Uno - Software Store

Instructor - Jia Zhang
Team 1 Yuejun Ma
Litong Xiao
Haoran Li
Sriharsha Bandaru

[18653] Software Architecture and Design, Spring 2019

Roadmap

Introduction - Uno

- Uno is an online software sales platform that allows software developers and customers to trade software online
- Registered software developers can post products on Uno and set prices. Customers can purchase softwares already on the site
- In addition, the website provides a social network for customers and developers

Motivation - Connecting developers and users

- High number of software developers in the world (we may be one of them) building software, and also higher number of people who use these software
- Most of these useful *software difficult to find for users* and developers *have difficulties in reaching out to users*
- Users want to be able to find all the softwares they need on a single platform and *buy them legally and conveniently*
- Built Uno which is more *focus on software trading*

Related Work

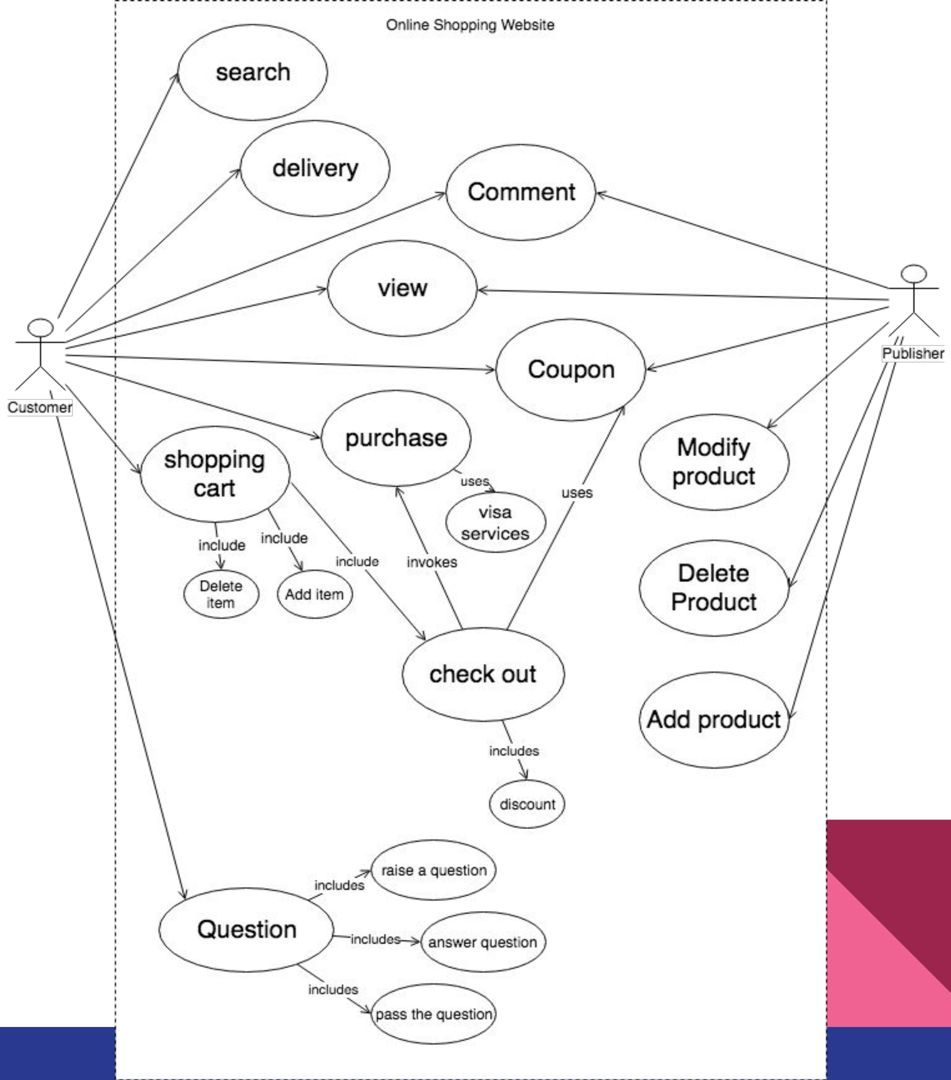
- Platforms like amazon.com and ebay.com provide a way to publish products and purchase them
- These platforms are much more generic
- Our platform focuses specifically on software products

System Design

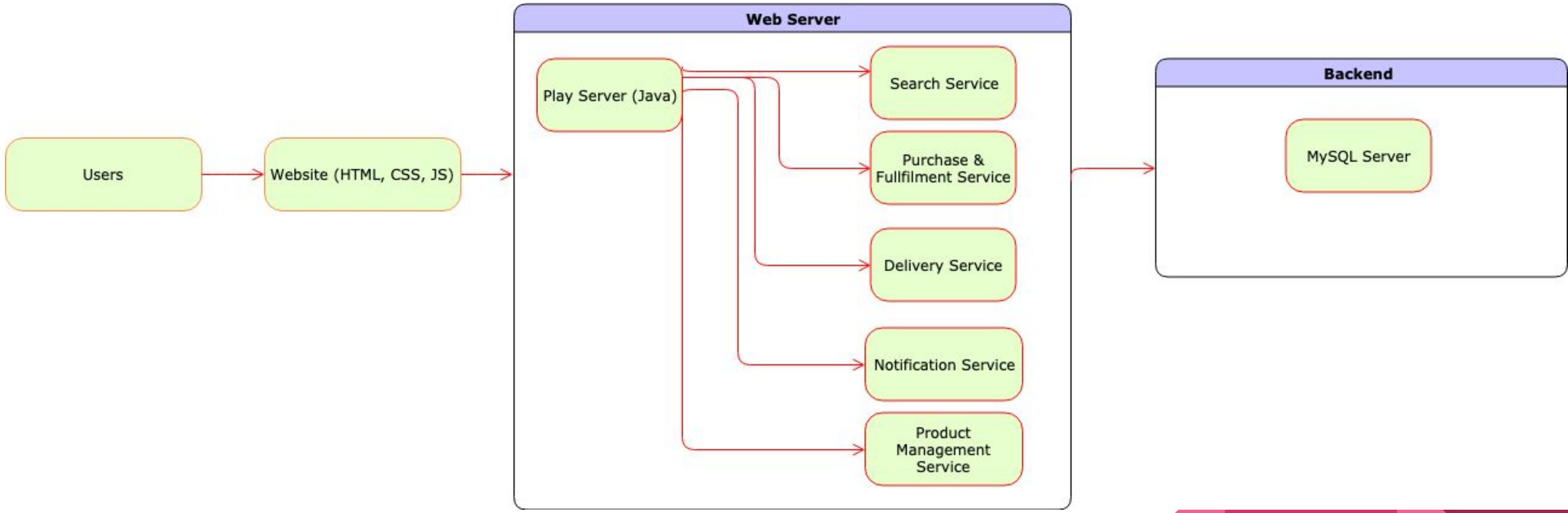
Use cases

- ❖ Search
- ❖ Delivery
- ❖ Comment
- ❖ Coupon
- ❖ Shopping cart
- ❖ Purchase
- ❖ Post question

- ★ Comment
- ★ Search
- ★ Manage coupons
- ★ Manage products



Architecture view - Monolithic



System Implementation

User Interface

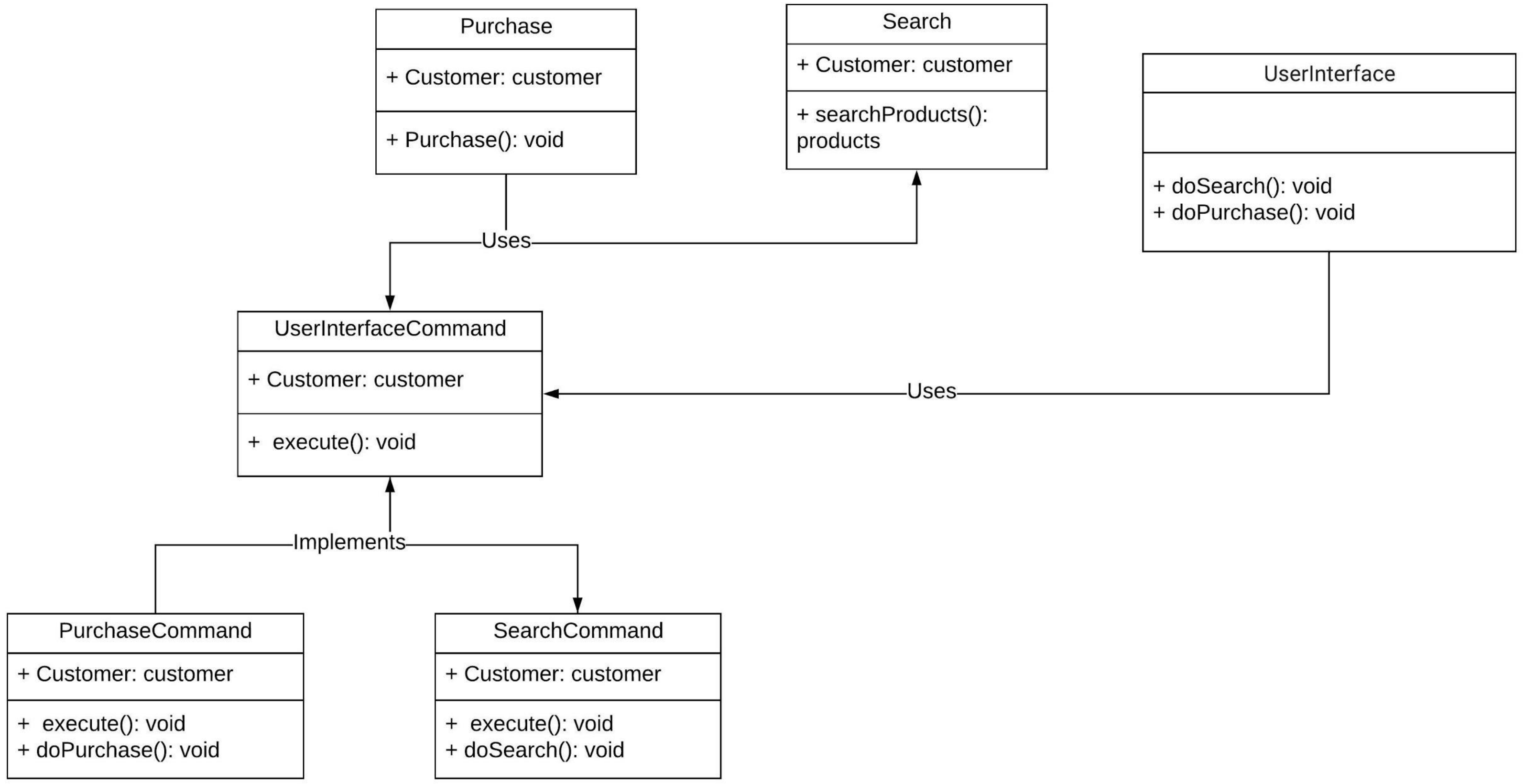
- Customer will be using the user interface to perform actions such as search, purchase, etc.

- **Design Pattern**: Facade Pattern and Command Pattern

- Facade pattern to hide the internal details of the functions from the user

- Command pattern for internally performing the right function (search, purchase, etc.) for the user

Implementation



Purchase and Discount System

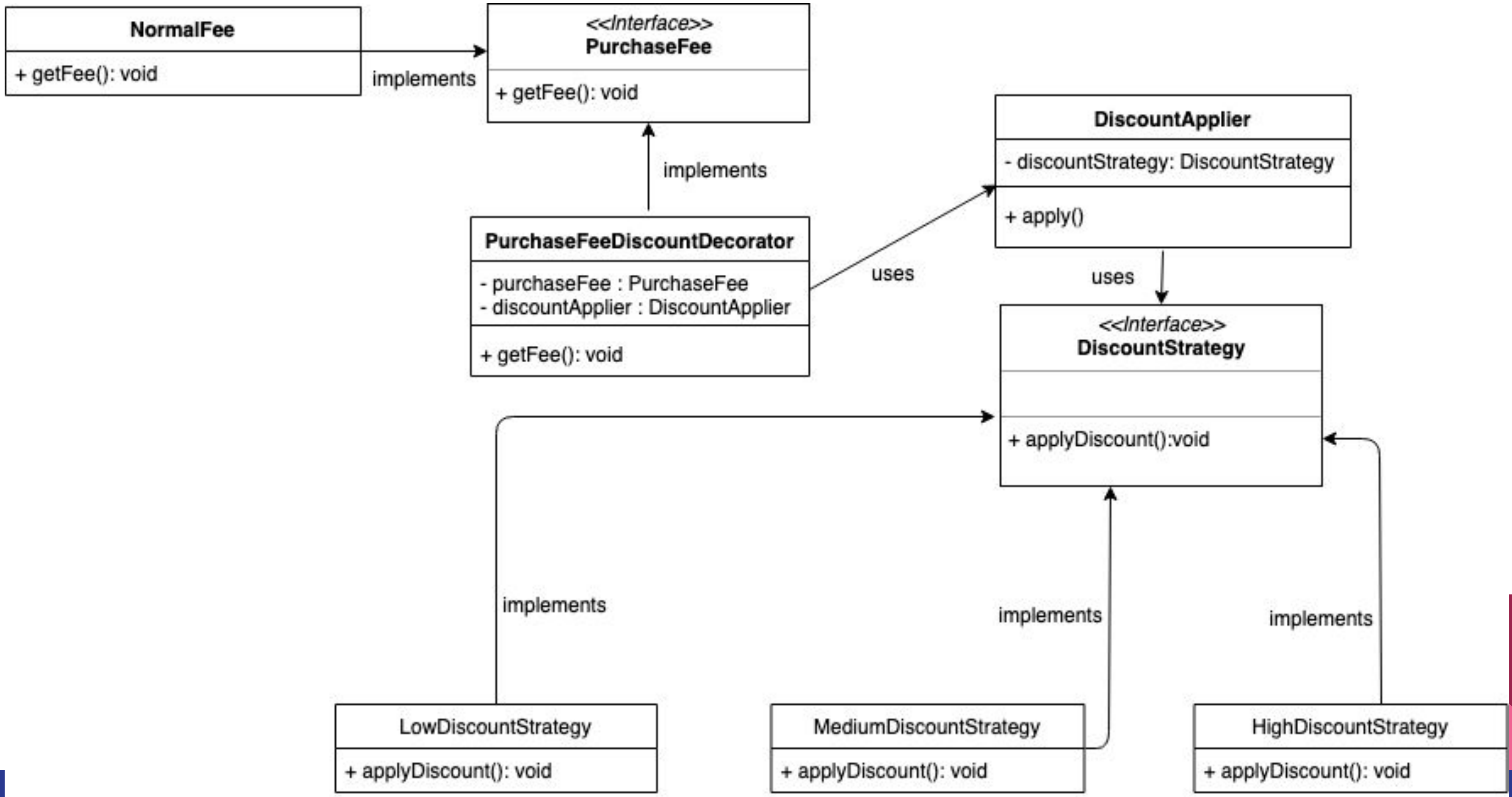
- Purchase and Discount systems helps the customers to view the total price and discounts applied on the products

- Design pattern : Strategy and Decorator Pattern

- Strategy pattern used for choosing the proper discount based on the total number of products

- Decorator pattern used for applying the discount on the the purchase fee during checkout

Implementation



Message system

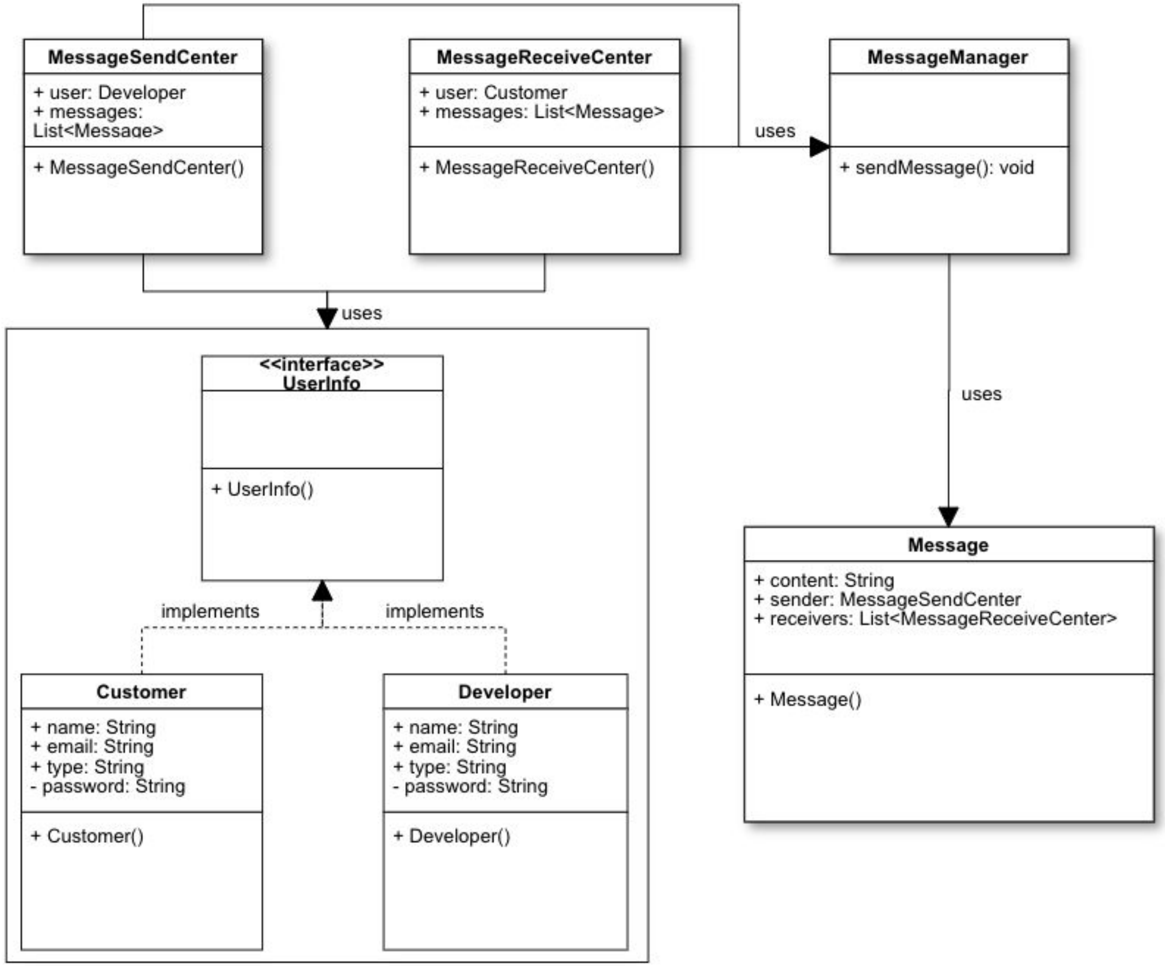
- Developers can send messages to all the customer that purchased their product before

- Customer can view all the messages they receive.

- **Design pattern** : Mediator Pattern

- Each customer has a message receiver center, and developer has a message send center. And all operations about message are done in MessageController.

Implementation



Comment system

- Customers can make comment on a product.

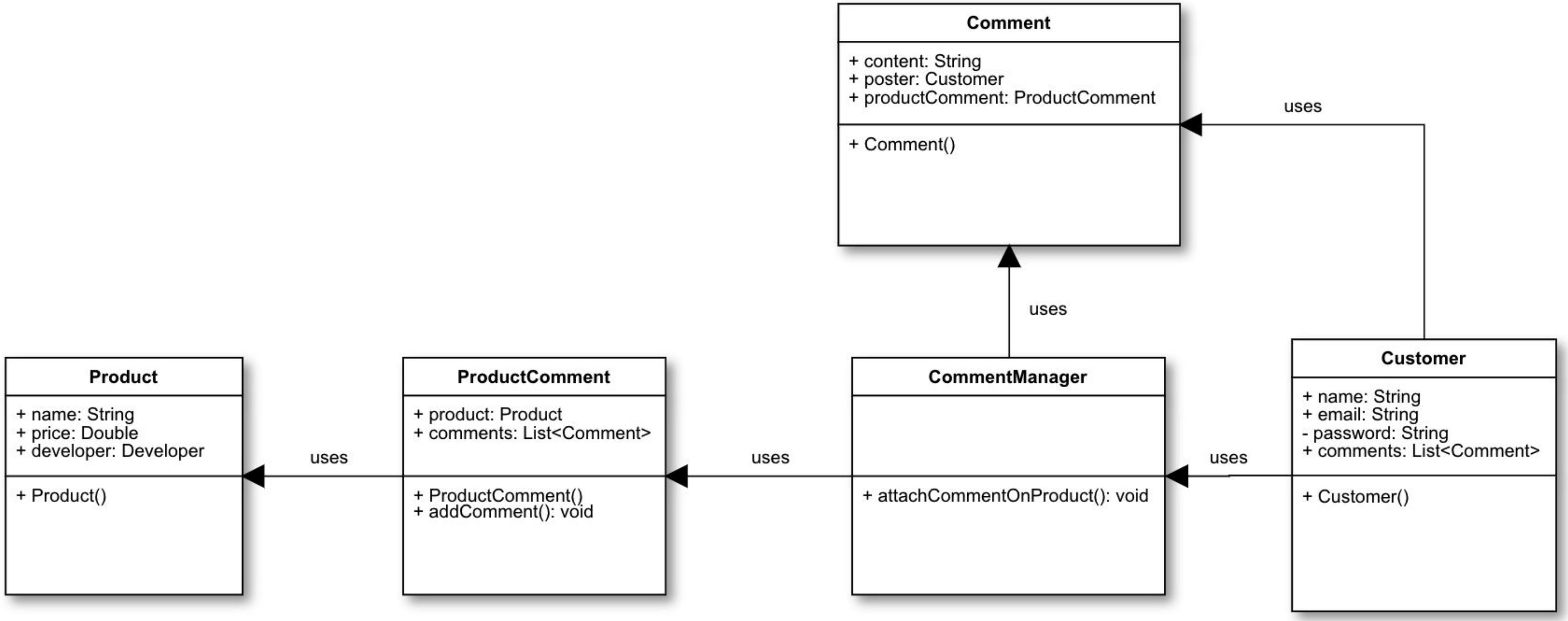
- All other customers can view comments of any product.

- **Design Pattern** : Mediator Pattern

- Each product has a special class to manage all comments of this product.

- Comment Manager will handle all operations about comments

Implementation



Question system

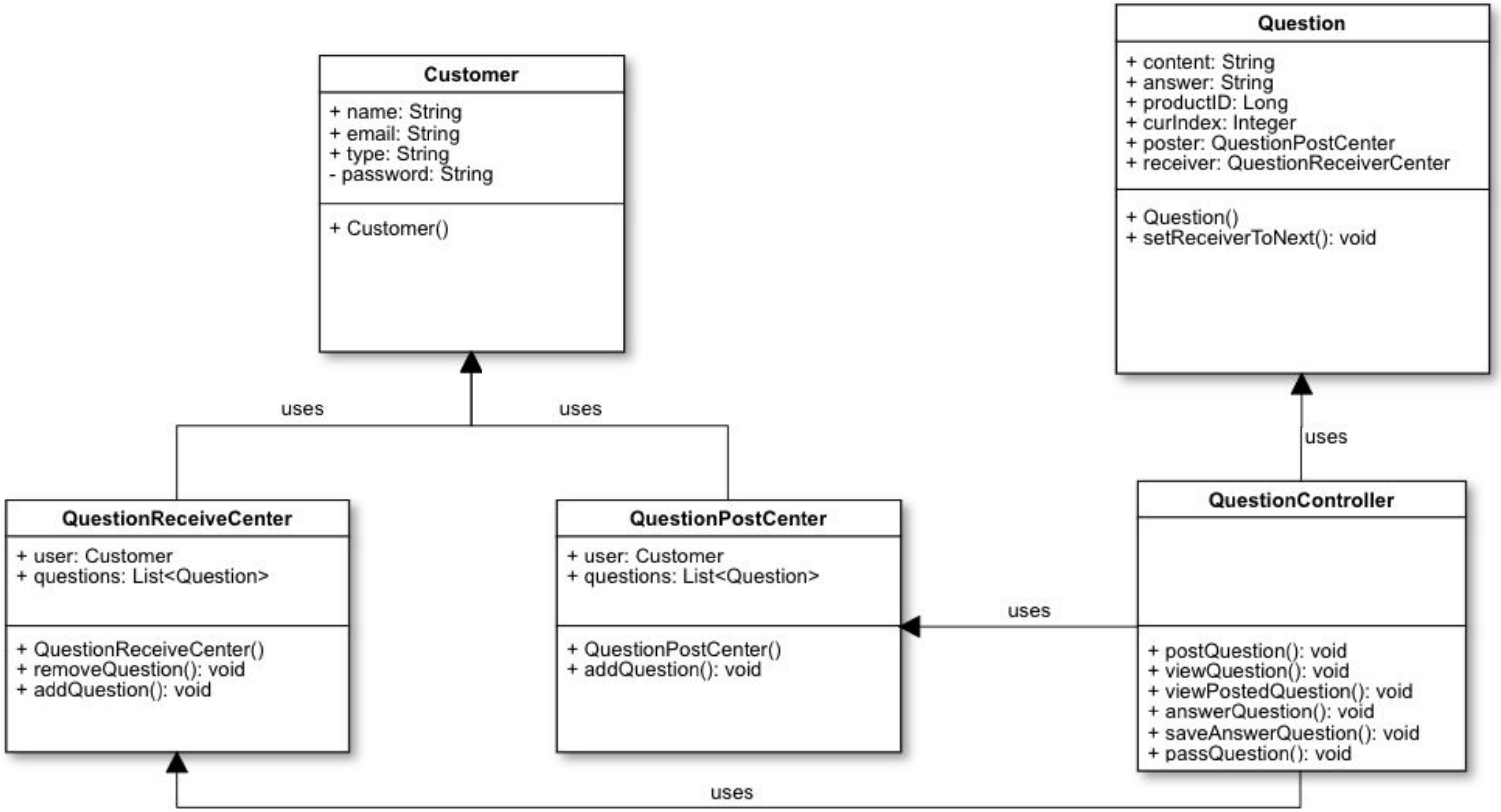
- Customers can post question about a product

- Customers can view all questions from others, they can either answer the question or pass the question to other customer who also purchased the product

- **Design pattern**: Chain of Responsibility Pattern + Mediator Pattern

- Once the question is answered, the question won't be passed to the next customer and question poster can see the answer in his own question pool

Implementation



Inventory system

- Each customer has an inventory to monitor the lifecycle of products they ordered

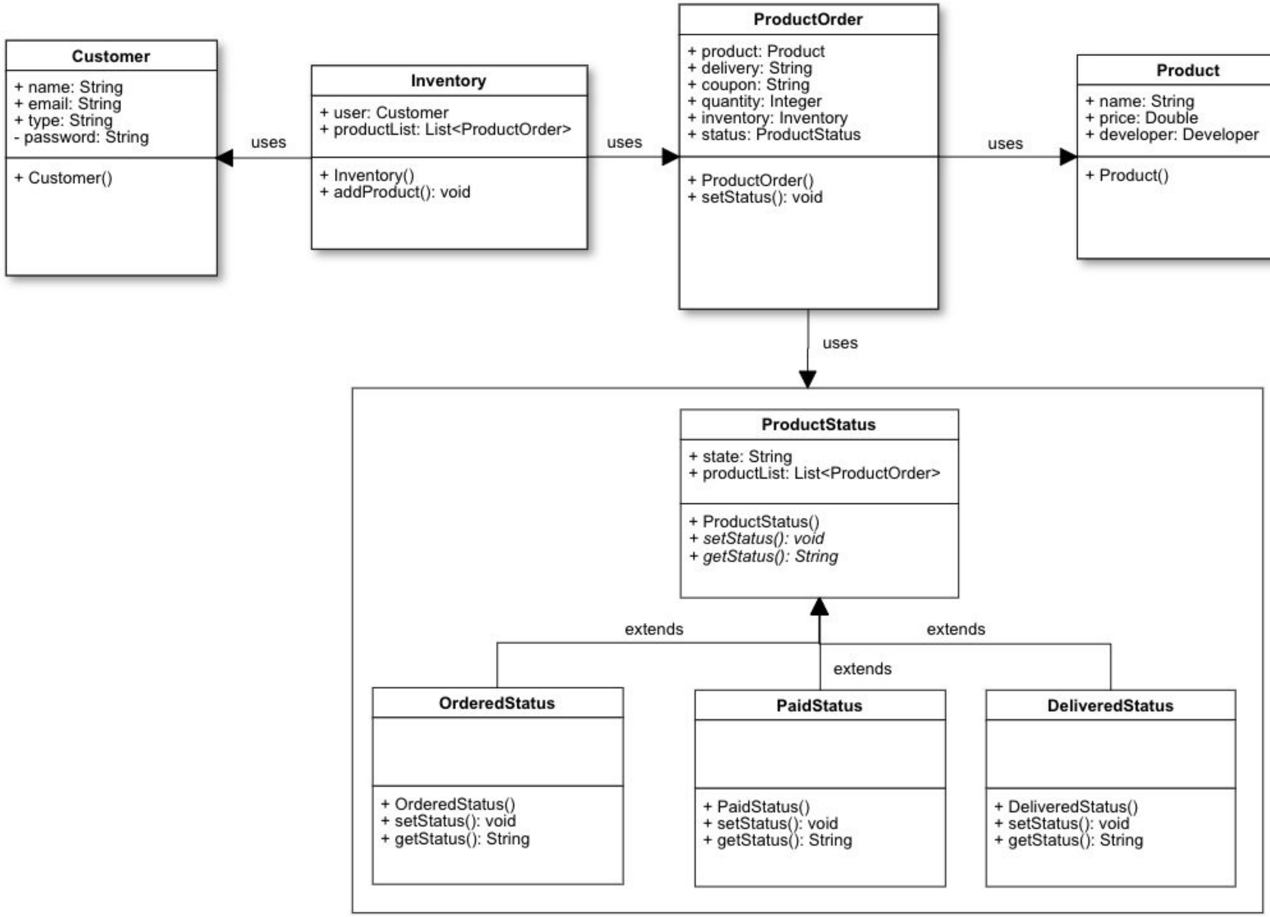
- **Design Pattern** : State Pattern

- Each product order has a status field that can be set by a status object

- Three status: ordered - paid - delivered

- Once the product is added into order from cart, its status becomes “ordered”
- Once customer pay for the order, its status becomes “paid”
- The delivered status need information about logistics, currently not support

Implementation



Coupon system

- ***Developers*** can send coupon to all the customer that purchased their product before; they can choose the coupon discount between 20%/40%/60%

- ***Customers*** only can use the coupon when they buy the same product again (there is no option for coupon if customer don't have the coupon); In addition, they can view all the coupon they have (including discount and product).

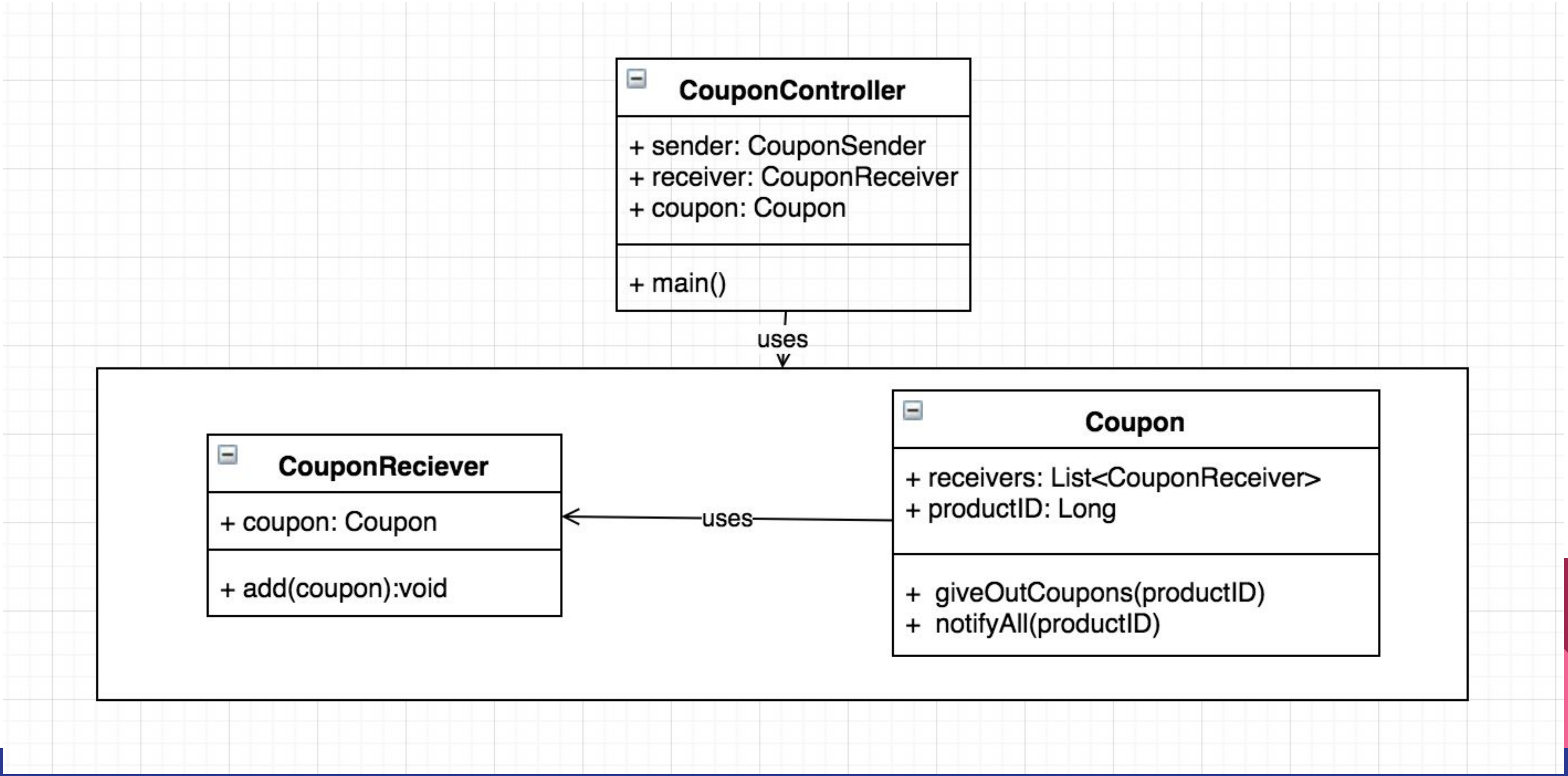
- **Design Pattern** : Observer Pattern + Decorator Pattern

- Use Notifyall() function to send the coupons to the certain customers.

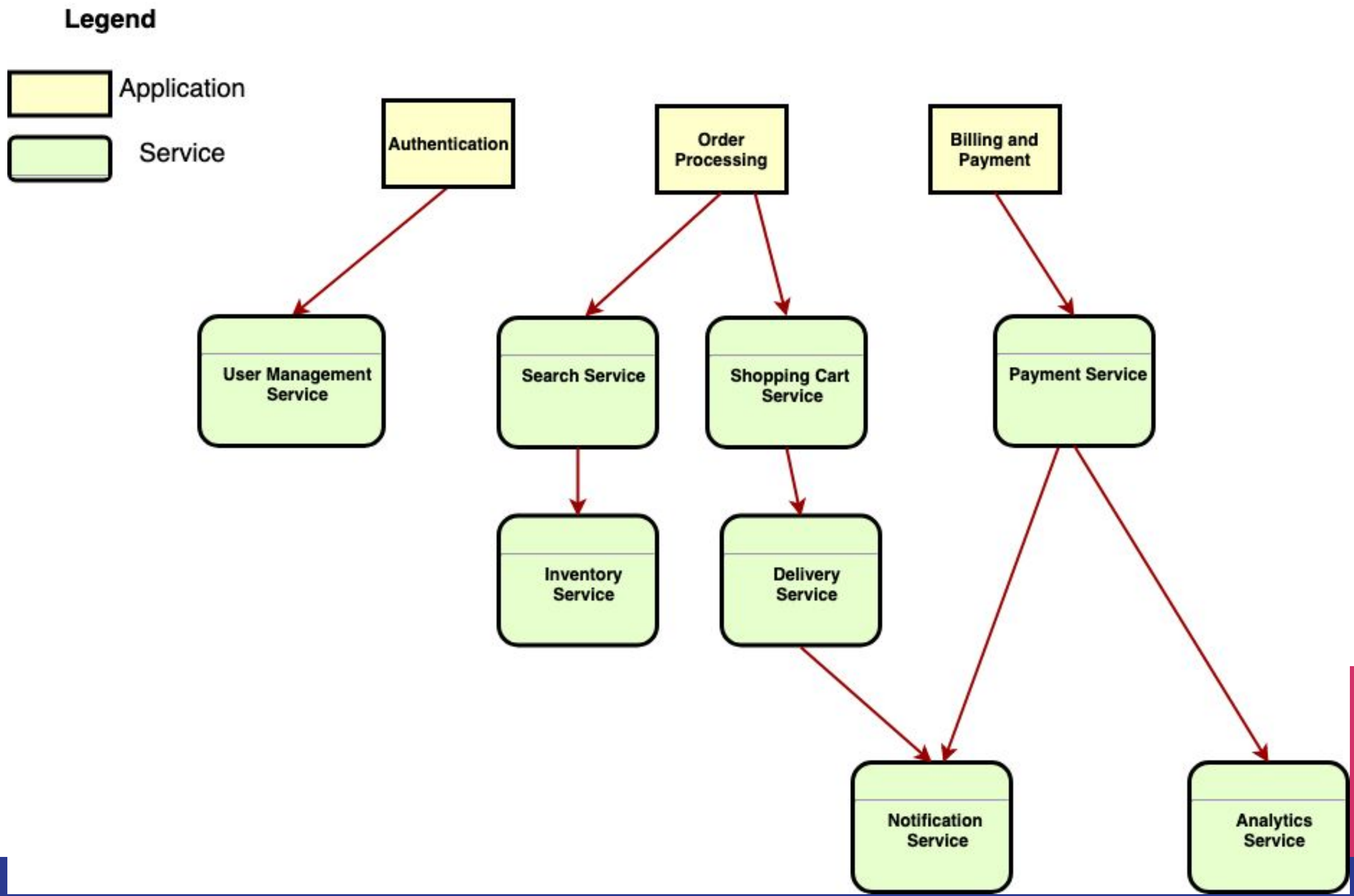
- Use the same discount decorator to calculate price

- Future work: Delete the coupon after it used

Implementation - Observer Pattern



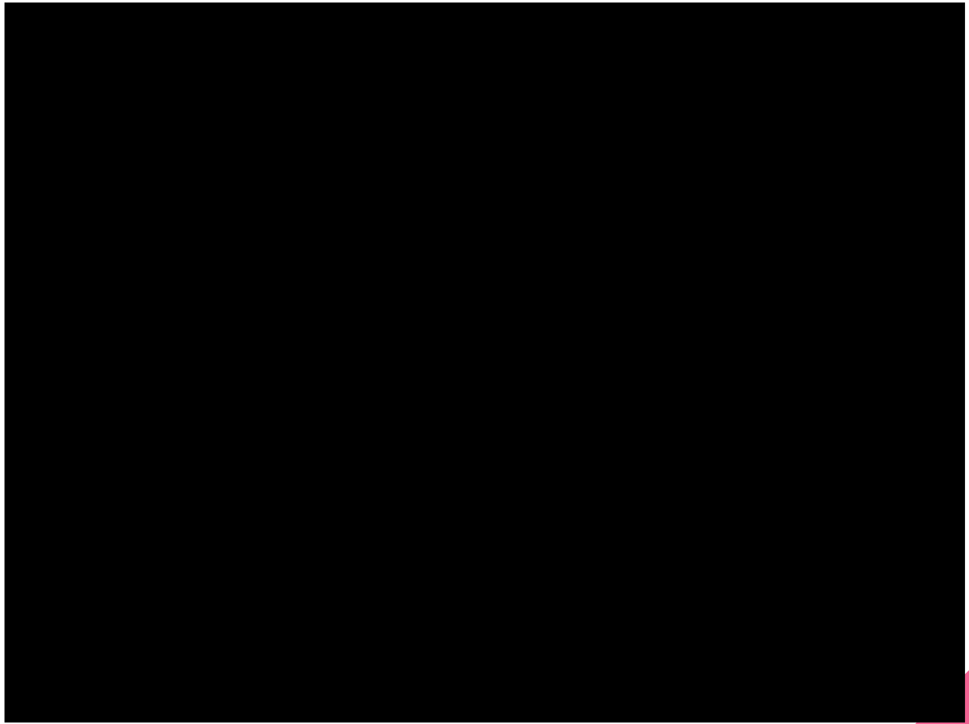
Service Oriented Architecture



Environment

- Play framework (Java)
- IntelliJ IDE for development.
- Source Code: https://github.com/cmusv-sc/18653-Spring2019-Team1
- Library dependencies:
  - Java 1.8
  - SBT (scala)
- How to run:
  - Import project into IntelliJ
  - Run the application using IDE or use “sbt run” from terminal

Demo



Conclusion and future work

- Used design patterns and architecture principles for the application development
- Multiple instances of the web server and database for increased availability of the server

● Improved security for authentication system

● Adopt Devops using Docker containers for a faster test and deploy

Thank you.