Desenvolvimento para a Internet e Aplicações Móveis

2022/23

Integração de Django e React
no desenvolvimento para dispositivos móveis

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* para dispositivos móveis 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 dispositivos móveis. Outras bibliotecas baseadas em JavaScript e também utilizadas para o desenvolvimento de SPAs para dispositivos móveis são Ember, Angular, BackBone e Vue.

Este texto exemplifica a integração de React em Django para o desenvolvimento de uma SPA para dispositivos móveis. A partir do exemplo da Votação que tem vindo a ser desenvolvido nas aulas, é programada uma SPA para a apresentação de Questões e a votação em Opções.

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 crepresentational 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.

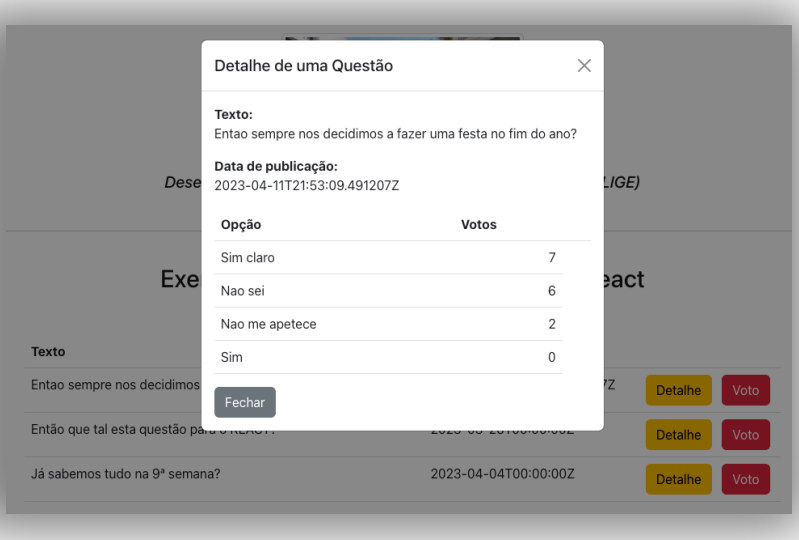
Outras bibliotecas, como Bootstrap para apresentação gráfica, Reactstrap para a programação de componentes React e Axios para gerir os pedidos HTTP, serão instaladas na altura do desenvolvimento do frontend.

4. Exemplo de Integração de Django e React

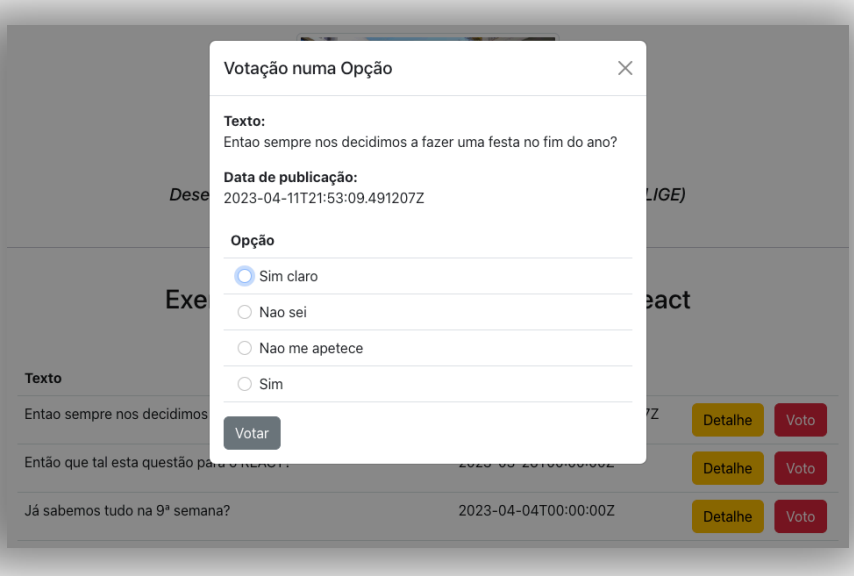
4.1 Resultado pretendido

Este exemplo pretende demonstrar a programação de uma página React, associada ao projeto Django, para a apresentação de questões e a votação em opções. A página React apresenta a lista de questões e será a seguinte:

O detalhe de cada questão será apresentado num pop-up modal:



Finalmente, a votação será realizada noutro pop-up modal:



4.2 Backend em Django

A partir do projeto Django das Questões e das Opções, adapte a APP votacao para que o backend possa responder a invocações REST. Deverá programar pontos de entrada e serializadores. Os serializadores permitem a conversão de dados (instâncias do modelo de dados) entre os formatos JSON e Python. Por outro lado os pontos de entrada no backend

serão programados através de funções em views.py, invocadas por URLs definidos em urls.py.

4.2.1 Instalação de bibliotecas e parametrização

Deverá ter instalado no seu computador o Python 3, o editor PyCharm, a plataforma JavaScript Node.js e a ferramenta npm.

Edite o seu projeto da Votação no PyCharm. No terminal execute o comando seguinte:

```
pip install djangorestframework django-cors-headers
```

Este comando instala o djangorestframework, uma biblioteca para desenvolver aplicações web REST num projeto Django. É também instalada no projeto a biblioteca django-cors-headers, que permite associar *Cross-Origin Resource Sharing* (CORS) aos pedidos HTTP.

Seguidamente, no ficheiro siteiscte/settings.py atualize a lista de INSTALLED_APPS com as dependências para djangorestframework e django-cors-headers. settings.py passa a conter:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'votacao',
    'rest_framework',
    'corsheaders',
]
```

Também em settings.py complete a lista MIDDLEWARE

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
]
```

A seguir à lista MIDDLEWARE introduza a instrução seguinte, para desativar CORS enquanto desenvolve em localhost:

```
CORS_ORIGIN_ALLOW_ALL = True
```

O seu projeto Votação tem as bibliotecas instaladas e está parametrizado, pronto para passar à fase de programação.

4.2.2 Serializadores

Crie um novo ficheiro serializers.py posicionado na diretoria de votacao. Este ficheiro deverá conter o código seguinte, que permitirá serializar, isto é converter entre JSON e Python, as futuras instâncias de Questoes e de Opcoes.

```
from rest_framework import serializers
from .models import Questao, Opcao


class QuestaoSerializer(serializers.ModelSerializer):

    class Meta:                                          #(1)
        model = Questao
        fields = ('pk', 'questao_texto', 'pub_data')


class OpcaoSerializer(serializers.ModelSerializer):

    class Meta:
        model = Opcao
        fields = ('pk', 'questao', 'opcao_texto', 'votos')
```

(1) Repare que as classes de serialização são programadas na forma de classes Meta, que neste caso permitem criar instâncias no formato de outras classes (ver https://docs.python.org/3/reference/datamodel.html#metaclasses). Estas classes Meta identificam os atributos de cada modelo de dados existente na Base de Dados do projeto.

4.2.3 Pontos de Entrada

Para criar os pontos de entrada, comece por colocar os respetivos URLs em urls.py da votação. Junte os URLs seguintes à lista de urlpatterns:

```
path('api/questoes/', views.questoes_lista),
path('api/questoes/<int:pk>', views.questoes_edita),
path('api/opcoes/', views.opcoes_lista),
path('api/opcoes/<int:pk>', views.opcoes_edita),
```

Estes URLs encaminham a execução para as funções em views.py que são os pontos de entrada, isto é que permitirão ao React obter a lista de questões e de opções, e também editar questões e opções, a partir dos URLs.

As novas funções no views.py são as seguintes:

```
from rest_framework.response import Response
from rest_framework.decorators import api_view
from rest_framework import status
from .serializers import *     #(2)
from .models import Questao, Opcao

@api_view(['GET', 'POST'])      #(3)
def questoes_lista(request):
    if request.method == 'GET':     #(4)
        questoes = Questao.objects.all()
        serializerQ = QuestaoSerializer(questoes, context={'request':
request}, many=True)
        return Response(serializerQ.data)
    elif request.method == 'POST':     #(4)
        serializer = QuestaoSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(status=status.HTTP_201_CREATED)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

@api_view(['PUT', 'DELETE'])      #(3) e (5)
def questoes_edita(request, pk):
    try:
        questao = Questao.objects.get(pk=pk)
    except Questao.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)
    if request.method == 'PUT':
        serializer = QuestaoSerializer(questao,
data=request.data,context={'request': request})
        if serializer.is_valid():
            serializer.save()
            return Response(status=status.HTTP_204_NO_CONTENT)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
    elif request.method == 'DELETE':
        questao.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)

@api_view(['GET', 'POST'])
def opcoes_lista(request):
    if request.method == 'GET':
        opcoes = Opcao.objects.all()
        serializerO = OpcaoSerializer(opcoes, context={'request':
request}, many=True)
        return Response(serializerO.data)
    elif request.method == 'POST':
```

```
        serializer = OpcaoSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(status=status.HTTP_201_CREATED)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

@api_view(['PUT', 'DELETE'])
def opcoes_edita(request, pk):
    try:
        opcao = Opcao.objects.get(pk=pk)
    except Opcao.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)
    if request.method == 'PUT':
        serializer = OpcaoSerializer(opcao,
data=request.data,context={'request': request})
        if serializer.is_valid():
            opcao.votos = opcao.votos + 1
            opcao.save()
            #serializer.save()
            return Response(status=status.HTTP_204_NO_CONTENT)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
    elif request.method == 'DELETE':
        opcao.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

Notas sobre os pontos de entrada no views.py:

(2) Importação das classes de serialização.

(3) Cada função é anotada com a natureza dos pontos de entrada que gere: GET e POST, ou então PUT e DELETE.

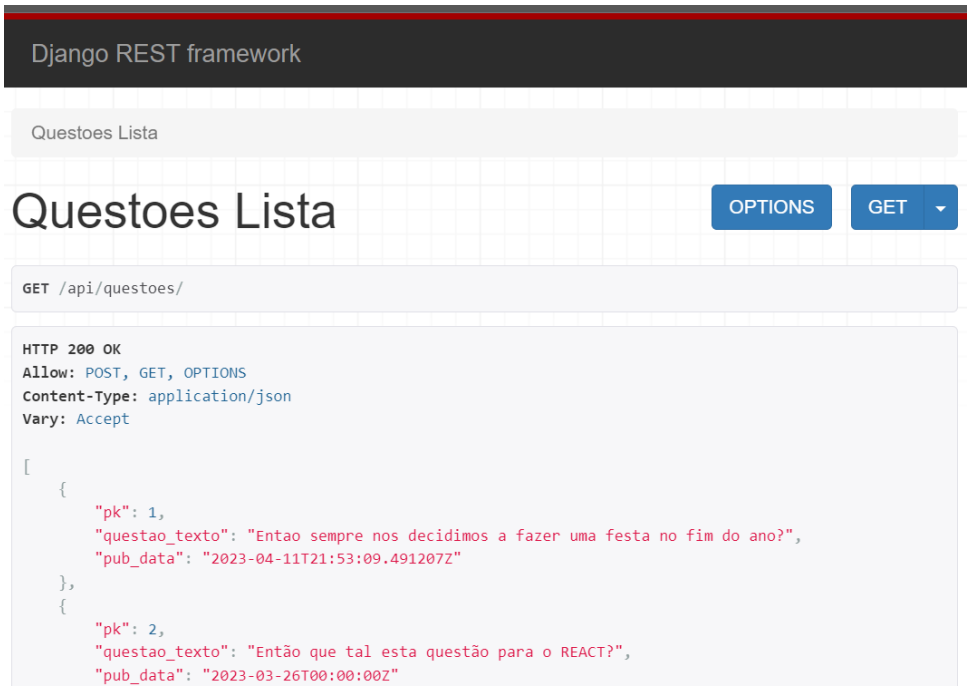
(4) Na função questoes_lista, se o método de request for GET então retorna a lista de todas as questões serializadas. Se o método de request for POST, isto é de criação de uma nova questão, obtém os dados de request.data, vindos do React ainda em formato JSON, serializa-os para os converter em Python e finalmente, se a serialização for válida, grava a nova questão. A função opcoes_lista é semelhante.

(5) Quanto à função questoes_edita, se o método de request for PUT, isto é para a atualização de uma questão existente, obtém os dados de request.data, vindos do React ainda em formato JSON, serializa-os para os converter em Python e finalmente, se a serialização for válida, grava a questão existente com as alterações pretendidas. Se o método de request for DELETE, obtém a questão através do id vindo pelo request e apaga-a com delete().

4.2.4 Teste dos Pontos de Entrada

Para testar os pontos de entrada, comece por executar o servidor Django.

Seguidamente abra http://localhost:8000/votacao/api/questoes/ no seu browser. O acesso a este ponto de entrada invoca GET e retorna a lista de questões formatadas em JSON:



4.3 Frontend em React

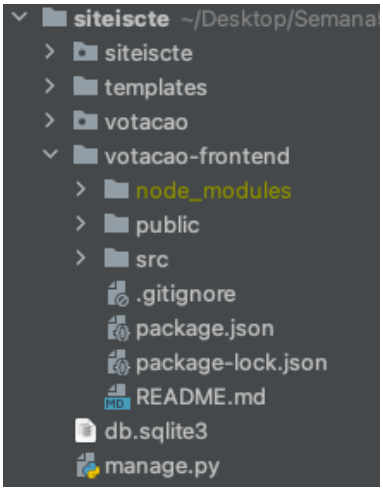
O frontend será programado em React, dentro de uma diretoria própria integrada no nosso projeto Django. Esta diretoria conterá as bibliotecas necessárias para o React e também os nossos programas React programados em JavaScript.

4.3.1 Instalação de bibliotecas

No terminal posicione-se na raiz do projeto (diretoria onde está manage.py). Execute a instrução de criação de uma nova app React chamada votacao-frontend:

```
npx create-react-app votacao-frontend
```

O seu projeto fica agora com a seguinte estrutura.



No terminal entre na diretoria votacao-frontend. Aí instale as bibliotecas Bootstrap e Reactstrap para apresentação gráfica e Axios para gerir os pedidos HTTP. Use o comando seguinte.

```
npm install bootstrap reactstrap axios --save
```

No passo seguinte adicione a instrução ao ficheiro votacao-frontend/src/index.js :

```
import "bootstrap/dist/css/bootstrap.min.css";
```

que fica com o código:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import "bootstrap/dist/css/bootstrap.min.css";

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

reportWebVitals();
```

Ainda na diretoria src crie uma nova diretoria “constants”, e dentro desta um novo ficheiro “index.js”. Neste ficheiro votacao-frontend/src/constants/index.js adicione as seguintes constantes JavaScript, que serão úteis mais tarde:

```
export const API_URL_QUESTOES =
"http://localhost:8000/votacao/api/questoes/";

export const API_URL_OPCOES =
"http://localhost:8000/votacao/api/opcoes/";
```

4.3.2 Componentes CRUD

O frontend é frequentemente composto por diferentes componentes. Este exemplo incluirá os componentes Header para o cabeçalho da página e Home para o seu conteúdo, que apresenta a lista de questões. Serão também programadas dois forms, um para mostrar o detalhe de uma questão e o outro para permitir ao utilizador votar numa opção.

Dentro de src crie uma nova diretoria “components”. Nesta diretoria serão colocados os componentes Header, Home e os dois forms.

O arranque do frontend React começa com a execução do ficheiro src/App.js, que abre Header e Home. Esta última apresentará botões para abrir os dois forms. Assim, o código de src/App.js deverá ser o seguinte:

```
import React, { Component, Fragment } from "react";     //(7)
import Header from "./components/Header";
import Home from "./components/Home";

class App extends Component {     //(6)
  render() {
    return (
      <Fragment>
        <Header />
        <Home />
      </Fragment>
    );
  }
}

export default App;     //(7)
```

(6) O componente App retorna um bloco Fragment, que permite agrupar e retornar vários elementos. Neste caso são retornados dois componentes, Header seguido de Home.

(7) Repare que o código é iniciado com instruções import e é concluído com uma instrução export. Estas instruções são permitidas pela versão ES6 do JavaScript, para

importar e exportar módulos. A instrução export default App indica que o valor a ser exportado por este módulo será uma instância da classe App.

4.3.3 Componente Header

Crie o ficheiro Header.js dentro da diretoria componentes. Introduza o código seguinte:

```
import React from 'react';

function Header() {     //(8)
  return (              //(9)
    <>
      <div className="text-center">
        <img
          src="https://keystoneacademic-
res.cloudinary.com/image/upload/q_auto,f_auto,w_743,c_limit/element/11/111948_2.jpg"
          width="300"
          alt="ISCTE"
          className="img-thumbnail"
          style={{ marginTop: "20px" }}     //(10)
        />
        <br />
        <br />
        <h5>
          <i>Desenvolvimento para a Internet e Aplicações Móveis (LEI e LIGE)</i>
        </h5>
        <br />
        <hr />
        <br />
        <h2>Exemplo de integração de Django com React</h2>
        <br />
      </div>
    </>     //(9)
    );
}

export default Header;
```

Notas sobre Header.js:

(8) Os componentes React podem ser programados através de classes ou de funções. Diversamente da componente App, que foi programada numa classe, a componente Header é aqui programada numa função, com o objetivo de exemplificar esta forma de desenvolver componentes.

(9) A tag <> ... </> existe para incluir todo o código HTML, pois a função return pode retornar apenas um elemento.

(10) As instruções CSS, por exemplo para posicionar a imagem, são colocadas dentro de um bloco {{ ... }} .

(11) O código restante é HTML5.

4.3.4 Componente Home

Crie o ficheiro Home.js dentro da diretoria componentes. O objetivo de Home é de preparar a apresentação das questões com as suas opções.

Introduza o código seguinte:

```
import React, { Component } from "react";
import { Col, Container, Row } from "reactstrap";
import QuestaoLista from "./QuestaoLista";

import axios from "axios";  //(12)

import { API_URL_QUESTOES, API_URL_OPCOES } from "../constants";
//(13)

class Home extends Component {  //(14)
  state = {    //(15)
    questoes: [],
    opcoes: []
  };

  componentDidMount() {    //(16)
    this.resetState();
  }

  getQuestoes = () => {
    axios.get(API_URL_QUESTOES).then(res => this.setState({ questoes:
res.data }));    //(17)
  };

  getOpcoes = () => {
    axios.get(API_URL_OPCOES).then(res => this.setState({ opcoes:
res.data }));       //(18)
  };

  resetState = () => {    //(16)
    this.getQuestoes();
    this.getOpcoes();
  };

  render() {
    return (
      <Container style={{ marginTop: "20px" }}>
        <Row>
          <Col>
            <QuestaoLista
              questoes={this.state.questoes}
              opcoes={this.state.opcoes}
              resetState={this.resetState}
            />
          </Col>
```

```
          </Row>
        </Container>
      );
    }
}

export default Home;
```

Notas sobre Home.js:

(12) É importada a biblioteca Axios, uma API HTTP baseada no interface XMLHttpRequest disponível nos browsers. Axios permitirá fazer a ligação aos pontos de entrada do Django na app votação.

(13) São importadas as constantes que indicam o caminho para os pontos de entrada da app votação.

(14) Este componente é programado com recurso a uma classe.

(15) O estado de Home é registado no dicionário state. Inicialmente a lista de questões e a lista de opções estão vazias.

(16) Quando o componente é inicializado a função componentDidMount() chama resetState(), que por sua vez invoca getQuestoes() e getOpcoes() para carregar as listas de questões e de opções no dicionário state.

(17) getQuestoes() acede ao ponto de entrada registado em API_URL_QUESTOES através da função GET, o que invoca questoes_lista(*request*) no views da app votacao. É assim obtida a lista de questões, que é guardada em this.state.questoes.

(18) Em getOpcoes() o procedimento é semelhante a getQuestoes().

(19) Home retorna um Container, que inclui uma única linha com uma única coluna, isto é uma célula, onde será apresentada a lista de questões. Para isso é invocado o componente QuestaoLista, ao qual são passadas as listas de questões e de opções.

4.3.5 Componente QuestaoLista

Crie o ficheiro QuestaoLista.js dentro da diretoria componentes. O objetivo de QuestaoLista é de apresentar a lista de questões e também de dar acesso aos dois forms, um para mostrar o detalhe de uma questão e o outro para votar numa opção de uma questão.

```
import React, { Component } from "react";
import { Table } from "reactstrap";
```

```
import DetalheQuestaoModal from "./DetalheQuestaoModal";
import VotaModal from "./VotaModal";

class QuestaoLista extends Component {
  render() {
    const questoes = this.props.questoes;            {/* (20) */}
    const opcoes = this.props.opcoes;
    return (
      <Table light>                                  {/* (21) */}
        <thead>                                      {/* (22) */}
          <tr>
            <th>Texto</th>
            <th>Data de criação</th>
            <th></th>
          </tr>
        </thead>
        <tbody>                                      {/* (23) */}
          {!questoes || questoes.length <= 0 ? (
            <tr>
              <td colSpan="6" align="center">
                <b>Não há questões</b>
              </td>
            </tr>
          ) : (
            questoes.map(questao => (                // (24)
              <tr key={questao.pk}>
                <td>{questao.questao_texto}</td>
                <td>{questao.pub_data}</td>
                <td align="center">
                  <DetalheQuestaoModal
                    create={false}
                    questao={questao}
                    opcoes={opcoes}
                    resetState={this.props.resetState}
                  />
                    
                  <VotaModal
                    questao={questao}
                    opcoes={opcoes}
                    resetState={this.props.resetState}
                  />
                </td>
              </tr>
            ))
          )}
        </tbody>
      </Table>
    );
  }
}

export default QuestaoLista;
```

(20) QuestaoLista começa por receber as listas de questões e de opções nas suas propriedades (this.props).

(21) A lista de questões será apresentada numa tabela, em que cada questão surgirá numa linha.

(22) O cabeçalho da tabela tem uma linha com três células: uma para o texto da questão, outra para a sua data e outra célula vazia. Esta célula vazia é o cabeçalho da coluna onde serão colocados os botões de detalhe e de votação.

(23) O corpo da tabela apresenta a lista de questões. Se a lista não existe ou se a sua dimensão é 0 e é escrito na tabela"Não há questões".

(24) A lista de questões é percorrida com map (ver https://www.w3schools.com/jsref/jsref_map.asp). Cada linha da tabela é ocupada por uma questão. Na primeira célula é colocado o seu texto, na segunda a sua data e na terceira célula são colocadas instâncias dos componentes DetalheQuestaoModal e de VotaModal, que apresentam botões para o detalhe da questão e para votar numa das suas opções. Estes componentes serão apresentados de seguida.

4.3.5 Componente DetalheQuestaoModal

Crie o ficheiro DetalheQuestaoModal.js dentro da diretoria componentes. O objetivo de DetalheQuestaoModal é de abrir uma janela modal, isto é uma janela de pop-up subordinada à nossa página SPA (Single Page Application). Esta janela modal será depois preenchida com o form de detalhe.

DetalheQuestaoModal.js deverá conter o código seguinte:

```
import React, { Component, Fragment } from "react";
import { Button, Modal, ModalHeader, ModalBody } from "reactstrap";
import DetalheQuestaoForm from "./DetalheQuestaoForm";

class DetalheQuestaoModal extends Component {
  state = {
    modal: false                                          // (25)
  };

  toggle = () => {                                        // (26)
    this.setState(previous => ({
      modal: !previous.modal
    }));
  };

  render() {
    var title = "Detalhe de uma Questão";
    var button = <Button onClick={this.toggle}
color="warning">Detalhe</Button>;                         // (27)
    return (
      <Fragment>
        {button}                                          {/*(28)*/}
        <Modal isOpen={this.state.modal} toggle={this.toggle}>
                                                          {/*(29)*/}
          <ModalHeader toggle={this.toggle}>{title}</ModalHeader>
```

```
                                                                {/*(30)*/}
          <ModalBody>                                           {/*  (31)  */}
            <DetalheQuestaoForm
              resetState={this.props.resetState}
              toggle={this.toggle}
              questao={this.props.questao}
              opcoes={this.props.opcoes}
            />
          </ModalBody>
        </Modal>
      </Fragment>

    );
  }
}

export default DetalheQuestaoModal;
```

(25) O estado inicial da variável boleana modal é false.

(26) A função toggle inverte o estado da variável modal.

(27) O botão chama toggle() e inverte o valor boleano de modal.

(28) É retornada uma tag Fragment que inicialmente afixa o botão Detalhe na linha da questão.

(29) Se this.state.modal é true, então isOpen é true e abre a janela modal.

(30) Quando abre mostra o cabeçalho com o título.

(31) O corpo da janela modal inclui o componente DetalheQuestaoForm e passa-lhe a questão e a lista de opções.

4.3.6 Componente DetalheQuestaoForm

Crie o ficheiro DetalheQuestaoForm.js dentro da diretoria componentes. O objetivo deste componente apresentar um form que mostra o detalhe de uma questão.

O ficheiro deverá conter o código seguinte:

```
import React from "react";
import { Button, Form, FormGroup, Table } from "reactstrap";

class DetalheQuestaoForm extends React.Component {
  state = {                                                     // (32)
    pk: 0,
    questao_texto: "",
```

```
    pub_data: "",
    opcao_set: []
  };

  componentDidMount() {                                    // (33)
    if (this.props.questao) {
      const { pk, questao_texto, pub_data } = this.props.questao;
      this.setState({ pk, questao_texto, pub_data});
    }
    if (this.props.opcoes) {                               // (34)
      const todasOpcoes = this.props.opcoes;
      const questao = this.props.questao;
      const opcao_set = [];
      for (let i = 0; i < todasOpcoes.length; i++) {
        if (questao.pk === todasOpcoes[i].questao){
                opcao_set.push(todasOpcoes[i]);
            }
      }
      this.setState({ opcao_set });
    }
  }

  fechaModalQuestao = e => {
    e.preventDefault();
    this.props.toggle();
  };

  defaultIfEmpty = value => {
    return value === "" ? "" : value;
  };

  render() {                                               // (35)
    return (
      <Form onSubmit={this.fechaModalQuestao}>
        <FormGroup>
          <b>Texto:</b>
          <p>{this.defaultIfEmpty(this.state.questao_texto)} </p>
        </FormGroup>
        <FormGroup>
          <b>Data de publicação:</b>  
          <p>{this.defaultIfEmpty(this.state.pub_data)}</p>
        </FormGroup>
        <FormGroup>
            <Table>
                <thead>
                    <tr>
                        <th colSpan="6" align="left">Opção</th>
                        <th colSpan="6" align="right">Votos</th>
                        <th></th>
                        </tr>
                </thead>
                <tbody>                                    {/* (36) */}
                    {
                    !this.state.opcao_set ||
this.state.opcao_set.length <= 0 ? (
                        <tr>
                            <td colSpan="14" align="left">
                                <i>Não há opções</i>
                            </td>
                        </tr>
                    ) : (
```

```
                    this.state.opcao_set.map((opcao) =>
                        <tr>
                            <td colSpan="6" align="left">
                            {opcao.opcao_texto}
                            </td>
                            <td colSpan="6" align="right">
                            {opcao.votos}
                            </td>
                        </tr>
                    ))
                    }
                </tbody>
            </Table>
        </FormGroup>
        <Button>Fechar</Button>                                    {/* (37) */}
      </Form>
    );
  }
}

export default DetalheQuestaoForm;
```

(32) O estado do DetalheQuestaoForm inclui os atributos da questão, desde a sua chave primária até à sua lista de opções.

(33) A inicialização do form é operada na função componentDidMount(). A partir da questão são carregadas as variáveis pk, questão_texto e pub_data em this.state. Fica a faltar apenas a lista de opções.

(34) Este ciclo percorre todas as opções, identifica as que são associadas a esta questão e coloca-as na lista opção_set de this.state. A inicialização fica completa.

(35) O retorno de DetalheQuestaoForm é um form. O primeiro FormGroup respeita ao texto da questão. É invocado o método defaultIfEmpty que mostra a questão_texto, ou então “” se esta não existe. O FormGroup seguinte refere-se à data de publicação e o procedimento é o mesmo.

(36) Finalmente, o terceiro FormGroup apresenta uma tabela com as opções da questão, incluindo o texto de cada opção e o seu número de votos.

(37) O form termina com um botão que executa a função fechaModalQuestao indicada no onSubmit inicial.

4.3.7 Componente VotaModal

Crie o ficheiro VotaModal.js dentro da diretoria componentes. O objetivo de VotaModal é de abrir uma janela modal que permite votar numa opção da questão. VotaModal.js deve conter o código seguinte:

```
import React, { Component, Fragment } from "react";
import { Button, Modal, ModalHeader, ModalBody } from "reactstrap";
import VotaForm from "./VotaForm";

class VotaModal extends Component {
  state = {
    modal: false
  };

  toggle = () => {
    this.setState(previous => ({
      modal: !previous.modal
    }));
  };

  render() {
    var title = "Votação numa Opção";
    var button = <Button onClick={this.toggle}
color="danger">Voto</Button>;
    return (
      <Fragment>
        {button}
        <Modal isOpen={this.state.modal} toggle={this.toggle}>
          <ModalHeader toggle={this.toggle}>{title}</ModalHeader>
          <ModalBody>
            <VotaForm
              resetState={this.props.resetState}
              toggle={this.toggle}
              questao={this.props.questao}
              opcoes={this.props.opcoes}
            />
          </ModalBody>
        </Modal>
      </Fragment>

    );
  }
}

export default VotaModal;
```

O funcionamento de VotaModal é semelhante ao de DetalheQuestaoModal, não sendo necessárias explicações detalhadas.

4.3.8 Componente VotaForm

Crie o ficheiro VotaForm.js dentro da diretoria componentes. Este ficheiro deve conter o código seguinte:

```
import React from "react";
import { Button, Form, FormGroup, Table } from "reactstrap";
import axios from "axios";
import { API_URL_OPCOES } from "../constants";

class VotaForm extends React.Component {
  state = {
    pk: 0,
    questao_texto: "",
    pub_data: "",
    opcao_set: [],
    selectedOption: 0                                          // (38)
  };

  componentDidMount() {
    if (this.props.questao) {
      const { pk, questao_texto, pub_data } = this.props.questao;
      this.setState({ pk, questao_texto, pub_data});
    }
    if (this.props.opcoes) {
      const todasOpcoes = this.props.opcoes;
      const questao = this.props.questao;
      const opcao_set = [];
      for (let i = 0; i < todasOpcoes.length; i++) {
        if (questao.pk === todasOpcoes[i].questao){
                opcao_set.push(todasOpcoes[i]);
             }
      }
      this.setState({ opcao_set });
    }
  }

  onChange = e => {
    this.setState({ [e.target.name]: e.target.value });
  };

  votaEFechaModal = e => {                                     // (41)
    e.preventDefault();
    for(let i=0; i<this.props.opcoes.length; i++){
        if(this.props.opcoes[i].pk == this.state.selectedOption){
            axios.put(API_URL_OPCOES + this.props.opcoes[i].pk,{'pk':
this.props.opcoes[i].pk,
                                                        'questao':
this.props.opcoes[i].questao,
                                                        'opcao_texto':
this.props.opcoes[i].opcao_texto,
                                                        'votos':
this.props.opcoes[i].votos++} );
        }
    }
    this.props.toggle();
  };

  defaultIfEmpty = value => {
    return value === "" ? "" : value;
  };

handleOptionChange = changeEvent => {                          // (40)
  const optionid = changeEvent.target.value;
  this.state.selectedOption = optionid;
```

```
};

  render() {
    return (
      <Form onSubmit={this.votaEFechaModal}>
        <FormGroup>
          <b>Texto:</b>
          <p>{this.defaultIfEmpty(this.state.questao_texto)} </p>
        </FormGroup>
        <FormGroup>
          <b>Data de publicação:</b>  
          <p>{this.defaultIfEmpty(this.state.pub_data)}</p>
        </FormGroup>
        <FormGroup>
            <Table>
                <thead>
                    <tr>
                        <th colSpan="6" align="left">Opção</th>
                    </tr>
                </thead>
                <tbody>
                    {
                    !this.state.opcao_set ||
this.state.opcao_set.length <= 0 ? (
                        <tr>
                            <td colSpan="14" align="left">
                                <i>Não há opções</i>
                            </td>
                        </tr>
                    ) : (
                    this.state.opcao_set.map((opcao) =>
                        <tr>
                            <td colSpan="6" align="left">
                              
                              <label>
                                <input
                                    type="radio"
                                    name="react-radio"
                                    value={opcao.pk}
                                    checked={false}
                                    className="form-check-input"

onChange={this.handleOptionChange}      // (39)
                                />
                                  {opcao.opcao_texto}
                              </label>
                            </td>
                        </tr>
                    ))
                    }
                </tbody>
            </Table>
        </FormGroup>
        <Button>Votar</Button>
      </Form>
    );
  }
}

export default VotaForm;
```

O funcionamento de VotaForm é semelhante ao de DetalheQuestaoForm, exceto nos aspetos seguintes.

(38) O dicionário state inclui a opção selecionada.

(39) Um evento de alteração num radiobutton invoca a função handleOptionChange.

(40) A função handleOptionChange guarda o id da opção em this.state.selectedOption.

(41) A função votaEFechaModal acede ao ponto de contacto de opções com PUT e invoca opções_edita(request, pk) para atualizar a questão e incrementar o seu atributo votos.

4.4 Execução do exemplo

Depois da programação concluída execute o backend (app Django) e o frontend (app React.

Comece por executar o backend na diretoria da raiz do projeto, com a instrução do costume:

```
python manage.py runserver
```

Seguidamente, para o frontend, abra uma nova janela de terminal no sistema operativo. Navegue até à diretoria da app React, isto é .../siteiscte/votação-frontend. Execute a app:

```
npm start
```

Em http://localhost:3000/ poderá testar a app React !

4.5 Simulação do comportamento num dispositivo móvel

Finalmente, pode simular a execução da app React num dispositivo móvel com dimensões variadas. Existem vários simuladores disponíveis. Por exemplo, no browser Chrome abra http://localhost:3000/ com a opção do menu Developer Tools. Depois teste o comportamento gráfico da app React em várias dimensões de dispositivos móveis.

Termine o frontend e o backend com CTRL-C.

5. Bibliografia

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.