Engenharia de Telecomunicações e Informática

Armazenamento de Dados em Ambientes Distribuídos

Pratical Assessment - Frontend

João Rabuge | 98509

Hugo Costa | 93910

Docente:

João Matos

Ano curricular: 4º

Semestre: 1º Semestre

2023/2024

Índice

Introdução

Neste relatório iremos explicar o código criado para a parte **FrontEnd** do projeto, tendo como base o BackEnd criado previamente, utilizando agora **React** e **Bootstrap** para a criação de um website mais apelativo.

Criámos componentes para serem utilizados nas págidas e estilizamos com classes CSS e classes Bootstrap predefinidas de modo que estas ficassem esteticamente mais apelativas e de mais **fácil utilização para o utilizador**. Apenas existe **2 fotos** de filmes utilizadas apenas porque considerámos que não era necessário colocar as fotos de todos os filmes para a avaliação deste projeto, por isso apenas o *Toy Story 3* terá a sua foto correta e todos os outros terão a foto do filme *Interstellar*.

As classes que vão ser explicadas no relatório são:

- **Componentes**
  - MovieCard.js
  - UserCard.js
- **Páginas**
  - Movie.js
  - Movies.js
  - User.js
  - Users.js
- CSS
  - App.css

1. Componentes

Agora iremos primeiro explorar os **componentes** que modificámos e criámos e por isso considerámos importantes para serem explicados:

- **MovieCard.js**
- **UserCard.js**

1.1. MovieCard.js

Este componente é uma parte do projeto de frontend, responsável por exibir informações de

um filme em um formato de cartão. O componente 'MovieCard' utiliza duas bibliotecas de componentes **Bootstrap React**, **Button** e **Card**, para criar um layout consistente e de alta qualidade.

1.1.1. Código

```
1  import Button from 'react-bootstrap/Button';
2  import Card from 'react-bootstrap/Card';
3
4  function MovieCard(props) {
5    return (
6      <Card style={{ width: '18rem'}} className="mb-3 shadow">
7        <Card.Body>
8          <Card.Title className="card-title"><strong>{props.title}</strong></Card.Title>
9          <div className='text-center'>
10           <Card.Img src={"/interstellar.jpg"} className='mb-4'/>
11         </div>
12         <Card.Text>
13           <strong>Genres: </strong>{props.genres.map((genre, index) => <span key={index}>{genre} </span>)}
14         </Card.Text>
15         <Card.Text>
16           <strong>Year: </strong>{props.year}
17         </Card.Text>
18       </Card.Body>
19       <Button href={"/movie/" + props._id} className='mb-4  btn-outline-primary btn-outline-primary:hover button-margin'>Open Movie</Button>
20   </Card>
21
22   );
23 }
24
25 export default MovieCard;
```

Figura 1. Classe MovieCard.js

1.1.2. Explicação do Código

As linhas iniciais do código envolvem a **importação** dos componentes **Button** e **Card** da biblioteca *react-bootstrap*. Esses componentes são essenciais para a estruturação e estilização do componente *MovieCard* e, portanto, representam uma parte crucial do código (linha 1).

O componente é definido como uma função denominada ***MovieCard*** que recebe ***props*** como argumento. Esta função representa a base para a **criação** de qualquer **componente React** e é, portanto, uma parte intrínseca do código (linha 2).

Dentro do componente, uma **imagem** é exibida por meio de uma **tag** *Card.Img* (linhas 7-9).

Embora a imagem seja fixa neste exemplo, em uma aplicação real, o caminho da imagem provavelmente seria fornecido como uma propriedade.

Um aspeto fundamental deste componente é a maneira como lida com listas. O código utiliza a função ***map()*** para iterar sobre o array de gêneros e exibir cada gênero (linhas 10-12). Esta técnica é fundamental para a exibição **eficiente** de arrays em React.

Além disso, o componente inclui um botão que permite ao usuário navegar para a página do filme específico. O URL para esta página é criado dinamicamente, concatenando ***/movie/*** com o **id do filme** (linha 16). Esta linha demonstra como criar URLs dinâmicos em React.
Por fim, o componente ***MovieCard*** é exportado para permitir que seja usado em outros lugares da aplicação (linha 20). Isto é essencial para a modularidade e **reutilização do código** em React.
Adicionalmente, tivemos o cuidado de alterar o MovieCard para todos os cartões da mesma fila terem o mesmo tamanho e o os restantes elementos estarem sempre alinhados com os outros cartões da fila, como por exemplo o título e o botão 'Open Movie'.
A cor do botão também foi alterada para roxo e foi adicionado um efeito de sombra a volta do cartão.

Em conclusão, o componente ***MovieCard*** exemplifica vários aspetos importantes do desenvolvimento com React, incluindo a importação e utilização de componentes externos, a manipulação de listas, a criação de URLs dinâmicos e a exportação de componentes para uso em outros lugares da aplicação. Ao entender esses aspetos, é possível compreender mais aprofundadamente as práticas recomendadas de desenvolvimento frontend em React.

1.2. UserCard.js

Este documento tem como objetivo fornecer uma análise do componente React chamada ***UserCard***. Este componente é responsável por exibir informações de um utilizador num formato de cartão. Utiliza componentes da biblioteca *react-bootstrap* para proporcionar uma **interface gráfica de utilizador** consistente e atraente.

1.2.1. Código

```
1  import Button from 'react-bootstrap/Button';
2  import Card from 'react-bootstrap/Card';
3
4  function UserCard(props) {
5    let image;
6    if(props.gender == "M"){
7      image ="/maleIcon.png";
8    }else{
9      image ="/femaleIcon.png";
10   }
11   return (
12     <Card style={{ width: '18rem'}} className="mb-3 shadow">
13
14       <Card.Body>
15         <Card.Title><strong>{props.name}</strong></Card.Title>
16         <div className='text-center m-4'>
17           <Card.Img src={image} className='userImage'/>
18         </div>
19         <Card.Text>
20           <strong>Gender: </strong>{props.gender}
21         </Card.Text>
22         <Card.Text>
23           <strong>Age: </strong>{props.age}
24         </Card.Text>
25         <Card.Text>
26           <strong>Occupation: </strong>{props.occupation.split}
27         </Card.Text>
28         <Card.Text>
29           <strong>Number of ratings: </strong>{props.num_ratings}
30         </Card.Text>
31       </Card.Body>
32       <Button href={"/user/" + props._id} className='mb-4  btn-outline-primary button-margin'>Open User</Button>
33     </Card>
34
35   );
36 }
37
38 export default UserCard;
```

Figura 2. Classe UserCard.js

1.2.2. Explicação Código

As primeiras linhas do código abordam a importação dos componentes **Button** e **Card** da biblioteca ***react-bootstrap*** (linha 1). Estes componentes são peças fundamentais para a construção do componente ***UserCard***.

A função *UserCard* aceita ***props*** como argumento, o que são os dados passados para este

componente por um componente "pai" (linha 3). Esta função é a base para a **criação** de qualquer componente **React**.

No início da função, uma variável ***image*** é declarada e atribuída um valor baseado no **género** do utilizador (linhas 4-8). Este exemplo demonstra como a lógica condicional pode ser usada para manipular a apresentação dos dados em componentes React.

Dentro do componente, um elemento ***Card.Img*** é usado para exibir a imagem do utilizador, cujo caminho é determinado pela variável *image* (linhas 10-12). Isto destaca como as imagens podem ser manipuladas **dinamicamente em React**.

O código em seguida exibe várias peças de informação sobre o utilizador, incluindo o seu **género**, **idade**, **ocupação** e **número de avaliações** (linhas 13-22). Estes dados são exibidos usando o componente ***Card.Text*** e são passados como propriedades para o componente ***UserCard***.

O componente também inclui um botão que, quando clicado, redireciona o utilizador para a **página de perfil** do utilizador específico (linha 23). A URL para esta página é criada **dinamicamente** usando a propriedade ***_id***. Isto é um exemplo de como as ações do utilizador podem ser manipuladas em React.

Por fim, o componente *UserCard* é exportado para ser usado em outros lugares da aplicação (linha 27). Isto é essencial para a modularidade e reutilização de código em React.

Em conclusão, o componente ***UserCard*** fornece assim um bom exemplo das boas e várias práticas no desenvolvimento com React. Isto inclui a utilização de lógica condicional, manipulação dinâmica de imagens, exibição de várias peças de informação, manipulação de ações do utilizador e exportação de componentes para uso em outros lugares da aplicação.

2. Páginas

Agora iremos primeiro explorar as **páginas** onde temos 4 classes que considerámos

importantes para serem explicadas, pois foram as classes que criámos ou modificámos:

- **Movie.js**
- **Movies.js**
- **User.js**
- **Users.js**

2.1. Movie.js

Este componente é encarregue de exibir detalhes de um filme e permite aos usuários fazer uma **transação blockchain**, desde que o usuário esteja **autenticado**.

2.1.1. Código

```
import React, {useState, useEffect} from "react";
import { useParams, useNavigate } from "react-router-dom";
import { openContractCall } from '@stacks/connect';
import {
  bufferCV,
} from '@stacks/transactions';
import { utf8ToBytes } from '@stacks/common';
import { userSession } from '../auth';
import Button from 'react-bootstrap/Button';
const bytes = utf8ToBytes('foo');
const bufCV = bufferCV(bytes);

export default function App() {
  let params = useParams();
  let [movie, setMovie] = useState([]);

  const submitMessage = async (e) => {
    e.preventDefault();

    const functionArgs = [
      bufCV
    ];

    const options = {
      contractAddress: '',
      contractName: '',
      functionName: 'set-value',
      functionArgs,
      appDetails: {
        name: 'Movies App Rating',
        icon: window.location.origin + '/my-app-logo.svg',
      },
      onFinish: data => {
        console.log('Stacks Transaction:', data.stacksTransaction);
        console.log('Transaction ID:', data.txId);
        console.log('Raw transaction:', data.txRaw);


        window.location.reload();
      },
    };

    const response = await openContractCall(options);

  };

  const getMovie = async () => {
    try {
      const response = await fetch(`http://localhost:3000/movies/${params.id}`, {
        method: 'GET',
        headers: {
        'Content-Type': 'application/json'
        },
      });

      const data = await response.json();
      setMovie(data);
      console.log(data);
    } catch (error) {
      console.error('Error:', error);
    }
  }

  useEffect(() => {
    getMovie();
  }, []);

  function renderStars(avgScore) {
    const filledStars = Math.round(avgScore);
    return '★'.repeat(filledStars) + '☆'.repeat(5 - filledStars);
  }

  return (
    <>
      {movie.length > 0 ?
        <div className="container pt-5 pb-5">
          <div className="d-flex align-items-center mb-5">
            <h1 classNAme="title me-3"><strong>{movie[0].title}</strong></h1>
            <div classNAme="star-rating pb-3">
              {renderStars(movie[0].avg_score)}
            </div>
          </div>
          <div classNAme="mb-5">
            <p><strong>Genres: </strong>{movie[0].genres.join(', ')}</p>
            <p><strong>Year: </strong>{movie[0].year}</p>
          </div>
          <div classNAme="text-center mb-5 min-height p-5 rounded-4 border shadow-sm bg-purple text-white">
            <h4 classNAme="mb-4"><strong>Comments</strong></h4>
            {movie[0].comments.length > 0 ? <p>{movie[0].comments.join('\n ')}</p> : <p> This movie doesn't have comments</p>}
          </div>
          <div classNAme="text-center">
                <Button variant='outline-primary' href="/movies" >Go Back</Button>
          </div>
          {userSession.isUserSignedIn() ? <a href="#" onClick={submitMessage}>Blockchain transaction</a> : null}
        </div>
        : null
      }
    </>
    )
}
```

Figura 3. Classe Movie.js

2.1.2. Explicação Código

O componente App faz uso de várias importações relevantes, incluindo recursos de bibliotecas externas como *react-router-dom*, *@stacks/connect*, *@stacks/transactions*, *@stacks/common* e *react-bootstrap* (linhas 1-7).

A função ***useState*** e ***useEffect*** do React são usadas para gerir o estado do componente e efeitos, respetivamente. ***useParams*** e ***useNavigate*** são **ganchos** do *react-router-dom* que permitem o acesso aos parâmetros da URL e a capacidade de navegar entre páginas (linha 1). O objeto ***userSession*** importado é provavelmente usado para gerenciar a sessão do usuário (linha 7).

O componente App declara uma função chamada ***submitMessage*** que faz uma chamada de contrato aberto usando a biblioteca *@stacks/connect*. A **transação** é configurada com os detalhes da aplicação e um manipulador de conclusão que regista os detalhes da **transação** (linhas 14-35).

Além disso, uma função ***getMovie*** é declarada para recuperar os detalhes do filme de um servidor backend (linhas 37-51). Esta função é chamada uma vez quando o componente é montado (linha 53), devido ao **gancho useEffect** do React.

A função ***renderStars*** é usada para criar uma representação visual do **rating** do filme (linhas 55-57).

Na **renderização** do componente, verifica-se se o filme foi recuperado e, em caso afirmativo, exibe os **detalhes do filme**.

Em conclusão, o componente ***App*** fornece assim um bom exemplo das boas e várias práticas no desenvolvimento com React. Este componente demonstra como usar o estado e os efeitos do React, como fazer **chamadas de API**, como **interagir com uma blockchain** e como **renderizar** condicionalmente elementos da interface do usuário. Ao entender esses aspetos, é possível compreender mais aprofundadamente as práticas recomendadas de desenvolvimento frontend em React.

2.2. Movies.js

Esta página é responsável por exibir uma **lista de 50 filmes** e **detalhes do último lançamento da** coleção movies da nossa base de dados MongoDB.

2.2.1. Código

```
1  import React, {useState, useEffect} from "react";
2  import CardGroup from 'react-bootstrap/CardGroup';
3  import Row from 'react-bootstrap/Row';
4  import Button from 'react-bootstrap/Button';
5
6  import MovieCard from "../components/MovieCard";
7
8  export default function App() {
9    let [movies, setMovies] = useState([]);
10
11   const getMovies = async () => {
12     try {
13       const response = await fetch('http://localhost:3000/movies', {
14         method: 'GET',
15         headers: {
16         'Content-Type': 'application/json'
17         },
18       });
19
20       const data = await response.json();
21       setMovies(data);
22
23     } catch (error) {
24       console.error('Error:', error);
25     }
26   }
27
28   useEffect(() => {
29     getMovies();
30   }, []);
31
32   return (
33     <div className="pb-5">
34       <div className="bg-purple d-flex justify-content-center p-5 shadow">
35         <div className="container bg-white row flex-lg-row-reverse align-items-center p-5 rounded-3 border shadow-lg">
36           <div className="col-10 col-sm-8 col-lg-6">
37             <img src="/toystory.jpg" className="d-block mx-lg-auto img-fluid shadow-lg rounded-3" width="400"  loading="lazy"/>
38           </div>
39           <div className="col-lg-6">
40             <h1 className="display-5 fw-bold m-5">Latest Release</h1>
41             <p className="lead m-5">Toy Story 3 is a 2010 American animated comedy-drama film produced by Pixar Animation Studios for Walt Disney Pictures.</p>
42             <div className=" m-5 d-grid gap-3 d-md-flex">
43               <Button className="btn btn-outline-section btn-lg" href={"/movie/" + 1}>Open Movie Details</Button>
44               <Button className="btn btn-outline-secondary btn-lg" href={"#movieDetails"}>See other Movies</Button>
45             </div>
46           </div>
47         </div>
48       </div>
49       <div className="container">
50         <h2 id="movieDetails" className="title mt-5 mb-5 text-center">Movies</h2>
51         <CardGroup>
52             <Row xs={1} md={2} className="d-flex justify-content-around">
53             {movies && movies.map((movie) => {
54                 return (
55                     <MovieCard
56                         key={movie._id}
57                         {...movie}
58                     />
59                 );
60             })}
61             </Row>
62         </CardGroup>
63         </div>
64     </div>
65   )
66 }
```

Figura 4. Classe Movies.js

2.2.2. Explicação Código

O componente *App'*começa com várias importações de pacotes relevantes, incluindo ***react***, ***react-bootstrap*** e o componente personalizado ***MovieCard*** (linhas 1-5).

A função ***useState*** e ***useEffect*** do React são usadas para gerir o estado do componente e efeitos, respetivamente (linhas 7 e 15). O estado ***movies*** é inicializado como um array vazio e é preenchido com dados recuperados de um servidor **backend** pela função assíncrona ***getMovies*** (linhas 9-18) que retorna 50 filmes da nossa coleção movies do MongoDB.

A função ***getMovies*** usa a **API Fetch** para fazer uma solicitação **GET** para um servidor backend e armazena a resposta no estado ***movies***. Se ocorrer um erro durante a solicitação, esse erro será registado na consola (linhas 9-18).

A função ***getMovies*** é chamada uma vez quando o componente é montado, graças ao **gancho** ***useEffect*** do React (linha 20).

Na **renderização** do componente, um grande banner é exibido no topo da página, com detalhes sobre o último lançamento (linhas 22-40). Abaixo do banner, uma **lista de filmes** é **renderizada** usando o componente ***MovieCard*** personalizado. Cada filme no estado ***movies*** é mapeado para um componente ***MovieCard***, demonstrando uma maneira eficaz de renderizar listas em React (linhas 42-51).

Em conclusão, o componente *App* fornece assim um bom exemplo das boas e várias práticas no desenvolvimento com React. Este componente demonstra como usar estado e efeitos em React, como fazer **chamadas de API**, e como renderizar listas de componentes. Ao entender esses aspetos, é possível compreender mais aprofundadamente as práticas recomendadas de desenvolvimento **frontend** em React, especialmente no contexto de um aplicativo que interage com um servidor **backend**.

2.3. User.js

Esta página é responsável por exibir detalhes de **um usuário e os seus cinco Top Movies**, bem como permitir apagar **o utilizador**.

2.3.1. Código

```
import React, { useState, useEffect } from "react";
import { useParams, useNavigate } from "react-router-dom";
import CardGroup from 'react-bootstrap/CardGroup';
import Row from 'react-bootstrap/Row';
import MovieCard from "../components/MovieCard";
import Button from 'react-bootstrap/Button';

export default function App() {
  let params = useParams();
  let navigate = useNavigate();
  let [user, setUser] = useState();
  let [movies, setMovies] = useState([]);

  const getUser = async () => {
    try {
      const response = await fetch(`http://localhost:3000/users/${params.id}`, {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json'
        },
      });

      const data = await response.json();
      setUser(data);
      console.log(data[0].topMovies);
      return data[0].topMovies; // Return topMovies for further processing
    } catch (error) {
      console.error('Error:', error);
    }
  }

  const deleteUser = async () => {
    try {
      await fetch(`http://localhost:3000/users/${params.id}`, {
        method: 'DELETE',
        headers: {
          'Content-Type': 'application/json'
        },
      });

      alert('User was successfully deleted');
      navigate('/'); // Redirect to the home page
    } catch (error) {
      console.error('Error:', error);
      alert('An error occurred while deleting the user.');
    }
  }

  useEffect(() => {
    getUser().then(movies => {
      if (movies && Array.isArray(movies)) {
        Promise.all(
          movies.map(movie =>
            fetch(`http://localhost:3000/movies/${movie.movieid}`)
              .then(res => res.json())
              .then(movieArray => movieArray[0]) // Extract the first element
          )
        ).then(fetchedMovies => {
            console.log("Movies", fetchedMovies);
            setMovies(fetchedMovies);
        }).catch(error => {
          console.error('Error fetching movie details:', error);
        });
      }
    });
  }, [params.id]);

  return (
    <>
      {user ?
        <div className="container pt-5 pb-5">
            <h1 className="title">{user[0].name}</h1>
            <div className="top-margin">
              <p><strong>Gender: </strong> {user[0].gender}</p>
              <p><strong>Age: </strong> {user[0].age}</p>
              <p><strong>Occupation: </strong> {user[0].occupation}</p>
              <p><strong>Number of Ratings: </strong> {user[0].num_ratings}</p>
            </div>
            <div className="pt-5">
              <h4 className="text-center"><strong>TOP 5 MOVIES</strong></h4>
              <div className="container pt-5 pb-5">
                <CardGroup>
                  <Row xs={1} md={2} className="d-flex justify-content-around">
                    {movies.map((movie) => {
                      return (
                        <MovieCard
                          key={movie._id}
                          {...movie}
                        />
                      );
                    })}
                  </Row>
                </CardGroup>
              </div>
              <div className="text-center">
                <Button variant="outline-secondary m-2" onClick={deleteUser}>Delete User</Button>
                <Button variant="outline-primary" href="/users" >Go Back</Button>
              </div>
            </div>
        </div>
        : null
      }
    </>
  )
}
```

Figura 5. Classe User.js

2.3.2. Explicação do Código

O componente ***App*** começa com várias importações de pacotes relevantes, incluindo ***react***, ***react-router-dom***, ***react-bootstrap*** e um componente personalizado chamado ***MovieCard*** (linhas 1-6).

A função ***useState*** e ***useEffect*** do React são usadas para gerir o estado do componente e efeitos colaterais, respetivamente. ***useParams*** e ***useNavigate*** são **ganchos** do *react-router-dom* que permitem o acesso aos parâmetros da URL e a capacidade de navegar programaticamente (linha 1).

O componente utiliza duas funções assíncronas, ***getUser*** e ***deleteUser***. ***getUser*** faz uma **chamada de API** para buscar os **detalhes do usuário e os cinco principais filmes** desse utilizador (linhas 14-27). ***deleteUser*** faz uma **chamada de API** para **apagar o utilizador** e, em seguida, **redireciona o utilizador para a página inicial** (linhas 29-41).

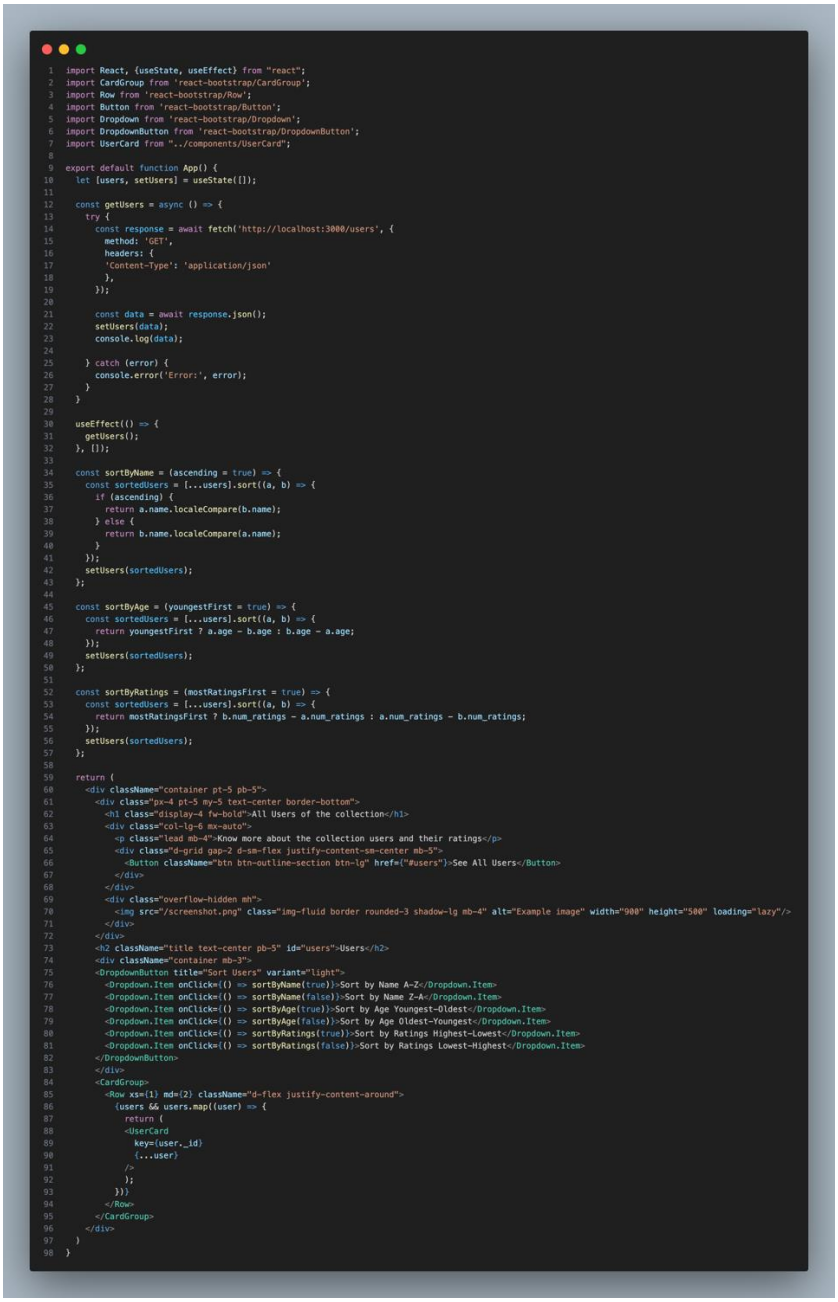
A função ***getUser*** é chamada uma vez quando o componente é montado, e os filmes são pesquisados com base na resposta de ***getUser*** (linhas 43-60).

Na **renderização** do componente, verifica-se se o **utilizador foi recuperado** e, se sim, exibe os **detalhes do utilzador e seus Top Movies**. Um botão para apagar o utilziador também foi implementado (linhas 62-100).

2.4. Users.js

Esta página é responsável por exibir uma **lista de 50 utilizadores** da coleção users da nossa base de dados MongoDB.

2.4.1.Código



```
import React, {useState, useEffect} from "react";
import CardGroup from 'react-bootstrap/CardGroup';
import Row from 'react-bootstrap/Row';
import Button from 'react-bootstrap/Button';
import Dropdown from 'react-bootstrap/Dropdown';
import DropdownButton from 'react-bootstrap/DropdownButton';
import UserCard from "../components/UserCard";

export default function App() {
  let [users, setUsers] = useState([]);

  const getUsers = async () => {
    try {
      const response = await fetch('http://localhost:3000/users', {
        method: 'GET',
        headers: {
        'Content-Type': 'application/json'
        },
      });

      const data = await response.json();
      setUsers(data);
      console.log(data);

    } catch (error) {
      console.error('Error:', error);
    }
  }

  useEffect(() => {
    getUsers();
  }, []);

  const sortByName = (ascending = true) => {
    const sortedUsers = [...users].sort((a, b) => {
      if (ascending) {
        return a.name.localeCompare(b.name);
      } else {
        return b.name.localeCompare(a.name);
      }
    });
    setUsers(sortedUsers);
  };

  const sortByAge = (youngestFirst = true) => {
    const sortedUsers = [...users].sort((a, b) => {
      return youngestFirst ? a.age - b.age : b.age - a.age;
    });
    setUsers(sortedUsers);
  };

  const sortByRatings = (mostRatingsFirst = true) => {
    const sortedUsers = [...users].sort((a, b) => {
      return mostRatingsFirst ? b.num_ratings - a.num_ratings : a.num_ratings - b.num_ratings;
    });
    setUsers(sortedUsers);
  };

  return (
    <div className="container pt-5 pb-5">
      <div class="px-4 pt-5 my-5 text-center border-bottom">
        <h1 class="display-4 fw-bold">All Users of the collection</h1>
        <div class="col-lg-6 mx-auto">
          <p class="lead mb-4">Know more about the collection users and their ratings</p>
          <div class="d-grid gap-2 d-sm-flex justify-content-sm-center mb-5">
            <Button className="btn btn-outline-section btn-lg" href={"#users"}>See All Users</Button>
          </div>
        </div>
        <div class="overflow-hidden mh">
          <img src="/screenshot.png" class="img-fluid border rounded-3 shadow-lg mb-4" alt="Example image" width="900" height="500" loading="lazy"/>
        </div>
      </div>
      <h2 className="title text-center pb-5" id="users">Users</h2>
      <div className="container mb-3">
      <DropdownButton title="Sort Users" variant="light">
        <Dropdown.Item onClick={() => sortByName(true)}>Sort by Name A-Z</Dropdown.Item>
        <Dropdown.Item onClick={() => sortByName(false)}>Sort by Name Z-A</Dropdown.Item>
        <Dropdown.Item onClick={() => sortByAge(true)}>Sort by Age Youngest-Oldest</Dropdown.Item>
        <Dropdown.Item onClick={() => sortByAge(false)}>Sort by Age Oldest-Youngest</Dropdown.Item>
        <Dropdown.Item onClick={() => sortByRatings(true)}>Sort by Ratings Highest-Lowest</Dropdown.Item>
        <Dropdown.Item onClick={() => sortByRatings(false)}>Sort by Ratings Lowest-Highest</Dropdown.Item>
      </DropdownButton>
      </div>
      <CardGroup>
        <Row xs={1} md={2} className="d-flex justify-content-around">
          {users && users.map((user) => {
            return (
            <UserCard
              key={user._id}
              {...user}
            />
            );
          })}
        </Row>
      </CardGroup>
    </div>
  )
}
```

Figura 6. Classe Users.js

2.4.2.Explicação do Código

O componente ***App'***começa com várias importações de pacotes relevantes, incluindo ***react***, ***react-bootstrap*** e um componente personalizado chamado ***UserCard*** (linhas 1-5).

A função ***useState*** e ***useEffect*** do React são usadas para gerir o estado do componente e

efeitos, respetivamente. O estado ***users*** é inicializado como um array vazio e é preenchido com dados recuperados de um servidor **backend** pela função assíncrona ***getUsers*** (linhas 7-19).

A função ***getUsers*** usa a **API Fetch** para fazer uma solicitação GET para um servidor backend e armazena a resposta no estado **users**. Se ocorrer um erro durante a solicitação, esse erro será registado na consola (linhas 9-19).

São declaradas três funções de ordenação - ***sortByName***, ***sortByAge*** e ***sortByRatings***, que ordenam os utilizadores com base no seu **nome**, **idade** e **número de avaliações**, respetivamente (linhas 23-39). Essas funções são acionadas quando o **utilziador seleciona uma opção no menu**.

Na **renderização** do componente, uma lista de **utilizadores é renderizada** usando o componente ***UserCard*** personalizado. Cada utilizador no estado ***users*** é mapeado para um componente ***UserCard***. Além disso, um drop down menu foi implementado para permitir que o **utilizador ordene a lista de utilizadores com base em diferentes critérios** (linhas 41-71).

3. App.css

Este documento analisa uma folha de estilos **CSS** que é usada no frontend do projeto. A folha de estilos define várias **classes** que são usadas para estilizar componentes e **elementos na aplicação**.

3.1. Código

```
.jumbotro {
  background-color: #e9ecef;
}

.title {
  margin-bottom: 20px;
}

.button-margin{
  margin-left: 20px;
  margin-right: 20px;
}

.star-rating {
  font-size: 24px;
  color: gold;
}

.min-height{
  min-height: 50vh;
}

.btn-outline-primary {
  color: rgb(127, 62, 231) !important;
  background-color: white !important;
  border-color: rgb(127, 62, 231) !important;
  font-weight: 500 !important;
  border-radius: 200px !important;
  border: 2px solid rgb(127, 62, 231) !important ;

}

.btn-outline-primary:hover {
  color: white !important;
  background-color: rgb(127, 62, 231) !important;
}

.btn-outline-secondary {
  color: rgb(77, 77, 77) !important;
  background-color: white !important;
  border-color: rgb(77, 77, 77) !important;
  font-weight: 500 !important;
  border-radius: 200px !important;
  border: 2px !important ;
}

.btn-outline-secondary:hover {
  color: white !important;
  background-color: rgb(77, 77, 77) !important;
  border-color: rgb(77, 77, 77) !important;
  font-weight: 500 !important;
  border-radius: 200px !important;
  border: 2px !important ;
}

.btn-outline-section {
  color: white !important;
  background-color: rgb(127, 62, 231) !important;
  border-color: rgb(127, 62, 231) !important;
  font-weight: 500 !important;
  border-radius: 200px !important;
  border: 2px  !important ;

}
.btn-outline-section:hover {
  color: white !important;
  background-color: rgb(87, 5, 219) !important;
  border-color: rgb(87, 5, 219) !important;
  font-weight: 500 !important;
  border-radius: 200px !important;
  border: 2px  !important ;
}

.bg-purple{
  background-color: rgb(24, 8, 49);
}

.mh{
  max-height: 50vh;
}

.userImage{
  max-width: 90px;
  border-radius: 50% !important;
  background-color: rgb(87, 5, 219);
  padding: 3%;
}

.card-title {
  min-height: 50px;
}
```

Figura 7. Classe App.css

3.2. Explicação do Código

Apenas vamos explicar as classes que nós desenvolvemos.

A classe ***.title*** é usada para adicionar uma margem inferior aos títulos (linha 5-7).

A classe ***.button-margin*** adiciona margens esquerda e direita aos botões (linha 9-11).

A classe ***.star-rating*** define a cor e o tamanho das estrelas de rating (linha 13-15).

A classe ***.min-height*** define a altura mínima de um elemento para 50vh (linha 17-19).

A classe ***.btn-outline-primary*** e ***.btn-outline-primary:hover*** definem a estilização para um botão primário e seu estado de hover, respetivamente. A cor do texto, a cor de fundo, a cor da borda, o tamanho da fonte, o raio da borda e a largura da borda são definidos (linha 21-33).

De maneira semelhante, a classe ***.btn-outline-secondary*** e ***.btn-outline-secondary:hover*** definem a estilização para um botão secundário e seu estado de hover, respetivamente (linha 35-49).

A classe ***.btn-outline-section*** e ***.btn-outline-section:hover*** definem a estilização para um botão de seção e o seu estado de hover, respetivamente (linha 51-63).

A classe ***.bg-purple*** é usada para definir a cor de fundo de um elemento para um tom de roxo (linha 65-67).

A classe ***.mh*** define a altura máxima de um elemento para 50vh (linha 69-71).

A classe ***.userImage*** define a estilização para a imagem do usuário. A largura máxima, o raio da borda, a cor de fundo e o preenchimento são definidos (linha 73-77).

A classe ***.card-title*** define a altura mínima de um título de cartão para 50px (linha 79-81).

Em conclusão. a folha de estilo contém várias **classes CSS** que ajudam a estilizar os componentes e elementos na aplicação. Cada classe serve a um propósito específico e contribui para a aparência do projeto.

4. Conclusão

Este projeto de Engenharia de Telecomunicações e Informática focado no armazenamento de dados em ambientes distribuídos permitiu a criação de um **website apelativo** através do uso de **React** e **Bootstrap**. As classes de componentes e as páginas HTML foram cuidadosamente construídas e estilizadas, proporcionando uma interface de utilizador fácil de utilizar.

Foi dada especial atenção à **estruturação** e **explicação do código**, permitindo uma compreensão clara do seu funcionamento. Através de componentes como **MovieCard.js** e **UserCard.js**, demonstrou-se a versatilidade do React e a importância da modularidade e reutilização do código.

O projeto também demonstrou a eficácia da aplicação de técnicas de desenvolvimento frontend, como a manipulação de listas, criação de URLs dinâmicos, lógica condicional e **chamadas de API**. Através da análise de páginas como **Movie.js, Movies.js, User.js** e **Users.js**, foi possível entender a interação com um servidor backend e a manipulação da navegação do utilizador.

A folha de estilo CSS desempenhou um papel crucial na estilização dos componentes e elementos da aplicação, contribuindo para a **aparência geral e a experiência do utilizador**.

Em conclusão, este projeto de armazenamento de dados em ambientes distribuídos demonstrou a **eficácia do uso de React e Bootstrap** no **desenvolvimento de um website funcional e apelativo**. As técnicas e práticas utilizadas neste projeto oferecem um modelo valioso para o desenvolvimento frontend em React.