



**UNIVERSITATEA DIN  
BUCUREȘTI**

**FACULTATEA DE  
MATEMATICĂ ȘI  
INFORMATICĂ**



**SPECIALIZAREA CALCULATOARE ȘI TEHNOLOGIA  
INFORMAȚIEI**

**Proiect de diplomă**

**APLICAȚIE WEB PENTRU GESTIONAREA  
PRODUSELOR ELECTRONICE**

**Absolvent  
NISTOR ANDREEA-CRISTINA**

**Coordonator științific  
Lect. Dr. Carmen Elena Chiriță**

**București, iunie-iulie 2023**

## Rezumat

Aplicația „Device World” este o aplicație web pentru gestionarea produselor electronice din diferite categorii. Scopul proiectului este dezvoltarea unei platforme intuitive și atractive pentru utilizatori, unde aceștia au posibilitatea de a achiziționa și vizualiza produse electronice, în timp ce administratorul are acces la funcționalități suplimentare pentru gestionarea produselor și a utilizatorilor.

Relevanța temei constă în faptul că piața produselor electronice este în continuă dezvoltare și necesită o modalitate eficientă de gestionare a acestora. O aplicație de acest fel poate facilita procesul de cumpărare, oferind utilizatorilor o experiență personalizată și ușor de utilizat.

Prin intermediul aplicației, utilizatorii își pot crea conturi și se pot autentifica în calitate de client sau administrator. Autentificarea este necesară pentru a beneficia de diferite funcționalități specifice fiecărui rol. Utilizatorii pot explora o gamă largă de produse electronice și pot utiliza filtre pentru a găsi cu ușurință produsele dorite.

Aplicația conține o serie de caracteristici cheie pentru utilizatori: coșul de cumpărături, lista de dorințe, istoricul comenzilor, descărcarea facturilor pentru comenzi, recomandări ale altor produse relevante. Astfel, fiecare utilizator își va adăuga produsele în coșul de cumpărături, de unde va mai primi și alte recomandări de produse și în cazul în care dorește să plaseze comanda, acest lucru va fi accesibil în aceeași pagină, iar informațiile despre comenzi vor fi salvate într-o pagină separată de istoric al comenzilor. Lista de dorințe permite utilizatorilor să păstreze produsele care le-au atras atenția pentru a putea fi accesate cu ușurință în viitor.

Pentru administratori, aplicația oferă, în plus, instrumente de gestionare a produselor și utilizatorilor. Aceștia pot adăuga noi produse în baza de date și pot vizualiza, modifica sau șterge produsele existente deja în baza de date. De asemenea, ei pot gestiona utilizatorii având posibilitatea de a modifica sau șterge conturile din baza de date.

## **Abstract**

“Device World” is a web application designed for managing electronic products across different categories. The aim of the project is to develop an intuitive and appealing platform for users, where they can purchase and view electronic products, while the administrator has access to additional functionalities for product and user management.

The relevance of the topic lies in the fact that the market for electronic products is continuously evolving and requires an efficient way for managing. Such an application can facilitate the purchasing process, offering users a personalized and user-friendly experience.

Through the application, users can create accounts and authenticate themselves as customers or administrators. Authentication is necessary to access role-specific functionalities. Users can explore a wide range of electronic products and utilize filters to easily find the desired items.

The application includes several key features for users: the shopping cart, wishlist, order history, invoice downloads, and recommendations for other relevant products. Each user can add products to their shopping cart, receive additional product recommendations and place an order if desired, all within the same page. Information about orders is stored separately in an order history page. The wishlist allows users to keep track of products they find interesting for future reference.

For administrators, the application provides tools for product and user management in addition to the functionalities available to customers. They can add new products to the database, view, modify, or delete existing products. Furthermore, administrators can manage users by modifying or deleting user accounts from the database.

# Cuprins

<b>1. Introducere.....</b>	<b>6</b>
1.1.    Prezentare generală .....	6
1.2.    Scopul și motivația alegerii temei.....	7
1.3.    Structura lucrării .....	9
<b>2. Tehnologii utilizate .....</b>	<b>10</b>
2.1.    Mediul de dezvoltare .....	10
2.2.    Frontend.....	10
2.2.1.    HTML.....	11
2.2.2.    CSS.....	11
2.2.3.    JavaScript .....	11
2.2.4.    React.js .....	11
2.2.5.    Material-UI.....	15
2.3.    Backend .....	15
2.3.1.    Node.js .....	16
2.3.2.    PostgreSQL.....	17
<b>3. Implementarea aplicației .....</b>	<b>18</b>
3.1.    Structura fișierelor.....	18
3.2.    Baza de date.....	19
3.3.    Conținutul fișierului principal.....	20
3.4.    Selectarea limbii în aplicație.....	22
3.5.    Înregistrarea și autentificarea utilizatorilor .....	23
3.6.    Produsele .....	27
3.6.1.    Pagina de produse .....	27
3.6.2.    Pagina fiecărui produs .....	30
3.7.    Coș de cumpărături .....	31
3.8.    Comenzi .....	33
3.9.    Listă de dorințe .....	35
3.10.    Gestionarea produselor.....	35
3.11.    Gestionarea utilizatorilor.....	38
3.12.    Aplicație responsive.....	38

<b>4. Ghid de utilizare.....</b>	<b>41</b>
4.1. Înregistrarea și autentificarea .....	41
4.2. Vizualizarea produselor.....	42
4.3. Coșul de cumpărături .....	43
4.4. Lista de dorințe.....	44
4.5. Vizualizarea comenzilor .....	45
4.6. Funcționalitățile administratorului.....	45
4.6.1. Gestionarea utilizatorilor .....	46
4.6.2. Gestionarea produselor .....	46
4.7. Delogarea utilizatorului.....	47
<b>5. Concluzii .....</b>	<b>48</b>
<b>Bibliografie.....</b>	<b>49</b>
<b>Listă figuri .....</b>	<b>50</b>

# 1. Introducere

## 1.1. Prezentare generală

„Device World” este o aplicație web dezvoltată având partea de client realizată cu ajutorul bibliotecii React.js și partea de server construită folosind Node.js la care este conectată baza de date creată în PostgreSQL. Aplicația are două tipuri de utilizatori, administrator și client. Aceștia trebuie să își creeze un cont în aplicație. O funcționalitate importantă a contului de administrator este gestionarea produselor și a utilizatorilor, acesta având dreptul să modifice sau să șteargă conturile celorlalți utilizatori, dar și să adauge produse noi, să modifice și să șteargă produsele existente în baza de date. Clientul, fiind autentificat, poate să adauge produse în lista de dorințe sau în coșul de cumpărături și poate plasa comenzi, primind atât informațiile despre comenzi, cât și factura fiecărei comenzi în pagina destinată comenzilor.

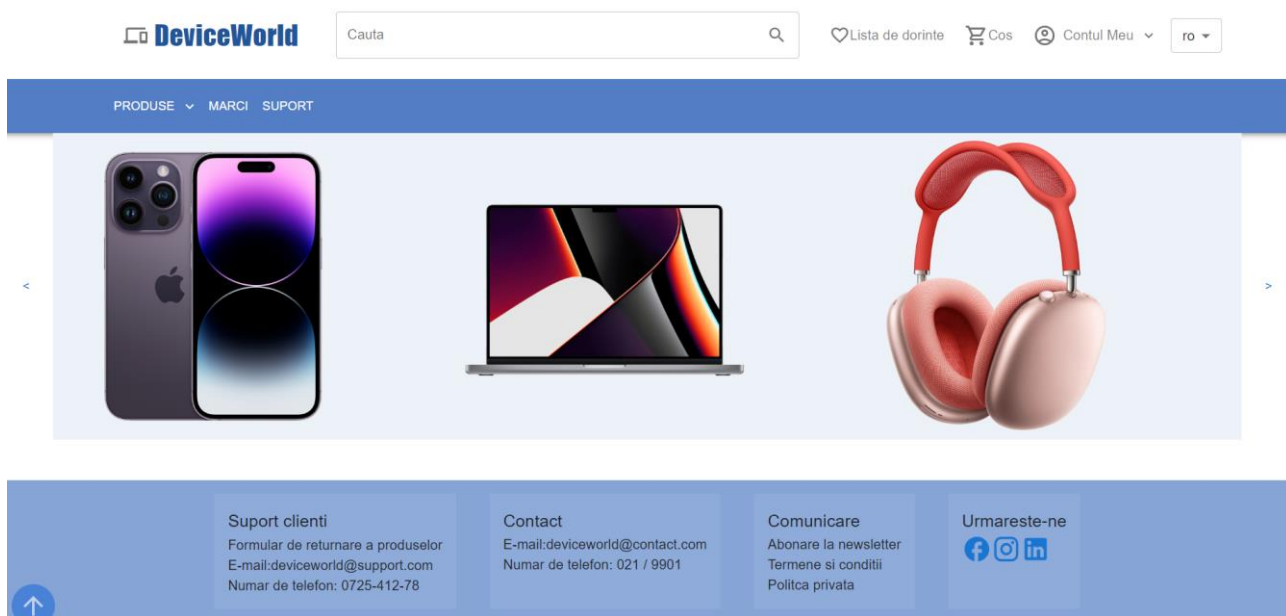


Fig. 1.1. – Pagina de acasă în aplicația Device World

Coșul de produse este o componentă esențială a aplicației, pentru că adăugând produsele în coș, utilizatorii pot realiza chiar de acolo plasarea comenzii. În plus, aceștia vor primi și recomandări de produse care s-ar potrivi cu ce este deja în coș.

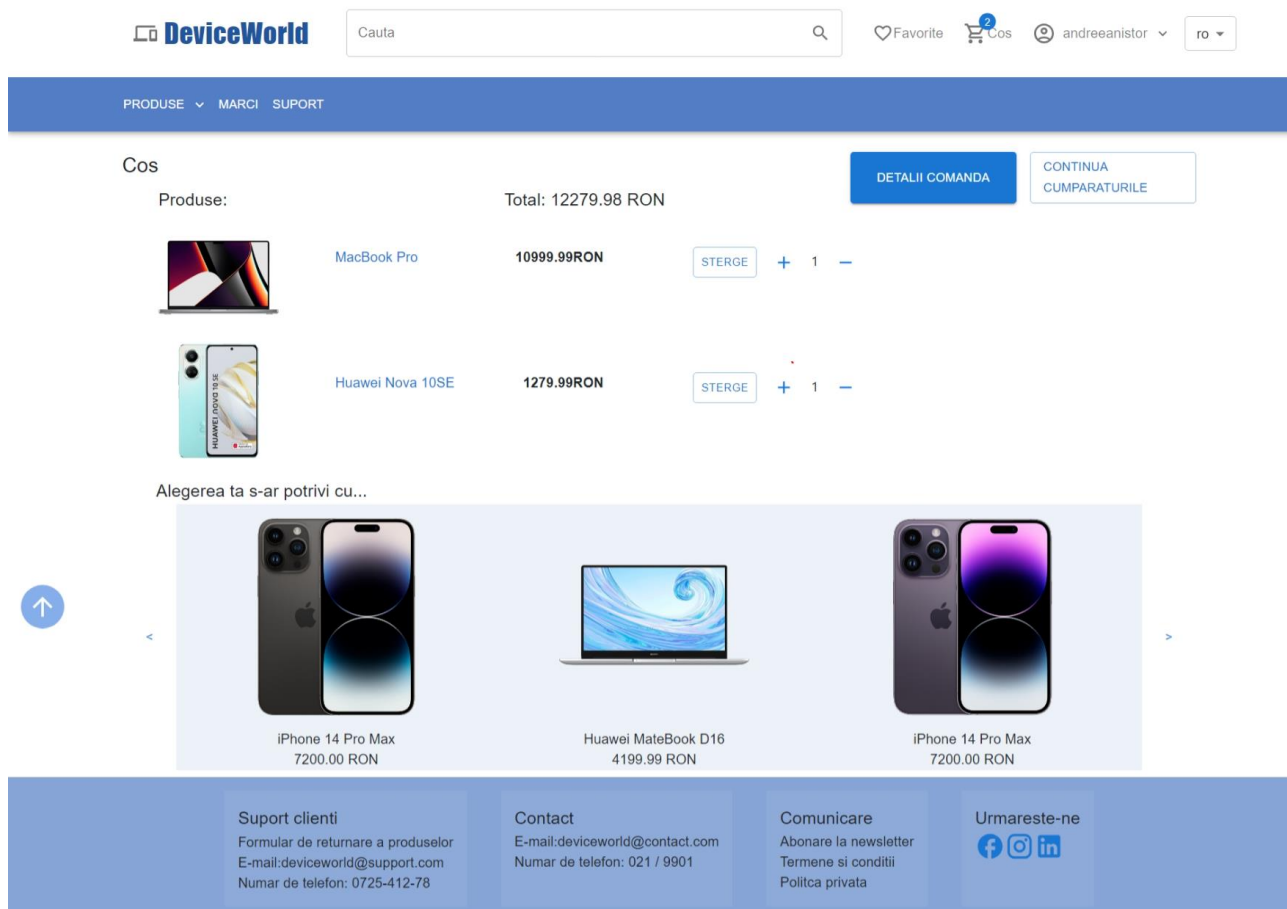


Fig. 1.2. – Pagina coșului de produse din aplicația Device World

## 1.2. Scopul și motivația alegerii temei

### 1.2.1. Motivația alegerii temei

Piața produselor electronice este în continuă creștere în ultimul timp și este necesară o metodă eficientă de gestionare a acestora. O aplicație de gestionare a produselor electronice oferă utilizatorilor posibilitatea de a avea o experiență simplă și convenabilă în ceea ce privește achiziționarea de produse electronice online. Cu ajutorul conturilor de clienți, utilizatorii pot adăuga produse în lista de dorințe sau coșul de cumpărături, pot accesa comenzile și pot filtra produsele în funcție de preferințele lor.

Un alt motiv pentru care am ales dezvoltarea unei astfel de aplicații este faptul că în ultimii trei ani, odată cu instaurarea restricțiilor din cauza pandemiei cauzată de virusul Covid-19, oamenii au început tot mai mult să achiziționeze produse electronice online și continuă să facă acest lucru, pentru că este mult mai ușor pentru aceștia să vizualizeze specificațiile produselor, să compare prețurile și produsul dorit ajunge direct la adresa specificată. De asemenea, în ultimul timp, lumea

pune din ce în ce mai mult accent pe produsele electronice, deoarece au nevoie ca acestea să fie cât mai calitative pentru a face față nevoilor fiecăruia. Cererea pentru produsele electronice este într-o continuă creștere, ceea ce reprezintă un motiv în plus pentru a dezvolta o aplicație destinată gestionării produselor de acest tip.

Având în vedere că totul trebuie să fie cât mai simplu și accesibil pentru orice utilizator, prin analizarea aplicațiilor deja existente pe piață am ales să implementez funcționalități care să fie cât mai ușor de înțeles pentru toată lumea, fapt pentru care am dezvoltat o aplicație cu o interfață intuitivă și prietenoasă cu utilizatorul, ajutându-l uneori cu recomandări sau indicații.

### 1.2.2. Motivația alegerii tipului de aplicație

Alegerea pentru construirea unei aplicații web este strâns legată de interesul pentru acest domeniu, dar și faptul că acest tip de aplicație vine cu anumite avantaje:

- Utilizatorii accesează ultima versiune a aplicației fără să fie nevoiți să instaleze versiuni actualizate ale acestora pe dispozitivele proprii
- Nu este nevoie ca aplicația să fie instalată pe dispozitivele utilizatorilor
- Aplicația poate fi accesată de pe toate tipurile de platforme (desktop, laptop sau dispozitive mobile), adaptându-și aspectul în funcție de dimensiunea dispozitivului
- Aplicația poate fi accesată de pe orice tip de browser, având nevoie doar de conexiune la internet și neconsumând resurse suplimentare pentru utilizare.

[1]

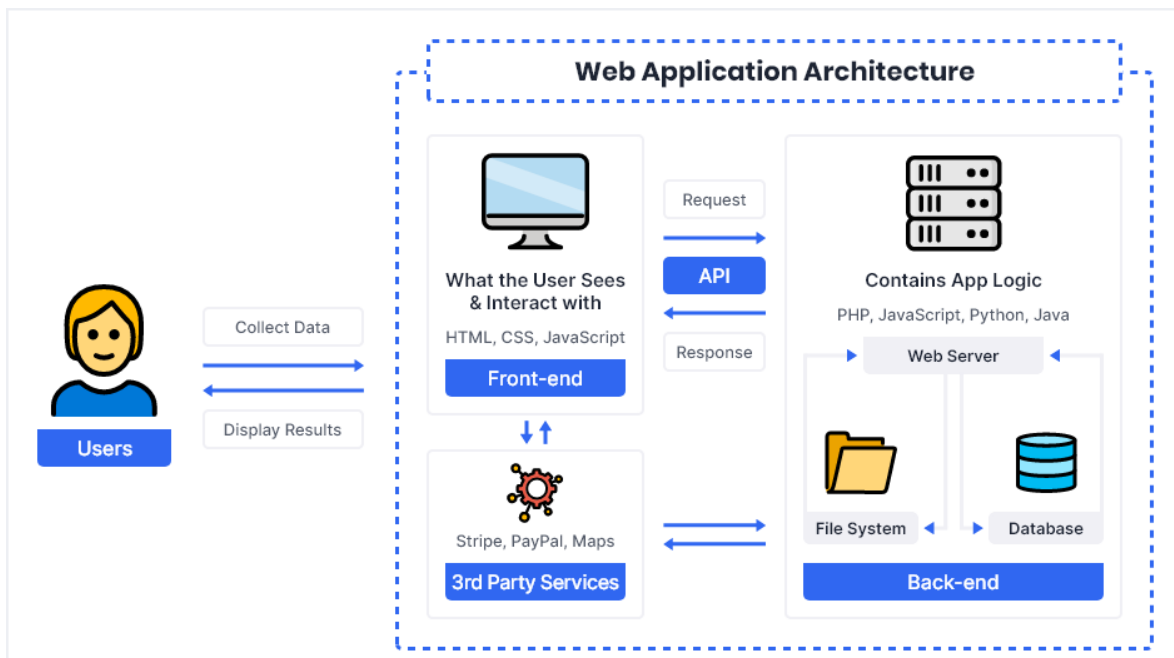


Fig. 1.3. – Structura unei aplicații web [2]



### 1.3. Structura lucrării

Documentația acestui proiect este alcătuită din 5 capitole care prezintă câte un aspect important al aplicației.

În primul capitol, **introducerea**, sunt descrise funcționalitățile aplicației fără a intra foarte mult în detalii. Acest capitol pune accentul mai mult pe motivele care au determinat alegerea temei și tipului de aplicație.

Al doilea capitol face referire la **tehnologiile utilizate** pentru dezvoltarea aplicației, prezentând câteva noțiuni generale despre fiecare în parte și justificând particularitățile alese în implementarea aplicației.

Al treilea capitol detaliază **modul de implementare** al aplicației folosind tehnologiile prezentate în capitolul anterior.

Al patrulea capitol este reprezentat de **ghidul de utilizare** al aplicației unde toate funcționalitățile pentru client și administrator sunt explicate mai ales cu ajutorul imaginilor pentru a fi mult mai sugestiv.

Ultimul capitol conține **concluzii** referitoare la modul de implementare și eventuale sugestii de dezvoltare ce se doresc a fi introduse în aplicație.

## 2. Tehnologii utilizate

### 2.1. Mediul de dezvoltare

Pentru dezvoltarea aplicației de gestionare a produselor electronice, am ales să utilizez mediul de dezvoltare Visual Studio Code, deoarece permite dezvoltarea aplicațiilor care folosesc JavaScript, atât pentru partea client, cât și pentru partea de sever.

Visual Studio Code este un editor ușor, dar complex compatibil cu Windows, macOS și Linux. Deși are suport încorporat pentru limbajele JavaScript, TypeScript și Node.js, acesta permite și instalarea extensiilor pentru celelalte limbaje de programare (C++, Python, Go, .NET, PHP, C#, Java, etc). De asemenea, are mai multe funcționalități care ajută la scrierea rapidă a codului și aranjarea în pagină. [3]

Pentru dezvoltarea aplicației am folosit următoarele extensii:

- Prettier – Code formatter ajută la indentarea codului pentru a fi mai ușor de citit și urmărit.
- ES7+ React/Redux/React-Native Snippets care ajută la reducerea timpului de scriere a codului pentru că este generat un template specific. De exemplu *rfce* generează un template pentru scrierea unei componente funcționale în React.
- Gitlens permite celui care scrie cod să vadă cine a scris fiecare linie de cod, când și în ce commit.
- Conectarea la Github prin care din când în când (preferabil după fiecare operațiune esențială implementată) se încarcă codul în repository-ul său păstrându-se cât mai organizat. De asemenea, dacă o modificare nouă strică codul deja existent, se poate reveni la versiunea dinaintea modificării.

### 2.2. Frontend

Partea de frontend reprezintă partea clientului, adică interacțiunea directă cu utilizatorul prin interfața intuitivă, prietenoasă și cât mai ușor de utilizat de către toată lumea și trimiterea cererilor către server pentru anumite rute. Tehnologiile utilizate pentru dezvoltarea acestei părți sunt biblioteca React.js (sintaxa JSX) a limbajului JavaScript și pentru stilizare limbajul CSS și biblioteca MaterialUI. La baza bibliotecii React.js stau limbajele HTML, CSS și JavaScript.

### **2.2.1. HTML**

HTML (Hypertext Markup Language) este un limbaj folosit pentru a crea structura și conținutul paginilor web. HTML utilizează diferite tag-uri (titlu, text, header, paragraf, etc) care definesc structura și elementele de pe pagină. Ultima versiune este HTML5 și a adus noi elemente semantice (<header>, <nav>, <section>, <footer>, <article>) pentru o structură mai clară a conținutului, suport nativ pentru video și audio, elemente de formular îmbunătățite, stocare locală pentru date persistente (Local Storage și Session Storage). Aceste îmbunătățiri permit crearea experiențelor web mai avansate și mai interactive. [4]

### **2.2.2. CSS**

CSS (Cascading Style Sheet) este un limbaj de stilizare care se folosește pentru îmbunătățirea aspectului vizual al elementelor HTML în paginile web. Acest limbaj permite definirea proprietăților stilistice (culoarea și dimensiunea). CSS folosește reguli și selector pentru aplicarea stilurilor și le poate adapta în funcție de dimensiunea ecranului. Cu ajutorul acestui limbaj se pot crea animații și tranziții, este flexibil și modular, premittând reutilizarea și gestionarea eficientă a stilurilor. [5]

### **2.2.3. JavaScript**

JavaScript este un limbaj de programare utilizat pentru a crea funcționalitățile paginilor web. JavaScript rulează în browser-ul web al utilizatorului și permite manipularea și controlul elementelor HTML, interacțiunea cu utilizatorul și efectuarea de cereri către servere web. Acest limbaj permite adăugarea evenimentelor în paginile web, răspunzând la acțiunile utilizatorului (click-uri, apăsarea tastelor etc). JavaScript permite efectuarea de cereri către servere web, manipularea datelor și actualizarea paginii fără să blocheze interfața prin intermediul promisiunilor și API-urilor de gestionare a evenimentelor. Limbajul permite organizarea codului în module și funcții reutilizabile, ceea ce face ca aplicația să aibă o structură organizată și codul să fie ușor de gestionat. [6]

### **2.2.4. React.js**

React este o bibliotecă dezvoltată de Facebook, ce are la bază limbajul de programare JavaScript. React este considerat ca fiind o bibliotecă mai mult decât un limbaj și este renumit pentru dezvoltarea de aplicații web. Acesta a apărut în luna mai a anului 2013 și acum este una dintre cele

mai populare biblioteci în domeniul dezvoltării web. Biblioteca conține multe extensii pentru suportul arhitectural al întregii aplicații (Flux sau React Native), în afara de interfața de utilizare obișnuită.

Această bibliotecă a ajuns foarte populară astăzi comparativ cu altele din mai multe motive:

- Aplicații dinamice create ușor – React nu necesită mult cod și oferă mai multă funcționalitate, opus comparativ cu JavaScript, unde codul devine complex foarte rapid.
- Îmbunătățirea performanței – React utilizează Virtual DOM (Document Object Model) pentru crearea mai rapidă a aplicațiilor web; Virtual DOM face o comparație între stările anterioare ale componentelor și actualizează numai obiectele care au suferit schimbări, fără a le reactualiza pe toate.
- Reutilizarea componentelor – componentele sunt blocurile de construcție a oricărei aplicații React și o singură aplicație este compusă din mai multe componente. Acestea conțin parte logică și controale și pot fi utilizate de mai multe ori în aplicație, astfel reducând timpul petrecut pe dezvoltarea aplicației.
- Flux de date unidirecțional – la proiectarea unei aplicații React, în mod frecvent, dezvoltatorii leagă componentele copil de componentele părinte, astfel fluxul de date mergând într-o singură direcție și ușurând găsirea eventualelor erori acolo unde apare o problemă în aplicație.
- Ușor de învățat – biblioteca React combină elemente uzuale de HTML și concepte de JavaScript, adăugând și câteva beneficii, pe care dezvoltatorul trebuie să le studieze înainte de a începe să dezvolte o aplicație cu această bibliotecă.
- Unelte pentru depanare specifice – există o extensie Chrome lansată de Facebook care poate fi utilizată pentru a găsi mai ușor erorile dintr-o aplicație React, ceea ce facilitează procesul de depanare făcându-l mai rapid. [7]

#### **2.2.4.1. Sintaxa JSX**

Sintaxa JSX este o noțiune utilizată în biblioteca React care descrie în ce fel ar trebui să arate interfața utilizatorului. Utilizând aceeași sintaxă, pot fi scrise structuri HTML și cod JavaScript pentru funcționalități. Astfel, partea de JavaScript este responsabilă cu gestionarea evenimentelor și modul în care se schimbă stările, pe când partea de HTML se ocupă cu modul în care sunt afișate datele. [7]

#### **2.2.4.2. Virtual DOM**

Virtual DOM (Document Object Model) în React este o versiune abstractă de Real DOM. Real DOM este substanțial mai încet comparativ cu virtual DOM, întrucât real DOM reactualizează

stările tuturor obiectelor după modificare, pe când virtual DOM actualizează doar în obiectele care au suferit modificări. [7]

### 2.2.4.3. Componentele React

React separă interfața utilizatorului în mai multe componente, facilitând astfel găsirea erorilor, o componentă având propriul set de proprietăți și funcții.

Caracteristicile componentelor React:

- Reutilizarea – o componentă folosită într-un loc din aplicație poate fi folosită și în alte locuri, fapt ce ajută la diminuarea timpului în procesul de dezvoltare al aplicației
- Componente compuse – o componentă poate conține și alte componente
- Metoda de randare – este necesar ca o componentă să furnizeze o metodă de randare care descrie modul în care aceasta este afișată în DOM
- Proprietăți de trecere – componenta părinte poate trimite proprietăți (props) către componenta copil specificându-i valorile [8]

#### 2.2.4.3.1. Proprietăți (props)

Proprietățile React (props) permit utilizatorului să transfere datele între componente. Cu ajutorul acestora, componentele sunt mai dinamice. Aceste proprietăți sunt read-only și nu pot fi schimbate.

#### 2.2.4.3.2 Hooks

Hook-urile permit componentelor funcționale să aibă acces la stări și alte caracteristici React. Hook-urile folosite în aplicație sunt `useState`, `useEffect` și `useRef`.

1. **useState** – permite urmărirea stării în componentele funcționale. Starea, în general, se referă la date sau proprietăți care trebuie urmărite în aplicație. Acest hook acceptă o stare inițială și returnează două valori: starea curentă și o funcție care actualizează starea curentă.

Exemplu: `const [name, setName] = useState("");`

În acest exemplu, `useState` acceptă starea inițială și anume `""` și ulterior, dacă va fi apelată funcția `setName("andreea")`, starea curentă `name` va deveni `"andreea"`, datorită funcției `setName` care actualizează starea `name`. [8]

2. **useEffect** – permite executarea efectelor secundare (preluarea datelor, actualizări în DOM și timere) într-un anumit moment al ciclului de viață al componentei; `useEffect` acceptă două

argumente, al doilea fiind opțional: *useEffect(<funcție>, <dependință>)*.

Exemplu:

```
const [data, setData]=useState([])
useEffect(() => {
  fetchInfo();
}, []);
const fetchInfo = async () => {
  const resp = await fetch("/info");
  const info = await resp.json();
  setData(info)};
```

În exemplul de mai sus, utilizarea hook-ului *useEffect* cu al doilea argument *[]* asigură că efectul secundar *fetchInfo* (acesta face o solicitare către ruta „/info” și primește un răspuns sub formă de obiect JSON, apoi datele din acel obiect sunt salvate în starea componentei apelând funcția *setData* furnizată de hook-ul *useState*) este executat o singură dată, imediat după montarea componentei și actualizează starea acesteia cu datele primite de la ruta „/info”.  
[8]

3. **useRef** – permite persistența valorilor între randări; acesta poate fi utilizat pentru stocarea unei valori mutabile care nu cauzează re-randarea atunci când este actualizată și poate fi utilizat și pentru accesarea unui element din DOM în mod direct.

Exemplu:

```
const fileInputRef = useRef(null);
const [imageName, setImageName] = useState("");
const handleChooseFile = () => {
  fileInputRef.current.click();
};
const handleChangeFile = (e) => {
  const file = e.target.files[0];
  setImageName(file.name);
};
return (
  <input
    type="file"
```

```
accept="image/*"  
ref={fileInputRef}  
onChange={handleFileChange} />)
```

Acest exemplu folosește *useRef* pentru a declara o referință către elementul de input de tip fișier, inițializată cu *null* pentru a nu avea nicio valoare inițială. În funcția *handleChooseFile*, *useRef* este utilizat pentru a obține referința către elementul de input de tip fișier și pentru a apela metoda *click* care va deschide dialogul de selectare a fișierului, după care prin funcția *handleChangeFile* se extrage numele fișierului și se salvează într-o stare inițializată cu *useState*. [8]

### 2.2.5. Material-UI

Material-UI reprezintă o bibliotecă open-source de componente React care implementează design-ul materialelor Google. Această bibliotecă cuprinde o colecție vastă de componente predefinite care pot fi stilizate în funcție de preferințe. Există câteva motive pentru care dezvoltatorii aleg utilizarea acestei biblioteci:

- **Livrarea rapidă** – există o multitudine de componente care sunt deja implementate și doar trebuie importate în proiect, ele având deja un aspect plăcut, ceea ce diminuează timpul consumat pe stilizare.
- **Componente aspectuoase** – biblioteca are componente stilizate frumos care se potrivesc cu orice tip de aplicație și nu trebuie lucrat foarte mult la aspect, ci doar la poziționare.
- **Stilizare** – biblioteca permite stilizarea componentelor și utilizarea componentelor stilizate de fiecare developer, acest lucru fiind posibil utilizând elementul „*styled*”.

[9]

## 2.3. Backend

Partea de backend este responsabilă de gestionarea și procesarea datelor, rulând pe server și interacționând cu baza de date, servicii externe și oferind API-uri pentru interacțiunea cu clientul. Backend-ul este responsabil de funcționarea corectă și eficientă a aplicației, asigurând manipularea datelor în mod corespunzător și procesarea cererilor de la clienți conform regulilor și logicii aplicației. Tehnologiile utilizate pentru dezvoltarea acestei părți sunt Node.js pentru server și PostgreSQL pentru baza de date.

### 2.3.1. Node.js

Node.js reprezintă un mediu de execuție JavaScript care permite rularea codului JavaScript pe server, fără a fi executat doar pe browser. Acesta este bazat pe motorul V8 JavaScript care este utilizat de Google Chrome și alte browsere. Node.js funcționează într-un singur proces și rularea este asincronă și bazată pe evenimente, permițând server-ului să gestioneze mai multe cereri concurente într-un mod eficient fără a bloca execuția. De asemenea, acest mediu de execuție beneficiază de suportul *npm*, un suport de gestionare a pachetelor JavaScript care conține o colecție complexă de module și resurse.

La fel ca JavaScript, Node.js nu este greu de învățat și este implementat pe scară largă, având o mare comunitate de utilizatori activi. Node.js este potrivit pentru dezvoltarea aplicațiilor în timp real care nu necesită calcule grele pe partea de server, deoarece acest lucru ar putea bloca solicitările primite și ar degrada performanța generală.

În dezvoltarea aplicației au fost utilizate diferite metode HTTP cu rol în gestionarea interacțiunii cu resursele, utilizând framework-ul web Express.js:

- GET – această metodă se folosește pentru preluarea datelor de la server; când clientul face o cerere de acest tip către o anumită rută, serverul va returna resursa asociată rutei respective (exemplu: preluarea datelor din tabela de produse)
- POST – această metodă se folosește pentru trimiterea datelor către server pentru a fi prelucrate; este utilizată atunci când se adaugă o resursă nouă în server (exemplu: înregistrarea utilizatorilor)
- PUT – se folosește pentru actualizarea unei resurse existente pe server, ceea ce presupune înlocuirea resursei existente cu una nouă care este inclusă în corpul cererii PUT (exemplu: modificarea datelor unui utilizator)
- DELETE – se folosește pentru ștergerea unei resurse existente de pe server în momentul lansării unei cereri către o rută specifică (exemplu: ștergerea unui produs din coșul de cumpărături)

[10]



### 2.3.2. PostgreSQL

PostgreSQL este un sistem de gestionare a bazelor de date relaționale care este dezvoltat de Universitatea din California din cadrul Departamentului de Informatică Berkeley. PostgreSQL are câteva caracteristici cheie prin care în aplicația de gestionare a produselor electronice asigură un sistem rezistent de gestionare a bazelor de date care permite stocarea, interogarea și manipularea eficientă și sigură a datelor. Aceste caracteristici sunt:

- Structura relațională – permite definirea și utilizarea tabelor relaționale care sunt utilizate pentru stocarea datelor pentru produse, clienți și comenzi
- Suport pentru SQL – permite interogări complexe și manipularea eficientă a datelor
- Performanță și scalabilitate – PostgreSQL este optimizat pentru performanță și poate gestiona un volum mare de date
- Securitate – PostgreSQL oferă opțiuni avansate de securitate cum ar fi criptarea datelor care este esențială pentru protejarea datelor confidențiale ale utilizatorilor.

[11]

## 3. Implementarea aplicației

### 3.1. Structura fișierelor

Aplicația este construită având două foldere, unul pentru client și celălalt pentru server. Folderul destinat clientului conține partea de frontend a proiectului, adică o aplicație React creată folosind comanda *npx create-react-app client*. Folderul destinat server-ului conține serverul în care am instalat pachetul npm, folosind comenzile *npm init* și *npm install express*.

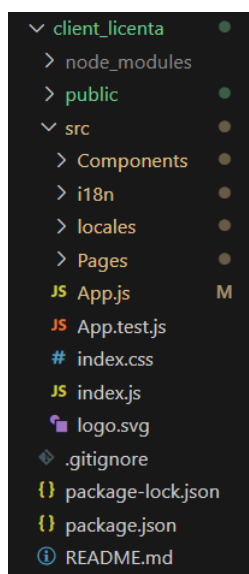


Fig. 3.1. – Structura fișierelor aplicației React

Aplicația de tip React conține un folder public și unul src. În folderul public sunt stocate imaginile randate în aplicație și fișierul index.html folosit de React pentru a afișa toate componentele sale. Folderul src conține mai multe subfoldere, unul pentru componente în care sunt componentele reutilizabile în mai multe locuri, unul pentru pagini în care sunt componentele pentru fiecare pagina din aplicație și folderele i18n și locales care conțin setări pentru traducerea paginilor în engleză și română. Folderul destinat părții de backend conține un fișier server.js unde este inclus codul pentru funcționalitățile precum realizarea conexiunii la baza de date, cererile HTTP, securitate etc.

## 3.2. Baza de date

Baza de date este realizată utilizând PostgreSQL. În baza de date există tabele pentru: utilizatori, produse, comenzi, obiecte\_comanda.

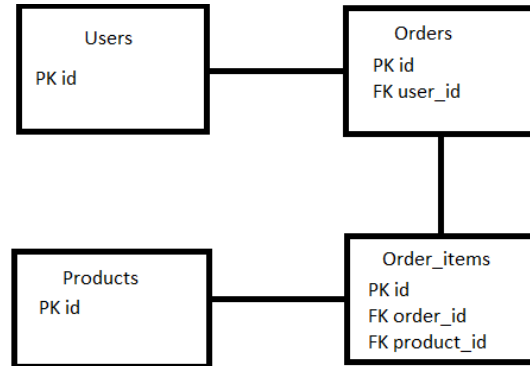


Fig. 3.2. – Reprezentarea relațiilor dintre tabelele din baza de date

În tabela pentru utilizatori (Users) sunt stocate date despre utilizatori precum id, nume, prenume, nume de utilizator, email, parolă, rol, coș de cumpărături și listă de dorințe. Ultimele două coloane stochează lista actualizată cu obiectele din coșul de cumpărături, respectiv lista de dorințe. Rolurile utilizatorilor sunt stocate într-un enum și au valori pentru administrator și client.

Tabela de produse (Products) conține date despre fiecare produs din aplicație: id, denumire, descriere, tipul produsului, marca, culoare, imagine, specificații. Coloanele pentru tipul produsului și marcă au valori setate din două enum-uri. Produsele pot fi de mai multe tipuri: telefoane, tablete, laptopuri, televizoare, accesorii (phone, tablet, laptop, tv, accessories). Mărcile produselor sunt stocate de asemenea într-un enum și au valorile: Apple, Samsung, Huawei, Asus, Dell, Sony, Lenovo.

Tabela de comenzi (Orders) conține date despre comenzile plasate de utilizatori, fapt pentru care conține cheia străină *user\_id* care face legătura cu tabela utilizatorilor. În afară de acest câmp, tabela conține câmpuri pentru prețul total al comenzii și adresa de livrare.

Tabela de obiecte\_comandă (Order\_items) stochează fiecare produs care a fost comandat. Aceasta are două chei străine, una pentru tabela de comenzi, *order\_id* și cealaltă pentru tabela de produse, *product\_id*. Această tabelă este utilă pentru afișarea fiecărui produs în cadrul fiecărei comenzi din pagina destinată comenzilor.

Conexiunea la baza de date este realizată în fișierul *server.js* prin instalarea modulului *pg*, specific pentru PostgreSQL. Un obiect numit *client* stochează datele utilizatorului, serverul local, portul acestuia și numele bazei de date, apoi conexiunea la baza de date este realizată prin *client.connect()*.

### 3.3. Conținutul fișierului principal

Fișierul principal al aplicației este App.js și aici sunt stocate rutele către componentele aplicației, funcționalitatea barei de căutare care este afișată în header și funcționalitatea pentru actualizarea numărului de elemente din coșul de cumpărături lângă iconița de coș.

Funcționalitatea pentru căutare este scrisă aici, deoarece acea bară de căutare primește ca input denumirea unui produs și trebuie să afișeze produsul indiferent care este pagina deschisă în acel moment. Astfel, elementul căutat primește valoarea din header unde este transmis ca proprietate.

Funcționalitatea pentru actualizarea numărului elementelor din coșul de cumpărături este scrisă în acest loc pentru că este nevoie să se actualizeze acel număr de fiecare dată când utilizatorul adaugă sau scoate produse din coșul de cumpărături și va fi trimisă ca proprietate pentru componentele Product, Products, Wishlist și Cart, deoarece din componenta fiecărui produs, din componenta de produse și din lista de dorințe se pot adăuga produsele în coș, iar din componenta coșului se pot șterge componente din coș dacă clientul se răzgândește în privința unui produs adăugat.



Fig. 3.3. – Header-ul și bara de navigare

```
function App() {
  const [searchedEl, setSearchEl] = useState("");

  const handleSearchSubmit = (searchTerm) => {
    const encodedSearchTerm = encodeURIComponent(searchTerm);
    window.location.href = `/products?search=${encodedSearchTerm}`;
  };

  const [badgeNr, setBadgeNr] = useState(0);

  const updateCart = async () => {
    try {
      const response = await fetch("/cart");
      const jsonData = await response.json();
      setBadgeNr(jsonData.cart_length);
    } catch (error) {
      console.error("Error fetching cart:", error);
    }
  };
};
```

```

return (
  <Router>
    <Header
      searchedEl={searchedEl}
      setSearchEl={setSearchedEl}
      handleSearchSubmit={handleSearchSubmit}
      badgeNr={badgeNr}
    />

    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/brands" element={<Brands />} />
      <Route
        path="/products"
        element={<Products updateCart={updateCart} />}
      />
      <Route
        path="/product/:id"
        element={<Product updateCart={updateCart} />}
      />
      <Route path="/promotions" element={<Promotions />} />
      <Route path="/support" element={<Support />} />
      <Route path="/wishlist" element={<Wishlist updateCart={updateCart />} />} />
      <Route path="/cart" element={<Cart updateCart={updateCart} />} />
      <Route path="/account" element={<Account />} />
      <Route path="/login" element={<Login />} />
      <Route path="/signup" element={<Signup />} />
      <Route path="/users" element={<Users />} />
      <Route path="/orders" element={<OrderHistory />} />
      <Route path="/users" element={<Users />} />
      <Route path="/adminProducts" element={<AdminProducts />} />
      <Route path="/terms_conditions" element={<TermsAndConditions />} />
      <Route path="/privacy_policy" element={<PrivacyPolicy />} />
    </Routes>
    <ScrollToTopButton />
    <Footer />
  </Router>
);
}

```

### 3.4. Selectarea limbii în aplicație

În partea de header a aplicației există un buton de selectare a limbii afișate și sunt disponibile două opțiuni de limbă: română și engleză. Pentru realizarea acestei funcționalități în clientul aplicației a fost instalat pachetul *react-i18next* care permite traducerea elementelor din aplicație prin utilizarea hook-ului *useTranslation*. Din pachetul *react-i18next* a fost instalat și pachetul *i18next* cu ajutorul căruia sunt setate opțiunile de limbă dorite.

```
const resources = {  
  en: {  
    translation: translationsInEng,  
  },  
  ro: {  
    translation: translationsInRo,  
  },  
};
```

```
i18n  
  .use(initReactI18next)  
  .init({  
    resources,  
    lng: "ro",  
    debug: true,  
    fallbackLng: "en",  
    interpolation: {  
      escapeValue: false,  
    },  
    ns: "translation",  
    defaultNS: "translation",  
  });
```

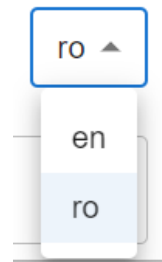


Fig. 3.4. –  
Selectarea limbii  
afișate

În obiectul *resources* sunt stocate cele două obiecte de tip JSON pentru cele două traduceri. Ambele obiecte au aceleași proprietăți, dar valoarea fiecărei proprietăți diferă în funcție de limba selectată (exemplu: *"wishlist": "Wishlist"*, *"wishlist": "Listă de dorințe"*). Limba setată standard este română (*lng*), iar în cazul în care aceasta nu ar putea fi afișată, va fi afișată limba engleză

(*fallbackLng*); *interpolation.escapeValue* este utilizată pentru a evita atacurile XSS(Cross-site scripting attacks – atacuri care injectează cod rău intenționat în site-uri web sigure de altfel), dar este setată ca fiind falsă pentru că React face deja acest lucru.

În restul codului, pentru a putea folosi traducerile, fiecare fișier are importat hook-ul *useTranslation* din pachetul *react-i18next* astfel:

```
const { t } = useTranslation();
```

În acest fel se extrage în mod specific doar metoda *t* din obiectul returnat de *useTranslation()*, această metodă fiind folosită pentru a traduce șirurile de text în limba dorită. Pentru a obține traducerea, se utilizează între tag-urile HTML *{t("wishlist")}* și astfel când limba selectată este engleză se va afișa *Wishlist*, iar pentru română va fi afișat *Listă de dorințe*.

## 3.5. Înregistrarea și autentificarea utilizatorilor

### 3.5.1. Înregistrare

Pagina destinată înregistrării utilizatorilor este proiectată în componenta Signup.js care redă un formular alcătuit din mai multe câmpuri pentru datele utilizatorului. În componentă, valorile câmpurilor sunt setate inițial ca fiind vide cu ajutorul hook-ului *useState* și preiau datele introduse de utilizator în formularul din pagină. Câmpurile trebuie să fie setate obligatoriu, așa că fiecare input are setată proprietatea *required*. Valorile câmpului trebuie să îndeplinească anumite condiții de scriere (numele și prenumele pot conține numai litere și caracterul „-” sau spațiu, numele de utilizator nu trebuie să aibă alte caractere în afară de litere și cifre, email-ul trebuie să respecte forma „username@example.com”, parola trebuie să conțină litere mici și mari, cifre și caractere speciale). Aceste validări au fost realizate utilizând expresii regulate și în funcția de validare a formularului se verifică dacă datele introduse de utilizator se potrivesc cu formatul cerut. Rolul utilizatorului este de client la crearea contului. Rolul de administrator poate fi atribuit doar din baza de date.

La apăsarea butonului de înregistrare se apelează funcția *submitSignup* care trimite datele la server prin metoda *POST* la ruta */signup*, unde se verifică dacă utilizatorul există deja prin compararea numelui de utilizator cu numele de utilizator existente deja în baza de date. În cazul în care utilizatorul nu există, acesta este adăugat în baza de date și autentificat în aplicație și datele sale sunt stocate în sesiunea curentă în obiectul „*user*”, în caz contrar, înregistrarea nu se efectuează, deoarece numele de utilizator este deja folosit în baza de date de altcineva.

```

const submitSignup = async (e) => {
  e.preventDefault();
  try {
    const body = { username, firstName, lastName, email, password };
    if (password === confirmPassword) {
      const response = await fetch("/signup", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(body),
      });
      const data = await response.json();
      if (data.loggedIn) {
        localStorage.setItem("user", JSON.stringify(data.user));
        navigate("/");
        window.location.reload(false);
      } else {
        alert(data.status);
      }
    } else {
      alert("Different passwords");
    }
  } catch (err) {
    console.error(err.message);
  }
};

```

În server datele introduse de utilizator sunt accesate inițializând un obiect care conține câmpurile din formular cu *req.body*.

```

const { username, firstName, lastName, email, password } = req.body;

```

Sunt introduse și aici teste pentru formatul acestor câmpuri pentru a nu putea introduce un altfel de format direct în baza de date. Testele sunt realizate în acest format:



```
const firstNameRegex = /^[A-Za-z\s-]+$/;
if (!firstName.match(firstNameRegex)) {
  return res
    .status(400)
    .json({ loggedIn: false, status: "Invalid first name" });
}
```

Codul de mai jos adaugă utilizatorul în baza de date dacă nu există deja un utilizator cu numele de utilizator introdus și îl adaugă în sesiunea curentă. Parola utilizatorului este criptată cu ajutorul pachetului *bcrypt* care crează un hash din string-ul introdus ca parolă. Astfel parola utilizatorului este salvată sub formă criptată în baza de date. La final este setat statusul utilizatorului ca fiind logat (*loggedIn = true*) și în *user* este salvat utilizatorul curent.

```
const existingUser = await client.query(
  `SELECT username from users where username=$1`,
  [username]
);
if (existingUser.rowCount === 0) {
  const hashedPassword = await bcrypt.hash(password, 10);
  const signupUser = await client.query(
    "INSERT INTO users (username, first_name, last_name, email, password)
values ($1, $2, $3, $4, $5) RETURNING *",
    [username, firstName, lastName, email, hashedPassword]
  );
  req.session.user = {
    id: signupUser.rows[0].id,
    username: signupUser.rows[0].username,
    first_name: signupUser.rows[0].first_name,
    last_name: signupUser.rows[0].last_name,
    email: signupUser.rows[0].email,
  };
  res.json({ loggedIn: true, user: signupUser.rows[0] });
}
```

### 3.5.2. Autentificare

Autentificarea utilizatorilor este realizată în componenta Login.js care este asemănătoare cu componenta paginii de înregistrare, însă conține doar două câmpuri în formular, unul pentru numele de utilizator și celălalt pentru parolă. Spre deosebire de înregistrare, în componenta de autentificare se face cerere de tip *POST* către ruta */login* din server așa cum s-a realizat și în componenta de înregistrare și serverul va compara datele introduse cu cele deja existente. Dacă există un utilizator cu aceste date, acesta este autentificat cu succes, setând proprietatea *loggedIn* să fie adevărată (*true*) și utilizatorul din sesiunea curentă să fie cel care tocmai s-a autentificat.

```
const potentialLogin = await client.query(
  "SELECT username, password, first_name, last_name, email, id, role FROM users
  WHERE username=$1",[username]);
if (potentialLogin.rowCount > 0) {
  const isSamePassword = await bcrypt.compare(password,
    potentialLogin.rows[0].password);}
```

Compararea numelui de utilizator cu cel din baza de date este realizată printr-o interogare din baza de date unde este selectat utilizatorul care are numele de utilizator introdus; dacă această interogare returnează un utilizator înseamnă că numele de utilizator este valid, în caz contrar, acesta nu există. Compararea parolei introduse de utilizator care este sub formă de string se realizează utilizând funcția *bcrypt.compare* care criptează parola introdusă și o compară cu cea existentă în baza de date pentru utilizatorul găsit anterior.

DeviceWorld

Cauta

Favorite Cos Contul Meu ro

PRODUSE MARCI SUPPORT

## Autentificare

Nume de utilizator \*

Parola \*

AUTENTIFICARE

Nu aveti cont? → [Inregistrare](#)

**Suport clienti**  
Formular de returnare a produselor  
E-mail: [deviceworld@support.com](mailto:deviceworld@support.com)  
Numar de telefon: 0725-412-78

**Contact**  
E-mail: [deviceworld@contact.com](mailto:deviceworld@contact.com)  
Numar de telefon: 021 / 9901

**Comunicare**  
Abonare la newsletter  
Termene si conditii  
Politica privata

**Urmareste-ne**  
f i n

Fig. 3.5. – Pagina de autentificare a aplicației

## 3.6. Produsele

### 3.6.1. Pagina de produse

Pagina de produse reprezintă o componentă esențială a acestei aplicații și permite vizualizarea, filtrarea, sortarea și adăugarea în coșul de cumpărături sau în lista de dorințe a produselor. Această pagină este realizată în componenta `Products.js` unde sunt preluate produsele din baza de date printr-o cerere de tip *GET* către ruta `/products` din server.

```
const getProducts = async (type, brand) => {
  try {
    let url = "/products";
    if (type) {
      url += `?type=${type}`;
      if (brand) {
        url += `&brand=${brand}`;
      }
    } else if (brand) {
      url += `?brand=${brand}`;
    }
    const response = await fetch(url);
    const data = await response.json();
    setProducts(data.products);
    console.log(colors);
    if (!type) {
      if (brand)
        setBrandOptions(data.brandOptions);
    }
  } catch (err) {
    console.error(err.message);
  }
};
```

În partea de server se realizează o interogare de tip *SELECT* din baza de date și sunt preluate produsele care fac parte din categoria și marca selectate. Dacă nu sunt selectate nicio categorie și nicio marcă, atunci sunt randate toate produsele în pagină. Tipul produselor poate fi ales din bara de navigare și în url-ul aplicației va fi adăugat *type=tip\_selectat*. Dacă sunt selectate categoria *telefon* și marca *Apple*, pagina va afișa doar telefoanele care au marca Apple. Dacă este selectată doar categoria *telefon* și marca nu este selectată, atunci se afișează toate telefoanele; similar, dacă este aleasă marca *Apple* din pagina cu toate produsele, atunci sunt randate doar produsele mărcii respective.

De asemenea, în pagina de produse există mai multe filtre care facilitează găsirea produsului dorit. Produsele pot fi ordonate crescător sau descrescător în funcție de preț sau denumire și acestea pot fi afișate și în funcție de intervalul de preț dorit de fiecare utilizator. Aceste funcționalități sunt construite în partea de client a aplicației și compară câmpurile existente ale produselor cu ceea ce dorește utilizatorul. Funcția *filterProducts* conține toate filtrele din această pagină și este apelată într-un hook de tip *useEffect* unde se actualizează pagina cu produsele filtrate. Butonul de resetare a filtrelor anulează filtrele selectate de utilizator și vor fi afișate toate produsele din pagină, acest lucru fiind realizat cu ajutorul funcției *handleResetFilters*.

```
const filterProducts = () => {
  let filteredProducts = products;
  // ...
  if (selectedBrand) {
    filteredProducts = filteredProducts.filter(
      (product) => product.brand.toLowerCase() === selectedBrand.toLowerCase()
    );
  }
  if (selectedPriceRanges.length > 0) {
    filteredProducts = filteredProducts.filter((product) => {
      for (const range of selectedPriceRanges) {
        const [min, max] = range.split("-");
        if (
          product.price >= parseInt(min) &&
          product.price <= parseInt(max)
        ) {
          return true;
        }
      }
      return false;
    });
  }
  setFilteredProducts(filteredProducts);
};

const handleResetFilters = () => {
  setSortOrder("asc");
  setSortField("");
  setSelectedBrand("");
  setSelectedPriceRanges([]);
  setResetFilters(!resetFilters);
};

useEffect(() => {
  filterProducts();
}, [
```

```
products,  
sortField,  
sortOrder,  
searchQuery,  
selectedBrand,  
selectedPriceRanges,  
resetFilters]);
```

Pentru afișarea produselor în pagină s-a realizat o mapare a vectorului de produse și s-a afișat fiecare produs sub formă de listă stilizată (s-au stilizat componentele *List* și *ListItem* din biblioteca Material-UI pentru a fi setat background-ul și border-ul fiecărui produs). Fiecare produs conține butonul de adăugare în coș și inima din colțul stâng care este un buton ce adaugă produsele în lista de dorințe. Aceste butoane sunt dezactivate dacă utilizatorul nu este autentificat astfel că nu se pot salva sau cumpăra produse fără autentificare. De asemenea, atunci când se face click pe imaginea produsului, va fi randată pagina produsului respectiv.

### 3.5.1.1. Adăugarea în coșul de cumpărături

Pentru adăugarea produsului dorit în coșul de cumpărături, utilizatorul trebuie să apese pe butonul de adăugare în coș care va prelua id-ul, denumirea, prețul, imaginea, tipul produsului, marca și va adăuga produsul în coș setându-i cantitatea 1 prima dată. Această operațiune se va efectua printr-o cerere de tip *POST* către ruta */cart/add* din server, clientul furnizând datele de mai sus.

În server aceste date sunt accesate cu *req.body* și apoi se testează dacă utilizatorul este autentificat în sesiunea curentă. În caz afirmativ, coșul de cumpărături din tabela utilizatori se populează cu datele produsului adăugat, în caz contrar, nu se va putea realiza adăugarea întrucât utilizatorul nu este autentificat. Coșul de cumpărături din baza de date este de tipul *JSONB[]* și dacă este gol inițial are valoarea *null*, așa că trebuie mai întâi convertit să fie vector. De asemenea, tot aici se testează dacă produsul exista deja în coș, iar dacă există, acesta nu mai este adăugat a doua oară.

```
if (!req.session.user) {  
  return res.status(401).json({ error: "Unauthorized" });  
}  
const userId = req.session.user.id;  
const userCart = await client.query(  
  "SELECT cart FROM users WHERE id = $1",  
  [userId]  
);  
let currentCart = userCart.rows.length > 0 ? userCart.rows[0].cart : [];  
if (!Array.isArray(currentCart)) {
```

```

    currentCart = [];
  }
  const itemIndex = currentCart.findIndex(
    (item) => item.productId === productId
  );
  if (itemIndex !== -1) {
    return res.json({ message: "Product already in the cart." });
  }
  const updatedCart = [...currentCart, cartItem];
  await client.query("UPDATE users SET cart = $1 WHERE id = $2", [
    updatedCart,
    userId,
  ]);
  res.json({ message: "success" });

```

### 3.5.1.2. Adăugarea unui produs în lista de dorințe

Adăugarea unui produs în lista de dorințe se face în mod similar cu adăugarea produsului în coșul de cumpărături cu mențiunea că aici nu este adăugată și cantitatea produsului. Câmpurile din tabela de utilizatori pentru coș și lista de dorințe sunt similare, deci procedura de adăugare este aceeași.

### 3.6.2. Pagina fiecărui produs

Pagina pentru fiecare produs apare atunci când utilizatorul apasă pe imaginea produsului din pagina cu toate produsele, pe denumirea unui produs din lista de dorințe sau din coșul de cumpărături. Pagina este construită în componenta Product.js și aici se randează produsul selectat în funcție de id-ul său, împreună cu toate detaliile despre acesta din baza de date.

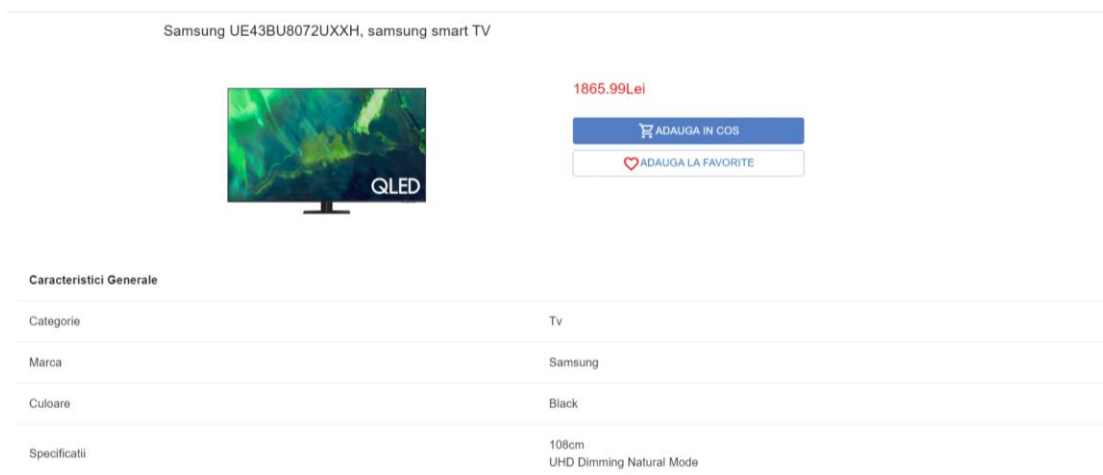


Fig. 3.6. – Pagina unui produs din aplicație

Așa cum se observă și în imagine, pagina produsului conține imaginea produsului împreună cu prețul acestuia sub care se află butoanele de adăugare în coș sau în lista de dorințe. Tot aici se găsește și tabelul cu toate caracteristicile produsului, astfel că utilizatorul poate vedea toate detaliile acestuia. Pentru a reda datele despre produs, în componenta `Product.js` s-a realizat o cerere de tip *GET* pentru ruta `/product/id`, unde `id`-ul produsului era trimis ca parametru al funcției. În partea de server cererea a fost prelucrată selectând din baza de date produsul cu `id`-ul respectiv și returnându-l sub forma unui obiect în format JSON.

```
const id = req.params.id;
const query = "SELECT * FROM products WHERE id = $1";
const result = await client.query(query, [id]);

if (result.rows.length === 0) {
  return res.status(404).json({ error: "Product not found" });
}
const product = result.rows[0];
res.json({ product });
```

### 3.7. Coș de cumpărături

Coșul de cumpărături conține produsele adăugate de către utilizatori prin apăsarea butonului de adăugă în coș din pagina de produse, din pagina unui produs sau din lista de dorințe. Atunci când sunt adăugate în coș produsele au cantitatea 1, dar dacă utilizatorul vrea să ia mai multe bucăți, acest lucru este posibil din coșul de cumpărături unde poate adăuga mai multe bucăți ale produsului respectiv.

Schimbarea numărului de bucăți este posibilă datorită cererii *PUT* realizată către ruta `/cart/update/productId` care actualizează cantitatea setată de utilizator în coș. `Id`-ul produsului este transmis ca parametru al funcției și preluat de server prin `req.params`. Cantitatea este și ea transmisă ca parametru al funcției însă este preluată de server prin `req.body`.

```
await client.query("UPDATE users SET cart = $1 WHERE id = $2", [
  currentCart,
  userId,
]);

const response = await fetch(`/cart/update/${productId}`, {
  method: "PUT",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({ quantity }),
```

```
});
if (response.ok) {
  fetchCart();
}
```

În cazul în care acesta se răzgândește și nu dorește să mai achiziționeze un produs, îl poate șterge din coșul de cumpărături apăsând butonul de ștergere. Acest lucru este posibil, deoarece în client s-a realizat o cerere de tip *DELETE* către ruta */cart/remove/id* prin care produsul va fi șters din câmpul coș al respectivului utilizator. După ștergere se vor afișa produsele din coș actualizate și numărul lor din header va scădea.

```
const response = await fetch(`/cart/remove/${productId}`, {
  method: "DELETE",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({ quantity: 1 }),
});
if (response.ok) {
  fetchCart();
  updateCart();
}
```

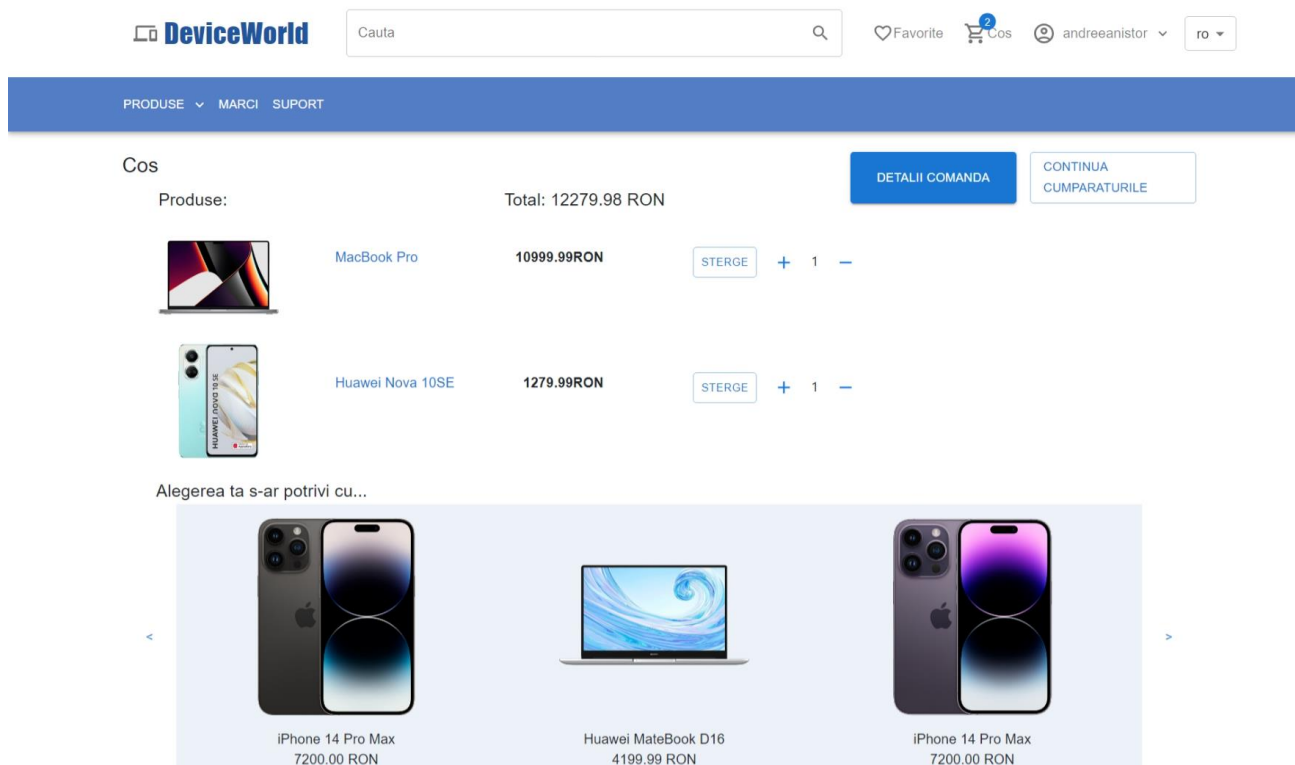


Fig. 3.7. – Pagina coșului de cumpărături



Coșul de cumpărături conține, de asemenea, prețul total al produselor, recomandări de produse care s-ar potrivi cu cele adăugate în coș și două butoane, unul pentru a introduce detaliile comenzii și altul pentru a reveni la pagina de produse.

- Prețul total al produselor este calculat în funcție de prețul fiecărui produs și numărul de bucăți din coș pentru fiecare produs.
- Recomandările pentru alte produse similare sunt o sugestie pentru utilizator de a cumpăra și alte produse din același ecosistem. Pentru realizarea acestei funcționalități s-au comparat produsele din coș cu toate celelalte produse și s-au randat doar cele care au aceeași marcă, dar nu sunt din aceeași categorie, astfel că dacă un client dorește un iPhone, magazinul îi recomandă și alte produse Apple (căști, laptopuri etc).
- Butonul pentru introducerea detaliilor comenzii va afișa câmpul adresă atunci când este apăsat, iar utilizatorul va trebui să adauge adresa de livrare ca să poată fi plasată comanda respectivă. După plasarea comenzii, se va afișa un mesaj de confirmare a efectuării operațiunii.



The image shows a user interface for order details. It consists of a blue button labeled 'DETALII COMANDA' and a white button with a blue border labeled 'CONTINUA CUMPARATURILE'. Below these is a white input field with the placeholder text 'Adresa \*'. At the bottom is a large blue button labeled 'PLASEAZA COMANDA'.

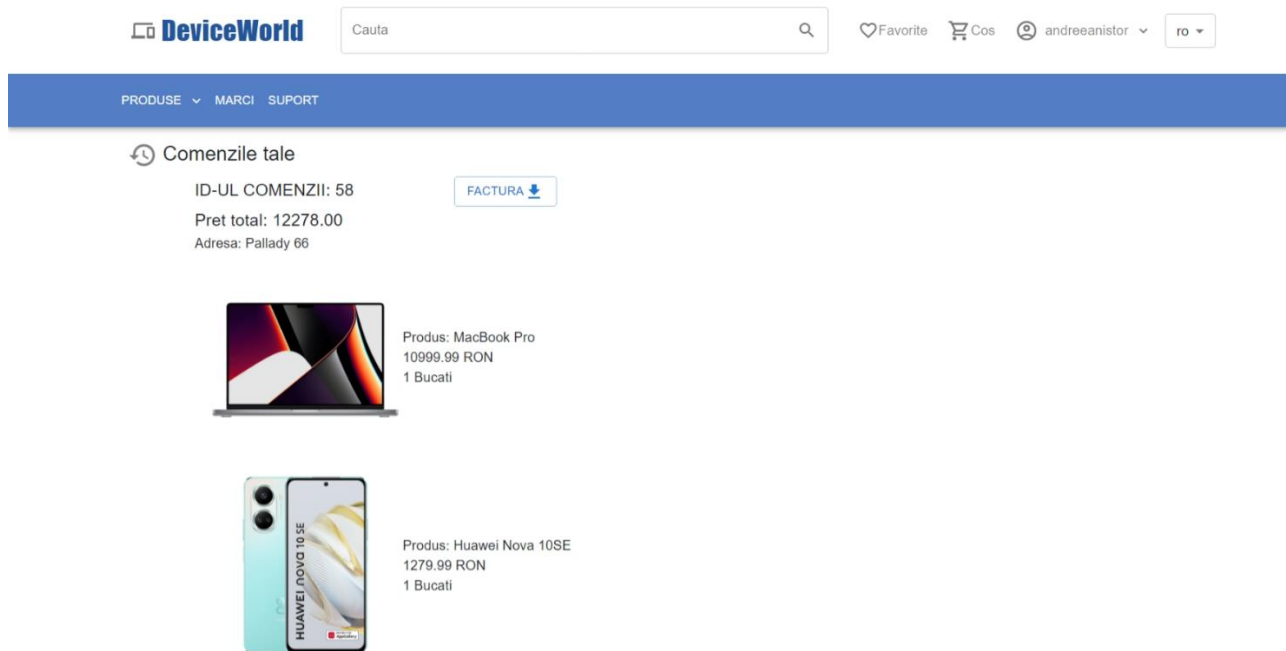
Fig. 3.8. – Detaliile comenzii produselor din coșul de cumpărături

### 3.8. Comenzi

După plasarea comenzilor, acestea vor fi afișate într-o pagină destinată acestora. Componenta din partea de client responsabilă de această parte este OrderHistory.js.

În această componentă se face o cerere de tip *GET* către ruta */orders* din server unde sunt preluate datele despre toate comenzile din baza de date. Se compară id-ul utilizatorului autentificat în sesiunea curentă cu id-urile din tabela de comenzi (*orders*) din baza de date, după care se vor prelua comenzile utilizatorului respective.

Atunci când se plasează comanda se adaugă în tabela *orders* datele comenzii (id-ul utilizatorului, prețul total și adresa) și se populează și tabela *order\_items* cu date despre produse. Astfel se randează în pagina comenzilor atât detaliile comenzii cât și produsele achiziționate.



Figură 3.9. – comenzile utilizatorului autentificat

Din pagina de comenzi se poate descărca factura pentru fiecare comandă în format PDF. Pentru a genera factura în acest format a fost instalat pachetul *react-pdf/renderer*, a fost afișat documentul cu ajutorul componentelor *Document*, *Page*, *Text*, *Image*, *PDFDownloadLink* și a fost stilizat folosind *Stylesheet*.

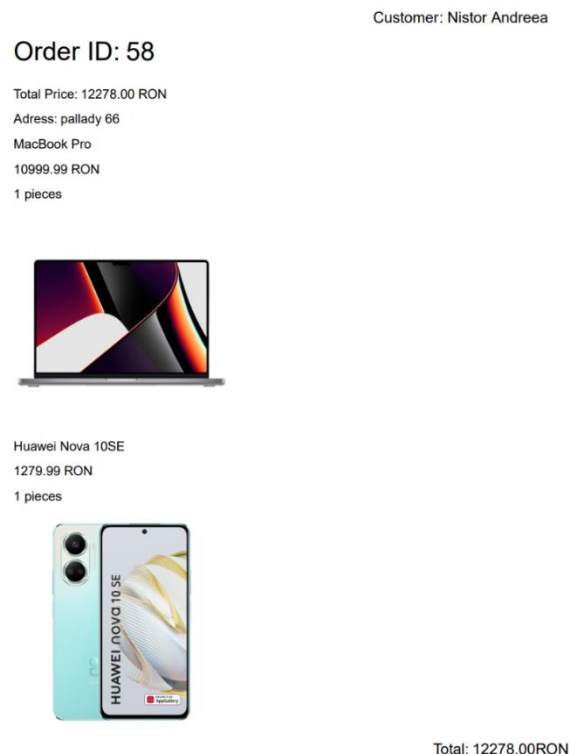


Fig. 3.10 – Factura unei comenzi în format PDF

### 3.9. Listă de dorințe

Pagina listei de dorințe este construită în componenta Wishlist.js și are funcționalități pe care le conține și coșul de cumpărături, mai exact afișarea produselor adăugate în lista de către utilizator, opțiunea de ștergerea unui produs din listă și de a adăuga produsul dorit în coșul de cumpărături.

Afișarea produselor în această pagină se face printr-o cerere de tip *GET* către ruta */wishlist* din server, unde cu o interogare de tip *SELECT* sunt preluate obiectele din lista de dorințe ale utilizatorului autentificat în sesiunea curentă. Ștergerea unui produs din lista de dorințe este realizată similar cu ștergerea unui produs din coșul de cumpărături, adică printr-o cerere de tip *DELETE* către server care va șterge elementul dorit din coloana asociată listei de dorințe din tabela utilizatori a utilizatorului curent.

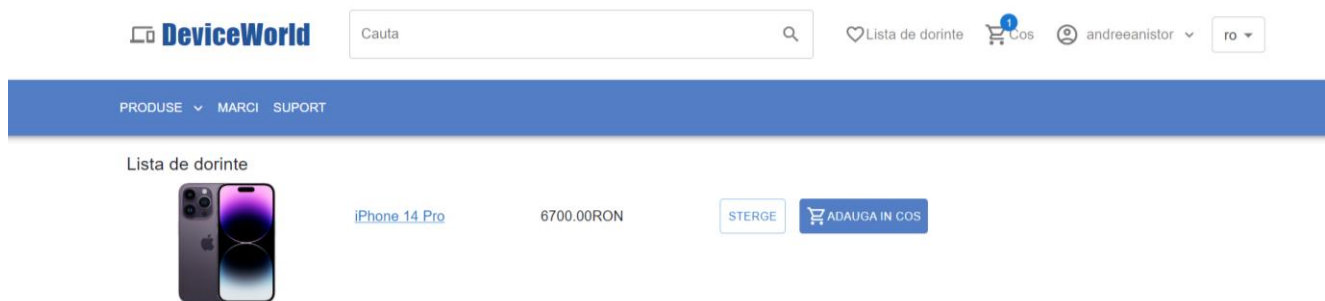


Fig. 3.11. – Un produs adăugat în lista de dorințe

### 3.10. Gestionarea produselor

Gestionarea produselor este realizată de către utilizatorii care au rolul de administrator. Aceștia vizualizează un tabel cu toate produsele, unde pot efectua modificări asupra datelor produselor sau pot șterge un produs din tabel. De asemenea, în această pagină există și un buton pentru adăugarea unui produs nou în tabel. Toate aceste operații se efectuează și în baza de date. Condiția care se aplică pentru ca operațiile să fie efectuate doar de către administratori este testarea în server a condiției ca utilizatorul autentificat în sesiune să dețină rolul de administrator.

```
req.session && req.session.user && req.session.user.role === "admin"
```

### 3.10.1. Adăugarea unui produs nou

Pentru a adăuga un produs nou în lista de produse, administratorul va apăsa pe butonul de adăugare din pagina destinată gestionării de produse, după care va apărea un dialog cu formularul ce conține toate câmpurile necesare adăugării unui produs în baza de date.

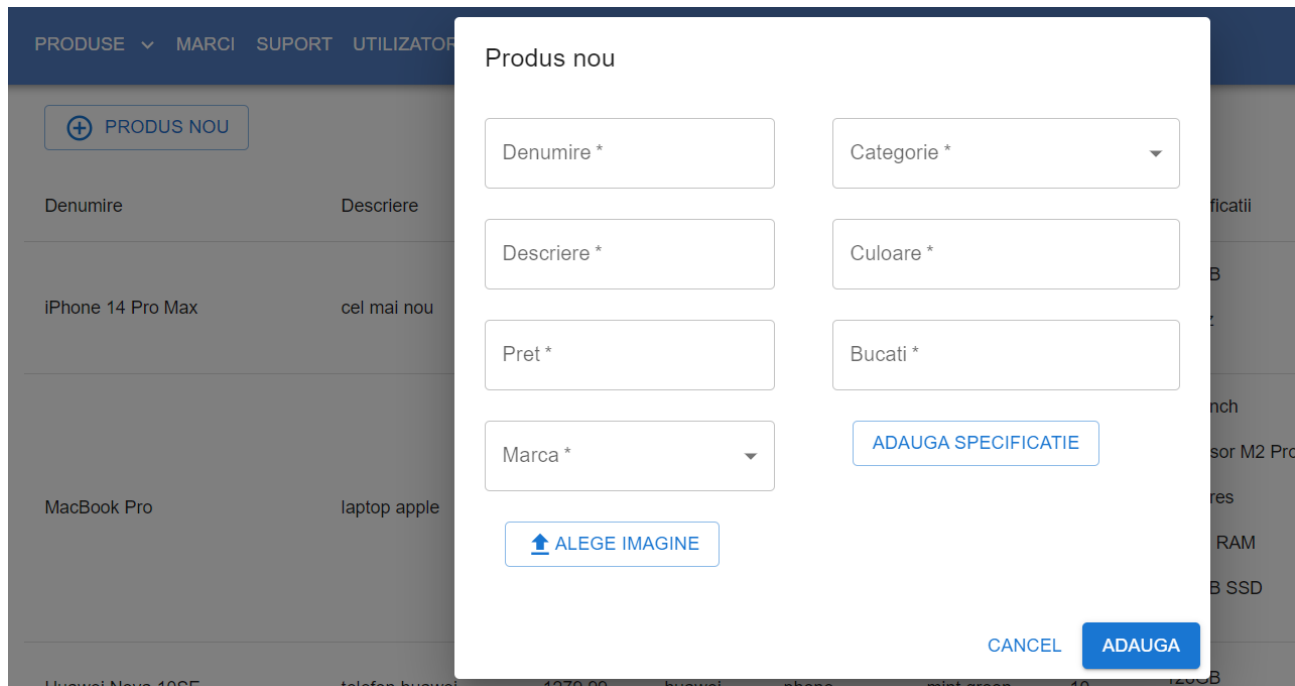


Fig. 3.12. – Formularul pentru adăugarea unui produs

În partea de client s-a realizat o cerere de tip *POST* pentru a trimite datele introduse de utilizator către server. Pentru a efectua operația de adăugare a unui produs în baza de date, în server s-a realizat inserarea datelor preluate din formular prin *req.body* cu ajutorul interogării de inserare în tabela de produse.

### 3.10.2. Modificarea datelor unui produs

În tabelul care conține toate produsele există câte un buton de editare pe fiecare linie astfel că administratorul poate modifica datele despre fiecare produs direct din tabel. După ce a terminat modificările acestea se salvează la apăsarea butonului de confirmare.

Funcționalitatea de modificare a produselor este realizată printr-o cerere de tip *PUT* către ruta */manageProducts/id* în care se furnizează datele actualizate și id-ul produsului către server (accesibil prin *req.params*.), unde se testează din nou dacă utilizatorul din sesiunea curentă are rolul de administrator și în caz afirmativ, printr-o interogare de actualizare a tabelului de produse, sunt setate

noile date care au fost preluate cu *req.body*, la fel ca în cazul adăugării unui produs nou în lista de produse.

```
app.put("/manageProducts/:id", async (req, res) => {
  try {
    if (req.session && req.session.user && req.session.user.role === "admin") {
      const { id } = req.params;
      const { name, description, price, brand, product_type, color, quantity } =
        req.body;
      const updatedProduct = await client.query(
        "UPDATE products SET name = $1, description = $2, price = $3, brand = $4,
product_type = $5, color = $6, quantity = $7 WHERE id = $8 RETURNING *",
        [name, description, price, brand, product_type, color, quantity, id]
      );
      res.json(updatedProduct.rows[0]);
    }
  }
});
```

### 3.10.3. Ștergerea unui produs

Operația de ștergere a unui produs din baza de date se efectuează tot din acest tabel, apăsând butonul care conține iconița de coș de gunoi. Pentru a nu șterge din greșeală produsele, va apărea o alertă care cere confirmarea intenției de ștergere a produsului. Ștergerea unui produs este realizată printr-o cerere de tip *DELETE* din partea clientului către ruta *manageProducts/remove/id*, unde *id*-ul produsului este trimis ca parametru și preluat de către server prin *req.params*. În server se testează dacă utilizatorul are drept de ștergere, adică dacă are rolul de administrator, după care se verifică faptul că există produsul respectiv în listă și dacă există, se efectuează ștergerea printr-o interogare de ștergere a produsului în baza de date, unde produsul trebuie să aibă *id*-ul furnizat din rută. Dacă nu există, este returnată eroarea *404* care atenționează că produsul selectat nu există în listă.

```
if (req.session && req.session.user && req.session.user.role === "admin") {
  const { id } = req.params;
  const product = await client.query("SELECT * FROM products WHERE id = $1",
[id]);
  if (product.rows.length === 0) {
    return res.status(404).json({ error: "Product not found in the list" });
  }
  await client.query("DELETE FROM products WHERE id=$1", [id]);
}
```

### 3.11. Gestionarea utilizatorilor

Gestionarea utilizatorilor este realizată de către utilizatorii care au rolul de administrator, la fel ca în cazul gestionării produselor. Administratorul are dreptul de a modifica datele utilizatorului și de a șterge utilizatori din listă. Ambele operațiuni sunt realizate similar cu operațiile de modificare și ștergere a produselor.

- Modificarea datelor unui utilizator este permisă doar dacă utilizatorul autentificat are rolul de administrator și se efectuează prin trimiterea unei cereri a clientului de tip *PUT*, unde sunt furnizate datele actualizate, acestea fiind preluate de către server prin *req.body*, după care se efectuează actualizarea în baza de date setând aceste date.
- Ștergerea datelor unui utilizator se realizează verificând că utilizatorul din sesiunea curentă are rolul de administrator, după care se șterge utilizatorul selectat din tabelă. Această operațiune se efectuează printr-o cerere de tip *DELETE* către ruta */users/remove/id*, unde se furnizează id-ul utilizatorului selectat și serverul preia acest id prin *req.params*. urmând să șteargă utilizatorul din baza de date prin interogarea de ștergere în tabela de utilizatori.

### 3.12. Aplicație responsive

Aplicația respectă conceptul de *responsive*, adaptându-se la dimensiunea dispozitivului utilizat. Această adaptare s-a realizat în majoritatea componentelor, pentru ecran mic fiind construită o altfel de așezare aliniere în pagină.

Pentru a realiza design-ul pe ambele tipuri de ecrane, în componenta părinte care conține așezarea în pagină a elementelor pe ecran mic am introdus în partea de stiluri următorul comportament: *{display: {xs: "flex", md: "none"}}*, unde *xs* și *md* sunt punctele de întrerupere care corespund diferitelor dimensiuni ale ecranului, fiecare având asociată o valoare specifică pentru proprietatea *display*. Astfel, conținutul componentei este afișat cu *display: flex* pe ecranele mici și cu *display: none* pe ecranele medii. Pentru stilizarea componentelor pe ecran mediu și mare, valorile celor două puncte de întrerupere se inversează astfel *{display: {xs: "none", md: "flex"}}*.

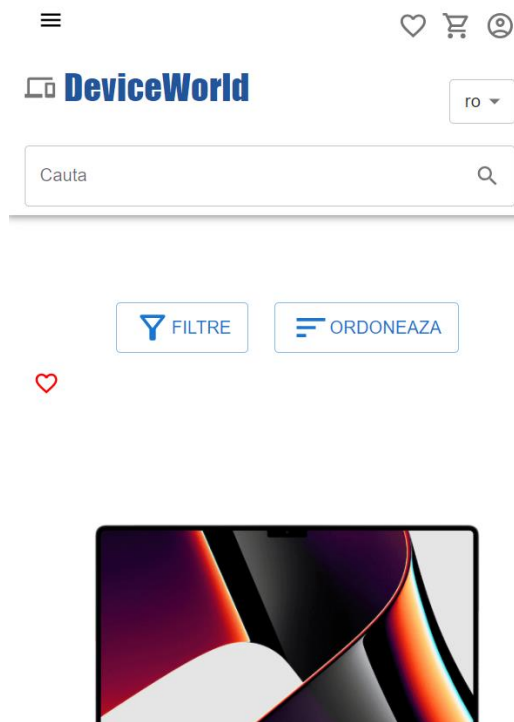


Fig. 3.13. – Design-ul pe ecran mic

Pe ecran mic, elementele au fost așezate în pagină într-un mod compact. Pentru implementarea barei de navigare pe acest tip de ecran este mai potrivită alegerea unui meniu de tip hamburger care are același conținut ca bara de navigare, iar pentru butoanele de coș, cont și listă de dorințe au fost lăsate doar iconițele sugestive pentru a nu încărca pagina.



Fig. 3.14. – Meniul de tip hamburger

Pe ecran mediu și mare, pentru a așeza elementele în pagină într-un mod vizual plăcut, acestea au fost incorporate în componente de tip Grid din biblioteca Material-UI care oferă paginii un aspect ordonat și ușor de urmărit.

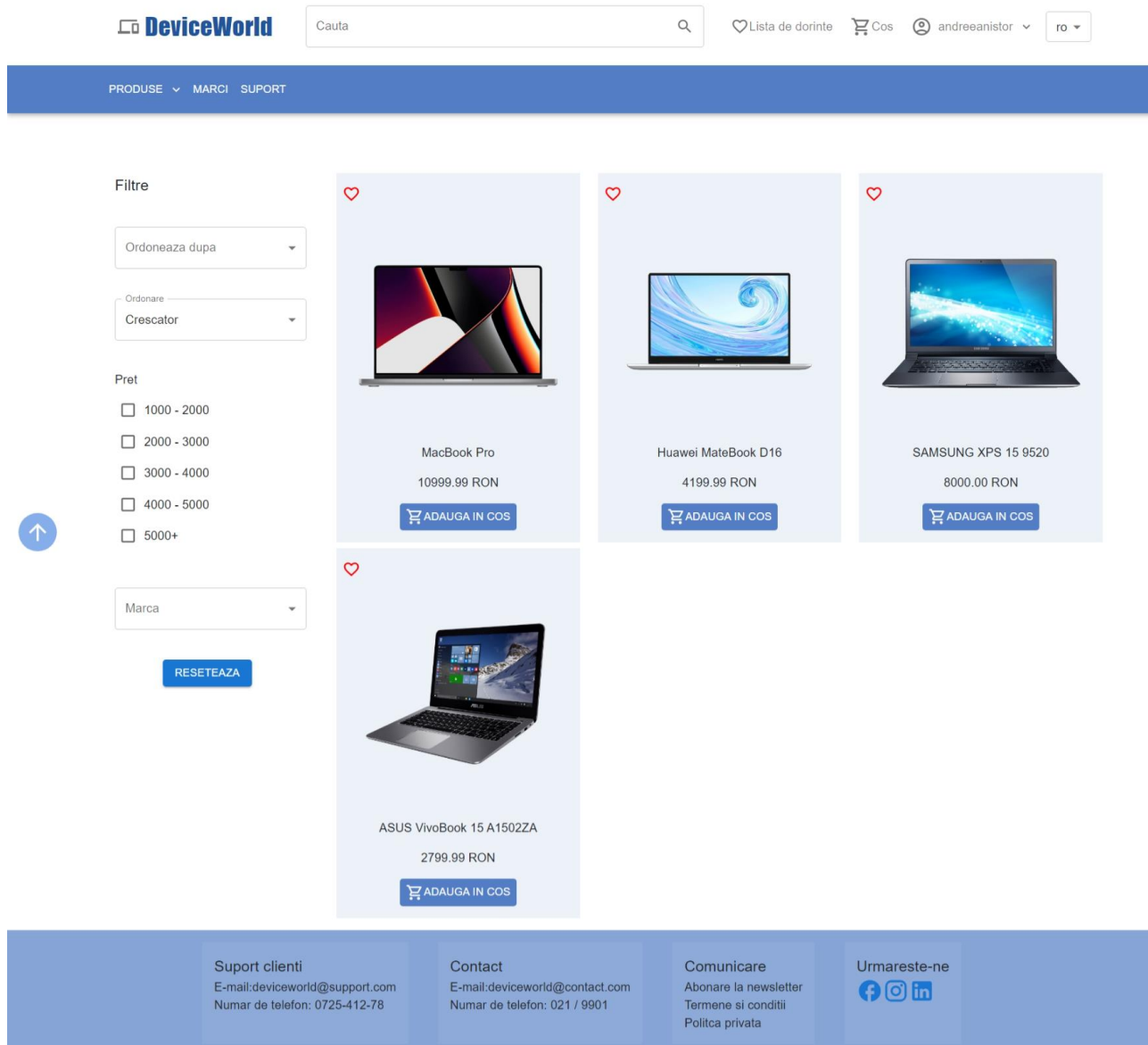


Fig. 3.15. – Design-ul pe ecran mare



## 4. Ghid de utilizare

### 4.1. Înregistrarea și autentificarea

În primul rând, utilizatorul trebuie să se conecteze cu un cont în aplicație. Pentru acest lucru, în partea de sus a aplicației va selecta butonul „Contul Meu” care deschide un meniu de unde are opțiunea de autentificare sau înregistrare. Dacă are deja un cont, va selecta „Autentificare” și va fi redirecționat către pagina destinată acestei acțiuni, unde va trebui să introducă numele de utilizator și parola aferente contului său. În cazul în care nu a mai folosit aplicația, utilizatorul va trebui să creeze un cont, iar pentru acest lucru va selecta butonul „Înregistrare” care îl va conduce către pagina destinată acestui lucru. În pagina de înregistrare va trebui să completeze toate câmpurile formularului pentru a putea ulterior să apese pe butonul care îi va crea contul.

DeviceWorld

Cauta

Lista de dorinte Cos Contul Meu ro

PRODUSE MARCI SUPPORT

## Creeaza un cont

Aveti deja un cont? → [Autentificare](#)

Nume de utilizator \*

Prenume \*

Nume \*

E-mail \*

Parola \*

Confirmare parola \*

INREGISTRARE

Support clienti  
Formular de returnare a produselor  
E-mail: [deviceworld@support.com](mailto:deviceworld@support.com)  
Numar de telefon: 0725-412-78

Contact  
E-mail: [deviceworld@contact.com](mailto:deviceworld@contact.com)  
Numar de telefon: 021 / 9901

Comunicare  
Abonare la newsletter  
Termene si conditii  
Politica privata

Urmareste-ne

Fig. 4.1. – Pagina de înregistrare

## 4.2. Vizualizarea produselor

Pentru a vizualiza produsele, utilizatorul trebuie să aleagă din bara de navigare „Produse” și să selecteze toate produsele sau produse dintr-o anumită categorie. Pagina cu produse va fi afișată și aici va putea să ordoneze produsele în funcție de preț sau denumire și să le filtreze în funcție de prețul dorit sau marca produselor.

Dacă utilizatorul intră pe site de pe un dispozitiv cu ecran mare, filtrele se vor afișa în stânga produselor și produsele vor fi afișate câte 3 pe linie.

Dacă utilizatorul folosește un dispozitiv cu ecran mic, produsele vor apărea unul sub altul și la început apar două butoane: unul pentru filtre și celălalt pentru ordonarea produselor; fiecare dintre cele două butoane afișează câte un dialog în care utilizatorul își va seta preferințele, după care le va salva pentru a se randa produsele dorite în pagină.

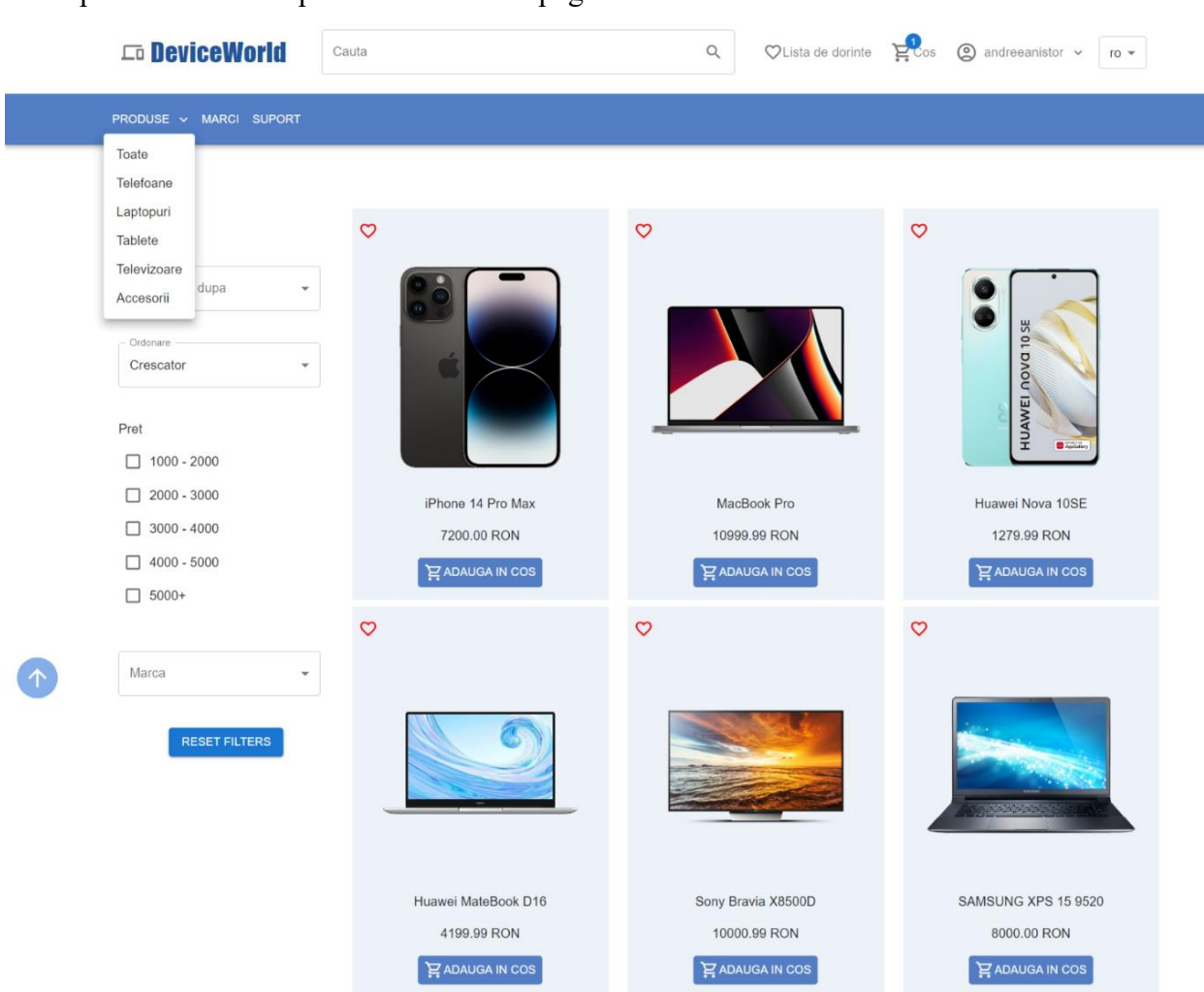


Fig. 4.2. – Pagina de produse pentru ecran mare

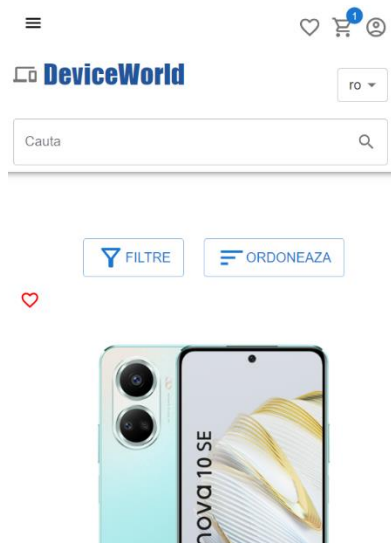


Fig. 4.3 – Pagină produse ecran mic

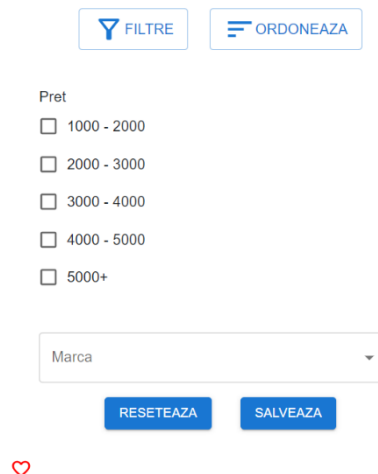


Fig. 4.4 – Dialogul de filtre pe ecran mic

Dacă vrea să vizualizeze mai multe informații despre un produs, utilizatorul poate apăsa pe imaginea produsului și este redirecționat către pagina produsului respectiv unde se află toate detaliile despre acesta. Dacă unul sau mai multe produse i-au atras atenția, utilizatorul le poate adăuga în coșul de cumpărături apăsând butonul „Adaugă în coș” din dreptul produsului respectiv sau în lista de dorințe apăsând pe inima din dreptul produsului ales.

### 4.3. Coșul de cumpărături

Dacă utilizatorul vrea să achiziționeze produsul sau produsele adăugate în coș la pasul anterior, acesta va trebui să apese pe butonul cu iconița de coș din partea de sus a aplicației și în acea pagină vor fi afișate produsele adăugate în coș. În această pagină utilizatorul va mai primi și niște recomandări de produse care s-ar potrivi cu ce este deja în coș. Dacă prezintă interes, poate apăsa pe imaginea produsului respectiv și va fi condus către pagina corespunzătoare acelui produs.

Dacă s-a răzândit în privința unui produs din coș, acesta poate fi șters apăsând butonul „Șterge”. Dacă dorește să cumpere mai multe produse de același fel, poate crește numărul de bucăți apăsând pe butonul + sau dacă a selectat prea multe bucăți, poate scădea din cantitate apăsând pe butonul –.

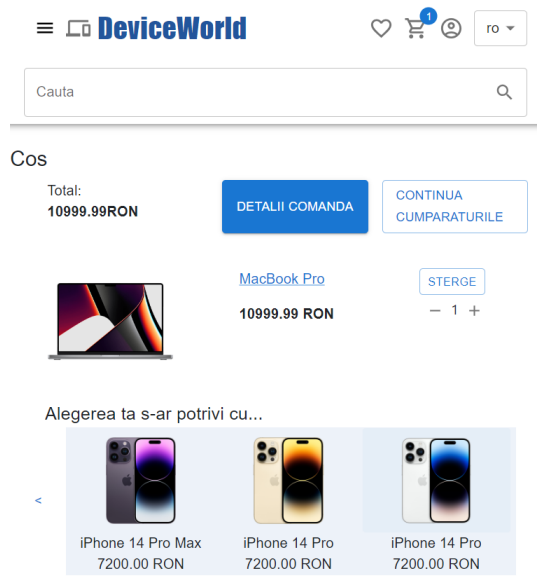


Fig. 4.5. – Coșul de cumpărături pe ecran mic

Dacă utilizatorul dorește să plaseze comanda trebuie să apese pe butonul de „Detalii comandă” care va afișa un câmp în care trebuie să introducă adresa unde vor fi livrate produsele, după care comanda fi efectuată la apăsarea butonului de „Plasează comanda”.

## 4.4. Lista de dorințe

Dacă utilizatorul a salvat anumite produse în lista de dorințe acestea pot fi vizualizate accesând pagina specifică prin apăsarea butonului care are iconița de inimă. Produsele din această pagină pot fi șterse prin apăsarea butonului sugestiv în cazul în care nu mai prezintă interes pentru utilizator sau pot fi adăugate în coș dacă utilizatorul se hotărăște să le cumpere.

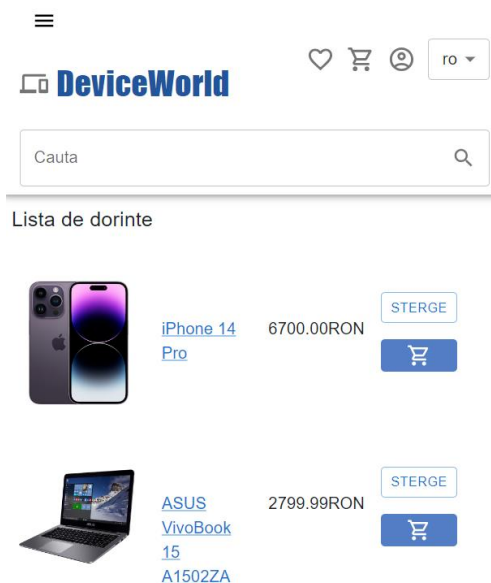


Fig. 4.6. -Lista de dorințe

## 4.5. Vizualizarea comenzilor

Dacă utilizatorul dorește să vadă detalii despre comenzile anterioare, el poate apăsa pe butonul care are iconița de cont și se va deschide un meniu ce conține și butonul „Comenzile mele” care îl va duce pe pagina cu toate comenzile. În dreptul fiecărei comenzi apare un buton pentru descărcarea facturii. Aceasta se va descărca în format PDF.

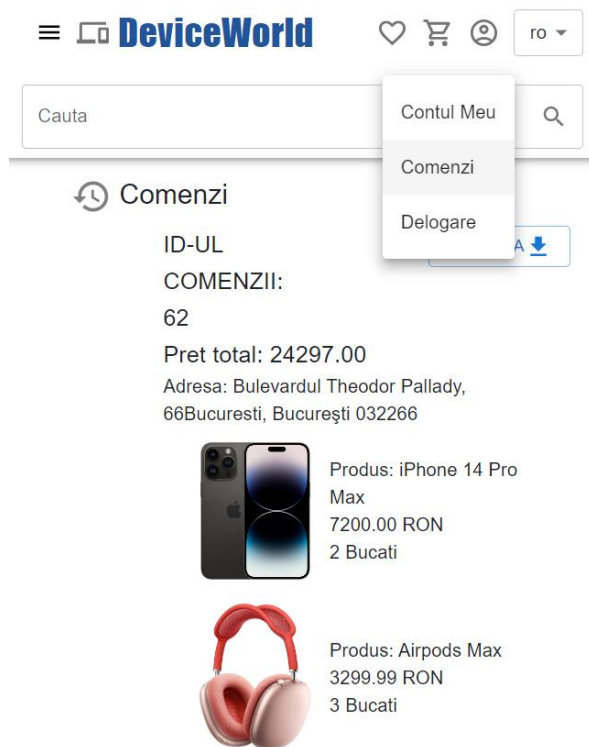


Fig. 4.7. – Pagina cu toate comenzile pe ecran mic

## 4.6. Funcționalitățile administratorului

Dacă utilizatorul autentificat este administrator, acesta beneficiază de drepturile clienților, dar are și dreptul de a gestiona ceilalți utilizatori și produsele magazinului. Astfel, atunci când se autentifică în aplicație, vor apărea două noi opțiuni pentru acesta în bara de navigare, una pentru gestionarea utilizatorilor și cealaltă pentru gestionarea produselor.

### 4.6.1. Gestionarea utilizatorilor

În pagina pentru gestionarea utilizatorilor administratorul poate modifica datele unui utilizator apăsând pe butonul de editare (creionul) și după modificare le salvează apăsând butonul corespunzător.

Dacă dorește eliminarea unui utilizator din listă, administratorul va apăsa butonul de ștergere (coșul); în urma acestei acțiuni va apărea o alertă care cere confirmarea acțiunii; după confirmare, utilizatorul respectiv este șters. Alerta este utilă în cazul în care butonul de ștergere este apăsător din greșeală pentru a nu șterge din baza de date un utilizator care nu necesită acest lucru.

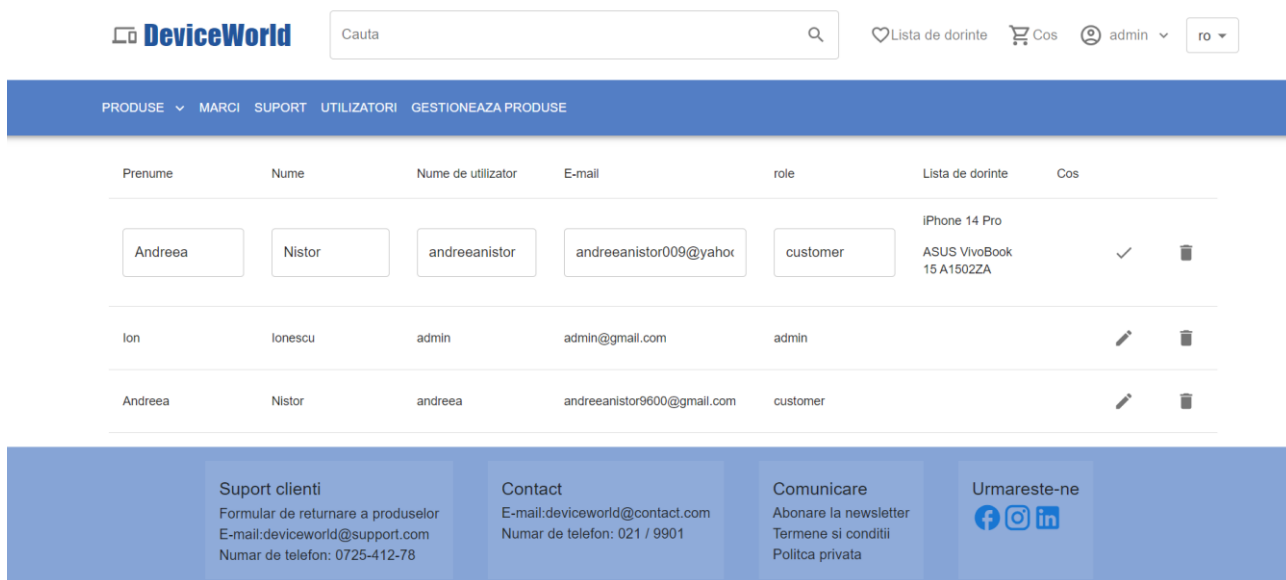


Fig. 4.8. – Gestionarea utilizatorilor

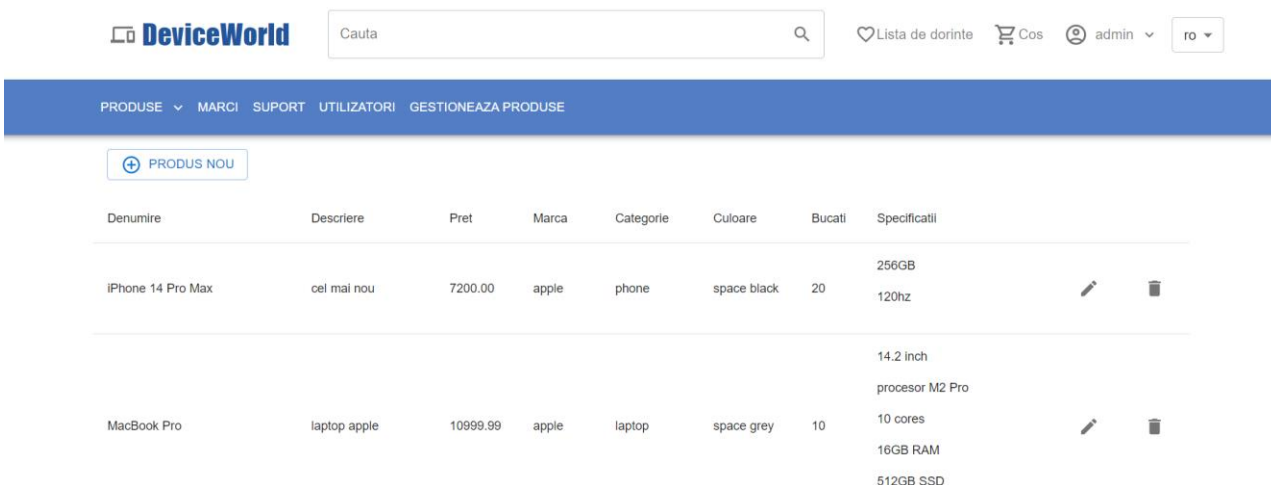
### 4.6.2. Gestionarea produselor

Pagina de gestionare a produselor arată similar cu cea de gestionare a utilizatorilor dar apare în plus butonul pentru adăugarea unui produs nou.

Pentru efectuarea unei schimbări în câmpul unui produs, administratorul va acționa butonul de editare (creionul) care îi va permite să facă modificări în cadrul produsului, modificări ce se vor salva și în baza de date după apăsarea butonului de salvare după terminarea acțiunii.

În cazul în care un produs nu se mai găsește în magazin, administratorul îl va elimina din lista produselor existente în magazin prin apăsarea butonului de ștergere din dreptul liniei produsului respectiv din tabelul cu toate produsele. La fel ca în cazul ștergerii unui utilizator va apărea un mesaj

de alertă care necesită confirmarea acțiunii pentru a nu șterge produse accidental.



The screenshot shows the DeviceWorld application interface. At the top, there is a navigation bar with the logo, a search bar, and links for 'Lista de dorinte', 'Cos', 'admin', and a language selector 'ro'. Below this is a blue header bar with navigation links: 'PRODUSE', 'MARCI', 'SUPPORT', 'UTILIZATORI', and 'GESTIONEAZA PRODUSE'. A '+ PRODUS NOU' button is located below the header. The main content area features a table with columns: 'Denumire', 'Descriere', 'Pret', 'Marca', 'Categorie', 'Culoare', 'Bucati', and 'Specificatii'. Two products are listed: 'iPhone 14 Pro Max' and 'MacBook Pro'. Each product row has edit and delete icons at the end. The 'Specificatii' column for 'MacBook Pro' is expanded, showing details like '14.2 inch', 'procesor M2 Pro', '10 cores', '16GB RAM', and '512GB SSD'.

Denumire	Descriere	Pret	Marca	Categorie	Culoare	Bucati	Specificatii
iPhone 14 Pro Max	cel mai nou	7200.00	apple	phone	space black	20	256GB 120hz
MacBook Pro	laptop apple	10999.99	apple	laptop	space grey	10	14.2 inch procesor M2 Pro 10 cores 16GB RAM 512GB SSD

Fig. 4.9 – Gestionarea produselor

## 4.7. Delogarea utilizatorului

Dacă vrea să iasă din cont, utilizatorul va trebui să apese pe butonul cu numele său de utilizator din partea de sus și să aleagă din meniul care se deschide opțiunea de „Delogare” care îl va scoate din contul curent, astfel permițându-i să se logheze cu alt cont de utilizator dacă acesta există sau să creeze unul.

## 5. Concluzii

Aplicația Device World este o aplicație de comerț electronic care se ocupă cu gestionarea și vânzarea produselor electronice care aparțin mai multor categorii (telefoane, laptop-uri, tablete, televizoare, accesorii). Aplicația este destinată persoanelor care doresc să afle informații despre produsele electronice existente pe piață, permițând și achiziționarea acestora. Având o interfață prietenoasă și intuitivă, aplicația este destinată tuturor persoanelor. Fiecare utilizator dispune de propriul coș de cumpărături care se păstrează în cont, permițând plasarea comenzii în orice moment. De asemenea, lista de dorințe este și ea specifică fiecărui utilizator, fiind utilă atunci când persoana nu este sigură de cumpărarea unui produs, dar vrea să îl acceseze mai rapid.

Provocările întâmpinate la nivelul proiectării aplicației au ajutat la o mai bună înțelegere a relației dintre partea de client și cea de server, comunicarea dintre cele două părți fiind necesară în majoritatea funcționalităților aplicației.

Aplicației i-ar putea fi aduse niște îmbunătățiri, de exemplu adăugarea unei metode de plată cu cardul în aplicație, lucru care va ușura achitarea comenzilor de către clienți. O altă îmbunătățire ce ar putea fi adusă aplicației este implementarea unui sistem de recenzii pentru produse în care fiecare utilizator să își poată scrie părerea și să poată acorda o notă produsului comandat, astfel ajutând la convingerea altor clienți să achiziționeze produse în funcție de notele pe care le au.



# Bibliografie

- [1]. TechTarget [Accesat pe 12-06-2023]:  
<https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app>
- [2]. axisbits [Accesat pe 12-06-2023]: <https://axisbits.com/blog/Types-of-Web-Application-Architecture-A-Concise-Summary>
- [3]. Geek for geeks [Accesat pe 12-06-2023]: <https://www.geeksforgeeks.org/html5-introduction/>
- [4]. Visual Studio Code [Accesat pe 12-06-2023]: <https://code.visualstudio.com/docs>
- [5]. Wikipedia [Accesat pe 12-06-2023]: [https://ro.wikipedia.org/wiki/Cascading\\_Style\\_Sheets](https://ro.wikipedia.org/wiki/Cascading_Style_Sheets)
- [6]. MDN [Accesat pe 12-06-2023]: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language\\_overview](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_overview)
- [7]. TechMagic [Accesat pe 12-06-2023]: <https://www.techmagic.co/blog/why-we-use-react-js-in-the-development/>
- [8]. W3Schools [Accesat pe 13-06-2023]: <https://www.w3schools.com/react>
- [9]. Material-UI [Accesat pe 13-06-2023]: <https://mui.com/material-ui/getting-started/overview/>
- [10]. TechTarget [Accesat pe 13-06-2023]: <https://www.techtarget.com/whatis/definition/Nodejs>
- [11]. PostgreSQL [Accesat pe 13-06-2023]: <https://www.postgresql.org/docs/15/intro-what-is.html>

# Listă figuri

1. Fig. 1.1. – Pagina de acasă în aplicația Device World
2. Fig. 1.2. – Pagina coșului de produse din aplicația Device World
3. Fig. 1.3. – Structura unei aplicații web [2]
4. Fig. 3.1. – Structura fișierelor aplicației React
5. Fig. 3.2. – Reprezentarea relațiilor dintre tabelele din baza de date
6. Fig. 3.3. – Header-ul și bara de navigare
7. Fig. 3.4. – Selectarea limbii afișate
8. Fig. 3.5. – Pagina de autentificare a aplicației
9. Fig. 3.6. – Pagina unui produs din aplicație
10. Fig. 3.7. – Pagina coșului de cumpărături
11. Fig. 3.8. – Detaliile comenzii produselor din coșul de cumpărături
12. Fig. 3.9. – comenzile utilizatorului autentificat
13. Fig. 3.10 – Factura unei comenzi în format PDF
14. Fig. 3.11. – Un produs adăugat în lista de dorințe
15. Fig. 3.12. – Formularul pentru adăugarea unui produs
16. Fig. 3.13. – Design-ul pe ecran mic
17. Fig. 3.14. – Meniul de tip hamburger
18. Fig. 3.15. – Design-ul pe ecran mare
19. Fig. 4.1. – Pagina de înregistrare
20. Fig. 4.2. – Pagina de produse pentru ecran mare
21. Fig. 4.3 – Pagină produse ecran mic
22. Fig. 4.4 – Dialogul de filtre pe ecran mic
23. Fig. 4.5. – Coșul de cumpărături pe ecran mic
24. Fig. 4.6. – Lista de dorințe
25. Fig. 4.7. – Pagina cu toate comenzile pe ecran mic
26. Fig. 4.8. – Gestionarea utilizatorilor
27. Fig. 4.9 – Gestionarea produselor