Universitatea Natională de Știință și Tehnologie POLITEHNICA BUCURESTI

Facultatea de Electronică, Telecomunicații si Tehnologia Informației

PROIECT 3

3. Aplicatie web pentru informatii despre cartile Hearthstone – JavaScript/NodeJS

Student: Jitianu Stefan

Grupa: 445A

Profesor coordonator:

Conf. Dr. Ing Bogdan Florea

Cerintele Aplicatiei

Aplicatia web va permite cautarea cartilor dupa nume, clasa sau rasa folosind API-ul Hearthstone (<https://rapidapi.com/omgvamp/api/hearthstone>). Rezultatul cautarii va afisa grafica cartilor si informatiile acestora intr-o interfata responsive. In cazul in care cartile au varianta gold, se va implementa posibilitatea vizualizarii acesteia printr-o metoda la alegere (hover imagine, modal, etc.). Aplicatia trebuie sa aiba o interfata responsive.

Cuprins

[1 Descrierea tehnologiilor folosite 4](#_Toc153626641)

[1.1 React.js 4](#_Toc153626642)

[1.2 API (Application Programming Interface) 6](#_Toc153626643)

[1.3 JavaScript si NodeJS 7](#_Toc153626644)

[2 Descrierea aplicatiei 8](#_Toc153626645)

[2.1 Diagrama logica 8](#_Toc153626646)

[2.2 Rute 9](#_Toc153626647)

[2.3 Functionalitati implementate 10](#_Toc153626648)

[2.3.1 Preiau date de la utilizator. 11](#_Toc153626649)

[2.3.2 Trasmit cererea http. 11](#_Toc153626650)

[2.3.3 Filtrez datele. 13](#_Toc153626651)

[2.3.4 Afisez datele. 14](#_Toc153626652)

[2.3.5 Hover imagine. 14](#_Toc153626653)

[2.3.6 Paginatie. 14](#_Toc153626654)

[Concluzie 16](#_Toc153626655)

[Bibliografe 17](#_Toc153626656)

# Descrierea tehnologiilor folosite

## React.js

React este o librarie JavaScript,open-source, pentru construirea de interfete utilizator, creata de compania Meta(fost Facebook) ([[1]](#endnote-1)) .Aceasta a fost publicata prima data in 2013.

Problemele pe care le intampina Facebook in 2012 si care au dus la dezvoltarea React au fost ca nu puteau sa mareasca proportia de JavaScript din proiectele lor deoarece cu cat duceau mai mult cod care sa se ruleze pe cartea de client(browser, device) cu atat acesta se misca mai incet. Din acest motiv balanta se inclina inspre cod rulat pe server si folosirea de cantitati minime de JavaScript. Facebook este mai mult o companie de entertain, de a-ti da o stare de spirit placuta, si nu poate sa faca acest lucru decat cu o interfata utilizator atragatoare, interactiva, si acest lucru se face cu JavaScript. Insa cu conceptele ce erau pe piata la acea vreme nu aveau cum sa obtina rezultatele pe care ei si le doreau, partea de dezvoltare pe client nu functiona si de asemnea partea de server nu avea cum sa ii faca sa isi atinga obictivele.

In acest context ingineri precum Lee Byron, Tom Occhino, Adam Wolff, Shane O’Sullivan, Christopher Chedeau si alti, condusi de Jordan Walke au fost provocati si stransi sub aceeasi idee de a dezvolta un nou framework care sa reincarce aplicatia de fiecare data cand ceva se schimba, fie o stare a unui **API**(**Application Programming Interface**), fie utilizatorul a tastat ceva ([[2]](#endnote-2)).

React a introdus **JSX**, care este o sintaxa ce combina codul JavaScript cu HTML. Defapt este o extensie de sintaxa pentru JavaScript care iti permite sa scrii limbaj de marcare intr-un fiser JavaScript.Desi exista si alte modalitati de a scrie componente majoritatea dezvoltatorilor web prefera sa foloseasca JSX. In mod normal dezvoltatorii de aplicatii web foloseau HTML pentru continut, CSS pentru design, si JavaScript pentru logica aplicatiei, de multe ori in fisere diferite. Insa in React folosind sintaxa JSX se pot folosi logica de redare si limbajul de marcare in acelasi loc ([[3]](#endnote-3)).

A screenshot of a computer program

Description automatically generated

Un alt mare avantaj al React este ca inveti odata sintaxa si logica acestuia si poti dezvolta cu el folosind aceleasi principii o aplicatie pe orice platforma indiferent de tehnologia deja existenta. Se pot dezvolta inclusiv aplicatii Android, iOS folosind React Native.Acest lucru poarta numele de principiu „learn once write anywhere” (2).

Figurile centrale ale sistemului React sunt **componentele** care asigura organizarea eficienta a mediului de utilizator a unei aplicatii, adica asigura ca acesta sa fie impartit in unitati mai mici, fiecare unitate avand o sarcina unica. Cu toate acestea si o componenta este compusa la randul ei din unul sau mai multe elemente, care reprezinta cea mai mica entitate a sistemului. De aici si simbolul pe care il are React, un atom, pentru a pune in evidenta gandirea oarecum atomica atunci cand se dezvolta o aplicatie (2).

La nivel de concept o componenta este precum o junctie JavaScript. Aceepta intrari (numiti ‚props’) si intoarce un element React care descrie ceea ce ar trebui sa apara pe ecran([[4]](#endnote-4)).

**React Hooks** (carligele) au fost adaugate in React incepand cu versiunea 16.8. Acestea permit componentelor sa aibe acces la diferite trasaturi ale React cum ar fi starile, efectele si altele ([[5]](#endnote-5)). Exemple de React Hooks incorporate sunt:

* State Hooks care permite unei componente sa retina informatiile pe care le introduce utilizatorul spre exemplu intr-un camp de input.
* Effect Hooks care permit unei componente sa se conecteze si sa se sincronizeze cu sistemele externe. Aceasta include restrctionarea retelei, a browserului DOM, a animatiilor si a altor coduri non-React ([[6]](#endnote-6)).

In concluzie atunci cand vine vorba de alegerea unei tehnologii pentru dezvoltarea unei aplicatii web React este una dintre cele mai bune variante datorita modului sau de lucru si a functionalitatilor sale dar si a comunitatii foarte mari care exista si dezvoltarii continue de care are parte. De aceea a si fost ales de companii de renume precum Netflix, AirBnb, Microsoft etc.

## API (Application Programming Interface)

API ( interfata de programare a aplicatiei) este utilizat pentru a transmite bidirectional date intre aplicatiile software intr-un mod fomalizat ([[7]](#endnote-7)) de tip JSON sau XML. API-urile au existat intr-o forma sau alta de la inceputul calculatoarelor insa forma pe care o au in ziua de astazi s-a format pe la inceputul anilor 2000 cu contribuitia unor companii precum eBay, Amazon, Salesforce ([[8]](#endnote-8)). In ziua de azi API-urile nu se mai folosesc doar la aplicatii desktop , web sau mobile ci se folosesc la ceea ce se numeste Internet of Things. Numeroase servicii oferă API‑uri publice, care permit oricui să trimită și să primească conținut din serviciul respectiv. API‑urile care funcționează prin intermediul Internetului utilizând URL‑uri de tip http:// sunt denumite API‑uri web . Pe web, puteți trimite o solicitare către un API pentru a obține și posta informații (7).

Sunt nenumarate API-uri care exista,spre exemplu atunci cand ne luam un mouse sau o tastatura noua si le conectam la calculator acestea functioneaza pur si simplu, ei bine si acolo este tot un API fiind bine definit un protocol de comunicare intre dispozitive si sistemul de operare ([[9]](#endnote-9)). Unele dintre cele mai cunoscute API-uri web sunt REST si SOAP.

REST(Representational state transfer) API este metoda mai simpla si mai flexibila de a construi un API si este bazat pe protocolul HTTP si foloseste cereri http pentru accesa si utiliza date. Operatiile pe care le poate executa asupra datelor sunt citirea, crearea, modificarea sau stergerea de date, ce sunt in general in format JSON. Aceste API-uri sunt adesea expuse pe o adresa url iar aplicatiile pot interactiona cu aceste API facand solicitari http catre acest url (9). Aceste API-uri sunt in general dezvoltate de utilizatori terti.

SOAP (Simple Object Access Protocol) este o modalitate foarte stricta de a construii un API. Acesta este folosit de obicei pentru aplicatii mari la nivel de interprindere fiind si mai complexe. Acestea necesita de multe ori setare si configurare suplimentara fata de o REST API. Dificultatea folosiri acestui tip vine si din faptul ca datele folosite in comunicare sunt in format XML, care este un format mai strict si dificil de lucrat (9).

## JavaScript si NodeJS

NodeJS este un mediu de rulare JavaScript multiplatforma, open-source, ce permite codului Javascript sa fie rulat pe partea de server. El permite executarea codului JavaScript in afara browser-ului web. O aplicatie Node.js ruleaza intr-un singur proces, fara a crea thread-uri noi pentru fiecare solicitare. Node.js ofera un ser de primitive I/O asincrone in biblioteca sa standard care impiedica blocarea JavaScript si in general, bibliotecile Node.js sunt scrise folosind paradigme neblocante(10).

Atunci cand se efectueaza o operatie I/O, precum citirea din retea, accesarea bazei de date sau a sistemului de fisiere, in loc sa blocheze thread-ul si sa iroseasca ciclurile CPU in asteptare, Node.js va relua operatiile cand raspunsul revine(10).

Acest lucru permite Node.js sa gestioneze mii de conexiuni simultane cu un singur server fara a introduce modalitati de management al thread-urilor concurente care de multe ori este o sursa semnificativa de erori ([[10]](#endnote-10)).

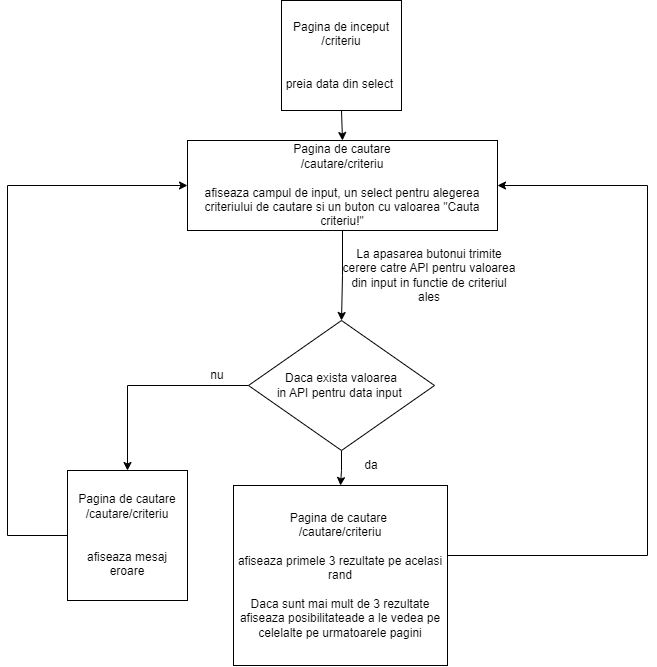
JavaScript este un limbaj de programare orientat pe obiect folosit pentru a face paginile web interactive ( animatii, pop-up, meniuri etc). In interiorul unui mediu gazda , cum ar fi un browser, JavaScript poate fi folosit in controlul obiectelor in mediul respectiv([[11]](#endnote-11)).

A fost creat in 1995 de Brendan Eich pe vremea cand lucra ca inginer la Netscape numindul LiveScript. Spre deosebire de alte limbaje de programare, JavaScript nu are conceptul de intrare si iesire. El este proiectat sa ruleze ca limbaj de scripting pe un mediu gazda si depinde de mediul de gazda sa ofere mecanisme de comunicare cu exteriorul , cel mai comun mediu fiind browserul web ([[12]](#endnote-12)).

Desi este de foarte multe ori confundat cu Java datorita numelui dar si pentru faptul ca ambele sunt folosite in mediul web acest lucru nu inseamna ca sunt unul si acelasi limbaj. In momentul in care in ani 90 cand browserul Netscape a inclus suport pentru limbajul de programare Java s-a modificat si numele limbajului de scripting client din LiveScript in JavaScript de aici apare confuzia de nume. Atunci cand spunem suport pentru Java ne referim la posibilitatea de a activa Java Applet.

# Descrierea aplicatiei

## Diagrama logica



## Rute

Rutele au fost implementate in felul urmator folosit React Router:

<Routes>

      <Route path="/" element={<Navigate to="/criteriu"/>} />

      <Route path="/criteriu" element={<Criteriu />} />

      <Route path="/cautare/:tip" element={<Cauta />}/>

      <Route path="/notfound" element={<NotFound/>}/>

      <Route path="\*" element={<NotFound />} />

    </Routes>

Pagina default este pagina /criteriu. Care afiseaza componenta Criteriu.

La alegerea unei valori din selectul din pagina /criteriu sunteti redirectionat catre pagina /cautare/”valoareSelect”. In ruta „/cautare/:tip” folosind :tip am definit un parametru din url ce se va modifica dinamic in functie de valoarea pe care a primit-o din selectul din pagina /criteriu, aceasta fiind valoarea default dar apoi se poate modifica si in functie de valoarea selectata . Aceasta valoare am preluat-o cu React Hook „useParams” care imi returneaza un obiect ce contine toti parametrii definiti personalizat in rute. Pentru ca definirea :tip practica inseamna orice valore din url am fost nevoit sa folosesc un alt React Hook useEffect ca sa verific valoarea din parametrul tip. Daca acesta nu are valoarea nume, clasa sau rasa sunteti redirectionat catre pagina NotFound.

Apoi prin ruta „\*” orice accesare a unui url care nu a fost definit in rute este redirectionata catre pagina NotFound unde se afla un mesaj corespunzator.

Structura de fisiere:

A screen shot of a computer

Description automatically generated

## Functionalitati implementate

Scopul acestui proiect este sa realizez cautarea cartilor Hearthstone dupa nume, clasa sau rasa. In vederea implementarii acestei cerinte a fost nevoie sa implementez alte functionalitati mai mici si sa folosesc API-ul (<https://rapidapi.com/omgvamp/api/hearthstone>).

Eu am folosit framework-ul React si alte librari ale sale.

### Preiau date de la utilizator.

In primul rand in aplicatia mea era nevoie sa preiau date de la utilizator. Aceste date erau numele carti, a clasei din care ea face parte sau a rasei din care facea parte si mai aveam nevoie de o modalitate de a alege un criteriu dintre cele enumerate mai devreme. Asa ca pentru a prelua numele am folosit un input de tip text iar pentru a prelua criteriu am folosit un select cu valorile corespunzatoare cele doua fiin incluse intr-un formular. Pe acest formular am adaugat tot felul de functii care sa se execute atunci cand utilizatorul interactioneaza cu el. Varianta sa finala arata in felul urmator:

<input className="input\_cauta" type='text' id='caut' name='caut' value={nume} onChange={e => setNume(e.target.value)}/>

                    <select className='select\_cauta' id="criteriu" value={criteriu} name="criteriu" onChange={potiCauta}>

                        <option value="nume">Nume</option>

                        <option value="clasa">Clasa</option>

                        <option value="rasa">Rasa</option>

                    </select>

Se vede ca pentru cele doua am adaugat cate un atribut de eveniment „onChange” care atunci cand sesizeaza o schimbare in element declanseaza/apeleaza functia atribuita. In acest caz sunt apelate doua functii: setNume si potiCauta(puteam sa il denumesc modificaUrl).

setNume este o functie provenita din hook-ul useState care atunci cand este apelata actulizeaza valoarea din starea curenta a numelui.

const[nume, setNume] = useState("");

potiCauta este functia care schimba url-ul si valoarea criteriului de cautare la schimbarea valorii din select

function potiCauta(e)

    {

        setCriteriu(e.target.value);

        navigate(`/cautare/${e.target.value}`);

    }

setCriteriu modifica valoarea din starea curenta pentru criteriu de cautarea. Iar navigate este o functie care este apartine de hook-ul useNavigate pen care am definit-o mai sus si am folosit-o pentru a updata ruta.

### Trasmit cererea http.

Datele erau acum gata de a fi preluate de aplicatie, mai era nevoie de un buton pentru a permite aplicatiei sa preia aceste date si sa le prelucreze. Asa ca eu am folosit 3 butoane dar se afiseaza numai unul in functie de criteriul ales si fiecare buton apeleaza la randul sau cate o functie.

{criteriu === 'nume' ? (<button className='buton' onClick={cautare\_nume}>Cauta nume!</button>): null}

                    {criteriu === 'clasa' ? (<button className='buton' onClick={cautare\_clasa}>Cauta clasa!</button>): null}

                    {criteriu === 'rasa' ? (<button className='buton' onClick={cautare\_rasa}>Cauta rasa!</button>): null}

La apasarea butonului acesta apelaza functia care face cererea http care API-ul Hearthstone pentru a aduce datele in pagina in functie de criteriu ales si numele transmis.

In continuare voi prezenta dor una dintre aceste functii intrucat toate cele 3 sunt asemanatoare difera doar ruta pe care o apeleaza.

async function cautare\_nume(e){

        e.preventDefault();

        console.log("Valoare Nume:", nume);

        //cautare search

        const options = {

            method: 'GET',

            url: `https://omgvamp-hearthstone-v1.p.rapidapi.com/cards/search/${nume}`,

            headers: {

              'X-RapidAPI-Key': '003f468fddmsheef85277d8b2277p11b0a3jsn71cb8a1c94df',

              'X-RapidAPI-Host': 'omgvamp-hearthstone-v1.p.rapidapi.com'

            }

          };

        try {

            const response = await axios.request(options);

            console.log("Date primite cerere: ",response.data);

            //setDate(response.data.filter(componenta=> componenta.img).slice(0,3));

            setDate(response.data.filter(componenta=> componenta.img && (componenta.type !== 'Hero') ));

            //curat\_date();

            setEroare("");

            setNume("");

        } catch (error) {

            console.error(error);

            setDate([]);

            alert('Nu exista date pentru valoarea introdusa. Verifica criteriu de cautare!');

            setEroare('Nu exista date pentru valoarea introdusa. Verifica criteriu de cautare!');

            setNume("");

        }

    }

In primul rand functia de mai sus este asincrona ceea ce inseamna ca nu blocheaza aplicatia cat timp asteapta rezultate de la API. Uneori acest lucru poate dura o perioada de timp deranjanta daca aplicatia s-ar bloca si nu ar permite rularea altor operatiuni, interactiuni cu utilizatorul.

e.preventDefault(); previne comportamentul implicit al formularului care in mod normal atunci cand este trimis reincarca pagina, ceea ce inseamna ca starea componentei s-ar fi resetat iar eu nu voiam acest scenariu pentru aplicatia mea.

Console.log l-am folosit peste tot pentru depanare.

options este o constanta care stocheaza un obiect ce contine informatii de configurare pentru cererea http catre API-ul hearthstone.

* method – specifica metoda http utilizata pentru cerere;
* url – este url-ul catre care se face cererea. Aici este foarte important pentru ca valoarea luata din input este transmisa in url in mod dinamic prin expresia ${nume}. Daca nu as fi folosit aceasta expresie apelarea functiei ar fi fost inutila iar aplicatia nu ar fi mers conform cerintelor.
* headers – ofera informatii suplimentare pentru API necesare pentru autentificarea cererii.

Apoi am folosit o structura de tip try-catch si biblioteca axios pentru a efectua cererea. Await este folosit pentru a astepta pana cand cererea este completa inainte sa treaca la urmatoarea linie de cod. Prin setData am salvat datele in variabila dateComplete insa inainte de le salva le-am filtrat si le-am salvat doar pe cele care au cheia imagine iar tipul este diferit de Hero. In catch afisez o alerta in cazul in care cererea nu a fost facuta corespunzator catre API setez mesjul de eroare si golesc valoarea din nume. Diferenta la celelate doua functii ar fi ca url-ul se modifica.

### Filtrez datele.

Pe langa filtrarea de mai sus a fost nevoie sa mai fac o filtrare suplimentara intrucat erau foarte multe carti care aveau acelasi nume si se repetau doar pentru faptul ca faceau parte din alte colectii asa ca am ales sa pastrez prima carte care apare in cerere.

useEffect(() => {

        const cartiUnice = [];

        const cartiExistente = {};

        dateComplete.forEach((carte) => {

            const numeCarte = carte.name;

            const cardSet = carte.cardSet;

            console.log("Nume carte", numeCarte);

            //console.log("Carti de nume carte", cartiExistente['name']);

            //console.log("tip carte", tipCarte);

            console.log("sunt in harta");

            console.log("carti existente", cartiExistente);

            if(!cartiExistente[numeCarte])

            {

                console.log("sunt in if");

                cartiUnice.push(carte);

                cartiExistente[numeCarte] = {nume: numeCarte, cardSet: cardSet}

            }

        });

        setDateFiltrate(cartiUnice);

        setPaginaCurenta(1);

    },[dateComplete]);

Am folosit hook-ul useEffect care se declanseaza atunci cand se modifica valoarea din dateComplete. Pentru a filtra am folosit un array si un obiect de perechi chei -valoare. Am parcurs fiecare obiect json adus din API si am verifcat daca in obiectul cartiExistente am intrgistrata o valoare pentru numele obiectului curent din API. Daca nu am il salvez in array-ul cartiUnice si salvez valoarea sa in obiectul cartiExistente astfel incat la urmatoare intalnire a numelui respectiv sa nu mai salvez nimic. La sfarsit am salvat datele din sirul de obiecte json cartiUnice in dateFiltrate.

### Afisez datele.

Pentru a afisa datele a fost nevoie sa definesc o noua componenta Carte.js pentru ca aveam nevoie ca fiecare chenar cu carte sa aibe starea lui independenta in ceea ce priveste hoover-ul pentru imagine in varianta gold daca exista.

<Carte key={index} dateT={dateT} />

Pentru a afisa datele nu am facut nimic decat am transmis parametrii corespunzatori componentei pentru chenarul fiecarei carti, am mapat datele si le-am afisat in elementele corespunzatoare.

### Hover imagine.

Am definit o stare folosindu-ma de hook-ul useState() pentru a inregistra daca trec cu cursorul mouse-ului peste imagine sau nu.

const [hoverImg, setHoverImg] = useState(null);

onMouseEnter={() => setHoverImg(true)}

onMouseLeave={() => setHoverImg(false)}>

Apoi am folosit un operator ternar care sa afiseze imaginea gold daca mouse-ul este peste imagine si daca am varianta gold iar daca nu imaginea originala. Am mai adugat o transformare care sa maresca imaginea daca sunt cu cursorul peste din css.

### Paginatie.

Modul in care am gandit eu aplicatia a fost sa afisez doar trei carti pe ecran chiar daca erau mai multe din acest motiv a fost nevoie sa implementez o functie de paginatie pentru a afisa si celelalte carti in cazul in care sunt mai multe. Pentru asta am definit o noua componenta Paginatie.js care are urmatoarele proprietati (props) eu ii consider niste parametrii:

<Paginatie

                  postariPePagina={postariPePagina}

                  postariTotale={dateFiltrate.length}

                  paginare= {paginare}

                  paginaPrec={paginaPrec}

                  paginaCurenta = {paginaCurenta}

                  paginaUrm={paginaUrm}

                />

Am definit o stare pentru paginaCurenta si pentru numarul de postari pe pagina.

const [paginaCurenta, setPaginaCurenta] = useState(1);

const [postariPePagina, setPostariPePagina] = useState("");

Apoi am folosit urmatoarea logica pentru a obtine indexi imaginilor si am facut un slice.

const indexUltimaimagine = paginaCurenta\* postariPePagina;

const indexPrimaImagine = indexUltimaimagine - postariPePagina;

const postariCurente=dateFiltrate.slice(indexPrimaImagine,indexUltimaimagine);

A definit trei functii: paginare, paginaPrec, paginaUrm cu scopul de a trece de la o pagina la alta folosindu-ma de indexi descrisi mai sus.

const paginare = () => {

        setPaginaCurenta(paginaCurenta+2);

     };

     const paginaPrec = () => {

        if (paginaCurenta !== 1) {

           setPaginaCurenta(paginaCurenta - 1);

        }

     };

     const paginaUrm = () => {

        if (paginaCurenta!== Math.ceil(dateFiltrate.length / postariPePagina)) {

           setPaginaCurenta(paginaCurenta + 1);

        }

     };

Deci aceasta componenta arata efectul unei paginatii chiar daca practic nu se trece de la o pagina la alta pentru ca doar se face un slice pe date si se afiseaza alte informatii, modificand dinamic valorile indexilor de imagine.

Tot aici am implementat si un useEffect ce tine de partea responsive care sa imi verifice dimensiunea ferestrei iar daca aceasta este mai mica de 800px sa imi afiseze o singura imagine pe pagina.Practic printr-un if am actualizat valoarea din postariPePagina.

useEffect(()=>{

        //console.log("sunt in useEffectpt paginare afisare 1 \*\*\*")

        const gesNrPostari = () => {

            if(window.innerWidth < 800)

            {

                //console.log("sunt in if pag@@@")

                setPostariPePagina(1);

            }

            else{

                //console.log("sun in if pt 3 img pe pagina")

                setPostariPePagina(3);

            }

        }

        // Adăugare ascultător de evenimente la încărcarea componentei

        window.addEventListener('resize', gesNrPostari);

        // Eliminare ascultător de evenimente la demontarea componentei

        return () => {

            window.removeEventListener('resize', gesNrPostari);

        };

    },[]);

window.addEventListener(‚resize’,gesNrPoastari); este un ascultator de evenimente;

addEventListener este metoda la obiectul window care are doi parametrii, resize fiind tipul de eveniment pe care vrem sa il ascultam iar gesNrPostari este functia care se apeleaza la modificarea dimensiunii ferestrei.

Este important si sa elimin acest ascultator de evenimente in cazul in care componenta in care il folosesc este scoasa din DOM si acest lucru in fac prin functia return.

# Concluzie

In concluzie React a fost o alegere potrivita pentru acest proiect intrucat daca as fi folosit alte framework-uri sau doar JavaScript si Node.js ar fi fost mult mai dificila implemenarea pentru a ajunge la aceleasi performate. Am ajuns la concluzia ca React este un framework foartea util atunci cand vine vorba de UI datorita modalitati sale de a gestiona DOM-ul si datorita comunitatii extinse care este de un adevarat ajutor atunci cand intampin probleme in implementare pentru ca sunt sanse foarte mari ca cineva sa fi avut aceeasi problema ca si mine si sa gasesc o solutie in comunitate chiar daca apoi vin cu modificarile mele asupra acelei solutii pentru a o imbunatatii.

# Bibliografe

1. React – a JavaScript library for building user interfaces.

   <https://legacy.reactjs.org/> [↑](#endnote-ref-1)
2. How A Small Team of Developers Created React at Facebook | React.js: The Documentary

   <https://www.youtube.com/watch?v=8pDqJVdNa44&list=PL8q0RDaH87JopPoApu6o3p2u2_CHTp8pF&index=7&ab_channel=Honeypot> [↑](#endnote-ref-2)
3. Writing Markup with JSX – React

   <https://react.dev/learn/writing-markup-with-jsx> [↑](#endnote-ref-3)
4. Component and Props - React

   <https://legacy.reactjs.org/docs/components-and-props.html> [↑](#endnote-ref-4)
5. React Hooks

   <https://www.w3schools.com/react/react_hooks.asp> [↑](#endnote-ref-5)
6. Build-in React Hooks

   <https://react.dev/reference/react/hooks> [↑](#endnote-ref-6)
7. Ce este un API – Apple Suport(Ro)

   <https://support.apple.com/ro-ro/guide/shortcuts-mac/apd2e30c9d45/mac> [↑](#endnote-ref-7)
8. Intro to APIs: History of APIs

   <https://blog.postman.com/intro-to-apis-history-of-apis/> [↑](#endnote-ref-8)
9. API For Dumies: Learning The Basics of API

   <https://www.scrapingbee.com/blog/api-for-dummies-learning-api/> [↑](#endnote-ref-9)
10. Introduction to Node.js | Node.js

    <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs> [↑](#endnote-ref-10)
11. Introductio JavaScript | MDN

    <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction#what_is_javascript> [↑](#endnote-ref-11)
12. Introduction to JavaScript – GeeksForGeeks

    <https://www.geeksforgeeks.org/introduction-to-javascript/> [↑](#endnote-ref-12)