

ACADEMIA DE STUDII ECONOMICE DIN BUCUREȘTI



PROIECT Afaceri Electronice **Dezvoltarea unei aplicații de comerț electronic**

Profesori Coordonatori: Timofte Carmen Manuela

Lungu Mihai Adrian

Student:
Niculae Ioana-Daniela

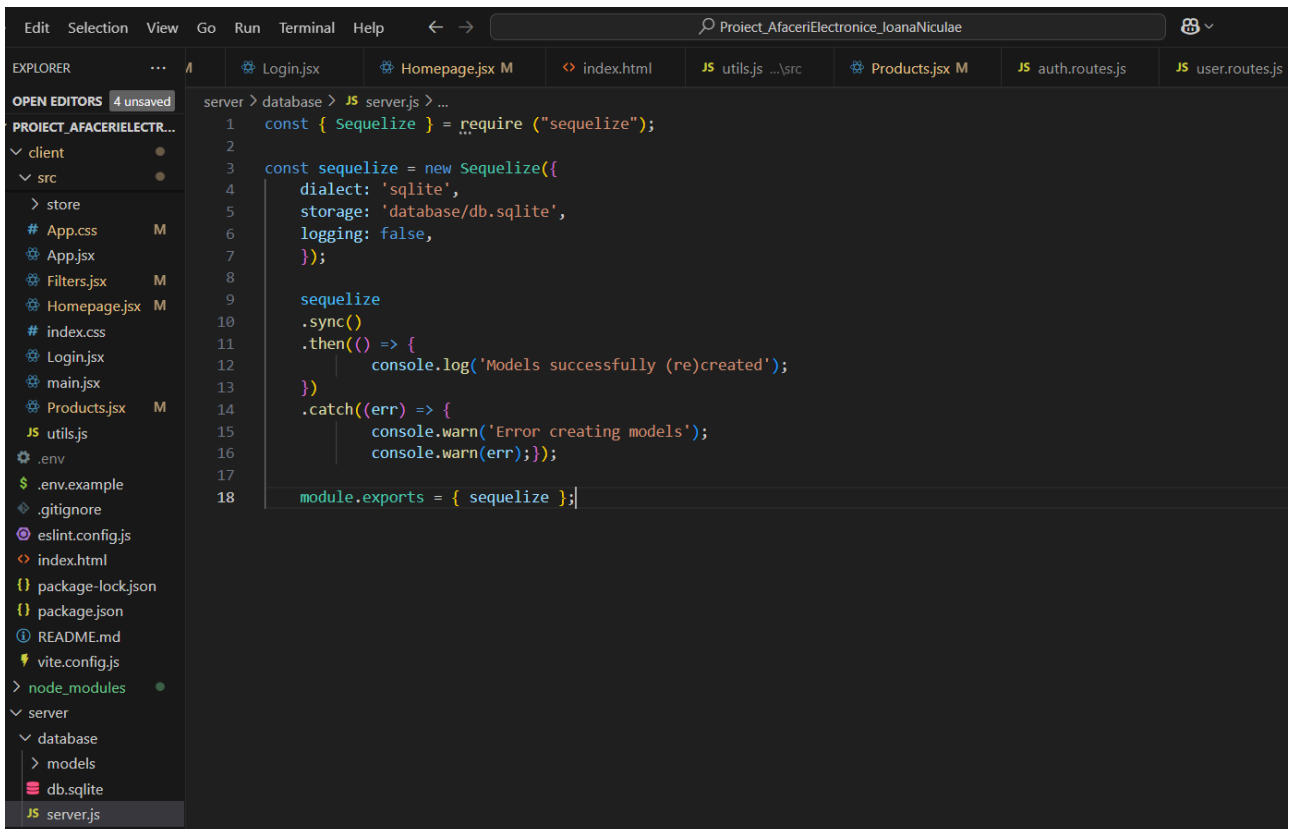
Link Github: <https://github.com/ioana-daniela-niculae>

1. Front-end SPA cu framework la alegere

În vederea realizării proiectului am folosit React pentru a crea o aplicație de tip Single Page Application (SPA). Astfel, aplicația se încarcă doar o singură dată și nu mai trebuie să reîncarce întreaga pagină de fiecare dată când navighezi pe site, iar în acest fel utilizatorul poate naviga între pagina de autentificare și cea a magazinului online cu produse, foarte rapid.

2. Configurarea bazei de date folosind un ORM (Object-Relational Mapper) – Sequelize

Am folosit Sequelize pentru a crea o bază de date SQLite care este utilizată pentru a stoca informațiile aplicației.



```
server > database > JS server.js > ...
1  const { Sequelize } = require("sequelize");
2
3  const sequelize = new Sequelize({
4    dialect: 'sqlite',
5    storage: 'database/db.sqlite',
6    logging: false,
7  });
8
9  sequelize
10 .sync()
11 .then(() => {
12   console.log('Models successfully (re)created');
13 })
14 .catch((err) => {
15   console.warn('Error creating models');
16   console.warn(err);
17 });
18 module.exports = { sequelize };
```

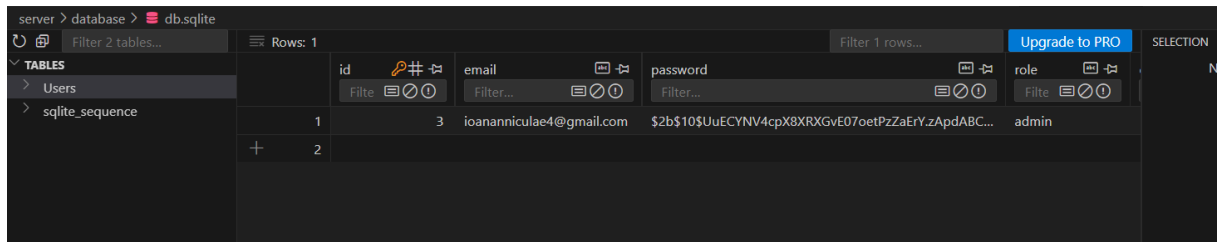
`const { Sequelize } = require("sequelize");`: Importă clasa Sequelize din librăria sequelize. Aceasta este folosită pentru a crea o instanță a ORM-ului.

`const sequelize = new Sequelize({ ... });`: crează o instanță a clasei Sequelize și o configurează cu următoarele opțiuni:

`dialect: 'sqlite'`: specifică tipul bazei de date.

`storage: 'database/db.sqlite'`: definește locația fișierului bazei de date SQLite. Acest fișier a fost creat în directorul database.

logging: false: dezactivează logarea interogărilor SQL în consolă.



id	email	password	role
1	ioananniculae4@gmail.com	\$2b\$10\$UuECYNV4cpX8XRXGvE07oetPzZaErY.zApdABC...	admin
2			

3. Implementarea unui mecanism de autentificare și gestionare a permisiunilor

Am implementat un mecanism de autentificare utilizând JSON Web Tokens care permite autentificarea utilizatorilor. La login, serverul generează un token care este trimis către client. Acest token este folosit pentru a verifica autenticitatea request-urilor viitoare.

Dacă email-ul și parola sunt valide, vom fi redirecționați direct în pagina cu produse.

Email:

Password:

Login

Dacă utilizatorul nu completează vreunul dintre câmpuri, se va afișa următorul mesaj:

Email:

Password:

! Completați acest câmp.

Login

În cazul în care adresa de mail nu este valida, va fi afișat mesajul “User not found”

Email

Password

Login

User not found

În cazul în care parola este introdusă incorect, va fi afișat următorul mesaj:

Email

Password

Login

Incorrect password. Try Again!

Application

Manifest

Service workers

Storage

Storage

Local storage

http://localhost:5173

Session storage

IndexedDB

Cookies

Private state tokens

Interest groups

Shared storage

Cache storage

Storage buckets

Background services

Back/forward cache

Background fetch

Background sync

Bounce tracking mitigati...

Notifications

Payment handler

Filter

http://localhost:5173

Origin http://localhost:5173

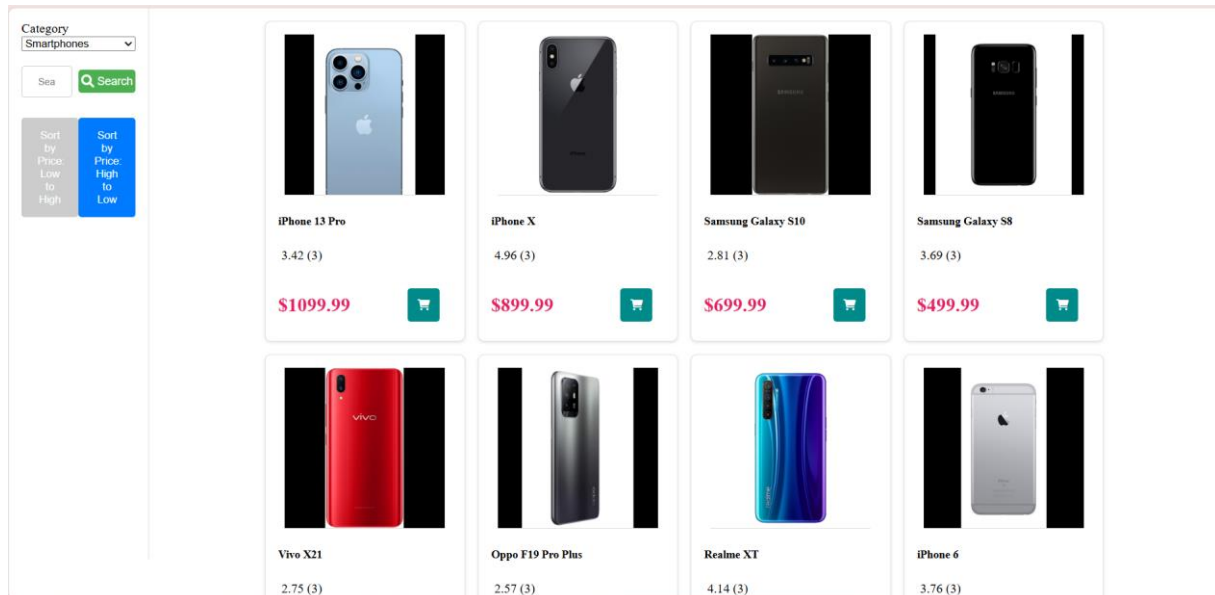
Key	Value
token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e...

1 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MywiaWF0IjoxNzMONzZ/

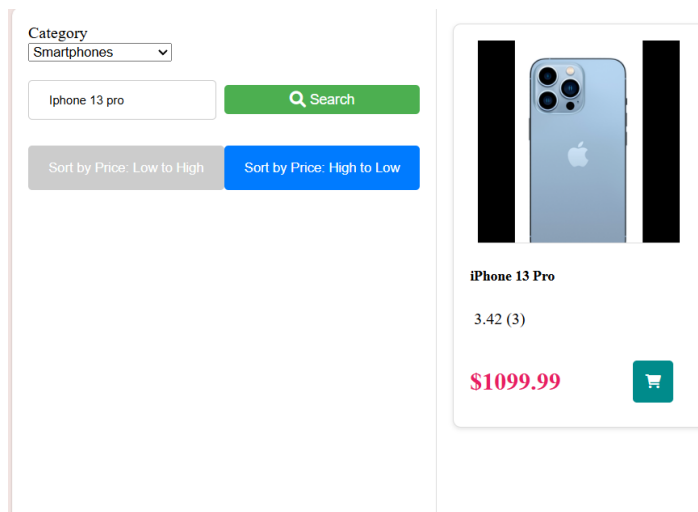
4. Pagina aferenta magazinului online

In urma procesului de login, suntem introdusi in pagina magazinului ce oferă utilizatorilor posibilitatea de a vizualiza și interacționa cu produsele disponibile pe site.

Fiecare produs are asociata o imagine sugestiva, denumirea, rating-ul, numărul de recenzii și prețul, iar utilizatorul poate adăuga produsele în coș direct din această pagină.



Produsele pot fi filtrate pe baza termenului de căutare introdus de utilizator in sectiunea de search, afișându-se doar acele produse ale căror denumiri conțin cuvintele căutate. În plus, există opțiunea de a sorta produsele după preț, fie în ordine ascendentă, fie descendentă, astfel încât utilizatorul să poată găsi rapid produsele la prețul dorit.



Dacă nu sunt produse care să îndeplinească filtrele de căutare, este afișat un mesaj care indică acest lucru, astfel încât utilizatorul să știe că nu există produse disponibile conform criteriilor sale.

<div>Category All products ▼</div> <div><input type="text" value="Iphone 16"/> Q Search</div> <div>Sort by Price: Low to High Sort by Price: High to Low</div>	No products found
---	-------------------

Autentificarea persistă chiar și după refreshuirea paginii.

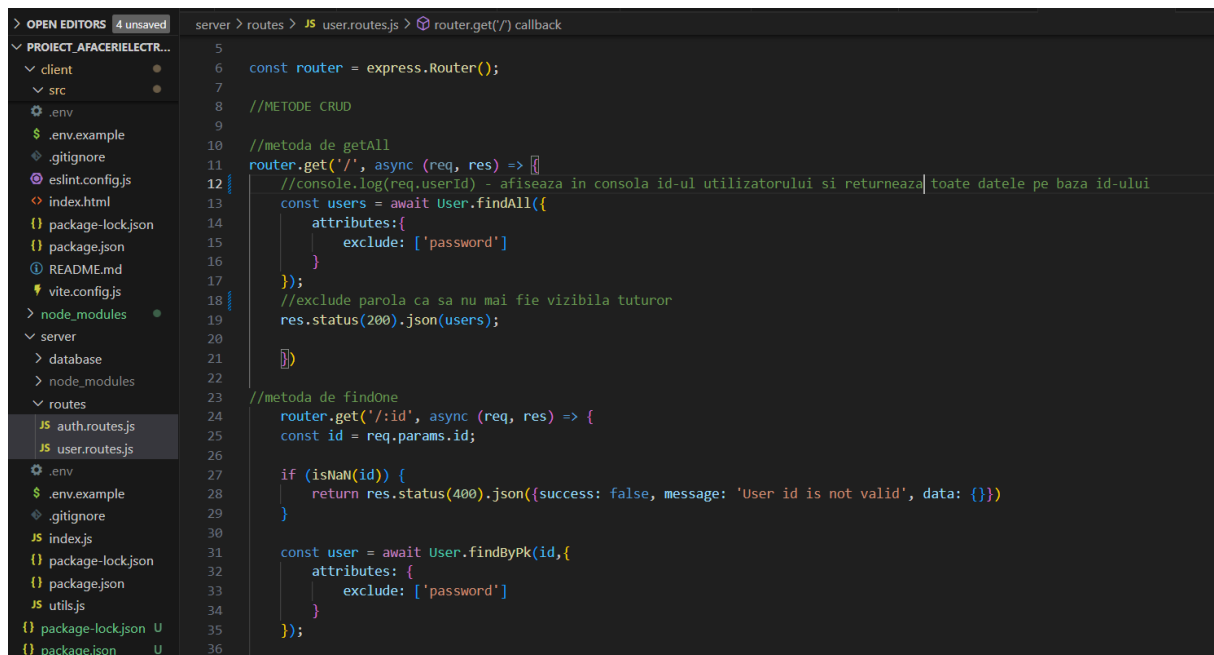
5. Operații expuse asupra entităților printr-o interfață REST

În cadrul proiectului am implementat o interfață REST pentru gestionarea utilizatorilor și autentificarea acestora. Am folosit framework-ul Express.js pentru a crea rute care expun operații CRUD (Create, Read, Update, Delete) asupra entității User, respectând principiile unei arhitecturi RESTful.

Am creat o rută POST /login, care permite utilizatorilor să se autentifice folosind email-ul și parola. Dacă datele sunt corecte, serverul generează un token JWT, care va fi folosit pentru a valida cererile viitoare ale utilizatorilor autentificați și am implementat și o rută POST /check, care permite verificarea validității token-ului JWT, asigurându-se că numai utilizatorii autentificați pot accesa resursele protejate ale aplicației.

M-am folosit și de operațiile CRUD pe care le-am efectuat atât utilizând Postman, cât și direct din cod, cu următorul scop:

- Crearea unui utilizator (POST /users): Permite înregistrarea unui nou utilizator, cu verificarea prealabilă a existenței unui utilizator cu același email. Parola este criptată înainte de a fi stocată în baza de date.
- Citirea utilizatorilor (GET /users și GET /users/:id): Permite obținerea unui utilizator sau a tuturor utilizatorilor din sistem. Informațiile sensibile, precum parola, sunt excluse din răspunsuri pentru a asigura protecția datelor utilizatorilor.
- Actualizarea unui utilizator (PUT /users/:id): Permite modificarea datelor unui utilizator existent pe baza ID-ului său.
- Ștergerea unui utilizator (DELETE /users/:id): Permite ștergerea unui utilizator existent din sistem.



```
server > routes > JS user.routes.js > router.get(/) callback
5
6 const router = express.Router();
7
8 //METODE CRUD
9
10 //metoda de getAll
11 router.get('/', async (req, res) => {
12   //console.log(req.userId) - afiseaza in consola id-ul utilizatorului si returneaza toate datele pe baza id-ului
13   const users = await User.findAll({
14     attributes: {
15       exclude: ['password']
16     }
17   });
18   //exclude parola ca sa nu mai fie vizibila tuturor
19   res.status(200).json(users);
20 }
21
22 //metoda de findOne
23 router.get('/:id', async (req, res) => {
24   const id = req.params.id;
25
26   if (isNaN(id)) {
27     return res.status(400).json({success: false, message: 'User id is not valid', data: {}})
28   }
29
30   const user = await User.findById(id, {
31     attributes: {
32       exclude: ['password']
33     }
34   });
35 }
36
```

Aplicația folosește următoarele tehnologii și tool-uri:

- **Postman** - pentru testarea și validarea endpoint-urilor API-ului.
- **VSCode** - editorul de cod
- **Git** - pentru a publica codul pe repository
- **Node.js** - pentru rularea aplicației pe server folosind JavaScript.
- **Vite** – pentru dezvoltarea părții de frontend
- **Express.js** - Framework web pentru Node.js, folosit pentru a crea servere web și pentru a gestiona rutele aplicației
- **Sequelize** - ORM (Object-Relational Mapping) pentru Node.js, folosit pentru a interacționa cu baza de date (SQLite)
- **SQLite** - baza de date relațională folosită pentru stocarea datelor
- **JWT (JSON Web Tokens)** - pentru autentificare și autorizare, asigurând că utilizatorii sunt verificați pe baza unui token securizat.
- **Bcrypt.js** – bibliotecă JavaScript folosită pentru criptarea și verificarea parolelor utilizatorilor, asigurând securitatea datelor sensibile.
- **CORS (Cross-Origin Resource Sharing)** - permite accesul aplicației, asigurând că permisiunile de acces sunt corecte.
- **Morgan** - Un middleware pentru Express care ajută la logarea informațiilor despre cererile HTTP făcute către server.
- **React.js** (pentru frontend) - bibliotecă JavaScript folosită pentru a construi interfața de utilizator, ce permite crearea de aplicații SPA.
- **.env** - permite gestionarea variabilelor de mediu (de exemplu, chei secrete pentru tokenuri sau informații de conexiune la bazele de date) într-un fișier .env, pentru a păstra informațiile sensibile în afacerea de cod.
- **react-router-dom** - permite navigarea între rute
- **@vitejs/plugin-react** - Plugin care ajută la integrarea React cu Vite.
- **@eslint/js** - Configurare ESLint pentru JavaScript.
- **@types/react** - Tipuri TypeScript pentru React.
- **@types/react-dom** - Tipuri TypeScript pentru React DOM.
- **Redux toolkit**