Shop Online

Cătălina Racolța

October 2025

Contents

1 Introducere

Am ales această temă deoarece, în petrec destul de mult timp cumpărând de pe online. M-am gândit să implementez o aplicație web care să îi ajute pe utilizatori să își achiziționeze produsele dorite într-un mod rapid, eficient și sigur, fără a mai fi nevoiți să meargă fizic în magazine.

Aplicația "Shop Online" oferă posibilitatea utilizatorilor de a vizualiza produse disponibile, de a le adăuga în coșul de cumpărături, precum și de a consulta sau șterge produse din acesta. Ideea proiectului a pornit de la dorința de a crea o platformă ușor de utilizat, accesibilă oricărui tip de utilizator, indiferent de nivelul de experiență tehnologică.

Cumpărăturile online reprezintă o alternativă modernă și practică, permițându-le oamenilor să își cumpere produsele preferate prin doar câteva clickuri. Prin intermediul acestei aplicații, îmi propun să aduc o soluție simplă și eficientă pentru procesul de cumpărare, oferind o experiență plăcută și intuitivă.

Prin realizarea acestui proiect, îmi doresc să evidențiez importanța digitalizării comerțului și să demonstrez modul în care tehnologia poate simplifica activitățile de zi cu zi. Aplicația are scopul de a face cumpărăturile mai accesibile, mai rapide și mai organizate, contribuind astfel la modernizarea experienței de shopping online.

2 Context, cerințe, MoSCoW, cazuri de utilizare, Arhitectura

2.1 Context

M-am gândit ca aplicația să fie ușor de utilizat pentru clienți și administratori, astfel adăugând câteva secțiuni cheie. Utilizatorii vor avea un formular de înregistrare cât și de logare unde vor fi redirecționați în aplicație. După ce au intrat pe site-ul principal, vor avea o primă pagină de unde vor putea selecta diferite articole.

2.2 Cerințe funcționale

Cerințele funcționale au rolul de a defini și modela modul în care aplicația rulează, cum se realizează procesele interne și cum este vizualizat de către utilizator.

1. Clienții o să poată să filtreze produsele, să adauge produse în coșul de cumpărături, să vadă produsele disponibile și prețurile acestora.
2. Administratorii o să poată să adauge, steargă sau modifica un produs.
3. Clienții au să poată sterge din coșul de cumpărături produse și să vadă prețul total.

2.3 Cerințe non-funționale

Aceste cerințe non-funcționale au rol la fel de important ca și cele funcționale, definind performanța dar și calitatea platformei. Fără a avea aceste cerințe aplicația poate să fie instabilă cât și nesigură pentru utilizatori, lipsindu-i eficiența.

1. Securitatea este unul dintre cele mai importante aspecte ale aplicației deoarece este stocat în ea date cu caracter personal al utilizatorilor. Va fi nevoie să asigur protecția acestor date printr-un autentificator în doi factori dar și cripatrea datelor.

2. Printr-o interfață interactivă și ușor de folosit, o să ajut pe utilizatori să petreacă cât mai puțin timp și să găsească produsul dorit în cel mai scurt timp posibil.

2.4 Schema MoSCoW

Schema sau metoda MoSCoW are rolul de prioritizare a cerințelor, asigurând implementarea funcționalităților de bază, de care este obligatoriu nevoie, mai apoi în cazul în care este nevoie, având posibilitatea de a adăuga unele cerințe opționale.

Must Have

1. Bază de date pentru produse
2. Bază de date pentru utilizatori
3. Sistem de logare cu drepturi diferite (admin/utilizator)
4. Gestionare coș de cumpărături (adăugare/ștergere produse)
5. Vizualizare produse în pagină web
6. Persistența datelor

Should Have

1. Istoric comenzi
2. Verificare stoc
3. Pagină detalii produs
4. Notificare la adăugarea produselor în coș

Could have

1. Donare haine
2. Termen de livrare
3. Notificare termen predare/primire comandă

4. Rezervare produse

Won't have

1. Sistem complet de plată online

2.5 Cazuri de utilizare

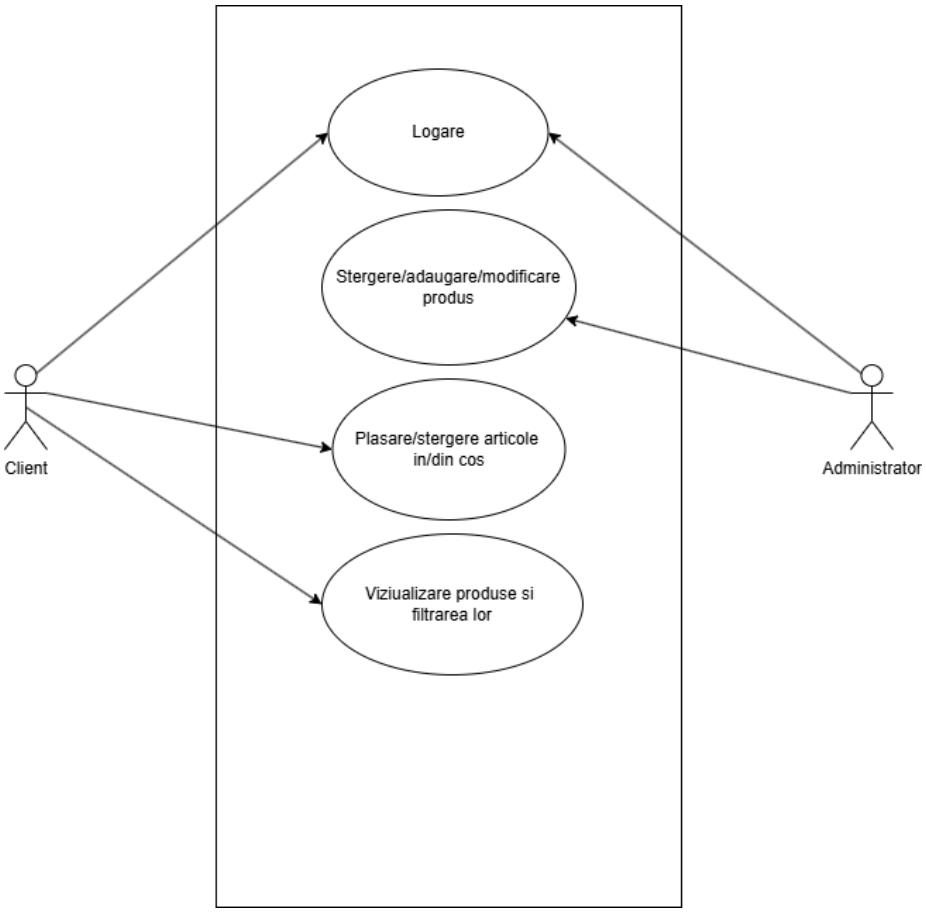

Figure 1: Cazuri de utilizare

2.6 Arhitectura

Arhitectura aplicației este construită pentru a fi scalabilă, modulară și eficientă, integrând tehnologii moderne precum React pentru partea de front-end, Spring Boot pentru back-end și MySQL pentru gestionarea datelor. Pentru comunicarea în timp real, aplicația utilizează WebSockets, permițând actualizarea

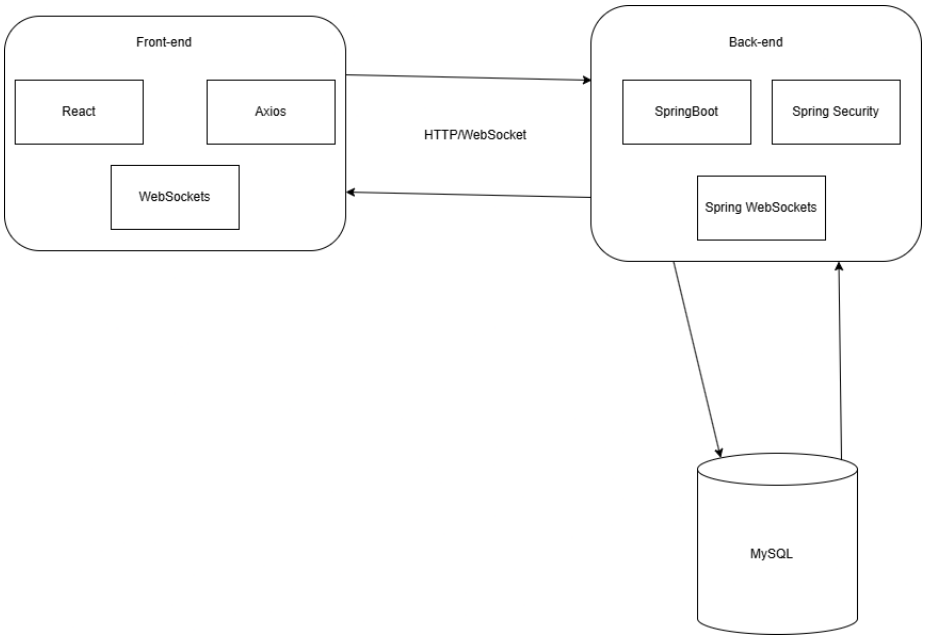

Figure 2: Arhitectura

instantanee a informațiilor fără reîncărcarea paginii. De asemenea, securitatea este asigurată prin mecanisme moderne de autentificare și autorizare, bazate pe Web Session sau JSON Web Token (JWT), oferind astfel protecție robustă și control sigur al accesului utilizatorilor.

2.6.1 Front-End React

React este utilizat pentru realizarea interfeței aplicației datorită modularității și performanței sale ridicate. Prin utilizarea componentelor React, este posibilă gestionarea eficientă a stării aplicației și crearea unei experiențe interactive și dinamice pentru utilizator.

2.6.2 Back-End Spring Boot

Pentru partea de back-end, aplicația utilizează Spring Boot, un framework modern și robust din ecosistemul Spring, care facilitează dezvoltarea rapidă a aplicațiilor Java bazate pe arhitectura REST. Spring Boot oferă o structură clară, modulară și ușor de extins, permițând implementarea logicii de afaceri, gestionarea datelor și comunicarea eficientă cu front-end-ul.

2.6.3 Comunicare între Front-End și Back-End

Comunicarea dintre front-end și back-end se realizează prin servicii RESTful, care asigură schimbul de date între React și Spring Boot. Front-end-ul utilizează biblioteca Axios pentru trimiterea cererilor HTTP și primirea răspunsurilor, datele fiind transmise în format JSON. Pentru actualizări în timp real, aplicația folosește WebSockets, permițând o interacțiune dinamică fără reîncărcarea paginii.

2.6.4 Baza de date MySQL

Pentru stocarea informațiilor aplicației "Shop Online" s-a utilizat sistemul de gestiune a bazelor de date MySQL. Baza de date este esențială pentru gestionarea produselor, utilizatorilor și a coșului de cumpărături, asigurând persistența datelor între sesiunile aplicației.

2.6.5 Beneficii ale acestei arhitecturii

Arhitectura aplicației "Shop Online", bazată pe Front-End React, Back-End Spring Boot și baza de date MySQL, oferă numeroase avantaje care contribuie la performanța, scalabilitatea și flexibilitatea sistemului:

1. Scalabilitate: Arhitectura permite extinderea facilă a capacităților aplicației pe măsură ce numărul de utilizatori și volumul de date cresc.

2. Eficiență în dezvoltare: Utilizarea framework-urilor populare și bine documentate, precum React și Spring Boot, facilitează o dezvoltare rapidă și structurată, reducând timpul necesar implementării funcționalităților.

3. Performanță optimizată: Comunicarea eficientă între Front-End și Back-End, împreună cu utilizarea unei baze de date robuste precum MySQL, asigură un timp de răspuns rapid și o experiență fluidă pentru utilizator.

4. Flexibilitate și extensibilitate: Modularitatea componentelor aplicației permite adăugarea de funcționalități noi sau modificarea celor existente fără a afecta restul sistemului, oferind astfel adaptabilitate la cerințele viitoare.