



E-Commerce Clothing Website

Name: Sirbu Paul
Dan
Group: 30236

Specificații ale Proiectului

1.1 Scopul Proiectului

Scopul acestui proiect constă în crearea unui site pentru haine , oferind o experiență similară de cumpărături online și o gamă largă de produse cum ar fi tricouri, hanorace, etc. Obiectivul este de a satisface nevoile clienților din întreaga țară, oferindu-le acces ușor și convenabil la produse de calitate, la prețuri competitive și cu servicii de înaltă calitate.

1.2 Obiectivele Proiectului

Prin utilizarea tehnologiilor full stack, în special Spring și React, obiectivele proiectului sunt:

Ce folosim in aplicatia noastra: Utilizând Spring pentru backend și React pentru frontend, se vizează navigarea autentica pentru shopping online, oferind o interfață modernă și interactivă.

Diversitate și Disponibilitate a Produselor: Prin intermediul Spring Data JPA și React.js, se va gestiona eficient o gamă variată de produse și categorii, prezentându-le într-un mod atrăgător și ușor de navigat.

Conveniență și Accesibilitate: Utilizând Spring Boot pentru backend și React.js pentru frontend, se va crea o experiență de cumpărături online convenabilă și accesibilă pentru utilizatori.

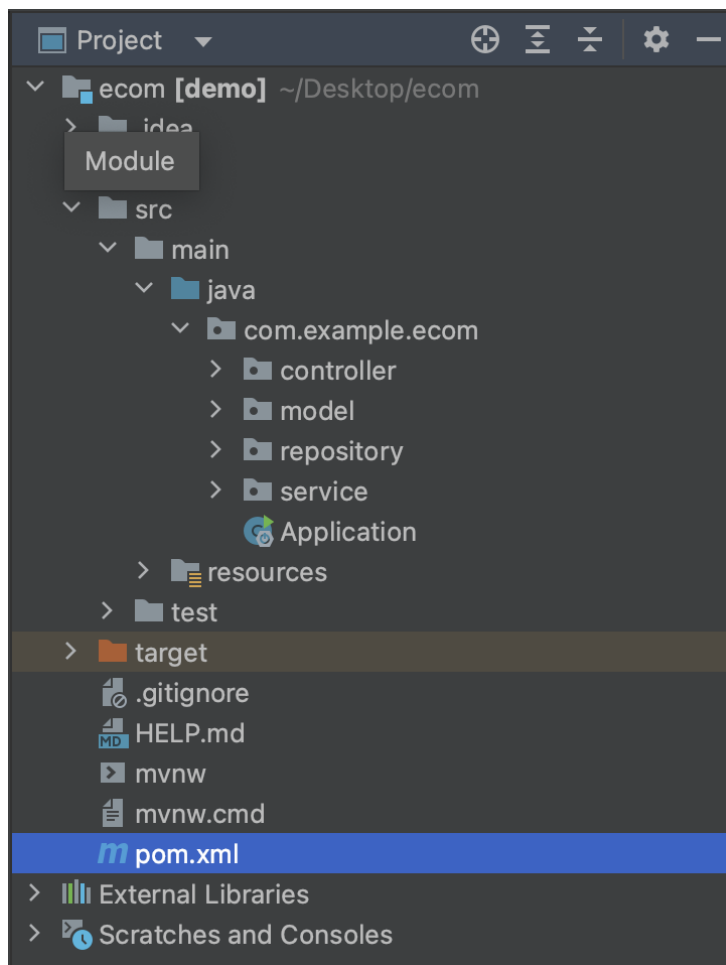
Servicii de Înaltă Calitate: Prin Spring Security și alte module din cadrul Spring, se va asigura securitatea datelor utilizatorilor și se va oferi o experiență de utilizare sigură și protejată împotriva amenințărilor online.

1.3 Cerințele Proiectului

Platforma propusă va reprezenta un exemplu de excelență în domeniul comerțului electronic, folosind cele mai recente tehnologii și practici în dezvoltarea aplicațiilor full stack. Backend-ul va fi implementat folosind cadrul de lucru Spring Java, iar frontend-ul va fi dezvoltat cu React.js.

1.4 Arhitectura Proiectului

Arhitectura proiectului se bazează pe o abordare modernă de tip client-server, în care frontend-ul și backend-ul comunică între ele. Pe partea de backend, s-a adoptat o arhitectură bazată pe model-view-controller (MVC), folosind Spring Java, iar comunicarea cu baza de date se realizează prin PostgreSQL. Pe partea de frontend, s-a ales React.js pentru dezvoltarea interfeței de utilizator.



1.5 Dependințele Proiectului

Proiectul utilizează Maven pentru gestionarea dependențelor. Fișierul pom.xml conține lista dependențelor necesare pentru a rula aplicația. Mai jos sunt enumerate principalele dependențe utilizate în proiect:

spring-boot-starter-data-jpa: Oferă funcționalități pentru interacțiunea cu baza de date folosind Spring Data JPA.

spring-boot-starter-web: Oferă funcționalități pentru dezvoltarea aplicațiilor web folosind Spring MVC.

postgresql: Driver-ul pentru conectarea la baza de date PostgreSQL, utilizat în timpul runtime-ului.

spring-boot-starter-test: Oferă dependențele necesare pentru testarea unitară și de integrare a aplicației.

org.projectlombok:lombok: Oferă funcționalități de generare automată a codului boilerplate pentru Java, reducând astfel cantitatea de cod necesară.

hibernate-core: Implementarea framework-ului Hibernate, care este utilizată de Spring Data JPA pentru interacțiunea cu baza de date.

spring-boot-starter-validation: Oferă funcționalități pentru validarea datelor în cadrul aplicației folosind Spring Validation.

Aceste dependențe sunt esențiale pentru funcționarea și dezvoltarea aplicației noastre, oferind funcționalitățile necesare pentru interacțiunea cu baza de date, dezvoltarea web, testarea și validarea datelor. Utilizarea Maven și specificarea dependențelor în fișierul pom.xml facilitează gestionarea și actualizarea acestora în cadrul proiectului.

2 Cerinte functionale

2.1 Sign In

2.2 Sign Up

2.3 CRUD op

Managementul Adreselor:

Utilizatorii ar trebui să poată vizualiza toate adresele asociate contului lor.

Utilizatorii ar trebui să poată vizualiza o adresă specifică după ID-ul acesteia.

Utilizatorii ar trebui să poată crea o adresă nouă.

Utilizatorii ar trebui să poată actualiza o adresă existentă.

Utilizatorii ar trebui să poată șterge o adresă.

Managementul Utilizatorilor:

Utilizatorii ar trebui să poată crea un cont nou.

Utilizatorii ar trebui să poată vizualiza toți utilizatorii înregistrați.

Utilizatorii ar trebui să poată vizualiza profilul unui utilizator specific după ID-ul acestuia.

Utilizatorii ar trebui să poată actualiza informațiile profilului lor.

Utilizatorii ar trebui să poată șterge contul lor.

Managementul Produselor:

Utilizatorii ar trebui să poată adăuga un produs nou în inventarul magazinului.

Utilizatorii ar trebui să poată vizualiza toate produsele disponibile.

Utilizatorii ar trebui să poată vizualiza detaliile unui produs specific.

Utilizatorii ar trebui să poată actualiza informațiile despre produs.

Utilizatorii ar trebui să poată elimina un produs din inventar.

Managementul Comenzilor:

Utilizatorii ar trebui să poată plasa o comandă nouă.

Utilizatorii ar trebui să poată vizualiza toate comenzile anterioare.

Utilizatorii ar trebui să poată vizualiza detaliile unei comenzi specifice.

Utilizatorii ar trebui să poată actualiza informațiile despre comandă.

Utilizatorii ar trebui să poată anula o comandă.

Managementul Cantităților din Comenzi:

Utilizatorii ar trebui să poată specifica cantitățile de produse atunci când plasează o comandă.

Utilizatorii ar trebui să poată vizualiza toate cantitățile din comenzi.

Utilizatorii ar trebui să poată vizualiza detaliile unei cantități de comandă specifice.

Utilizatorii ar trebui să poată actualiza cantitățile din comenzi.

Utilizatorii ar trebui să poată elimina cantitățile din comenzi.

Aceste cerințe funcționale acoperă operațiile de bază pe care utilizatorii le pot efectua pe site-ul magazinului tău online de îmbrăcăminte, inclusiv gestionarea adreselor, a conturilor utilizatorilor, a produselor, a comenzilor și a cantităților din comenzi. Dacă ai orice alte caracteristici sau cerințe specifice, nu ezita să le adaugi în această listă.

2.4 ORM relations

Adresă (Address):

Fiecare utilizator poate avea una sau mai multe adrese asociate contului său. Aceasta este o relație One-to-Many între utilizator (LocalUser) și adrese.

O adresă poate fi asociată unui singur utilizator. Aceasta este o relație Many-to-One între adrese și utilizator (LocalUser).

Utilizator Local (LocalUser):

Un utilizator local poate plasa una sau mai multe comenzi pe site. Acesta este o relație One-to-Many între utilizatorul local (LocalUser) și comenzile web (WebOrder).

O comandă web poate fi plasată de un singur utilizator local. Acesta este o relație Many-to-One între comenzile web și utilizatorul local.

Produs (Product):

Un produs poate fi inclus în una sau mai multe comenzi. Aceasta este o relație One-to-Many între produse și cantitățile comenzilor web (WebOrderQuantities).

O cantitate de comandă web poate fi asociată unui singur produs. Aceasta este o relație Many-to-One între cantitățile comenzilor web și produs.

Comandă Web (WebOrder):

O comandă web poate conține una sau mai multe cantități de produse. Aceasta este o relație One-to-Many între comenzile web și cantitățile comenzilor web.

O cantitate de comandă web este asociată unei singure comenzi web. Aceasta este o relație Many-to-One între cantitățile comenzilor web și comanda web.

Cantități de Comandă Web (WebOrderQuantities):

O cantitate de comandă web este legată de un singur produs. Aceasta este o relație Many-to-One între cantitățile comenzilor web și produs.

O cantitate de comandă web este asociată unei singure comenzi web. Aceasta este o relație Many-to-One între cantitățile comenzilor web și comanda web.

Aceste relații sunt importante pentru a înțelege modul în care obiectele sunt legate între ele în cadrul aplicației tale de e-commerce. Ele facilitează accesul și manipularea datelor în baza de date în mod corespunzător și eficient.

3 Use Case UML

Diagrama de cazuri de utilizare în limbajul de modelare UML (Unified Modeling Language) este o tehnică folosită în ingineria software pentru a defini și a prezenta interacțiunile dintre actori (utilizatori externi) și sistem. Această diagramă oferă o perspectivă structurală asupra funcționalităților sistemului și modului în care acestea sunt accesate de către actori.

Un caz de utilizare descrie o situație sau un scenariu în care sistemul este utilizat de către unul sau mai mulți actori pentru a atinge un anumit obiectiv. Aceste cazuri de utilizare sunt reprezentate sub formă de elipse în diagrama UML, iar relațiile dintre cazurile de utilizare și actori sunt ilustrate prin linii de asociere.

În diagrama de cazuri de utilizare, actorii reprezintă entitățile externe care interacționează cu sistemul, indiferent dacă sunt utilizatori umani, alte sisteme sau dispozitive externe. Cazurile de utilizare sunt acțiuni sau secvențe de acțiuni care descriu modul în care actorii interacționează cu sistemul pentru a îndeplini anumite obiective.

3.1 Caz de utilizare LocalUser

Caz de utilizare: Gestionarea contului și vizualizarea produselor

Nivel: Principal (core)

Actor Principal: Utilizatorul (LocalUser)

Scenariul principal de succes:

Utilizatorul dorește să-și creeze un cont:

Utilizatorul accesează funcția "Sign up".

Sistemul solicită introducerea datelor necesare pentru înregistrare (ex: nume, email, parolă).

Utilizatorul completează detaliile și confirmă înregistrarea.

Sistemul validează datele și creează contul utilizatorului.

Utilizatorul primește confirmarea că contul a fost creat cu succes.

Utilizatorul dorește să se autentifice în contul său existent:

Utilizatorul accesează funcția "Sign in".

Sistemul solicită introducerea adresei de email și a parolei.

Utilizatorul furnizează detaliile de autentificare.

Sistemul validează datele și autentifică utilizatorul.

Utilizatorul primește confirmarea că autentificarea a fost efectuată cu succes.

După autentificare, utilizatorul dorește să vizualizeze toate produsele disponibile:

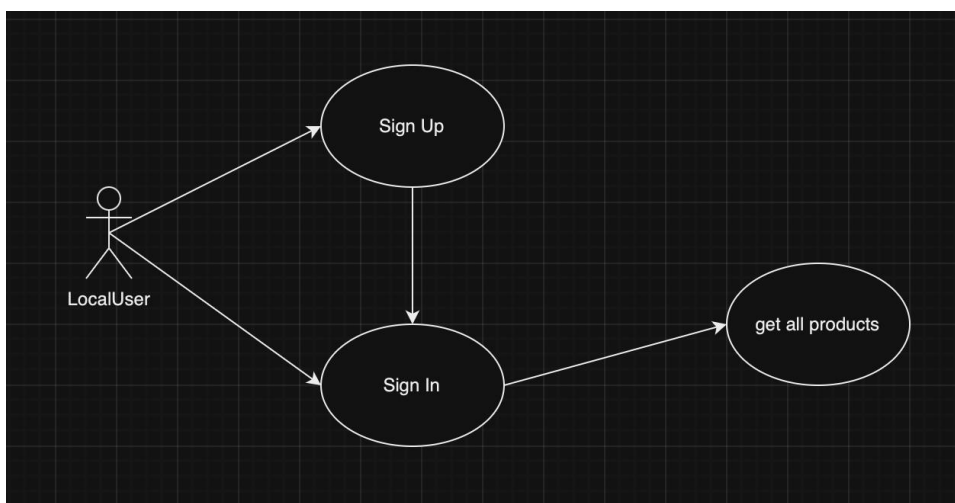
Utilizatorul accesează funcția "get all products".

Sistemul afișează o listă cu toate produsele disponibile în magazin.

Extensii:

Dacă datele introduse în timpul înregistrării nu sunt valide sau există deja un cont asociat cu adresa de email introdusă, sistemul afișează un mesaj de eroare și solicită utilizatorului să furnizeze informații valide sau să încerce să se autentifice în contul existent.

Dacă datele de autentificare introduse nu sunt valide sau nu corespund cu un cont existent, sistemul afișează un mesaj de eroare și solicită utilizatorului să reintroducă datele corecte sau să își recupereze parola.



3.2 Caz de utilizare Admin

Use-Case: Gestionarea funcționalităților ca administrator

Nivel: Principal (core)

Actor Principal: Administratorul (ADMIN)

Scenariul principal de succes:

Administratorul dorește să se autentifice în sistem:

Actorul accesează funcția "Sign in".

Sistemul solicită introducerea adresei de email și a parolei administratorului.

Administratorul furnizează detaliile de autentificare.

Sistemul validează datele și autentifică administratorul.

Administratorul primește confirmarea că autentificarea a fost efectuată cu succes.

După autentificare, administratorul are patru opțiuni disponibile (pana la momentul implementarii):

a) Adăugare de produse noi:

Administratorul accesează funcția "saveProduct".

Sistemul permite introducerea detaliilor noului produs (ex: nume, descriere, preț, etc.).

Administratorul completează informațiile necesare și confirmă adăugarea produsului.

Sistemul validează datele și adaugă noul produs în baza de date.

b) Vizualizare produselor după id:

Administratorul accesează funcția " getProductById ".

Sistemul afișează produsul cu id-ul pe care îl căutam.

c) Actualizarea unui produs: updateProduct

Administratorul accesează funcția " updateproduct".

Sistemul solicită confirmarea administratorului.

După confirmare, produsul este actualizat din baza de date.

d) Stergerea unui produs:

Administratorul selectează opțiunea "deleteproduct" pentru a șterge un anumit produs.

Sistemul solicită confirmarea administratorului.

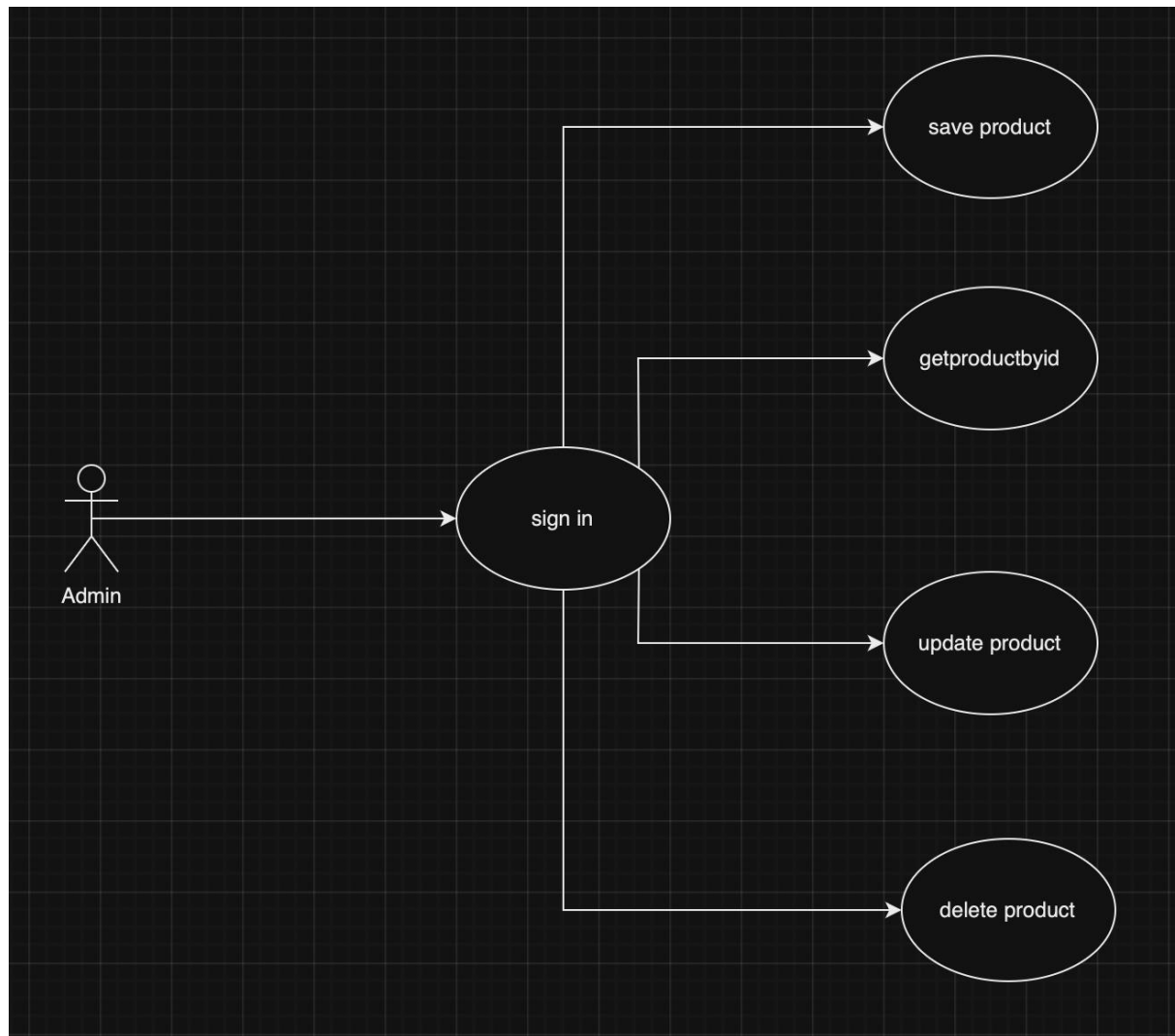
După confirmare, produsul este șters din baza de date.

Extensii:

În cazul în care datele introduse în timpul autentificării nu sunt valide sau nu corespund cu un cont de administrator existent, sistemul afișează un mesaj de eroare și solicită reintroducerea datelor corecte sau oferă posibilitatea recuperării parolei.

Dacă administratorul nu are drepturile necesare pentru a accesa anumite funcționalități (de exemplu, dacă este un cont de tip "guest"), sistemul îi va afișa un mesaj corespunzător și îi va limita accesul la anumite operații.

În cazul în care administratorul încearcă să efectueze operații care necesită permisiuni suplimentare (cum ar fi ștergerea unui utilizator sau a unui produs), sistemul



poate solicita confirmarea suplimentara a administratorului pentru a preveni stergerile accidentale sau neautorizate.

4 Specificatie Suplimentare

4.1 Cerințe Non-funcționale

Performanță: Descriere: Sistemul trebuie să fie capabil să gestioneze un număr mare de interacțiuni simultane ale utilizatorilor fără o degradare semnificativă a timpului de răspuns sau a capacității sistemului. Justificare: Într-un site de comerț electronic pentru îmbrăcăminte, performanța este crucială pentru a asigura o experiență de cumpărare fără probleme pentru utilizatori. Timpul de răspuns rapid și capacitatea mare de procesare sunt necesare pentru a gestiona un trafic potențial ridicat în perioadele de vârf, cum ar fi reducerile sau promoțiile.

Securitate: Descriere: Sistemul trebuie să implementeze măsuri de securitate robuste pentru a proteja datele utilizatorilor, inclusiv informațiile personale, detaliile de plată și istoricul comenzilor, împotriva accesului neautorizat, manipulării sau furtului. Justificare: Ca platformă de comerț electronic care gestionează date sensibile ale utilizatorilor, asigurarea unor măsuri de securitate puternice este crucială. Orice încălcare a securității ar putea duce la pierderea încrederii clienților, la responsabilități financiare și consecințe legale.

Scalabilitate: Descriere: Sistemul ar trebui să fie proiectat pentru a crește atât vertical, cât și orizontal, pentru a putea să facă față creșterii traficului utilizatorilor, a inventarului de produse și a complexității sistemului în timp. Justificare: Pe măsură ce afacerea crește, site-ul de comerț electronic trebuie să gestioneze volume mai mari de date, utilizatori și tranzacții. Scalabilitatea asigură că sistemul poate să se adapteze și să își extindă resursele fără probleme pentru a răspunde cerințelor crescânde fără a sacrifica performanța sau experiența utilizatorului.

Ușurința în utilizare: Descriere: Interfața utilizator trebuie să fie intuitivă, plăcută din punct de vedere vizual și accesibilă pe diferite dispozitive și dimensiuni de ecran pentru a oferi o experiență de cumpărare prietenoasă pentru utilizatori. Justificare: Ușurința în utilizare afectează direct satisfacția și retenția utilizatorilor. O interfață bine proiectată și ușor de utilizat îmbunătățește implicarea utilizatorului, reduce rata de respingere și încurajează vizitele și achizițiile repetate.

4.2 Constrângeri de Design

Stiva Tehnologică: Sistemul trebuie dezvoltat folosind Java pentru dezvoltarea backend-ului și React.js pentru dezvoltarea frontend-ului, conform standardelor și expertizei tehnologice ale organizației.

Unelte de Dezvoltare: Proiectul trebuie să utilizeze Git pentru controlul versiunilor și să urmeze metodologiile Agile de dezvoltare software pentru gestionarea proiectului.

Implementare în Cloud: Sistemul trebuie să fie implementat pe infrastructura de cloud AWS (Amazon Web Services) pentru scalabilitate, fiabilitate și conformitate cu standardele de securitate.

Pentru baza noastră de date, am optat pentru un sistem de gestionare a bazelor de date PostgreSQL, care este un sistem de bază de date relațional fiabil și larg utilizat în industrie. PostgreSQL oferă suport pentru operațiuni complexe de interogare și manipulare a datelor, iar modelul său relațional este potrivit pentru stocarea și gestionarea datelor noastre.

În ceea ce privește partea de frontend a

aplicației noastre, am ales să folosim React, o bibliotecă JavaScript modernă și populară pentru dezvoltarea interfețelor de utilizator interactive și dinamice. React oferă un mod eficient de a construi componente reutilizabile și de a gestiona starea aplicației noastre într-un mod clar și predictibil.

4.3 Glosar

Site de Comerț Electronic pentru Îmbrăcămintă: O platformă online în care utilizatorii pot naviga, cumpăra și vinde îmbrăcămintă și produse conexe peste internet.

Informații Personale: Date care identifică sau pot fi folosite pentru a identifica o persoană, cum ar fi numele, adresa, adresa de email și numărul de telefon.

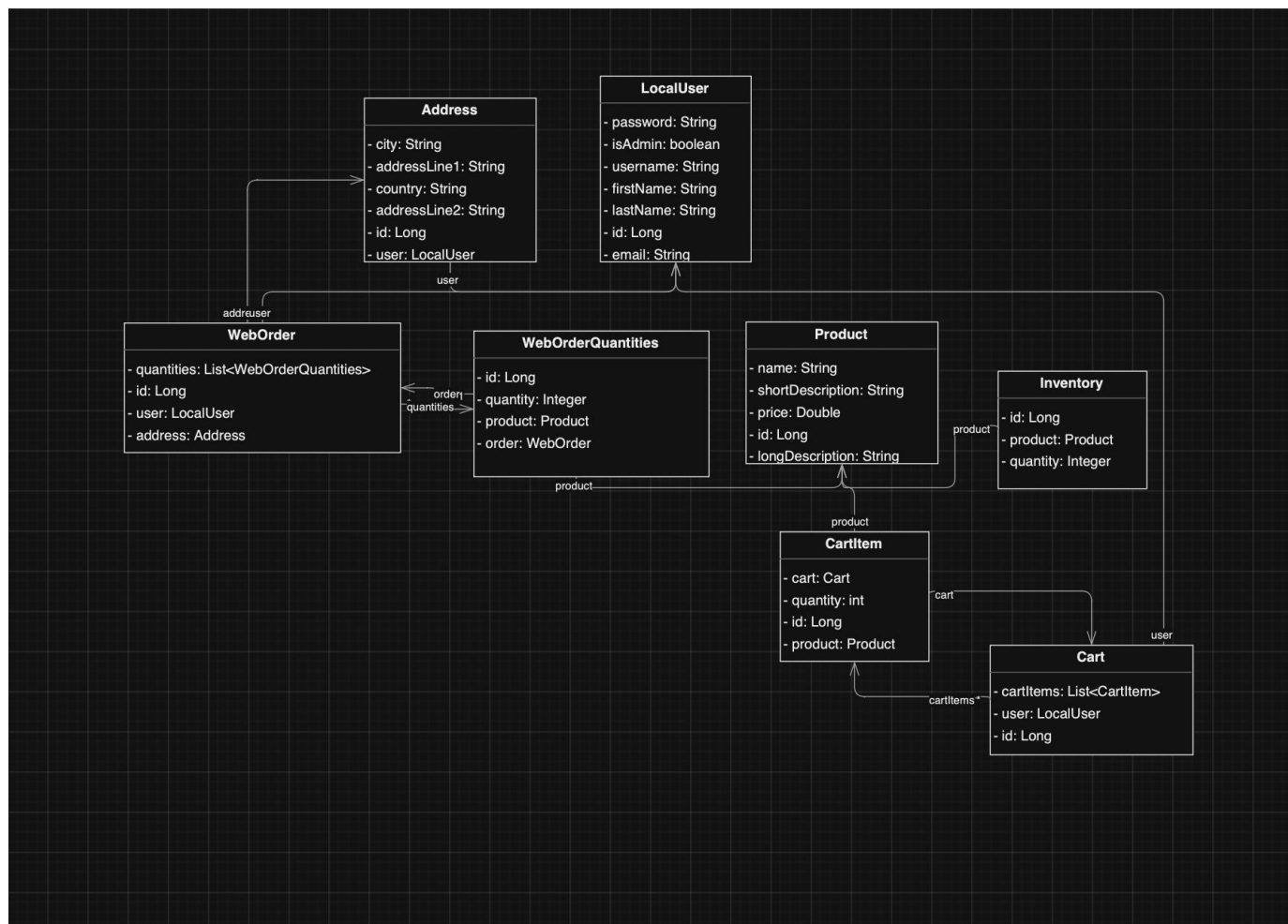
Detalii de Plată: Informații legate de metodele de plată, cum ar fi numerele de card de credit, detaliile contului bancar și adresele de facturare.

Istoricul Comenzilor: Un înregistrat al achizițiilor anterioare făcute de un utilizator, inclusiv detalii precum data comenzii, produsele achiziționate și starea tranzacției.

Scalare Verticală: Creșterea resurselor (CPU, memorie, stocare) ale unui singur server pentru a gestiona o încărcătură de lucru crescută.

Scalare Orizontală: Adăugarea de mai multe servere pentru a distribui încărcătura de lucru și a crește capacitatea sistemului.

5.Domain Model



Address (Adresă):

Această clasă stochează informații despre adresele utilizatorilor, cum ar fi linia de adresă, orașul și țara.

Are o relație Many-to-One cu clasa LocalUser pentru a indica că o adresă aparține unui utilizator local.

Cart (Coș de cumpărături):

Reprezintă coșul de cumpărături al unui utilizator.

Are o listă de CartItem-uri (elementele din coșul de cumpărături).

Deține o referință către utilizatorul local care deține coșul de cumpărături.

CartItem (ElementCoșCumpărături):

Stochează informații despre un anumit produs și cantitatea acestuia din coșul de cumpărături.

Este legat de un produs și de coșul de cumpărături căruia îi aparține.

Inventory (Stoc):

Reprezintă stocul unui anumit produs disponibil pentru achiziție.

Are o relație One-to-One cu clasa Product pentru a indica că fiecare înregistrare de inventar este asociată cu un singur produs.

LocalUser (UtilizatorLocal):

Describe un utilizator înregistrat în sistemul de comerț electronic.

Conține informații precum numele de utilizator, parola, adresă de email, prenume și nume de familie.

Poate fi marcat ca administrator sau utilizator obișnuit.

Product (Produs):

Reprezintă un produs disponibil în magazinul online.

Include detalii despre produs, cum ar fi numele, descrierea scurtă și lungă și prețul.

WebOrder (ComandăWeb):

Este o comandă plasată de un utilizator pe site-ul web.

Conține detalii precum utilizatorul care a plasat comanda și adresa de expediere asociată.

Are o listă de WebOrderQuantities pentru a ține evidența cantităților de produse comandate.

WebOrderQuantities (CantitățiComandăWeb):

Stochează cantitățile de produse comandate împreună cu informațiile de comandă asociate.

Este legat de un produs și de o comandă web căruia îi aparține.

Payment

Integrarea plăților cu Stripe într-o aplicație Spring Boot și React implică configurarea și comunicarea între backend-ul Spring Boot și frontend-ul React, folosind serviciile și bibliotecile oferite de Stripe.

Pe partea de backend avem :

Clasa StripeConfig este responsabilă pentru configurarea inițială a Stripe în cadrul aplicației. Aceasta setează cheia API Stripe necesară pentru autentificarea și efectuarea tranzacțiilor.

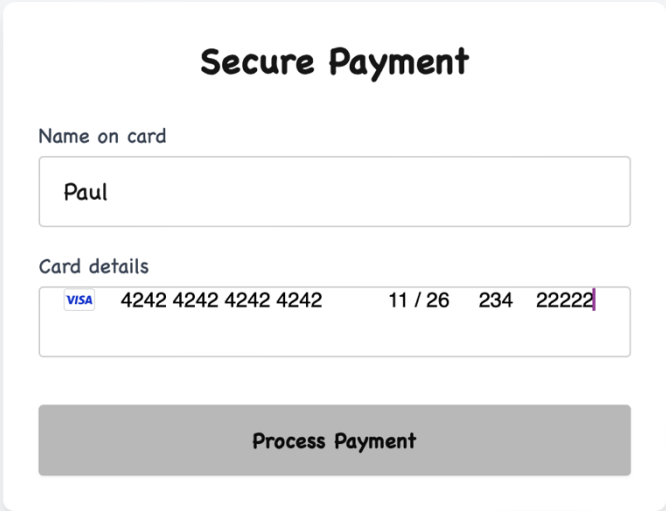
Clasa PaymentController expune un endpoint REST pentru procesarea plăților prin intermediul Stripe.

Clasa CardDetails reprezintă modelul pentru detaliile cardului de credit utilizate în procesarea plăților.

Interfața CardDetailsRepository gestionează operațiunile CRUD pentru entitatea CardDetails.

Clasa PaymentService conține logica principală pentru procesarea plăților utilizând Stripe.

Pe partea de frontend avem o pagina dedicata platii si anume clasa Payment care este responsabilă de afișarea și gestionarea formularului de plată în aplicația ta React.



A secure payment form titled "Secure Payment" is displayed on a light gray background. The form is white with rounded corners and a subtle shadow. It contains two input fields: "Name on card" with the value "Paul" and "Card details" with the value "VISA 4242 4242 4242 4242 11 / 26 234 22222". Below these fields is a gray button labeled "Process Payment".

Chat

Chat-ul in timp real intre utilizatori este o componenta esentiala a oricarei aplicatii care vizeaza interactiunea si comunicarea intre utilizatori. Aceasta functionalitate permite utilizatorilor sa comunice in timp real, sa impartaseasca idei, sa discute sau sa colaboreze, imbunatatind astfel experienta de utilizare si promovand interactivitatea.

Arhitectura si Implementare

WebSocket-uri si Protocol STOMP: Implementarea chat-ului in timp real se bazeaza pe WebSocket-uri, un protocol de comunicatie bidirectionala care permite comunicarea in timp real intre server si client. Protocolul STOMP (Simple Text Oriented Messaging Protocol) este folosit pentru a defini mesajele si destinatiile pe care le foloseste aplicatia.

Biblioteci Utilizate: Pentru gestionarea comunicarii WebSocket si a protocolului STOMP, aceasta aplicatie foloseste bibliotecile `sockjs-client` si `@stomp/stompjs`. Aceste biblioteci faciliteaza conectarea la serverul WebSocket, trimiterea si receptia mesajelor intr-un mod eficient si sigur.

Comunicare intre Frontend si Backend: Chat-ul in timp real este implementat intr-o componenta React pe frontend si comunica cu serverul backend prin intermediul API-ului REST. In plus, pentru a obtine numele utilizatorului, componenta face o cerere catre backend pentru a extrage aceasta informatie pe baza ID-ului utilizatorului.

Functionalitati si Interfata Utilizator Conectarea la Serverul WebSocket: In momentul incarcarii componentei, aceasta se conecteaza la serverul WebSocket utilizand bibliotecile

mentionate anterior. Aceasta conexiune este esentiala pentru trimiterea si receptia mesajelor in timp real.

Trimiterea si Receptia Mesajelor: Utilizatorii pot trimite si receptiona mesaje in timp real prin intermediul chat-ului. Mesajele sunt afisate intr-o lista continand numele utilizatorului si continutul mesajului.

Identificarea Utilizatorilor: Pentru a identifica utilizatorii, numele lor sunt afisate in campul de introducere a mesajului. Acest lucru se realizeaza prin intermediul unei cereri catre API-ul nostru pentru a extrage numele utilizatorului pe baza ID-ului furnizat in parametrii rutei.

Interactiune Intuitiva: Interfata utilizatorului este intuitiva si usoara de folosit. Utilizatorii pot introduce mesaje intr-un camp de text si le pot trimite printr-un clic pe butonul de trimitere.

Importanta si Impactul in Aplicatie

Chat-ul in timp real intre utilizatori are un impact semnificativ asupra experientei de utilizare si interactivitatii aplicatiei. Acesta permite utilizatorilor sa comunice si sa interactioneze in timp real, imbunatatind colaborarea si facilitand schimbul de informatii. De asemenea, promoveaza implicarea utilizatorilor si ofera o modalitate eficienta de a rezolva probleme sau de a clarifica intrebari in timp real.

Chat

Home | Chat

P

Paul
buna

R

Robert
buna

Enter your nicknam Type a message

SEND

XML

Funcționalitatea XML este responsabilă pentru convertirea datelor utilizatorului într-un format XML și exportul acestor date sub forma unui fișier XML. Acest lucru permite utilizatorului să-și descarce datele de profil într-un format structurat și ușor de citit, care poate fi apoi folosit pentru diverse scopuri, cum ar fi stocarea locală sau importul în alte aplicații.

Atunci când utilizatorul apasă butonul "Export as XML", funcția `convertUserDataToXML` este apelată pentru a transforma datele utilizatorului din starea componentei într-un șir XML. Acest șir XML este apoi transformat într-un obiect Blob, care este un obiect binar utilizat pentru stocarea datelor brute. Prin crearea unui URL pentru acest obiect Blob și adăugarea

unui element de ancoră care să-l utilizeze ca sursă, utilizatorul poate descărca apoi fișierul XML creat.

În general, această funcționalitate XML oferă utilizatorilor posibilitatea de a păstra o copie a datelor lor de profil într-un format ușor de gestionat și interoperabil, ceea ce poate fi util în diverse contexte, cum ar fi backup-ul datelor, migrarea între platforme sau partajarea datelor cu alte date.

```
<?xml version="1.0" encoding="UTF-8"?>
<user>
  <id>553</id>
  <firstName>Customer</firstName>
  <lastName>Haaaaaa</lastName>
  <email>custom_email@example.com</email>
  <username>custom_username</username>
  <password>$2a$10$z07xKHLxUqHzKipc.blHo0.1bJg9LrcBL4xcy/
cLE9MJDYH//KHUy</password>
</user>
```

User Profile

Home | User Profile

Your Firstname

Customer

Your Username

custom_username

Your Lastname

Haaaaaa

Your Password

.....

Your Email

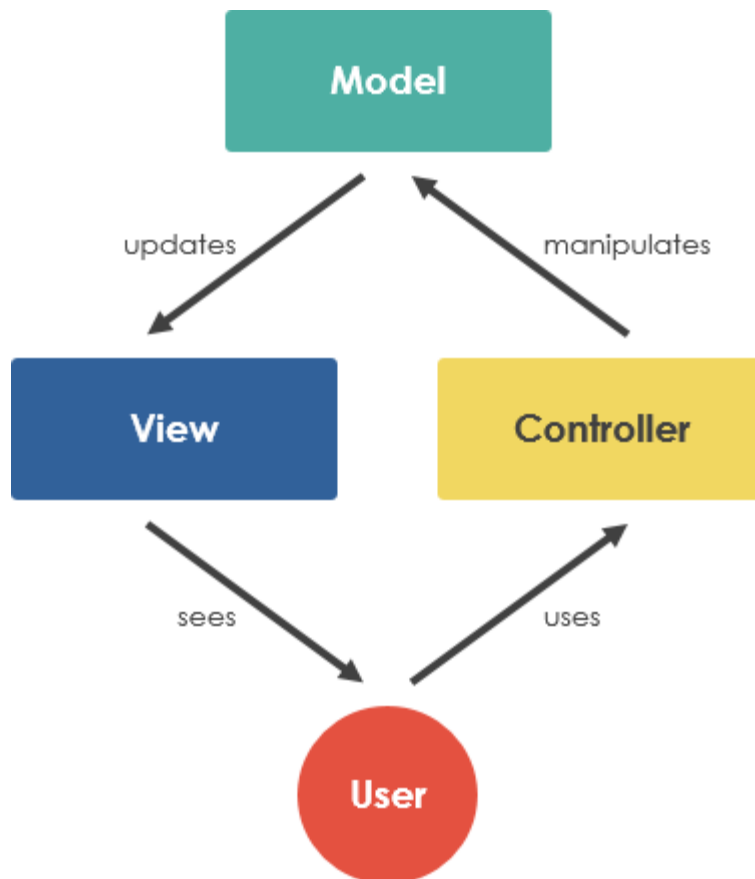
custom_email@example.com

Update Profile

Export as XML

6. Architectural design

6.1 Conceptual Architecture



Modelul de arhitectură/design Model-View-Controller (MVC) este un model care separă o aplicație în trei componente logice principale: Model, View și Controller. Fiecare componentă arhitecturală este construită pentru a gestiona aspecte specifice de dezvoltare a unei aplicații. Izolează logica de afaceri și stratul de prezentare unul față de celălalt. Inițial, a fost folosit tradițional pentru interfețele grafice cu utilizatorul (GUI) desktop. În prezent, MVC este unul dintre cele mai frecvent utilizate cadre de dezvoltare web standard din industrie pentru crearea de proiecte scalabile și extensibile. Este folosit și pentru proiectarea aplicațiilor mobile.

MVC a fost creat de Trygve Reenskaug. Scopul principal al acestui model de design a fost rezolvarea problemei controlului de către utilizatori a unui set

mare și complex de date prin împărțirea unei aplicații mari în secțiuni specifice, fiecare având propria sa scop.

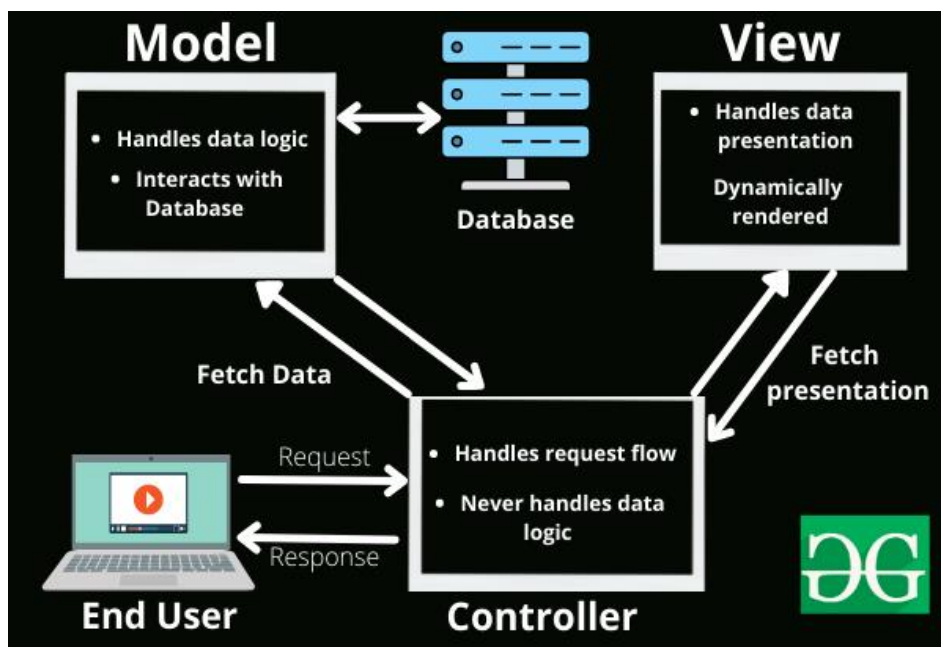
Caracteristici ale MVC:

Oferă o separare clară a logicii de afaceri, logicii UI și logicii de intrare.

Oferă control complet asupra HTML-ului și URL-urilor, ceea ce face ușoară proiectarea arhitecturii aplicațiilor web.

Este un component puternic de mapare a URL-urilor cu ajutorul căruia putem construi aplicații cu URL-uri inteligibile și căutabile.

Suportă Dezvoltarea Dirijată de Teste (TDD).



Modelul MVC include următoarele 3 componente:

Controller

Controller este componenta care permite interconexiunea între vederi și model, acționând ca un intermediar. Controlorul nu trebuie să se preocupe de logica de manipulare a datelor, ci doar îi spune modelului ce să facă. Procesează toată logica de afaceri și cererile primite, manipulează datele folosind componenta Model și interacționează cu Vederea pentru a afișa rezultatul final.

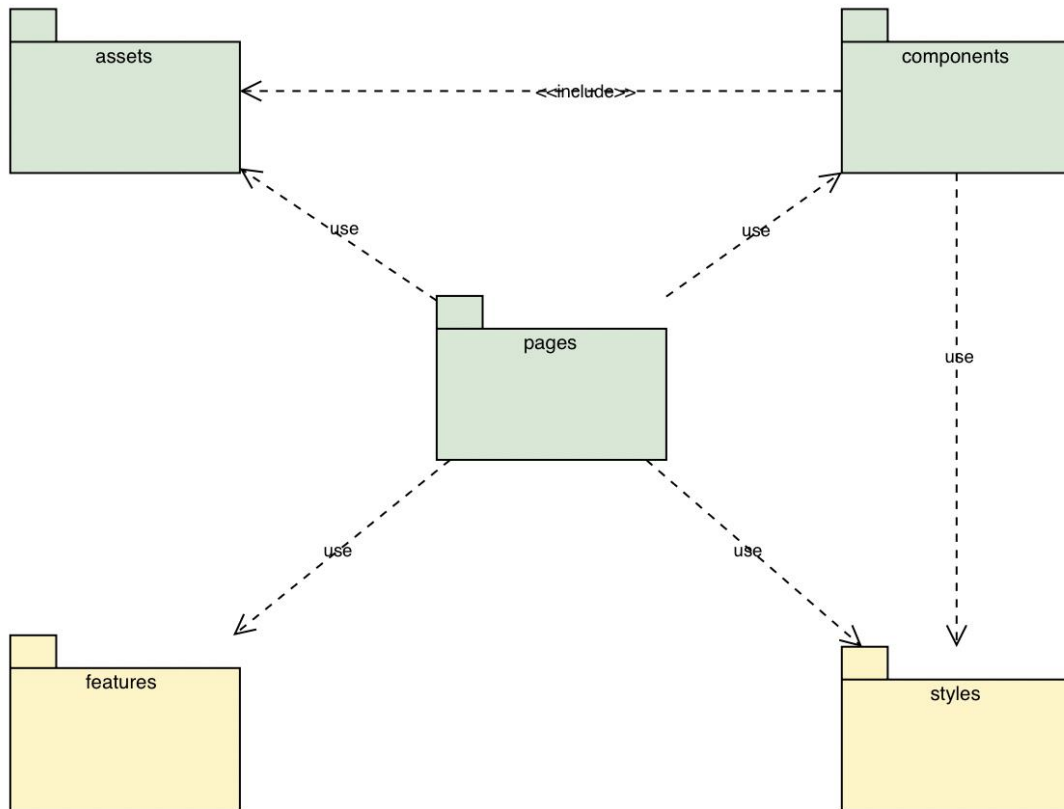
Model

Componenta Model corespunde întregii logici legate de date cu care lucrează utilizatorul. Aceasta poate reprezenta fie datele care sunt transferate între componentele Vedere și Controlor, fie orice alte date legate de logica de afaceri. Poate adăuga sau prelua date din baza de date. Răspunde cererilor controlorului deoarece controlorul nu poate interacționa cu baza de date de unul singur. Modelul interacționează cu baza de date și furnizează datele necesare înapoi controlorului.

View

Componenta View este folosită pentru toată logica UI (interfață utilizator) a aplicației. Generează o interfață utilizator pentru utilizator. Vederea este creată de datele colectate de componenta model, dar aceste date nu sunt preluate direct, ci prin intermediul controlorului. Vederea interacționează doar cu controlorul.

6.2 Package Design



1.Assets :

Pachetul "assets" conține în mod obișnuit fișiere statice utilizate de aplicație, cum ar fi imagini, fonturi, pictograme sau alte fișiere media.

Aceste resurse sunt adesea folosite de componente interfeței de utilizator (UI) pentru a îmbunătăți reprezentarea vizuală a aplicației.

În dezvoltarea web, resursele sunt de obicei stocate în directoare precum "imagini", "fonturi", "pictograme", etc.

2.Components :

Pachetul "components" de obicei conține componente reutilizabile de UI sau blocuri de construcție ale interfeței de utilizator a aplicației.

Aceste componente pot include butoane, formulare, bare de navigare, carduri, etc., care pot fi reutilizate pe mai multe pagini sau funcționalități în cadrul aplicației.

Organizarea componentelor UI într-un pachet separat ajută la menținerea unui cod modular și scalabil.

3.Features :

Pachetul "features" probabil reprezintă diferite funcționalități sau module ale aplicației.

Fiecare funcționalitate poate cuprinde un set de funcționalități, componente și logica de gestionare a datelor asociate.

Organizarea codului în funcționalități ajută la menținerea unei arhitecturi modulare, în care fiecare funcționalitate este autonomă și poate fi dezvoltată, testată și implementată independent.

4.Pages :

Pachetul "pages" conține în mod obișnuit vizualizările sau ecranele de nivel superior ale aplicației, fiecare reprezentând o pagină sau rută diferită.

Paginile sunt compuse din componente UI și adesea corespund unor URL-uri sau rute specifice în aplicație.

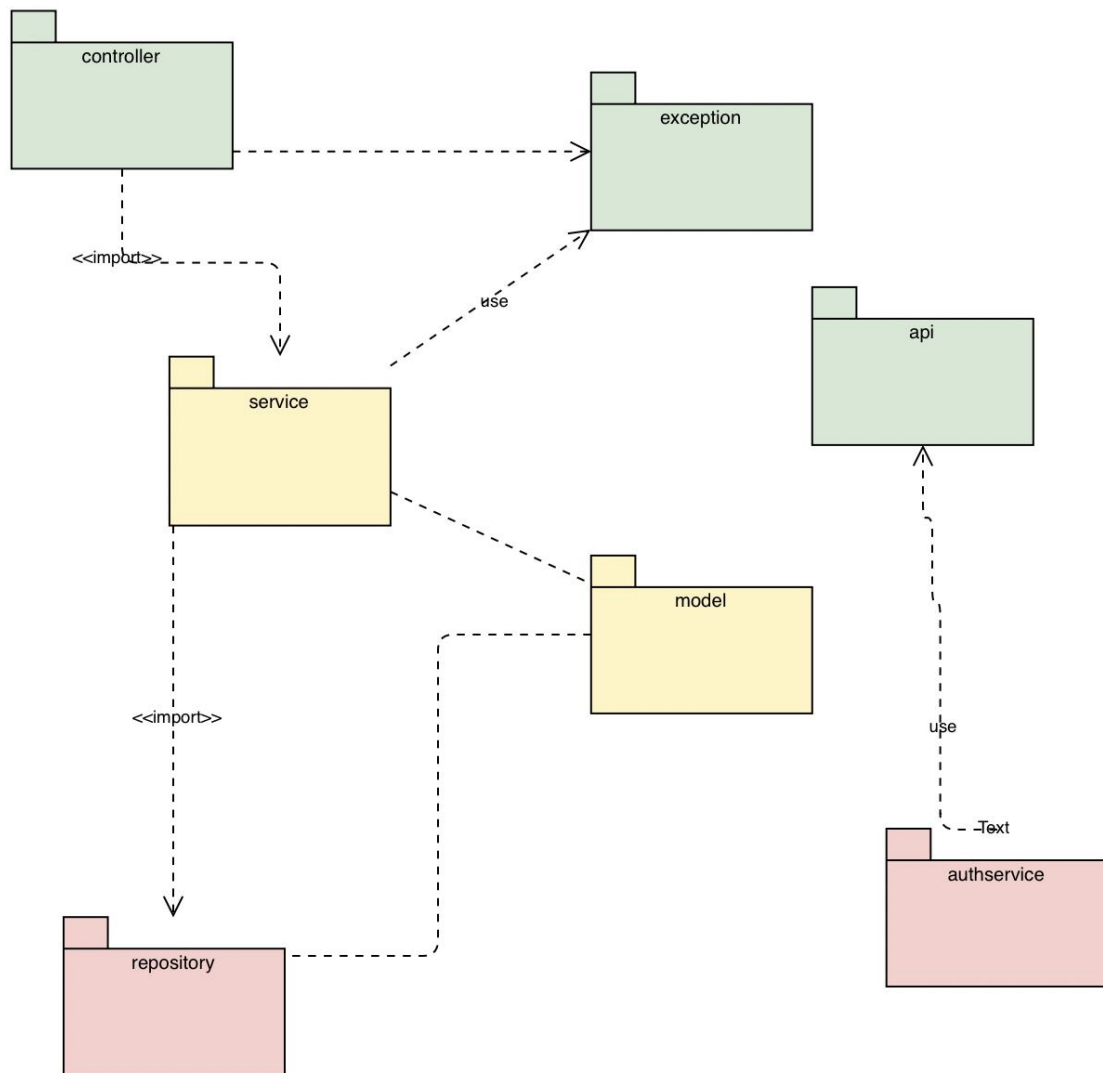
Fiecare pagină poate încapsula propria sa logică pentru gestionarea interacțiunilor cu utilizatorul și preluarea datelor specifică acelei pagini.

5.Styles :

Pachetul "styles" conține foi de stil sau fișiere legate de stiluri folosite pentru a defini prezentarea vizuală a aplicației.

Acest pachet poate include fișiere CSS, fișiere preprocesor (de exemplu, SCSS sau LESS), sau chiar stiluri inline definite folosind JavaScript.

Foile de stil definesc aspectul, aspectul și efectele vizuale ale componentelor UI și paginilor din întreaga aplicație.



1.API :

Pachetul "api" poate conține definiții ale endpoint-urilor API-ului care permit comunicarea între client și server.

Aici sunt definite rutele și logica asociată pentru a gestiona cererile primite de la client și pentru a trimite răspunsurile corespunzătoare.

2.ControllerAuth :

Acest pachet conține controlorii sau clasele responsabile de gestionarea cererilor legate de autentificare și autorizare.

Aici este implementată logica de autentificare, inclusiv verificarea și validarea identității utilizatorilor și a credențialelor acestora.

3.Model:

Pachetul "model" include clasele care definesc structura datelor și modelele de date utilizate în aplicație.

Aceste modele pot reprezenta obiecte de date, entități de bază sau structuri de date utilizate în întreaga aplicație.

4.Security :

Acest pachet se ocupă de aspectele legate de securitate în aplicație, cum ar fi autentificarea, autorizarea și gestionarea accesului.

Aici sunt implementate politici de securitate, filtrarea și validarea cererilor, precum și alte funcționalități legate de securitate.

5.AuthService :

Pachetul "authservice" conține logica de afaceri și serviciile asociate cu procesul de autentificare și autorizare.

Aici sunt implementate funcționalități precum gestionarea sesiunilor de autentificare, generarea și verificarea token-urilor de acces, și alte operațiuni de autentificare.

6.Controller:

Acest pachet poate include alte controlori sau clase care gestionează cererile primite de la client și coordonează logica de afaceri asociată.

Controlorii servesc ca intermediari între nivelul de prezentare și nivelul de acces la date, manipulând și direcționând cererile în funcție de logica aplicației.

7.Exception :

Aici sunt definite clasele de excepții sau mecanismele de gestionare a excepțiilor pentru a gestiona situațiile neașteptate sau erorile care apar în timpul execuției aplicației.

Excepțiile pot fi utilizate pentru a indica erori de validare, erori de acces la date sau alte condiții de eroare în aplicație.

8.Repository :

Acest pachet conține clasele responsabile de interacțiunea cu baza de date sau alte surse de date.

Repositoryile sunt utilizate pentru a abstractiza operațiile de acces și manipulare a datelor, oferind o interfață simplificată pentru a interacționa cu datele.

9.Service (Serviciu):

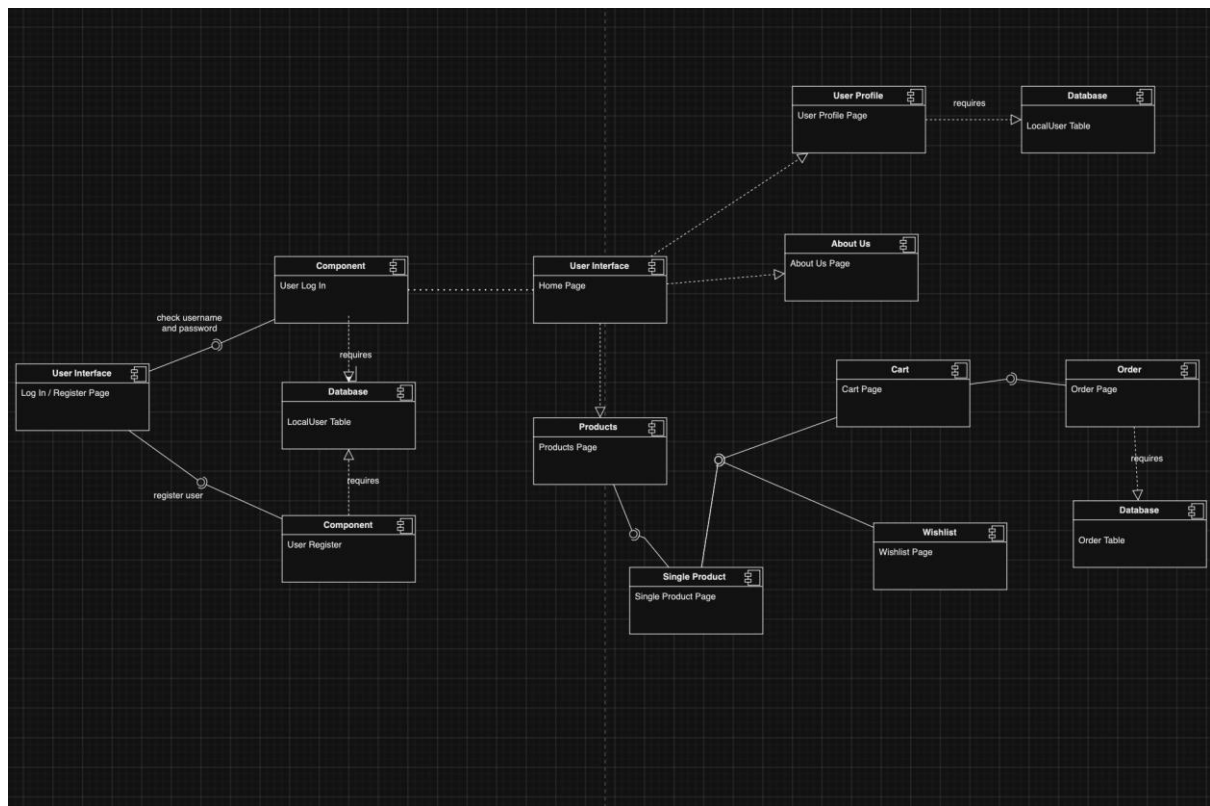
Pachetul "service" include serviciile și logica de afaceri asociate cu diferite funcționalități ale aplicației.

Aceste servicii pot include operații de afișare, modificare sau gestionare a datelor, precum și alte funcționalități specifice domeniului de aplicație.

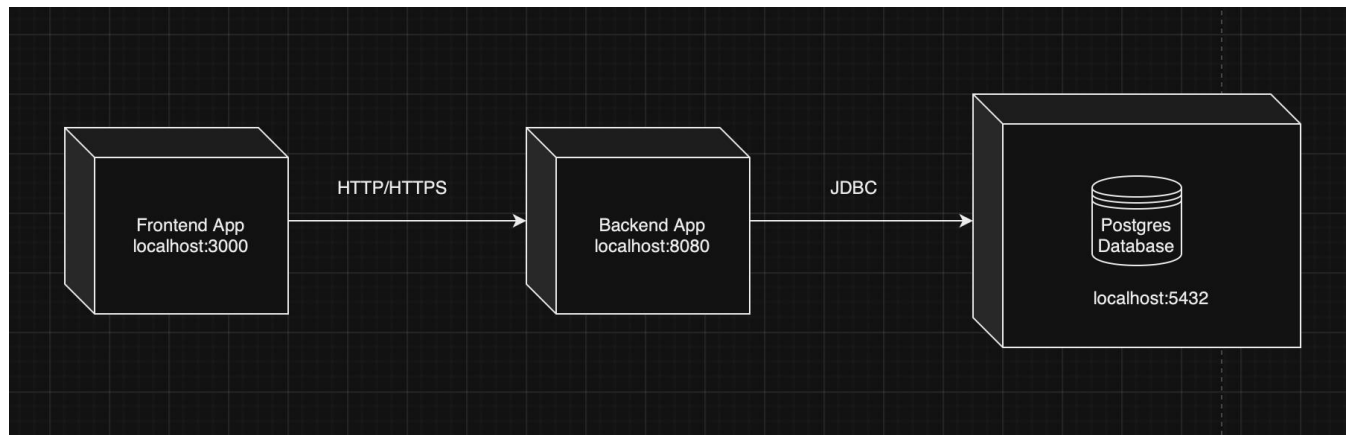
Aceste pachete și componente lucrează împreună pentru a construi o aplicație backend robustă și eficientă, care oferă funcționalitățile și serviciile necesare pentru a răspunde cerințelor utilizatorilor și a îndeplini obiectivele de afaceri.

6.3 Component and Deployment Diagram

Component Diagram

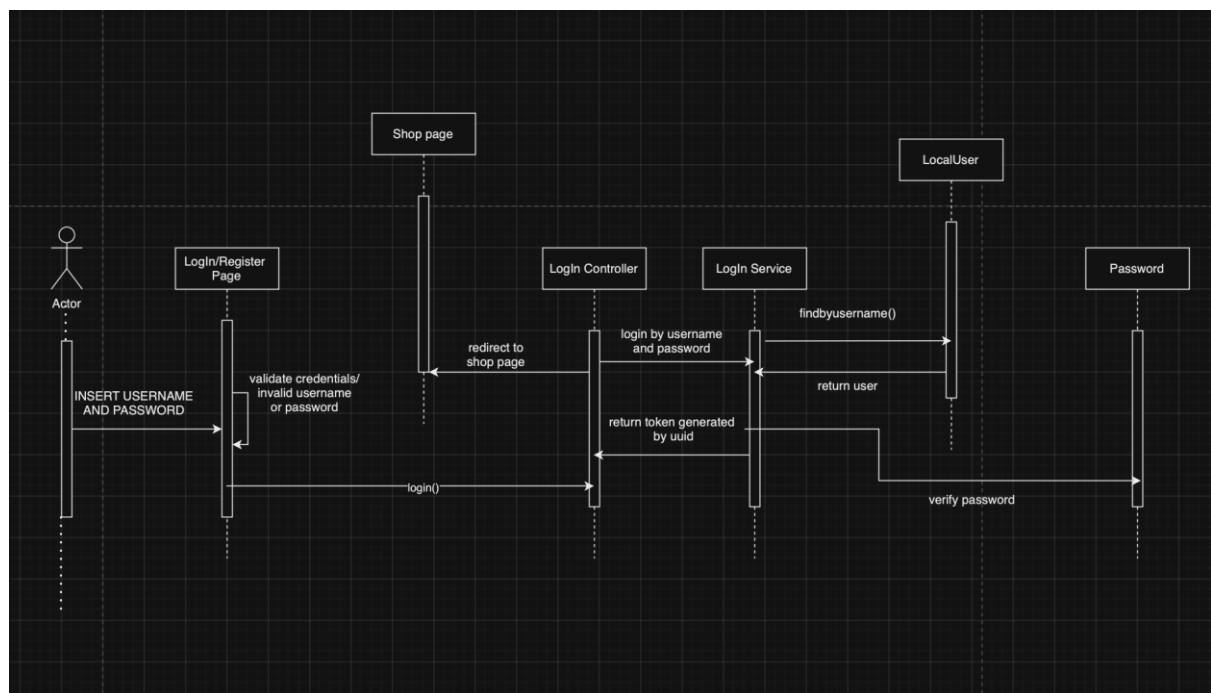


Deployment Diagram

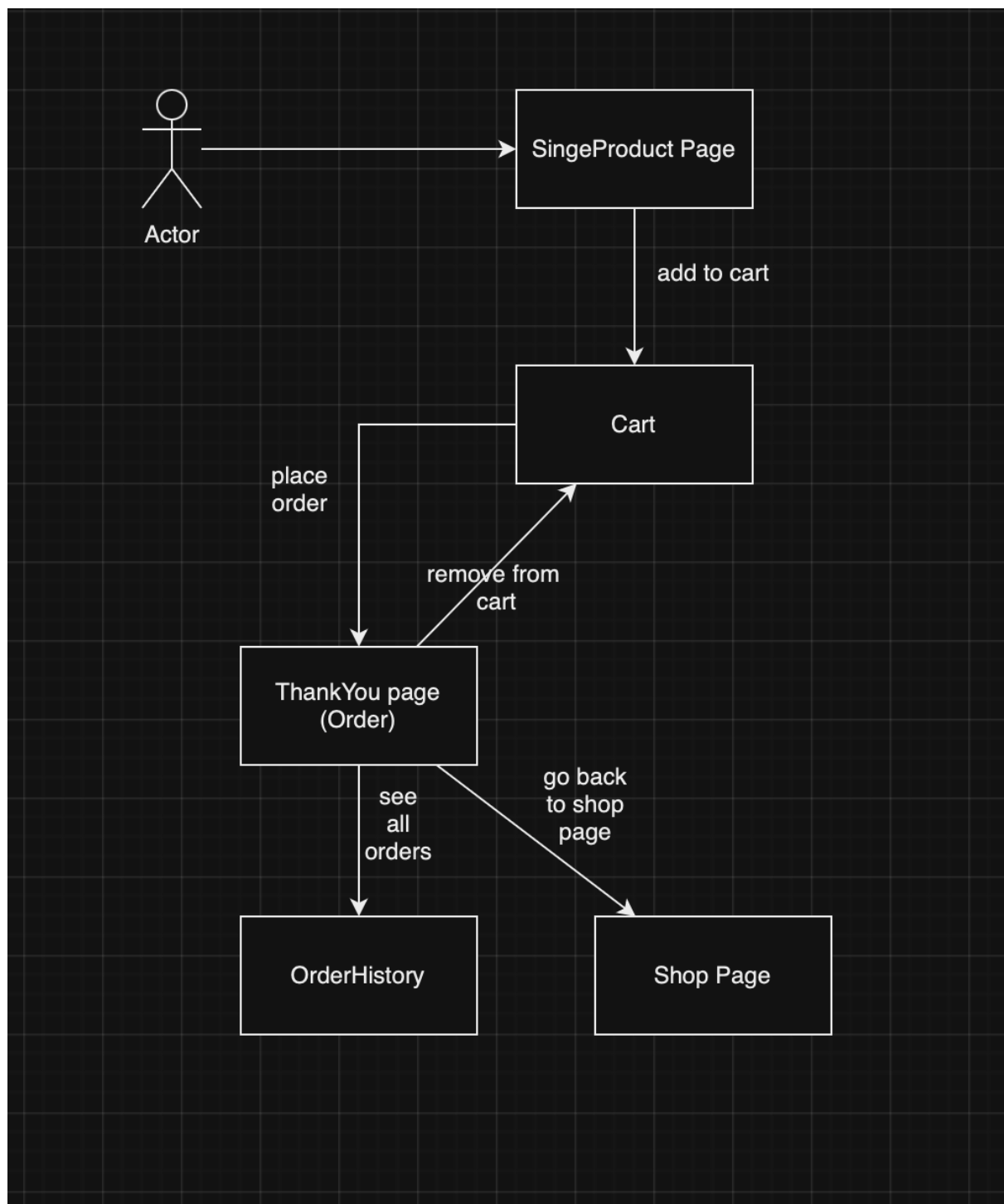


7. Dynamic Behaviour

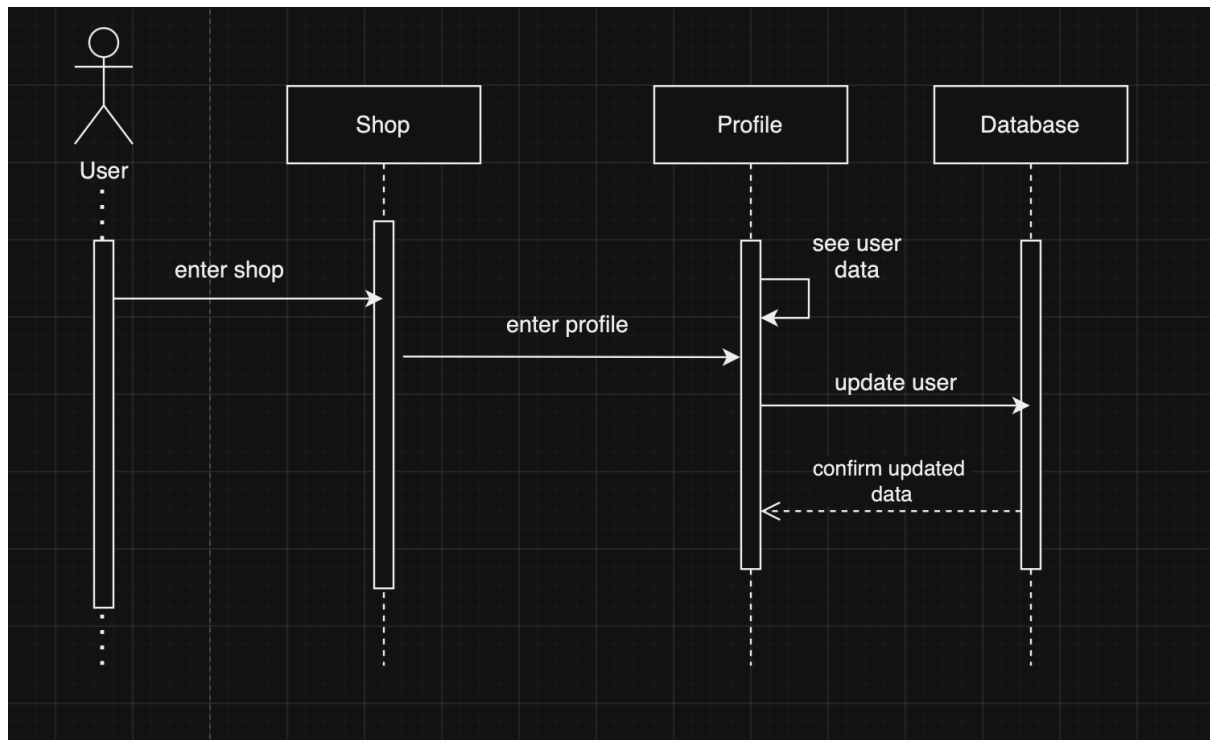
7.1 Diagrama de Secventa LOGIN:



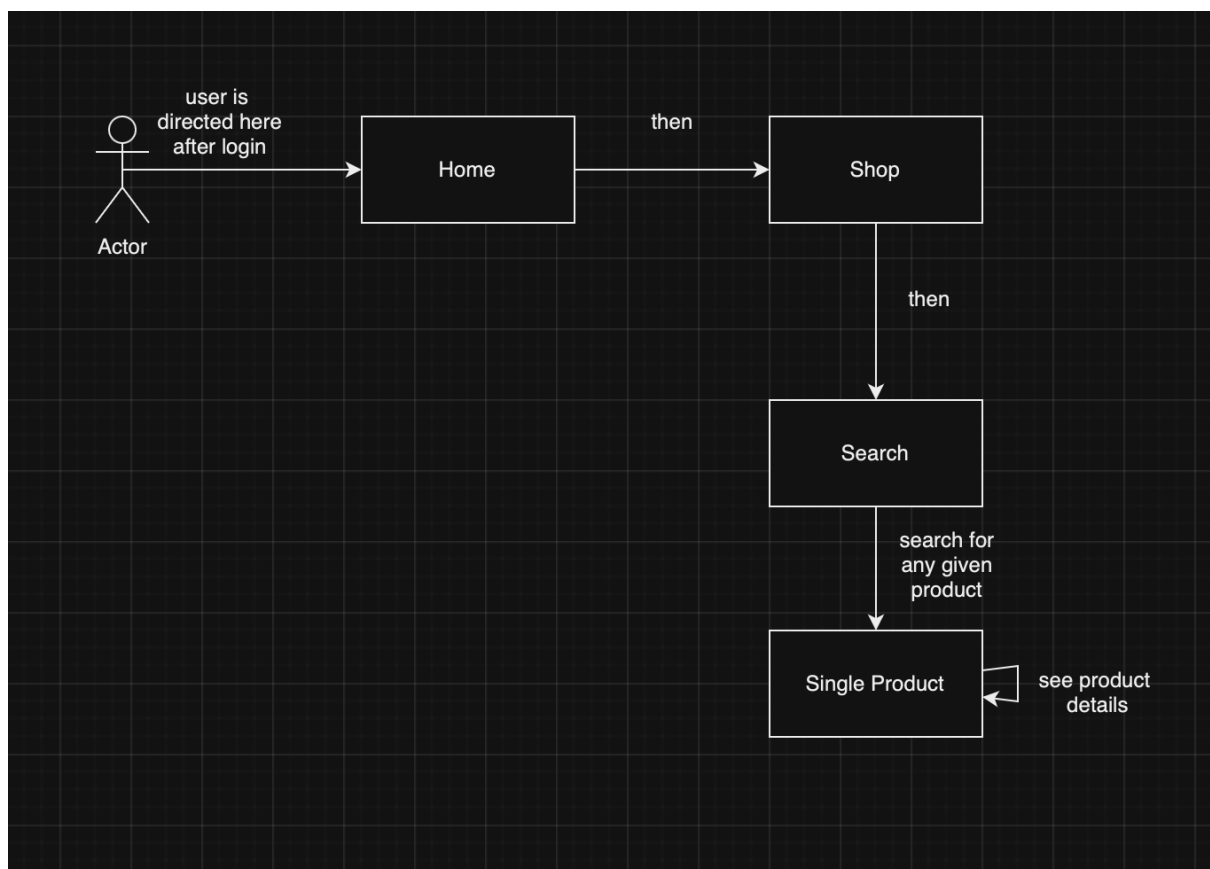
7.2 Diagrama de Comunicare ADD PRODUCT:



7.3 Diagram de secventa Update User:



7.4 Diagrama de comunicare Search a specific product:



8. Class Diagram

MVC (Model-View-Controller)

MVC (Model-View-Controller) este un model în designul software-ului, utilizat frecvent pentru a implementa interfețe utilizator, date și logică de control. Acesta pune accentul pe separarea dintre logica de afaceri a software-ului și afișaj. Această "separare a preocupărilor" asigură o diviziune mai bună a muncii și o întreținere îmbunătățită. Alte modele de design se bazează pe MVC, cum ar fi MVVM (Model-View-ViewModel), MVP (Model-View-Presenter) și MVW (Model-View-Whatever).

Cele trei părți ale modelului de design software MVC pot fi descrise astfel:

Model: Gestionează datele și logica de afaceri.

View: Se ocupă de aspect și afișare.

Controller: Direcționează comenzile către părțile model și view.

Exemplu de Model-View-Controller

Imaginați-vă o aplicație simplă de listă de cumpărături. Tot ce ne dorim este o listă cu numele, cantitatea și prețul fiecărui articol pe care trebuie să-l cumpărăm în această săptămână. Mai jos vom descrie cum am putea implementa o parte din această funcționalitate folosind MVC.

Modelul:

Modelul definește ce date ar trebui să conțină aplicația. Dacă starea acestor date se schimbă, modelul va notifica, de obicei, view-ul (pentru ca afișajul să se schimbe după cum este necesar) și uneori controller-ul (dacă este necesară o logică diferită pentru a controla view-ul actualizat).

Revenind la aplicația noastră de listă de cumpărături, modelul ar specifica ce date ar trebui să conțină articolele din listă — articol, preț, etc. — și ce articole sunt deja prezente în listă.

View-ul:

View-ul definește cum ar trebui să fie afișate datele aplicației.

În aplicația noastră de listă de cumpărături, view-ul ar defini cum este prezentată lista utilizatorului și ar primi datele pentru afișare de la model.

Controller-ul:

Controller-ul conține logica ce actualizează modelul și/sau view-ul în răspuns la intrările utilizatorilor aplicației.

De exemplu, lista noastră de cumpărături ar putea avea formulare de introducere și butoane care ne permit să adăugăm sau să ștergem articole. Aceste acțiuni necesită actualizarea modelului, astfel încât intrarea este trimisă la controller, care apoi manipulează modelul corespunzător, iar acesta trimite datele actualizate către view.

De asemenea, s-ar putea dori să actualizăm doar view-ul pentru a afișa datele într-un format diferit, de exemplu, schimbarea ordinii articolelor în ordine alfabetică sau de la cel mai mic la cel mai mare preț. În acest caz, controller-ul ar putea gestiona aceasta direct, fără a fi nevoie să actualizeze modelul.

JpaRepository Pattern

Descriere:

JpaRepository este un model de design utilizat în dezvoltarea aplicațiilor Java Spring pentru a facilita gestionarea operațiilor de bază de date. Acest pattern oferă o interfață simplificată pentru efectuarea operațiilor CRUD (Create, Read, Update, Delete) pe entități, eliminând astfel necesitatea de a scrie cod repetitiv pentru manipularea datelor.

Componente Principale:

JpaRepository Interface: Este o interfață furnizată de Spring Data JPA care oferă metode predefinite pentru operațiunile CRUD. Metodele includ salvarea, actualizarea, ștergerea și căutarea entităților în baza de date.

Entități: Reprezintă modelele de date care sunt mapate la tabelele din baza de date. Aceste entități sunt utilizate pentru a realiza operații de bază de date folosind JpaRepository.

Spring Data JPA: Este un modul din cadrul Spring Framework care facilitează dezvoltarea aplicațiilor bazate pe JPA (Java Persistence API). Oferă funcționalități suplimentare pentru a simplifica interacțiunea cu baza de date, inclusiv integrarea cu JpaRepository.

Avantaje:

Cod simplificat: Utilizarea JpaRepository reduce nevoia de a scrie cod repetitiv pentru operațiile de bază de date, ceea ce duce la un cod mai curat și mai ușor de înțeles.

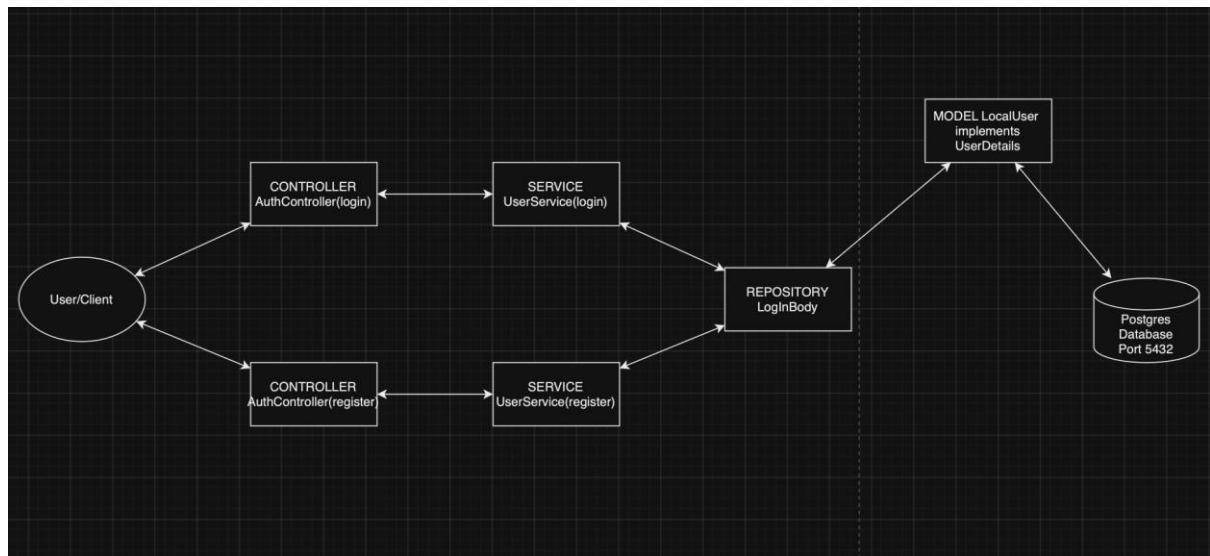
Productivitate crescută: JpaRepository oferă o interfață simplificată pentru efectuarea operațiilor de bază de date, permițând dezvoltatorilor să se concentreze pe logica de afaceri a aplicației.

Abstracție și modularitate: Oferă un nivel ridicat de abstracție, permițând dezvoltatorilor să lucreze la un nivel mai înalt și să lase gestionarea detaliilor de persistență a datelor la nivelul framework-ului.

Dezavantaje:

Limitări ale abstracției: JpaRepository oferă doar operații de bază de date și poate să nu fie adecvat pentru cerințele avansate de gestionare a datelor. Pentru cazuri mai complexe, ar putea fi necesară personalizarea sau extinderea JpaRepository.

Dependența de JPA: Utilizarea JpaRepository implică o dependență de JPA și Hibernate, ceea ce poate aduce complexitate și dependențe suplimentare în aplicație. De asemenea, performanța poate fi influențată de configurațiile și limitările JPA/Hibernate.



9. Testing

Unit Testing

Descriere:

Unit testing este un tip de testare software care se concentrează pe testarea unităților individuale de cod, cum ar fi funcțiile sau metodele, pentru a se asigura că acestea funcționează corect. Scopul principal al unit testing-ului este de a valida că fiecare unitate de cod se comportă conform așteptărilor.

Avantaje:

Depistarea timpurie a erorilor: Testele unitare pot detecta erori în fazele incipiente ale dezvoltării software-ului, ceea ce reduce costurile și timpul necesar pentru remedierea problemelor.

Îmbunătățirea calității codului: Prin scrierea testelor unitare, dezvoltatorii sunt adesea încurajați să scrie cod mai modular și mai curat.

Facilitarea refactorizării: Unit testing permite dezvoltatorilor să refactorizeze codul cu încredere, știind că testele vor semnaliza orice problemă cauzată de modificări.

Documentare: Testele unitare pot servi ca o formă de documentație vie, explicând cum ar trebui să funcționeze codul.

Exemple:

JUnit: Un framework popular pentru testarea unitară în Java.

NUnit: Un framework similar pentru .NET.

Adnotări

Descriere:

Adnotările (Annotations) sunt o caracteristică din multe limbaje de programare moderne, inclusiv Java și Python, care permit adăugarea de meta-informații la codul sursă. Aceste meta-informații pot fi utilizate de compilator, de runtime sau de alte instrumente și framework-uri pentru a modifica comportamentul programului sau pentru a adăuga funcționalități suplimentare.

Avantaje:

Claritate și citibilitate: Adnotările pot face codul mai clar și mai ușor de citit, oferind informații suplimentare despre comportamentul sau scopul anumitor elemente de cod.

Reducerea codului boilerplate: Adnotările pot reduce cantitatea de cod repetitiv necesar pentru anumite funcționalități, prin aplicarea unor comportamente comune într-un mod declarativ.

Extensibilitate: Adnotările permit adăugarea de comportamente personalizate și extensii fără a modifica codul de bază.

Exemple Comune în Java:

@Override: Indică faptul că o metodă suprascrie o metodă din clasa părinte.

@Deprecated: Marchează un element de cod ca fiind învechit și descurajează utilizarea sa.

@SuppressWarnings: Suprimă avertismentele specifice generate de compilator.

@Entity: Folosită în JPA pentru a indica că o clasă este o entitate de bază de date.

10. Specificatii suplimentare

1. Experiență de Utilizare Îmbunătățită

Recomandări Personalizate: Implementarea algoritmilor de învățare automată pentru a analiza comportamentul și preferințele utilizatorilor, oferind recomandări personalizate de produse.

Probă Virtuală: Integrarea realității augmentate (AR) pentru a permite clienților să încerce virtual articolele de îmbrăcăminte înainte de a le achiziționa.

Recomandări de Mărimi: Utilizarea AI pentru a sugera cea mai bună mărime pentru un client pe baza achizițiilor și feedback-urilor anterioare, reducând retururile din cauza problemelor de mărime.

2. Căutare și Navigare Avansată

Căutare Vocală: Adăugarea funcționalității de căutare vocală pentru a facilita găsirea rapidă a produselor.

Căutare Vizuală: Permite utilizatorilor să încarce imagini și să găsească produse similare disponibile în magazin.

Căutare Facetată: Îmbunătățirea capacităților de căutare prin permiterea clienților să filtreze produsele pe baza mai multor atribute, cum ar fi mărimea, culoarea, gama de prețuri și brandul.

3. Pagini de Produs Îmbunătățite

Vizualizări 360 de Grade: Oferirea de vizualizări 360 de grade ale produselor pentru a oferi clienților o idee mai clară despre aspectul și senzația articolelor.

Conținut Generat de Utilizatori: Încurajarea clienților să încarce fotografii și recenzii ale produselor pe care le-au achiziționat, oferind feedback autentic potențialilor cumpărători.

Informații Detaliate despre Produse: Includerea de detalii cuprinzătoare despre produse, cum ar fi compoziția materialului, instrucțiuni de îngrijire și sfaturi de stil.

4. Proces de Checkout Îmbunătățit

Checkout cu un Singur Click: Simplificarea procesului de checkout cu opțiuni de cumpărare cu un singur click pentru clienții care revin.

Mai Multe Opțiuni de Plată: Oferirea unei varietăți de metode de plată, inclusiv carduri de credit/debit, PayPal, Apple Pay, Google Wallet și servicii de tip "Cumpără acum, plătește mai târziu".

Checkout ca Vizitator: Permite clienților să facă achiziții fără a crea un cont pentru a simplifica procesul de cumpărare pentru cumpărătorii noi.

5. Programe de Fidelizare a Clienților

Puncte și Recompense: Introducerea unui program de fidelitate în care clienții câștigă puncte pentru achiziții, recenzii și recomandări, puncte ce pot fi răscumpărate pentru reduceri sau produse gratuite.

Oferte Exclusive: Oferirea membrilor acces anticipat la noi colecții, reduceri exclusive și evenimente speciale.

Recompense de Ziua de Naștere: Trimiterea de oferte personalizate sau cadouri clienților de ziua lor de naștere pentru a spori loialitatea acestora.

6. Experiență Mobilă Îmbunătățită

Aplicație Mobilă: Dezvoltarea unei aplicații mobile dedicate pentru iOS și Android pentru a oferi o experiență de cumpărături fără cusur în mișcare.

Notificări Push: Utilizarea notificărilor push pentru a alerta clienții despre vânzări, noutăți și actualizări de stare ale comenzilor.

Website Optimizat pentru Mobil: Asigurarea faptului că site-ul este complet optimizat pentru dispozitive mobile, cu design responsiv și timpi de încărcare rapizi.

7. Îmbunătățiri Operaționale

Managementul Inventarului: Implementarea unor sisteme avansate de management al inventarului pentru a urmări nivelurile stocurilor în timp real și pentru a evita situațiile de stoc epuizat sau supra-stoc.

Îndeplinirea Comenzilor: Optimizarea procesului de îndeplinire a comenzilor cu automatizare și integrarea cu furnizori terți de logistică pentru livrări mai rapide și mai precise.

Inițiative de Sustenabilitate: Adoptarea practicilor sustenabile în aprovizionare, ambalare și expediere pentru a atrage consumatorii conștienți de mediu.

8. Analiza și Informații de Date

Analiza Clienților: Utilizarea analizei datelor pentru a obține informații despre comportamentul, preferințele și tendințele clienților pentru a lua decizii de afaceri informate.

Analiza Vânzărilor: Monitorizarea performanței vânzărilor pe diferite canale și produse pentru a identifica cele mai bine vândute și cele cu performanțe scăzute.

Tendențe de Piață: Menținerea la curent cu tendințele din industrie prin analiza datelor de piață și feedback-ul clienților pentru a ajusta ofertele de produse și strategiile de marketing în consecință.

9. Marketing și Promovare

Email Marketing: Crearea de campanii de email personalizate pe baza preferințelor clienților, istoricului achizițiilor și comportamentului de navigare.

Integrarea Social Media: Utilizarea platformelor de social media pentru a promova produsele, a interacționa cu clienții și a atrage trafic către site.

Parteneriate cu Influenceri: Colaborarea cu influenceri și bloggeri de modă pentru a ajunge la un public mai larg și pentru a construi credibilitatea brandului.

Prin implementarea acestor funcționalități și îmbunătățiri, magazinul online de haine poate îmbunătăți semnificativ experiența de cumpărături, poate crește satisfacția clienților și poate stimula creșterea vânzărilor.

11. Concluzie

În concluzie, prin utilizarea unei suite variate de tehnologii și funcționalități, platforma Pravalia lui Paul a reușit să ofere o experiență completă și sigură pentru utilizatori. Frontend-ul, construit folosind React, a furnizat o interfață de utilizator modernă și interactivă, care a facilitat navigarea și interacțiunea cu platforma.

Baza de date PostgreSQL a servit drept stocare pentru informațiile esențiale ale utilizatorilor, produselor și comenzilor, asigurând o gestionare eficientă și scalabilă a datelor.

Funcționalitățile implementate au adus valoare adăugată platformei Pravalia lui Paul. Login-ul și înregistrarea au permis utilizatorilor să-și creeze și să-și gestioneze conturile în mod securizat. Securitatea pe endpoint-uri a asigurat că datele sensibile sunt protejate împotriva accesului neautorizat.

Adăugarea de produse și posibilitatea de a lăsa recenzii au îmbunătățit experiența de cumpărare a utilizatorilor, oferindu-le o gamă variată de produse și feedback util pentru deciziile de achiziție. Chat-ul cu WebSockets a adăugat o dimensiune de interacțiune în timp real între utilizatori și a facilitat comunicarea rapidă între clienți și administratori.

Sistemul de comenzi, împreună cu generarea automată a facturilor în format PDF și trimiterea lor prin email, au simplificat procesul de cumpărare și au oferit o experiență completă și profesională pentru utilizatori.

În ansamblu, utilizarea acestor tehnologii și funcționalități a contribuit la crearea unei platforme Ecommerce robuste, sigure și ușor de utilizat, care satisface nevoile și așteptările clienților și contribuie la succesul și creșterea continuă a afacerii Pravalia lui Paul.

12. Bibliografie

Bibliografie

1.React Documentation. Disponibil online: <https://reactjs.org/docs/getting-started.html>

2.Spring Framework Documentation. Disponibil online: <https://spring.io/projects/spring-framework>

3."JUnit 5 User Guide." Disponibil online:

<https://junit.org/junit5/docs/current/user-guide/>

4."WebSocket API." Mozilla Developer Network. Disponibil online:

<https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>

5."Understanding MVC Architecture with Example." GeeksforGeeks. Disponibil online: <https://www.geeksforgeeks.org/mvc-design-pattern/>

6."PostgreSQL Documentation." Disponibil online:

<https://www.postgresql.org/docs/>