Assignment 3
Analysis and Design Document

Student: Roca Eric
Group: 30238

Table of Contents

1. Requirements Analysis

1. Assignment Specification

You are tasked to build a deal search engine for furniture products.

2. Functional Requirements

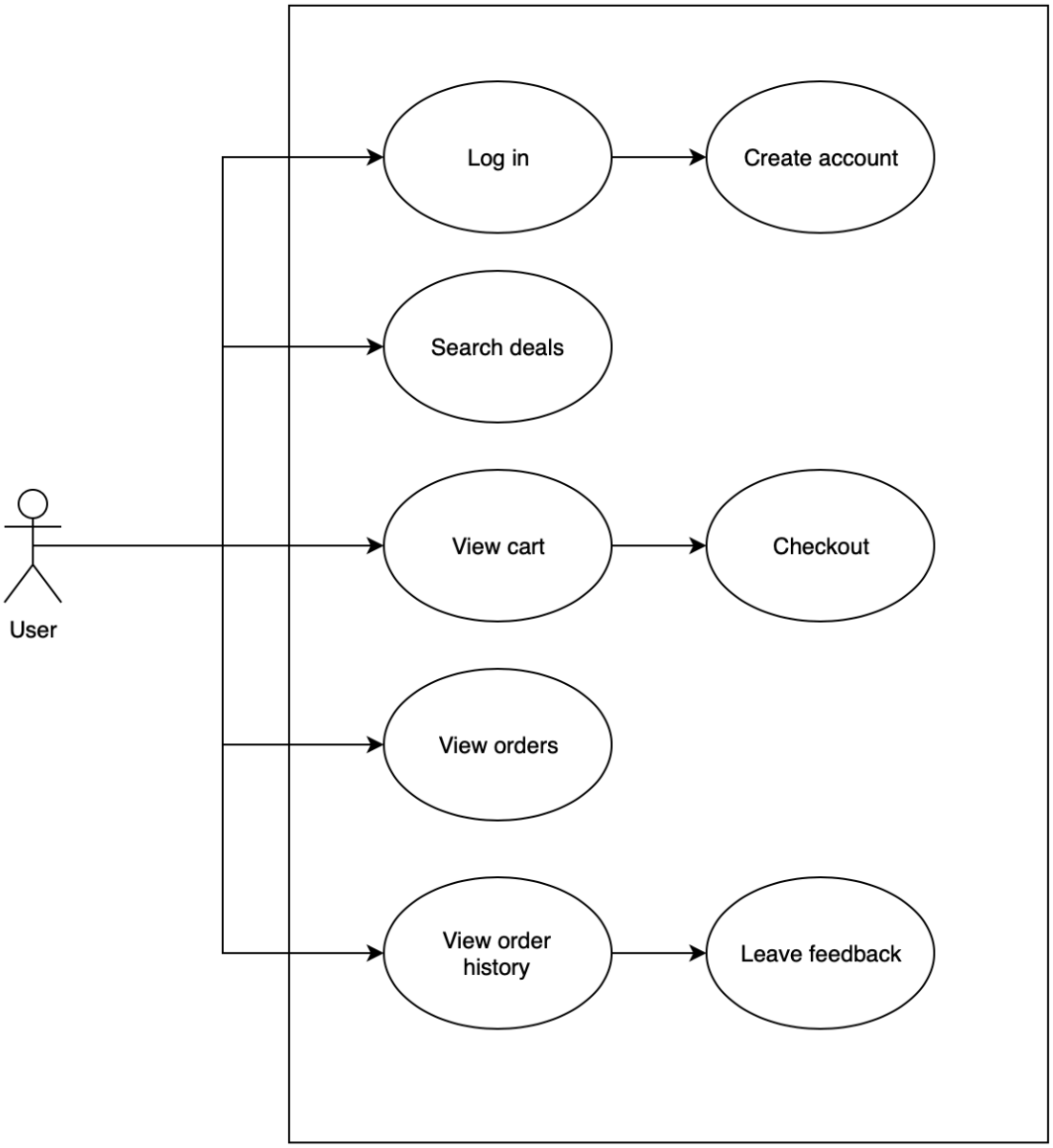
- A user should be able to create an account and login to search for various provided deals.
- Deals are managed by staff and can be filtered by price, name, type.
- If a deal is available users can add the associated product to his cart and proceed to checkout.
- Payments can be done via a cash only policy and need to be validated by staff. This creates an order in the system that can be tracked by the user from the Order History section. The state of an order is updated by staff.
- Once an order is delivered the user can provide feedback in a form on the specific Order History entry details.

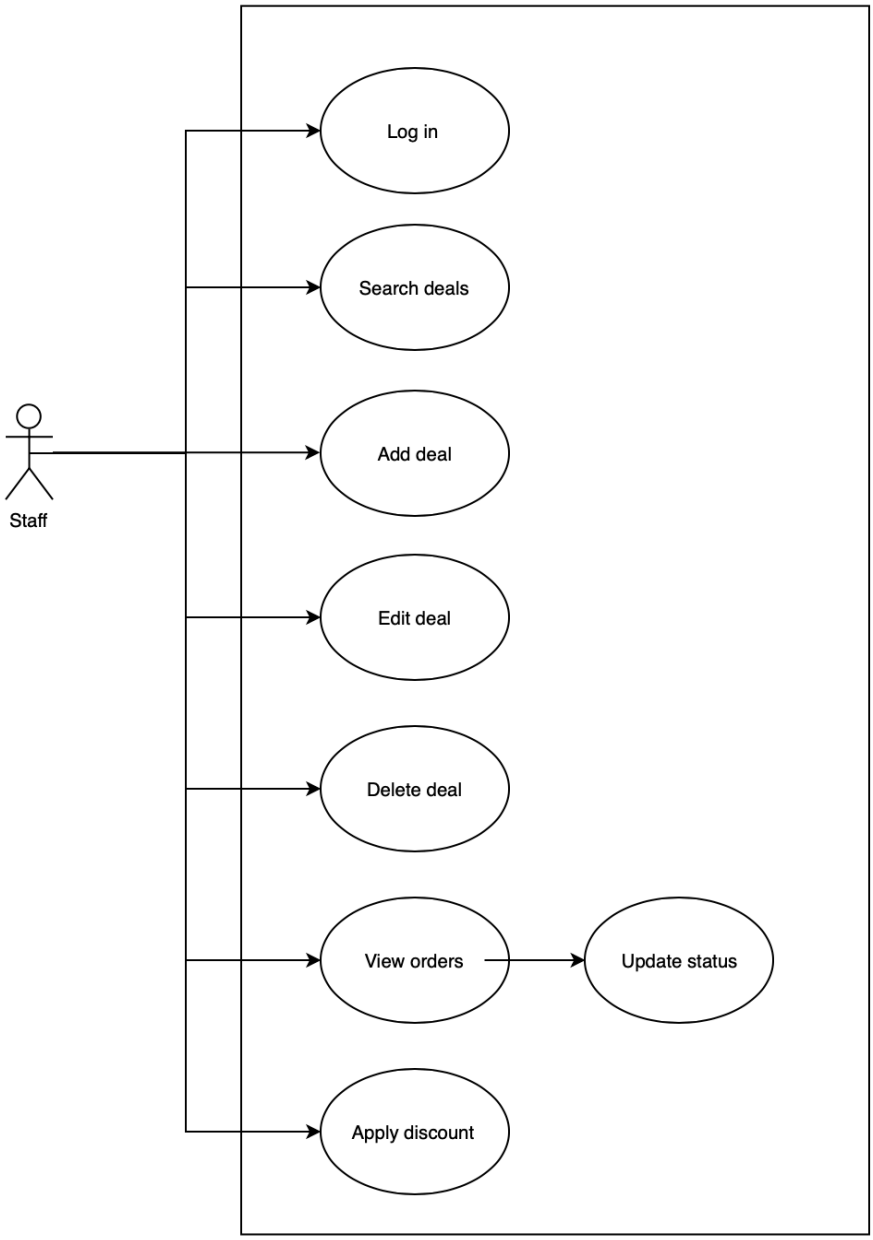
3. Non-functional Requirements

- Implement and test the application
- Commit the work you do on your Git repository. Do it iteratively as you progress, not all at once (this will incur a penalty on your final mark)
- Use any OOP language you like. Non-exhaustive: Python, C#, Java, Ruby, C/C++, JS+Typescript
- Use a layered architecture
- Use a factory method for
    - A1.2 To build and apply discounts on existing products which change the order quantity and overall price accordingly
- Use an observer for
    - A1.2 Getting notifications when the state of an order changes
- Use a CQRS architecture, use a mediator pattern to handle requests
- Use a decorator pattern for
    - A1.2 Applying deals to the shopping cart
- The data will be stored in a database
- All the inputs of the application will be validated against invalid data before submitting the data and saving it in the database.

2. Use-Case Model

Use case: add deal
Level: user-goal level
Primary actor: staff
Main success scenario: login -> enter deal data -> deal is added to database
Extensions: login -> no such user exists
login -> incorrect password
login -> enter deal data -> deal already exists


Log in
Create account
Search deals
View cart
Checkout
User
View orders
View order history
Leave feedback


Log in
Search deals
Staff
Add deal
Edit deal
Delete deal
View orders
Update status
Apply discount

3. System Architectural Design

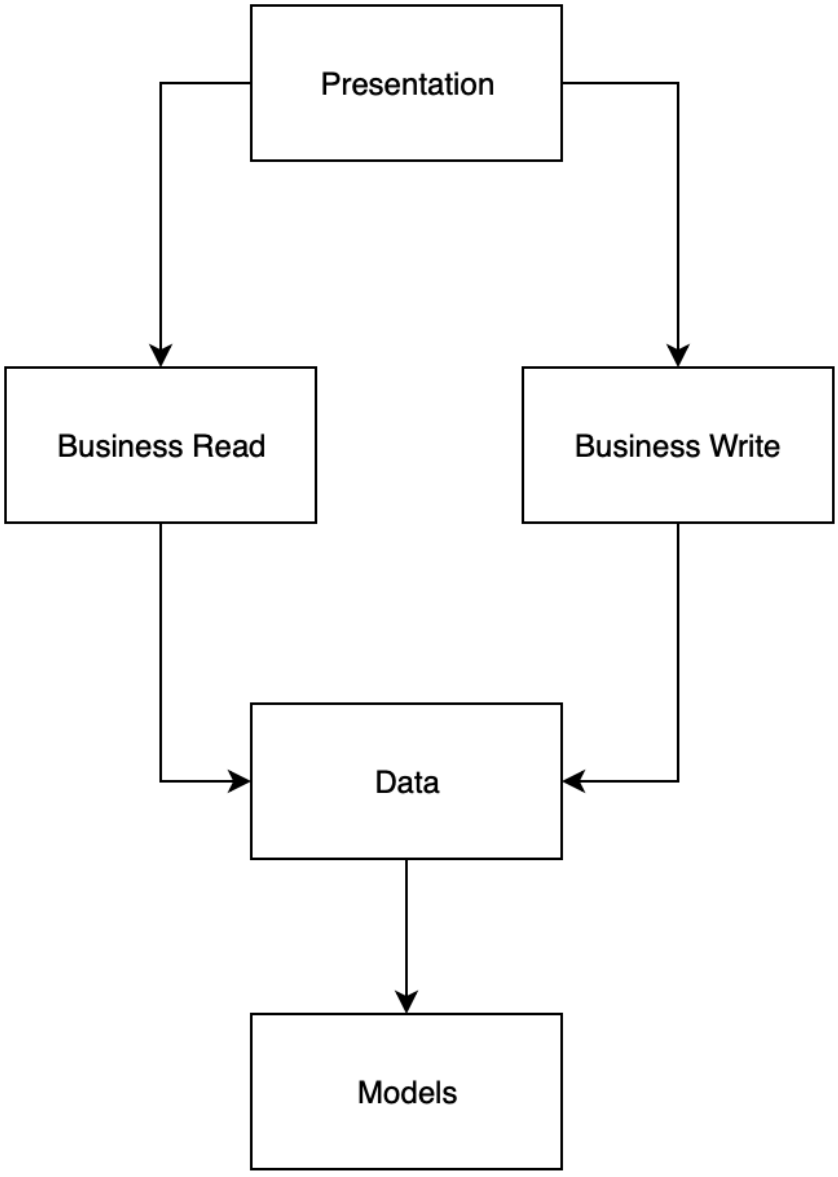
3.1 Architectural Pattern Description

The architectural pattern used in this application is Layers. Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g. service logic or model logic). One of the powerful features of the layered architecture pattern is the separation of concerns among components. Components within a specific layer deal only with logic that pertains to that layer. This type of component classification makes it easy to build effective roles and responsibility models into the architecture, and also makes it easy to develop, test, govern, and maintain applications using this architecture pattern due to well-defined component interfaces and limited component scope.

Representational State Transfer (REST) was used to expose the application's services to the Web. REST, or REpresentational State Transfer, is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other. REST-compliant systems, often called RESTful systems, are characterized by how they are stateless and separate the concerns of client and server. We will go into what these terms mean and why they are beneficial characteristics for services on the Web.

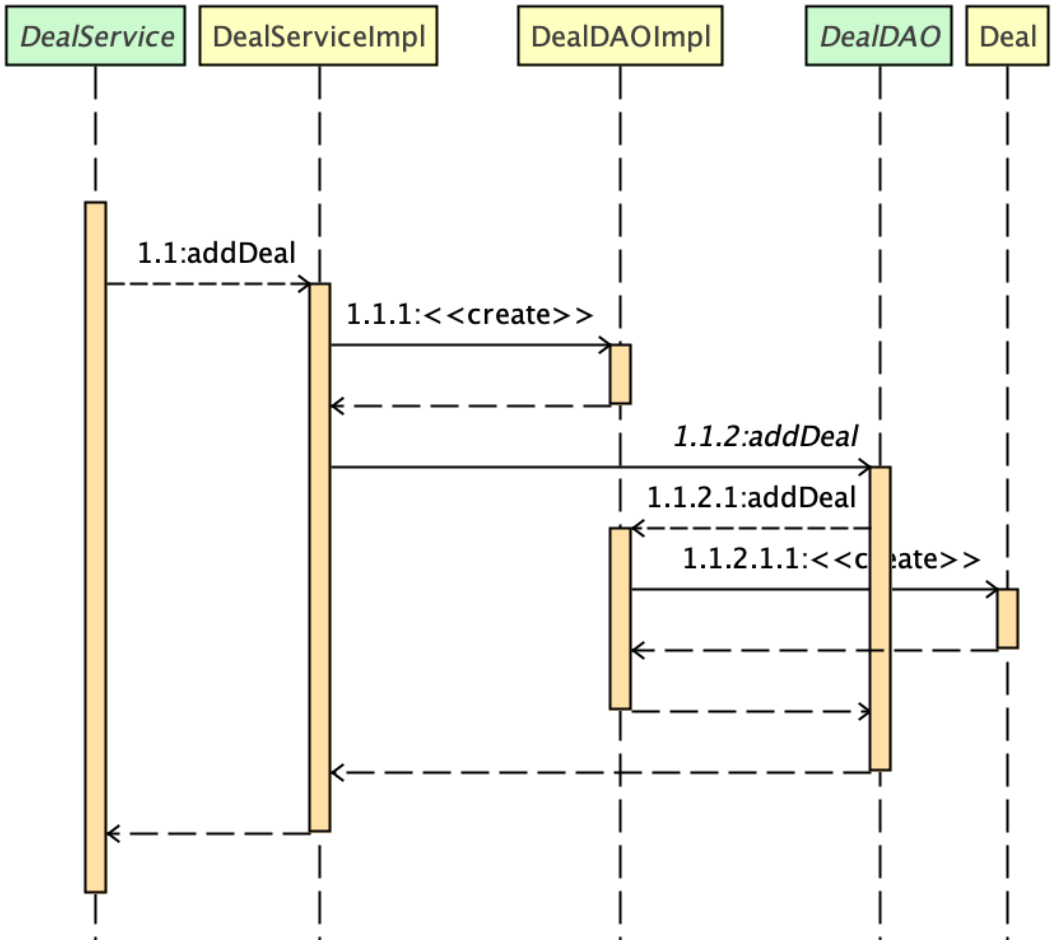
Another pattern used in this application is the Command and Query Responsibility Segregation (CQRS) pattern. Command Query Responsibility Segregation (CQRS) is an architectural pattern that separates reading and writing into two different models. This means that every method should either be a Command that performs an action or a Query that returns data. A Command cannot return data and a Query cannot change the data.

3.2 Diagrams



4. UML Sequence Diagrams

The following is the sequence diagram for the add deal action:



5. Class Design

5.1 Design Patterns Description

The Factory design pattern was used to apply discounts to deals. The Factory pattern is one of the most used design patterns in Java. This type of design pattern comes under the creational category as this pattern provides one of the best ways to create an object. In the Factory pattern, we create objects without exposing the creation logic to the client and refer to newly created objects using a common interface.

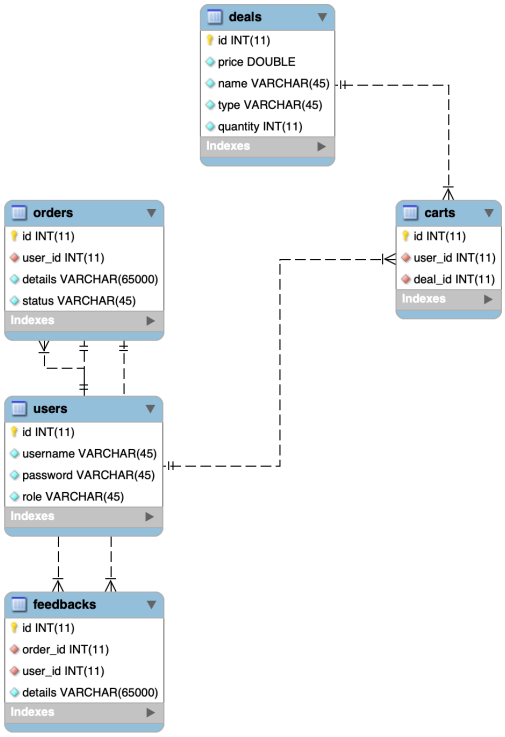
The Observer design pattern was used to notify users when an order changes its status. An observer defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. The Observer pattern falls under the behavioral pattern category.

The Decorator pattern was used to apply deals to a shopping cart. A Decorator pattern can be used to attach additional responsibilities to an object either statically or dynamically. A Decorator provides an enhanced interface to the original object.

5.2 UML Class Diagram

6. Data Model

The five data models used in the system's implementation are Users, Deals, Carts, Orders, and Feedbacks. These five data models are described in the SQL language as follows:



7. System Testing

Manual testing was used to test the application.

8. Bibliography

Google
Stack Overflow
Tutorialspoint