Desenvolvimento para a Internet e Aplicações Móveis

2023/24

Desenvolvimento e integração de frontend REACT e backend Django

Índice

1. Introdução

Django é uma biblioteca Python utilizada para o desenvolvimento de um website num servidor, isto é, no *backend*. No entanto, para que o website fique também adaptado à sua apresentação em dispositivos móveis, são necessárias tecnologias complementares de *frontend*, de execução rápida nos clientes. Esta necessidade conduziu a arquiteturas mistas, em que o *backend* e o *frontend* estão separados e respeitam um protocolo de comunicação. Atualmente, as melhores soluções para o desenvolvimento de *frontend* são as *single-page applications* (SPAs). Estas aplicações comunicam com o *backend* através de pontos de contacto HTTP, por onde recebem e transmitem dados no formato JSON. React é uma biblioteca JavaScript, versão ES6, frequentemente utilizada na programação de SPAs para frontend. React recorre a HTML, CSS e JavaScript. A versão para dispositivos móveis, chamada React Native, usa as suas próprias APIs e componentes gráficos. Outras bibliotecas baseadas em JavaScript e também utilizadas para o desenvolvimento de SPAs para dispositivos móveis são Ember, Angular, BackBone e Vue.

Este texto apresenta os conceitos fundamentais de React baseado em HTML, CSS e JavaScript e a sua ligação ao Django.

2. Conceitos Utilizados

1.1 SPA - Single Page Application

Páginas que funcionam por substituição de componentes do lado do cliente (usualmente em resposta a acções do utilizador) e não por refrescamento total da página. Este tipo de soluções emergiu na linha da passagem de parte do peso computacional para o cliente de modo a tornar as páginas mais rápidas (responsive). A performance é influenciada pela existência de uma única página HTML, atualizada dinamicamente com dados originários do servidor.

1.2 CRUD - Create, Read, Update and Delete

Indica as quatro operações de acesso e de alteração de dados persistentes, isto e, de dados que se mantém entre várias execuções de um programa. É usado este termo para interfaces que fornecem as operações de criação, leitura, atualização e eliminação. As APIs REST invocam os quatro tipos de operações CRUD.

1.3 REST - Representational State Transfer Architectural Style

Uma API REST ou RESTful (*representational state transfer architectural style*) é um método de acesso a "serviços" de uma aplicação externa (API) que respeita os seguintes princípios:

- Interface uniforme - O interface é o sempre o mesmo independentemente da fonte do pedido e da linguagem usada pela fonte;
- Independência (desacoplamento) entre cliente e servidor - as aplicações cliente e servido são totalmente independentes;
- Ausência de estado - O serviço de um pedido não se prolonga por várias chamadas, tem que ser totalmente executável numa chamada da API (para evitar a necessidade de criação do conceito de sessão a este nível);
- Possibilidade de usar cache do lado do cliente - Deve ser possível manter respostas do lado do cliente para evitar pedidos repetidos;
- Assumir arquitetura por camadas - nem cliente, nem servidor podem assumir que comunicam diretamente um com o outro, mas que pode haver vários intermediários;

- Passagem de código (opcional) - A informação passada entre cliente e servidor pode ser composta por dados ou código a executar (neste caso o código só deverá executar a pedido do utilizador).

Para permitir o desacoplamento, normalmente as mensagens trocadas têm formatos independentes da linguagem. JavaScript Object Notation (JSON) e eXtensible Markup Language ou (XML) são os formatos mais usados.

1.4 CORS - Cross-Origin Resource Sharing

É um mecanismo que permite a partilha de recursos de origens diferentes do domínio do pedido original, por exemplo imagens de uma página externa. CORS define a forma de um browser interagir com um servidor para determinar se é seguro permitir o pedido de origem cruzada. Por exemplo, um pedido CORS existe quando um servidor https://domain-a.com faz um pedido para obter dados de outro servidor https://domain-b.com/data.json através de um script existente no seu código. Por razões de segurança os browsers restringem pedidos CORS originados em scripts.

3. Instalação do Software de Base

Para além do Python 3, do Django e do editor PyCharm, deve ter instalado no seu computador:

- A biblioteca Node.js que gere eventos JavaScript de forma concorrente;
- A ferramenta npm, que permite instalar e gerir versões de bibliotecas JavaScript.

Outras bibliotecas, como Bootstrap para apresentação gráfica, Reactstrap para a programação de componentes React e Axios para gerir os pedidos HTTP, serão instaladas na altura do desenvolvimento do frontend.

4. App React

4.1 Criação

Na linha de comando do seu sistema operativo:

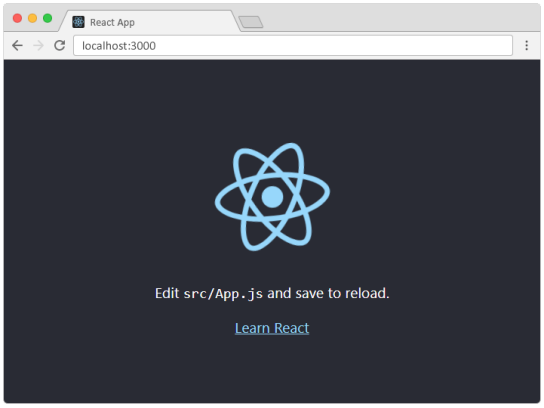
```
npx create-react-app my-react-app
```

4.2 Execução

Entre na diretoria my-react-app e execute a app:

```
cd my-react-app
npm start
```

Deverá obter o seguinte resultado no seu browser. A app React corre na porta 3000 do localhost.



Faça CTRL-C para interromper o servidor React.

4.3 Modificação

Entre na subdiretoria src e edite o ficheiro App.js . Substitua o código que encontrar pelo seguinte:

```
function App() {
  return (
    <div className="App">
      <h1>Os alunos de DIAM são os maiores!</h1>
    </div>
  );
}
```

export default App;

Execute o servidor e verifique o resultado no browser.

Agora apague todos os ficheiros na diretoria src e crie nesta diretoria o ficheiro index.js com o código seguinte:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const myFirstElement = <h1>Cá está o React de novo!</h1>

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myFirstElement);
```

Execute o servidor de novo e verifique o resultado no browser. Criou o seu primeiro elemento React e fez render deste elemento no document, isto é na sua página gerada por JavaScript. O código será explicado em detalhe no ponto seguinte.

5. Renderização de HTML

5.1 createRoot()

Função que define em que elemento HTML onde o componente React será renderizado.

5.2 render()

Função que define o componente React a renderizar. O resultado da renderização é um ficheiro HTML gravado na diretoria public da app.

Edite o ficheiro index.html e verifique que o body tem uma única div:

```
<div id="root"></div>
```

5.3 JSX

O código JSX (JavaScript XML) permite escrever HTML dentro de código JavaScript em index.js.

JSX avalia as expressões dentro de { }. Exemplo:

```
const myElement = <h1>React é {999 + 1} vezes melhor com JSX</h1>;
```

Um componente deve ter apenas um elemento HTML. Se quiser colocar vários elementos HTML, coloque-os dentro de um div:

```
const myElement = (
  <div>
    <p>I am a paragraph.</p>
    <p>I am a paragraph too.</p>
  </div>
);
```

O atributo class de HTML em JSX é denominado className.

```
const myElement = <h1 className="myclass">Cá está o React!</h1>;
```

JSX não admite condições if. Alternativa: programar o if fora do JSX.

```
const x = 5;
let text = "Até breve";
if (x < 10) {
  text = "Olá";
}

const myelement = <h1>{text}</h1>;
```

Teste e verifique o resultado no browser.

Exemplo completo de componente definido com JSX:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const myelement = (
```

```
  <table>
    <tr>
      <th>Nome</th>
    </tr>
    <tr>
      <td>Joana</td>
    </tr>
    <tr>
      <td>João</td>
    </tr>
  </table>
);

const container = document.getElementById('root');
const root = ReactDOM.createRoot(container);
root.render(myelement);
```

Verifique o resultado no browser.

5.4 *root*

O nó root é o elemento HTML onde é mostrado o resultado do componente React. Por convenção tem id=’root’, mas não é obrigatório.

6. Componentes

Os componentes React retornam código HTML.

6.1 Componentes definidos por uma classe

O nome de um componente deve começar com uma maiúscula.

A classe herda de `React.Component` e deve implementar a função `render()`. O construtor não é obrigatório, mas se existe é invocado na inicialização.

O objeto `state` permite guardar propriedades de um componente. Na função `setState` altera o conteúdo do `state`.

Uma instância da classe é renderizada na forma de uma tag HTML.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

class Bilhete extends React.Component {
constructor() {
    super();
    this.state = {preco: "5"};
  }
  render() {
    return <h2>Esta é uma entrada para a festa de finalistas!</h2>;
  }
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Bilhete />);
```

Verifique o resultado no browser.

6.1.1 Ciclo de vida: Mounting

1. Inicialmente é chamado o construtor de um componente.

2. Seguidamente é chamada a função `getDerivedStateFromProps(props, state)`, se existir. É o local para definir o `state` baseado nas `props`. Retorna o `state` atualizado. Ver https://legacy.reactjs.org/docs/react-component.html#static-getderivedstatefromprops

3. Em seguida é chamada a função `render()`, que envia o código HTML para o DOM.

4. Finalmente, depois da renderização do componente, é chamada a função `componentDidMount()`, que eventualmente executa ações após ser feita a renderização. Ver https://legacy.reactjs.org/docs/react-component.html#static-getderivedstatefromprops

6.1.2 Ciclo de vida: Updating

Um componente é atualizado quando há alterações em `state` ou em `props`. Os métodos invocados são as seguintes:

1. `getDerivedStateFromProps()` – neste método deve ser estabelecido o `state` baseado nas `props` iniciais. Ver https://legacy.reactjs.org/docs/react-component.html#static-getderivedstatefromprops

2. `shouldComponentUpdate()` – retorna um booleano que indica se continua com a renderização. O valor por defeito é true, isto é se o método não for definido a renderização continua. Ver https://legacy.reactjs.org/docs/react-component.html#shouldcomponentupdate

3. `render()` – este é o único método obrigatório, já referido neste texto.

4. `getSnapshotBeforeUpdate()` – permite aceder aos `props` e `state` antes da atualização. A existência deste método implica a existência do método seguinte. Ver https://legacy.reactjs.org/docs/react-component.html#static-getderivedstatefromprops

5. `componentDidUpdate()` – é chamado depois da atualização do componente no DOM. Ver https://legacy.reactjs.org/docs/react-component.html#static-getderivedstatefromprops

6.1.3 Ciclo de vida: Unmounting

Esta fase refere-se à remoção do componente do DOM. Tem um único método, componentWillUnmount(), que é chamado quando o componente é removido.

6.2 Componentes definidos por uma função

O mesmo resultado de uma classe pode ser obtido com uma função.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Bilhete() {
  return <h2>Esta é uma entrada para a festa de finalistas! - desta vez com
função</h2>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Bilhete />);
```

Verifique o resultado no browser.

6.3 Props em componentes

Props são as propriedades dos componentes. São passadas aos componentes através dos atributos HTML.

Props numa classe:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

class Bilhete extends React.Component {
constructor(props) {
    super(props);
  }
  render() {
    return <h2>Esta é uma {this.props.model}!</h2>;
  }
}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Bilhete model="entrada para a festa de finalistas"/>);
```

Props numa função:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Bilhete(props) {
  return <h2>Esta é uma entrada para a festa de finalistas ao preço de
{props.preco}</h2>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Bilhete preco="10€"/>);
```

6.4 Invocação de componentes em vários ficheiros

Os componentes podem ser definidos num ficheiro, em que são exportados, por exemplo com

```
export default Bilhete;
```

e depois importados noutro ficheiro, por exemplo com

```
import Bilhete from './Bilhete.js';
```

7. Eventos

7.1 Adição de eventos

React considera os mesmos eventos do HTML, isto é os que são originados pelo rato. Os eventos em React são identificados por nomes ligeiramente diferentes ao seu nome no HTML. Por exemplo, `onClick` em vez de `onclick` e `onClick={shoot}` em vez de `onclick="shoot()"`.

A passagem de argumentos para a gestão de um evento utilizam funções no seu formato compacto "arrow".
(Ver https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions)

Finalmente, o objeto que representa o evento pode ser acedido na função de gestão. Neste exemplo, o argumento "b" da função representa o evento lançado pelo click no rato:

```
function Jogo() {
  const jogada = (a, b) => {
    alert(b.type);
  }

  return (
    <button onClick={(event) => jogada("golo!", event)}>Remata aqui</button>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Jogo />);
```

8. Condições

8.1 Condição if

```
function Golo(props) {
  const eGolo = props.eGolo;
  if (eGolo) {
    return <FezGolo/>;
  }
  return <FalhouGolo/>;
}
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Golo eGolo={false} />);
```

8.2 Condição &&

Exemplo de utilização de &&: se `veiculos.length > 0`, então mostra o `<h2>` que vem a seguir.

```
function Garagem(props) {
  const veiculos = props.veiculos;
  return (
    <>
      <h1>Garagem</h1>
      {veiculos.length > 0 &&
        <h2>
          Existem {veiculos.length} veiculos na garagem.
        </h2>
      }
    </>
  );
}

const veiculos = ['Ford', 'BMW', 'Audi'];
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Garagem veiculos={veiculos} />);
```

9. Listas

9.1 Renderização de listas com map()

A função `map()` é útil para mostrar o conteúdo de uma lista JavaScript no HTML. Para a função `map()` veja https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Exemplo com map() e keys:

```
function Veiculo(props) {
  return <li>Eu sou um { props.marca }</li>;
}

function Garagem() {
  const veiculos = [
    {id: 1, marca: 'Ford'},
    {id: 2, marca: 'BMW'},
```

```
    {id: 3, marca: 'Audi'}
  ];
  return (
    <>
      <h1>Que veículos existem na garagem?</h1>
      <ul>
        {veiculos.map((veiculo) => < Veiculo key={veiculo.id}
                                            marca={veiculo.marca} />)}
      </ul>
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Garagem />);
```

10. Formulários

10.1 Abertura de um formulário

```
function UmForm() {
  return (
    <form>
      <label>Nome:
        <input type="text" />
      </label>
    </form>
  )
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<UmForm />);
```

10.2 Tratamento dos dados de um formulário

Os dados de um formulário são guardados no state de um componente. Exemplo:

```
import { useState } from 'react';
import ReactDOM from 'react-dom/client';

function UmForm() {
  const [nome, setNome] = useState("");

  return (
    <form>
      <label>Nome:
```

```
        <input
          type="text"
          value={nome}
          onChange={(e) => setNome(e.target.value)}
        />
      </label>
    </form>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<UmForm />);
```

10.3 Submissão de um formulário

Submissão com recurso ao atributo onSubmit.

Exemplo com um campo de input do tipo text:

```
import { useState } from 'react';
import ReactDOM from 'react-dom/client';

function UmForm() {
  const [nome, setNome] = useState("");

  const handleSubmit = (event) => {
    // impede o browser de atualizar o form
    event.preventDefault();
    //abre um popup com o valor no campo nome que está no state
    alert('Nome introduzido: ${nome}')
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>Nome:
        <input
          type="text"
          value={nome}
          // quando o campo é atualizado guarda o valor
          // no campo nome que está no state
          onChange={(e) => setNome(e.target.value)}
        />
      </label>
      <input type="submit" />
    </form>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(<UmForm />);
```

Para exemplos com vários campos de input do tipo text, ou com um campo de input do tipo textarea ou select, veja em https://www.w3schools.com/REACT/react_forms.asp

11. Ligação de React a Django – exemplo de Login

11.1 Exemplo do lado de uma app Django

Prepare a sua app com um *Endpoint Django REST API*.

Em primeiro lugar instale as bibliotecas djangorestframework, django-cors-headers e django-rest-framework no ambiente virtual.

Seguidamente prepare o settings.py:

```
INSTALLED_APPS = [
    ...
    'rest_framework',
    'corsheaders',
    'rest_framework.authtoken',
    ...
]

MIDDLEWARE = [
    ...
    'corsheaders.middleware.CorsMiddleware',
    ...
]

CORS_ALLOW_ALL_ORIGINS = True
```

Corra as migrações para atualizar a base de dados com as novas tabelas.

Crie uma view em views.py. Neste exemplo recorremos à APIView de rest_framework e implementámos a classe LoginView. O seu método post recebe o username e a password e retorna um token, em caso de sucesso, ou uma mensagem de insucesso.

```
from django.shortcuts import get_object_or_404, render
from django.contrib.auth import authenticate
from django.http import JsonResponse
from rest_framework.views import APIView
from rest_framework.authtoken.models import Token
```

```
from django.contrib.auth.models import User

class LoginView(APIView):
    def post(self, request):
        username = request.data.get('username')
        password = request.data.get('password')

        user = authenticate(username=username, password=password)
        if user:
            token, _ = Token.objects.get_or_create(user=user)
            return JsonResponse({'token': token.key})
        else:
            return JsonResponse({'error': 'Credenciais inválidas'}, status=400)
```

Finalmente, crie um acesso a LoginView a partir do urls.py:

```
from django.urls import path
from .views import LoginView
from . import views

app_name = 'djreapp'
urlpatterns = [
    (…)
    path('login/', LoginView.as_view(), name='login'),
]
```

O seu endpoint está pronto no Django.

11.2 Exemplo do lado de React

Do lado de React programe o ficheiro index.js com o seguinte código:

```
import React, { useState } from 'react';
import ReactDOM from 'react-dom/client';

const Login = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [msg, setMsg] = useState('');
  const [error, setError] = useState('');

  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      const response = await fetch('http://localhost:8000/djreapp/login/', {
        method: 'POST',
```

```
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ username, password }),
      });

      const data = await response.json();
      if (response.ok) {
        // Fez Login com sucesso
        // Guarda o token para invocações futuras durante a sessão
        localStorage.setItem('token', data.token);
        // Imprime msg de sucesso ou redireciona
          setMsg('Viva, fez login !');
      } else {
        setError(data.error || 'Algo errado');
      }
    } catch (error) {
      console.error('Login falhou:', error);
      setError('Algo errado');
    }
  };

  return (
    <div>
      <h2>Login</h2>
      {error && <p>{error}</p>}
      {<p>{msg}</p>}
      <form onSubmit={handleLogin}>
        <input
          type="text"
          placeholder="Username"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
        />
        <input
          type="password"
          placeholder="Password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
        <button type="submit">Login</button>
      </form>
    </div>
  );
};

export default Login;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Login />);
```

Repare que a função Login contacta o endpoint do Django e envia-lhe o username e a password. Em resposta obtém um token e está logado, ou recebe uma mensagem de erro.

12. Exercícios

12.1 Exercício sobre state

props e state guardam atributos de um componente. No caso de state estes atributos podem ser alterados devido a ações do utilizador.

Programe um componente denominado Contador, na forma de uma função, que inicializa um contador a 0 e depois o incrementa no state do componente a cada vez que é dado um click num botão. O valor do contador deve ser apresentado na página. Teste o componente Contador num projeto React. O seu código React deve ser gravado no ficheiro contador.js.

12.2 Exercício sobre condição &&

Crie um componente React que, quando é feito click num botão, um texto é escondido e uma imagem é mostrada. Quando é de novo feito click no botão, o texto é mostrado e a imagem é escondida. O seu código React deve ser gravado no ficheiro mostraeesconde.js.

12.3 Exercício sobre eventos

Crie uma página React com um campo de input de texto e um botão. O utilizador introduz um texto no campo de input. Quando é feito um click no botão é lançado um popup com o texto introduzido pelo utilizador.

12.4 Exercício sobre ligação de React a Django

Considere o seu projeto Django da Votação. Programe um endpoint para login a partir do React. Crie um projeto React e programe uma página para contacto com o endpoint e para fazer login.

13. Bibliografia

"React Top-Level API", https://legacy.reactjs.org/docs/react-api.html . Acedido em Acedido em 16-04-2024.

"React Tutorial", https://www.w3schools.com/REACT/default.asp . Acedido em 16-04-2024.

Purohit, Rakesh: "A Comprehensive Guide to React Practical Exercises and Coding Challenges", https://www.dhiwise.com/post/a-comprehensive-guide-to-react-practical-exercises-and-coding-challenges . Acedido em 18-04-2024.

Souza, "Using React with Django to create an app: Tutorial" https://blog.logrocket.com/using-react-django-create-app-tutorial/ . Acedido em 24-04-2023.

Singhal, "How to create a REST API with Django REST framework" https://blog.logrocket.com/django-rest-framework-create-api/ . Acedido em 24-04-2023.

Parker, "How to NPM Start for React Tutorial Project" https://www.pluralsight.com/guides/npm-start-for-react-tutorial-project . Acedido em 24-04-2023.

Basques and Emelianova, "Simulate mobile devices with Device Mode" https://developer.chrome.com/docs/devtools/device-mode/ . Acedido em 24-04-2023.