

Next Level Week

📌 Aulas realizadas em 01/06/20 - 05/06/20

1. Introdução

▼ MVC X API Restful

A **API Restful** permite que o back-end apenas retorne os **dados da aplicação**, geralmente em formato **JSON**, que pode ser **consumido** pelo **front-end** e pela aplicação **mobile**. Enquanto isso, o modelo MVC só retorna diretamente HTML, que não pode ser tão facilmente consumido por qualquer framework front-end ou pela aplicação mobile.



▼ TypeScript

Por que TypeScript?

```
/**
 * Função que exibe dados do usuário
 */
function displayUserInformation(user) {
  return `${user.name} - ${user.email}`;
}

export default displayUserInformation;
```

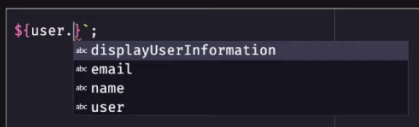
Desafio: Incluir a informação da cidade e UF do usuário nesse retorno.

- Qual o formato do objeto de usuário?
- Utilizo `user.city` ou `user.address.city`?
- Será que a cidade e UF são obrigatórias?



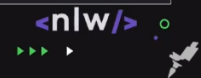
IntelliSense

O editor não conhece o formato da variável `user` e, por isso, não consegue determinar suas propriedades.



Nesse caso, o editor consegue saber exatamente os dados que um usuário pode ter e oferecer inteligência de IDE.

```
erInformation(user: User) {
  ame) - ${user.email} de ${user.}`;
  address
  email
  pl name
  (property) User.address: {
    city: string;
    uf: string;
  }
```



Mitos e verdades

- TypeScript diminui a produtividade
- TypeScript é transformar JavaScript em Java ou C#
- O mercado não usa TypeScript
- TypeScript substitui o JavaScript por completo
- TypeScript atrapalha quem é iniciante



▼ Configurando o projeto back-end

- Instalação do Node.js
- Após criar a pasta **server**, executar os seguintes comandos dentro dela:
 - Para criar o arquivo package.json:

```
npm init -y
```

- Para instalar o **Express**, que permite o uso de rotas:

```
npm install express
```

- Como será usado o TypeScript, é preciso instalar a biblioteca de definição de tipos do Express.

```
npm install @types/express -D
```



Como todo código TypeScript é convertido para JavaScript quando o projeto é executado, no bundle de produção não são necessárias bibliotecas voltadas para o TypeScript. Por isso, várias dependências do TypeScript são instaladas como **dependências de desenvolvimento** (com o -D no final).

- Para instalar o TypeScript:

```
npm install typescript -D
```

- Para instalar o TypeScript para Node.js:

```
npm install ts-node -D
```

- Para criar arquivo de configuração do TypeScript:

```
npx tsc --init
```



Comandos que começam com **npx** servem para executar algum pacote do Node já instalado.

- Para atualizar o servidor com as mudanças em tempo real:

```
npm install ts-node-dev -D
```

- Para executar o projeto:

```
npx ts-node src/server.ts
```

- Ou, para criar um outro comando para executar o projeto mais facilmente, adicionar o seguinte script no **package.json**:

```
.gitignore  server.ts  package.json X
server > package.json > {} scripts > dev
1  {
2    "name": "server",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1",
8      "dev": "ts-node-dev src/server.ts"
9    },
10   "keywords": [],
11   "author": "",
12   "license": "ISC",
```

- Assim, para executar o projeto basta usar:

```
npm run dev
```

▼ ReactJS

O que é React?

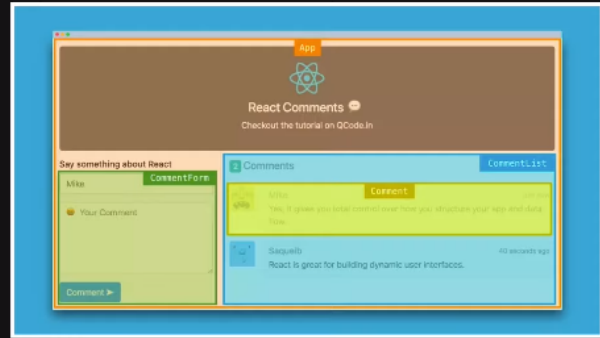
- Biblioteca para construção de interfaces;
- Utilizado para construção de Single-Page Applications;
- Podemos chamar de framework?
- Tudo fica dentro do Javascript;
- React / ReactJS / React Native;

```
import React from 'react';
import './button.css';
import icon from './button.png';

function Button() {
  return (
    <button>
      <img src={icon} />
    </button>
  );
}
```

Vantagens

- Organização do código;
- Componentização;
- Divisão de responsabilidades;
- Back-end: Regra de negócio
- Front-end: Interface
- Uma API, múltiplos clientes;



▼ Configurando o projeto front-end

- Criar o projeto web com a template para o TypeScript:

```
npx create-react-app web --template-typescript
```

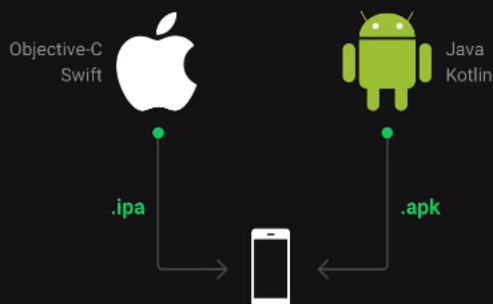
- Comando para executar o projeto:

```
npm start
```

▼ React Native & Expo

Entendendo o React Native

Abordagem tradicional



Na abordagem tradicional, criamos uma aplicação para iOS e outra para Android, e nesses casos, o trabalho se torna repetido tanto para criação quanto para as alterações no projeto.

Abordagem do React Native



Todo código feito e em JavaScript, esse código não é convertido em código nativo, melhor do que isso, o dispositivo passa a entender o código JavaScript e a interface gerada é totalmente nativa.

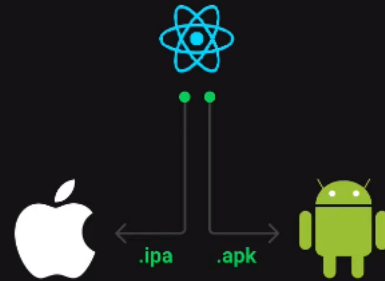
Entendendo o React Native

Abordagem tradicional



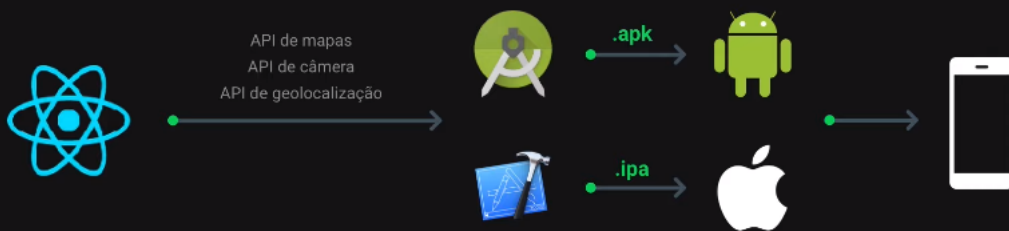
Na abordagem tradicional, criamos uma aplicação para iOS e outra para Android, e nesses casos, o trabalho se torna repetido tanto para criação quanto para as alterações no projeto.

Abordagem do React Native



Todo código feito é em JavaScript, esse código não é convertido em código nativo, melhor do que isso, o dispositivo passa a entender o código. JavaScript e a interface gerada é totalmente nativa.

Por que utilizaremos o Expo?



Sem o Expo, precisamos instalar em nosso sistema tanto o Android Studio para obter a SDK de desenvolvimento Android, e o Xcode (apenas no Mac) para obter a SDK do iOS.

Nesse caso, nossa iniciação no desenvolvimento fica mais penosa, já que essas SDK's não são extremamente simples de instalar e livres de erros.



2. Back-End

▼ Identificando entidades da aplicação

- **points** (Pontos de Coleta)
 - image
 - name
 - email
 - whatsapp
 - latitude
 - longitude
 - city
 - uf
- **items** (Itens para Coleta)
 - image
 - title
- Relacionamento **Muitos para Muitos (N-N)** gera tabela **point_items** (itens que um ponto coleta)
 - point_id
 - item_id

▼ Identificando funcionalidades da aplicação

- Cadastro de pontos de coleta
- Listar itens de coleta
- Listar pontos de coleta (filtro por estado/cidade/itens)
- Listar um ponto de coleta específico

▼ Banco de dados SQLite + Knex

- ▼ Criando dados que já serão previamente cadastrados no banco (itens de coleta)
 - Será feito usando a funcionalidade **Seeds** do Knex

- Criar arquivo **criar_items.ts** dentro de database > seeds, com o comando **insert** e os dados a serem inseridos

```

1  import Knex from 'knex';
2
3  export async function seed(knex: Knex) {
4    await knex('items').insert([
5      { title: 'Lâmpadas', image: 'lampadas.svg' },
6      { title: 'Pilhas e Baterias', image: 'baterias.svg' },
7      { title: 'Papéis e Papelão', image: 'papeis-papelao.svg' },
8      { title: 'Resíduos Eletrônicos', image: 'eletronicos.svg' },
9      { title: 'Resíduos Orgânicos', image: 'organicos.svg' },
10     { title: 'Óleo de Cozinha', image: 'oleo.svg' }
11   ]);
12 }

```

- Adicionar configuração para os seeds dentro de **knex.file.ts**, junto com as migrations

```

1  // path é uma lib do node que serve para padronizar caminhos de arquivos
2  import path from 'path';
3
4  module.exports = {
5    client: 'sqlite3',
6    connection: {
7      filename: path.resolve(__dirname, 'src', 'database', 'database.sqlite'),
8    },
9    migrations: {
10     directory: path.resolve(__dirname, 'src', 'database', 'migrations')
11   },
12   seeds: {
13     directory: path.resolve(__dirname, 'src', 'database', 'seeds')
14   },
15   useNullAsDefault: true,
16 };

```

- Para facilitar a criação das seeds, criar um script dentro de **package.json** com um comando que será executado para criar as seeds

```

1  {
2    "name": "server",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \\\"Error: no test specified\\\" && exit 1",
8      "dev": "ts-node-dev src/server.ts",
9      "knex:seed": "knex --knexfile knexfile.ts seed:run",
10     "knex:migrate": "knex --knexfile knexfile.ts migrate:latest"
11   },
12   "keywords": [],
13   "author": "",
14   "license": "ISC",
15   "dependencies": {
16     "express": "^4.17.1",

```


- Com isso, basta rodar o seguinte comando para criar as seeds:

```
npm run knex:seed
```

- Depois disso, já é possível visualizar a tabela **items** com os dados inseridos (usando a extensão do SQLite)

The screenshot shows the VS Code interface with the SQLite Explorer on the left, displaying the 'items' table. The main editor shows the 'create_items.ts' file with a seed function. The SQLite Explorer on the right shows the result of a 'SELECT * FROM items;' query.

#	id	image	title
1	1	lampadas.svg	Lâmpadas
2	2	baterias.svg	Pilhas e Baterias
3	3	papeis-papelao.svg	Papéis e Papelão
4	4	eletronicos.svg	Resíduos Eletrônicos
5	5	organicos.svg	Resíduos Orgânicos
6	6	oleo.svg	Óleo de Cozinha

▼ Selecionando tudo de uma tabela do banco (SELECT *)

- Exemplo de método GET selecionando todos os dados da tabela de itens

```
routes.get('/items', async (request, response) => {
  const items = await knex('items').select('*');

  // serialização de dados: transformar os dados para torná-los mais acessíveis, adequados
  const serializedItems = items.map(item => {
    return {
      id: item.id,
      name: item.title,
      image_url: `http://localhost:3333/uploads/${item.image}`,
    };
  });

  return response.json({serializedItems});
});
```

```
routes.get('/items', async (request, response) => {
  const items = await knex('items').select('*');

  // serialização de dados: transformar os dados para torná-los mais acessíveis, adequados
  const serializedItems = items.map(item => {
    return {
      id: item.id,
      name: item.title,
      image_url: `http://localhost:3333/uploads/${item.image}`,
    };
  });

  return response.json({serializedItems});
});
```

- Os dados também foram **serializados**: isso quer dizer que foram adaptados para um retorno de acordo com o desejado (neste caso, trocando **title** por **name**, adicionando a url completa da imagem em **image_url**)

▼ Inserindo dados no banco e usando TRANSAÇÕES

- Cada ponto de coleta deve coletar um ou mais itens recicláveis. Por isso, cada ponto de coleta está relacionado a um ou mais itens, e cada item está relacionado a um ou mais pontos de coleta.
- Por isso, ao inserir um ponto de coleta deve-se também receber e inserir os itens coletados por ele na tabela point_items do banco. Essa é uma tabela que relaciona pontos de coleta x itens coletados.

```
SELECT * FROM 'point_items';
```

#	id	point_id	item_id
1	1	2	1
2	2	2	2
3	3	2	6

- Exemplo de um método POST para inserir um ponto de coleta no banco:

```
async create(request: Request, response: Response) {

  // aqui é usada a desestruturação do JS
  // cada um é igual a ex.: const name = request.body.name
  const {
    name,
    email,
    whatsapp,
    latitude,
    longitude,
    city,
    uf,
    items
  } = request.body;

  // usando uma transaction com o knex
  const trx = await knex.transaction();

  // aqui é usada short syntax, porque as variáveis são iguais aos nomes
  // cada um é igual a ex.: name: name
  const point = {
    image: 'image-fake',
    name,
    email,
    whatsapp,
    latitude,
    longitude,
    city,
    uf
  };

  // o knex retorna um array de ids (insertedIds) após cada inserção, contendo todos os registros inseridos
  const insertedIds = await trx('points').insert(point);

  // iremos pegar o insertedIds[0], pois somente 1 registro foi inserido na tabela
  const point_id = insertedIds[0];

  // mapeando o array de itens cadastrados no formulário para esse ponto de coleta
  // para cada item, retornar um objeto contendo esse próprio item (o id dele) e o id do ponto
  // (pegado acima após fazer a inserção no bd)
  const pointItems = items.map((item_id: number) => {
    return {
      item_id,
      point_id,
    }
  });

  // com isso, inserir na tabela point_items cada registro contendo o relacionamento ponto x item
  await trx('point_items').insert(pointItems);

  await trx.commit();

  // retornando o ponto cadastrado + o id criado automaticamente
  return response.json({
    id: point_id,
    ...point,
  });
}
```

- Como existem dois **inserts**, é necessário fazer uma **transação**, para que ou os dois inserts sejam executados, ou nenhum; para fazer uma transação, basta criar uma constante, atribuindo a ela `const trx = await knex.transaction();` e depois usar essa mesma constante (`trx`) no lugar do knex ao fazer os inserts.
- No final dos inserts, é preciso fazer o **commit**, escrevendo `await trx.commit();`

💡 Nomes padrão de métodos dentro de Controllers: **index, show, create, update, delete.**

▼ Selecionando um único ponto de coleta a partir do seu ID + seus itens coletados (JOIN)

- Exemplo de código do método **show** que fica no **PointsController**:

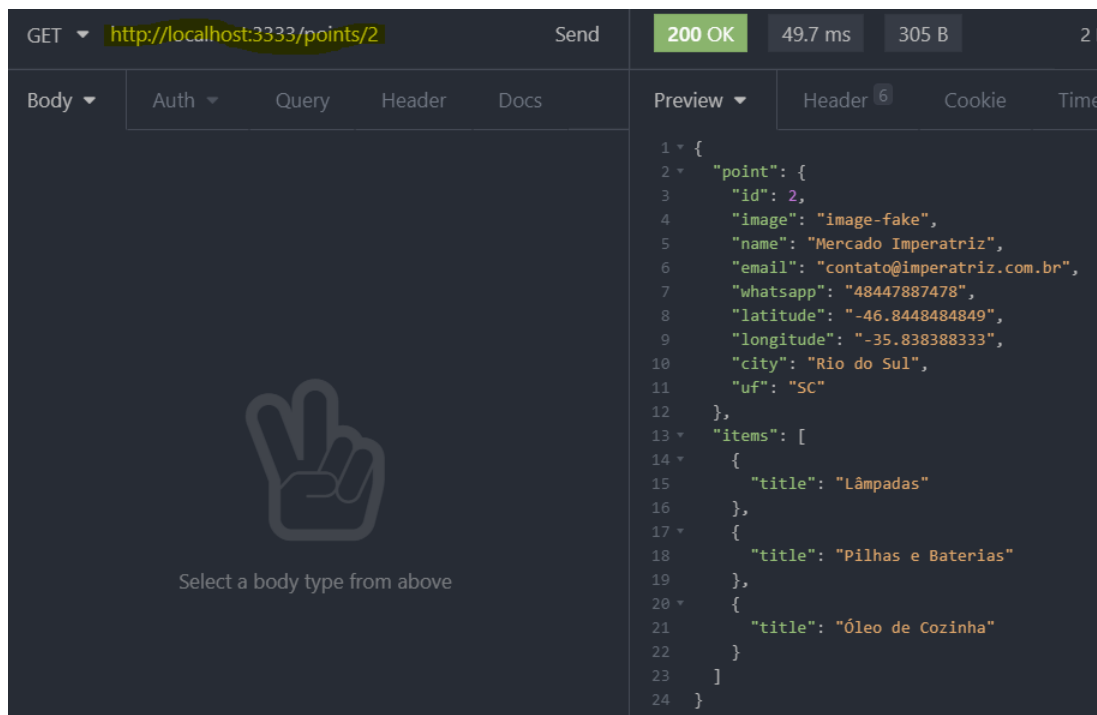
```
async show(request: Request, response: Response) {
  const { id } = request.params;

  const point = await knex('points').where('id', id).first();

  if (!point) {
    return response.status(400).json({ message: 'Point not found.' });
  }

  /**
   * SELECT * FROM items
   * JOIN point_items ON items.id = point_items.item_id
   * WHERE point_items.point_id = {id}
   */
  const items = await knex('items')
    .join('point_items', 'items.id', '=', 'point_items.item_id')
    .where('point_items.point_id', id)
    .select('items.title');

  return response.json({ point, items });
}
```



GET <http://localhost:3333/points/2> Send 200 OK 49.7 ms 305 B

Body Auth Query Header Docs Preview Header 6 Cookie Time

Select a body type from above

```
1 {
2   "point": {
3     "id": 2,
4     "image": "image-fake",
5     "name": "Mercado Imperatriz",
6     "email": "contato@imperatriz.com.br",
7     "whatsapp": "48447887478",
8     "latitude": "-46.8448484849",
9     "longitude": "-35.838388333",
10    "city": "Rio do Sul",
11    "uf": "SC"
12  },
13  "items": [
14    {
15      "title": "Lâmpadas"
16    },
17    {
18      "title": "Pilhas e Baterias"
19    },
20    {
21      "title": "Óleo de Cozinha"
22    }
23  ]
24 }
```

▼ Selecionando pontos de coleta filtrados por cidade, UF e itens coletados

- Exemplo do método **index** dentro do **PointsController**:

```
async index(request: Request, response: Response) {
  // filtros de cidade, uf, items => Query Params
  const { city, uf, items } = request.query;
```

```
// pega os itens enviados (formato 1,2,3) e converte em um array, dividindo pela ,
// e pegando cada item sem espaços e convertido em número
const parsedItems = String(items)
  .split(',')
  .map(item => Number(item.trim()));

const points = await knex('points')
  .join('point_items', 'points.id', '=', 'point_items.point_id')
  .whereIn('point_items.item_id', parsedItems)
  .where('city', String(city))
  .where('uf', String(uf))
  .distinct()
  .select('points.*');

return response.json(points);
}
```

GET http://localhost:3333/points Send 200 OK 10.1 ms 411 B 24 Minutes Ago

Body Auth Query 3 Header Docs

URL PREVIEW
<http://localhost:3333/points?city=Rio%20do%20Sul&uf=SC&items=6>

city	Rio do Sul
uf	SC
items	6
New name	New value

Preview 6 Header Cookie Timeline

```
1 [
2 {
3   "id": 2,
4   "image": "image-fake",
5   "name": "Mercado Imperatriz",
6   "email": "contato@imperatriz.com.br",
7   "whatsapp": "48447887478",
8   "latitude": "-46.8448484849",
9   "longitude": "-35.838388333",
10  "city": "Rio do Sul",
11  "uf": "SC"
12 },
13 {
14   "id": 3,
15   "image": "image-fake",
16   "name": "Mercado do seu Zé",
17   "email": "contato@imperatriz.com.br",
18   "whatsapp": "48447887478",
19   "latitude": "-46.8448484849",
20   "longitude": "-35.838388333",
21   "city": "Rio do Sul",
22   "uf": "SC"
23 }
24 ]
```

▼ Usando CORS

- O **CORS (Cross-Origin Resource Sharing)** define quais URLs terão acesso à API (backend), geralmente é apenas a URL do frontend
- Adicioná-lo executando os comandos (o segundo apenas por causa do TypeScript, por isso é uma dependência de desenvolvimento)

```
npm install cors
npm install @types/cors -D
```

- Configurar o CORS dentro do **server.ts**. Quando a aplicação vai para produção, adicionar dentro dele as URLs (frontend) que terão acesso à API.

```
server > src > server.ts > origin
1 import express from 'express';
2 import cors from 'cors';
3 import routes from './routes';
4 // path é uma lib do node que serve para pad
5 import path from 'path';
6
7 // criando a aplicação
8 const app = express();
9
10 app.use(cors({
11   origin: 'www...com.br'
12 }));
13
14 app.use(express.json());
15
16 app.use(routes);
17
18 // diretório de uploads
19 app.use('/uploads', express.static(path.resolve(
```

- No ambiente de desenvolvimento, não é necessário colocar nada.

```
server > src > server.ts > ...
1 import express from 'express';
2 import cors from 'cors';
3 import routes from './routes';
4 // path é uma lib do node que serv
5 import path from 'path';
6
7 // criando a aplicação
8 const app = express();
9
10 app.use(cors());
11
12 app.use(express.json());
13
14 app.use(routes);
15
16 // diretório de uploads
17 app.use('/uploads', express.static
18
```

3. Front-End

▼ Configurando projeto

- Adicionar TypeScript usando

```
yarn add typescript
```

▼ TypeScript + Propriedades

- As **props** são diferentes usando TypeScript.

```
App.tsx x
src > App.tsx > ...
1 import React from 'react';
2
3 import './App.css';
4 import Header from './Header';
5
6 function App() {
7   return (
8     <div>
9       <Header title="Ecoleta" />
10     </div>
11   );
12 }
13
14 export default App;
15

Header.tsx x
src > Header.tsx > default
1 import React from 'react';
2
3 interface HeaderProps {
4   title: string;
5 }
6
7 const Header: React.FC<HeaderProps> = (props) => {
8   return (
9     <header>
10       <h1>{props.title}</h1>
11     </header>
12   );
13 }
14
15 export default Header;
```

- Ao serem passadas para o componente filho, no componente filho `Header` deve existir uma interface `HeaderProps` que especifica o **tipo** e o **nome** da(s) prop(s) passada(s). A **arrow function** `Header` também recebe um tipo (`React.FC` ou `React.FunctionComponent`) que deve ter um genérico/parâmetro (`<>`). Nesse caso, esse parâmetro é o `HeaderProps`.
 - Assim, é possível **especificar qual o tipo** das variáveis recebidas como props.

▼ Pacotes para instalar:

- Para usar ícones do Material Icons, FontAwesome, etc. dentro do React:

```
npm install react-icons
```

- Para usar rotas:

```
npm install react-router-dom
npm install @types/react-router-dom -D
```

- Para fazer requisições HTTP à API:

```
npm install axios
```

▼ Adicionando mapas ao formulário com Leaflet

- Instalar o pacote

```
npm install leaflet react-leaflet
npm install @types/react-leaflet -D
```

- Adicionar ao `index.html` o código abaixo

```
<link
  rel="stylesheet"
  href="https://unpkg.com/leaflet@1.6.0/dist/leaflet.css"
  integrity="sha512-xwE/Az9zrjBIPhAcBb3F6JVqxf46+CDLwFLMH1oNu6KEQCAWi6HcDUbeOfBIptF7tcCzusKFjFw2yuvEpDL9wQ=="
  crossorigin=""
/>
```



```

    api.get('items').then(response => {
      setItems(response.data);
    });
  }, []);

// para pegar as UFs da api do IBGE
useEffect(() => {
  axios.get<IBGEUFResponse[]>('https://servicodados.ibge.gov.br/api/v1/localidades/estados?orderBy=nome')
    .then(response => {
      const ufInitials = response.data.map(uf => uf.sigla);
      setUfs(ufInitials);
    })
  }, []);

// para pegar as cidades da api do IBGE sempre que o usuário selecionar uma UF
useEffect(() => {
  // para evitar que execute quando for renderizado da primeira vez (uf = 0)
  if (selectedUf === '0') {
    return;
  }

  axios.get<IBGECityResponse[]>(`https://servicodados.ibge.gov.br/api/v1/localidades/estados/${selectedUf}/distritos?or
    .then(response => {
      const cityNames = response.data.map(city => city.nome);
      setCities(cityNames);
    })
  }, [selectedUf]);

```



Sempre que usar **state** com um **array** ou **objeto**, deve-se **informar o tipo da variável**.

4. Mobile

▼ Configurando projeto

- Instalando o Expo

```
npm install -g expo-cli
```

- Criando projeto "mobile"

```
expo init mobile
```

- Executando projeto

```
npm start
```

▼ Outras instalações

- Instalando fontes

```
expo install expo-font @expo-google-fonts/ubuntu @expo-google-fonts/roboto
```

- Instalando bibliotecas para navegação/botões

```

npm install @react-navigation/native
expo install react-native-gesture-handler react-native-reanimated react-native-screens react-native-safe-area-context @
npm install @react-navigation/stack

```

- Instalando biblioteca para usar mapas

```
expo install react-native-maps
```

- Para permitir adicionar arquivos .svg ao projeto


```
expo install react-native-svg
```

- Para fazer requisições HTTP

```
npm install axios
```

- Para pegar a localização atual do usuário

```
expo install expo-location
```

- Para permitir abrir um app de e-mail automaticamente a partir do app

```
expo install expo-mail-composer
```

- Para usar Inputs Select no React Native

```
npm install react-native-picker-select
```

5. Funcionalidades Avançadas

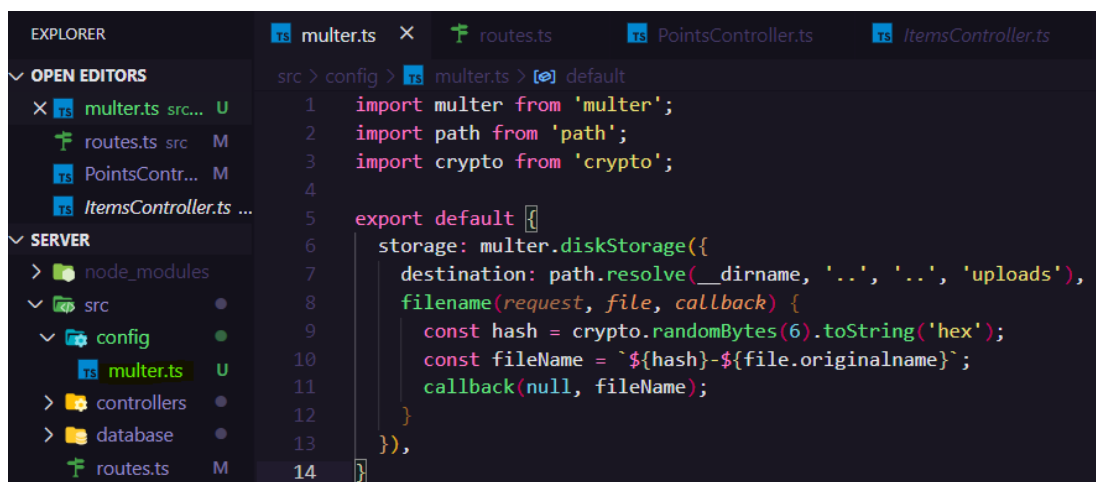
▼ Upload de imagens

▼ Back-end

- Instalar a biblioteca Multer para upload de imagens

```
npm install multer
npm install @types/multer -D
```

- Criar o arquivo **multer.ts** dentro da pasta **config**, onde ficarão as configurações do Multer. Aqui é configurado o diretório para onde irão os arquivos, além de ser gerado um novo nome para o arquivo usando uma chave Hash.



- Adicionar o Multer ao **routes.ts**, importando ele e criando a variável upload, que receberá essa configuração do Multer. Passar essa variável upload à rota em que ocorrerá o upload de imagens (o método **single** é usado porque é passado um único arquivo; o parâmetro passado é o nome da variável).

```
routes.ts x PointsController.ts ItemsController.ts
src > routes.ts > default
1 import express from 'express';
2 import multer from 'multer';
3
4 import multerConfig from './config/multer';
5
6 import PointsController from './controllers/PointsController';
7 import ItemsController from './controllers/ItemsController';
8
9 const routes = express.Router();
10
11 const upload = multer(multerConfig);
12
13 const pointsController = new PointsController();
14 const itemsController = new ItemsController();
15
16 routes.get('/items', itemsController.index);
17
18 routes.post('/points', upload.single('image'), pointsController.create);
19
20 routes.get('/points', pointsController.index);
21
22 routes.get('/points/:id', pointsController.show);
23
24 export default routes;
```

- Nos métodos **show** e **index**, retornar o JSON com o endereço da imagem fazendo uma **serialização**.



Serialização de dados: transformar os dados que vem do banco de dados para uma visualização mais acessível, adequada ao front-end da aplicação.

```
PointsController.ts x
src > controllers > PointsController.ts > PointsController > show
5 async index(request: Request, response: Response) {
6   // filtros de cidade, uf, items => Query Params
7   const { city, uf, items } = request.query;
8
9   // pega os itens enviados (formato 1,2,3) e converte em um array, dividindo pe
10  // e pegando cada item sem espaços e convertido em número
11  const parsedItems = String(items)
12    .split(',')
13    .map(item => Number(item.trim()));
14
15  const points = await knex('points')
16    .join('point_items', 'points.id', '=', 'point_items.point_id')
17    .whereIn('point_items.item_id', parsedItems)
18    .where('city', String(city))
19    .where('uf', String(uf))
20    .distinct()
21    .select('points.*');
22
23  // serialização de dados: transformar os dados para torná-los mais acessíveis,
24  const serializedPoints = points.map(point => {
25    return {
26      ...point,
27      image_url: `http://192.168.2.102:3333/uploads/${point.image}`,
28    };
29  });
30}
```

- No método **create**, pegar somente o nome do arquivo (**filename**) da imagem, usando a requisição HTTP recebida (**request**)

```
async create(request: Request, response: Response) {
  // aqui é usada a desestruturação do JS
  // cada um é igual a ex.: const name = request.body.name
  const {
    name,
    email,
    whatsapp,
    latitude,
    longitude,
    city,
    uf,
    items
  } = request.body;

  // usando uma transaction com o knex
  const trx = await knex.transaction();

  // aqui é usada short syntax, porque as variáveis são iguais aos nomes
  // cada um é igual a ex.: name: name
  const point = {
    image: request.file.filename,
    name,
    email,
    whatsapp,
    latitude,
    longitude,
    city,
    uf,
    items
  };

  // aqui é usada a short syntax, porque as variáveis são iguais aos nomes
  // cada um é igual a ex.: name: name
  const point = {
    image: request.file.filename,
    name,
    email,
    whatsapp,
    latitude,
    longitude,
    city,
    uf,
    items
  };

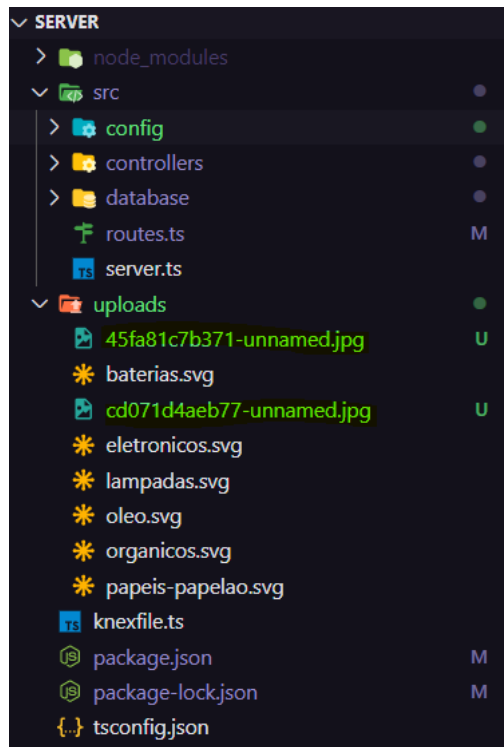
  // aqui é usada a short syntax, porque as variáveis são iguais aos nomes
  // cada um é igual a ex.: name: name
  const point = {
    image: request.file.filename,
    name,
    email,
    whatsapp,
    latitude,
    longitude,
    city,
    uf,
    items
  };
}
```

- Para testar no **Insomnia**, usar o **Multipart Form** (somente com ele é possível enviar arquivos)

Field	Value
name	Mercado do seu Zé
email	contato@imperatriz.com.br
whatsapp	31986250620
latitude	-46.8448484849
longitude	-35.838388333
city	Rio do Sul
uf	SC
items	1, 2, 6
image	unnamed.jpg

```
{
  "id": 9,
  "image": "45fa81c7b371-unnamed.jpg",
  "name": "Mercado do seu Zé",
  "email": "contato@imperatriz.com.br",
  "whatsapp": "31986250620",
  "latitude": "-46.8448484849",
  "longitude": "-35.838388333",
  "city": "Rio do Sul",
  "uf": "SC"
}
```

- As imagens vão para a pasta upload da aplicação, com os seus nomes seguindo o padrão estabelecido no código (hash-nomedaimagem)



▼ Front-end

- Instalar a biblioteca Dropzone (<https://react-dropzone.js.org/>)

```
npm install react-dropzone
```

- Criar um componente Dropzone

```
import React, {useCallback, useState} from 'react'
import {useDropzone} from 'react-dropzone'
import { FiUpload } from 'react-icons/fi';

import './styles.css';

interface Props {
  onFileUploaded: (file: File) => void;
}

const Dropzone: React.FC<Props> = ({ onFileUploaded }) => {
  const [selectedFileUrl, setSelectedFileUrl] = useState('');

  // após o usuário enviar o arquivo, pega a url do arquivo e a salva em selectedFileUrl
  const onDrop = useCallback(acceptedFiles => {
    const file = acceptedFiles[0];
    const fileUrl = URL.createObjectURL(file);
    setSelectedFileUrl(fileUrl);

    onFileUploaded(file);
  }, [onFileUploaded])

  const {getRootProps, getInputProps, isDragActive} = useDropzone({
    onDrop,
    accept: 'image/*'
  })

  return (
    <div className="dropzone" {...getRootProps()}>
      <input {...getInputProps()} accept="image/*" />
      { /* se o selectedFileUrl existir (imagem cadastrada), exibe a imagem; senão, exibe as mensagens */ }
      { selectedFileUrl
        ? <img src={selectedFileUrl} alt="Point thumbnail" />
        : (
            isDragActive ?
            <p>Solte o arquivo aqui...</p> :
            <p><FiUpload />Arraste e solte um arquivo aqui ou clique para selecionar um arquivo</p>
          )
      }
    </div>
  )
}
```

```

    }

    </div>
  )
}

export default Dropzone;

```

- Na página do formulário, adicionar o estado da imagem

```
const [selectedFile, setSelectedFile] = useState<File>();
```

- No formulário, adicionar o Dropzone, passando como props a função setSelectedFile da página de formulário para pegar o arquivo enviado ao componente Dropzone (comunicação pai ⇒ filho via props e filho ⇒ pai via função)

```

<p>Imagem do estabelecimento</p>
<Dropzone onFileUploaded={setSelectedFile} />

```

- Alterar o método onSubmit para que os dados sejam enviados via **Multipart FormData** (permite envio de arquivos)

```

async function handleSubmit(event: FormEvent) {
  // para evitar que a página recarregue depois de enviar o form
  event.preventDefault();

  const { name, email, whatsapp } = formData;
  const uf = selectedUf;
  const city = selectedCity;
  const [latitude, longitude] = selectedPosition;
  const items = selectedItems;

  // deve ser feito com FormData para permitir o envio de arquivos
  const data = new FormData();

  data.append('name', name);
  data.append('email', email);
  data.append('whatsapp', whatsapp);
  data.append('uf', uf);
  data.append('city', city);
  data.append('latitude', String(latitude));
  data.append('longitude', String(longitude));
  data.append('items', items.join(','));

  if (selectedFile) {
    data.append('image', selectedFile);
  }

  await api.post('points', data);

  alert('Cadastro realizado com sucesso!');

  // voltar para tela inicial
  history.push('/');
}

```

▼ Validação de dados (back-end)

```

npm install celebrate
npm install @types/hapi__joi -D

```

▼ Validação de dados (front-end)

- Usar a biblioteca **Yup** (<https://github.com/jquense/yup>)

▼ Deploy

▼ Back-end com NodeJS:

- Alternativas
 - **Heroku**: <https://www.youtube.com/watch?v=-j7vLmBMsEU&t=21s>

- **Digital Ocean:** <https://www.youtube.com/watch?v=IClz5dE3Xfg>
- Outras para projetos maiores: Amazon Web Services, Google Cloud, Microsoft Azure

▼ **Front-end com ReactJS e outros frameworks JavaScript:**

- **Netlify:** <https://www.netlify.com>
- **Vercel**
- **Para projetos maiores:** Amazon S3, Google Cloud Storage

▼ **Apps com React Native:**

- <https://www.youtube.com/watch?v=wYMvzbfBdYI>

Outros:

- **!!! Site para arte gratuita, em qualquer cor, para o seu projeto:** <https://undraw.co/>
- **Site para montagem de esquemas, flowcharts, mindmaps:** <https://whimsical.com>
- **Link para o protótipo do projeto:** <https://www.figma.com/file/1SxgOMojOB2zYT0Mdk28IB/Ecoleta>