**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)

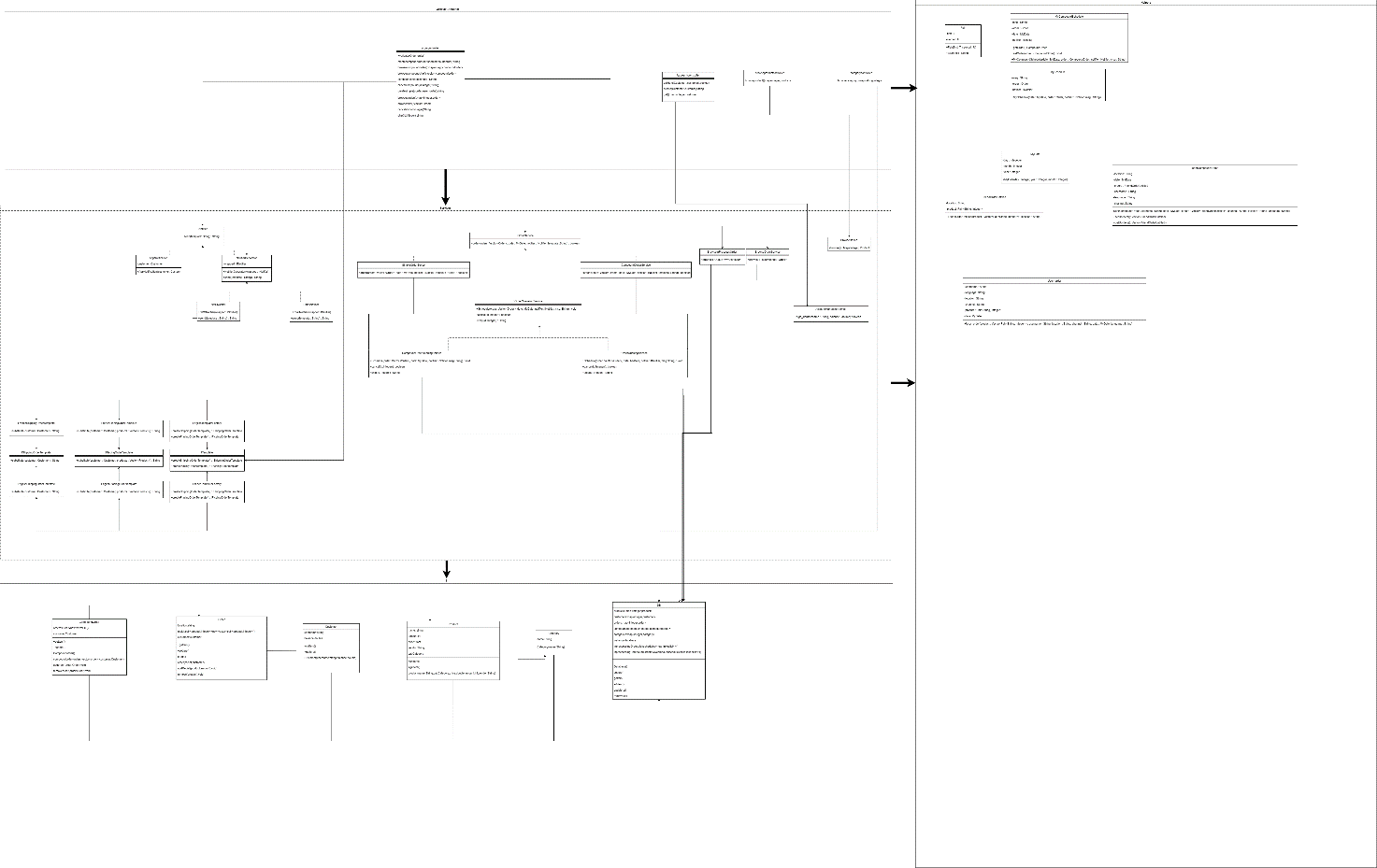
[Class diagram Explanation 3](#_1fob9te)

[Sequence diagram design 3](#_3znysh7)

[Github repository link 4](#_2et92p0)

# 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

|  |  |
| --- | --- |
| Requirement | API |
|  | http://localhost:8080/product/get |
|  | http://localhost:8080/account/get |
|  | http://localhost:8080/account/create |
|  | http://localhost:8080/category/get |
|  | http://localhost:8080/order/simple/get |
|  | http://localhost:8080/order/simple/get\_notification |
|  | http://localhost:8080/order/simple/create |
|  | http://localhost:8080/order/simple/cancel?id=0 |
|  | http://localhost:8080/order/simple/ship?id=0 |
|  | http://localhost:8080/order/compound/get |
|  | http://localhost:8080/order/compound/get\_notification |
|  | http://localhost:8080/order/compound/create |
|  | http://localhost:8080/order/compound/cancel?id=0 |
|  | http://localhost:8080/order/compound/ship?id=0 |

# Github repository link

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