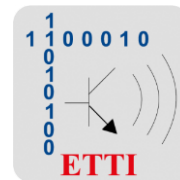




UNIVERSITATEA POLITEHNICA DIN BUCURESTI
Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei



Proiect 3

Aplicație web pentru căutare de hoteluri

Student: Chelariu Mihai - Silviu

grupa: 444 A

BUCUREȘTI 2024

Cuprins

Introducere	3
Obiective	3
Aplicabilitate	3
Tehnologii utilizate	4
JavaScript	4
TypeScript	4
Node.js	4
React	4
HTML	5
CSS	5
Vite	5
Visual Studio Code	6
Descrierea aplicației	6
Prezentarea endpoint-urilor	6
Implementarea aplicației	8
Funcționalitate:	21

Introducere

Călătoriile au devenit o parte intrinsecă a vieții moderne, fie că sunteți în căutarea unei escapade relaxante, a unei aventuri pline de adrenalină sau într-o călătorie de afaceri. Cu toate acestea, alegerea și rezervarea unui hotel pot reprezenta uneori o provocare.

Aplicație web de căutare a hotelurilor a fost dezvoltată cu scopul de a oferi utilizatorilor o experiență simplă și eficientă în căutarea și vizualizarea detaliilor despre diverse hoteluri.

Obiective

Principalele obiective ale aplicației includ:

- **Ușurința în Căutare:** Furnizarea unei interfețe prietenoase și intuitive pentru a facilita căutarea rapidă și precisă a hotelurilor.
- **Informații Detaliare:** Asigurarea accesului la informații detaliate despre fiecare hotel, inclusiv imagini relevante și recenzii verificate.
- **Sortare și Filtrare Eficiente:** Oferirea posibilității utilizatorilor de a sorta și filtra rezultatele în funcție de criterii precum prețul, recenziiile, etc.
- **Interfață Responsive:** Dezvoltarea unei interfețe responsive pentru a garanta o experiență uniformă și plăcută pe diferite dispozitive.

Aplicabilitate

Aplicația este destinată tuturor celor care călătoresc și doresc să găsească rapid și eficient opțiuni de cazare în funcție de preferințele lor. Indiferent dacă sunteți un călător de afaceri, un turist pasionat sau un planificator meticulos al vacanțelor, aplicația noastră de căutare hoteluri vă poate oferi suportul necesar în luarea deciziilor informate privind opțiunile de cazare.

Tehnologii utilizate

JavaScript

JavaScript este un limbaj de programare high-level, care este folosit pentru a crea pagini web dinamice. A fost inițial dezvoltat pentru a fi executat în browser, dar cu timpul, a devenit un limbaj de programare dinamic și versatil, fiind utilizat și în afara mediului de browser. JavaScript utilizează un model de programare asincronă pentru a gestiona operațiile care pot dura o perioadă mai lungă de timp, cum ar fi solicitările către servere. (1)

TypeScript

TypeScript este un limbaj de programare high-level, gratuit și open-source, dezvoltat de Microsoft, care adaugă tipizare statică cu adnotări de tip opționale la JavaScript. Este conceput pentru dezvoltarea de aplicații mari și transpilează către JavaScript. Deoarece TypeScript este un superset al JavaScript, toate programele JavaScript sunt sintactic valide în TypeScript, dar pot eșua în verificarea tipurilor din motive de siguranță. (2)

Node.js

Node.js este un mediu de execuție JavaScript open-source și multiplatformă. Este o unealtă populară pentru aproape orice tip de proiect. Node.js rulează motorul JavaScript V8, nucleul browser-ului Google Chrome, în afara browser-ului. Acest lucru permite Node.js să aibă o performanță foarte bună.

O aplicație Node.js rulează într-un singur proces, fără a crea un fir nou pentru fiecare solicitare. Node.js furnizează un set de primitive de I/O asincrone în biblioteca sa standard, care previne blocarea codului JavaScript și, în general, bibliotecile în Node.js sunt scrise folosind paradigme non-blocking, făcând ca comportamentul blocant să fie mai degrabă o excepție decât o normă. (3)

React

React (cunoscut și sub numele de React.js sau ReactJS) este o bibliotecă gratuită și open-source de JavaScript pentru dezvoltarea interfețelor utilizator bazate pe componente. Este întreținută de Meta (fostul Facebook) și o comunitate de dezvoltatori individuali și companii.

React poate fi folosit pentru dezvoltarea de aplicații cu o singură pagină, aplicații mobile sau aplicații redă-te pe server cu framework-uri precum Next.js. Deoarece React se ocupă doar de interfața utilizatorului și de randarea componentelor în DOM, aplicațiile React depind adesea de biblioteci pentru rutare și alte funcționalități client-side. Un avantaj-cheie al React este că re-renderizează doar acele părți ale paginii care s-au schimbat, evitând re-renderizarea inutilă a elementelor DOM (Document Object Model) neschimbate. (4)

Aplicațiile React sunt constituite dintr-un singur fișier HTML, toate celelalte pagini sunt create dinamic folosind JavaScript. Astfel este posibilă crearea de componente reutilizabile JavaScript ce sunt apoi traduse în elemente native HTML. (5)

HTML

HTML este blocul de bază al unei pagini Web. Acesta definește doar structura și conținutul unei pagini.

"Hypertext" se refera la linkuri ce conectează paginile Web între ele. Folosind acest procedeu, doar apăsând un click pe un text special se face trecerea de la o pagină web la altă pagina din cadrul acelui website sau a unui website diferit. Un document HTML este format din mai multe elemente numite taguri. În funcție de tagul ales, se schimbă felul în care este afișat conținutul în pagină. (6)

CSS

CSS este un limbaj folosit pentru a descrie felul în care elementele HTML sunt afișate în pagină. În CSS se poate specifica poziționarea, aspectul și îmbinarea elementelor.

Aplicarea stilurilor definite într-o clasă CSS se face aplicând attribute elementelor HTML. Astfel se pot defini selectori pentru a modifica în mod general aspectul unor elemente, sau se poate aplica un stil unui element specific. Crearea de selectori permite definirea de reguli de aplicare a stilului pentru a oferi flexibilitate în stilizarea paginii. (7)

Vite

Vite este o unealtă de construire care își propune să ofere o experiență de dezvoltare mai rapidă și mai eficientă pentru proiectele web moderne. Este compusă din două părți majore:

- Un server de dezvoltare care oferă îmbunătățiri semnificative față de modulele native ES, cum ar fi înlocuirea extrem de rapidă a modulelor (HMR - Hot Module Replacement).

- O comandă de construire care împachetează codul tău cu Rollup, configurat preînregistrat pentru a produce active statice extrem de optimizate pentru producție.

Vite are o abordare bine definită și vine cu setări sensibile din start. Poți citi despre ce este posibil în Ghidul de Funcționalități. Suportul pentru cadre sau integrarea cu alte instrumente este posibilă prin intermediul Plugin-urilor. (8)

Visual Studio Code

Visual Studio este un mediu de dezvoltare integrat (IDE) creat de Microsoft. Este utilizat în principal pentru dezvoltarea de software, incluzând dezvoltarea aplicațiilor desktop, web, mobile și cloud.

Acesta vine echipat cu un editor de cod avansat, care oferă funcționalități puternice precum evidențierea sintaxei, completarea automată, navigarea rapidă și refactoring-ul codului. Acesta suportă o gamă largă de limbaje de programare, inclusiv C#, C++, JavaScript, Python și multe altele.

Visual Studio este considerat un mediu de dezvoltare robust și eficient pentru o varietate de proiecte și limbaje de programare.

Descrierea aplicației

Această aplicație web dezvoltată în React are scopul de a oferi utilizatorilor posibilitatea de a căuta hoteluri ce se vor afișa sortate în funcție de diferite criterii, facilitând astfel procesul de planificare a călătoriilor.

Informațiile despre hoteluri sunt furnizate cu ajutorul API-ului Hotels by Api Dojo:

<https://rapidapi.com/apidojo/api/hotels4>

Prezentarea endpoint-urilor

Pentru a putea obține informațiile necesare am utilizat următoarele endpoint-uri ale API-ului:

- **v2/get-meta-data:**

- Acesta furnizează informații referitoare la limba utilizată în cadrul site-ului dedicat hotelurilor. Cu scopul de a optimiza eficiența aplicației, am optat pentru includerea directă a acestor informații în cod. Această abordare este adoptată întrucât utilizarea exclusivă a versiunii în limba engleză a site-ului elimină necesitatea de a accesa aceste date de la server în mod dinamic la fiecare solicitare.

- **location/v3/search:**
 - Endpont-ul este accesat prin furnizarea numelui orașului destinație, urmând să se extragă parametrul regionId pe care îl voi folosi pentru a apela următoarele endpoint-uri.

- **properties/v2/list:**
 - Acesta primește ca parametrii informațiile extrase din endpoint-urile anterioare, data sosirii, data plecării, numărul de persoane dar și modalitatea în care se vor afișa rezultatele prin intermediul parametrului "sort". În urma apelului se va stoca un array de hoteluri din care vom extrage id-ul fiecăruia.

(Parametrul "sort" poate avea următoarele valori:

PRICE_RELEVANT (Price + our picks)

REVIEW (Guest rating)

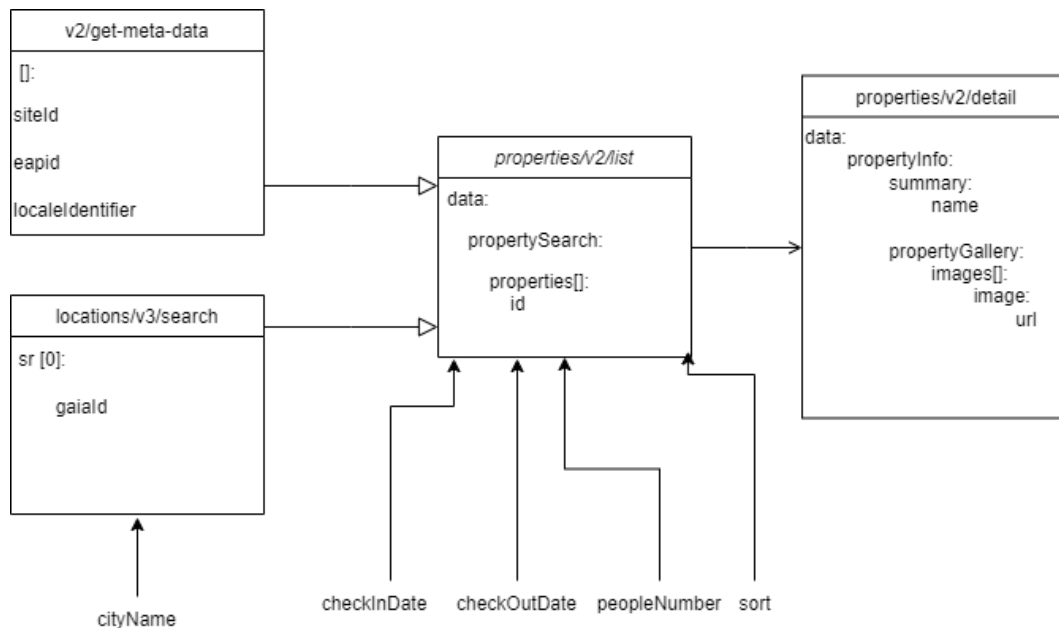
DISTANCE (Distance from downtown)|

PRICE_LOW_TO_HIGH (Price)

PROPERTY_CLASS (Star rating)

RECOMMENDED (Recommended)) (9)

- **properties/v2/detail:**
 - Endpont-ul primește ca parametrii informațiile extrase din v2/get-meta-data și id-ul unui hotel, urmând să întoarcă detalii despre respectivul hotel (review-uri, descrierea camerelor, imagini). Se va extrage și afișa numele fiecărui hotel împreună cu imaginile acestuia.



1. Diagrama rutelor

Implementarea aplicației

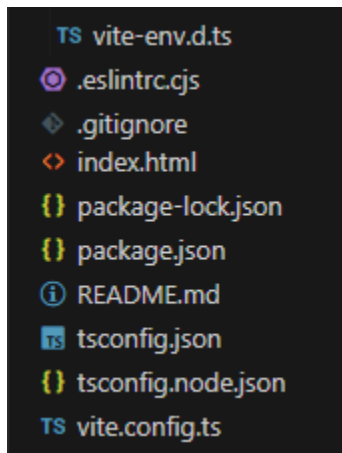
Pentru a iniția un nou proiect utilizând Vite în mediul de dezvoltare Visual Studio Code, se impune executarea comenzii `npm init vite 'numele_proiectului'` în terminal. În cadrul acestei proceduri, se va solicita specificarea framework-ului pentru noul proiect (React) și limbajul de programare (TypeScript).

```

PS C:\Mihai\Programare\JavaScript_Udemy\Code> npm init vite exemplu_documentatie
? Select a framework: » - Use arrow-keys. Return to submit.
> Vanilla
  Vue
  React
  Preact
  Lit
  Svelte
  Solid
  Qwik
  Others
  
```

2. Alegerea framework-ului pentru proiectul Vite

Apoi, în directorul în care proiectul a fost creat, se execută în terminal comanda `npm install`, care va instala dependențele specificate în fișierul "package.json". În urma executării comenzilor de inițializare următoarele fișiere vor apărea în proiect:



3 Fișiere de dependențe

Pentru a porni serverul local de dezvoltare se rulează comanda `npm run dev`. Acesta monitorizează fișierele sursă și reîncarcă automat aplicația la apariția schimbărilor în cod.

Pentru implementarea aplicației am realizat următoarele fișiere:

locationsCall.tsx:

```
src > TS locationsCall.tsx > getLocations > options > headers > 'X-RapidAPI-Key'
1  import axios from "axios";
2
3  export async function getLocations(regionName:String) {
4      const options = {
5          method: 'GET',
6          url: 'https://hotels4.p.rapidapi.com/locations/v3/search',
7          params: {
8              q: regionName,
9              locale: 'en_US',
10             langid: '1033',
11             siteid: '300000001'
12         },
13         headers: {
14             'X-RapidAPI-Key': '16f6894920msh2be1dfcd82b9b42p1614b2jsn31772ace51f9',
15             'X-RapidAPI-Host': 'hotels4.p.rapidapi.com'
16         }
17     };
18
19     try {
20         const response = await axios.request(options);
21         return await response.data;
22     } catch (error) {
23         console.error(error);
24     }
25 }
```

4. Funcția `getLocations`

Funcția `locationsCall.tsx` realizează o cerere către API, folosind biblioteca `axios` pentru a obține informații despre locația introdusă de utilizator.

propertiesListCall.tsx:

```
src > ts propertiesListCall.tsx > getPropertiesList
1 import axios from "axios";
2
3 export async function getPropertiesList(regionId: String, checkInDay: number, checkInMonth: number, checkInYear: number,
4 checkOutDay: number, checkOutMonth: number, checkOutYear: number, peopleNumber: number, sortMethod: String) {
5     const options = {
6         method: 'POST',
7         url: 'https://hotels4.p.rapidapi.com/properties/v2/list',
8         headers: {
9             'content-type': 'application/json',
10            'X-RapidAPI-Key': '16f6894920msh2be1dfcd82b9b42p1614b2jsn31772ace51f9',
11            'X-RapidAPI-Host': 'hotels4.p.rapidapi.com'
12        },
13        data: {
14            currency: 'USD',
15            eapid: 1,
16            locale: 'en_US',
17            siteId: 300000001,
18            destination: {
19                regionId: regionId
20            },
21            checkInDate: {
22                day: checkInDay,
23                month: checkInMonth,
24                year: checkInYear
25            },
26            checkOutDate: {
27                day: checkOutDay,
28                month: checkOutMonth,
29                year: checkOutYear
30            },
31            rooms: [
32                {
33                    adults: peopleNumber,
34                    children: [{age: 5}, {age: 7}]
35                }
36            ],
37            resultsStartingIndex: 0,
38            resultsSize: 200,
39            sort: sortMethod,
40            filters: {
41                price: {max: 150, min: 100}
42            }
43        }
44    };
45
46    try {
47        const response = await axios.request(options);
48        return await response.data;
49    } catch (error) {
50        console.error(error);
51    }
52 }
```

5. Funcția *getPropertiesList*

Funcția *getPropertiesList* este o funcție asincronă care efectuează o cerere POST către API-ul serviciului hotelier pentru a obține o listă de proprietăți (hoteluri) în funcție de diverse criterii.

detailsCall.tsx:

```
src > TS detailsCall.tsx > getDetails
1  import axios from "axios";
2
3  export async function getDetails(hotelId: String) {
4
5      const options = {
6          method: 'POST',
7          url: 'https://hotels4.p.rapidapi.com/properties/v2/detail',
8          headers: {
9              'content-type': 'application/json',
10             'X-RapidAPI-Key': '16f6894920msh2be1dfcd82b9b42p1614b2jsn31772ace51f9',
11             'X-RapidAPI-Host': 'hotels4.p.rapidapi.com'
12         },
13         data: {
14             currency: 'USD',
15             eapid: 1,
16             locale: 'en_US',
17             siteId: 300000001,
18             propertyId: hotelId
19         }
20     };
21
22     try {
23         const response = await axios.request(options);
24         return await response.data;
25     } catch (error) {
26         console.error(error);
27     }
28 }
```

6. Funcția *getDetails*

Funcția *getDetails* este o funcție asincronă care efectuează o cerere POST către API-ul serviciului hotelier pentru a obține detalii specifice despre un hotel, identificat printr-un ID specific.

Call-urile API-ului folosesc biblioteca JavaScript *axios*. Aceasta este utilizată pentru a efectua cereri HTTP din browser sau din Node.js.

hotel.tsx:

```
src > components > TS hotel.tsx > HotelComponent
1  import {useState} from "react";
2
3  import "../styling/hotelComponent.css"
4
5  export const HotelComponent = (hotel: Hotel) => {
6      const [currentImageIndex, setCurrentImageIndex] = useState(0);
7
8      const nextImage = () => {
9          setCurrentImageIndex((prevIndex) => (prevIndex + 1) % hotel.images.length);
10     };
11
12     const prevImage = () => {
13         setCurrentImageIndex((prevIndex) => (prevIndex - 1 + hotel.images.length) % hotel.images.length);
14     };
15
16     return (
17         <div>
18             <h2>
19                 {hotel.name}:
20             </h2>
21             <div className="image-overlay" onClick={nextImage}>
22                 <img src={hotel.images[currentImageIndex]} />
23                 <div className="prev-arrow" onClick={(e) => { e.stopPropagation(); prevImage(); }}>&#8249;</div>
24                 <div className="next-arrow" onClick={(e) => { e.stopPropagation(); nextImage(); }}>&#8250;</div>
25             </div>
26         </div>
27     );
28 };
29
30 export class Hotel {
31     name: string
32     images: string[]
33
34     constructor(name: string, images: string[]) {
35         this.name = name;
36         this.images = images;
37     }
38 }
```

7. Componenta hotel

Acest cod reprezintă o componentă React (HotelComponent) care afișează informații despre un hotel, precum numele și o galerie de imagini ale acestuia, prin care se poate naviga cu ajutorul funcțiilor *nextImage* și *prevImage*. De asemenea, sunt definite o funcție de hook (care permite componentelor funcționale să acceseze caracteristici specifice componentelor de clasă) *useState* și o clasă *Hotel* pentru a gestiona starea și a defini structura unui obiect hotel.

App.tsx:

Acest fișier conține funcționalitatea propriu-zisă a aplicației. Am început prin importarea bibliotecilor necesare:

```
import { ChangeEvent, FormEvent, useState } from "react";
import { getLocations } from "../locationsCall";
import { getPropertiesList } from "../propertiesListCall";
import { getDetails } from "../detailsCall";
import { Hotel, HotelComponent } from "../components/hotel";
```

8. Bibliotecile din App.tsx

- **useState:**
 - useState este un hook din React utilizat pentru a gestiona starea într-o componentă funcțională. Starea reprezintă informații care pot fi modificate în timpul vieții componentei și determină cum arată și se comportă componenta într-un anumit moment. useState returnează o pereche formată dintr-o valoare curentă a stării și o funcție care permite modificarea acestei stări. (10)
- **ChangeEvent:**
 - ChangeEvent este un tip de eveniment utilizat în React pentru a gestiona schimbările în elementele de intrare, cum ar fi câmpurile de text. Este utilizat în principal în manipulatoarele de evenimente pentru a accesa și a gestiona valorile introduse de utilizator în câmpurile de intrare. (11)
- **FormEvent:**
 - FormEvent este un tip de eveniment care are loc ori de câte ori un formular sau un element de formular primește sau modifică valoarea unui element sau la trimiterea formularului.

Funcția *locations* este o funcție asincronă ce primește ca parametru un String, reprezentând numele orașului destinație. Funcția folosește un bloc try-catch pentru a gestiona eventualele erori care pot apărea în timpul execuției.

În blocul try, funcția folosește await pentru a aștepta rezultatul unei promisiuni returnate de funcția *getLocations(city)*. După rezolvarea promisiunii, se extrage proprietatea *sr* din obiectul rezultat și se atribuie variabilei *locationList* din care extragem id-ul orașului în variabila *regionId*.

```

async function locations(city: String) {
  try {
    var locationList = await getLocations(city).then(response => response.sr);
    const regionId = locationList[0].gaiaId;
    return regionId;
  } catch (error) {
    console.error(error);
  }
}

```

9. Funcția locations

Funcția asincronă *propertiesList* este asemănătoare celei prezentate anterior. Aceasta primește id-ul orașului destinație, data sosirii, data plecării, numărul de persoane și metoda de sortare și întoarce lista de hoteluri corespunzătoare.

```

async function propertiesList(regionId: any, checkInDay: number, checkInMonth: number, checkInYear: number,
checkOutDay: number, checkOutMonth: number, checkOutYear: number, people: number, sortMethod: String) {
  try {
    const propertiesList: Array<any> = await getPropertiesList(regionId, checkInDay, checkInMonth, checkInYear, checkOutDay, checkOutMonth, checkOutYear, people, sortMethod)
      .then(response => response.data.propertySearch.properties);
    return propertiesList;
  } catch (error) {
    console.error(error);
  }
}

```

10. Funcția propertiesList

propertyDetails are o funcționalitate similară cu cea a funcțiilor prezentate mai sus. Aceasta primește ca parametru lista de hoteluri obținută anterior și întoarce un array ce conține detaliile hotelurilor care urmează să fie afișate (nume, array de imagini).

```

async function propertyDetails(hotelIds) {
  var promiseList = [];
  var idsList = [];
  hotelIds.forEach(element => {
    idsList.push(element.id + "");
  });
  for (var i = 0; i < 5; i++) {
    promiseList.push(await getDetails(idsList[i]).then(response => response));
  }
  return promiseList;
}

```

11. Funcția propertyDetails

În funcția principală *function App()* declar variabilele de stare:

```
function App() {
  const [city, setCity] = useState<String>("");
  const [checkInDate, setCheckInDate] = useState<Date>();
  const [checkOutDate, setCheckOutDate] = useState<Date>();
  const [peopleNumber, setPeopleNumber] = useState<number>();
  const [sortMethod, setSortMethod] = useState<String>("");
  const [hotelList, setHotelList] = useState<Hotel[]>([]);
```

12. Variabilele de stare din function App

- **city**: Reprezintă orașul introdus de utilizator.
- checkInDate: Reprezintă data selectată pentru check-in.
- checkOutDate: Reprezintă data selectată pentru check-out.
- peopleNumber: Reprezintă numărul de persoane pentru rezervare.
- sortMethod: Reprezintă metoda de sortare selectată pentru rezultatele hotelurilor.
- hotelList: Reprezintă lista de hoteluri care urmează să fie afișate.

Acestea sunt modificate cu ajutorul funcțiilor care gestionează evenimentele de introducere a datelor de la tastatură în interfața utilizatorului. Ele primesc un obiect de eveniment (ChangeEvent) care reprezintă evenimentul de schimbare a conținutului unui element de tip input (<HTMLInputElement >).

```
const handleCityInput = (city: ChangeEvent<HTMLInputElement>) => {
  setCity(city.target.value);
}

const handleCheckInInput = (checkInDate: ChangeEvent<HTMLInputElement>) => {
  const newCheckInDate = new Date(checkInDate.target.value);
  setCheckInDate(newCheckInDate);
}

const handleCheckOutInput = (checkOutDate: ChangeEvent<HTMLInputElement>) => {
  const newCheckOutDate = new Date(checkOutDate.target.value);
  setCheckOutDate(newCheckOutDate);
}

const handlePeopleNumber = (peopleNumber: ChangeEvent<HTMLInputElement>) => {
  setPeopleNumber(parseInt(peopleNumber.target.value));
}

const handleSort = (sort: ChangeEvent<HTMLInputElement>) => {
  setSortMethod(sort.target.value);
}
```

13. Funcțiile de modificare a stărilor

Manipularea evenimentelor se efectuează cu ajutorul metodelor `handleFormChange` și `handleSortChange`. Acestea sunt declanșate la trimiterea formularului principal, respectiv al celui de sortare prezente în structura de interfață a componentei `App`, prezentată mai jos, și apelează funcția `hotelSearch`.

```
const handleFormChange = (event: FormEvent<HTMLFormElement>) => {
  event.preventDefault();
  hotelSearch();
};

const handleSortChange = (event: FormEvent<HTMLFormElement>) => {
  event.preventDefault();
  hotelSearch();
};
```

14. Manipularea evenimentelor

Căutarea propriu-zisă a hotelurilor se realizează cu ajutorul funcției `hotelSearch`. Pentru verificarea stărilor variabilelor declarate mai sus, afișez în consolă valorile acestora.

Apoi, se apelează funcțiile asincrone descrise anterior într-un lanț de promisiuni. Prima oară se va apela funcția `locations` cu parametrul `city`, care va returna o promisiune cu `regionId`. Vom folosi parametrul returnat, împreună cu alte informații introduse de utilizator pentru a apela `propertiesList` ce returnează o altă promisiune cu lista de proprietăți. În final, se apelează funcția `propertyDetails` ce întoarce o promisiune cu detalii despre hoteluri. Pentru fiecare hotel se extrag imaginile și numele acestuia într-o variabilă de tip `Hotel`. Acest obiect este ulterior adăugat la lista `hotelList` folosind `setHotelList`.

```
const hotelSearch = () => {
  console.log(city);
  console.log(checkInDate?.getDate());
  console.log(checkInDate?.getMonth() + 1);
  console.log(checkInDate?.getFullYear());
  console.log(checkOutDate?.getDate());
  console.log(checkOutDate?.getMonth() + 1);
  console.log(checkOutDate?.getFullYear());
  console.log(peopleNumber);
  console.log(sortMethod);

  const checkInMonth = checkInDate?.getMonth() + 1;
  const checkOutMonth = checkOutDate?.getMonth() + 1;
  setHotelList([]);

  locations(city)
    .then(regionId => {
      console.log(regionId);
      return propertiesList(regionId, checkInDate?.getDate(), checkInMonth, checkInDate?.getFullYear(), checkOutDate?.getDate(), checkOutMonth, checkOutDate?.getFullYear(), peopleNumber, sortMethod);
    })
    .then(propertyList => {
      console.log(propertyList);
      return propertyDetails(propertyList);
    })
    .then(hotelDetailsList => {
      console.log(hotelDetailsList);
      hotelDetailsList.forEach(detail => {
        var image = detail.data.propertyInfo.propertyGallery.images.map(image => image.image.url);
        var hotel = new Hotel(detail.data.propertyInfo.summary.name, image);
        console.log(hotel);
        setHotelList((hotelList) => [...hotelList, hotel]);
      });
    });
};
```

15. Funcția `hotelSearch`

App.tsx returnează interfața aplicației și afișează rezultatele căutării. Aceasta încadrează întregul conținut al aplicației într-un container de tip `<div>` centrat cu ajutorul elementelor CSS.

Formularul de căutare (`<form>`) permite utilizatorului să introducă informații pentru căutarea de hoteluri. Acest formular este gestionat de funcția `handleFormChange` atunci când este depus (`onSubmit`). Câmpurile de intrare sunt legate de starea componentelor (`city`, `checkInDate`, `checkOutDate`, `peopleNumber`, `sortMethod`), asigurând actualizarea dinamică a interfeței în funcție de introducerea utilizatorului. Pentru declanșarea căutării este inclus un buton de tip submit "Search".

Dropdown-ul (`<select>`) oferă opțiuni de sortare pentru utilizator. Schimbările acestuia declanșează funcția `handleSort`, iar butonul de tip submit "Apply" aplică aceste schimbări prin apelul funcției `handleSortChange`.

Afișarea rezultatelor se realizează cu ajutorul funcției `map` ce interează prin fiecare obiect `Hotel` din `hotelList`.

```
return (
  <>
    <div className="center-div">
      <h1>Please enter the location you want to travel to: </h1>
      <div className="forms-div">
        <form className="forms-div" onSubmit={handleFormChange}>
          <input
            type="text"
            name="city"
            value={city || ""}
            onChange={handleCityInput}
            placeholder="Enter city name"
          />
          <input
            type="date"
            name="check in"
            value={formatDate(checkInDate) || ""}
            onChange={handleCheckInInput}
            placeholder="Enter check-in date"
          />
          <input
            type="date"
            name="check out"
            value={formatDate(checkOutDate) || ""}
            onChange={handleCheckOutInput}
            placeholder="Enter check-out date"
          />
          <input
            type="number"
            name="check in"
            value={peopleNumber}
            onChange={handlePeopleNumber}
            placeholder="How many people"
          />
          <button type="submit">Search</button>
          <div className="forms-div" onSubmit={handleSortChange}>
            <p>Sort by:</p>
            <select id="dropdown" onChange={handleSort} >
              <option value="PRICE_LOW_TO_HIGH">Lowest price</option>
              <option value="PRICE_RELEVANT">Relevance</option>
              <option value="REVIEW">Review</option>
              <option value="PROPERTY_CLASS">Star rating</option>
            </select>
            <button type="submit">Apply</button>
          </div>
        </form>
      </div>
      {hotelList.map((hotel: Hotel) => (
        <HotelComponent key={hotel.name} name={hotel.name} images={hotel.images} />
      ))}
    </div>
  </>
);
```

16. Interfața aplicației și afișarea rezultatelor

Pentru ca formatul datelor calendaristice de sosire și plecare să fie corect, am utilizat funcția `formatDate`.

```
const formatDate = (date: Date | null): string => {
  if (date) {
    const year = date.getFullYear();
    const month = String(date.getMonth() + 1).padStart(2, '0');
    const day = String(date.getDate()).padStart(2, '0');
    return `${year}-${month}-${day}`;
  }
  return '';
};
```

17. Funcția `formatDate`

Stilizarea CSS pentru pagina de căutare APP.tsx:

```
src > styling > # App.css > .forms-div input
1  h1{
2    text-align: center;
3  }
4
5  form {
6    max-width: auto;
7    padding: 20px;
8    background-color: #f4f4f4;
9    border-radius: 5px;
10   box-shadow: 0 0 10px rgba(0,0,0,0.2);
11 }
12
13 .forms-div {
14   display: flex;
15   flex-direction: column;
16   align-items: center;
17   justify-content: center;
18   text-align: center;
19 }
20
21 .forms-div input {
22   margin-bottom: 10px;
23 }
24
25 .forms-div button {
26   padding: 10px 20px;
27   background-color: #4c90af;
28   color: #fff;
29   border: none;
30   border-radius: 4px;
31   cursor: pointer;
32 }
33
34 .forms-div button:hover {
35   background-color: #5645a0;
36 }
```

18. Fișierul `App.css`

Stilizarea componentei Hotel:

```
src > styling > # hotelComponent.css > .image-overlay img
1  h2{
2    text-align: center;
3  }
4
5  div{
6    text-align: center;
7  }
8
9  .image-overlay {
10   position: relative;
11   cursor: pointer;
12 }
13
14 .image-overlay img {
15   max-width: 50%;
16   max-height: auto;
17   display: block;
18   margin: 0 auto;
19 }
20
21 .prev-arrow,
22 .next-arrow {
23   position: absolute;
24   top: 50%;
25   transform: translateY(-50%);
26   font-size: 50px;
27   color: rgb(33, 161, 212);
28   cursor: pointer;
29 }
31 .prev-arrow {
32   left: 300px;
33 }
34
35 .next-arrow {
36   right: 300px;
37 }
38
39 .prev-arrow:hover,
40 .next-arrow:hover {
41   color: #011dc0;
42 }
43
44
45 button {
46   padding: 10px 20px;
47   background-color: #4c90af;
48   color: #fff;
49   border: none;
50   border-radius: 4px;
51   cursor: pointer;
52   align-self: center;
53 }
54
55 button:hover {
56   background-color: #5645a0;
57 }
```

19. Fișierul hotelComponent.css

Acest document CSS stilizează componenta Hotel. Se observă faptul că săgețile reprezintă elemente interactive cu ajutorul cărora putem schimba imaginea care se vizualizează în aplicație (cursor: pointer).

main.tsx:

Funcția aceasta realizează render-ul aplicației React, plasând-o în elementul cu id-ul 'root' din fișierul HTML principal.

```
src > TS main.tsx
1  import React from 'react'
2  import ReactDOM from 'react-dom/client'
3  import App from './App.tsx'
4
5  ReactDOM.createRoot(document.getElementById('root')!).render(
6    <React.StrictMode>
7      <App />
8    </React.StrictMode>,
9  )
```

20. Fișierul main.tsx

index.html:

Acesta este fișierul HTML principal, care servește drept container pentru aplicația web creată.

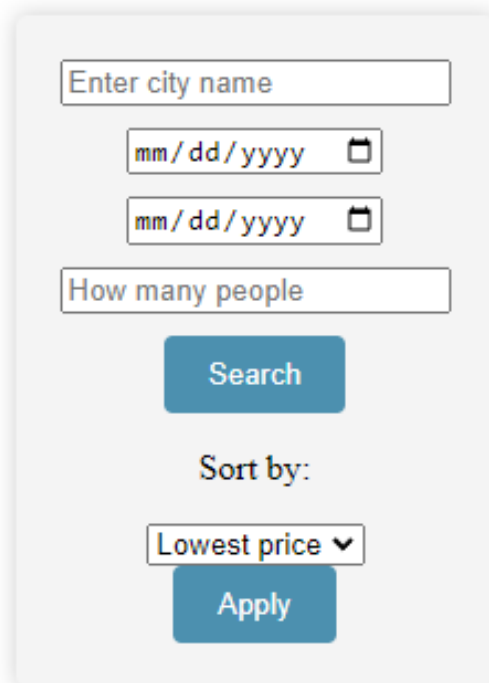
```
<> index.html > ...
1  <!doctype html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>Proiect 3</title>
8    </head>
9    <body>
10     <div id="root"></div>
11     <script type="module" src="/src/main.tsx"></script>
12   </body>
13 </html>
```

21. Fișierul index.html

Funcționalitate:

Interfața aplicației:

Please enter the location you want to travel to:



Enter city name

mm/dd/yyyy

mm/dd/yyyy

How many people

Search

Sort by:

Lowest price ▼

Apply

22. Interfața aplicației

Căutarea hotelurilor:

Sort by:

Grand Hotel De Turin:



Hotel Antin Saint-Georges:



23. Căutarea hotelurilor

Schimbarea imaginilor:



Grand Hotel De Turin:



Grand Hotel De Turin:



24. Schimbarea imaginilor hotelului

Sortarea hotelurilor:

Sort by:

Hotel Le Trente:

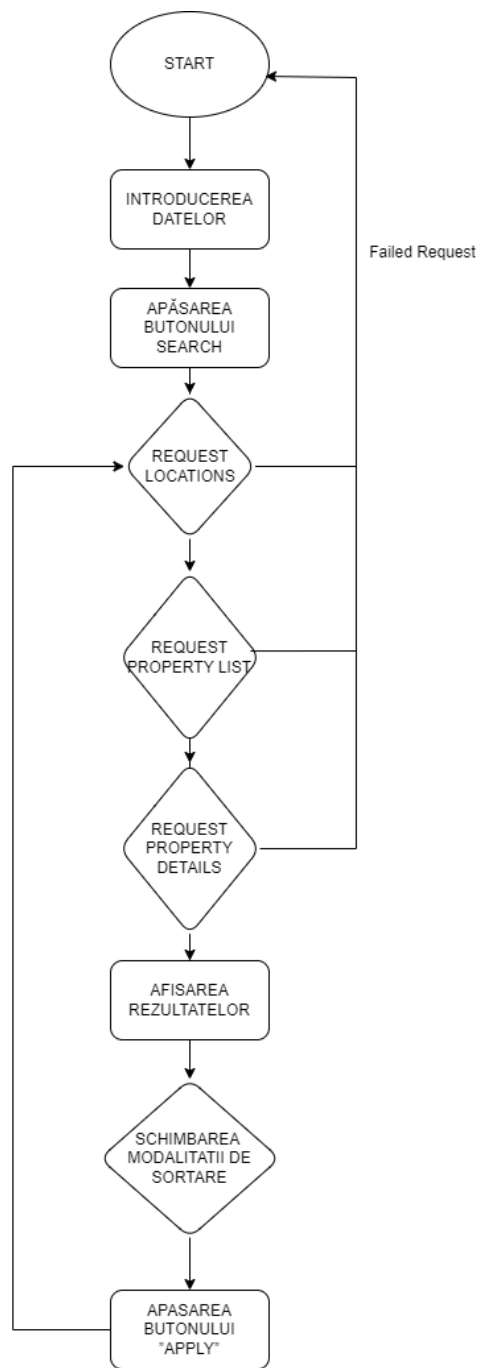


Grand Hotel De Turin:



25. Sortarea hotelurilor după review-uri

Diagrama funcționalității:



26. Diagrama funcționalității

Bibliografie

1. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
2. <https://en.wikipedia.org/wiki/TypeScript>
3. <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>
4. [https://en.wikipedia.org/wiki/React_\(software\)](https://en.wikipedia.org/wiki/React_(software))
5. <https://legacy.reactjs.org/docs/getting-started.html>
6. <https://developer.mozilla.org/en-US/docs/Web/HTML>
7. <https://developer.mozilla.org/en-US/docs/Web/CSS>
8. <https://vitejs.dev/guide/>
9. <https://rapidapi.com/apidojo/api/hotels4>
10. https://www.w3schools.com/react/react_usestate.asp
11. https://react-typescript-cheatsheet.netlify.app/docs/basic/getting-started/forms_and_events/