CSCI-4448/5448
Final Project Report

Team : **Antara Kolar**
Shriya Rodi
Hariram Natarajan

Project Title**: #39 Quickshop**

1. List the features that were implemented (table with ID and title).

User Requirements:

ID	Requirement	Topic Area	Actor	Priority
UR-01	User should be able to create an account to get started	Account Management	Customer and Seller	Critical
UR-02	User should be able to login	Account Management	Customer, Admin and seller	Critical
UR-03	Customers can choose the store	Order Management	Customer	critical
UR-04	Customers can choose the delivery or pick up option.	Order Management	Customer	High
UR-05	Customers can search for product by name or category	Product search	Customer	high
UR-06	Customers can add or remove items from cart	Order management	Customer	high

UR-08a	Sellers can view products.	Product management	Seller	Med
UR-09	Customers can make payment for their orders.	Payment Processing	Customers	Critical
UR-11	Users can view placed orders	Order Viewing	Sellers, Admin and customers	Critical
UR-12	Sellers can remove products from product list	Product management	Sellers	Critical
UR-13	Admin can add or remove accounts	Account management	Admin	Critical
UR-14	Sellers can add products to the product list	Product Management	Sellers	high
UR-15	Customers can view products in cart	Order Management	Customers	High

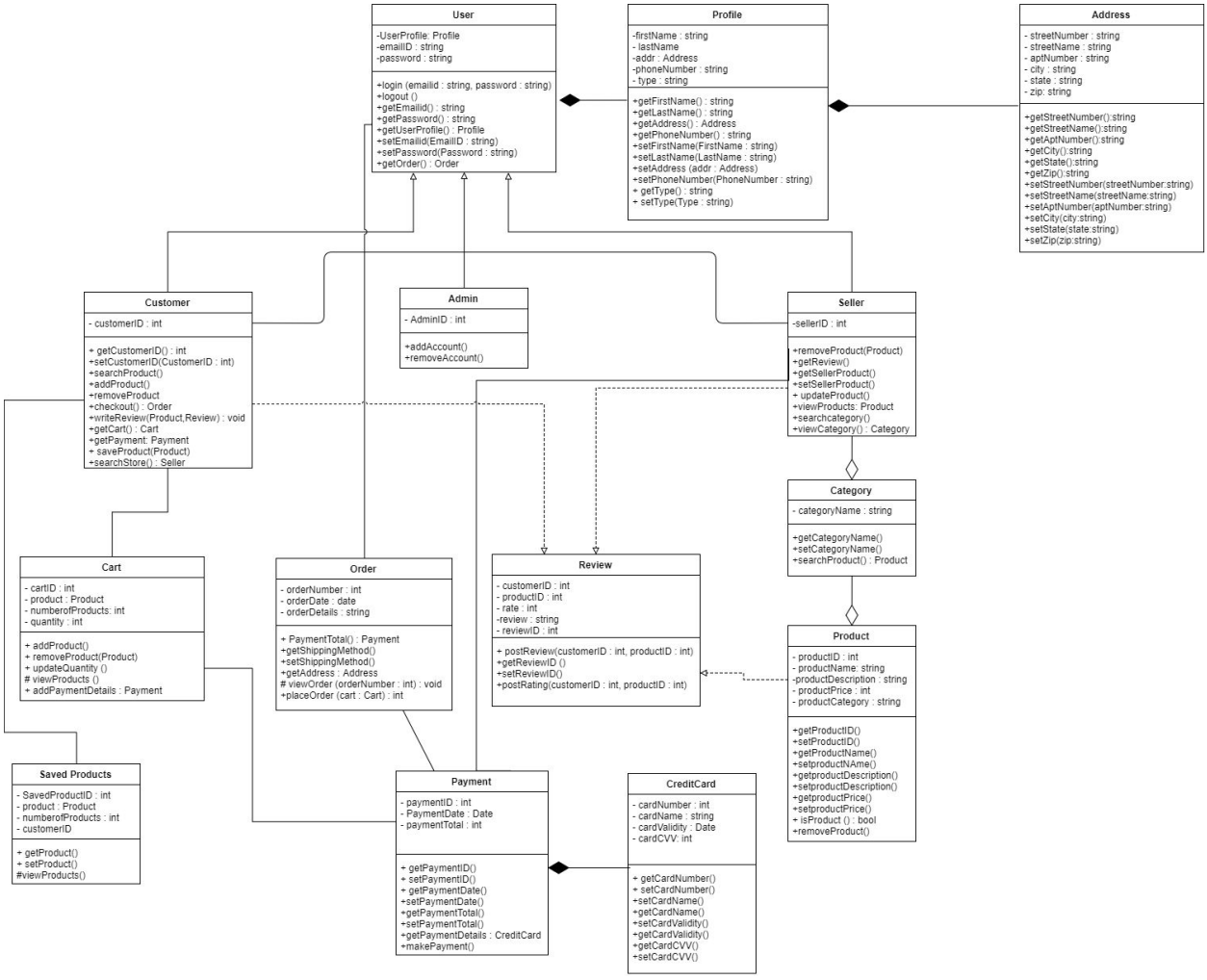
2. List the features were not implemented from Part 2 (table with ID and title).

UR-07	Customers can save products	Order management	Customer	low
UR-08b	Sellers can read reviews.	Product management	Seller	Med
UR-10	Customers can read, rate and provide product reviews	Review	Customers	high
UR-16	Sellers can modify products from the product list	Product management	Seller	high

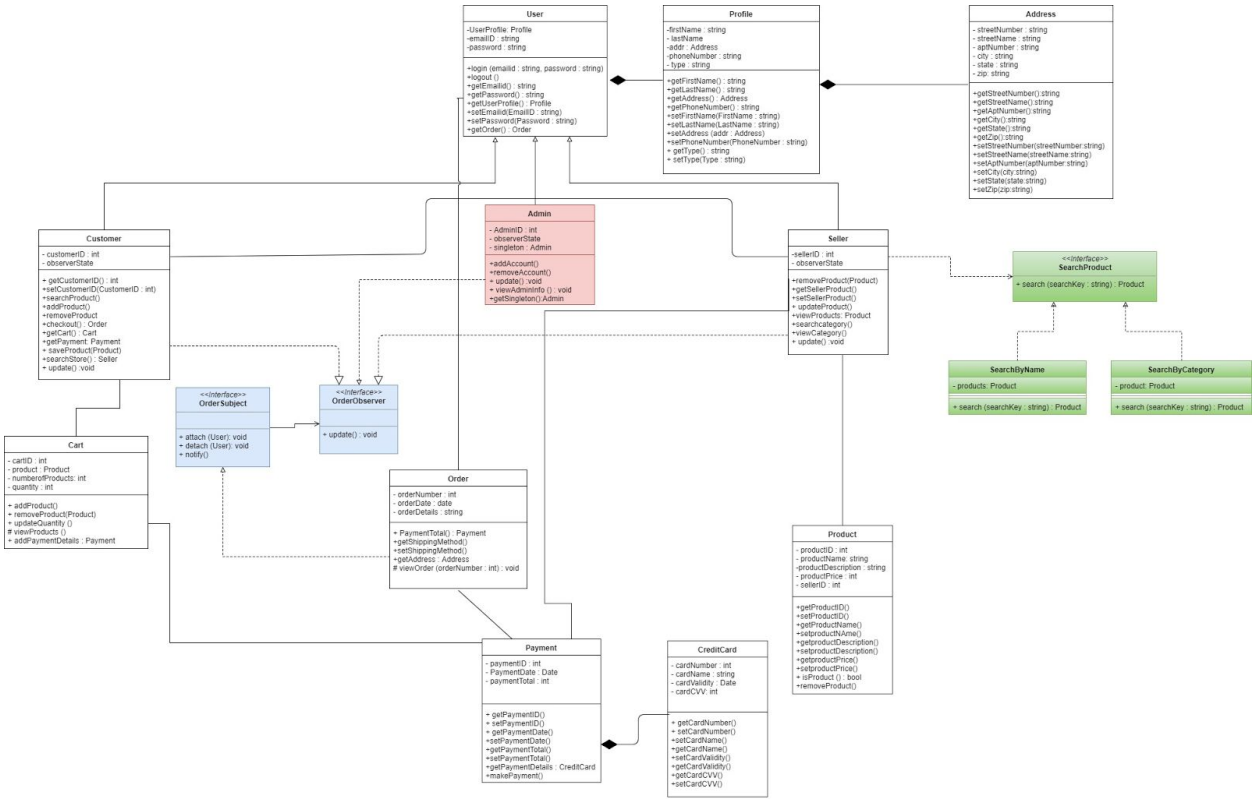
UR-17	Users can update profile.	Account management	Seller and customer	high

3. Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the designup front helped in the development.

Part 2 Class Diagram:



Final Class Diagram:



There have been many changes in final class diagram compared to part2 class diagram. We have refactored using three design patterns i.e. Singleton, Observer and Strategy design pattern.

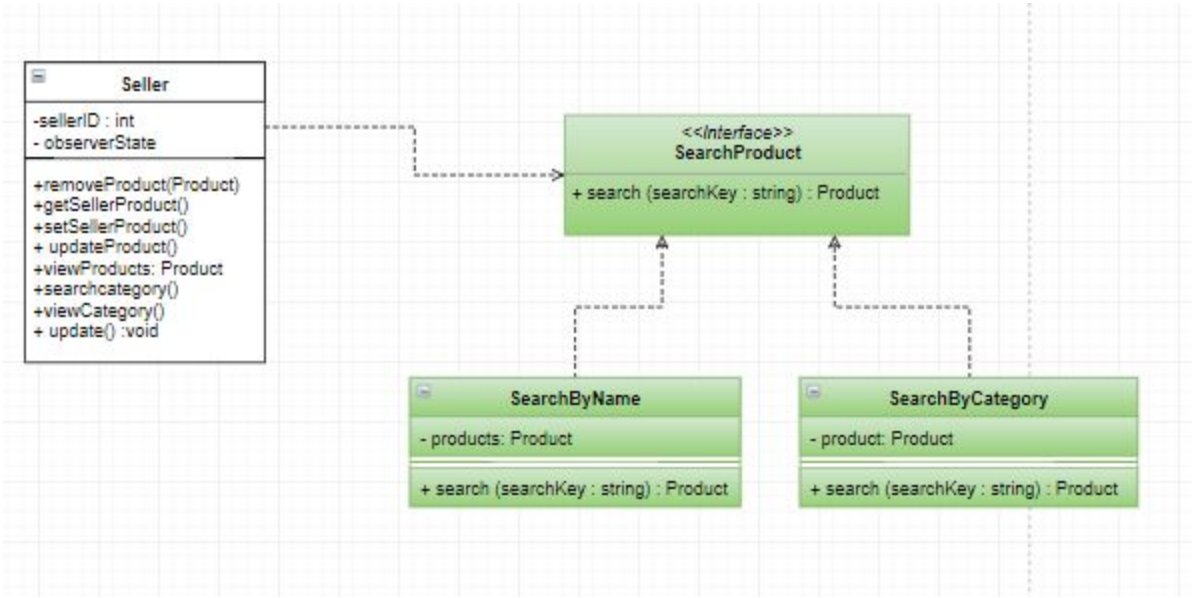
Some of the classes like category class were removed as they were not necessary and we thought of implementing design patterns which made the design and implementation simpler. Changes were made in variables and methods called in some of the classes depending upon the redundant classes which were removed. We have used the principle of high cohesion and low coupling to design and implement software. We were also planning to implement use cases like customer saving the product for further references and writing review which needed classes like saved products and review. Due to time constraints, we were not able to implement these classes. Planning the design in advance by working on class diagrams and UML diagrams was very helpful to implement the code and build the software.

4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? Show the classes from your class diagram that implement each design pattern (eachdesignpattern as a separate image in the PDF). If not, where could you make use of design patterns in your system? Show a class diagram of how you could implement each design pattern and compare how it would change from your current class diagram?

Yes, we made use of three Design Patterns in the implementation of our project that are described below.

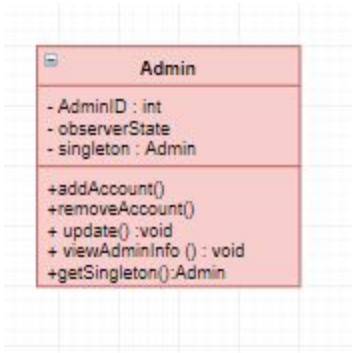
Strategy Design Pattern :

We made use of **Strategy Design Pattern** for our search product functionality. Whenever the customer wants to search for any product, he can search in two ways , a.Search by name and b. Search by category.
So at runtime, depending on the type of search specified by the user, the system makes use of that particular algorithm. We created a ***SearchProduct*** interface that captures the abstraction, and their implementation details are specified in the respective derived classes namely ***SearchbyName*** and ***SearchbyCategory***.
This helped us to reduce coupling by programming to the interface.



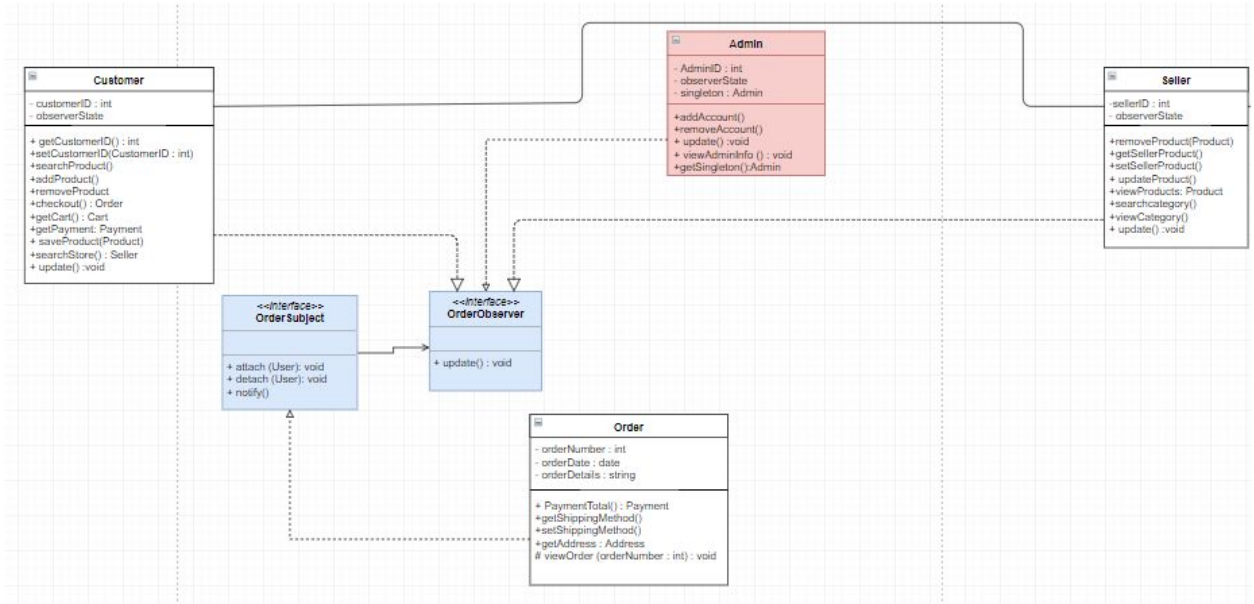
Singleton Design Pattern

We made use of **Singleton design pattern** for our ***Admin*** Class as there is only one admin account in our project. So this helped us ensure that no more instances for the Admin Class would be created.



Observer Design Pattern:

We used **Observer Design Pattern** for the ***Order*** class. Whenever an order is placed by the customer, we wanted it to be viewable by the customer, seller and the admin. So as the order gets placed, the observers customer, seller and admin get notified of the state change. The ***Order*** is the concrete subject class. Customer, Seller and Admin are the observers. The ***Ordersubject*** class keeps track of the number of observers and notification. ***OrderObserver*** class is the interface to update the state.



5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

During the process of analysis and design we learned about the systematic flow to design and implement a system. Initially, we realized how important it is to select a good project topic and work on major requirements for the project. Selecting proper user requirements, functional requirements and nonfunctional requirements helped us to design simple use cases for our project QuickShop. This helped us during implementation to divide use cases amongst ourselves equally and then integrate them later.

Secondly, using UML diagrams helped us to understand flow in each use case and eased our work while writing codes. During refactoring, using Design Patterns in the implementation of our project helped to us to capture the abstraction by encapsulating the variations. We could also reduce coupling between classes and follow the open closed principle. All these things made our design better. This was great learning experience not only to understand object oriented programming but also to know about industrial requirements and system needs.