



**UNIVERSITATEA
TEHNICĂ
DIN CLUJ-NAPOCA**

Altex E-commerce

Programare Software

Derivarable Final

Balint Cătălin-Vasile

30236/1

FACULTATEA DE AUTOMATICA
SI CALCULATOARE

Cuprins

1	Specificațiile proiectului	3
1.1	Scopul proiectului	3
1.2	Obiectivele proiectului	3
1.3	Cerințele proiectului	3
1.4	Arhitectura proiectului	4
1.5	Ierarhia structurală	4
1.6	Dependințele proiectului	6
2	Cerintele funcționale	9
2.1	Sign up	9
2.2	Sign in	11
2.3	Acces User/Admin	12
2.4	Email	13
2.5	Review	14
2.6	Wishlist	15
2.7	Cart	17
2.8	Order	18
2.9	Chat	19
2.10	XML	20
2.11	Payment	22
3	Domain and Data Model	23
4	Arhitectural Design	25
4.1	Conceptual Architecture	25
4.2	Package Design	26
5	Component and Deployment Diagram	28
5.1	Component Diagram	28
5.2	Deployment Diagram	29
6	Dynamic Behavior	30
6.1	Diagrama de Secvență LOGIN:	30
6.2	Diagrama de Comunicare LOGIN:	31
6.3	Diagrama de Secvență ADD PRODUCT:	31
6.4	Diagrama de Comunicare ORDER:	32
7	Use case UML	33
7.1	Use case USER	34
7.2	Use case ADMIN	35
8	Class Diagram	37
9	Testing	39
10	Specificații suplimentare	40
10.1	Cerințe non-funcționale	40
10.1.1	Performanță	40
10.1.2	Scalabilitate	40

10.1.3	Fiabilitatea	41
10.1.4	Uzabilitate	41
10.2	Design constraints	41
11	Future Improvements	42
12	Conclusion	42
13	Bibliography	43

1 Specificațiile proiectului

1.1 Scopul proiectului

Proiectul propus își propune să creeze o imitație sau o clonă a retailerului de succes AL-TEX, aducând o experiență de cumpărături online similară și oferind o gamă largă de produse electronice și electrocasnice. Scopul său 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

Obiectivele proiectului sunt strâns legate de utilizarea tehnologiilor full stack, în special Spring și React, pentru a atinge nivelul optim de funcționalitate și experiență pentru utilizatori. Prin implementarea acestor obiective, ne propunem să valorificăm beneficiile și capacitățile acestor tehnologii pentru a crea o platformă de comerț electronic robustă și eficientă.

Reproducerea Experienței ALTEX Online: Utilizarea tehnologiilor full stack, cum ar fi Spring pentru backend și React pentru frontend, ne permite să reproducem experiența ALTEX în mediul online într-un mod eficient și autentic. Cu ajutorul Spring, putem gestiona logică de afaceri complexă și interacțiunea cu baza de date, în timp ce React ne permite să construim interfețe de utilizator moderne și interactive care să ofere o experiență similară cu cea a magazinelor fizice.

Conveniență și Accesibilitate: Utilizând Spring Boot pentru backend și React.js pentru frontend, putem crea o experiență de cumpărături online convenabilă și accesibilă pentru utilizatori. Spring Boot ne permite să dezvoltăm rapid și să configurăm ușor aplicația noastră, în timp ce React.js ne oferă o interfață de utilizator reactivă și plină de funcționalități pentru a facilita navigarea și cumpărarea produselor.

Servicii de Înaltă Calitate: Cu ajutorul Spring Security și a altor module din cadrul Spring, putem asigura securitatea și confidențialitatea datelor utilizatorilor noștri, precum și să oferim o experiență de utilizator sigură și protejată împotriva fraudelor și a altor amenințări online. În plus, putem utiliza funcționalitățile avansate de gestionare a sesiunilor și a autentificării din Spring pentru a oferi o experiență de utilizare fără probleme și plăcută.

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, cunoscut pentru robustețea sa și capacitățile sale de gestionare a datelor și logică de afaceri. Acesta va asigura funcționarea fluentă a operațiilor CRUD pe entități, precum și gestionarea relațiilor complexe între entități, prin intermediul unui ORM (Object-Relational Mapping) eficient.

Frontend-ul va fi dezvoltat cu React.js, o bibliotecă JavaScript modernă și puternică, care va oferi o interfață de utilizator elegantă și interactivă. Utilizatorii vor beneficia de o experiență de cumpărături online plăcută și intuitivă, cu interfețe prietenoase și funcționalități avansate pentru navigare, căutare și comparare a produselor.

Platforma va integra accesul diferit pentru utilizatorii obișnuiți și administratori, oferind funcționalități specifice pentru fiecare categorie de utilizatori. Utilizatorii obișnuiți vor avea posibilitatea de a naviga și a cumpăra produse, precum și de a-și gestiona conturile personale și

istoricul comenzilor, în timp ce administratorii vor avea acces la funcționalități extinse pentru gestionarea produselor, utilizatorilor și comenzilor.

Cu o abordare profesională și integrată a dezvoltării, platforma va oferi o soluție robustă și scalabilă, menită să satisfacă cerințele și așteptările clienților săi. Integrarea tehnologiilor Spring Java și React.js va asigura performanțe excelente și o experiență de utilizare de neegalat, consolidându-și poziția ca un lider în domeniul comerțului electronic.

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 pentru a oferi o experiență de utilizare fluidă și plăcută. Pe partea de backend, am adoptat o arhitectură bazată pe model-view-controller (MVC), folosind cadrul de lucru Spring Java. Am organizat codul în pachete distincte, cum ar fi modele pentru clasele Java reprezentând entitățile, repository pentru definirea interfețelor care extind JpaRepository pentru manipularea datelor în baza de date, și controlere pentru gestionarea endpoint-urilor care facilitează comunicarea dintre frontend și backend.

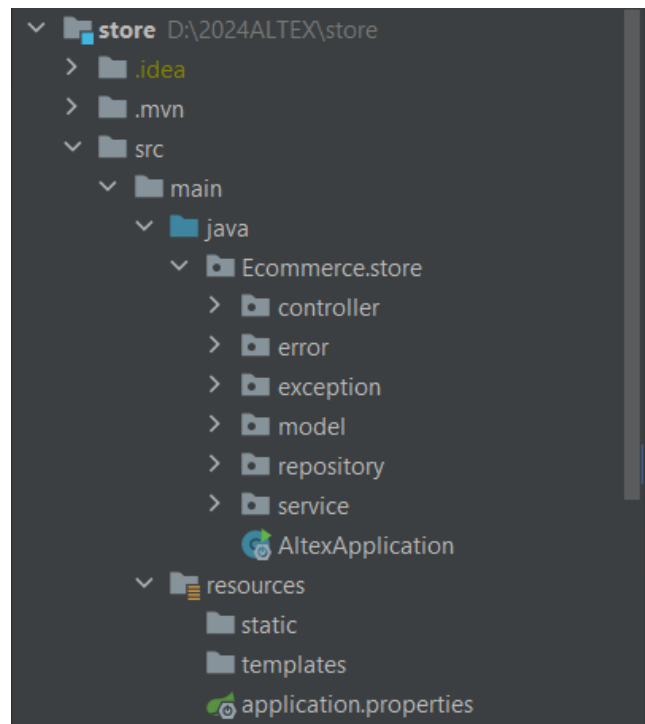
Pentru comunicarea cu baza de date, am optat pentru MySQL, un sistem de gestionare a bazelor de date relaționale puternic și fiabil, care utilizează portul JDBC 3306 pentru conexiunea la server. Această alegere se datorează fiabilității și performanței pe care MySQL le oferă în manipularea datelor.

Pe partea de frontend, am ales să utilizăm React.js, o bibliotecă JavaScript populară și eficientă, care oferă o abordare component-based pentru dezvoltarea interfețelor de utilizator. Această decizie a fost luată datorită flexibilității, performanței și scalabilității oferite de React.js în dezvoltarea de aplicații web moderne.

Pentru gestionarea datelor în backend, am folosit Spring Data JPA, împreună cu dependențele Lombok, Spring Web și MySQL Driver, pentru a asigura o integrare simplă și eficientă între aplicație și baza de date. Aceste tehnologii au fost alese datorită capacității lor de a accelera dezvoltarea și de a reduce cantitatea de cod necesară pentru a implementa funcționalități complexe în backend.

1.5 Ierarhia structurală

În cadrul aplicației noastre, am organizat clasele și interfețele în diferite pachete pentru a menține o ierarhie coerentă și ușor de gestionat. Această ierarhie de pachete și clase este esențială pentru structurarea logică a aplicației noastre și pentru gestionarea diferitelor aspecte ale funcționalității sale. În continuare, vom explora ierarhia de pachete și clase din aplicație.



Pachetul de bază al aplicației noastre este ‘Ecommerce.store’, care conține întreaga logică a magazinului nostru online. În acest pachet, avem mai multe subpachete care împart funcționalitatea aplicației în module distincte pentru o gestionare mai ușoară și o organizare mai clară.

Unul dintre aceste subpachete este ‘controller’, care conține clasele care gestionează cererile HTTP și răspunsurile asociate. Aici găsim clasele ‘AdminController’, ‘PersonController’ și ‘ProductCategoryController’, care sunt responsabile pentru gestionarea operațiunilor asociate administratorilor, utilizatorilor și produselor din magazinul nostru.

În pachetul ‘model’, definim clasele care reprezintă obiectele de bază ale aplicației noastre. Acestea includ clasele ‘Admin’, ‘Category’, ‘Person’, ‘Product’ și ‘User’, care descriu entitățile noastre principale și relațiile între ele.

Pachetul ‘repository’ conține interfețele și clasele care se ocupă de interacțiunea cu baza de date. Aici avem clasele ‘AdminRepository’, ‘CategoryRepository’, ‘PersonRepository’, ‘ProductRepository’ și ‘UserRepository’, care sunt responsabile pentru realizarea operațiilor de bază cu entitățile respective.

În ceea ce privește gestionarea erorilor și excepțiilor, acestea sunt tratate în pachetul ‘error’. Aici găsim clasa ‘UserNotFoundAdvice’, care furnizează sfaturi personalizate pentru gestionarea excepțiilor legate de utilizatori care nu pot fi găsiți în sistem.

În plus, avem clasa ‘AltexApplication’, care servește drept punct de intrare principal pentru aplicația noastră Spring Boot. Aceasta inițiază contextul aplicației și configurează setările inițiale necesare pentru funcționarea corectă a acesteia.

Ierarhia de pachete și clase din aplicația noastră oferă o structură bine definită și organizată, care facilitează dezvoltarea, testarea și întreținerea aplicației noastre de comerț electronic. Prin împărțirea funcționalității în module separate și clar definite, asigurăm un cod curat și ușor de gestionat, care să satisfacă cerințele și standardele noastre de calitate.

1.6 Dependințele proiectului

Spring Boot Starter Data JPA: Această dependență reprezintă un starter pentru dezvoltarea aplicațiilor Java care utilizează Spring Data JPA pentru interacțiunea cu baza de date. Spring Data JPA simplifică gestionarea operațiilor CRUD (Create, Read, Update, Delete) și a relațiilor între obiecte și tabele în baza de date, permițând dezvoltatorilor să se concentreze mai mult pe logica de afaceri a aplicației.

Spring Boot Starter Websocket: Această dependență furnizează suport pentru implementarea comunicării în timp real în aplicațiile web Java folosind tehnologia WebSocket. Prin utilizarea acestui starter, dezvoltatorii pot crea aplicații care permit schimbul bidirecțional de mesaje între client și server într-un mod eficient și scalabil.

Spring Boot Starter Mail: Această dependență oferă suport pentru trimiterea de emailuri din aplicațiile Java folosind Spring Boot. Spring Boot Starter Mail facilitează configurarea și trimiterea de emailuri, inclusiv trimiterea de emailuri text sau HTML și atașarea de fișiere.

Spring Boot Starter Validation: Această dependență furnizează suport pentru validarea datelor în aplicațiile Spring Boot, permițând dezvoltatorilor să definească reguli de validare pentru obiectele lor de domeniu. Prin utilizarea acestui starter, dezvoltatorii pot asigura integritatea și coerența datelor în cadrul aplicației lor.

iText7 Core: Această dependență oferă funcționalități pentru generarea și manipularea de documente PDF în aplicațiile Java. Prin intermediul iText7 Core, dezvoltatorii pot crea și personaliza documente PDF în mod dinamic, facilitând generarea de rapoarte și documente în aplicațiile lor.

Spring Boot Starter Security: Această dependență furnizează suport pentru implementarea funcționalităților de securitate în aplicațiile Spring Boot, inclusiv autentificare, autorizare și gestionarea sesiunilor. Prin utilizarea acestui starter, dezvoltatorii pot proteja aplicația lor și datele utilizatorilor într-un mod eficient și scalabil.

Jackson Dataformat XML: Această dependență oferă suport pentru serializarea și deserializarea datelor în format XML în aplicațiile Java folosind biblioteca Jackson. Prin intermediul acestei dependențe, dezvoltatorii pot converti obiectele lor Java în format XML și viceversa, facilitând interoperabilitatea cu alte sisteme și servicii care utilizează acest format.

Jackson Datatype JSR310: Această dependență furnizează suport pentru serializarea și deserializarea datelor temporale (precum `LocalDate` și `LocalDateTime`) în aplicațiile Java folosind biblioteca Jackson. Prin utilizarea acestei dependențe, dezvoltatorii pot gestiona în mod eficient datele temporale în aplicațiile lor, asigurând compatibilitatea și coerența datelor.

Spring Boot Starter Web: Această dependență furnizează suport pentru dezvoltarea aplicațiilor web în Spring Boot, inclusiv gestionarea cererilor HTTP și a răspunsurilor, crearea de controlere web și configurarea de servicii RESTful. Prin utilizarea acestui starter, dezvoltatorii pot crea rapid și eficient aplicații web scalabile și robuste.

MySQL Connector/J: Această dependență oferă suport pentru conectarea aplicației Java la baza de date MySQL, permițând dezvoltatorilor să efectueze operațiuni de citire și scriere în baza de date. Prin intermediul MySQL Connector/J, dezvoltatorii pot interacționa cu baza de date MySQL într-un mod eficient și sigur, asigurând integritatea și securitatea datelor.

Lombok: Această dependență simplifică dezvoltarea aplicațiilor Java prin generarea automată a metodelor getter, setter și constructori, reducând astfel cantitatea de cod boilerplate. Prin intermediul Lombok, dezvoltatorii pot scrie cod mai curat și mai concis, concentrându-se mai mult pe logica de afaceri a aplicației lor.

Spring Boot Starter Test: Această dependență furnizează suport pentru testarea unitară și integrată a aplicațiilor Spring Boot, permițând dezvoltatorilor să testeze și să valideze funcționalitățile aplicației lor într-un mod automatizat și reproducibil.

Spring Security Test: Această dependență oferă un set de instrumente pentru testarea securității în aplicațiile Spring, permițând dezvoltatorilor să testeze funcționalitățile de autentificare și autorizare într-un mod eficient și sigur. Prin intermediul acestor instrumente, dezvoltatorii pot asigura că aplicația lor respectă cele mai înalte standarde de securitate și protecție a datelor.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.17</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>Store</groupId>
    <artifactId>Altex</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Altex</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-websocket</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-mail</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
        <dependency>
            <groupId>com.itextpdf</groupId>
            <artifactId>itext7-core</artifactId>
            <version>7.1.9</version>
            <type>pom</type>
        </dependency>
```



```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
  <version>2.13.0</version>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.datatype</groupId>
  <artifactId>jackson-datatype-jsr310</artifactId>
  <version>2.13.5</version>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <scope>runtime</scope>

</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>

```

```

        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
            <excludes>
                <exclude>
                    <groupId>org.projectlombok</groupId>
                    <artifactId>lombok</artifactId>
                </exclude>
            </excludes>
        </configuration>
    </plugin>
</plugins>
</build>

</project>

```

2 Cerintele funcționale

2.1 Sign up

Sistemul trebuie să permită utilizatorilor să se înregistreze furnizându-și adresa de e-mail și o parolă. La înregistrare, utilizatorii trebuie să poată alege un rol (USER sau ADMIN). Cu toate acestea, pentru a menține simplificată experiența de înregistrare, sistemul poate să permită doar înregistrarea ca USER, dar să aibă deja un ADMIN existent în baza de date.

Pentru funcționalitatea de Sign Up, sistemul trebuie să ofere utilizatorilor posibilitatea de a-și crea conturi furnizând adresa lor de e-mail și o parolă. În momentul înregistrării, utilizatorii ar trebui să poată selecta un rol, fie ca USER sau ADMIN. Totuși, pentru a menține o experiență de înregistrare simplificată, s-ar putea permite doar înregistrarea ca USER, având deja un ADMIN existent în baza de date. În acest mod, se asigură că doar utilizatorii obișnuiți pot înregistra conturi noi, în timp ce administratorii sunt predefiniți în sistem.

Pentru a garanta unicitatea adresei de e-mail furnizate, sistemul trebuie să efectueze o verificare în baza de date și să furnizeze un mesaj de eroare în cazul în care adresa de e-mail este deja înregistrată. În plus, pentru a asigura corectitudinea introducerii parolei, utilizatorului i se va cere să reintroducă parola într-un câmp de confirmare a parolei. Doar parola va fi stocată în baza de date, întrucât nu este necesar să se stocheze parțial sau complet datele sensibile ale utilizatorilor.

Implementarea acestor cerințe ar trebui să ofere o experiență de autentificare și înregistrare eficientă și securizată pentru utilizatorii sistemului, asigurând în același timp că informațiile necesare sunt colectate și stocate în mod adecvat în baza de date.

Metoda POST este utilizată pentru a permite utilizatorilor să își înregistreze conturi noi. Atunci când un utilizator trimite o cerere HTTP POST către endpoint-ul /person, această cerere este gestionată de clasa PersonController, care este responsabilă pentru tratarea cererilor legate de persoane.

Metoda newPerson din PersonController primește obiectul JSON care conține detaliile persoanei nou înregistrate. În primul rând, această metodă apelează serviciul PersonService pentru a seta rolul persoanei ca USER, folosind metoda savePersonRole. Aceasta este o etapă esențială pentru a asigura că toți utilizatorii noi sunt înregistrați cu rolul corespunzător.

```

11 @Service
12 public class PersonService {
13
14     @Autowired
15     private PersonRepository personRepository;
16
17
29     @Transactional
30     public void sentToEntity(Person person) {
31         if (person.getPersonRole().equals(PersonRole.ADMIN)){
32             Admin admin = new Admin();
33             admin.setPerson(person);
34             adminRepository.save(admin);
35             person.setAdmin(admin);
36             personRepository.save(person);
37
38         } else if (person.getPersonRole().equals(PersonRole.USER)){
39             User user = new User();
40             user.setPerson(person);
41             userRepository.save(user);
42             person.setUser(user);
43             personRepository.save(person);
44         }
45     }

```

După ce rolul persoanei este setat, obiectul Person este salvat în baza de date utilizând obiectul PersonRepository. În același timp, metoda sentToEntity din PersonService este apelată pentru a gestiona crearea entităților asociate și stabilirea relațiilor între acestea și persoană.

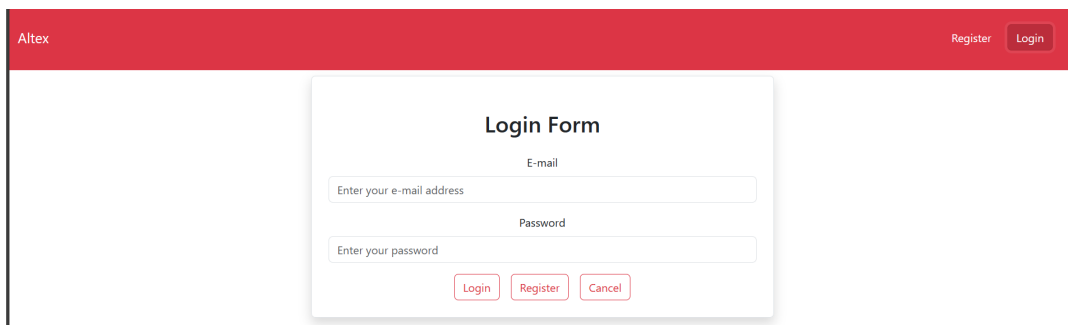
În cadrul metodei `sentToEntity`, se verifică rolul persoanei și se creează și se salvează obiectele `User` sau `Admin` corespunzătoare în funcție de acesta. Relațiile între persoană și entitățile asociate sunt stabilite, iar obiectul `Person` actualizat, care include informațiile generate de baza de date, este salvat din nou.

Astfel, prin intermediul acestei metode `POST`, aplicația noastră permite înregistrarea eficientă și securizată a utilizatorilor noi, asigurând că toate informațiile necesare sunt gestionate și stocate corespunzător în baza de date.

```
25     @PostMapping("/person")
26     Person newPerson(@RequestBody Person newPerson) {
27         personService.savePersonRole(newPerson);
28         personRepository.save(newPerson);
29         personService.sentToEntity(newPerson);
30         return personRepository.save(newPerson);
31     }
```

2.2 Sign in

Metoda `loginUser` este o parte critică a sistemului nostru de autentificare și este responsabilă pentru gestionarea cererilor de autentificare trimise de către utilizatori. Această metodă este implementată ca un endpoint de tip `GET` și este accesibilă prin URL-ul `/login`. În urma unei cereri `HTTP GET` către acest endpoint, metoda primește două parametri: adresa de e-mail și parola utilizatorului. Acești parametri sunt trimiși ca parametri de interogare în cadrul cererii `HTTP`.



Principalele adnotări folosite pentru a configura această metodă sunt `@GetMapping` și `@RequestParam`. Adnotarea `@GetMapping` specifică că această metodă va fi apelată atunci când este trimisă o cerere `HTTP GET` către endpoint-ul `/login`. Pe de altă parte, adnotarea `@RequestParam` este folosită pentru a extrage parametrii de interogare din URL-ul cererii `HTTP` și pentru a le asocia variabilelor locale din metoda `loginUser`.

În interiorul metodei, se începe prin căutarea în baza de date a unei persoane care să corespundă adresei de e-mail primite ca parametru. Aceasta se realizează prin apelarea metodei `findByEmail` a obiectului `personRepository`. Rezultatul căutării este încapsulat într-un obiect `Optional`, care poate conține o persoană sau poate fi gol în cazul în care nu există o persoană cu adresa de e-mail specificată.

Dacă obiectul `Optional` conține o persoană, se verifică dacă parola furnizată de utilizator se potrivește cu parola stocată în baza de date pentru acea persoană. Aceasta se face prin comparația parolei primite ca parametru cu parola stocată în obiectul `Person`. Dacă parolele coincid, este returnat un răspuns `HTTP 200 (OK)` împreună cu obiectul `Person` asociat cu adresa de e-mail dată.

În caz contrar, dacă parolele nu coincid, este returnat un răspuns HTTP 401 (Unauthorized), semnalând că autentificarea a eșuat din cauza unei parole greșite. În plus, dacă nu este găsită nicio persoană cu adresa de e-mail specificată, este returnat un răspuns HTTP 404 (Not Found), indicând că adresa de e-mail nu este asociată cu niciun cont din sistem.

```
54     @GetMapping(value = "/login")
55     public ResponseEntity<Person> loginUser(@RequestParam String email, @RequestParam String password) {
56         Optional<Person> optionalPerson = personRepository.findByEmail(email);
57
58         if (optionalPerson.isPresent()) {
59             Person person = optionalPerson.get();
60
61             if (person.getPassword().equals(password)) {
62                 return ResponseEntity.ok(person);
63             } else {
64                 return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
65             }
66         } else {
67             return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
68         }
69     }
```

2.3 Acces User/Admin

În aplicația noastră, accesul utilizatorilor și administratorilor este gestionat în funcție de rolurile atribuite fiecărei persoane. Pentru a permite navigarea către paginile corespunzătoare în funcție de rolul utilizatorului sau administratorului, am implementat o metodă `role` în `PersonController` și o funcționalitate de rutare în codul JavaScript al aplicației.

În cadrul metodei `role` din `PersonController`, se primește adresa de e-mail și parola utilizatorului și se caută în baza de date o persoană cu aceste date de autentificare. Dacă persoana este găsită și parola corespunde cu cea stocată în baza de date, metoda returnează rolul asociat persoanei. În caz contrar, se returnează rolul implicit al persoanei.

```
71     @GetMapping(value = "/role")
72     public PersonRole role(@RequestParam String email, @RequestParam String password) {
73         Optional<Person> optionalPerson = personRepository.findByEmail(email);
74
75         if (optionalPerson.isPresent()) {
76             Person person = optionalPerson.get();
77
78             if (person.getPassword().equals(password)) {
79                 return person.getPersonRole();
80             }
81         }
82         return optionalPerson.get().getPersonRole();
83     }
```

În partea de frontend a aplicației, utilizăm o funcție `onSubmit` pentru a gestiona procesul de autentificare. După ce primim un răspuns de la endpoint-ul `/login` care confirmă autentificarea cu succes, efectuăm o altă cerere către endpoint-ul `/role` pentru a obține rolul asociat utilizatorului sau administratorului autentificat.

Dacă rolul returnat este ADMIN, rutăm aplicația către pagina destinată administratorului, iar dacă rolul este USER, rutăm către pagina destinată utilizatorului. Acest lucru se realizează cu ajutorul funcției `navigate` din pachetul `@reach/router`, care ne permite să navigăm între rutele definite în aplicație.

În acest mod, utilizatorii și administratorii sunt direcționați către interfețele corespunzătoare în funcție de rolurile lor, oferindu-le o experiență personalizată și securizată în cadrul aplicației noastre. Această abordare îmbunătățește nu numai accesul utilizatorilor la funcționalitățile adecvate, dar și securitatea generală a aplicației.

```
18  const onSubmit = async (e) => {
19    e.preventDefault();
20
21    try {
22      const response = await axios.get(
23        `http://localhost:8080/login?email=${email}&password=${password}`
24      );
25
26      const loggedInUser = response.data;
27
28      if (loggedInUser) {
29        console.log("Login successful:", loggedInUser);
30        const role = await axios.get(
31          `http://localhost:8080/role?email=${email}&password=${password}`
32        );
33        if (role.data=="ADMIN") {
34          navigate("/admin");
35        } else {
36          navigate("/user");
37        }
38      } else {
39        console.error("Login failed");
40      }
41    } catch (error) {
42      console.error("Error during login:", error);
43    }
44  };
```

2.4 Email

În cadrul acestui proiect, MailDev joacă un rol esențial în procesele de înregistrare a utilizatorilor și confirmarea comenzilor. MailDev este un server SMTP de testare care rulează local pe mașina de dezvoltare și este folosit pentru capturarea și vizualizarea tuturor emailurilor trimise de aplicație în timpul dezvoltării, fără a le trimite de fapt în rețeaua externă.

Pentru procesul de validare și activare a contului, după ce un utilizator își creează un cont prin intermediul paginii de "Login/Register" din interfața utilizatorului, sistemul generează un token de verificare și îl stochează în baza de date asociată cu noul cont de utilizator. Componenta "Account Creation" folosește apoi MailDev pentru a trimite un email de verificare către adresa furnizată de utilizator în timpul înregistrării. Emailul conține un link cu token-ul de verificare

care, odată accesat de utilizator, validează și activează contul prin intermediul componentei "Verification Token".

În ceea ce privește procesul de confirmare a comenzilor, atunci când un utilizator finalizează o comandă și selectează produsele dorite, sistemul generează un PDF cu detaliile comenzii, incluzând descrierea și prețul produselor comandate. Componenta "User Wishlist/Management/Order" coordonează crearea acestui PDF. După finalizarea comenzii, componenta "Admin OrderManagement" utilizează MailDev pentru a trimite automat un email utilizatorului, care include PDF-ul ca atașament. Acest lucru confirmă că comanda a fost primită și furnizează utilizatorului un document de referință pentru produsele comandate.

MailDev este ideal într-un astfel de context, deoarece permite dezvoltatorilor să testeze și să vadă exact ce emailuri ar fi trimise într-un mediu de producție, fără a risca trimiterea de emailuri de test către utilizatorii reali. Odată ce aplicația este pregătită pentru lansare în producție, MailDev poate fi înlocuit cu un server SMTP real, care va trimite emailurile în rețeaua externă, către utilizatorii efectivi.



2.5 Review

Serviciul 'ReviewService' reprezintă o componentă esențială în cadrul aplicației noastre, având responsabilitatea de a gestiona recenziile asociate produselor comercializate în magazinul nostru online. Prin intermediul acestui serviciu, utilizatorii au posibilitatea să își exprime opiniile și experiențele lor cu privire la produsele achiziționate, ceea ce contribuie la transparența și încrederea în magazinul nostru online.

Una dintre funcționalitățile cheie ale serviciului 'ReviewService' este de a permite obținerea tuturor recenziilor pentru un produs specific. Această funcționalitate este crucială pentru transparența și informarea clienților, oferindu-le posibilitatea să vadă experiențele și opiniile altor utilizatori în legătură cu produsul respectiv.

De asemenea, serviciul permite și obținerea unei recenzii specifice după ID-ul său unic, permițând utilizatorilor să revizuiască sau să modifice recenziile anterioare. Această caracteristică oferă flexibilitate și control utilizatorilor asupra propriilor lor recenzii.

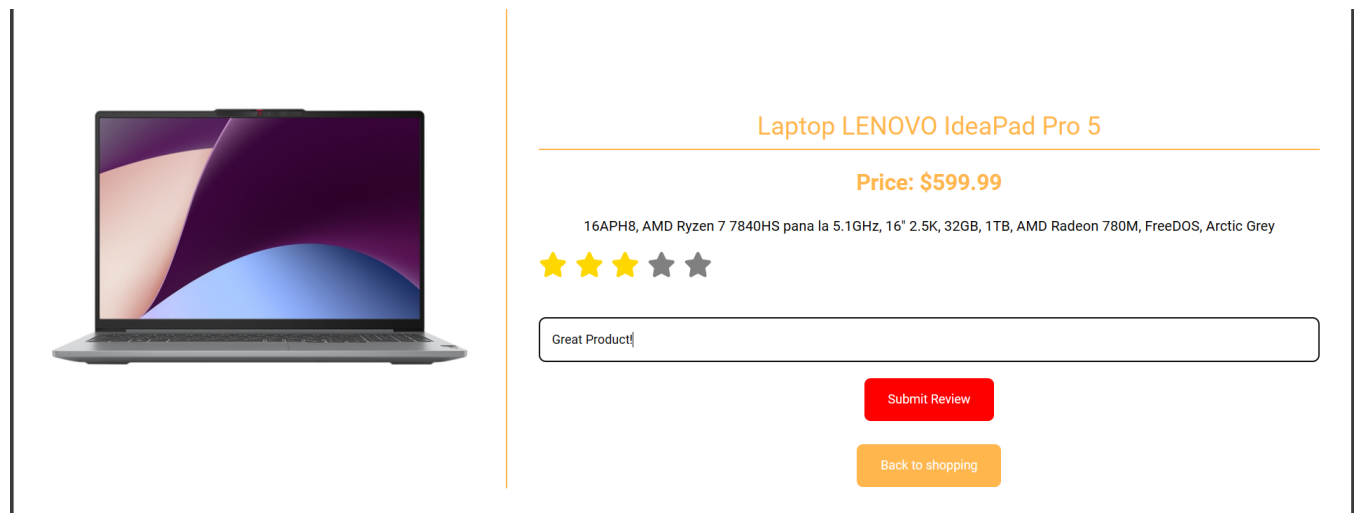
Pentru a asigura consistența și integritatea datelor, serviciul 'ReviewService' oferă funcționalitatea de salvare și actualizare a recenziilor. Utilizatorii pot adăuga noi recenzii sau pot actualiza recenziile existente, iar aceste informații sunt stocate și actualizate corespunzător în baza de date.

Un aspect esențial al serviciului este calculul ratingului mediu al produsului, care se bazează pe evaluările primite de la toți utilizatorii. Acest rating mediu este important pentru a oferi o perspectivă generală asupra calității unui produs și ajută clienții în procesul decizional de cumpărare.

O altă funcționalitate importantă a serviciului este gestionarea adăugării/recalculării unei recenzii. Utilizatorii pot adăuga noi recenzii sau pot actualiza recenziile existente, iar serviciul se ocupă de gestionarea corectă a acestor operațiuni pentru a asigura coerența și integritatea datelor.

În final, serviciul 'ReviewService' utilizează repository-urile asociate pentru a facilita interacțiunea cu baza de date și pentru a asigura un acces eficient și sigur la date. Aceste repository-uri joacă un rol crucial în gestionarea operațiilor CRUD pe recenzii și asigură integritatea și consistența datelor stocate.

Prin intermediul serviciului 'ReviewService', ne propunem să oferim o experiență de cumpărare transparentă și încredere clienților noștri, permițându-le să își împărtășească opiniile și să acceseze informațiile necesare pentru a lua decizii de cumpărare înțelepte.



Sistem de evaluare în formă de stele, utilizat pentru a permite utilizatorilor să acorde un rating unui anumit element sau conținut. În interiorul acestui div, se găsește o buclă care generează cinci componente `FontAwesomeIcon`, fiecare reprezentând o stea. Aceste stele sunt generate dinamic, iar fiecare stea este asociată unui anumit index, începând de la 1 și ajungând până la 5.

Componentele `FontAwesomeIcon` sunt utilizate pentru a reprezenta stelele grafic, iar fiecare stea are un icon specific, în acest caz, icon-ul unei stele. Fiecare stea are o clasă de stilizare atribuită (`star`) pentru a permite personalizarea aspectului vizual al acestora.

În timpul generării dinamice a stelelor, se verifică dacă index-ul stelei este mai mic sau egal cu valoarea rating-ului. Dacă da, steaua va fi colorată în auriu (gold), indicând faptul că acea stea face parte din rating-ul acordat de utilizator. În caz contrar, steaua va fi colorată în gri (gray), sugerând că acea stea nu face parte din rating-ul acordat.

În plus, fiecare stea are asociată o funcție de tratare a evenimentului `onClick`, care este activată atunci când utilizatorul dă clic pe o stea. Această funcție (`handleRating`) este responsabilă pentru gestionarea rating-ului acordat de utilizator, în funcție de steaua pe care a dat clic.

2.6 Wishlist

În cadrul aplicației noastre, clasa `Wishlist` joacă un rol esențial în gestionarea listei de dorințe a utilizatorilor. Această clasă reprezintă o entitate în modelul nostru de date și este responsabilă pentru stocarea și gestionarea informațiilor referitoare la produsele pe care utilizatorii doresc să le adauge în lista lor de dorințe.

Structura Clasei `Wishlist`:

Clasa `Wishlist` este definită ca o entitate JPA, marcându-se cu adnotarea `@Entity`. Acest lucru indică faptul că obiectele de tip `Wishlist` sunt persistate într-o bază de date relațională.

Funcționalitatea Metodei `addToWishlist()`:

Metoda `addToWishlist` este definită pentru a permite utilizatorilor să adauge noi elemente în lista lor de dorințe. Această metodă primește doi parametri: `userId` și `productId`, reprezentând identificatorii utilizatorului și produsului pe care utilizatorul dorește să-l adauge.

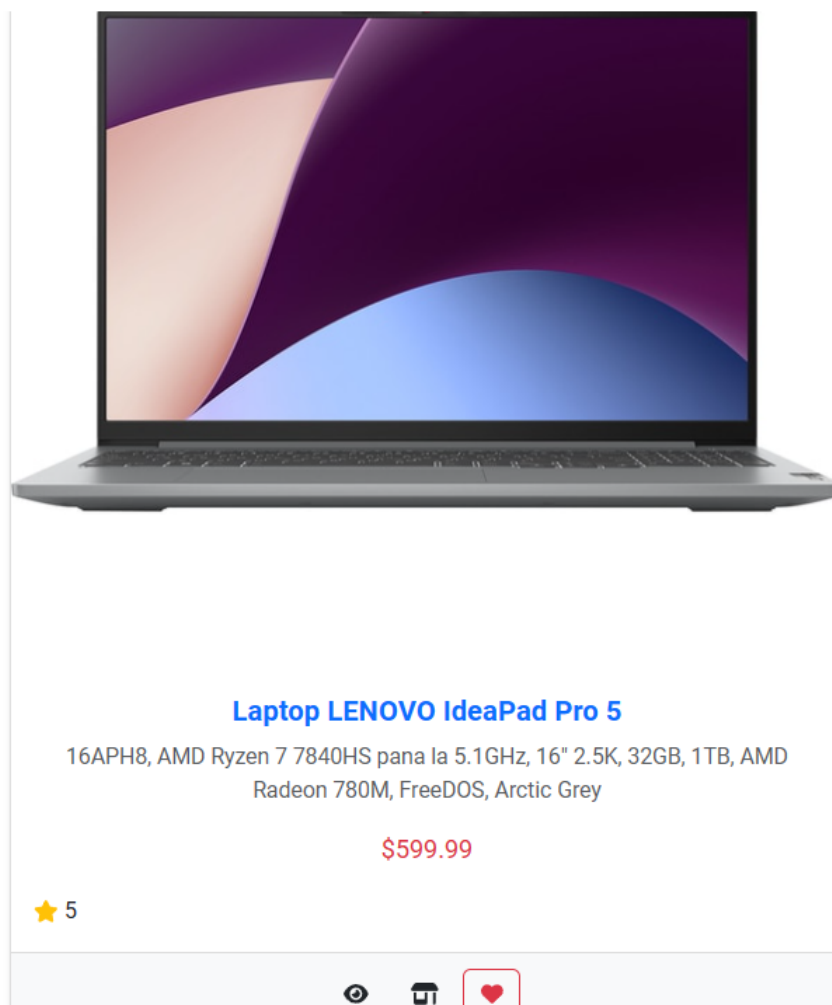
În interiorul metodei, utilizatorul și produsul sunt recuperate din baza de date utilizând repository-urile corespunzătoare. Dacă utilizatorul sau produsul nu există în baza de date, este aruncată o excepție `RuntimeException`.

Un nou obiect `Wishlist` este creat și inițializat cu utilizatorul și produsul corespunzător. Apoi, acest obiect este salvat în baza de date utilizând repository-ul asociat.

Importanța și Rolul în Aplicație:

Clasa `Wishlist` este crucială în funcționarea aplicației noastre, oferind utilizatorilor o modalitate simplă și convenabilă de a-și organiza și gestiona preferințele de cumpărare. Prin intermediul listei de dorințe, utilizatorii pot să-și exprime interesele și să-și planifice achizițiile viitoare într-un mod organizat și eficient.

Această funcționalitate este esențială într-un mediu de e-commerce, contribuind la creșterea angajamentului utilizatorilor și la îmbunătățirea experienței lor în cadrul platformei noastre. Prin intermediul clasei `Wishlist`, ne asigurăm că utilizatorii noștri beneficiază de un instrument util și intuitiv pentru a-și gestiona preferințele de cumpărare, consolidând astfel relația noastră cu aceștia și creând o experiență de cumpărare plăcută și personalizată.



Structura și Funcționalitatea Componentei Wishlist:

Componenta Wishlist este responsabilă pentru afișarea listei de produse din wishlist-ul utilizatorului curent. Aceasta primește ID-ul utilizatorului ca parametru în URL, extrăgându-l folosind hook-ul `useParams` din React Router. Apoi, utilizează acest ID pentru a încărca lista de produse asociate acestui utilizator din backend.

În momentul încărcării, componenta efectuează o cerere către backend pentru a obține lista de produse din wishlist. Această cerere este realizată către un endpoint specific, care primește ID-ul utilizatorului și returnează lista de produse asociate acestuia.

Afisarea Produselor în Wishlist:

Produsele sunt afișate sub formă de tabel, fiecare rând reprezentând un produs din wishlist-ul utilizatorului. Informațiile afișate pentru fiecare produs includ numele, prețul, scurta descriere și o imagine reprezentativă.

Pentru fiecare produs afișat, utilizatorul are opțiunea de a vedea detaliile acestuia și de a decide dacă dorește să-l adauge în coșul de cumpărături sau nu.

Interacțiunea cu Utilizatorul:

Pentru a adăuga un produs în coșul de cumpărături, utilizatorul trebuie să apese butonul cu iconița de inimă (faHeart) asociat fiecărui produs din wishlist. Acest buton va fi integrat în interfața utilizatorului pentru a oferi o modalitate intuitivă de a interacționa cu produsele din wishlist și de a le adăuga în coșul de cumpărături.

2.7 Cart

Funcționalitatea coșului de cumpărături este esențială în cadrul aplicației noastre, oferind utilizatorilor posibilitatea de a adăuga produse într-un coș temporar pentru a le achiziționa ulterior. Componenta `CartService` este responsabilă pentru gestionarea operațiilor legate de coșul de cumpărături, precum adăugarea și eliminarea produselor din coș.

Funcționalitatea de Adăugare în Coș:

Atunci când un utilizator adaugă un produs în coș, această acțiune nu afectează în mod direct cantitatea disponibilă în stoc a produsului. În schimb, produsul este temporar rezervat pentru utilizatorul respectiv și rămâne în coșul său până când acesta finalizează sau renunță la achiziție. Cantitatea rezervată în coș nu este scăzută din inventarul produsului până când utilizatorul finalizează comanda.

Gestionarea Produselor în Coș:

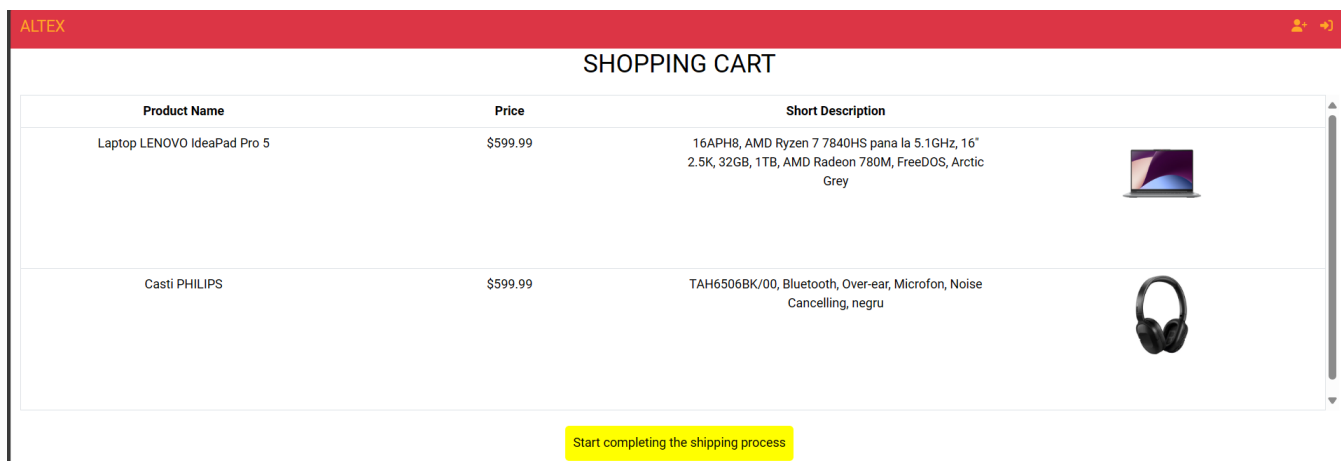
Componenta `CartService` oferă funcționalități pentru a obține și gestiona produsele din coșul de cumpărături al unui utilizator. Utilizând metodele adecvate, putem afișa produsele din coșul unui utilizator și putem permite utilizatorului să elimine produse din coș, dacă este necesar.

Procesul de Comandă:

Odată ce utilizatorul decide să finalizeze comanda, produsele din coșul său vor fi transferate în procesul de comandă, unde vor fi pregătite pentru expediere și livrare. În această etapă, cantitatea produselor va fi scăzută din inventarul disponibil pentru a reflecta achiziția efectuată.

Importanța și Rolul în Aplicație:

Funcționalitatea coșului de cumpărături este vitală pentru experiența utilizatorului și pentru procesul de achiziție. Oferind utilizatorilor un coș de cumpărături flexibil și eficient, ne asigurăm că aceștia pot să-și organizeze și să-și gestioneze achizițiile într-un mod convenabil și intuitiv. Prin intermediul acestei funcționalități, consolidăm relația noastră cu utilizatorii și îmbunătățim experiența lor pe platforma noastră, facilitând procesul de cumpărare și fidelizând clienții.



Vizualizarea Produselor în Coș:

Atunci când un utilizator accesează pagina coșului de cumpărături, sunt afișate detaliile produselor adăugate anterior în coș. Aceste detalii includ numele produsului, prețul, o scurtă descriere și o imagine reprezentativă pentru fiecare produs. Utilizatorii pot naviga în cadrul listei pentru a verifica produsele adăugate și informațiile asociate acestora.

Finalizarea Achiziției:

Pentru a finaliza achiziția, utilizatorii trebuie să apese butonul "Start completing the shipping process". Acest buton îi va redirecționa către pagina de procesare a comenzii, unde pot introduce detaliile de livrare și alte informații necesare pentru finalizarea achiziției.

2.8 Order

Funcția OrderService reprezintă un element esențial al sistemului nostru de comerț electronic, responsabil pentru gestionarea comenzilor și trimiterea confirmărilor de comandă către utilizatori.

Gestionarea Comenzilor:

Adăugarea în Coș: Utilizatorii pot adăuga produse în coșul de cumpărături prin intermediul acestei funcționalități. Atunci când un utilizator selectează un produs pentru achiziție, acesta este adăugat în coșul său de cumpărături. Informațiile despre utilizator și produs sunt stocate în baza de date pentru a fi utilizate ulterior în procesul de generare a confirmării comenzii.

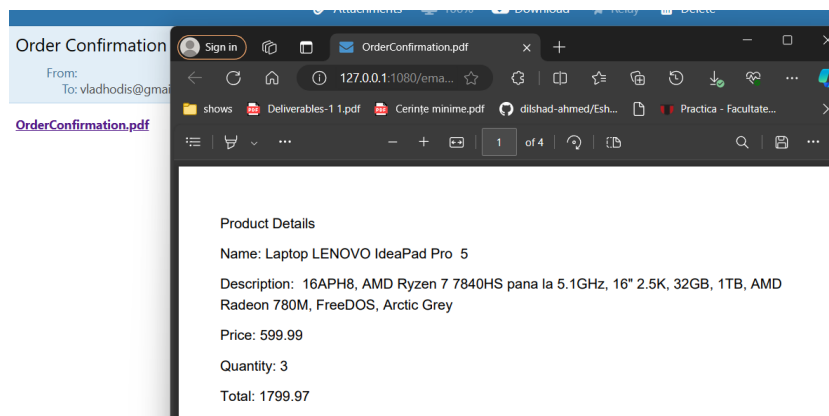
Eliminarea din Coș: Utilizatorii pot elimina produse din coșul de cumpărături în cazul în care doresc să renunțe la acestea sau să facă modificări înainte de finalizarea comenzii. Funcția `removeFromOrder` permite utilizatorilor să elimine produsele selectate din coș, actualizând astfel starea coșului lor de cumpărături.

Generarea Confirmării Comenzii: După ce utilizatorii finalizează procesul de cumpărături și plasează comanda, această funcție generează un fișier PDF care conține detalii complete despre produsele comandate și costurile asociate. Acest fișier PDF este atașat unui email de confirmare a comenzii și trimis către adresa de email furnizată de utilizator.

Trimiterea Confirmării Comenzii: Utilizând serviciul de email integrat, funcția `sendOrderConfirmationEmail` trimite un email de confirmare a comenzii către utilizatorii care au plasat o comandă. Acest email conține fișierul PDF generat anterior, care conține detalii complete despre comanda lor și costurile asociate.

Importanța și Rolul în Aplicație:

OrderService joacă un rol crucial în asigurarea unui proces de achiziție fără probleme și în oferirea unei experiențe plăcute utilizatorilor noștri. Prin intermediul acestei funcționalități, putem gestiona eficient comenzile utilizatorilor și le putem oferi informații clare și concise despre achizițiile lor. De asemenea, trimiterea confirmărilor de comandă prin email ajută la consolidarea încrederii utilizatorilor în platforma noastră și la asigurarea unei comunicări eficiente și transparente în cadrul procesului de cumpărare.



Buton pentru Plasarea Comenzii: La finalul listei de produse, utilizatorii au opțiunea de a plasa comanda prin intermediul butonului "Place Order". Acest buton va declanșa procesul de finalizare a comenzii și va trimite cererea către server pentru a finaliza procesul de cumpărare.

2.9 Chat

Chat-ul în timp real între utilizatori este o componentă esențială a oricărei aplicații care vizează interacțiunea și comunicarea între utilizatori. Această funcționalitate permite utilizatorilor să comunice în timp real, să împărtășească idei, să discute sau să colaboreze, îmbunătățind astfel experiența de utilizare și promovând interactivitatea.

Arhitectură și Implementare

WebSocket-uri și Protocol STOMP: Implementarea chat-ului în timp real se bazează pe WebSocket-uri, un protocol de comunicație bidirecțională care permite comunicarea în timp real între server și client. Protocolul STOMP (Simple Text Oriented Messaging Protocol) este folosit pentru a defini mesajele și destinațiile pe care le folosește aplicația.

Biblioteci Utilizate: Pentru gestionarea comunicării WebSocket și a protocolului STOMP, această aplicație folosește bibliotecile sockjs-client și @stomp/stompjs. Aceste biblioteci facilitează conectarea la serverul WebSocket, trimiterea și recepționarea mesajelor într-un mod eficient și sigur.

Comunicare între Frontend și Backend: Chat-ul în timp real este implementat într-o componentă React pe frontend și comunică cu serverul backend prin intermediul API-ului REST. În plus, pentru a obține numele utilizatorului, componenta face o cerere către backend pentru a extrage această informație pe baza ID-ului utilizatorului.

Funcționalități și Interfață Utilizator Conectarea la Serverul WebSocket: În momentul încărcării componentei, aceasta se conectează la serverul WebSocket utilizând bibliotecile menționate anterior. Această conexiune este esențială pentru trimiterea și recepționarea mesajelor în timp real.

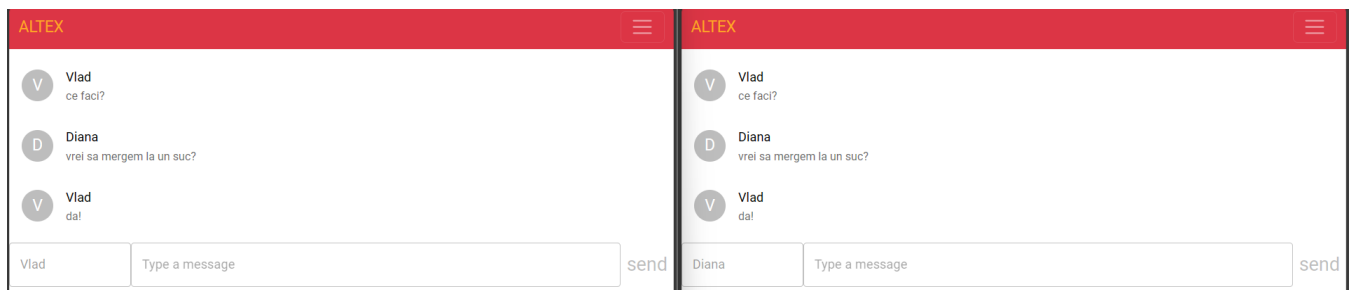
Trimiterea și Recepționarea Mesajelor: Utilizatorii pot trimite și recepționa mesaje în timp real prin intermediul chat-ului. Mesajele sunt afișate într-o listă și conțin numele utilizatorului și conținutul mesajului.

Identificarea Utilizatorilor: Pentru a identifica utilizatorii, numele lor sunt afișate în câmpul de introducere a mesajului. Acest lucru se realizează prin intermediul unei cereri către API-ul nostru pentru a extrage numele utilizatorului pe baza ID-ului furnizat în parametrii rutei.

Interacțiune Intuitivă: Interfața utilizatorului este intuitivă și ușor de folosit. Utilizatorii pot introduce mesaje într-un câmp de text și le pot trimite printr-un clic pe butonul de trimitere.

Importanța și Impactul în Aplicație

Chat-ul în timp real între utilizatori are un impact semnificativ asupra experienței de utilizare și interactivității aplicației. Acesta permite utilizatorilor să comunice și să interacționeze în timp real, îmbunătățind colaborarea și facilitând schimbul de informații. De asemenea, promovează implicarea utilizatorilor și oferă o modalitate eficientă de a rezolva probleme sau de a clarifica întrebări în timp real.



2.10 XML

Această funcționalitate implică două componente principale: serviciul XmlService și metoda getUserXml din clasa controlerului. Haideți să explicăm fiecare componentă în detaliu.

Serviciul XmlService

Serviciul XmlService este responsabil pentru conversia unui obiect User într-un format XML. Folosește biblioteca Jackson pentru a realiza această conversie. Iată o explicație detaliată a funcționalității sale:

Inițializarea și Configurarea: În constructorul său, XmlService inițializează un obiect XmlMapper din biblioteca Jackson. Acesta este configurat pentru a gestiona serializarea obiectelor în format XML conform cerințelor noastre.

Metoda userToXml: Această metodă primește un obiect User ca parametru și returnează reprezentarea sa în format XML. Folosind XmlMapper, obiectul User este serializat într-un șir de caractere XML folosind metoda writeValue a mapper-ului.

Metoda getUserXml din Controler Această metodă este responsabilă pentru gestionarea cererilor de generare a fișierelor XML pentru un utilizator specific. Iată cum funcționează:

Obținerea Utilizatorului: Metoda primește ID-ul utilizatorului pentru care se dorește generarea fișierului XML. Folosind repository-ul, obține obiectul User corespunzător ID-ului din baza de date.

Conversia în XML și Returnarea Răspunsului:

Utilizează serviciul XmlService pentru a converti obiectul User în format XML. Apoi, configurează răspunsul pentru a trimite fișierul XML către client ca un atașament. Acest lucru se realizează prin adăugarea unor antete specifice răspunsului HTTP, cum ar fi Content-Disposition și Content-Type.

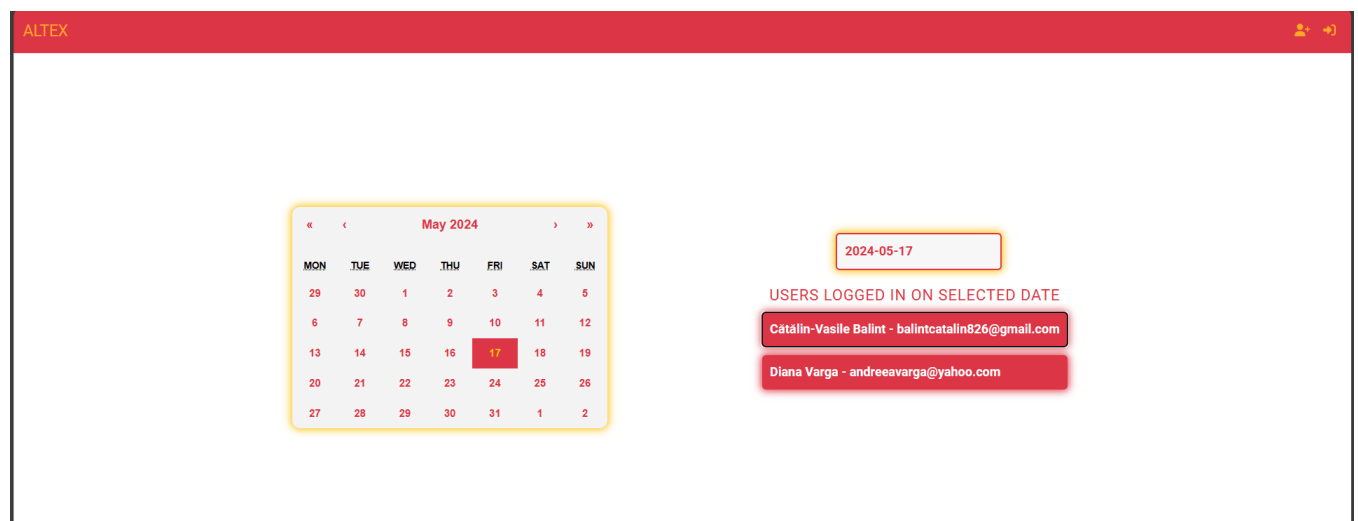
Accesul Limitat la Funcționalitate

Menționați că accesul la această funcționalitate este limitat doar la utilizatorii cu rol de administrator. Acest lucru poate fi implementat folosind mecanismele de securitate ale aplicației, cum ar fi autentificarea și autorizarea bazată pe roluri. De exemplu, puteți utiliza Spring Security pentru a restricționa accesul la această metodă doar utilizatorilor autentificați ca administratori.

Această funcționalitate este utilă pentru a permite administratorilor să genereze rapoarte sau exporturi în format XML pentru utilizatorii din sistem, permițând o administrare mai eficientă a datelor și o analiză mai ușoară.

Această explicație detaliată ar trebui să ofere o înțelegere clară a modului în care funcționează generarea fișierelor XML pentru utilizatori în cadrul aplicației.

```
This XML file does not appear to have any style information associated with it. The document tree is shown below.
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<User>
  <id>2</id>
  <firstName>Cătălin-Vasile</firstName>
  <lastName>Balint</lastName>
  <email>balintcatalin826@gmail.com</email>
  <password>$2a$10$cDwZ664g4nZ.9y7WVpZF.GFsZb0B0Kw/1atvIudXptB44bgwVEq2</password>
  <appUserRole>ADMIN</appUserRole>
  <locked>false</locked>
  <enabled>true</enabled>
  <wishlist/>
  <order/>
  <lastLogged>2024-05-15T13:15:38.615023</lastLogged>
  <counter>5</counter>
  <username>balintcatalin826@gmail.com</username>
  <authorities>
    <authority>ADMIN</authority>
  </authorities>
  <accountNonExpired>true</accountNonExpired>
  <credentialsNonExpired>true</credentialsNonExpired>
  <accountNonLocked>true</accountNonLocked>
</User>
```



CalendarComponent este o componentă React care utilizează mai multe biblioteci externe și funcționalități pentru a oferi utilizatorilor posibilitatea de a selecta o dată dintr-un calendar și de a vizualiza utilizatorii care s-au autentificat ultima dată la acea dată. Componenta importă React, utilizând hook-urile useState și useEffect pentru a gestiona starea și efectele secundare. De asemenea, importă Calendar din biblioteca react-calendar pentru afișarea calendarului, axios pentru realizarea cererilor HTTP și fișiere CSS pentru stilizare.

Componenta definește două state-uri prin hook-ul useState: date, care păstrează data selectată din calendar, și users, care stochează lista de utilizatori returnată de server. Data inițială este setată la data curentă. Funcția formatDate este utilizată pentru a formata data într-un format specific (YYYY-MM-DD), necesar pentru cererile către server.

Funcția downloadUserXml este responsabilă pentru descărcarea informațiilor unui utilizator în format XML. Aceasta trimite o cerere GET către server, primește datele ca un blob și creează

un URL de descărcare pentru fișierul XML. În cazul unei erori, aceasta este capturată și afișată în consolă.

Funcția `fetchUsers` trimite o cerere GET către server pentru a obține lista de utilizatori care s-au autentificat ultima dată la data selectată. Aceasta formatează data folosind `formatDate`, trimite cererea și setează state-ul `users` cu datele primite. Dacă cererea eșuează, eroarea este capturată și afișată în consolă, iar lista de utilizatori este resetată.

Hook-ul `useEffect` este folosit pentru a apela `fetchUsers` de fiecare dată când date se schimbă. Acesta asigură că lista de utilizatori este actualizată automat când utilizatorul selectează o nouă dată din calendar.

Funcția `onChange` actualizează state-ul `date` cu noua dată selectată din calendar. Aceasta permite componentului să răspundă la schimbările de dată făcute de utilizator.

În funcția de returnare, componenta afișează calendarul și lista de utilizatori. Calendarul permite utilizatorilor să selecteze o dată, iar lista afișează utilizatorii care s-au autentificat la data selectată, oferind și opțiunea de descărcare a datelor lor în format XML prin intermediul unor butoane. Dacă nu sunt găsiți utilizatori pentru data respectivă, este afișat un mesaj corespunzător.

În final, componenta este exportată pentru a putea fi utilizată în alte părți ale aplicației. Aceasta integrează eficient selecția de dată și interacțiunea cu serverul, oferind o experiență intuitivă și funcțională utilizatorilor.

2.11 Payment

În cadrul unui sistem de gestionare a plăților, componenta `CardService` dintr-o aplicație Spring oferă funcționalități esențiale pentru manipularea datelor referitoare la carduri și notificarea utilizatorilor prin email asupra succesului plății. Acest serviciu integrează un repository pentru accesul la date și un serviciu de trimitere a emailurilor, asigurând astfel o experiență completă și informată pentru utilizatori.

Structura și Dependente

Clasa `CardService` este declarată ca un serviciu Spring prin anotarea `@Service`, ceea ce permite gestionarea sa ca bean de către contextul Spring. Aceasta depinde de două componente externe: `CardRepository` pentru operațiunile CRUD asupra entităților `Card` și `EmailSender` pentru trimiterea emailurilor de notificare. Aceste dependențe sunt injectate prin constructor folosind anotarea `@Autowired`.

Metoda Privată de Trimitere a Emailurilor Metoda privată `sendEmailNotification()` este responsabilă pentru trimiterea unui email de notificare utilizând `EmailSender`. Emailul conține un mesaj simplu "PAYMENT SUCCESSFUL", trimis la adresa specificată.

Funcționalitatea de Plată cu Notificare prin Email

În cadrul unui sistem de gestionare a plăților, componenta `CardService` dintr-o aplicație Spring oferă funcționalități esențiale pentru manipularea datelor referitoare la carduri și notificarea utilizatorilor prin email asupra succesului plății. Acest serviciu integrează un repository pentru accesul la date și un serviciu de trimitere a emailurilor, asigurând astfel o experiență completă și informată pentru utilizatori.

Structura și Dependente

Clasa `CardService` este declarată ca un serviciu Spring prin anotarea `@Service`, ceea ce permite gestionarea sa ca bean de către contextul Spring. Aceasta depinde de două componente externe: `CardRepository` pentru operațiunile CRUD asupra entităților `Card` și `EmailSender`

pentru trimiterea emailurilor de notificare. Aceste dependențe sunt injectate prin constructor folosind anotarea `@Autowired`.

Fluxul de Plată

În momentul în care un card este salvat prin intermediul metodei `saveCard(Card card)`, se declanșează automat trimiterea emailului de notificare prin apelarea `sendEmailNotification()`. Aceasta asigură că, pentru fiecare operațiune de salvare (ce poate reprezenta și o plată), utilizatorul primește confirmarea succesului tranzacției.

Clasa `CardService` oferă funcționalități esențiale pentru gestionarea cardurilor și notificarea utilizatorilor într-un sistem de plată. Prin integrarea unui repository pentru accesul la date și a unui serviciu de trimitere a emailurilor, această componentă asigură atât persistența datelor, cât și informarea promptă a utilizatorilor asupra succesului plăților. Astfel, `CardService` contribuie la o experiență de utilizator eficientă și transparentă, esențială într-o aplicație de gestionare a plăților.

Payment Details

Full Name	
Balint Cătălin-Vasile	
Card Number	
123456789076	
CVV	Expiration Date
089	08/26

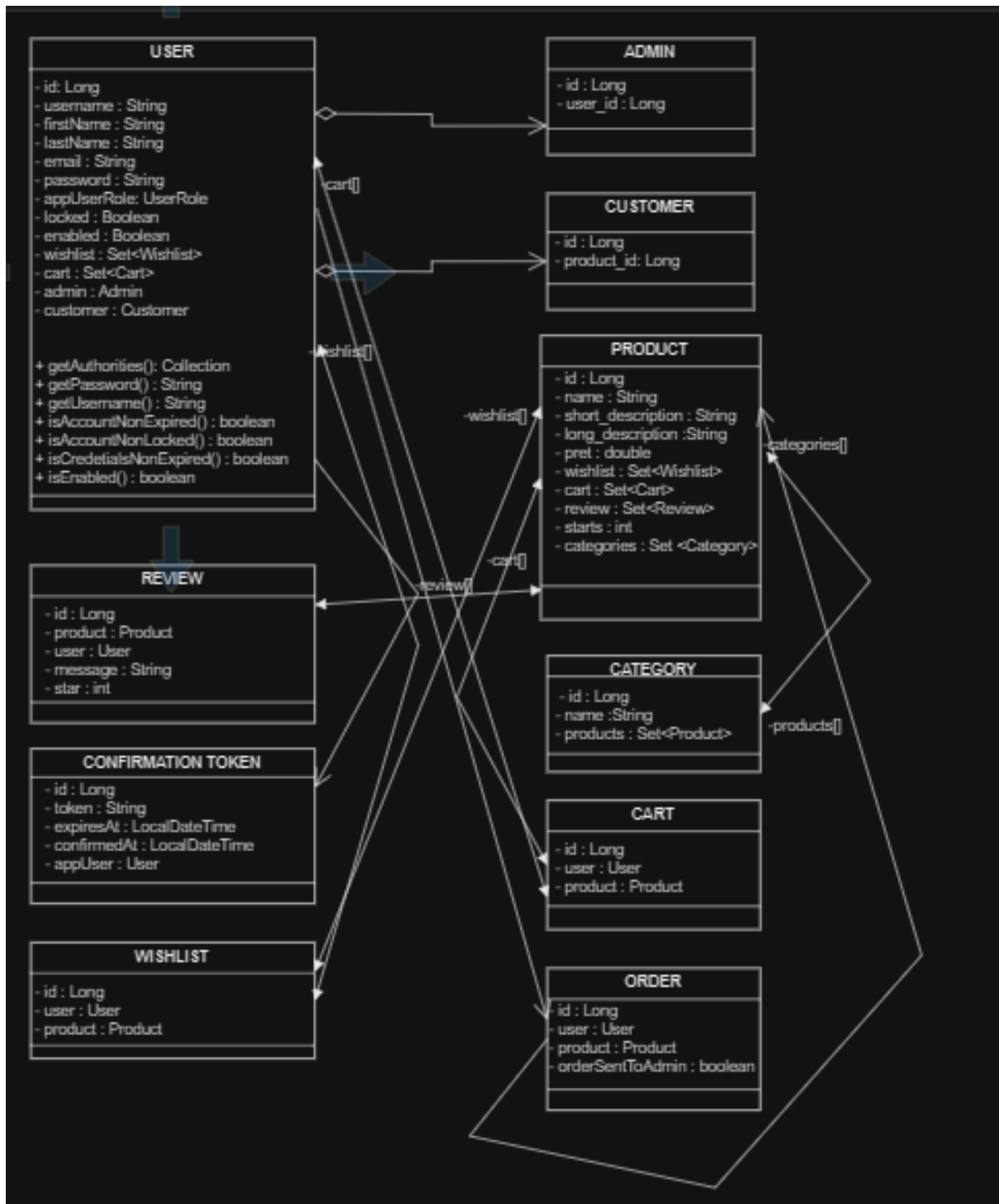
Address	
Strada Armeneasca numarul 8	
City	
Gherla	
Country	
Romania	
Zip Code	
405300	

Submit Payment Details and Place Order

3 Domain and Data Model

Un model de domeniu este o reprezentare vizuală care descrie structura și relațiile dintre diferitele entități dintr-un anumit domeniu de interes. Aceste entități sunt de obicei clasificate în clase, iar modelul include descrierea atributelor (proprietăți sau caracteristici) și metodelor (comportamente sau acțiuni) corespunzătoare fiecărei clase. Relațiile dintre clase, cum ar fi

asociațiile, moștenirile și agregările, sunt, de asemenea, evidențiate pentru a oferi o înțelegere clară a modului în care interacționează diferitele părți ale sistemului. Modelul de domeniu este crucial în fazele inițiale ale dezvoltării software pentru a asigura o înțelegere solidă a domeniului problemei și pentru a facilita o proiectare eficientă și orientată pe obiect a sistemului software.



Clasa Admin reprezintă un administrator într-un sistem, stocând detaliile sale în baza de date. Aceasta folosește Lombok pentru a genera automat gettere și settere și este legată de un obiect User printr-o relație unu-la-unu, gestionată cu ajutorul JPA (Java Persistence API), indicând o legătură strânsă între admin și contul său de utilizator.

Clasa `Cart` modelează un coș de cumpărături, legându-l de un utilizator și un produs printr-o relație `ManyToOne`. Folosește Lombok pentru funcționalități automate, cum ar fi getteri, setteri și egalitatea obiectelor. Relațiile sunt configurate pentru încărcare leneșă și sunt excluse din serializarea JSON pentru a optimiza performanța.

Clasa `Category` reprezintă o categorie de produse, identificată prin ID și nume, și are o relație `ManyToMany` cu produsele. Relația este configurată pentru încărcare leneșă și este exclusă din serializarea JSON.

Clasa `ConfirmationToken` gestionează token-uri de confirmare pentru utilizatori, având atribute pentru ID, token, data creării, data expirării și data confirmării. Include o relație `ManyToOne` cu utilizatorul asociat. Atributele sunt gestionate printr-un generator de secvență specific pentru asigurarea unicității ID-urilor.

Clasa `Customer` reprezintă un client, având un ID unic și o relație `OneToOne` cu un obiect `User`. Aceasta folosește Lombok pentru automatizarea getterelor, setterelor și metodelor esențiale. Relația cu utilizatorul permite asocierea directă a detaliilor de cont cu clientul.

Clasa `Order` reprezintă o comandă plasată în sistem, asociind un utilizator și un produs printr-o relație `ManyToOne`. Utilizează Lombok pentru a genera automat gettere, settere și alte metode. Comanda include și un flag `orderSentToAdmin` pentru a indica dacă comanda a fost trimisă administratorului. Relațiile sunt configurate pentru încărcare leneșă și sunt excluse din serializarea JSON.

Clasa `Product` reprezintă un produs într-un sistem de e-commerce, având atribute pentru ID, nume, descriere scurtă, descriere lungă, preț, și cantitate. Produsul este legat de multiple liste de dorințe, coșuri de cumpărături și recenzii printr-o relație `OneToMany`, toate fiind ignorate în serializarea JSON pentru a evita problemele de performanță. De asemenea, are o relație `ManyToMany` cu categoria de produse, facilitând clasificarea produselor în diverse categorii.

Clasa `Review` reprezintă o recenzie a unui produs, conținând un ID unic, mesajul recenziei, și un rating sub formă de stele. Aceasta are relații `ManyToOne` cu produsul și utilizatorul evaluat, ambele configurate pentru încărcare leneșă și excluse din serializarea JSON pentru a îmbunătăți performanța.

Clasa `User` implementează interfața `UserDetails` și reprezintă un utilizator în sistemul de autentificare, având atribute pentru ID, nume, prenume, email, parolă, și roluri definite prin enum-ul `UserRole`. Include funcționalități de securitate precum verificarea dacă contul este expirat sau blocat. Utilizatorul este asociat cu entități `Wishlist` și `Cart` prin relații `OneToMany`, și poate fi legat de un `Admin` sau un `Customer` printr-o relație `ManyToOne`, acestea din urmă fiind excluse din serializarea JSON. Autoritățile sunt gestionate printr-un `SimpleGrantedAuthority` bazat pe rolul utilizatorului.

Clasa `Wishlist` reprezintă o listă de dorințe a unui utilizator, conținând un ID unic și relații `ManyToOne` cu un utilizator și un produs. Ambele relații sunt configurate pentru încărcare leneșă și sunt excluse din serializarea JSON pentru a evita incluziunea lor în serializările obiectelor.

4 Architectural Design

4.1 Conceptual Architecture

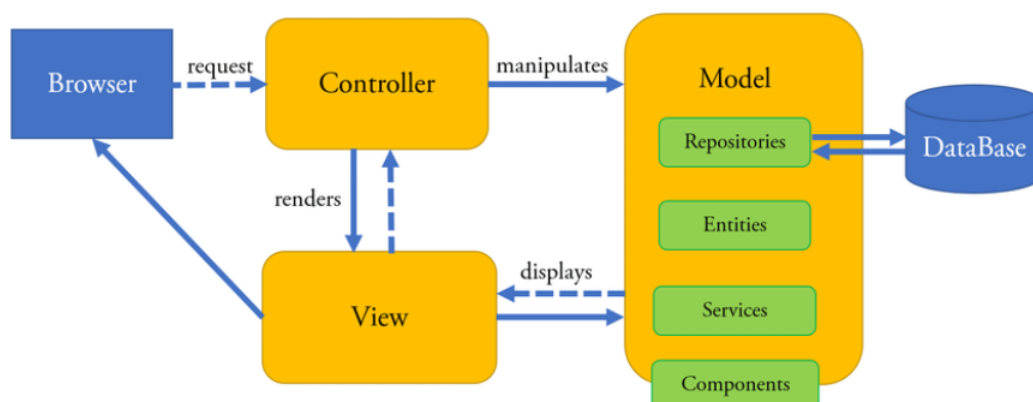
Arhitectura MVC (Model-View-Controller) este o abordare a proiectării software-ului care separă aplicația în trei componente principale, fiecare având roluri și responsabilități distincte.

Această separare ajută la organizarea codului, îmbunătățind scalabilitatea și întreținerea, și facilitează dezvoltarea colaborativă.

Modelul în MVC servește ca reprezentare a datelor și a logicii de afaceri. În interiorul componentei Model, avem entități care reflectă structuri de date și modelele de domeniu necesare pentru aplicație. Entitățile interacționează cu baza de date prin intermediul repository-urilor, care abstractizează și encapsulează operațiile de acces la date. Serviciile conțin logica de afaceri care operează asupra datelor și pot fi reutilizate în diferite părți ale aplicației. Componentele pot include funcționalități transversale, cum ar fi autentificarea, logarea sau gestionarea erorilor.

Controllerul este intermediarul dintre utilizator (sau alte sisteme) și aplicație. El primește solicitările de la utilizatori, le procesează - eventual interogând Modelul sau modificându-l - și decide ce răspuns să returneze. Controllerul mapează acțiunile utilizatorilor la logica de afaceri, delegând taskurile către Model și alegând View-ul corespunzător pentru a prezenta rezultatul.

View-ul este responsabil pentru prezentarea datelor utilizatorului și poate fi compus din cod HTML, CSS și JavaScript în contextul aplicațiilor web. Este partea aplicației pe care utilizatorul o vede și cu care interacționează. View-ul solicită informații de la Model (de obicei prin intermediul Controllerului) și le afișează într-un format accesibil. Deși View-ul poate fi tentat să acceseze Modelul direct pentru date, în MVC pur, toate schimburile de informații ar trebui să treacă prin Controller pentru a menține separarea clară a responsabilităților.



Această separare clară oferită de MVC aduce numeroase avantaje. Ea facilitează dezvoltarea modulară și testarea independentă a componentelor - View-urile pot fi schimbate sau modificate fără a influența Modelul sau Controllerul. De asemenea, echipele de dezvoltatori pot lucra pe componente separate simultan, accelerând dezvoltarea. Separarea îmbunătățește, de asemenea, posibilitatea de a extinde și de a menține aplicația, deoarece schimbările într-o parte a sistemului au impact minim asupra celorlalte. De exemplu, o schimbare în logica de afaceri în Model nu necesită modificări în View sau în Controller, atât timp cât interfața Modelului rămâne constantă. Aceasta poate duce la o mai bună gestionare a codului și la un cod mai curat și mai ușor de întreținut.

4.2 Package Design

Pe partea de backend pe langa pachetele specifice utilizate în cadrul arhitecturii MVC , mai se regasesc si alte pachete care contribuie la actionare facilitara a functionalitatilor proiectului.

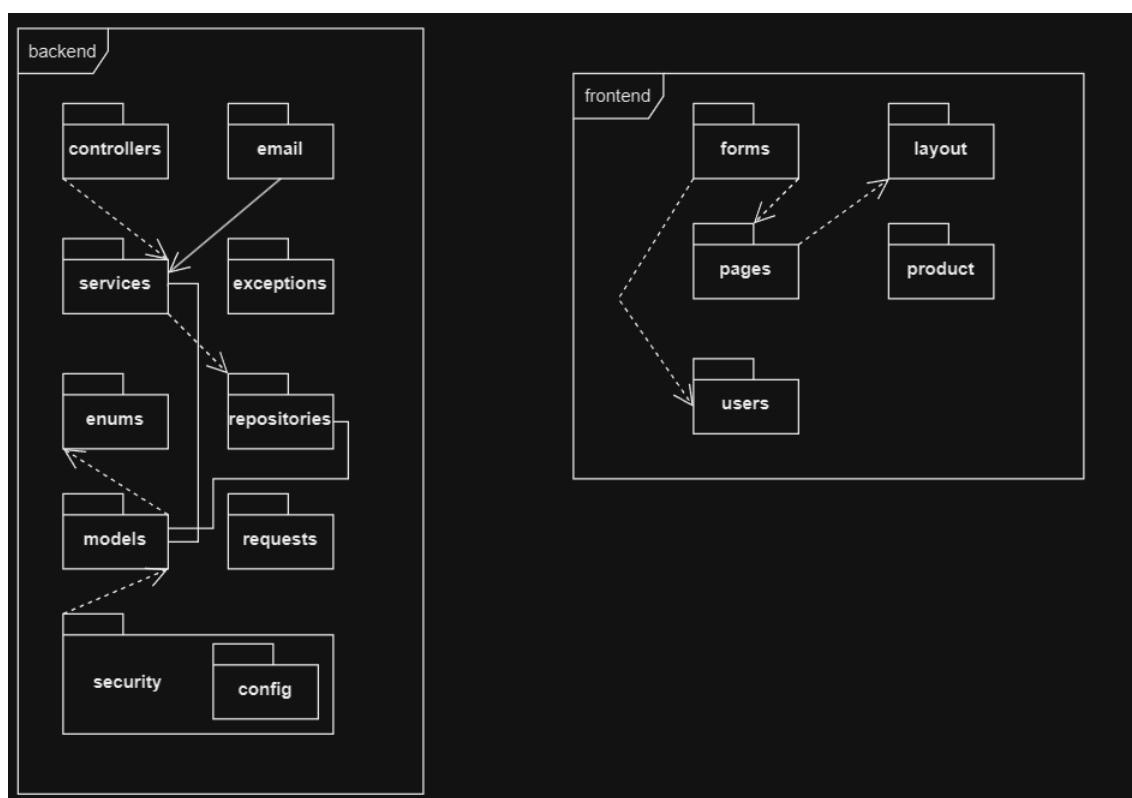
Pachetul requests pare să conțină clase Java care reprezintă diferite tipuri de cereri pe care aplicația web le poate primi. Numele claselor sugerează că ele se ocupă de operațiuni specifice,

cum ar fi gestionarea coșului de cumpărături (CartRequest), autentificarea utilizatorului (LoginRequest), procesarea comenzilor (OrderRequest), înregistrarea de noi utilizatori (RegistrationRequest) și gestionarea listei de dorințe (WishlistRequest). Aceste clase pot defini structura datelor de intrare pentru fiecare tip de cerere și pot fi utilizate pentru a valida și a transporta datele în aplicație.

Pachetul security se referă la aspectele legate de securitatea aplicației. Acesta conține un subpachet denumit config, care sugerează configurări legate de securitate. Clasa WebSecurityConfig poate fi utilizată pentru a configura aspecte de securitate web, cum ar fi autentificarea, autorizarea, reglementările CORS (Cross-Origin Resource Sharing) și alte setări de securitate. PasswordEncoder este probabil o clasă care se ocupă de codificarea (hashing) parolelor utilizatorilor înainte de a le stoca în baza de date, asigurând că parolele sunt gestionate într-un mod sigur.

Pachetul email include clase ce par să fie legate de trimiterea și validarea e-mailurilor. EmailSender este probabil o clasă care se ocupă de logica de trimitere a e-mailurilor în aplicație, poate prin conectarea la un server SMTP sau prin folosirea unui serviciu de e-mail terț. EmailValidator este posibil să fie o clasă responsabilă pentru validarea adreselor de e-mail, asigurându-se că sunt într-un format corect și poate chiar verificând existența lor.

Pachetul enums conține o enumerație UserRole, care este probabil utilizată pentru a defini diferitele roluri ale utilizatorilor în sistem, cum ar fi utilizator, admin sau customer.



Pachetul exceptions conține clase care sunt legate de gestionarea excepțiilor specifice aplicației. Clasele de tip `*NotFoundException` sugerează că aceste excepții sunt aruncate când o anumită entitate (admin, client sau utilizator) nu poate fi găsită.

Arhitectura frontend a platformei noastre de comerț electronic este structurată în diverse pachete pentru a asigura o separare clară a funcționalităților și pentru a ușura întreținerea unei arhitecturi.

Pachetul "layout" include componente reutilizabile ce definesc structura de bază a interfeței cu utilizatorul, cum ar fi bara de navigație și subsolul paginii. Componentele din pachetul "forms" se ocupă de gestionarea formularelor și colectarea datelor esențiale pentru procese precum autentificarea și înregistrarea, precum și alte interacțiuni cu utilizatorii. În "products", găsim elementele legate de afișarea și gestionarea gamei de produse, care sunt vitale pentru prezentarea și administrarea ofertei noastre de produse.

De asemenea, pachetul "users" este esențial, deoarece conține toate componentele care se ocupă de administrarea conturilor utilizatorilor, de la profiluri personale până la preferințe și istoricul comenzilor. Pachetul "pages" este destinat paginilor specifice ale aplicației, facilitând întreținerea și actualizarea conținutului static.

Fiecare dintre aceste pachete este meticolos proiectat pentru a izola logica specifică și pentru a sprijini o dezvoltare modulară, ceea ce îmbunătățește claritatea codului și permite o dezvoltare rapidă și eficientă. Această structură modulară contribuie la o mai bună organizare și la facilitarea evoluției continue a aplicației.

5 Component and Deployment Diagram

5.1 Component Diagram

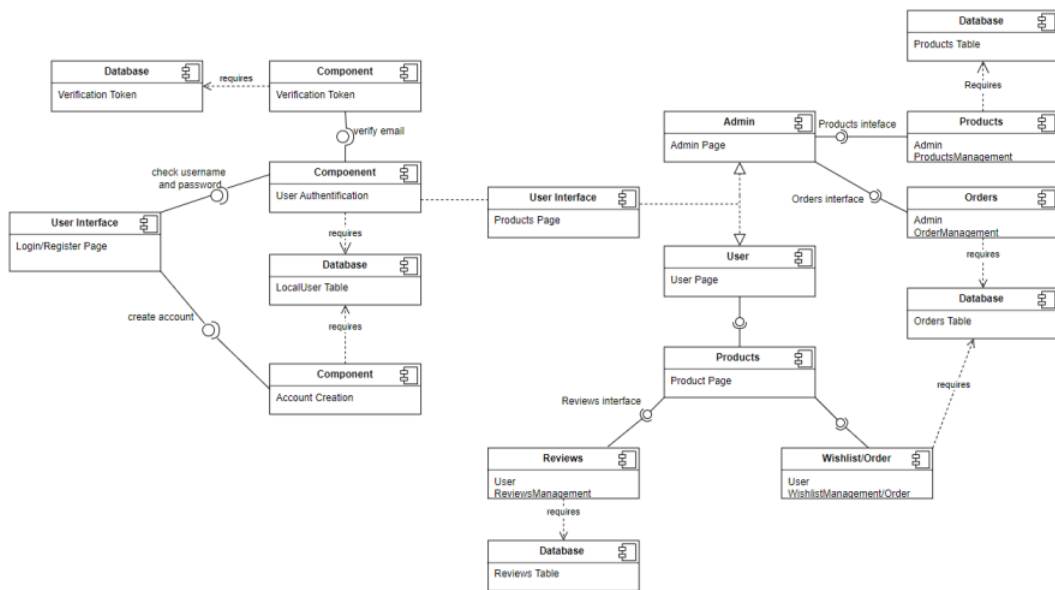
Această diagramă ilustrează arhitectura unui sistem software prin intermediul unei diagrame de componente, ce oferă o vedere structurală asupra diferitelor părți ale sistemului și a relațiilor dintre acestea. Diagrama este împărțită în mai multe sectoare ce reprezintă diferite aspecte ale sistemului.

În partea stângă a diagramei, avem componentele legate de interfața utilizatorului și autentificarea. Există o interfață pentru "Login/Register Page", unde utilizatorii își pot crea un cont sau se pot autentifica, verificând numele de utilizator și parola. Această interfață interacționează cu două componente: "User Authentication", care gestionează procesul de autentificare și "Account Creation", care gestionează crearea de noi conturi. Ambele componente necesită acces la baza de date, în special la tabelele "LocalUser Table" pentru stocarea informațiilor despre utilizatori și "Verification Token" pentru procesul de verificare a emailurilor utilizatorilor.

Centrul diagramei prezintă interfața "User Page", care permite utilizatorilor să acceseze și să interacționeze cu diferitele caracteristici ale site-ului, inclusiv paginile de produse și recenzii. "Product Page" este conectată la o bază de date care stochează informațiile despre produse ("Products Table") și recenzii ("Reviews Table"). De asemenea, există componente separate pentru "Reviews Management" și "User Wishlist/Management/Order", indicând că sistemul are capabilități de gestionare a recenziilor și a listelor de dorințe sau comenzi.

În partea dreaptă a diagramei, sunt prezentate componentele "Admin Page", "Products Interface" și "Orders Interface", indicând că există o separație între interfețele pentru utilizatorii obișnuiți și administrație. "Admin ProductsManagement" și "Admin OrderManagement" sunt componente care permit administrației să gestioneze produsele și comenzile, cu fiecare componentă necesitând acces la baza de date corespunzătoare.

Fiecare componentă este legată printr-o serie de relații care indică dependențe, precum și direcția de flux a datelor sau a controlului între acestea. Săgețile cu etichete precum "requires" sugerează că o componentă necesită date sau funcționalități de la o altă componentă sau bază de date pentru a funcționa corect. În esență, diagrama descrie cum se împart responsabilitățile între diferitele părți ale sistemului și cum acestea colaborează pentru a oferi o experiență completă utilizatorilor și administrației.



5.2 Deployment Diagram

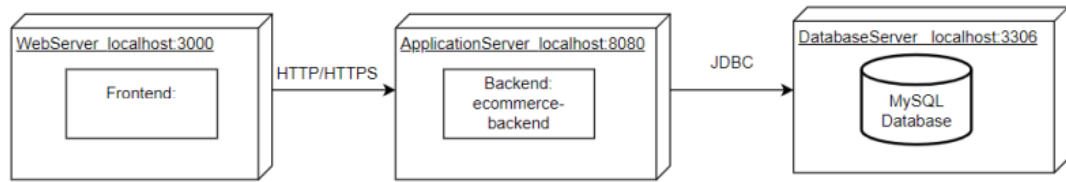
Descrierea prezentată ilustrează o diagramă de desfășurare pentru un sistem de e-commerce structurat în trei componente principale: un server web, un server de aplicații și un server de baze de date. Fiecare componentă are un rol specific în funcționarea sistemului.

Serverul web funcționează pe adresa locală localhost:3000 și găzduiește interfața utilizatorului, cunoscută sub numele de "Altex". Acest server web servește fișierele statice necesare (HTML, CSS, JavaScript) și este punctul de intrare pentru utilizatori, care accesează aplicația prin browser folosind protocoalele HTTP sau HTTPS. Portul 3000 este des utilizat pentru dezvoltarea locală a aplicațiilor web, permițând o testare și depanare eficientă.

Serverul de aplicații, care rulează pe localhost:8080, este responsabil pentru gestionarea logicii de afaceri a sistemului, denumită "ecommerce-backend". Acesta prelucrează datele și răspunde la solicitările venite din partea frontend-ului. Portul 8080 este frecvent folosit pentru servere de aplicații sau servicii web în contextul dezvoltării locale, facilitând conectivitatea și testarea funcționalităților backend.

Serverul de baze de date este situat pe localhost:3306 și găzduiește o bază de date MySQL, un sistem de gestionare a bazelor de date relaționale. Acest server de baze de date este conectat la serverul de aplicații prin JDBC (Java Database Connectivity), facilitând execuția operațiilor pe baza de date. Portul 3306 este standard pentru MySQL, oferind un punct de acces familiar pentru administrarea bazei de date.

Fluxul de date între aceste componente începe când utilizatorii interacționează cu interfața "Altex" în browser, trimițând solicitări HTTP/HTTPS către serverul de aplicații. Serverul de aplicații procesează aceste solicitări, efectuând acțiuni precum interogarea bazei de date sau executarea logicii de afaceri. Baza de date răspunde la interogări și trimite datele înapoi la serverul de aplicații, care apoi transmite răspunsurile corespunzătoare înapoi la frontend pentru a fi afișate utilizatorului. Această arhitectură tripartită oferă o separație clară a responsabilităților, îmbunătățind securitatea, scalabilitatea și mentenanța sistemului.



6 Dynamic Behavior

6.1 Diagrama de Secvență LOGIN:

Diagrama de secvență evidențiază secvența de acțiuni între actorii implicați în procesul de autentificare.

Participanți: Utilizatorul si Sistemul

Mesaje:

- Utilizatorul trimite o cerere HTTP GET către sistem pentru autentificare, furnizând adresa de e-mail și parola.

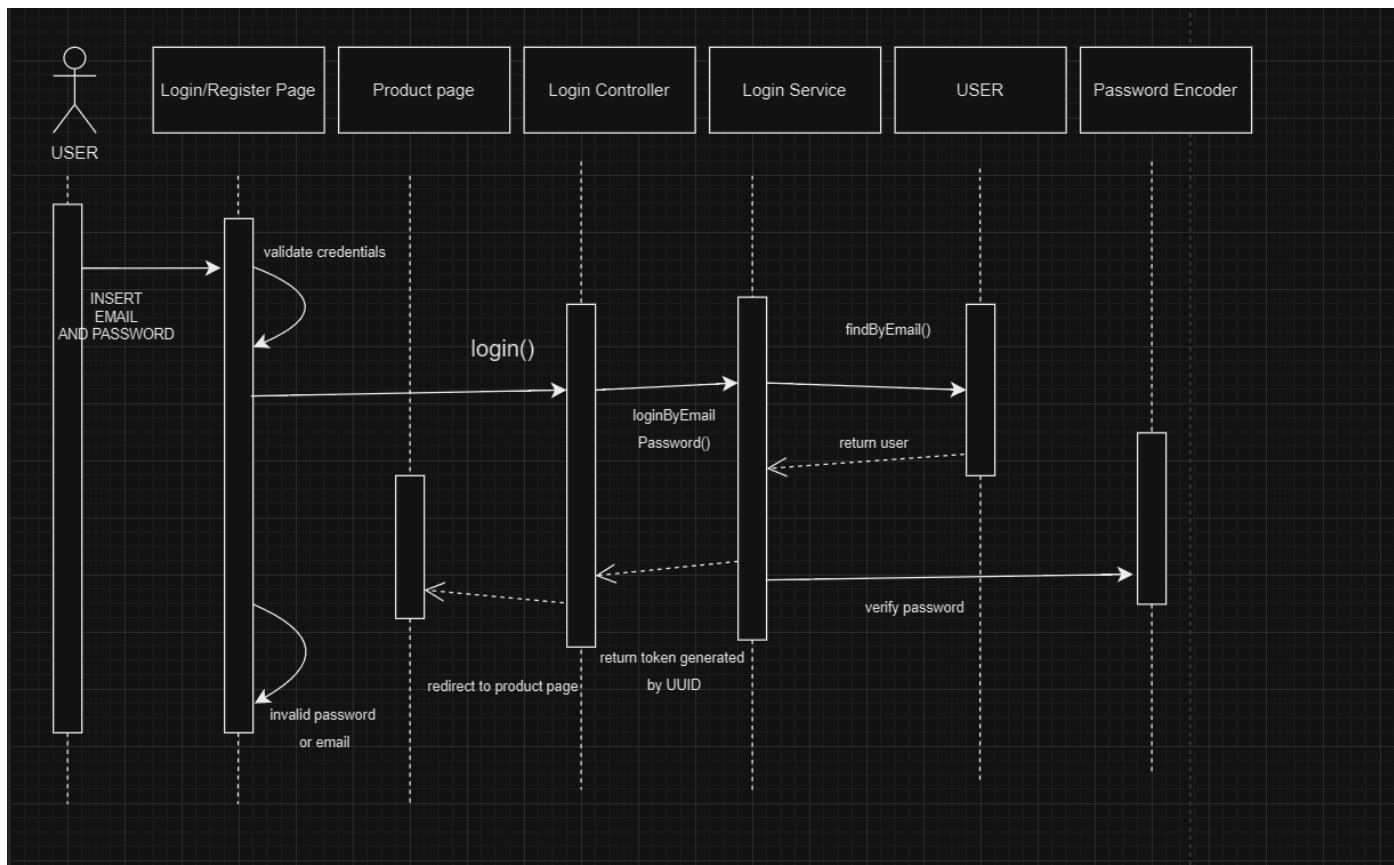
- Sistemul primește cererea și extrage parametrii (adresa de e-mail și parola).

- Sistemul caută în baza de date persoana asociată cu adresa de e-mail furnizată.

- Sistemul verifică parola furnizată cu parola asociată persoanei găsite în baza de date.

- Sistemul returnează un răspuns HTTP corespunzător rezultatului verificării.

Fragmente de Viață: Verificarea existenței persoanei în baza de date si Verificarea corespondenței parolelor.



6.2 Diagrama de Comunicare LOGIN:

Diagrama de comunicare ilustrează interacțiunea între obiectele implicate în procesul de autentificare.

Obiecte:

Controller-ul pentru autentificare

Repository pentru persoane (PersonRepository)

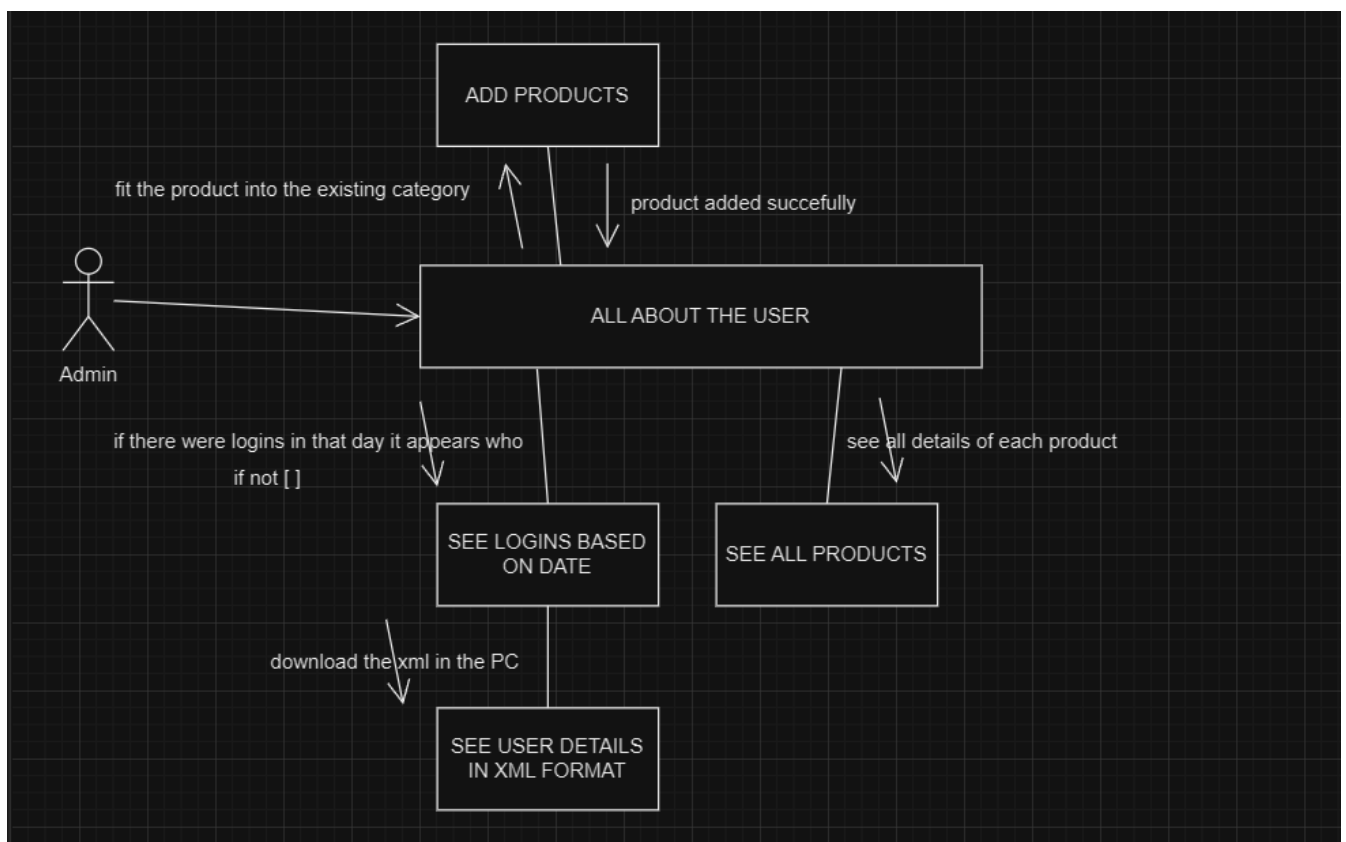
Mesaje:

Cererea HTTP GET trimisă către controller-ul pentru autentificare.

Interogarea bazei de date pentru a găsi persoana asociată cu adresa de e-mail.

Verificarea parolei.

Răspunsurile HTTP întoarse către client.



6.3 Diagrama de Secvență ADD PRODUCT:

Participanți: Administrator, Utilizator, Server

Mesaje:

Administratorul inițiază procesul de adăugare a unui produs prin interacțiunea cu interfața utilizatorului administrator.

Controlerul pentru produse primește cererea de adăugare a produsului de la administrator.

Controlerul apelează metoda corespunzătoare din serviciul de produse (ProductService) pentru a adăuga produsul în baza de date.

Serviciul de produse interacționează cu repository-ul de produse pentru a efectua operația de adăugare a produsului în baza de date.

Produsul este adăugat în baza de date de către repository-ul de produse.

Un utilizator vizualizează lista de produse prin interacțiunea cu interfața utilizatorului (UI).

Controlerul pentru produse primește cererea de afișare a listei de produse de la utilizator.

Controlerul solicită serviciului de produse (ProductService) să furnizeze lista de produse disponibile.

Serviciul de produse interoghează repository-ul de produse pentru a obține lista de produse.

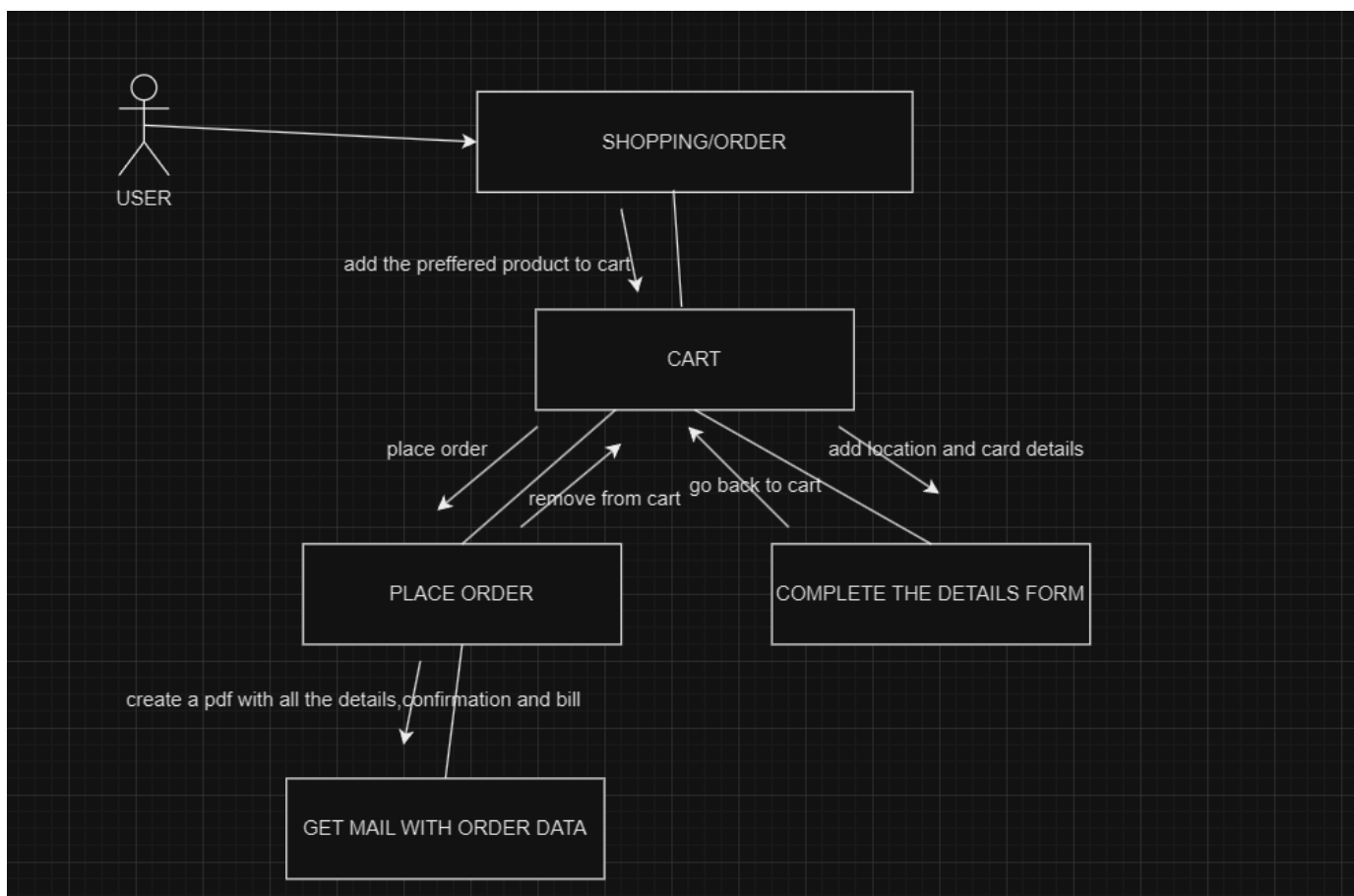
Lista de produse este returnată de repository-ul de produse către serviciul de produse și apoi către controler.

Controlerul furnizează lista de produse către interfața utilizatorului, unde aceasta este afișată pentru utilizator.

Fragmente de Viață:

Validarea și procesarea datelor introduse de administrator înainte de adăugarea produsului în baza de date.

Interogarea și extragerea listei de produse din baza de date pentru afișare.



6.4 Diagrama de Comunicare ORDER:

Participanți: Admin, Server

Mesaje:

Utilizatorul trimite detaliile comenzii (produsele selectate, informații de plată și livrare) către Sistemul de Comandă.

Sistemul de Comandă validează detaliile comenzii și trimite cererea către Serviciul de Generare a PDF-ului pentru a genera documentul PDF al comenzii.

Serviciul de Generare a PDF-ului generează documentul PDF și îl trimite înapoi către Sistemul de Comandă. vSistemul de Comandă trimite cererea către Serviciul de Trimitere Email pentru a trimite documentul PDF către utilizator prin email.

Serviciul de Trimitere Email trimite documentul PDF către utilizator.

Sistemul de Comandă trimite cererea către Serviciul de Procesare a Plăților pentru a procesa plata comenzii.

Serviciul de Procesare a Plăților procesează plata comenzii și trimite confirmarea înapoi către Sistemul de Comandă.

Sistemul de Comandă trimite confirmarea comenzii plasate cu succes către Admin.

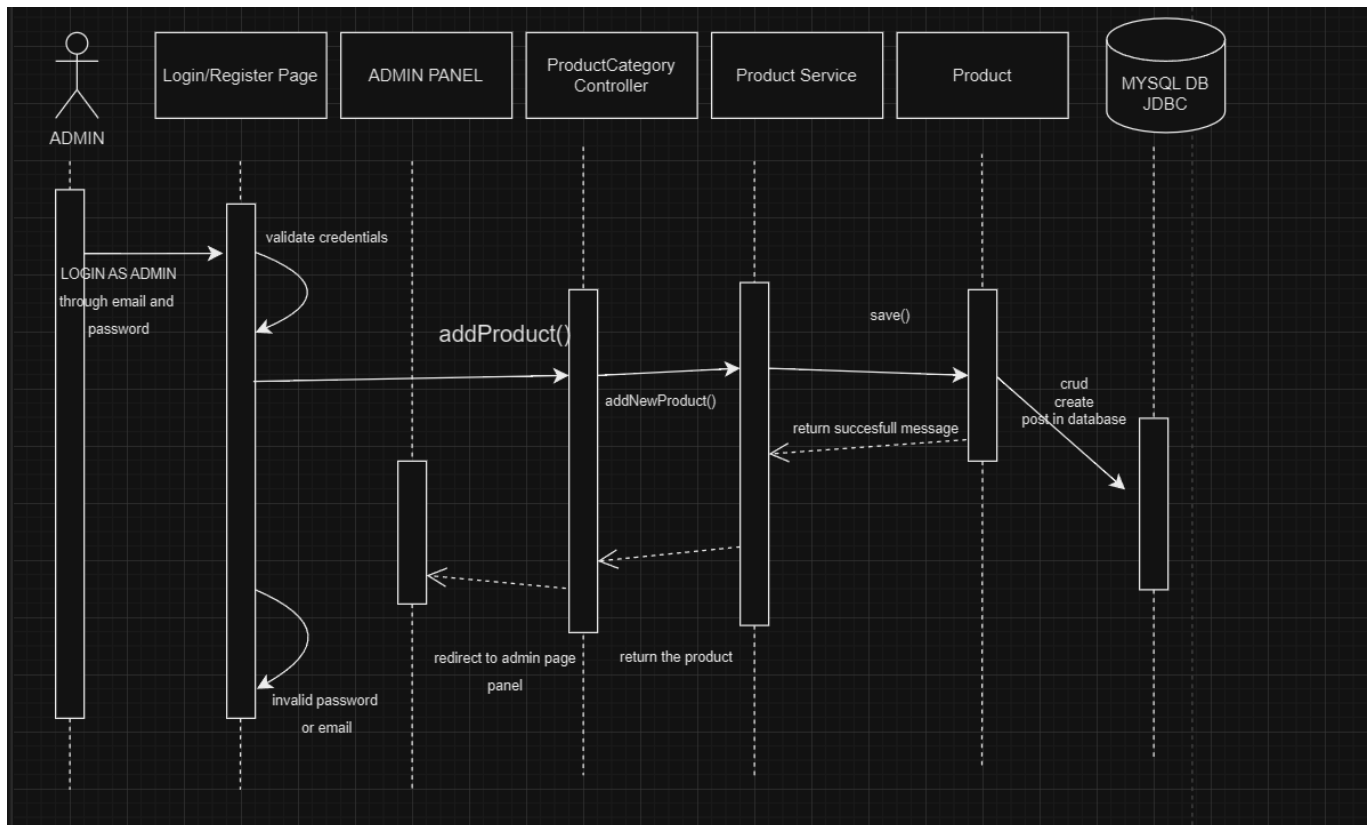
Fragmente de Viață:

Validarea detaliilor comenzii înainte de generarea PDF-ului.

Generarea documentului PDF cu detaliile comenzii.

Trimiterea emailului către utilizator cu documentul PDF atașat.

Procesarea plății comenzii.



7 Use case UML

Diagrama de cazuri de utilizare (Use Case Diagram) în limbajul de modelare UML (Unified Modeling Language) este o tehnică utilizată în ingineria software pentru a defini și a prezenta interacțiunile dintre actori (utilizatori externi) și sistem. Aceasta oferă o perspectivă structurală asupra funcționalităților oferite de sistem și modul în care acestea sunt accesate de către actori.

Un caz de utilizare reprezintă 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 sunt entitățile externe care interacționează cu sistemul, fie că 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 atinge anumite obiective.

7.1 Use case USER

Use-Case: Gestionarea contului și vizualizarea produselor

Level: Principal (core)

Primary Actor: Utilizatorul (USER)

Main success scenario:

Utilizatorul dorește să își creeze un cont:

- Actorul accesează funcția de "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:

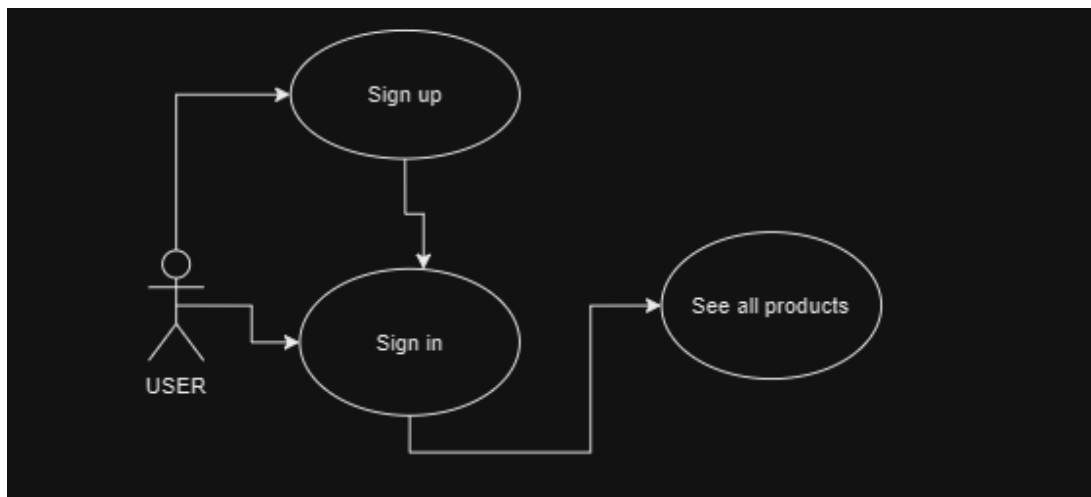
- Actorul accesează funcția de "Sign in".
- Sistemul solicită introducerea adresei de email și a parolei.
- Utilizatorul furnizează detaliile de autentificare.
- Sistemul validează datele și autentifică utilizatorul.
- Utilizatorul primește confirmarea că autentificarea a fost efectuată cu succes.

După autentificare, utilizatorul dorește să vizualizeze toate produsele disponibile:

- Actorul accesează funcția de "See all products".
- Sistemul afișează o listă cu toate produsele disponibile în magazin.

Extensions:

- 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.
- Dacă utilizatorul autentificat nu are permisiunea de a vizualiza produsele (de exemplu, dacă contul său este suspendat sau închis), sistemul îl va redirecționa către o pagină corespunzătoare de eroare sau îl va informa că nu are acces la această funcționalitate.



7.2 Use case ADMIN

Use-Case: Gestionarea funcționalităților ca administrator

Level: Principal (core)

Primary Actor: Administratorul (ADMIN)

Main success scenario:

Administratorul dorește să se autentifice în sistem:

- Actorul accesează funcția de "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 trei opțiuni disponibile:

a) Adăugare de produse noi:

- Administratorul accesează funcția "Add Products".
- 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 a tuturor utilizatorilor:

- Administratorul accesează funcția "See All Users".
- Sistemul afișează o listă cu toți utilizatorii înregistrați în sistem.

c) Vizualizare a tuturor produselor:

- Administratorul accesează funcția "See All Products".
- Sistemul afișează o listă cu toate produsele disponibile în magazin.

După vizualizarea tuturor utilizatorilor, administratorul are trei opțiuni suplimentare:

a) Editare utilizator:

- Administratorul selectează opțiunea "Edit User" pentru un anumit utilizator.
- Sistemul permite administratorului să modifice informațiile utilizatorului (ex: nume, email, rol, etc.).
- Administratorul finalizează modificările, iar sistemul validează și actualizează datele utilizatorului în baza de date.

b) Vizualizare detalii utilizator:

- Administratorul selectează opțiunea "View User" pentru a vizualiza detaliile unui anumit utilizator (ex: nume, email, istoric de achiziții, etc.).

c) Ștergere utilizator:

- Administratorul selectează opțiunea "Delete User" pentru a șterge un anumit utilizator.
- Sistemul solicită confirmarea administratorului.
- După confirmare, utilizatorul este șters din baza de date.

După vizualizarea tuturor produselor, administratorul are trei opțiuni suplimentare:

a) Editare produs:

- Administratorul selectează opțiunea "Edit Product" pentru un anumit produs.
- Sistemul permite administratorului să modifice informațiile produsului (ex: nume, descriere, preț, etc.).
- Administratorul finalizează modificările, iar sistemul validează și actualizează datele produsului în baza de date.

b) Vizualizare detalii produs:

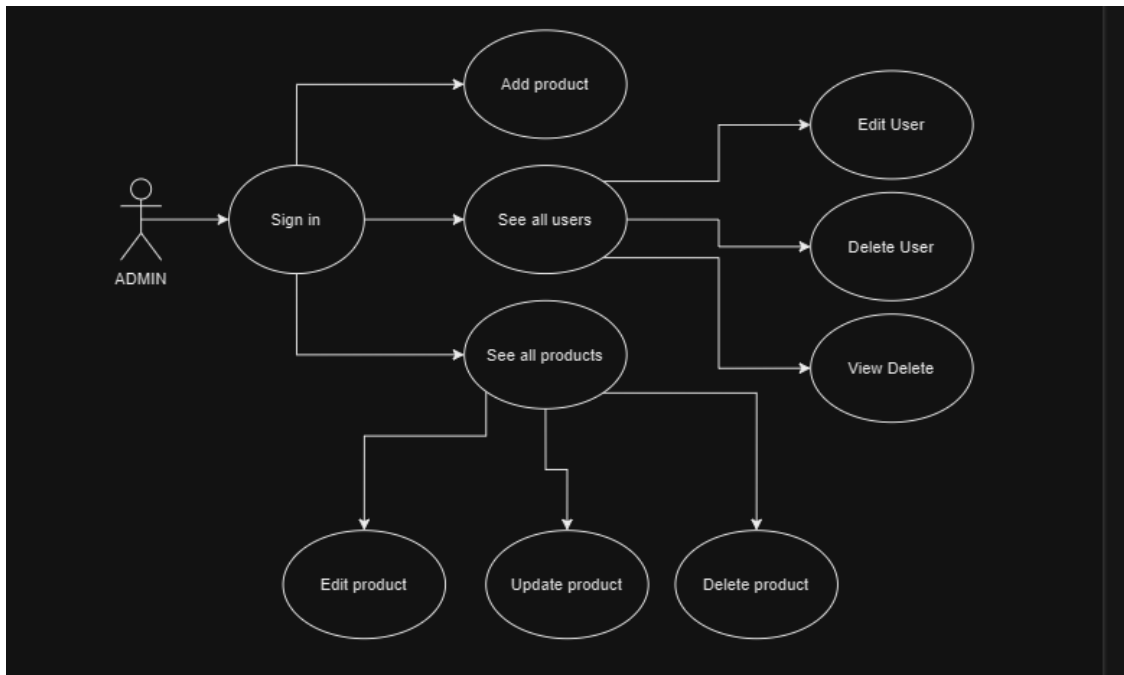
- Administratorul selectează opțiunea "View Product" pentru a vizualiza detaliile unui anumit produs (ex: nume, descriere, stoc disponibil, etc.).

c) Ștergere produs:

- Administratorul selectează opțiunea "Delete Product" pentru a șterge un anumit produs.
- Sistemul solicită confirmarea administratorului.
- După confirmare, produsul este șters din baza de date.

Extensions:

- 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 suplimentară a administratorului pentru a preveni ștergerile accidentale sau neautorizate.



8 Class Diagram

MVC (Model-View-Controller) Design Pattern:

Descriere:

MVC este un model de proiectare utilizat în dezvoltarea software pentru separarea logică între componentele unei aplicații. Împarte o aplicație în trei componente principale: Model, View și Controller.

Model: Reprezintă datele și logica de afaceri a aplicației. Modelul este responsabil pentru manipularea datelor, precum și pentru validare și procesarea acestora conform cerințelor aplicației. Este independent de interfața utilizatorului.

View: Reprezintă interfața utilizatorului și afișează informațiile utilizatorului. Acesta interacționează cu utilizatorul și trimite cereri către controller pentru a obține sau manipula datele.

Controller: Acționează ca intermediar între Model și View. Primește cererile de la View, procesează logica de afaceri utilizând Modelul corespunzător și returnează rezultatele înapoi la View pentru afișare. Controllerul gestionează fluxul de date și interacțiunile utilizatorului.

Avantaje:

Separarea responsabilităților: MVC permite separarea clară a responsabilităților între componentele aplicației, ceea ce facilitează dezvoltarea și întreținerea codului.

Reutilizare și testare ușoară: Componentele sunt independente și pot fi reutilizate sau testate separat, ceea ce duce la o dezvoltare mai eficientă și la un cod mai curat.

Dezavantaje:

Complexitate suplimentară: Implementarea MVC poate aduce o complexitate suplimentară, în special pentru aplicațiile mici sau simple.

Creșterea numărului de fișiere și clase: Divizarea aplicației în trei componente separate poate duce la o creștere a numărului de fișiere și clase, ceea ce poate complica gestionarea proiectului.

JpaRepository Pattern:

Descriere:

JpaRepository este un pattern folosit în dezvoltarea aplicațiilor Java Spring pentru gestionarea operațiilor de bază de date. Acesta oferă o interfață simplificată pentru a efectua operații CRUD (Create, Read, Update, Delete) pe entități.

JpaRepository Interface: Este o interfață furnizată de Spring Data JPA care oferă metode predefinite pentru a efectua operațiile CRUD pe entități. Aceste metode includ salvare, actualizare, ștergere și căutare.

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 utilizând JpaRepository.

Spring Data JPA: Este un modul din Spring Framework care facilitează dezvoltarea aplicațiilor JPA (Java Persistence API). Spring Data JPA oferă funcționalități suplimentare pentru a simplifica interacțiunea cu baza de date, inclusiv 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 a efectua operațiile de bază de date, ceea ce permite dezvoltatorilor să se concentreze pe logica de afaceri a aplicației.

Dezavantaje:

Limitări ale abstracției: JpaRepository oferă doar operații de bază de date și nu este întotdeauna adecvat pentru cerințele avansate de gestionare a datelor.

Dependența de JPA: Utilizarea JpaRepository implică o dependență de JPA și Hibernate, ceea ce poate duce la complexitate și dependențe suplimentare în aplicație.

Concluzie:

MVC și JpaRepository sunt două pattern-uri importante utilizate în dezvoltarea aplicațiilor moderne. MVC facilitează separarea clară a responsabilităților între componentele unei aplicații, în timp ce JpaRepository simplifică gestionarea operațiilor de bază de date în aplicațiile Java Spring. Utilizarea acestor pattern-uri poate îmbunătăți calitatea, structura și eficiența dezvoltării software.

Operațiile CRUD (Create, Read, Update, Delete) reprezintă un set de operații fundamentale utilizate în gestionarea datelor în cadrul sistemelor software. Fiecare operație are un rol distinct în manipularea informațiilor.

Operația de creare (Create) este gestionată prin intermediul metodei newPerson, asociată cu ruta POST /person. Această metodă primește un obiect Person sub formă de corp de solicitare și îl salvează în baza de date utilizând personRepository.save(newPerson). Înainte de salvare, sunt efectuate și alte operații specifice, precum gestionarea rolurilor și trimiterea unor informații către alte entități.

Operația de citire (Read) este implementată în două metode distincte. Prima metodă, getAllPersons, asociată cu ruta GET /persons, returnează o listă care conține toate persoanele din baza de date. A doua metodă, getPersonById, asociată cu ruta GET /person/id, furnizează detaliile unei persoane specifice, identificate prin intermediul ID-ului său unic.

Operația de actualizare (Update) este gestionată de metoda updatePerson, asociată cu ruta PUT /user/id. Această metodă primește un obiect Person actualizat sub formă de corp de solicitare, împreună cu ID-ul persoanei care urmează să fie actualizată. Persoana este identificată în baza de date, iar dacă este găsită, se actualizează câmpurile relevante cu noile valori și se salvează modificările.

Operația de ștergere (Delete) este gestionată de metoda deletePerson, asociată cu ruta DELETE /person/id. Această metodă primește ID-ul persoanei care urmează să fie ștearsă. Înainte

de a efectua ștergerea, se verifică dacă persoana există în baza de date. În cazul în care persoana nu este găsită, se aruncă o excepție `UserNotFoundException`, altfel persoana este ștearsă din baza de date.

Aceste operații CRUD furnizează funcționalitățile de bază necesare pentru gestionarea persoanelor în cadrul sistemului de comerț electronic, permitând crearea, citirea, actualizarea și ștergerea acestora.

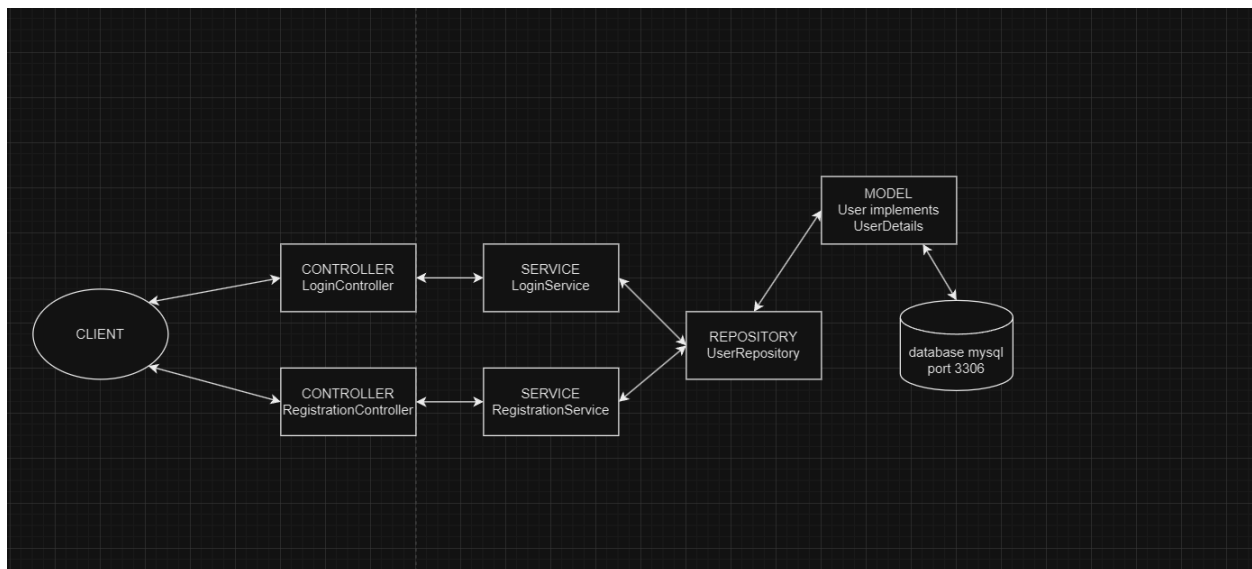
Operația de creare (Create) este gestionată prin intermediul metodei `createProduct`, asociată cu ruta `POST /product`. Această metodă primește un obiect `Product` sub formă de corp de solicitare și îl salvează în baza de date utilizând `productRepository.save(product)`. În timpul procesului de creare a produsului, se verifică și se adaugă categoriile corespunzătoare în cazul în care acestea nu există deja.

Pentru operația de citire (Read), este furnizată metoda `getAllProducts`, asociată cu ruta `GET /products`, care returnează o listă ce conține toate produsele din baza de date. De asemenea, există și metoda `getProductNameById`, asociată cu ruta `GET /product/id`, care returnează numele unui produs specific identificat după ID-ul său.

Operația de actualizare (Update) este gestionată de metoda `updateProductName`, asociată cu ruta `PUT /product/id`. Această metodă primește un ID de produs și un obiect `Product` actualizat sub formă de corp de solicitare. Noul nume al produsului este actualizat, iar modificările sunt salvate în baza de date.

Operația de ștergere (Delete) este implementată prin metoda `deleteProduct`, asociată cu ruta `DELETE /product/id`. Această metodă primește ID-ul produsului care urmează să fie șters. Înainte de ștergere, toate referințele către produs în tabela `productcategory` sunt eliminate, iar apoi produsul însuși este șters din baza de date.

Aceste operații CRUD oferă funcționalitățile necesare pentru gestionarea produselor și categoriilor în cadrul sistemului de comerț electronic, inclusiv crearea, citirea, actualizarea și ștergerea acestora.



9 Testing

Unit Testing și Adnotări:

Testarea unitară este o metodă de testare a celor mai mici componente ale unei aplicații, cum ar fi metodele individuale sau clasele. Scopul principal al testelor unitare este de a verifica corectitudinea logicii unei componente software și de a detecta erori cât mai devreme în procesul de dezvoltare.

Adnotările sunt folosite pentru a marca anumite metode sau clase în timpul testării unitare. În exemplul furnizat, se folosesc adnotări precum `@BeforeEach` pentru a marca metoda care este executată înainte de fiecare test, `@Test` pentru a marca metodele care sunt teste efective și `@Mock` pentru a marca obiectele simulări folosite în testare.

Construirea unui Test:

Inițializare și Pregătire: Înainte de a scrie teste, este important să inițializăm toate obiectele și resursele necesare. În exemplul dat, aceasta se face în metoda `setUp()` folosind `MockitoAnnotations.openMocks(this)` pentru a inițializa obiectele mock.

Definirea Scenariului de Test: Pentru fiecare metodă pe care dorim să o testăm, trebuie să definim scenariul pe care îl vom testa. Acest lucru include setarea parametrilor de intrare, definirea așteptărilor și a comportamentului dorit al metodei testate.

Execuția Testului: Folosind obiectele și resursele pregătite anterior, rulăm metoda pe care dorim să o testăm și obținem rezultatul.

Validarea Rezultatului: După ce metoda a fost rulată, trebuie să validăm rezultatul pentru a ne asigura că se comportă așa cum ne așteptăm. Aceasta se face folosind aserțiuni (assertions) pentru a verifica că rezultatul este în concordanță cu așteptările noastre.

Curățarea și Eliberarea Resurselor: După ce testul a fost rulat, este important să curățăm și să eliberăm resursele utilizate, astfel încât să nu interferăm cu alte teste sau să lăsăm resursele neutilizate.

Importanța Testării Unitare:

Testarea unitară este o practică esențială în dezvoltarea software-ului, deoarece oferă o modalitate eficientă de a verifica funcționalitatea codului și de a preveni introducerea de erori în aplicație. Testele unitare pot fi automate și rulate rapid, ceea ce permite dezvoltatorilor să detecteze și să remedieze erorile într-un stadiu incipient al dezvoltării, ceea ce duce la un cod mai fiabil și mai ușor de întreținut.

10 Specificații suplimentare

10.1 Cerințe non-funcționale

10.1.1 Performanță

Performanța este esențială pentru un magazin online, deoarece utilizatorii așteaptă o experiență rapidă și fără întârzieri. Astfel, avem nevoie de o cerință non-funcțională care să asigure că timpul de răspuns al aplicației noastre este scăzut, iar paginile se încarcă rapid atunci când sunt accesate de către utilizatori. Acest lucru este crucial pentru a menține angajamentul utilizatorilor și pentru a crește conversiile.

10.1.2 Scalabilitate

Un magazin online trebuie să poată gestiona o creștere bruscă a traficului și a volumului de date, în special în timpul perioadelor de vârf, cum ar fi sărbătorile sau campaniile de reduceri. Prin urmare, avem nevoie de o cerință non-funcțională care să asigure că aplicația noastră este scalabilă și poate fi extinsă fără probleme pentru a gestiona cerințele crescânde ale utilizatorilor.

Aceasta poate implica utilizarea serviciilor cloud pentru a crește resursele de calcul și stocare în mod dinamic în funcție de necesități.

10.1.3 Fiabilitatea

Fiabilitatea este esențială pentru un magazin online, deoarece orice indisponibilitate sau cădere a sistemului poate duce la pierderi de vânzări și la scăderea încrederii utilizatorilor. Avem nevoie de cerințe non-funcționale care să asigure că aplicația noastră este stabilă și că poate funcționa fără probleme pe termen lung. Implementarea redundanței în infrastructură și a mecanismelor de backup poate contribui la asigurarea unei funcționări fără probleme a sistemului.

10.1.4 Uzabilitate

Uzabilitatea este o altă cerință non-funcțională importantă pentru un magazin online, deoarece utilizatorii trebuie să poată naviga ușor în site și să găsească rapid produsele dorite. Avem nevoie de o interfață utilizator intuitivă și prietenoasă, care să ofere o experiență plăcută utilizatorilor și să îi încurajeze să revină pentru achiziții ulterioare. Testele de utilizabilitate și feedback-ul utilizatorilor pot fi folosite pentru a îmbunătăți continuu experiența utilizatorului și pentru a face ajustări în funcție de nevoile și preferințele acestora.

10.2 Design constraints

Există anumite constrângeri de design care au fost mandate și trebuie respectate în implementarea sistemului. Aceste constrângeri includ alegeri legate de limbajul de programare, procesele software, uneltele de dezvoltare, arhitectura și altele.

Pentru partea de backend a aplicației noastre, am ales să utilizăm Java Spring în cadrul mediului de dezvoltare IntelliJ IDEA. Java Spring oferă o platformă robustă și scalabilă pentru dezvoltarea aplicațiilor web, iar IntelliJ IDEA este un mediu de dezvoltare puternic și popular, care oferă unelte avansate pentru dezvoltarea și testarea aplicațiilor Java.

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

Pentru testarea și validarea funcționalităților backend-ului, folosim Postman, o platformă de testare API-uri care ne permite să testăm și să debugăm endpoint-urile din controllerele noastre într-un mod eficient și intuitiv. Postman oferă funcționalități avansate pentru testarea automatizată, monitorizarea și documentarea API-urilor 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.

Pentru a completa dezvoltarea frontend-ului, am instalat pachetele Axios, Bootstrap și react-router-dom. Axios este o librărie pentru efectuarea de cereri HTTP din JavaScript, Bootstrap este un framework CSS popular pentru stilizarea interfeței de utilizator, iar react-router-dom este o bibliotecă pentru gestionarea rutelor și navigarea în aplicația noastră React.

În ceea ce privește arhitectura aplicației noastre, am ales să urmărim modelul de arhitectură Model-View-Controller (MVC), care ne permite să separăm logic interfața utilizatorului, logica

de afaceri și logica de gestionare a datelor în straturi distincte și bine definite. Această abordare ne ajută să creăm aplicații modulare, ușor de întreținut și scalabile.

11 Future Improvements

Există numeroase îmbunătățiri și caracteristici suplimentare care pot fi implementate pentru a îmbunătăți experiența utilizatorului, eficiența și gestionarea memoriei. Iată o prezentare detaliată a acestor îmbunătățiri:

1. Eficiența și Gestionarea Memoriei:

Optimizarea Interacțiunilor Cu Baza de Date: Utilizarea de interogări eficiente și indexare adecvată pentru a îmbunătăți timpul de răspuns al aplicației și a reduce utilizarea resurselor de memorie.

Cache pentru Date Frecvent Accesate: Implementarea unui mecanism de cache pentru datele frecvent accesate poate reduce timpul de acces la baza de date și poate îmbunătăți performanța aplicației.

Gestionarea Memoriei pe Frontend: Utilizarea tehnologiei precum React cu o gestionare eficientă a stării și eliminarea ciclului de viață neutilizat pot reduce consumul de memorie pe frontend.

2. Funcționalități Adicionale:

Dublă Securitate prin SMS: Implementarea unei opțiuni de autentificare în doi pași prin SMS poate crește securitatea contului utilizatorilor și poate proteja împotriva accesului neautorizat.

Coduri de Reducere și Promovări: Adăugarea unui sistem de coduri de reducere pentru a oferi clienților reduceri speciale și a încuraja mai multe achiziții.

Sistem de Returnare a Produselor: Dezvoltarea unei funcționalități de gestionare a returnării produselor pentru a permite clienților să returneze produsele și să primească rambursări sau schimburi într-un mod simplu și eficient.

Implementarea în Contextul Aplicației Altex Ecommerce:

Dublă Securitate prin SMS: După ce utilizatorii își creează conturi pe platformă, aceștia pot opta să activeze autentificarea în doi pași prin SMS. Atunci când își încearcă să se autentifice, ei vor primi un cod unic prin SMS care trebuie introdus pentru a finaliza autentificarea.

Coduri de Reducere și Promovări: Altex poate organiza campanii periodice de promovare și vânzare prin intermediul codurilor de reducere. Utilizatorii pot introduce aceste coduri în timpul finalizării comenzii pentru a primi reduceri speciale sau alte beneficii.

Sistem de Returnare a Produselor: Pentru a asigura satisfacția clienților, Altex poate implementa un sistem de returnare a produselor care să permită clienților să returneze produsele într-o anumită perioadă de timp și să primească rambursări sau schimburi într-un mod simplu și eficient.

Aceste îmbunătățiri și caracteristici suplimentare pot contribui semnificativ la creșterea performanței, securității și satisfacției utilizatorilor pe platforma Altex Ecommerce. Prin abordarea lor detaliată și planificată, aceste îmbunătățiri pot consolida poziția Altex pe piața de comerț electronic și pot atrage mai mulți clienți.

12 Conclusion

În concluzie, prin utilizarea unei suite variate de tehnologii și funcționalități, platforma Altex Ecommerce a fost capabilă 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.

Pentru gestionarea și trimiterea de emailuri, s-a folosit Maildev, asigurându-se că comunicarea cu utilizatorii este eficientă și fiabilă. Backend-ul, construit pe baza framework-ului Spring, a furnizat o bază solidă pentru gestionarea datelor și a funcționalităților complexe ale aplicației.

Baza de date MySQL 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 Altex Ecommerce. 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 Altex.

13 Bibliography

React

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

Spring

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

Unit Testing

3. "JUnit 5 User Guide." Disponibil online: <https://junit.org/junit5/docs/current/user-guide/>

WebSockets

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

Arhitectura MVC

5. <https://www.geeksforgeeks.org/mvc-design-pattern/>

MySQL

6. "MySQL Documentation." Disponibil online: <https://dev.mysql.com/doc/>

Maildev

7. "Maildev." GitHub Repository. Disponibil online: <https://github.com/maildev/maildev>