



---

**UNIVERSITATEA TEHNICĂ**  
DIN CLUJ-NAPOCA  
CENTRUL UNIVERSITAR NORD DIN BAIA MARE  
FACULTATEA DE INGINERIE

**DEPARTAMENTUL DE INGINERIE ELECTRICĂ, ELECTRONICĂ ȘI CALCULATOARE**

**Programul de studii: CALCULATOARE**

**MAGAZIN ONLINE DE HAINE**

**PROIECT PROIECTAREA APLICAȚIILOR WEB**

Autor: **POP GIULIA ANA**

**2024**

# Cuprins

<b>1</b>	<b>INTRODUCERE.....</b>	<b>2</b>
1.1	CONTEXTUL GENERAL ȘI SCOPUL PROIECTULUI .....	2
1.2	OBIECTIVELE PROIECTULUI .....	2
1.3	LISTĂ MOSCOW .....	2
1.4	CAZURI DE UTILIZARE.....	3
<b>2</b>	<b>ARHITECTURĂ .....</b>	<b>4</b>
2.1	FRONTEND.....	4
2.2	BACKEND .....	4
2.3	COMUNICARE FRONTEND ȘI BACKEND .....	4
2.4	BAZA DE DATE .....	4
2.5	AVANTAJE/ DEZAVANTAJE ARHITECTURĂ .....	5
<b>3</b>	<b>IMPLEMENTARE FRONTEND .....</b>	<b>6</b>
3.1	COMPONENTE PRINCIPALE.....	6
3.2	INTEGRARE CU REDUX.....	7
3.3	STILIZARE .....	8
3.4	INTERACȚIUNEA CU BACKEND-UL .....	8
<b>4</b>	<b>IMPLEMENTARE BACKEND.....</b>	<b>8</b>
4.1	STUCTURA PROIECTULUI SPRING BOOT .....	8
4.2	PRIVIRE DE ANSAMBLU BACKEND .....	9
4.3	GESTIONAREA BAZEI DE DATE POSTGRESQL .....	10
<b>5</b>	<b>APLICAȚIA .....</b>	<b>11</b>
<b>6</b>	<b>CONCLUZII.....</b>	<b>14</b>
<b>7</b>	<b>DEZVOLTĂRI ULTERIOARE .....</b>	<b>14</b>
<b>8</b>	<b>BIBLIOGRAFIE.....</b>	<b>15</b>

# 1 INTRODUCERE

## 1.1 CONTEXTUL GENERAL ȘI SCOPUL PROIECTULUI

Proiectul "Shopper" reprezintă o aplicație web dedicată domeniului comerțului electronic, concentrându-se pe vânzarea de articole vestimentare.

Scopul principal al acestei aplicații este să ofere o platformă ușor de utilizat, care permite clienților să navigheze și să achiziționeze diverse produse vestimentare, precum jachete, bluze, sau pantaloni. Acest magazin online își propune să creeze o experiență plăcută de cumpărături pentru clienți, oferind produse variate și informații clare despre acestea

## 1.2 OBIECTIVELE PROIECTULUI

1. **Experiență de Cumpărături Fluidă:** Oferirea utilizatorilor o experiență de cumpărături online fluidă și intuitivă.
2. **Opțiuni pentru Categori:** Meniul de navigare să includă opțiuni clare pentru categoriile principale: "Femei," "Bărbați," și "Copii."
3. **Coș de Cumpărături:** Implementarea unui sistem de coș de cumpărături pentru a permite utilizatorilor să adauge produse în coș și să finalizeze achizițiile.
4. **Autentificare și Securitate:** Asigurarea unui sistem de autentificare sigur pentru utilizatori, protejând datele personale și oferindu-le o experiență de cumpărături personalizată.

## 1.3 LISTĂ MOSCOW

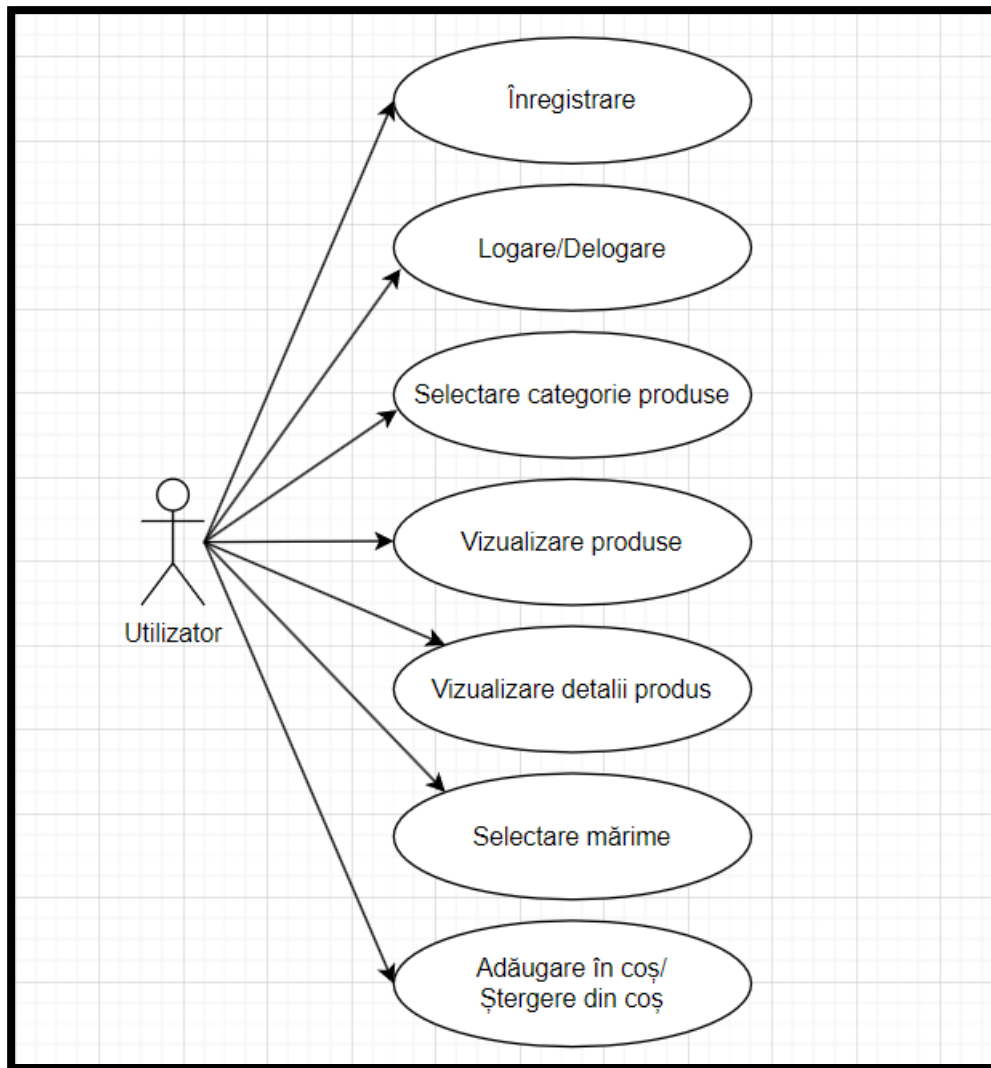
Pentru a defini prioritățile și a structura dezvoltarea aplicației "Shopper," am aplicat metodologia MOSCOW, evaluând cerințele în funcție de importanța lor.

Must Have	Should Have	Could Have	Won't Have
Înregistrare	Navigare intuitivă	Tranziții	Modalitate de plată
Login/Logout	Categorii produse		
Listă produse	Filtrare după mărimi		
Detalii produse	Vizualizare produse fără autentificare		
Coș de cumpărături			

Tabel 1 Lista MOSCOW

## 1.4 CAZURI DE UTILIZARE

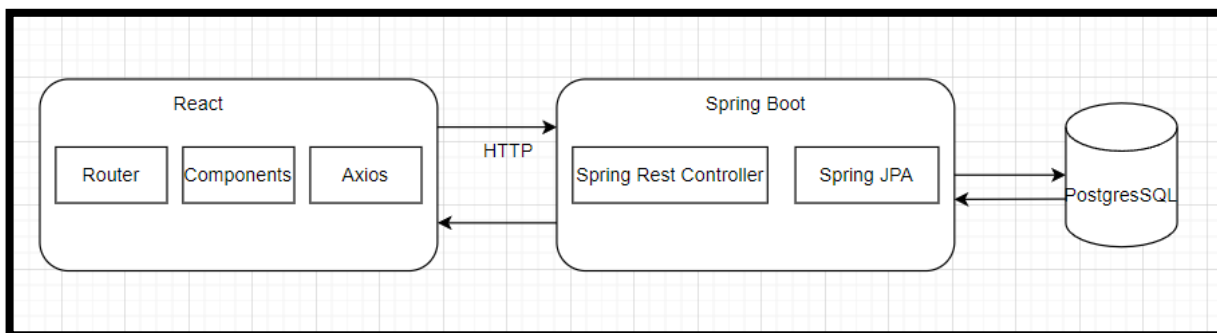
Această aplicație este destinată utilizatorilor care doresc să cumpere haine într-un mod cât mai ușor. În cadrul aplicației mele, am identificat și conturat diverse scenarii-cheie de utilizare, oferind o privire detaliată asupra interacțiunilor esențiale. Aceste cazuri de utilizare ilustrează modul în care utilizatorii se angajează cu aplicația, pentru a ne asigura că experiența lor este una memorabilă și eficientă. Aceste cazuri de utilizare sunt evidențiate în figura următoare:



Figură 1 Caz de utilizare

## 2 ARHITECTURĂ

Structura aplicației este gândită să ofere o performanță de înaltă calitate, să fie scalabilă și să aibă o interfață prietenoasă pentru utilizatori. Arhitectura este modulară și bine structurată, împărțită în trei componente principale: frontend, backend și baza de date.



### 2.1 FRONTEND

Frontend-ul aplicației "Shopper" este dezvoltat folosind o arhitectură bazată pe ReactJS, un framework modern pentru construirea interfețelor de utilizator. Structura proiectului este organizată în componente pentru a asigura modularitate și ușurință în dezvoltare. Aplicația frontend interacționează cu backend-ul prin intermediul API-urilor REST expuse.

### 2.2 BACKEND

Backend-ul aplicației "Shopper" utilizează Spring Boot, un cadru de dezvoltare Java pentru construirea de aplicații Java bazate pe microservicii. Structura proiectului este organizată în pachete care reflectă diferitele aspecte ale aplicației, precum configurații, controlere, servicii, modele, și repository-uri.

### 2.3 COMUNICARE FRONTEND ȘI BACKEND

Comunicarea dintre Frontend și Backend este realizată prin intermediul API-urilor REST. Backend-ul expune endpoint-uri care pot fi apelate de către frontend pentru a efectua diverse operații precum adăugarea produselor în coș, afișarea catalogului de produse, etc. Datele sunt transferate în format JSON.

### 2.4 BAZA DE DATE

Baza de date a proiectului "Shopper" este implementată folosind PostgreSQL, un sistem de gestionare a bazelor de date relaționale robust și open-source. PostgreSQL oferă o gamă

largă de funcționalități și beneficii, contribuind semnificativ la performanța și fiabilitatea aplicației.

### **Caracteristici cheie ale PostgreSQL:**

#### **Structură Relațională:**

- PostgreSQL utilizează un model relațional pentru organizarea datelor. Tabelele, relațiile și constrângerile asigură integritatea datelor și coerența relațională.

#### **Extensibilitate:**

- Suportă extensibilitatea prin adăugarea de funcționalități noi, cum ar fi tipuri de date personalizate, funcții și limbaje procedurale.

#### **Tranzacții și Conformitate ACID:**

- Asigură conformitatea cu principiile ACID (Atomicitate, Consistență, Izolare, Durabilitate) pentru a garanta integritatea și coerența datelor în cadrul tranzacțiilor.

#### **Optimizare a Interogărilor:**

- Motorul de optimizare a interogărilor al PostgreSQL permite execuția eficientă a interogărilor complexe, facilitând accesul rapid la date.

#### **Suport pentru JSON și Datetime:**

- Oferă suport nativ pentru manipularea datelor JSON și pentru gestionarea tipurilor de dată datetime, contribuind la flexibilitatea și diversitatea datelor stocate.

#### **Securitate Avansată:**

- Implementează funcționalități avansate de securitate, inclusiv autentificare, autorizare și criptare, pentru a proteja datele împotriva accesului neautorizat.

### **Structura Bazei de Date:**

Baza de date a aplicației "Shopper" este gestionată cu ajutorul Spring Data JPA. Entitățile precum „ProductModel”, „TypeModel”, „CategoryModel”, etc., sunt mapate în tabele relaționale. Operațiile de CRUD sunt efectuate prin intermediul repository-urilor.

## **2.5 AVANTAJE/ DEZAVANTAJE ARHITECTURĂ**

Arhitectura proiectului "Shopper" oferă o serie de beneficii, dar și întâmpină anumite provocări. Iată o analiză a acestora:

#### **AVANTAJE:**

##### **1. Modularitate și Scalabilitate:**

Structura modulară permite dezvoltarea și scalabilitatea simplă a fiecărei componente (frontend, backend, bază de date) individual sau împreună.

## 2. Separarea Responsabilităților:

Frontend-ul, backend-ul și baza de date au responsabilități clare și sunt izolate între ele, facilitând gestionarea și întreținerea proiectului.

## 3. Flexibilitate Tehnologică:

Alegerea React pentru frontend, Spring Boot pentru backend și PostgreSQL pentru baza de date oferă o flexibilitate crescută, permițând utilizarea unor tehnologii potrivite pentru fiecare componentă.

## 4. Performanță și Eficiență:

Utilizarea unor tehnologii consacrate precum React și Spring Boot contribuie la performanța și eficiența generală a aplicației.

### DEZAVANTAJE:

#### 1. Complexitatea Inițială:

Implementarea unei arhitecturi modulare poate aduce o anumită complexitate inițială, în special pentru dezvoltatori începători.

#### 2. Coordonarea Dezvoltării:

Este necesară o coordonare atentă între dezvoltatorii frontend și backend pentru a asigura o integrare corespunzătoare între componentele aplicației.

#### 3. Resurse Suplimentare:

Gestionarea și întreținerea a trei componente distincte pot necesita resurse suplimentare și cunoștințe tehnice extinse.

#### 4. Securitatea Interfeței API:

Asigurarea securității între frontend și backend necesită atenție specială pentru a preveni vulnerabilitățile și atacurile cibernetice.

Cu toate acestea, beneficiile unei arhitecturi bine gândite depășesc dezavantajele, oferind un cadru solid pentru dezvoltarea și extinderea proiectului în viitor.

## 3 IMPLEMENTARE FRONTEND

Proiectul "Shopper" pune un accent deosebit pe experiența utilizatorului, oferind o interfață modernă și plăcută. Implementarea frontend-ului este realizată în tehnologia React, o bibliotecă JavaScript populară pentru construirea interfețelor web interactive.

### 3.1 COMPONENTE PRINCIPALE

Componenta **Card** este utilizată pentru afișarea detaliilor produselor în diferite secțiuni ale aplicației. Aceasta include informații precum nume, imagine, descriere, preț și dimensiuni

disponibile. Componenta oferă, de asemenea, opțiuni pentru adăugarea, creșterea sau scăderea cantității produselor în coș și eliminarea produselor din coș.

Componenta **Toastr** furnizează notificări pentru evenimente relevante, cum ar fi adăugarea unui produs în coș sau trimiterea cu succes a unei comenzi. Notificările pot avea diferite niveluri de severitate (success, error) și sunt afișate în partea de sus a paginii pentru o scurtă perioadă de timp.

**Navbar** oferă navigarea între diverse secțiuni ale aplicației și conține butoane pentru acțiuni precum autentificare, înregistrare și gestionarea coșului de cumpărături.

**Product** este responsabilă pentru afișarea detaliilor complete ale unui produs specific. Utilizează componenta **Card** pentru a prezenta produsul și furnizează opțiuni pentru a selecta dimensiunea și adăuga produsul în coș.

**Cart** afișează produsele adăugate în coș, furnizează totalul coșului și opțiunea de a trimite comanda. Aceasta utilizează componenta **Card** pentru afișarea detaliilor produselor din coș.

Hook-ul **useStatus** furnizează informații despre starea unei operații, cum ar fi încărcare, succes sau eroare, bazată pe funcționalitatea specificată.

Hook-ul **useNavigateHook** permite navigarea la anumite rute în funcție de succes sau eșecul unei operații.

Hook-ul **useResetHook** resetează starea unei funcționalități după ce o operație s-a încheiat cu succes sau cu eroare.

## 3.2 INTEGRARE CU Redux

Redux este utilizat pentru gestionarea stării globale a aplicației. Acțiunile specifice, reducerii și starea asociată fiecărui slice sunt gestionate folosind Redux Toolkit.

Primul pas în integrarea cu Redux este definirea slice-urilor aplicației. Acestea sunt segmente ale stării globale care conțin reducerii și acțiunile specifice. De exemplu, putem avea un slice pentru coșul de cumpărături, care să conțină reducerii pentru adăugarea, eliminarea sau actualizarea produselor din coș, împreună cu acțiunile corespunzătoare.

După ce am definit slice-urile, trebuie să le conectăm la componentele noastre folosind hook-urile **useSelector** și **useDispatch** furnizate de Redux. **useSelector** ne permite să accesăm starea globală și să extragem datele de care avem nevoie în componentă, în timp ce **useDispatch** ne permite să dispecerim acțiuni către reducerii noștri pentru a actualiza starea.

Pentru a adăuga un produs în coș, de exemplu, putem dispeceriza o acțiune **addToCart** folosind **dispatch** și să extragem informațiile relevante despre produs folosind **useSelector**. Aceste acțiuni vor actualiza apoi starea corespunzătoare din slice-ul coșului de cumpărături, iar componentele care depind de această stare se vor reîncărca automat pentru a reflecta modificările.



### 3.3 STILIZARE

Stilurile sunt definite folosind **styled-components** și sunt structurate în componente separate pentru a menține codul organizat și ușor de întreținut.

Fiecare componentă stilizată este definită folosind funcția **styled**, care ne permite să definim reguli CSS specifice pentru respectiva componentă. Acest lucru încurajează reutilizarea și configurabilitatea componentelor, permițându-ne să le aplicăm și să le adaptăm în diverse părți ale aplicației fără a compromite coerența stilistică.

Structura fișierelor noastre urmărește o organizare clară și concisă, cu componente separate pentru logica JavaScript și stilurile asociate acestora. Această abordare facilitează localizarea și modificarea stilurilor specifice unei componente fără a afecta alte părți ale aplicației.

Pentru a menține consistența vizuală și pentru a facilita schimbarea aspectului aplicației, definim teme globale și variabile de design. Acestea includ culori, tipografii și alte proprietăți de design utilizate pe scară largă în întreaga aplicație. Utilizarea acestor variabile ne permite să adaptăm rapid și ușor aspectul aplicației în funcție de nevoile și preferințele utilizatorilor sau ale echipei de dezvoltare.

### 3.4 INTERACȚIUNEA CU BACKEND-UL

Comunicarea cu backend-ul este gestionată prin intermediul hook-ului `useApi`. Acesta utilizează `axios` pentru a efectua cereri HTTP către API-ul backend-ului, manipulând starea datelor, încărcarea și erorile în mod eficient.

## 4 IMPLEMENTARE BACKEND

În cadrul proiectului "Shopper", implementarea backend-ului se bazează pe framework-ul Spring Boot, ceea ce ne oferă o structură robustă și ușor de gestionat pentru dezvoltarea aplicației noastre. Utilizând Spring Boot, putem defini rapid și eficient API-ul REST pentru a comunica cu frontend-ul și pentru a gestiona datele din sistemul nostru.

### 4.1 STRUCTURA PROIECTULUI SPRING BOOT

Proiectul Spring Boot este organizat în jurul conceptului de module și componente, ceea ce ne permite să structurăm și să împărțim codul în părți corespunzătoare.

Această structură modulară ne ajută să menținem codul bine organizat și ușor de gestionat pe măsură ce aplicația noastră crește în complexitate.

Iată structura principală a proiectului:

```
shopper-backend
|-- src
|   |-- configurations
|   |-- constants
|   |-- controllers
|   |-- dtos
|   |-- exceptions
|   |-- mappers
|   |-- models
|   |-- repositories
|   |-- services
|   |-- ShopperBackendApplication.java
|-- pom.xml
|-- README.md
```

- **configurations:** Conține configurațiile aplicației, cum ar fi configurarea bazei de date sau a securității.
- **constants:** Definirea constantelor utilizate în întreaga aplicație, cum ar fi mesajele de eroare sau valorile implicite.
- **controllers:** Controlerele care gestionează cererile HTTP primite de la frontend.
- **dtos:** Obiecte de transfer de date care definesc structura datelor trimise între frontend și backend.
- **exceptions:** Definirea și gestionarea excepțiilor personalizate ale aplicației.
- **mappers:** Mapează obiectele de transfer de date (DTO) în modele de date și invers.
- **models:** Modelele de date care reflectă structura entităților din baza de date.
- **repositories:** Interfețele care definesc operațiile de acces la date pentru modelele de date.
- **services:** Servicii care conțin logica de afaceri a aplicației.

## 4.2 PRIVIRE DE ANSAMBLU BACKEND

**Configurarea Encoderului de Parole:** Clasa *PasswordEncoderConfiguration* este responsabilă pentru configurarea encoderului de parole folosit pentru a securiza parolele utilizatorilor.

**Modelarea Datelor:** În cadrul claselor de model din pachetul *models*, cum ar fi *ProductModel*, *TypeModel*, *SizeModel* etc., sunt definite proprietățile entităților din baza de date, precum și relațiile dintre aceste entități.

**Repository-uri JPA:** În pachetul **repositories**, fiecare interfață de repository, cum ar fi **ProductRepository**, extinde **JpaRepository** și definește metode pentru a accesa și manipula datele legate de entitățile respective. De exemplu, metoda **findByName** din **ProductRepository** returnează un produs după nume.

**Servicii:** Clasa **ProductService** conține logica de afaceri pentru operațiile legate de produse, cum ar fi crearea unui produs nou sau obținerea produselor după anumite criterii. Aici, metoda create este responsabilă pentru crearea unui nou produs.

**Controlere:** În clasa **ProductController**, sunt definite metodele care mapează cererile HTTP la operațiile corespunzătoare din serviciu. De exemplu, metoda **createProduct** gestionează cererile POST pentru crearea de noi produse.

**Excepții Personalizate:** În pachetul **exceptions**, clasele cum ar fi **ConflictException** și **NotFoundException** sunt folosite pentru a semnaliza erori specifice, cum ar fi conflictul de resurse sau lipsa resurselor, în funcție de situația întâlnită în timpul executării.

**Utilizarea Lombok:** Clasa **ProductService** și altele folosesc adnotările Lombok, cum ar fi **@RequiredArgsConstructor** și **@Slf4j**, pentru a genera automat constructorii, metodele **toString()**, **equals()** și **hashCode()**, precum și logger-ul.

**Mapearea Obiectelor DTO:** În **ProductService**, se folosește o clasă de mappare **ProductMapper** pentru a transforma obiectele DTO (Data Transfer Object) în modele de date și invers. Acest lucru ajută la separarea logică dintre datele primite prin API și structura datelor din baza de date.

**Tranzacții și Anotări Transactional:** Metoda create din **ProductService** este marcată ca fiind **@Transactional**, asigurând consistența datelor și atomicitatea operațiilor realizate în timpul creării unui nou produs.

**Logging:** Clasele de serviciu și controler folosesc logging-ul **slf4j** pentru a înregistra diverse evenimente și informații relevante, cum ar fi încercările de creare a unui produs sau obținerea de produse după anumite criterii.

**Gestionarea Excepțiilor:** Pe lângă excepțiile personalizate, implementarea backend-ului include și gestionarea excepțiilor prin intermediul blocurilor **try-catch**, asigurând o manipulare corespunzătoare a erorilor și o comunicare adecvată a acestora către utilizatorii aplicației.

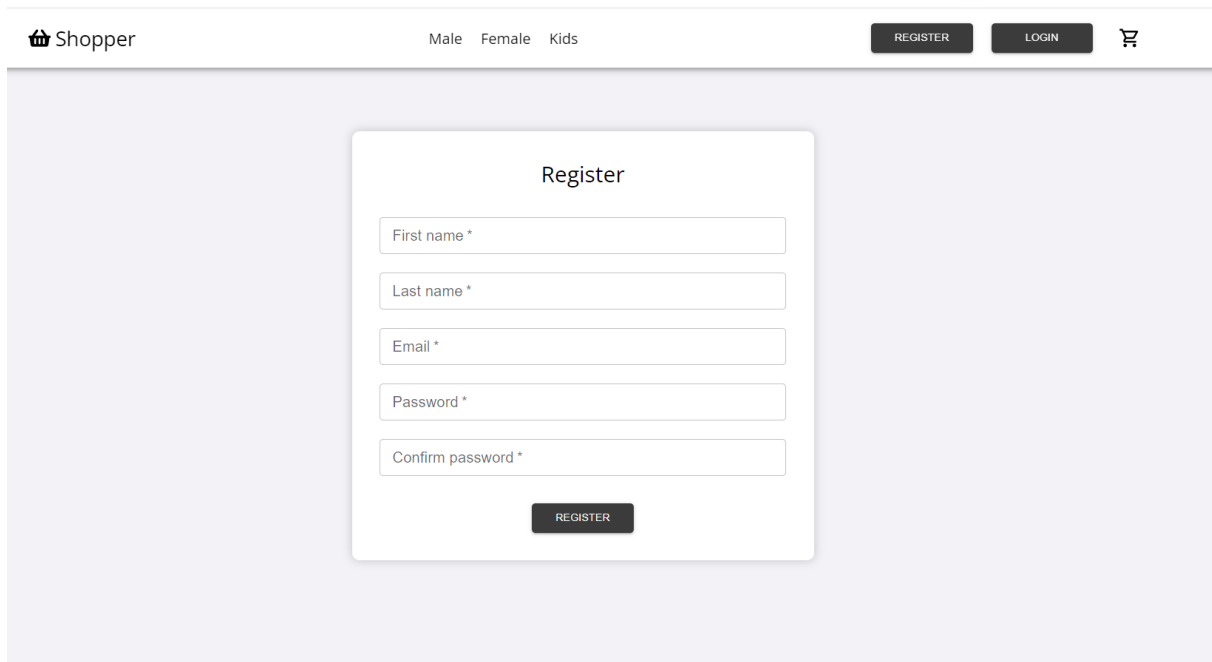
## 4.3 GESTIONAREA BAZEI DE DATE POSTGRESQL

Backend-ul utilizează o bază de date PostgreSQL pentru stocarea și recuperarea datelor. Configurările legate de bază de date sunt definite în fișierul **application.properties**, stabilind conexiunea cu baza de date și specificând dialectul Hibernate pentru PostgreSQL.

Backend-ul expune endpoint-uri HTTP gestionate de clasele de controlor. Aceste endpoint-uri sunt apelate de către frontend pentru a efectua cereri HTTP pentru obținerea, adăugarea, actualizarea sau ștergerea datelor.

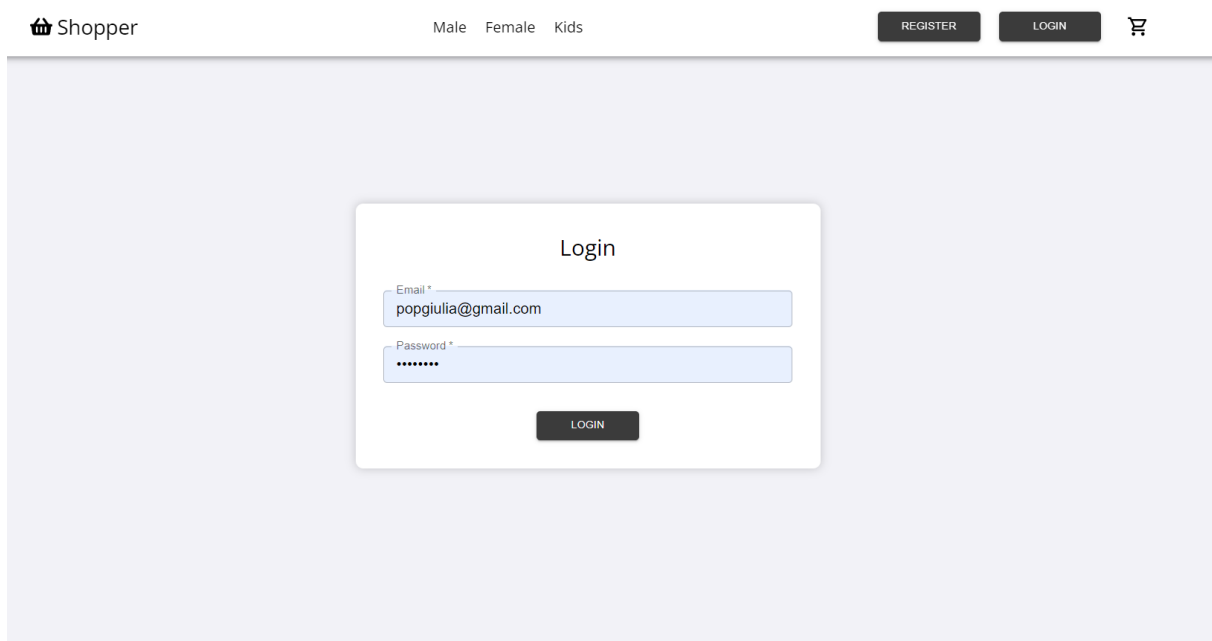
## 5 APLICAȚIA

Utilizatorii pot să vizualizeze hainele magazinul și fără a se înregistra, dar nu pot să le achiziționeze. În cazul în care un utilizator dorește să achiziționeze un produs, el va trebui să își introducă datele (nume, prenume, mail și parolă) pentru a-și putea face cont.



The screenshot shows the 'Register' form in the Shopper application. The header includes the 'Shopper' logo, category links 'Male', 'Female', and 'Kids', and buttons for 'REGISTER' and 'LOGIN'. The form itself is centered and contains five input fields: 'First name \*', 'Last name \*', 'Email \*', 'Password \*', and 'Confirm password \*'. A 'REGISTER' button is located at the bottom of the form.

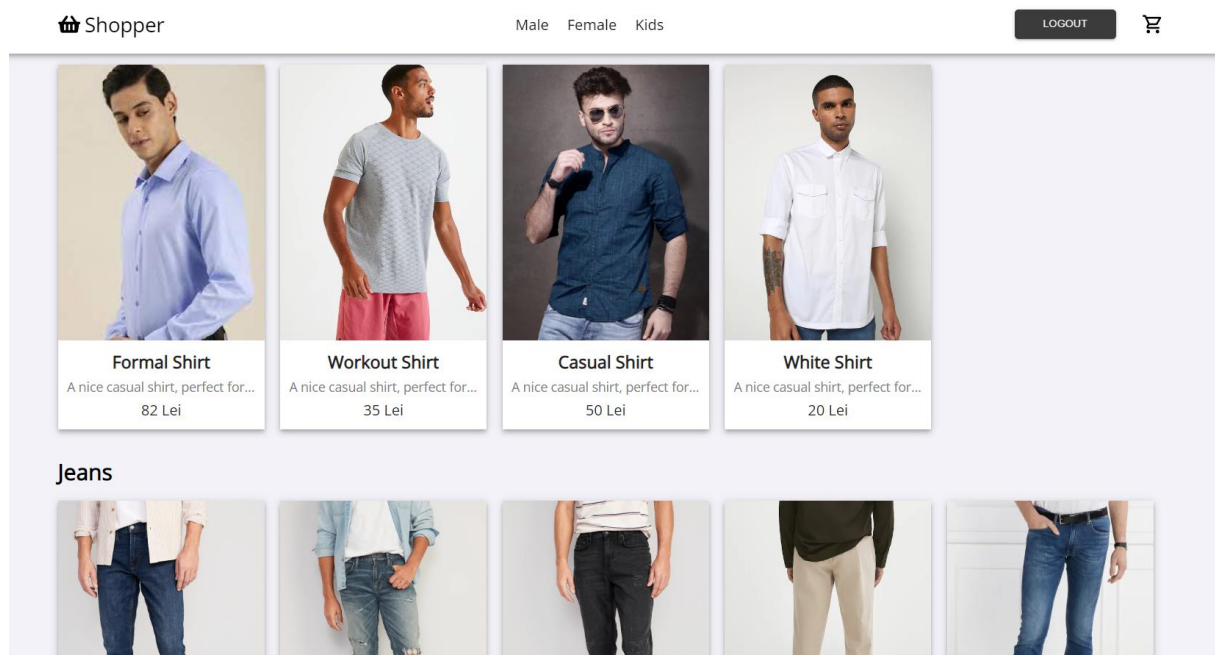
După ce utilizatorul și-a făcut contul, el se va loga.



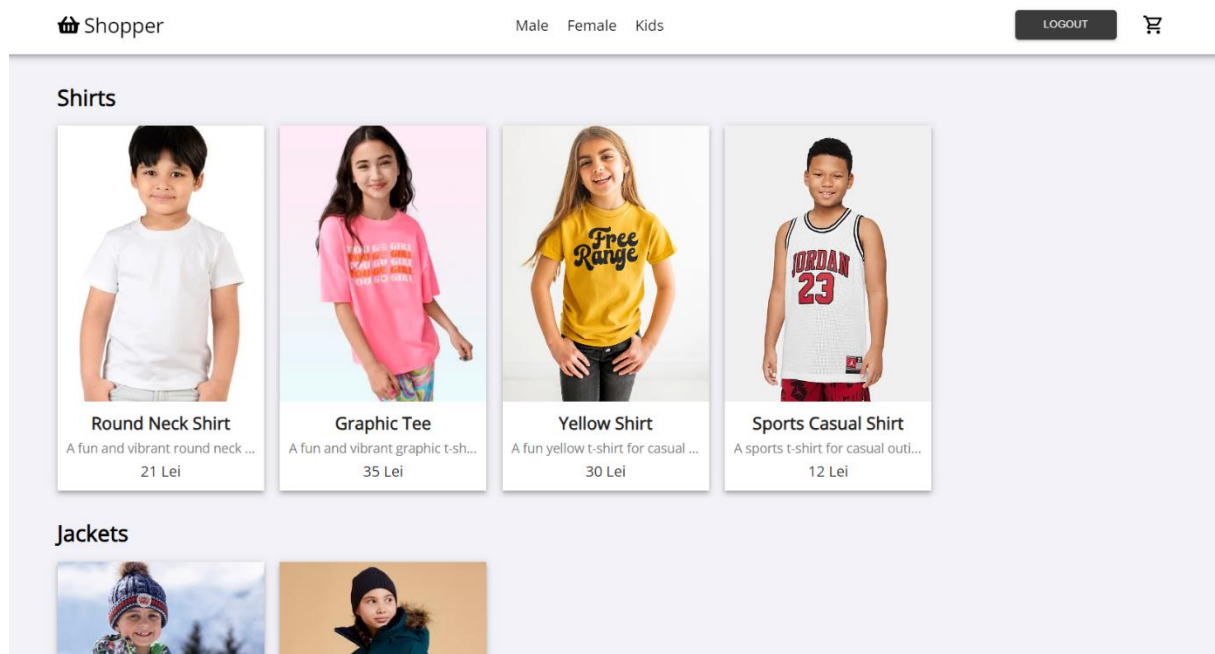
The screenshot shows the 'Login' form in the Shopper application. The header is identical to the previous one. The form is centered and contains two input fields: 'Email \*' with the value 'popgiulia@gmail.com' and 'Password \*' with masked characters '\*\*\*\*\*'. A 'LOGIN' button is located at the bottom of the form.

Pagina unde utilizatorul este redirecționat după logare este pagina dedicată vizualizării produselor și selectării produsului dorit. Aici el poate selecta din meniul de sus categoria ("male", "female", "kids") pe care o dorește și poate vedea detalii despre fiecare produs precum: denumire, descriere, poză și preț.

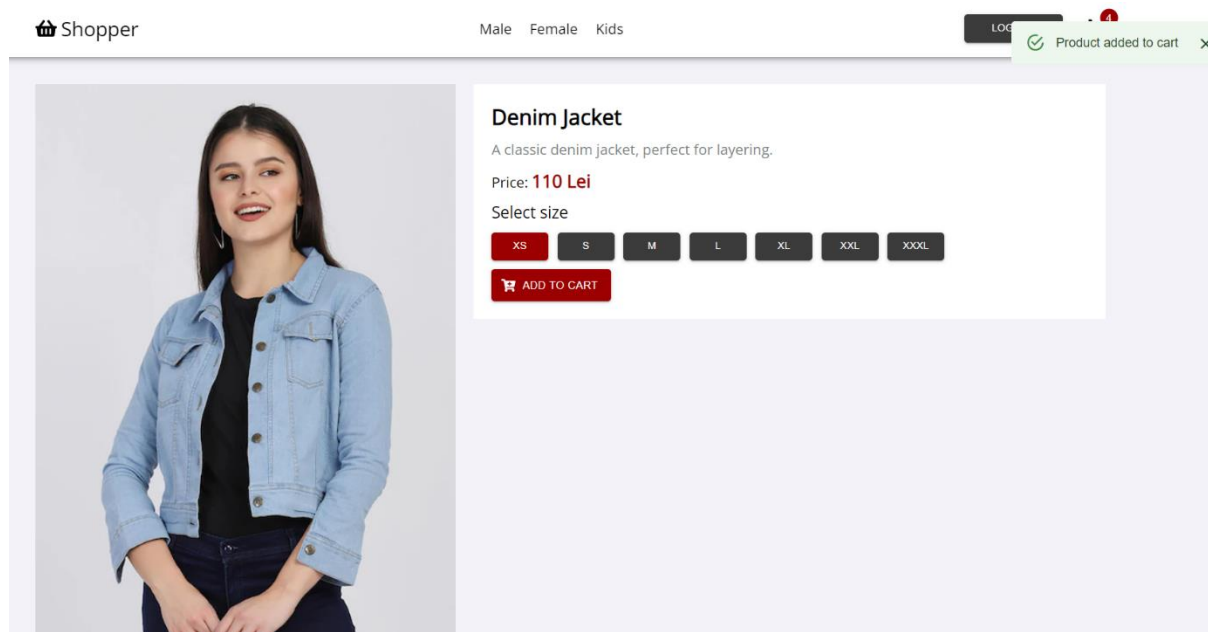
## Exemplu categorie bărbați:



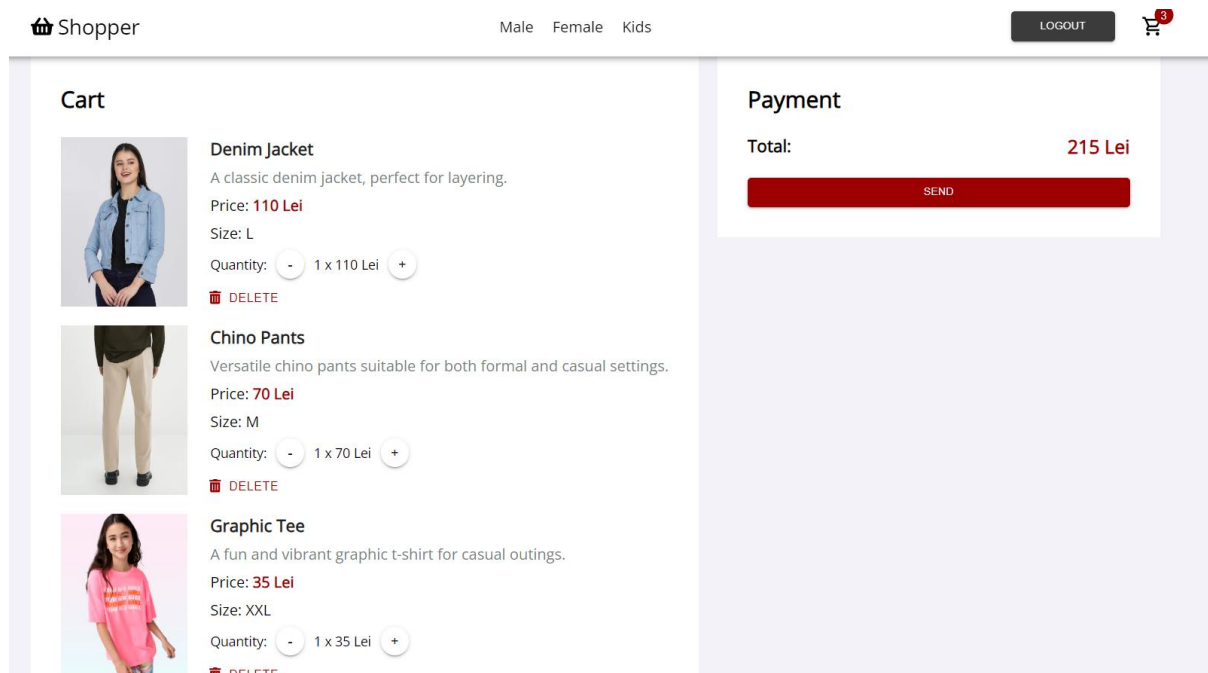
## Exemplu categorie copii:



Când se face click pe un produs, se va deschide acesta pagină cu detaliile lui. Aici se poate selecta mărimea dorită și se poate adăuga produsul în coș.



Când se apasă pe iconul coșului, se va intra pe această pagină în care se pot vedea toate produsele adăugate, cu detaliile lor. De asemenea, acetse produse se pot șterge în cazul în care utilizatorul nu le mai dorește. În partea dreaptă a ecranului, el va putea vizualiza totalul de plată și va putea apăsa send pentru a finaliza comanda. În momentul apăsării butonului send, coșul de cumpărături se va goli.



## 6 CONCLUZII

Proiectul "Shopper" reprezintă o aplicație web de succes, creată pentru a oferi utilizatorilor o experiență plăcută și eficientă în cumpărarea de haine. Implementarea backend-ului și frontend-ului a fost realizată cu succes, oferind o interfață intuitivă și funcționalități esențiale.

Prin utilizarea tehnologiilor moderne precum Spring Boot pentru backend și React pentru frontend, am reușit să construiesc o aplicație scalabilă și ușor de întreținut. Utilizarea unor practici de dezvoltare robuste, cum ar fi gestionarea excepțiilor, logging-ul adecvat și mapearea obiectelor DTO, a contribuit la realizarea unei aplicații fiabile și sigure.

Totuși, proiectul nu a fost lipsit de provocări. În timpul implementării, am întâlnit dificultăți în gestionarea comunicării între frontend și backend, precum și în optimizarea performanței aplicației. În plus, procesul de testare și depanare a necesitat eforturi considerabile pentru a asigura calitatea și funcționalitatea corectă a aplicației.

În concluzie, proiectul "Shopper" reprezintă o reușită în domeniul dezvoltării de aplicații web și oferă o bază solidă pentru extinderea și îmbunătățirea ulterioară. Lecțiile învățate în timpul implementării vor servi drept ghid pentru proiectele viitoare, iar feedback-ul utilizatorilor va fi valorificat pentru a aduce îmbunătățiri și inovații în continuare.

## 7 DEZVOLTĂRI ULTERIOARE

### **Optimizare Performanță:**

Se pot aplica tehnici de optimizare a performanței pentru a asigura un timp de încărcare redus și o experiență fluidă utilizatorilor.

### **Integrare Plăți Online:**

Adăugarea unui sistem de plăți online ar îmbunătăți experiența de cumpărare și ar face magazinul online mai funcțional.

### **Extinderea Gamei de Produse:**

Diversificarea Categoriilor de Produse: Adăugarea unor categorii noi de produse sau extinderea gamei actuale pentru a satisface nevoile variate ale clienților și pentru a atinge un public mai larg.

Link GITHUB Frontend:

<https://github.com/popgiulia/shopper-react>

Link GITHUB Backend:

[https://github.com/popgiulia/shopper\\_backend](https://github.com/popgiulia/shopper_backend)

## 8 BIBLIOGRAFIE

- [1] C. Walls, *Spring in Action*, editura Manning Publications, Greenwich, Connecticut, Statele Unite ale Americii.
- [2] M. Schwarzmüller, *React: The Complete Guide*, editura Packt Publishing, Birmingham, UK.
- [3] <https://docs.spring.io/spring-framework/docs/current/reference/html/web.html>
- [4] <https://www.postgresql.org/docs/>
- [5] <https://reactjs.org/docs/getting-started.html>