**Cairo University Faculty of Computers and Artificial Intelligence**



**Software design specification document**

**2022**

**The Crooks**

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Email** |
| 20200543 | Mostafa Mohammad | mostafakamel6001@gmail.com |
| 20210609 | Abd ElRahman ElHossini | abdelrahmanelhossini55@gmail.com |
| 20160161 | Omar Ali | Omaralitaha98@gmail.com |
| 11422022470433 | Sara Hany | sarahani600@gmail.com |

**Contents**

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

[Class diagram design 2](#_30j0zll)

[Class diagram Explanation 3](#_1fob9te)

[Sequence diagram design 3](#_3znysh7)

[Github repository link 4](#_2et92p0)

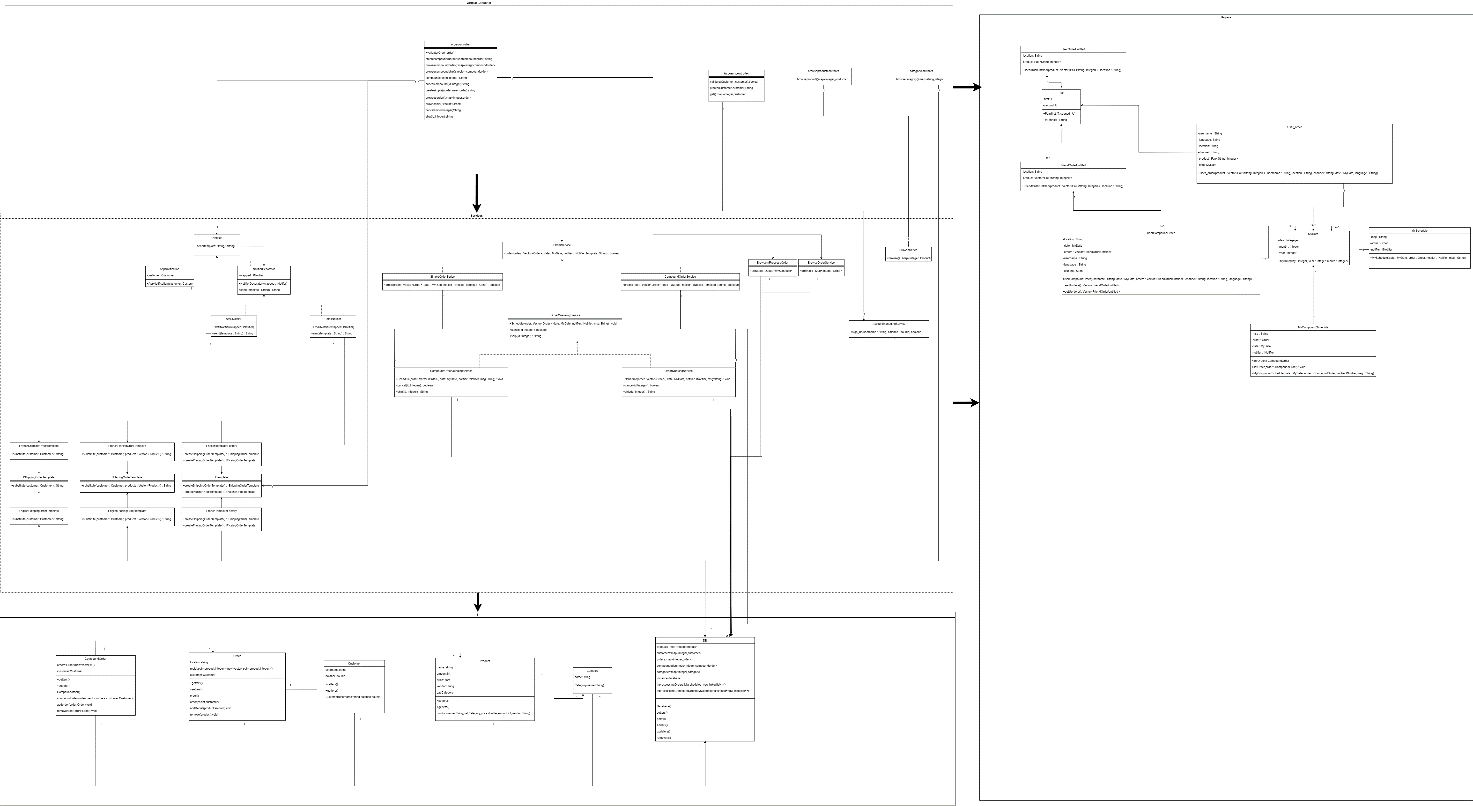
# Assumptions

# We assumed that customer can make compound order for himself and users outside of the system, since if the user in the system they can make simple order for themselves.

* The process customer places order. Once order is placed it takes place in the queue. And when the shipment is issued it should be removed from the queue and the prices is reduced from the customer.
* We have set the calculations between customers in compound order with dummy calculations like 0.25 \* number meters in distance.

# Class diagram design

* If this image is not clear please use the image provided with the project files and zoom in and it is clear.



# Class diagram Explanation

* We have used the abstract factory which is a creational design pattern that allows us to define different factories for each languages like French and English in our case(ITemplate, EnglishTemplateFactory, FrenchTemplateFactory) and for each we can create different products (IShippingOrderTemplate, IPlacingOrderTemplate, PlacingOrderTemplate, FrenchPlacingOrderTemplate, EnglishPlacingOrderTemplate, FrenchShippingOrderTemplate, EnglishShippingOrderTemplate) for our templates.
* We have used the decorator design pattern, behavioural, to allow us to use either an SMS template or Email template(SMSNotifer, EmailNotifier) by defining and interface for all notifiers (INotifer), and there is the decorator(NotifierDecorator) which wraps an object of The (INotifier) and the base componenet is the AppNotification which carries object of customer that will be notified. They all implement the send method to send the message across intended services.
* We have used the Strategy Design Pattern(SimpleOrderService, CompoundOrderService, IOrderService) is a behavioural design pattern that allows to define different algorithms for each ordering service whether it is compound or simple. This design pattern is also applied on notification queueing (IOrderQueuingService, CompoundOrderQueuingService, OrderQueuingService).
* Singleton Pattern: this pattern makes makes sure that one instance of the object exist only one time just like global variables in some languages. The class that uses singleton pattern is our DataBase.

# Requirements Exposure as Web Service API

The postman collection is put with the project files by the name of Shopping.postman\_collection.json.

|  |  |
| --- | --- |
| Requirement | API |
| A list of all the products currently avafiable for purchase should be displayed. | <http://localhost:8080/product/get> shows all products in the system |
|  | <http://localhost:8080/account/get> shows all the accounts in the system |
| A customer should be able to create an account and put a specific balance using that account.Such balance would be utilized during future purchasing operations. | <http://localhost:8080/account/create> create account give a body of username and balance  {  "username":"hamada",  "balance":20000  } |
| Furthermore, a count of the remaining parts from each category should be available. | <http://localhost:8080/category/get> shows all the category and the number of available products in that category in our system. |
| You should be able to list all the details of both simple orders. | <http://localhost:8080/order/simple/get> shows all the simple orders that were ordered |
| You should be able to list the current content of that Queue. | <http://localhost:8080/order/simple/get_notification> shows all simple orders that are scheduled in the notification queue |
| A customer can place a simple order, where such an order would include a single product or several products. | <http://localhost:8080/order/simple/create> creates a simple order given product list of pairs of the first being the product name and the second being the amount, username the name of the user that will make order, location which is the location that we sent order to, channel the language of the templates, date the date on which order should arrive.  {  "product": [  {"first":"Laptop", "second":1},  {"first":"Apple Smart Phone","second": 2},  {"first":"T-Shirt","second": 3}  ],  "username":"hamada",  "location":"giza",  "channel":"email",  "language":"en",  "date":{  "day":1,  "month":2,  "year":2023  }  } |
| Bonus part (Only the part highlighted in yellow is bonus):Customers can cancel an order placement, or cancel only its shipping within a pre-configuredautomated duration. Such cancellation should update appropriately within the system. | <http://localhost:8080/order/simple/cancel?id=0> cancel the simple order given its id. 0 is the example of such. |
| For created notifications, you should implement a "notifications Queue", where you insert | <http://localhost:8080/order/simple/ship?id=0> ship the simple order from the notification queue given its id. 0 is the example of such |
| You should be able to list all the details of both compound orders. | <http://localhost:8080/order/compound/get> shows all the compound orders that were ordered |
| You should be able to list the current content of that Queue. | <http://localhost:8080/order/compound/get_notification> shows all compound orders that are scheduled in the notification queue |
| a customer can make a compound order, where that order caninclude various orders headed to near-by locations, in addition to his own products, to lessen theshipping fees. | <http://localhost:8080/order/compound/create> creates a compound order given a username, language, channel, location, date and a list of orders in which the order has a product array and location attribute. The product has list of objects in which an object has first, name of product, second, amount of product.  {  "orders": [  {  "product": [  {"first":"Laptop", "second":1},  {"first":"Apple Smart Phone","second": 2},  {"first":"T-Shirt","second": 3}  ],  "location":"giza"  },  {  "product": [  {"first":"Laptop", "second":1},  {"first":"Apple Smart Phone","second": 2},  {"first":"T-Shirt","second": 3}  ],  "location":"moneib"  }  ],  "username":"hamada",  "location":"giza",  "channel":"email",  "language":"en",  "date":{  "day":1,  "month":2,  "year":2023  }  } |
| Bonus part (Only the part highlighted in yellow is bonus):Customers can cancel an order placement, or cancel only its shipping within a pre-configuredautomated duration. Such cancellation should update appropriately within the system. | <http://localhost:8080/order/compound/cancel?id=0> cancels a compound order given the id. The 0 is example in this case. |
| For created notifications, you should implement a "notifications Queue", where you insert"notifications" that ARE TO BE SENT. | <http://localhost:8080/order/compound/ship?id=0> ships an compound order from the notification queue. 0 is example of id in this case. |

# Github repository link

* https://github.com/Deasoul315/Shopping