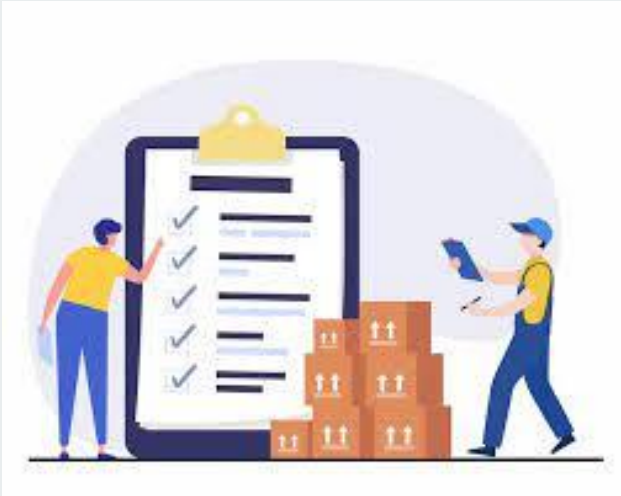
Inventory Management System

By
Chandana Joshi
Pranay Kasavaraju
Pranav Khismatrao
Akanksha Pawar
Yash Sah



Overview

In this project we have implemented the following design patterns :

- Command
- Decorator
- Singleton
- Factory
- Strategy
- Facade
- Observer
- State

Contributions

- **By Chandana :**
  Observer
  Command
- **By Pranay:**
  1. Strategy
  2. State
- **By Pranav:**
  1. Facade
  2. Implemented UI

Contributions

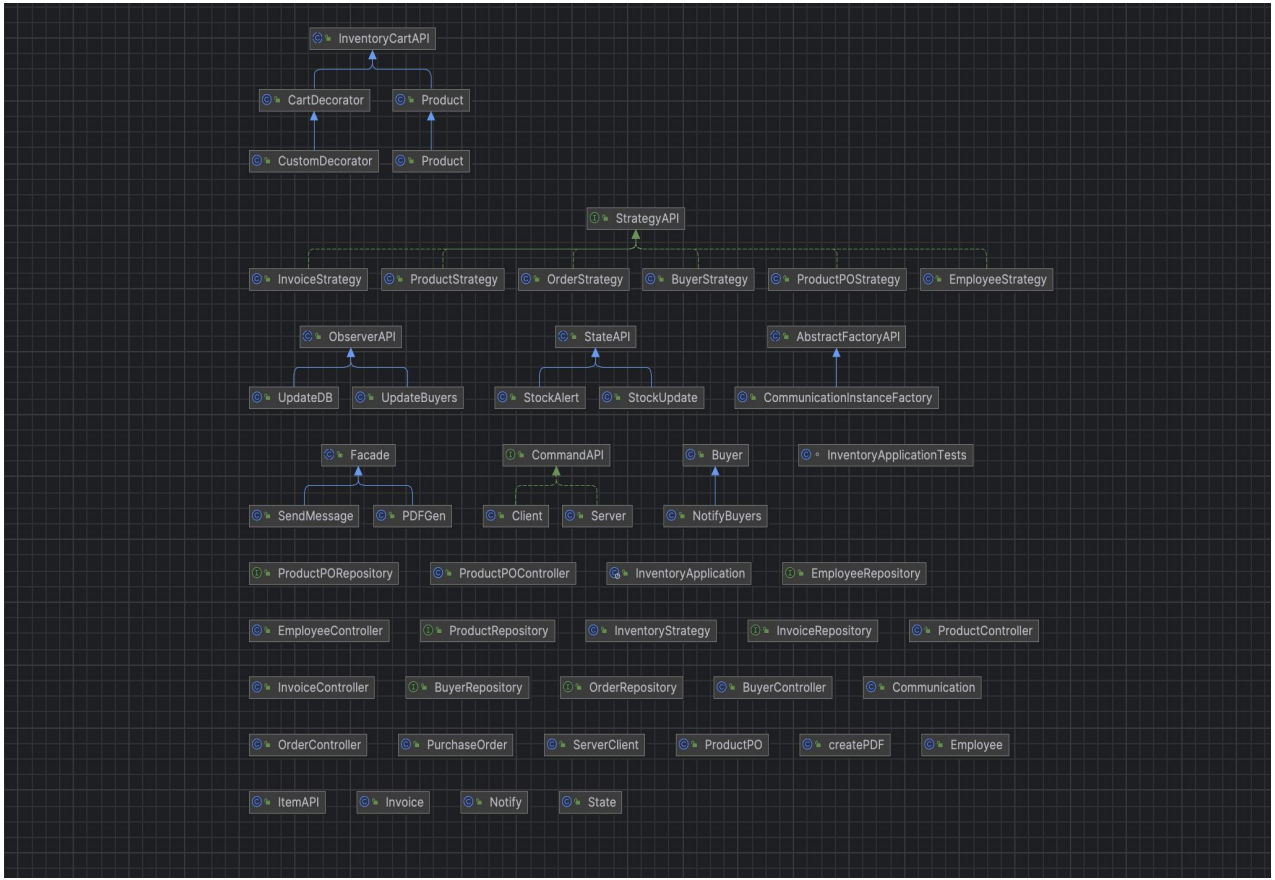
By Yash:

Decorator Pattern
Implemented the UI features
Creation of all the classes and Objects
Debugged and Resolved errors in multiple design patterns

By Akanksha:

State Pattern
Worked on the manageProducts and manageEmployees page in the UI

Class Diagram



Command Design Pattern

- Designed a CommandAPI which includes an execute() method.
- We have two implementing classes that when invoked would be responsible for :
  a. Server
  b. Client

Decorator

- Designed abstract class InventoryCartAPI implements
- Main purpose of this Design Pattern is to wrap up the existing object
- Created CartDecorator which is inherited by CustomDecorator
- Created Product Class to get the total Price of the products added

Facade

- Facade is a structural design pattern that provides a simplified interface to a complex system of classes, objects, and subsystems.
- Created a Abstract Class Facade which is inherited by CreatePDF and PDFGen to create and generate invoice as pdf
- **Facade** hides the complexity of the pdf generating system and provides a simple interface to the clients to perform the required operations.

Observer

- The Observer Design Pattern is used to define a one-to-many dependency between objects so that one object changes its state, all its dependents are notified and updated automatically
- Created ObserverAPI abstract class which is inherited by UpdateBuyers(Observer) and NotifyBuyers(Inform Buyers)

Strategy

- Strategy Design Pattern is used to enhance code flexibility, maintainability, and readability by allowing dynamic selection of algorithms, separating concerns and promoting reusable and testable code
- Implemented add(), update(), delete() methods in StrategyAPI Interface
- StrategyAPI is used by Buyer, Employee, Inventory, Invoice, Order, ProductPO, Product

State

- State Design Pattern is used to allow an object to alter it’s behaviour when it’s internal state changes which improves modularity, scalability and readability
- Implemented action() method in StockAPI interface and used in StockUpdate and StockAlert to alert when the stock is less than half and update to buy more stock

Factory

- It is a pattern that promotes loose coupling between classes and promotes code reuse.
- Created CommunicationInstance Factory which is responsible for creating communication objects which is used by udpTrigger method
- Implemented Factory using Lazy Singleton.

Thank You