransactionType()
setTransactionStatus()
getRefundTransaction().add()
return
your refund request send succefully

SDS document

TRANSACTION



View

- ✓ Grid
- ✓ Page Vi
- ☐ Backgr

Options

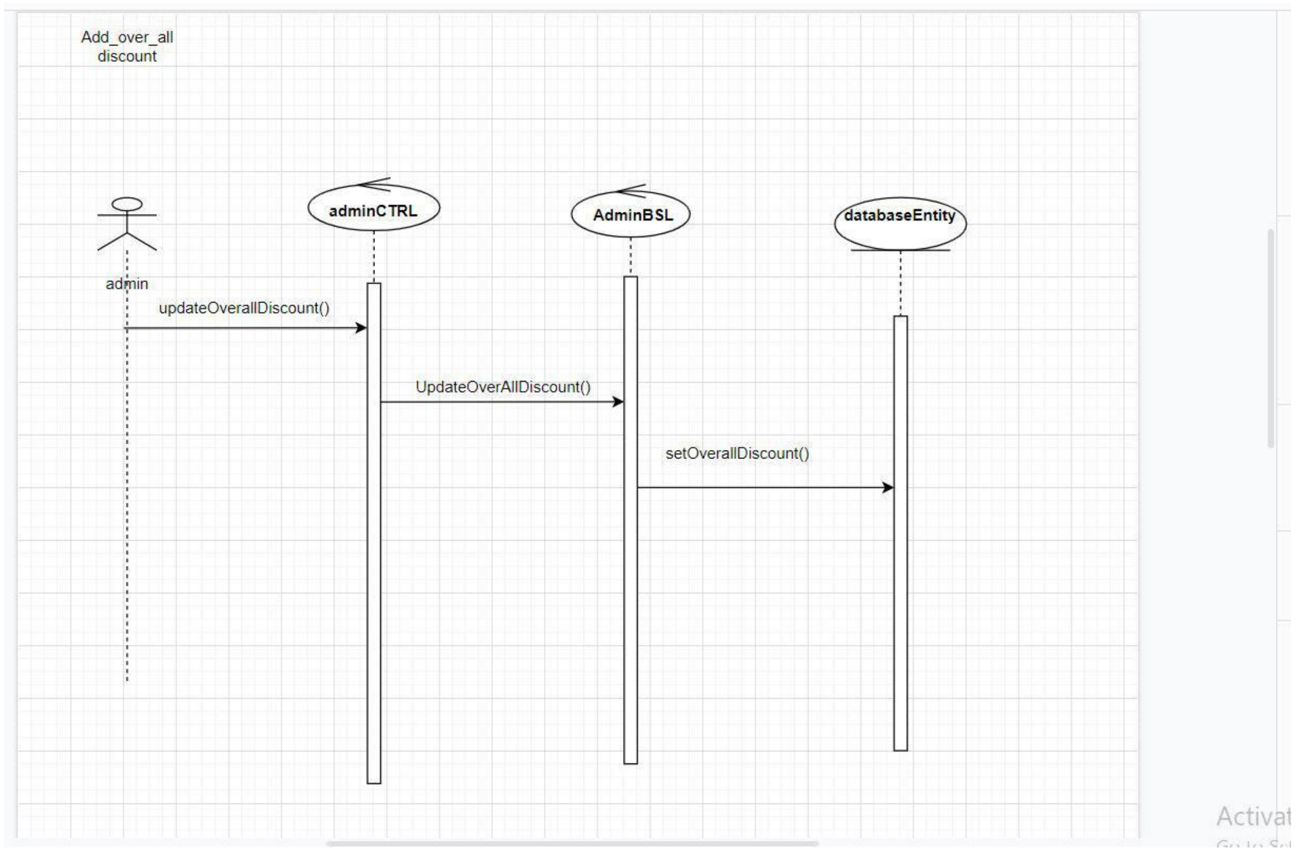
- ✓ Connec
- ✓ Connec
- ✓ Guides
- ☐ Autosa

Paper Size

US-Letter

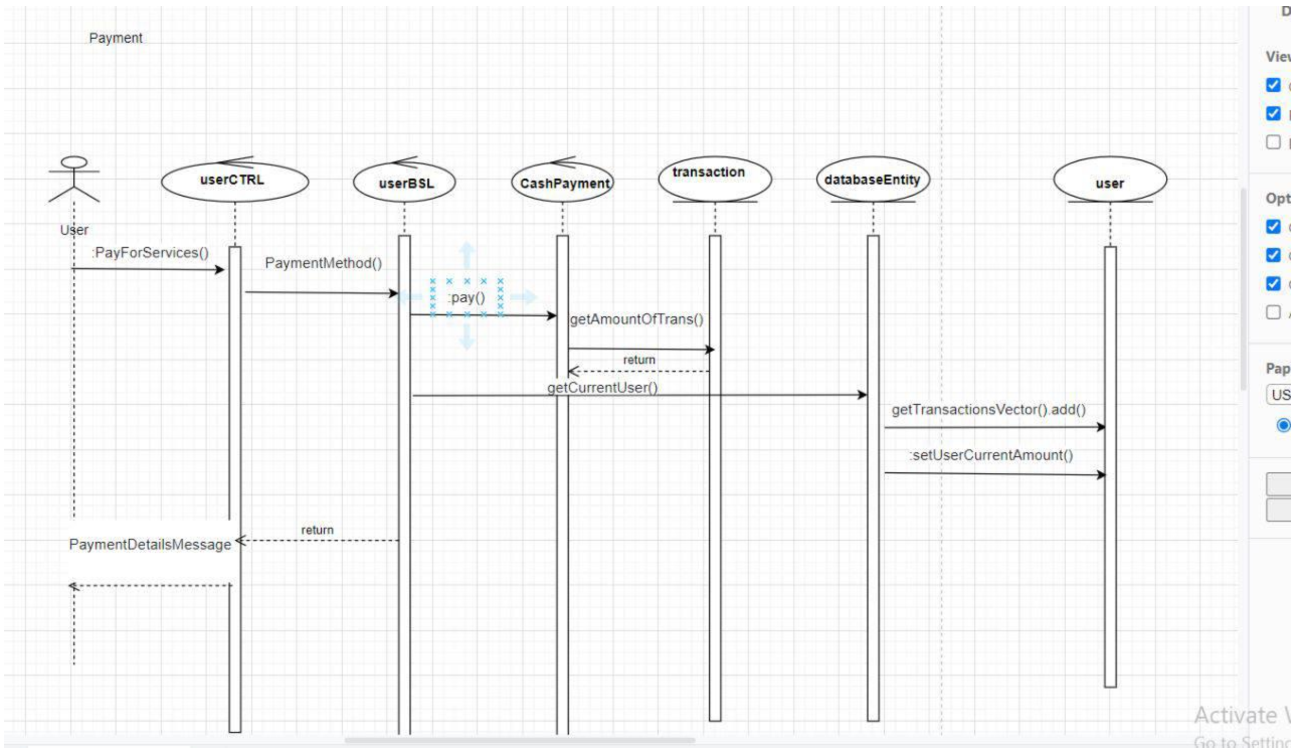
● Portra

ADD OVERALL DISCOUNT



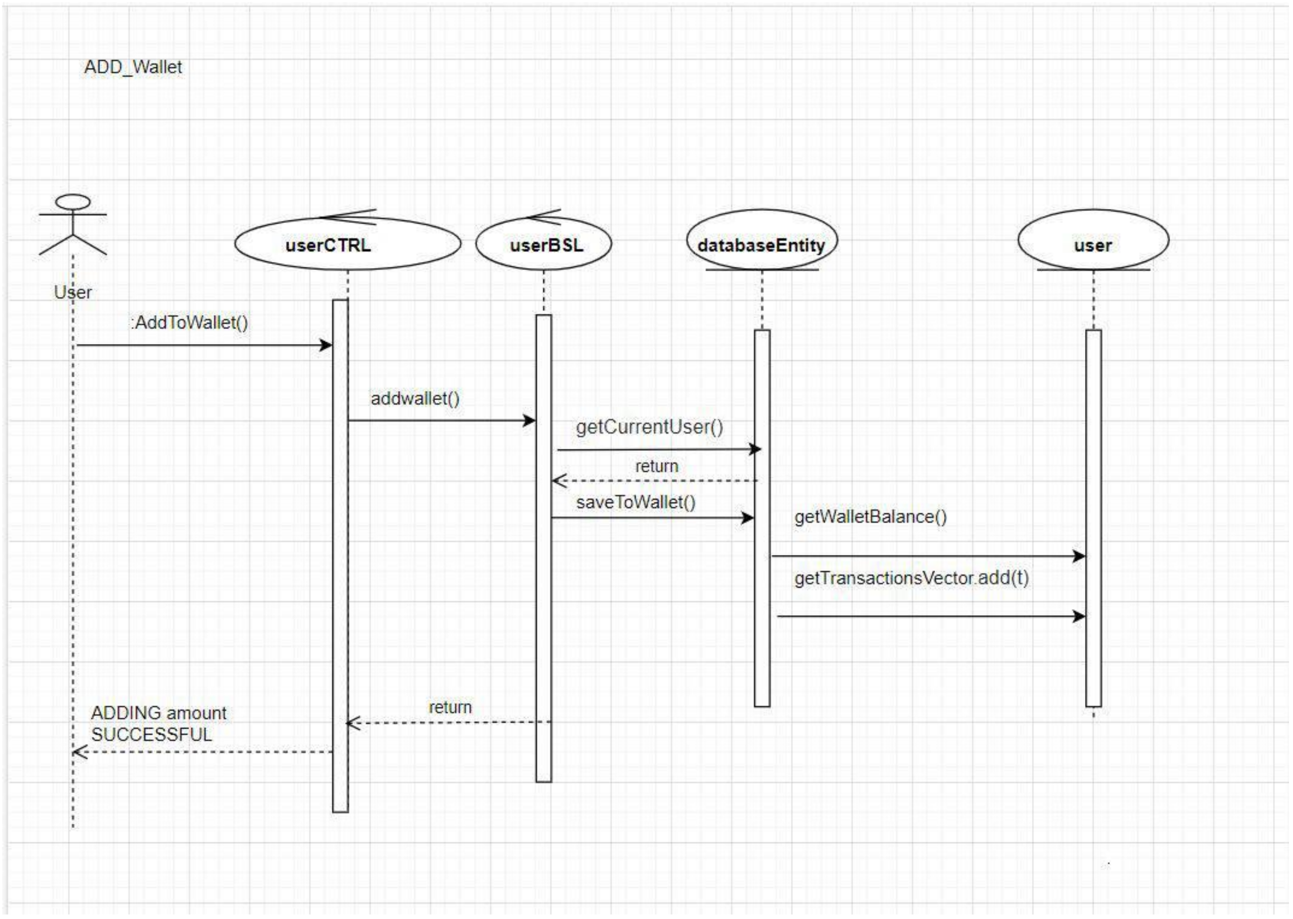
SDS document

PAYMENT

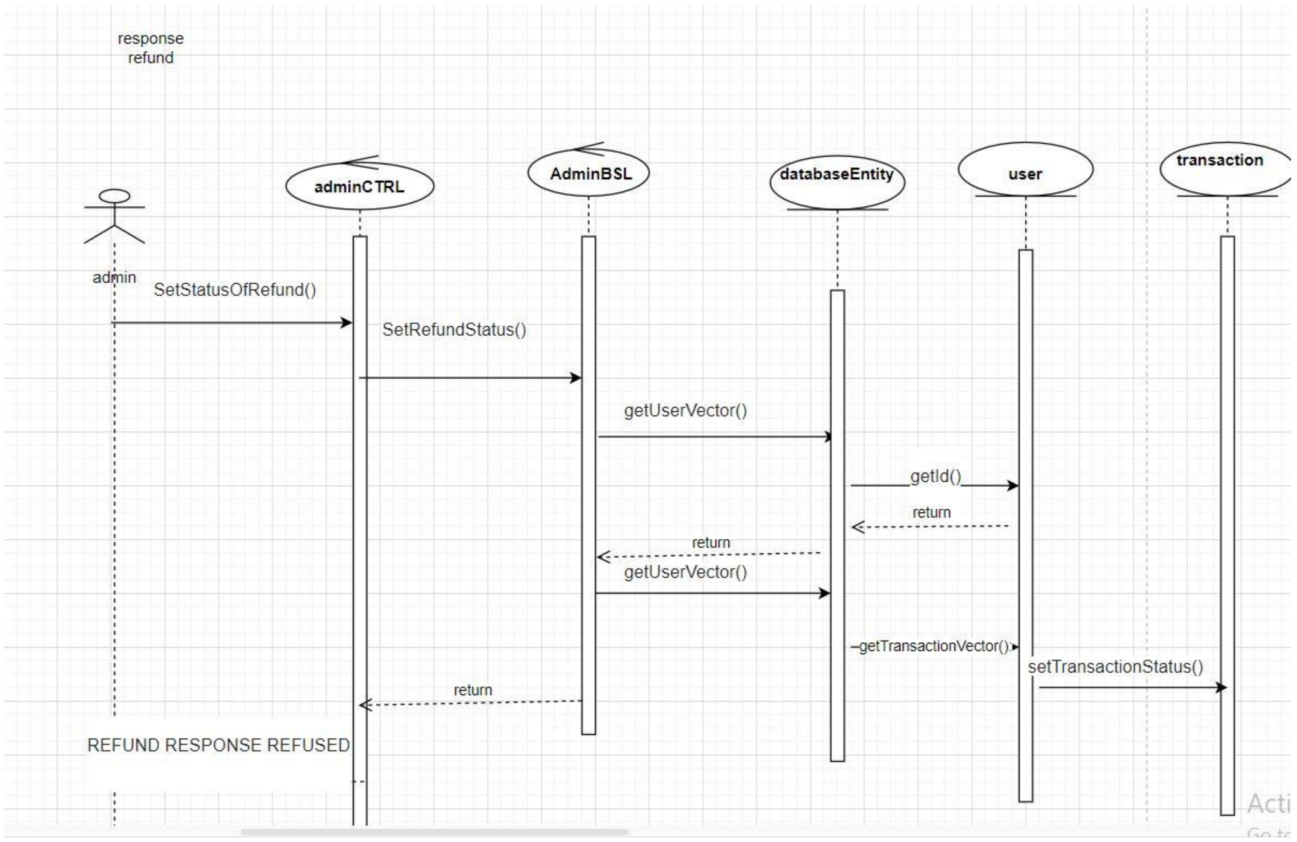


SDS document

ADD TO WALLET



RESPONSE REFUNFD



Requirements Exposure as Web Service API

Part 1: Exposed Postman Collection

PostMan.zip

Part 2:

Requirement		Exposed API	Input shape

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		this functionality was handling by this URL `(http://localhost:8080/signinuser` its post so you should choose (body) then (raw) then (jason)	{
			"email":"nada",
			"password":"123"
			}

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		this functionality was handling by this URL `(http://localhost:8080/signupuser)` its post so you should choose(body) then (raw) then (Jason)	{
			"userName":"nada",
			"email":"nada",
			"password":"123"
			}

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 use			

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
		this functionality was handling by this URL
		`(http://localhost:8080/searchForServices/we)`
		
		2)the user can choose the service he want throught this url
		`(http://localhost:8080/choiceForm/13)`
		
		3)the user input his data that was required after choosing the service throught this url
		`(http://localhost:8080/formInput)`
			its Get so you should write after /searchForServices/(service you search for)
			
			2) its Get so you should write after /choiceForm/ (the id of your choosen service which appear in the search list)
			Ex:13
			3) its post so you should choose body then raw you dont have to choose jason input should look like that ["13","01123456789","BAHY A","11234","2"](where the first index is for the service id (which is constant at any form))(the other inputs changes according to the requirments of each service form)

The user can pay for any service in the system. The system should prompt the user to		**this functionality was handling by two parts part 1)show the user a list of payment methods by this URL**
		`(http://localhost:8080/WayOfPaymentMethod)`
		`its Get method`
		part2)enter the way he want to pay by and this by choosing one of the choices in the past form throught the id by this URL
		`(http://localhost:8080/PaymentMethod)`
			Part2 is post so you should choose body then raw you dont have to choose jason input should look like that(3)
			which is the id of the payment method in the previous form (if the user

the payment form when the user asks to pay for any service			choose any other number it will pay by credit card by default)
The user can ask for a refund for any complete transaction to any given service		**this functionality was handling by this URL** (http://localhost:8080/MakeRefundRequest) **he can know the transaction id throught this url** (http://localhost:8080/ViewCurrentUserPaymentTransactions) its Get method	**its post so you should choose body then raw you dont have to choose jason input should look like that 3** (which is the id of transaction which the user want to make refund for)he can know the transaction id throught this url
The system maintain a wallet balance for each user. The user should be able to add any funds to the wallet		**functionality was handling by this URL** (http://localhost:8080/AddToWallet)	**its post so you should choose body then raw you dont have to choose jason input should look like that** **400 (which is the amount you want to add to wallet)**
The user should be able to check any discount for any service in the system		**functionality was handling by this URL** http://localhost:8080/DiscountWithID	**its Get method which show the available discount in the system**
the user can show all his information		**throught this url** (http://localhost:8080/showYourData)	**its Get method**
The admin should be able to		**this functionality was handling by this URL** (http://localhost:8080/AddOverallDiscount)	**its post so you should choose body then raw you dont have to choose**

add discounts to the system			jason the input should look like that 10 (which is the amount of overall discount)

To add spacific discount by admin this functionality		this functionality was handling by this URL
		`(http://localhost:8080/AddingSpecificDiscount)`
		
		in order to know the id of service this functionality was handling by this URL(get method)
		`(http://localhost:8080/listOfSpecificDiscountWithID)`
			its post so you should choose body then raw you dont have to choose jason input should look like that
			
			[1,10] (the first number is the id of the service you want to add spacific discount on it ,the secound is the amount of this discount)

The admin should be able to list all user transactions		this functionality was handling by this URL `(http://localhost:8080/listAllUserTransactions/1/1)`	its Get so you should write after/listAllUserTransactions/ (user id)/(type of transaction id)
		the admin could list the types of transactions and know the id by this URL `http://localhost:8080/TransactionsTypes`	is Get method
		the admin could list all the users info and know there id throught this URL `http://localhost:8080/UsersData`	is Get method
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		this functionality was handling by this URL `(http://localhost:8080/listOfRequestTransactions)` throught this it he know the users request to accept or reject this was handle throught this URL	is Get method its post so you should choose body then raw you dont have to choose jason input should look like that [3,1] (where the first number is the refund request id)(the secound

admin should be able to accept		(http://localhost:8080/SetStatusOfRefund)	number (1) for reject,(2) for accept)

Github repository link

https://github.com/mohamedanas00/Software_Architecture_PhaseOne

SOME INSTRUCTIONS TO OPEN THE CODE :

TO OPEN THE CODE ON ECLIPSE

1)CHOOSE FROM FILE IMPORT

2)CHOSSE EXISTING MAVEN PROJECT

3)THEN PUT THE PATH OF THE FILE WHERE YOU DOWNLOAD IT

4)YOU SHOULD HAVE JAVA 17 TO COULD RUN THE CODE

5)PLEASE CHECK POSTMAN ZIP IN THE FOLDER TO SHOW COLLECTION