UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Magazin de suplimente alimentare

Name: ȘchiopAdrian Marian
Group: 30234

Table of Contents

Part 1

Specificațiile proiectului

Scopul acestui proiect este de a dezvolta o aplicație web ce poate fi utilizată într-un lanț de magazine de suplimente alimentare. Aplicația are două tipuri de utilizatori: customer – ce poate sa vizualizeze toate produsele disponibile pe site, să caute un produs și să îl cumpere. De asemenea acesta poate lăsa sau șterge sau actualiza comentarii. Adminii pot să vadă toți utilizatorii, să îi modifice, sau să îi șteargă, poate să adauge, elimine sau modifice produsele din magazin. De asemenea adminii pot adauga un produs intr-o anumita categorie și să îi atribuie o eticheta specifică a acestuia.

Cerințe funcționale

Ca și cerințe funcționale avem:

- autentificarea și înregistrarea utilizatorilor prin mail și parolă;
- gestionarea produselor;
- căutare și filtrarea produselor;
- adăugarea cât și scoaterea produselor din coșul de cumpărături
- modelarea utilizatorilor de către admin

Use Case Model 1

Identificarea cazurilor de utilizare

Use-Case: Cumpărarea unui produs

Level: Customer-User

Primary Actor: Customer

Main success scenario: *-Clientul se autentifica pe site*

-Clientul navighează în magazin

-Clientul selectează produsul pe care dorește să îl cumpere

-Clientul adaugă produsul în coșul de cumpărături

-Clientul cumpără produsul

Extensions: - Dacă produsul nu există în stoc utilizatorul este informat de acest lucru.

Use-Case: Modificarea unui produs

Level: Admin-User

Primary Actor: Admin

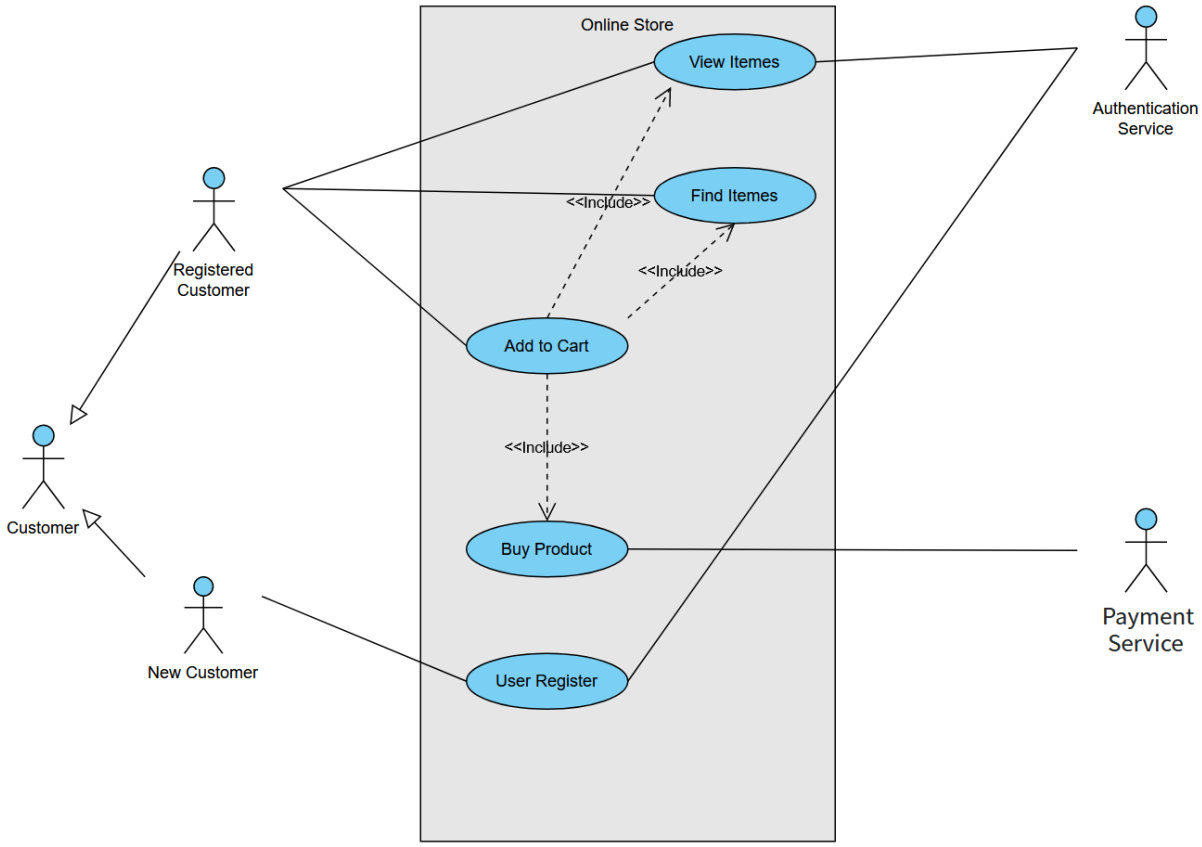
Main success scenario: *-Admin-ul se autentifică pe site*

-Admin-ul caută produsul pe care îl dorește să îl modifice

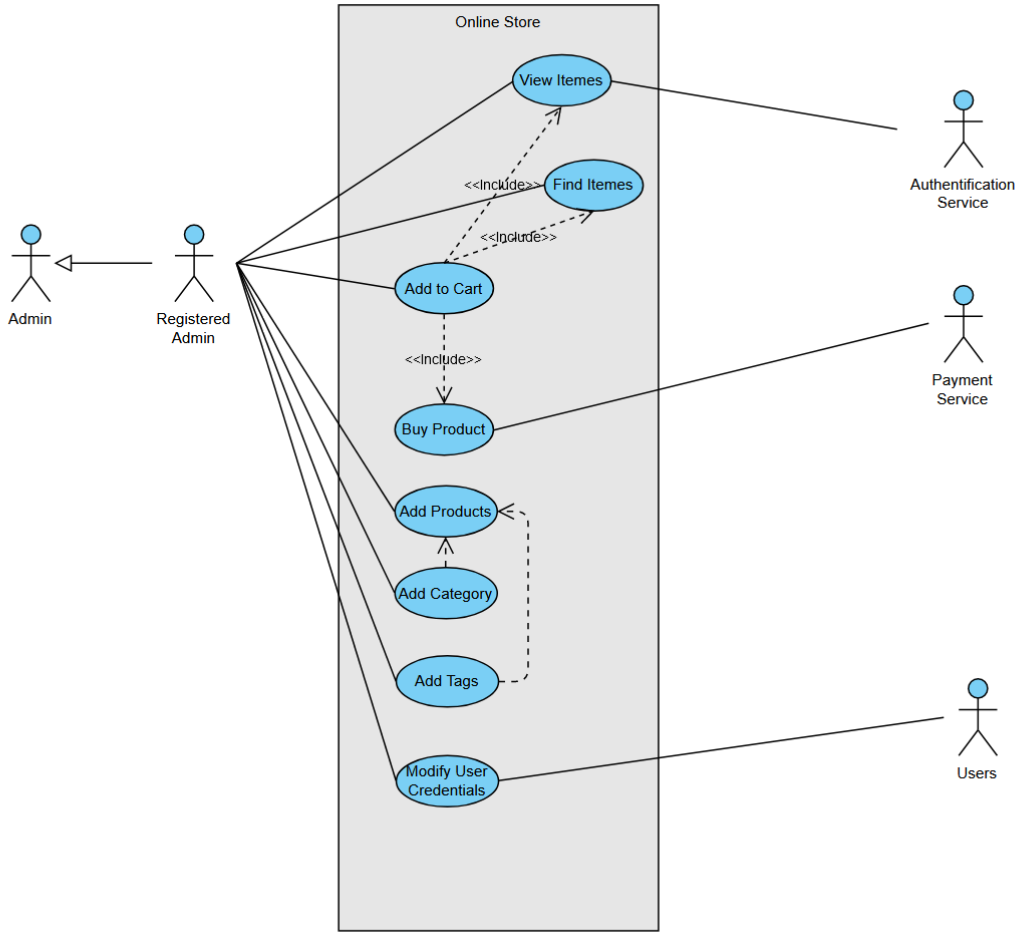
sau îl creează;

-Admin-ul completează noile date ale produsului(cantitate, denumire, descriere, categorie, etichete).

Diagrame Use Case



Use case diagram for user



Use case diagram for admin

Specificații suplimentare

Cerințe non-funcționale

Ca și cerinte non-funcționale avem:

- Stocarea parolelor în baza de date după ce au fost trecute printr-un algoritm de hashing precum bcrypt sau argon2. Acest lucru sporește securitatea bazei de date deoarece dacă un răufăcător pătrunde în baza de date acesta nu poate să acceseze conturile utilizatorilor defiindu-i accesibilă parola.
- Aplicarea unui **SALT.** Acestă implementare sporește de asemenea securitatea bazei de date deoarece ne ajută să combatem atacurile de tipul: dictionary attacks. Înainte ca parola să fie trecută prin algoritmul de hasing la aceasta se adaugă câteva caractere random.
- Scalabilitatea aplicației: respectarea principiilor SOLID, utilizarea de design patterns cât și implementarea corectă a claselor în aplicație pot duce la o scalabilitate mult mai ușoară a aplicației, reducând tight coupling-ul intre clase.
- Disponibilitatea: aplicația este disponibilă non-stop, fiind hostată pe un server, nedepinzând doar de funcționalitatea unui singur calculator.

- Performanța aplicației de a răspunde rapid la cerințele utilizatorului. Un exemplu elocvent îl consideră timpul de încărcare al paginii web. Paginile web trebuie să se încarce rapid pentru a oferi o experiență bună de utilizare.
- Portabilitatea ce este asigurată de faptul că aplicația este de tip web și nu este influențată de sistemul pe care aceasta rulează.

Constrângeri de implementare

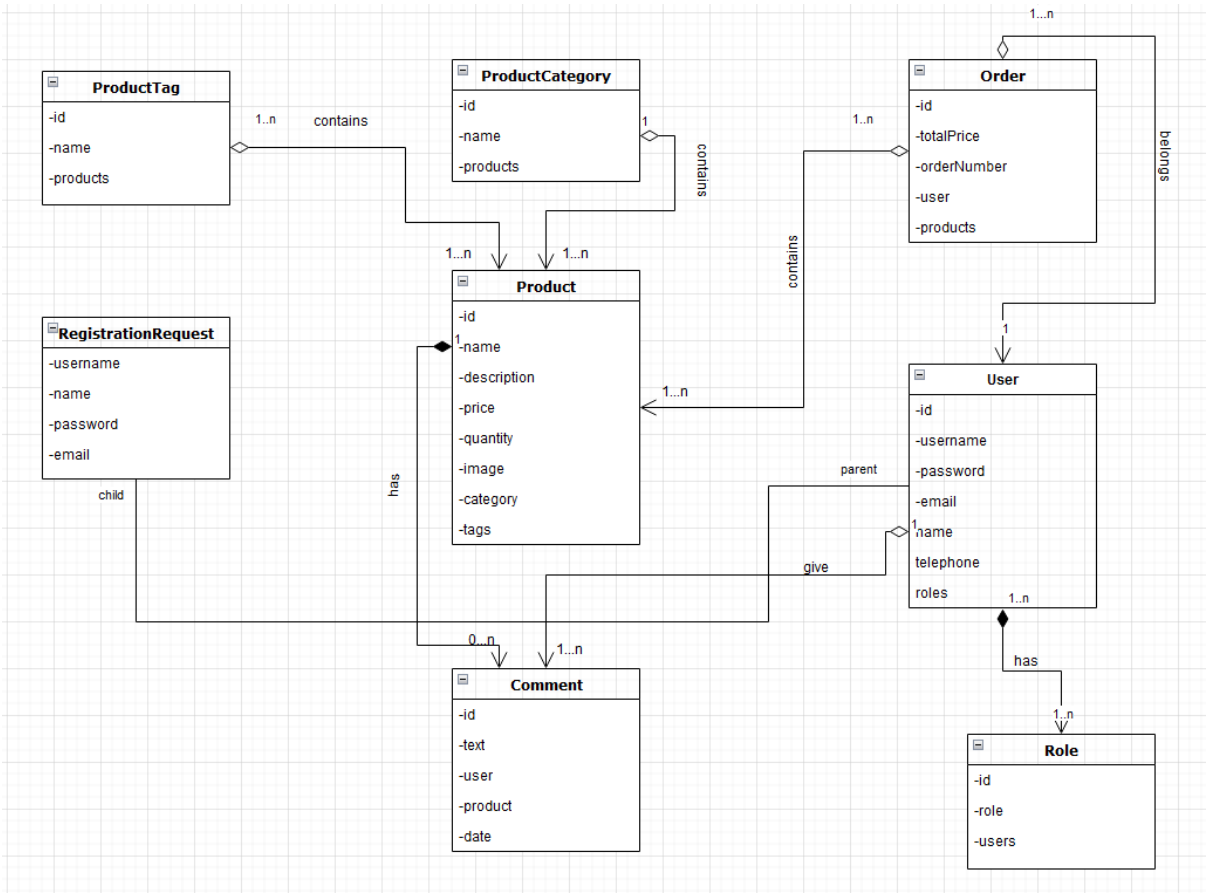
Partea de back-end a aplicației a fost scrisă în Java folosind framework-ul Spring Boot, iar partea de front-end a fost realizată cu ajutorul framework-ului ReactJS. Baza de date a aplicației a fost realizată în MySQL, o baza de date relațională. Arhitectura aplicației este MVC. Pe partea de biblioteci am folosit: Spring Security, Spring Web, Spring Boot DevTools, Lombok. Am folosit următoarele instrumente software IntelliJ IDEA, Visual Studio Code, MySQL Workbench și Git pentru controlul versiunilor.

Glosar

- Autentificare și autorizare – ambele procese au rolul de a securiza o aplicație web. Procesul de autentificare se realizează înainte de procesul de autorizare. Autentificarea reprezintă conectarea la un site web prin credențiale de tip mail și parola. Autorizarea reprezintă procesul prin care se stabilește ce atribuții și permisiuni îi revin unui anumit tip de utilizator.
- MVC - model view controller reprezintă un tip de arhitectura ce separă business logic-ul față de interfața de utilizator și datele de input.
- Validarea adreselor de mail. Acest lucru este posibil cu ajutorul adnotării din Spring *@Email* ce verifică daca utilizatorul a introdus un model de mail obișnuit.
- Algoritmii de hashing și saltingul reprezintă unele dintre metodele cele mai eficiente pentru securizarea bazei de date. Un “salt” este o valoare aleatorie unică adăugată la parola unui utilizator înainte de a fi trecută printr-un algoritm de hashing. Acest proces îmbunătățește securitatea parolelor stocate prin prevenirea atacurilor de tip “rainbow table”, în care un atacator încearcă să ghicească o parolă prin compararea hash-ului acesteia cu hash-urile precalculate din tabel. Uniformitate: Hash-urile generate de un algoritm bun sunt distribuite uniform, ceea ce înseamnă că fiecare valoare posibilă a hash-ului este la fel de probabilă. Determinism: Pentru aceleași date de intrare, algoritmul de hashing va produce întotdeauna același hash. Eficiență: Un algoritm de hashing bun este capabil să calculeze hash-ul rapid. **Rezistență la coliziuni**: Este foarte dificil (ideal, imposibil) să găsești două intrări diferite care produc același hash.

Part 2.

Domain Model



Domain Model Diagram

Architectural Design

Conceptual Architecture

Am ales sa folosesc arhitectura layers:

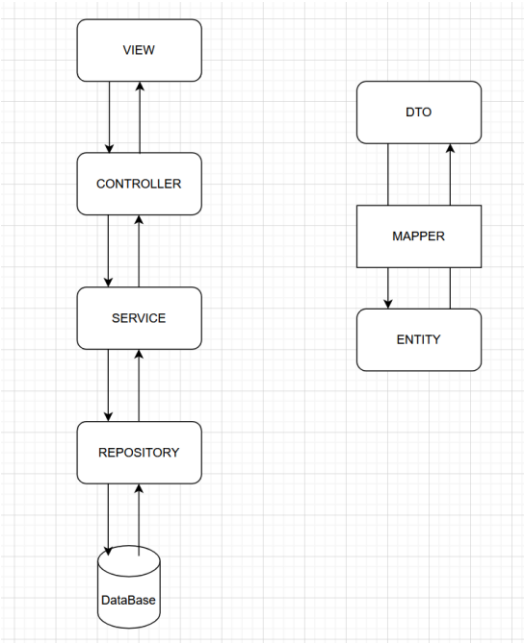


Diagrama Conceptuala

Package Design

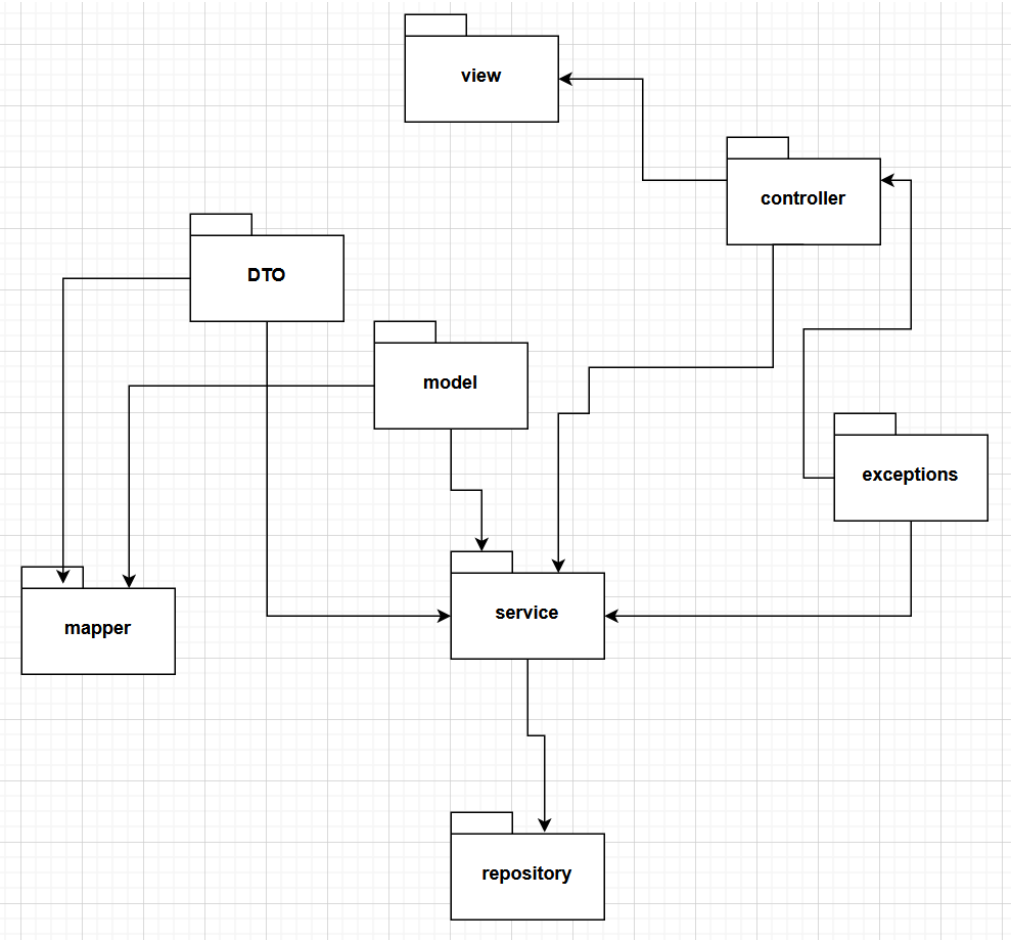


Diagrama de pachete

Component and Deployment Diagram

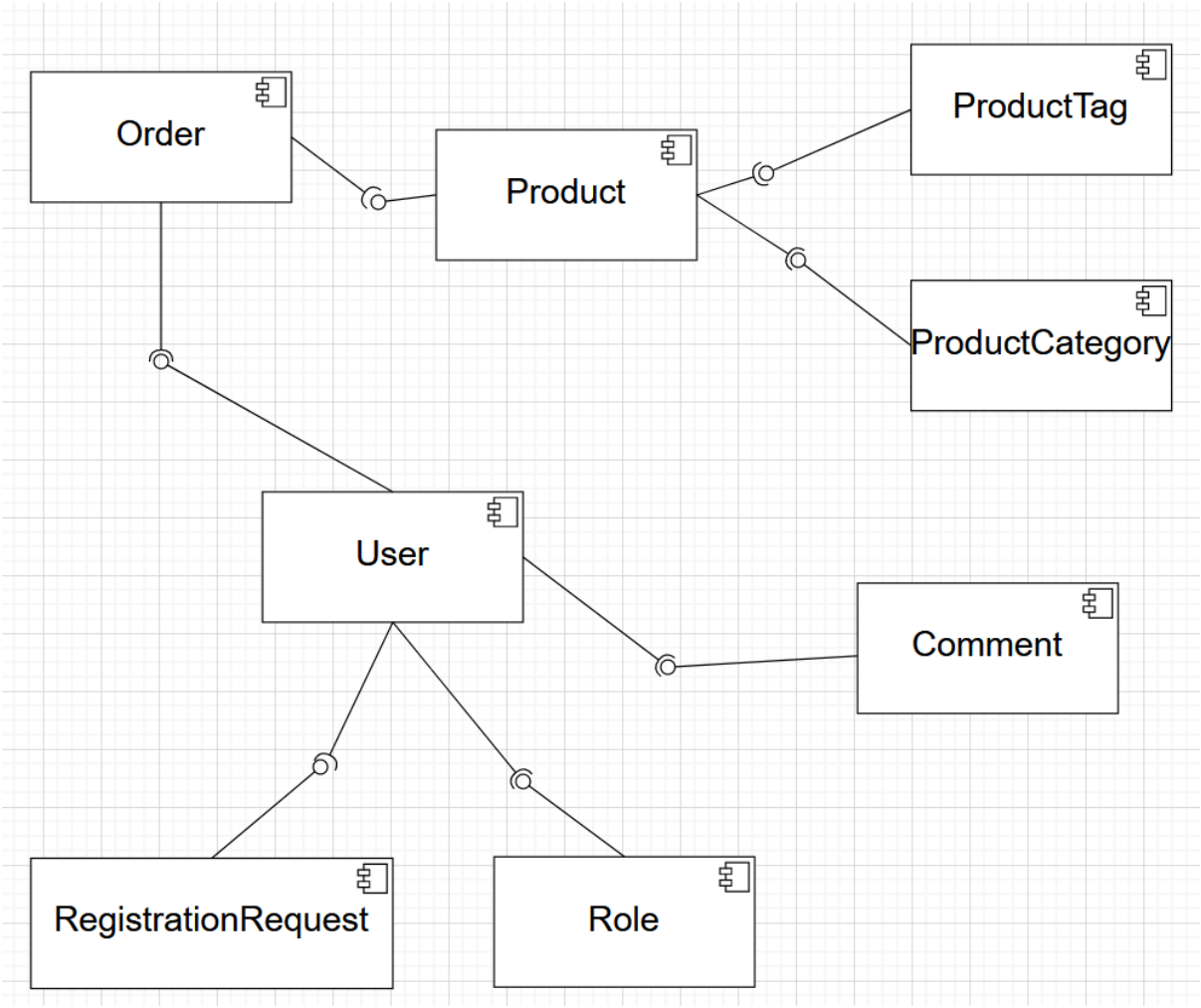
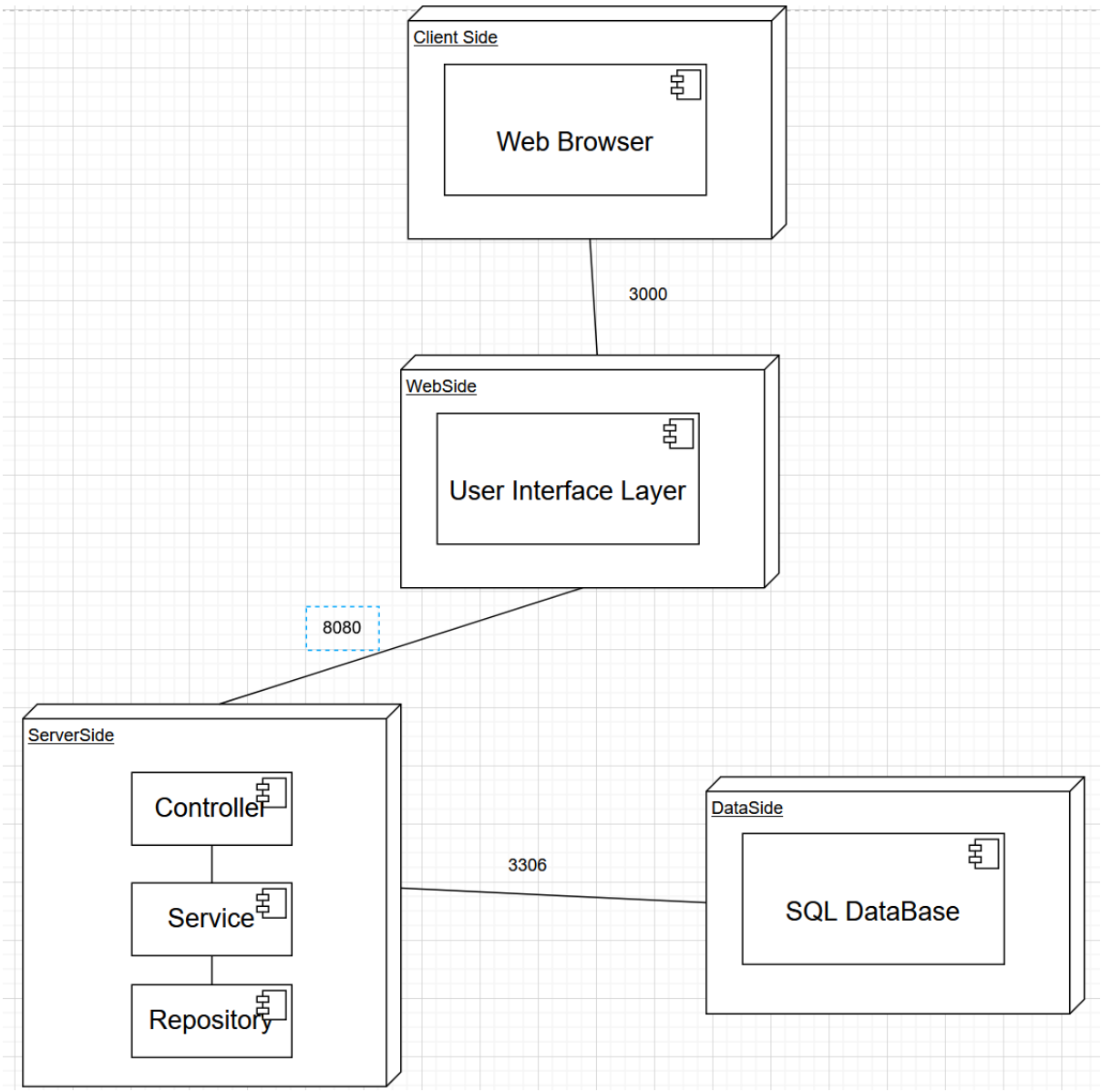


Diagrama de componente

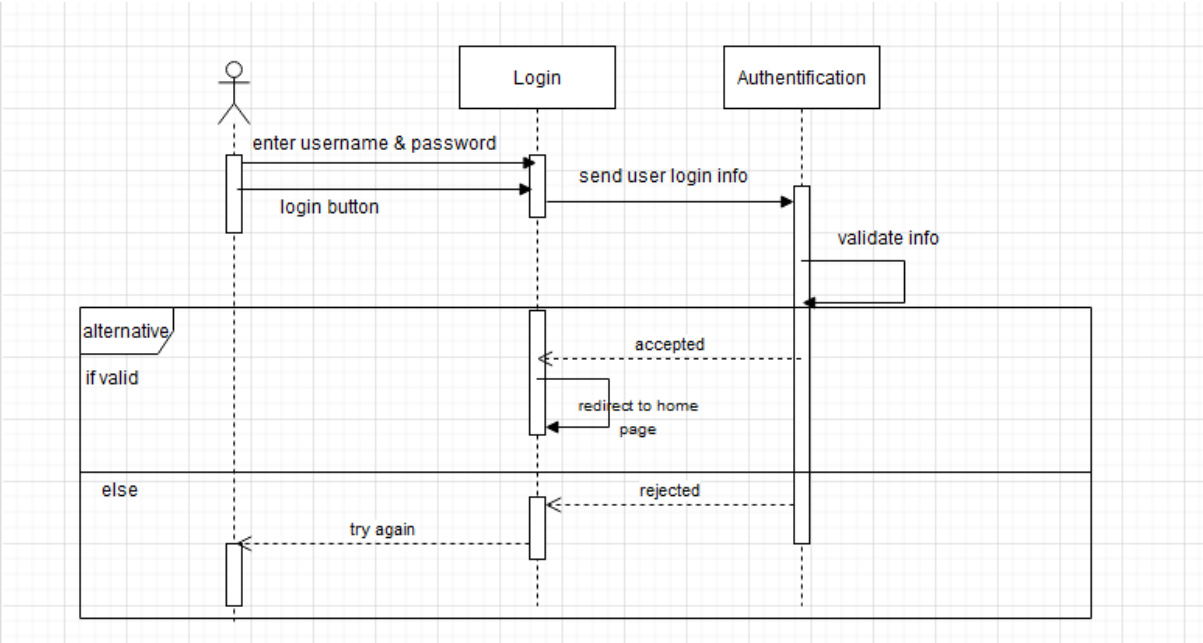


Deployment Diagram

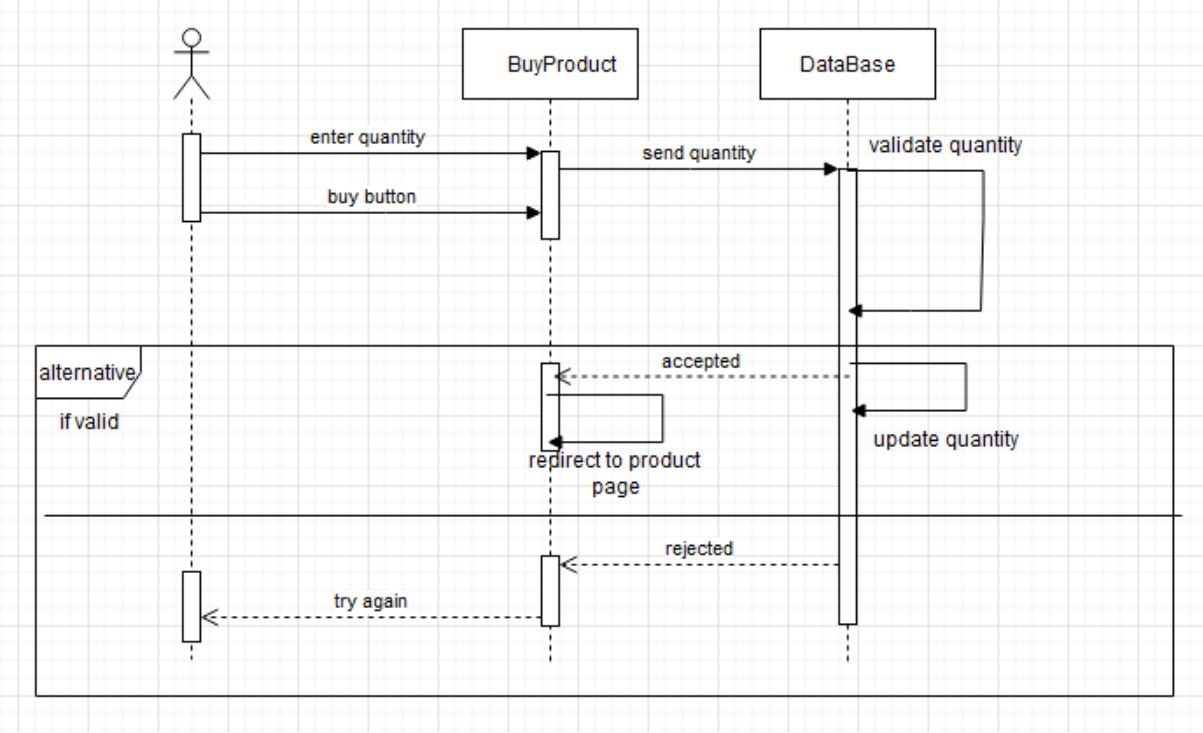
Deliverable 3

Design Model

Dynamic Behavior

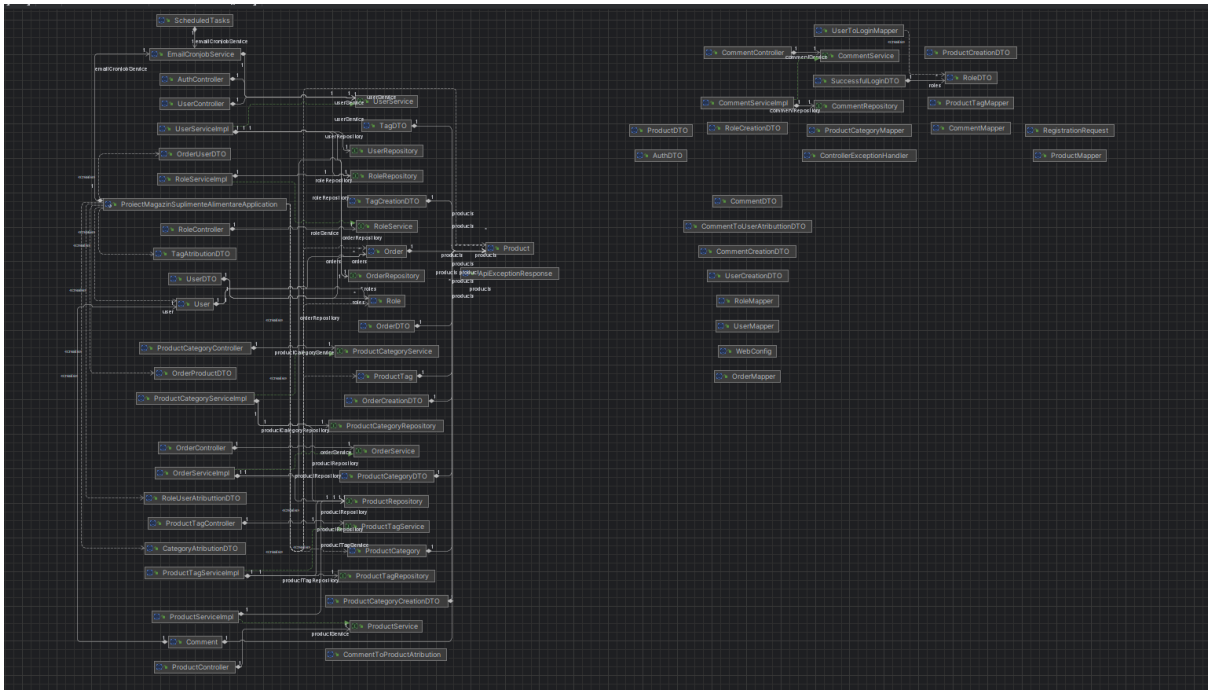


Login interaction diagram



Buy product interaction diagram

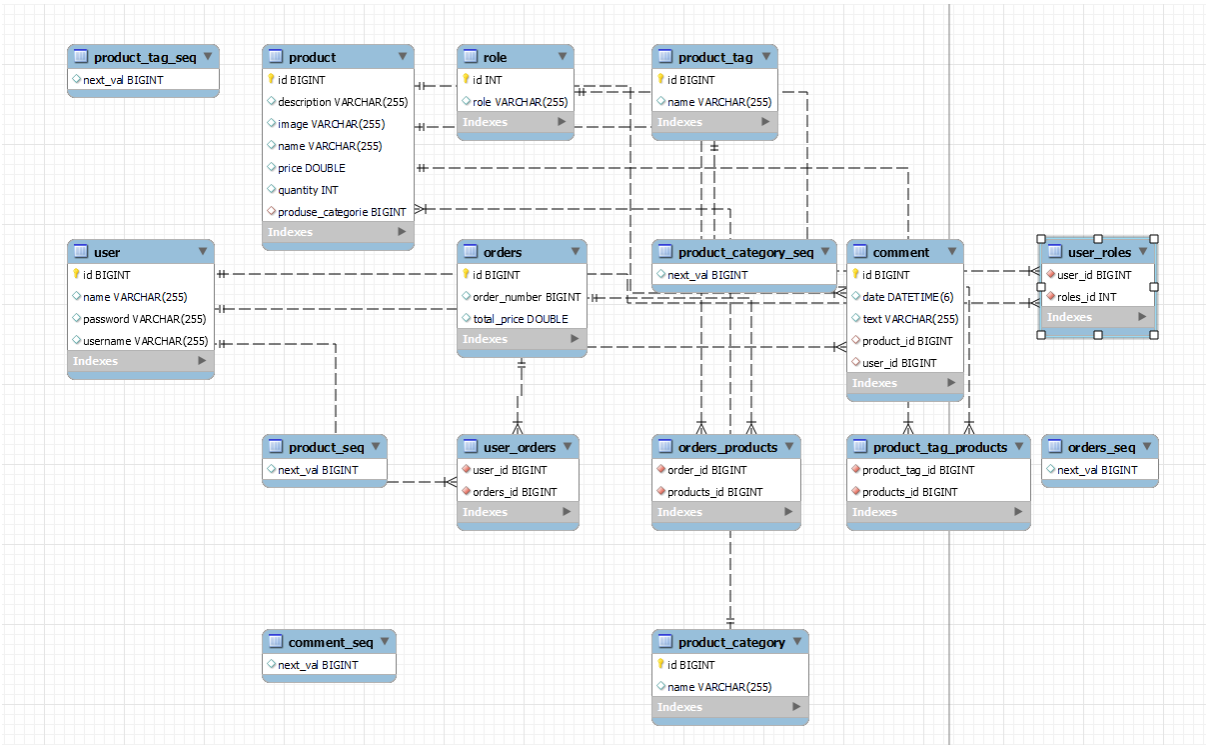
Class Diagram



Class diagram

Am ales să folosesc design pattern- ul creațional Builder pentru a nu avea cod repetitiv în aplicație și pentru a separa partea de creație a obiectelor care compun obiectul final. De asemenea am vrut să folosesc acest design pattern pentru a putea controla pașii de construcție a obiectelor.

Data Model



Model Diagram

System Testing

Pe parte de testare am folosit metodologia de testare mocking pentru a verifica salvarea și ștergerea comenzilor, cât și verificare salvării și ștergerii categoriilor produselor din baza de date. Se utilizează adnotarea @Mock pentru a declara un obiect de probă orderRepository. Metoda verify are rolul de a valida interacțiunile dintre service și repository. Avem următoarele cazuri de testare:

testSaveOrder:

Scop: Testează că metoda saveOrder a clasei OrderServiceImpl invocă corect metoda save a orderRepository cu comanda furnizată.

Configurare: Se creează un obiect Order și metoda save a repository-ului este stubbed pentru a returna același obiect order.

Execuție: Metoda saveOrder este apelată cu comanda.

Verificare: Se verifică că metoda save a repository-ului a fost apelată exact o dată cu comanda corectă.

testDeleteOrder:

Scop: Asigură că metoda deleteOrder a clasei OrderServiceImpl apelează metoda deleteById a repository-ului cu ID-ul corect.

Configurare: Se predefinește un ID pentru comandă (1L).

Execuție: Metoda deleteOrder este apelată cu ID-ul predefinit.

Verificare: Se confirmă că deleteById a fost invocat o dată cu ID-ul dat.

testSaveProductCategory:

Scop: Testează că metoda save a clasei ProductCategoryServiceImpl invocă corect metoda save a productCategoryRepository cu categoria de produs furnizată.

Configurare: Se creează un obiect ProductCategory și metoda save a repository-ului este stubbed pentru a returna același obiect categorie.

Execuție: Metoda save este apelată cu categoria de produs.

Verificare: Se verifică că metoda save a repository-ului a fost apelată exact o dată cu categoria de produs corectă.

testDeleteProductCategory:

Scop: Asigură că metoda deleteProductCategory a clasei ProductCategoryServiceImpl apelează metoda deleteById a repository-ului cu ID-ul corect.

Configurare: Se predefinește un ID pentru categoria de produs (1L).

Execuție: Metoda deleteProductCategory este apelată cu ID-ul predefinit.

Verificare: Se confirmă că deleteById a fost invocat o dată cu ID-ul dat.

De asemenea requesturile la server le-am testat cu Postman.

Future Improvements

Ca implementări viitoare, magazinul online ar putea integra un API de la un procesator de plata cum ar fi STRIPE pentru a putea procesa comenzile si pentru a simula acțiunea de cumpărare a produselor.

De asemenea am putea adăuga un chat online cu clienții, unde un client să vorbească un administrator al magazinului.

Conclusion

În concluzie, proiectul de magazinul online de suplimente alimentare a evidențiat o structură detaliată și bine planificată, care cuprinde cerințele funcționale și non-funcționale, modele de cazuri de utilizare, precum și o arhitectură robustă a sistemului. Proiectul incorporează tehnologii precum Java cu Spring Boot pentru backend și ReactJS pentru frontend.

Bibliography

1. https://www.baeldung.com/cs/layered-architecture
2. https://refactoring.guru/design-patterns
3. https://mailtrap.io/blog/sending-email-using-java/
4. https://medium.com/@bectorhimanshu/how-to-create-a-scheduled-task-using-a-cron-job-in-spring-boot-a1987e679d60
5. https://www.codeandweb.com/babeledit/tutorials/how-to-translate-your-react-app-with-react-i18next