МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «УНІВЕРСИТЕТ «КРОК»

Фаховий коледж Університету «КРОК»

ДИПЛОМНА РОБОТА

за темою

«Створення сайту для пошуків фільмів

Студент 4 курсу групи ІПЗ-20к

Кудряшов Костянтин Дмитрович

(прізвище, ім'я та по-батькові студента)

Керівник дипломної роботи

Кириченко Віктор Вікторович

(прізвище, ім'я та по-батькові керівника)

До захисту

(резолюція «До захисту»)

_____ 12.06.2024 _____

(підпис студента) (дата) (підпис викладача)

Київ, 2024

Скорочення

1. **API** - Application Programming Interface
2. **TMDb** - The Movie Database
3. **HTTP** - HyperText Transfer Protocol
4. **GET** - HTTP метод запиту для отримання даних
5. **SPA** - Single Page Application
6. **URL** - Uniform Resource Locator
7. **HD** - High Definition
8. **4K** - Роздільна здатність відео 4096x2160 пікселів
9. **GH** - GitHub
10. **JSX** - JavaScript XML
11. **JS** - JavaScript
12. **CSS** - Cascading Style Sheets
13. **SPA** - Single Page Application
14. **DOM** - Document Object Model
15. **CRUD** - Create, Read, Update, Delete
16. **UI** - User Interface
17. **UX** - User Experience
18. **JSON** - JavaScript Object Notation
19. **SDK** - Software Development Kit
20. **REST** - Representational State Transfer

Зміст

Вступ

Сучасна кіноіндустрія займає важливе місце в житті багатьох людей, надаючи їм можливість насолоджуватися різноманітними кінематографічними творами та занурюватися в багатогранний світ кіно. У цьому контексті глядачі прагнуть мати зручний та швидкий доступ до інформації про фільми, включаючи деталі про акторський склад, рецензії та оцінки. Відсутність зручних інструментів для задоволення цих потреб стала основною мотивацією для створення мною інноваційного веб-сайту, який виступає як онлайн-бібліотека фільмів.

Мій проект зосереджується на функціональній частині, забезпечуючи надійну інтеграцію бекенду з базою даних для ефективного та швидкого надання необхідної інформації користувачам. Завдяки цьому сайту, користувачі отримують можливість легко знаходити будь-який фільм за кілька кліків, переглядати повний список акторів, а також читати рецензії та оцінки.

Особлива увага була приділена ретельному проектуванню архітектури бекенду та оптимізації бази даних, що дозволяє забезпечити високу швидкість та надійність роботи сайту. Це сприяє не лише зручності користування, але й підвищенню рівня задоволення користувачів, роблячи сайт незамінним інструментом для всіх кінолюбителів.

У моїй дипломній роботі я детально розгляну аспекти розробки цього веб-сайту, включаючи вибір технологій, архітектуру системи, процес інтеграції з базою даних, а також тестування та оптимізацію. Проект демонструє мої знання та навички у сфері веб-розробки, а також моє прагнення створювати корисні та інноваційні рішення для сучасних користувачів. Цей веб-сайт не тільки відповідає потребам користувачів у доступі до кінематографічної інформації,

але й є прикладом моєї здатності реалізовувати складні проекти, орієнтовані на кінцевого користувача.

Таким чином, створення цього сайту-бібліотеки фільмів є не лише дипломною роботою, але й внеском у покращення доступу до кінематографічного контенту для широкої аудиторії, демонструючи важливість інтеграції сучасних технологій у повсякденне життя.

1: Огляд існуючих сайтів-бібліотек фільмів

1.1. Функціональність сайтів-бібліотек фільмів

1.1.1. Основні функції

Більшість сайтів-бібліотек фільмів надають користувачам стандартний набір функцій:

Каталогізація: Фільми впорядковані за жанрами, роками випуску, рейтингами та іншими критеріями.

Пошук: Можливість пошуку фільмів за назвою, режисером, акторами та ключовими словами.

Перегляд онлайн: Опція перегляду фільмів безпосередньо на сайті.

Збереження та списки: Користувачі можуть створювати свої списки улюблених фільмів чи тих, які планують подивитися.

Рецензії та рейтинги: Можливість перегляду та залишення відгуків про фільми, а також їх оцінювання.

1.1.2. Додаткові функції

Деякі сайти-бібліотеки пропонують розширені функції для поліпшення користувацького досвіду:

Персональні рекомендації: На основі попереднього перегляду та оцінок фільмів.

Трейлери та додатковий контент: Відео з-за лаштунків, інтерв'ю з акторами та режисерами.

Мобільні додатки: Забезпечення доступу до бібліотеки через мобільні пристрої.

Синхронізація з іншими сервісами: Інтеграція з соціальними мережами та сторонніми платформами для спільного перегляду та обговорення фільмів.

1.2. Дизайн та зручність використання

1.2.1. Інтерфейс користувача

Гарний дизайн сайту-бібліотеки фільмів повинен бути інтуїтивно зрозумілим та зручним у використанні. Основні аспекти включають:

Навігація: Чітко структурована та легко доступна, щоб користувачі могли швидко знайти потрібну інформацію.

Візуальна привабливість: Використання сучасного дизайну, якісних зображень та відео для приваблення користувачів.

Респонсівність: Адаптація до різних розмірів екранів та пристроїв.

1.2.2. Зручність користування

Зручність користування визначається такими чинниками:

Швидкість завантаження: Важливий фактор, який впливає на досвід користувача.

Простота реєстрації та входу: Можливість швидко створити обліковий запис або увійти через соціальні мережі.

Доступність інформації: Забезпечення легкого доступу до описів фільмів, трейлерів, відгуків та інших даних.

1.3. Приклади популярних сайтів-бібліотек фільмів

1.3.1. IMDb (Internet Movie Database)

IMDb є однією з найвідоміших платформ, яка надає інформацію про фільми, телевізійні шоу, акторів і знімальні групи. Вона пропонує користувачам можливість переглядати трейлери, читати відгуки та оцінювати фільми. Однак, можливість перегляду фільмів безпосередньо на IMDb обмежена.

1.3.2. Netflix

Netflix є прикладом платформи, яка поєднує функціональність бібліотеки фільмів із сервісом потокового відео. Він надає широкий вибір фільмів і серіалів, персональні рекомендації та можливість завантажувати контент для офлайн-перегляду.

1.3.3. Letterboxd

Letterboxd зосереджений на соціальному аспекті перегляду фільмів. Користувачі можуть створювати списки, оцінювати фільми та залишати рецензії, а також слідкувати за активністю своїх друзів та інших користувачів.

2: Аналіз аудиторії та їхніх потреб

2.1. Профіль аудиторії

2.1.1. Демографічні характеристики

Цільова аудиторія сайтів-бібліотек фільмів може включати широке коло користувачів, однак можна виділити декілька ключових груп:

Молодь та підлітки: Вікова група 16-24 роки, яка активно використовує Інтернет для розваг.

Дорослі (25-44 роки): Користувачі, які цінують зручний доступ до кінофільмів та серіалів у вільний час.

Сімейні користувачі: Люди, які шукають контент для перегляду всією родиною.

2.1.2. Географічні та культурні аспекти

Залежно від регіону можуть відрізнятися вподобання та вимоги до контенту:

Мовні вподобання: Необхідність локалізації сайту та контенту на різні мови.

Культурні особливості: Врахування специфіки національних смаків та жанрових уподобань.

2.2. Потреби та вимоги користувачів

2.2.1. Функціональні потреби

Основні вимоги користувачів до сайту-бібліотеки фільмів включають:

Великий вибір контенту: Широка бібліотека фільмів та серіалів різних жанрів і країн виробництва.

Висока якість відео: Можливість переглядати фільми в HD та 4K якості.

Зручна навігація та пошук: Інтуїтивно зрозумілий інтерфейс, який дозволяє швидко знаходити потрібний контент.

2.2.2. Нефункціональні потреби

Крім основних функцій, користувачі також цінують:

Швидкість завантаження: Миттєвий доступ до контенту без затримок.

Мобільна доступність: Наявність мобільного додатку або оптимізованої версії сайту для смартфонів і планшетів.

Безпека та конфіденційність: Захист персональних даних користувачів та безпечні платіжні системи2.3. Чинники, що впливають на вибір платформи

2.3.1. Вартість послуг

Ціна є одним із ключових факторів при виборі сайту-бібліотеки фільмів. Більшість користувачів віддають перевагу платформам з різними тарифними планами та можливістю безкоштовного доступу до базового контенту.

2.3.2. Персоналізація та рекомендації

Системи рекомендацій, що враховують вподобання користувача, сприяють довгостроковому залученню та задоволенню від використання платформи.

2.3.3. Соціальна взаємодія

Можливість залишати відгуки, ділитися враженнями та обговорювати фільми з іншими користувачами створює додаткову цінність для платформи.

3. Цілі

На основі проведеного аналізу існуючих сайтів-бібліотек фільмів та вивчення потреб і вимог аудиторії, було прийнято рішення створити власний невеликий сайт-бібліотеку фільмів для дипломної роботи.

1) Створити робочий мікросервіс яким особисто я матиму змогу користуватись.
2) Гарно оптимізувати роботу бази даних, та зробити сайт цілком адаптивним.
3) Продемонструвати свої навички розробки на React

4. Технічне забезпечення

4.1. Використані технології

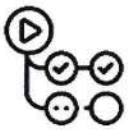
4.1.1. Основні технології

В якості мови написання я обрав React 

Програмою збереження версій обрав GIT 

Як IDE я обрав Visual Studio Code 

Для розміщення проекту я обрав GitHub 

Для запуску проекту я використовував локальний сервер та сервіс GH Actions 

4.1.2 Додаткові

- Babel
- Styled components
- React DOM
- Axios
- React loaders
- Modern Normalize
- Parcel

4.2. Технічні деталі

Мовою розробки я обрав React, бо це банально цікавіше ніж ванільний JS.

Сайт є повністю адаптивний, тобто дає змогу користувачу взаємодіяти з ним з будь якого пристрою з екраном та маніпулятором.

Використовується фреймворк Babel, для правильної роботи з Legacy.

Я використовую розробку по компонентам, тобто я пишу окремі частини проекту а потім з них збираю весь проект, так легше відлагоджувати помилки та відслідковувати прогрес. Також я використовую Styled-components, це дає змогу гнучко писати CSS стилі до кожного компонента.

Проведена робота з життєвим циклом проекту.

Також гарно налаштована робота з API.

Я використовую React хуки замість класів, це розширило можливості компонентів-функцій. Також проведена робота над маршрутизацією.

4.3. Принцип роботи

Сайт має дві сторінки, ті фільми які зараз найбільше переглядають та пошук будь якого фільму. Коли ви вибираєте фільм то сайт видає коротку інформацію про цей фільм картинку, назву та рейтинг. Далі ви самі можете подивитись каст акторів та рецензії видавництв на цей фільм.

5. Зразки та опис коду

В цьому розділі я розгляну основні аспекти коду цієї програми, а також додаю лінк на GH там ви можете подивитись весь проект одразу: My GitHub

5.1. Api.js

```
JS api.js  ×

src > components > JS api.js > ...
1   import axios from 'axios';
2
3   axios.defaults.baseURL = 'https://api.themoviedb.org/3/';
4   const key = 'd09efe7116463b03d38eef6e936584d9';
5
6   export const fetchHomepage = async () =>
7     await axios.get(`trending/movie/day?api_key=${key}`);
8
9   export const fetchMovieId = async movieId =>
10    await axios.get(`movie/${movieId}?api_key=${key}`);
11
12  export const fetchFindMovies = async query =>
13    await axios.get(`search/movie?query=${query}&api_key=${key}`);
14
15  export const fetchCast = async movieId =>
16    await axios.get(`movie/${movieId}/credits?api_key=${key}`);
17
18  export const fetchReviews = async movieId =>
19    await axios.get(`movie/${movieId}/reviews?api_key=${key}`);
20
```

```
`import axios from 'axios';

axios.defaults.baseURL = 'https://api.themoviedb.org/3/';
const key = 'd09efe7116463b03d38eef6e936584d9';

export const fetchHomepage = async () =>
  await axios.get(`trending/movie/day?api_key=${key}`);

export const fetchMovieId = async movieId =>
  await axios.get(`movie/${movieId}?api_key=${key}`);

export const fetchFindMovies = async query =>
  await axios.get(`search/movie?query=${query}&api_key=${key}`);

export const fetchCast = async movieId =>
  await axios.get(`movie/${movieId}/credits?api_key=${key}`);

export const fetchReviews = async movieId =>
  await axios.get(`movie/${movieId}/reviews?api_key=${key}`);
`
```

Цей код є прикладом використання бібліотеки Axios для виконання HTTP-запитів до API The Movie Database (TMDb). Спочатку імпортується бібліотека Axios, яка використовується для спрощення роботи з HTTP-запитами у JavaScript.

Потім задається базова URL-адреса для всіх запитів за допомогою `axios.defaults.baseURL`, що встановлює кореневу URL-адресу для подальших запитів до API TMDb. Також оголошується константа `key`, яка зберігає API-ключ для аутентифікації запитів.

Далі, код визначає кілька асинхронних функцій, які виконують різні запити до API TMDb:

1. `**fetchHomepage**` – ця функція отримує список популярних фільмів за день, використовуючи endpoint `trending/movie/day`.

2. `**fetchMovieId**` – функція приймає ідентифікатор фільму (movieId) і повертає детальну інформацію про конкретний фільм за допомогою endpoint `movie/{movieId}`.

3. `**fetchFindMovies**` – функція приймає пошуковий запит (query) і повертає результати пошуку фільмів, що відповідають цьому запиту, використовуючи endpoint `search/movie`.

4. `**fetchCast**` – ця функція приймає ідентифікатор фільму і повертає інформацію про акторський склад фільму за допомогою endpoint `movie/{movieId}/credits`.

5. `**fetchReviews**` – функція приймає ідентифікатор фільму і повертає відгуки на цей фільм за допомогою endpoint `movie/{movieId}/reviews`.

Кожна з цих функцій використовує метод `axios.get` для виконання GET-запиту до відповідного endpoint з вказаним API-ключем. Асинхронність функцій забезпечує використання ключового слова `async`, що дозволяє використовувати `await` для очікування завершення запиту і отримання результатів.

5.2. App.jsx

App.jsx

src > components > App.jsx > ...

```jsx
import { Suspense, lazy } from 'react';
import { Routes, Route, NavLink  } from 'react-router-dom';
import { Loader } from './Loader.js';
import { Nav } from './Nav.styled.js';

const Homepage = lazy(() => import('../pages/Homepage'));
const MovieDetailsPage = lazy(() => import('../pages/MovieDetailsPage'));
const MoviesPage = lazy(() => import('../pages/MoviesPage'));
const Cast = lazy(() => import('./Cast'));
const Reviews = lazy(() => import('./Reviews'));

export const App = () => {
  return (
    <>
    <Nav>
      <ul>
        <li>
          <NavLink to="/">Home</NavLink>
        </li>
        <li>
          <NavLink to="/movies">Movies</NavLink>
        </li>
      </ul>
    </Nav>
    <Suspense fallback={<Loader />}>
      <Routes>
        <Route path="/" element={<Homepage />} />
        <Route path="/movies" element={<MoviesPage />} />
        <Route path="/movies/:movieId" element={<MovieDetailsPage />}>
          <Route path="cast" element={<Cast />} />
          <Route path="reviews" element={<Reviews />} />
        </Route>
      </Routes>
    </Suspense>
  </>
  )
};
```

```
`import { Suspense, lazy } from 'react';
import { Routes, Route, NavLink  } from 'react-router-dom';
import { Loader } from './Loader.js';
import { Nav } from './Nav.styled.js';

const Homepage = lazy(() => import('../pages/Homepage'));
const MovieDetailsPage = lazy(() => import('../pages/MovieDetailsPage'));
const MoviesPage = lazy(() => import('../pages/MoviesPage'));
const Cast = lazy(() => import('./Cast'));
const Reviews = lazy(() => import('./Reviews'));

export const App = () => {
  return (
    <>
    <Nav>
     <ul>
       <li>
         <NavLink to="/">Home</NavLink>
       </li>
       <li>
         <NavLink to="/movies">Movies</NavLink>
       </li>
     </ul>
    </Nav>
    <Suspense fallback={<Loader />}>
     <Routes>
       <Route path="/" element={<Homepage />} />
```

```
        <Route path="/movies" element={<MoviesPage />} />
        <Route path="/movies/:movieId" element={<MovieDetailsPage />}>
          <Route path="cast" element={<Cast />} />
          <Route path="reviews" element={<Reviews />} />
        </Route>
      </Routes>
    </Suspense>
  </>
 )
};`
```

Цей код є прикладом використання бібліотеки React разом із React Router для створення одно-сторінкового застосунку (SPA). Застосунок дозволяє користувачам перемикатися між різними сторінками, використовуючи маршрутизацію.

На початку імпортуються необхідні модулі з **react**, **react-router-dom** та інші компоненти з локальних файлів. **Suspense** та **lazy** використовуються для динамічного завантаження компонентів, що дозволяє зменшити початковий розмір JavaScript-файлу і завантажувати компоненти тільки тоді, коли вони потрібні. **Loader** використовується для відображення індикатора завантаження під час очікування завантаження компонентів.

Далі, за допомогою **lazy** оголошуються компоненти **Homepage**, **MovieDetailsPage**, **MoviesPage**, **Cast** та **Reviews**. Ці компоненти завантажуються асинхронно, коли вони потрібні.

Компонент **App** визначає структуру застосунку. Він повертає фрагмент JSX-коду, який містить навігаційне меню (**Nav**) з посиланнями на головну сторінку та сторінку фільмів. Використовується компонент **NavLink** з **react-router-dom** для створення навігаційних посилань.

Усередині **Suspense** з вказаним **fallback**, що є компонентом **Loader**, визначаються маршрути застосунку за допомогою **Routes** та **Route** з **react-router-dom**. Головний маршрут (**"/"**) веде на компонент **Homepage**, маршрут **"/movies"** веде на компонент **MoviesPage**, а маршрут **"/movies/:movieId"** веде на компонент **MovieDetailsPage**. Внутрішні маршрути для **MovieDetailsPage** дозволяють завантажувати компоненти **Cast** та **Reviews** як вкладені маршрути.

Таким чином, цей код забезпечує базову маршрутизацію в React-застосунку з динамічним завантаженням компонентів та індикатором завантаження для покращення користувацького досвіду.

5.3. Homepage.js

Homepage.js

src > pages > Homepage.js > ...

```
import { fetchHomepage } from 'components/api';
import { useEffect, useState } from 'react';
import { Link, useLocation } from 'react-router-dom';
import { GlobalStyle } from 'components/GlobalStyle';

const Homepage = () => {
  const [movies, setMovies] = useState([]);
  const location = useLocation();

  useEffect(() => {
    try {
      fetchHomepage().then(response => {
        setMovies(response.data.results);
      });
    } catch (error) {
      console.error(error);
    }
  }, []);

  return (
    <div>
      <h2>Trending today</h2>
      <ul>
        {movies.map(({ id, title }) => (
          <li key={id}>
            <Link to={`/movies/${id}`} state={{ from: location }}>
              {title}
            </Link>
          </li>
        ))}
      </ul>
      <footer> ©2024 Created by Kostyiantyn Kudriashov </footer>
      <GlobalStyle />
    </div>

  );
};
export default Homepage;
```

```
`import { fetchHomepage } from 'components/api';
import { useEffect, useState } from 'react';
import { Link, useLocation } from 'react-router-dom';
import { GlobalStyle } from 'components/GlobalStyle';

const Homepage = () => {
 const [movies, setMovies] = useState([]);
 const location = useLocation();

 useEffect(() => {
  try {
   fetchHomepage().then(response => {
    setMovies(response.data.results);
   });
  } catch (error) {
   console.error(error);
  }
 }, []);

 return (
  <div>
   <h2>Trending today</h2>
   <ul>
    {movies.map(({ id, title }) => (
     <li key={id}>
      <Link to={`/movies/${id}`} state={{ from: location }}>
       {title}
```

```
            </Link>
          </li>
        ))}
      </ul>
      <footer> ©2024 Created by Kostyiantyn Kudriashov </footer>
      <GlobalStyle />
    </div>

  );
};
export default Homepage;`
```

Цей код представляє компонент `Homepage` у React, який відображає список популярних фільмів, використовуючи дані з API The Movie Database (TMDb). Він використовує хуки `useEffect` та `useState` для управління станом компоненту та виконання побічних ефектів.

На початку коду імпортуються необхідні модулі та функції, включаючи `fetchHomepage` для отримання даних про фільми, хуки `useEffect` та `useState` з React, а також компоненти `Link` та `useLocation` з `react-router-dom` для роботи з маршрутизацією. Також імпортується глобальний стиль `GlobalStyle`.

Всередині компонента `Homepage` оголошується стан `movies`, який ініціалізується порожнім масивом. Цей стан буде використовуватися для зберігання списку фільмів, отриманих з API. Хук `useLocation` зберігає інформацію про поточне місцезнаходження в маршрутизації.

Хук `useEffect` виконується при першому рендері компоненту. Всередині нього виконується асинхронний запит до API за допомогою функції `fetchHomepage`, яка повертає список популярних фільмів. Після отримання відповіді, дані зберігаються в стан `movies` за допомогою функції `setMovies`. Якщо виникає помилка під час запиту, вона ловиться та виводиться у консоль.

У рендер-методі компонента відображається заголовок "Trending today" та список фільмів. Для кожного фільму в масиві `movies` створюється елемент списку (`li`), в якому міститься посилання (`Link`) на сторінку з деталями фільму. Посилання містить стан `state` з поточним місцезнаходженням, що дозволяє зберігати попередній маршрут.

Також в компоненті присутній футер з авторським підписом та компонент `GlobalStyle`, який додає глобальні стилі до застосунку.

Загалом, цей компонент завантажує та відображає список популярних фільмів, використовуючи API TMDb, і надає можливість переходити до сторінок з деталями фільмів, зберігаючи попередній маршрут для навігації.

5.4. MoviesPage.js

JS MoviesPage.js ×

src > pages > JS MoviesPage.js > [e] MoviesPage

```
const MoviesPage = () => {
  const [searchedMovies, setSearchedMovies] = useState([]);
  const [searchParams, setSearchParams] = useSearchParams();
  const location = useLocation();
  const query = searchParams.get('query') ?? '';

  const searchMovies = event => {
    event.preventDefault();

    if (event.target.elements.searchMovie.value === '') {
      return setSearchParams({});
    } else {
      setSearchParams({ query: event.target.elements.searchMovie.value });
    }
  };

  useEffect(() => {
    try {
      fetchFindMovies(query).then(response => {
        setSearchedMovies(response.data.results);
      });
    } catch (error) {
      console.error(error);
    }
  }, [query]);

  return (
    <div>
      <form onSubmit={searchMovies}>
        <input type="text" name="searchMovie" placeholder="input movie" />
        <button type="submit">Search</button>
      </form>
      <ul>
        {searchedMovies.map(movie => (
          <li key={movie.id}>
            <Link to={`/movies/${movie.id}`} state={{ from: location }}>
              {movie.title}
            </Link>
          </li>
        ))}
      </ul>
    </div>
  );
};
export default MoviesPage;
```

```
`const MoviesPage = () => {
 const [searchedMovies, setSearchedMovies] = useState([]);
 const [searchParams, setSearchParams] = useSearchParams();
 const location = useLocation();
 const query = searchParams.get('query') ?? '';

 const searchMovies = event => {
  event.preventDefault();

  if (event.target.elements.searchMovie.value === '') {
   return setSearchParams({});
  } else {
   setSearchParams({ query: event.target.elements.searchMovie.value });
  }
 };

 useEffect(() => {
  try {
   fetchFindMovies(query).then(response => {
    setSearchedMovies(response.data.results);
   });
  } catch (error) {
   console.error(error);
  }
 }, [query]);

 return (
```

```
    <div>
     <form onSubmit={searchMovies}>
      <input type="text" name="searchMovie" placeholder="input movie" />
      <button type="submit">Search</button>
     </form>
     <ul>
      {searchedMovies.map(movie => (
       <li key={movie.id}>
        <Link to={`/movies/${movie.id}`} state={{ from: location }}>
         {movie.title}
        </Link>
       </li>
      ))}
     </ul>
    </div>
  );
};
export default MoviesPage;
`
```

Цей код представляє компонент MoviesPage у React, який дозволяє користувачам шукати фільми за допомогою API The Movie Database (TMDb) та відображає результати пошуку.

На початку коду імпортуються необхідні хуки useState, useEffect, useLocation, та useSearchParams з React та react-router-dom. Використовується функція fetchFindMovies для виконання запитів до API TMDb.

Всередині компонента MoviesPage оголошуються стан searchedMovies, який ініціалізується порожнім масивом, та searchParams для управління параметрами запиту в URL. Хук useLocation зберігає інформацію про поточне місцезнаходження, що використовується при навігації. Параметр query отримується з параметрів запиту URL або ініціалізується порожнім рядком, якщо він відсутній.

Функція searchMovies обробляє події відправки форми пошуку. При відправці форми, якщо поле вводу порожнє, параметри запиту очищуються. В іншому випадку, параметри запиту встановлюються відповідно до введеного значення в полі пошуку.

Хук useEffect виконується при зміні значення query. Всередині нього виконується асинхронний запит до API TMDb за допомогою функції fetchFindMovies, яка отримує список фільмів, що відповідають запиту. Результати запиту зберігаються в стан searchedMovies за допомогою функції setSearchedMovies. У випадку помилки, вона ловиться та виводиться у консоль.

У рендер-методі компонента відображається форма для введення пошукового запиту з полем вводу та кнопкою для відправки. Після форми відображається список фільмів, що відповідають пошуковому запиту. Для кожного фільму в масиві searchedMovies створюється елемент списку (li), в якому міститься посилання (Link) на сторінку з деталями фільму. Посилання містить стан state з поточним місцезнаходженням, що дозволяє зберігати попередній маршрут.

Таким чином, цей компонент забезпечує функціональність пошуку фільмів, відображення результатів пошуку та навігацію до сторінок з деталями фільмів, зберігаючи попередній маршрут для навігації.

6. Посилання на джерела

- https://www.youtube.com/user/TechGuyWeb
- https://www.youtube.com/c/Academind
- https://react.dev/
- https://devdocs.io/
- https://styled-components.com/docs
- https://babeljs.io/docs/
- https://mhnpd.github.io/react-loader-spinner/
- https://code.visualstudio.com/docs
- https://axios-http.com/docs/intro
- https://git-scm.com/doc
- https://github.com/spivet/parcel-doc/blob/master/doc/recipes.md
- https://docs.github.com/en/actions