**MPAI**

**Sistem de comenzi online**

Marin Andrei

Moldoveanu Robert

Nadeem Kashan

Udrescu Alexandru

1. Descrierea problemei
2. Descrierea solutiei
3. Modelele de proiectare utilizate
   1. Abstract Factory
   2. Builder
   3. MVP
   4. Strategy
4. **Descrierea problemei**

Mecanismul de preluare a comenzilor, in contextul economiei contemporane, este unul complex care in practica necesita sustinerea unei platforme informatice prin care acest proces sa se realizeze facil.

Pentru realizarea acestui deziderat este necesara construirea unei interfete eficiente pentru utilizator care sa-i permita alegerea unor produse, a cantitatilor acestora si a unei modalitati de plata, date care sa ajute furnizorul in orchestrarea comenzilor. Mai mult, solutia ar trebui sa ofere si o modalitate prin care utilizatorul sa-si identifice usor comanda, adica sa existe un identificator unic al comenzii generat la finalizarea acesteia, cat si alte informatii aditionale .

1. **Descrierea solutiei**

Solutia implementata vine in intampinarea dificultatilor care pot aparea intr-un flux de comenzi de produse prin utilizarea unor sabloane de proiectare care ofera atat flexibilitate codului sursa scris cat si performanta ridicata a aplicatiei. Modelele de proiectare utilizate sunt ,,Builder”, ,,Strategy”, ,,Abstract Factory” si ,,Model-View-Presenter”.

Rezultatul final al aplicatiei, din punct de vedere al utilizatorului, este reprezentat de interfata grafica prin care devine realizabil procesul de plasare de comenzi online.

Utilizatorul isi poate alege Brand-ul produselor pe care doreste sa si le comande, tipul acestor produse, cat si cantitatea . Modalitatea de plata se specifica printr-un control de tip combo, asemenea locatiei de livrare. Butonul cu label-ul ,,Add product” ajuta la adaugarea unui set de comenzi intr-o lista, iar butonul cu label-ul ,,Send order” ajuta la trimiterea comenzii.

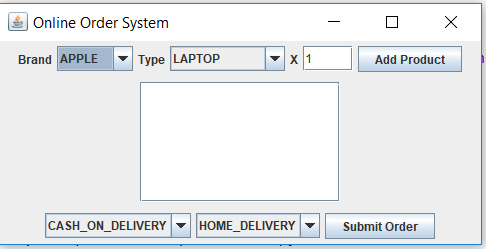


Figura 1.

1. **Modele de proiectare utilizate**

**Modelul Abstract Factory**

Abstract Factory este un design pattern creational, foarte similar cu Factory pattern. Este considerat a fi inca un layer de abstractizare peste cel de factory pattern. Abstract factory se comporta ca un super-factory care creaza alte factories.

Are scopul de a crea familli de obiecte fara a depinde de implementarea lor concreta.

Decupleaza generatorul de obiecte concrete de clientul care care le utiliezaza.

Asigura un control strict al instantierii obiectelor, deoarece acestea nu sunt create direct ci prin metodele de tip factory. Obiectele de acelasi tip sunt gestionate unitar prin interfata comuna si noi tipuri pot fi adaugate fara modificari.

Folosind Abstract Factory pattern am implementat urmatoarele clase si interfete:

• Factory

- interfață ce definește metodele abstracte pentru crearea de instanțe

- createLaptop() , createDesktop(), createAllInOne()

• Desktop, AllInOne, Laptop

- interfețe ce definesc tipurile abstracte de obiecte ce pot fi create

• AppleFactory, AsusFactory, DellFactory

- clase concrete ce implementează interfața Factory si metodele prin care sunt create

obiecte concrete

• AppleLaptop, AsusLaptop, AppleDesktop ..

- clase concrete ce definesc diferitele tipuri de obiecte ce pot fi create

**Modelul Builder**

Atributele acestei clase sunt urmatoarele:

private int id;

private User client;

private ArrayList<OrderItem> orderList;

private PaymentStrategy paymentStrategy;

Modelul implementat permite construirea dinamica a unei liste de ordine pentru diverse produse. Se va adauga cu ajutorul metodei addProduct cate un produs si se vor seta cantitatea, cat si alte detalii specifice, sau cu ajutorul metodei setOrderList intreaga comanda.

**Modelul Model-View-Presenter**

Modelul MVP este o derivare a modelului arhitectural Model-View-Controller si este folosit in mare parte pentru a crea UI-uri. Presenterul isi asuma functionalitatea de middle-man, preluand toata partea de logica. Modelul este o interfata care defineste datele ce urmeaza a fi afisate in UI. View-ul este o interfata pasiva care afiseaza datele (modelul) si ruteaza comenzile Prezenterului pentru a prelucra acele date si/sau eventuri.

**Modelul Strategy**

Modelul Strategy a fost utilizat pentru implementarea flexibila a sistemului de plati. Aceasta presupune crearea unei interfete prin care se realizeaza un contract de implementare a trei metode:

Boolean pay(Double payAmmount);

void getPaymentData(User user);

boolean verify();

Prima metoda ajuta la realizarea procesului efectiv de plata, prin ce-a de-a doua se colecteaza datele celui care face plata, iar prin ultima metoda se verifica daca datele introduse de utilizator au consistenta.

Interfata este implementata de trei clase, CashOnDeliveryPayment, CreditCardPayment si BankTransferPayment prin care se suprascrie comportamentul procesatorului de plati pentru fiecare situatie posibila.

Multimea de situatii posibile considerata in aplicatie este plata ramburs, plata cu ajutorul cardului de credit si plata prin transfer bancar (OP).

Aceste clase iau ca atribut o instanta de User pentru a putea accesa toate datele utilizatorului, incluzand informatiile cardului sau de credit. In functie de metoda de plata aleasa, se face verificarea si procesarea platii.

Un exemplu de reprezentare in interfata grafica este prezentat in figura 2, in care i sa da utilizatorului posibilitatea de a alege o metoda de plata si de livrare:



Figura 2.