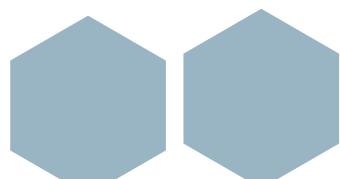


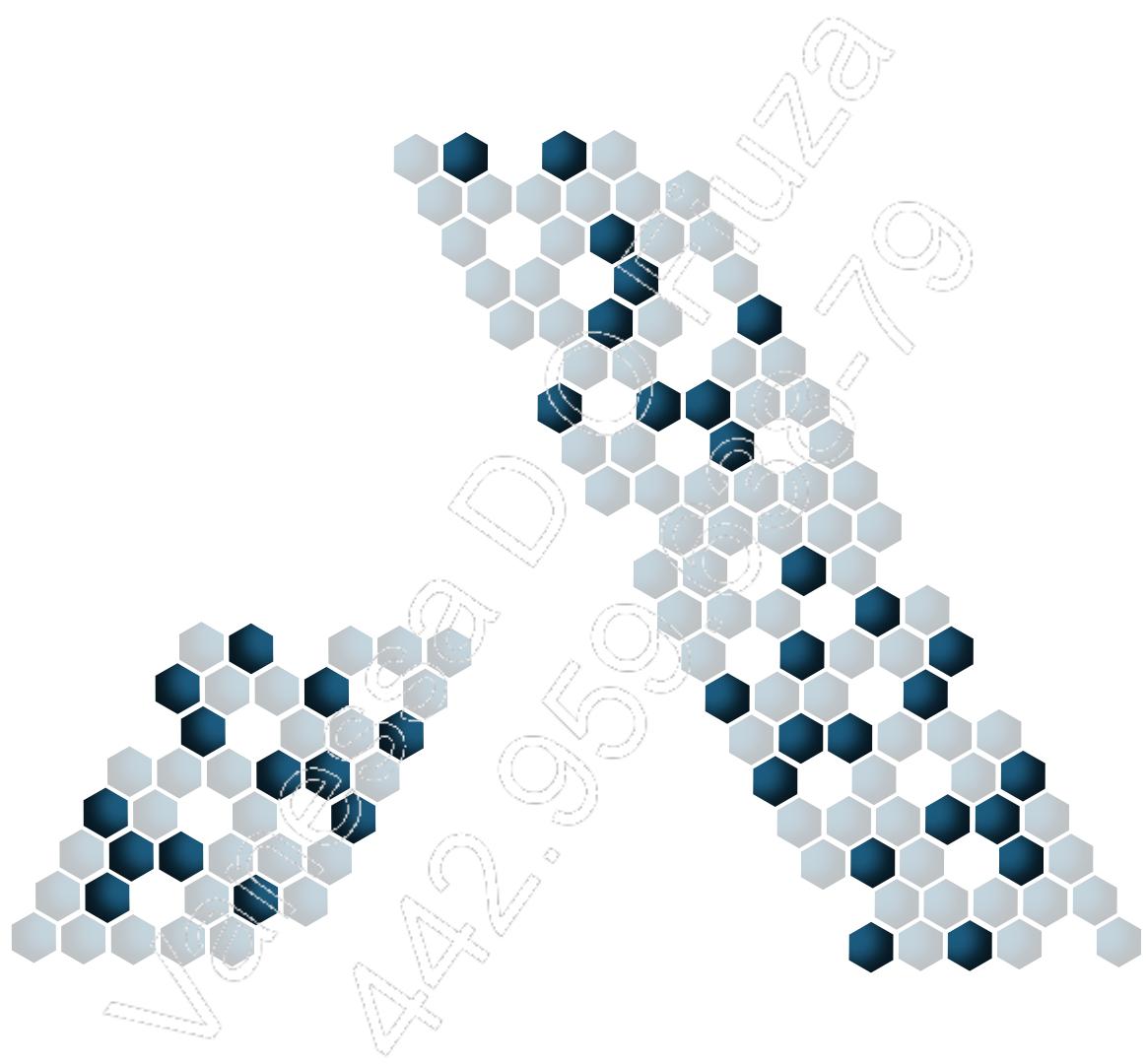


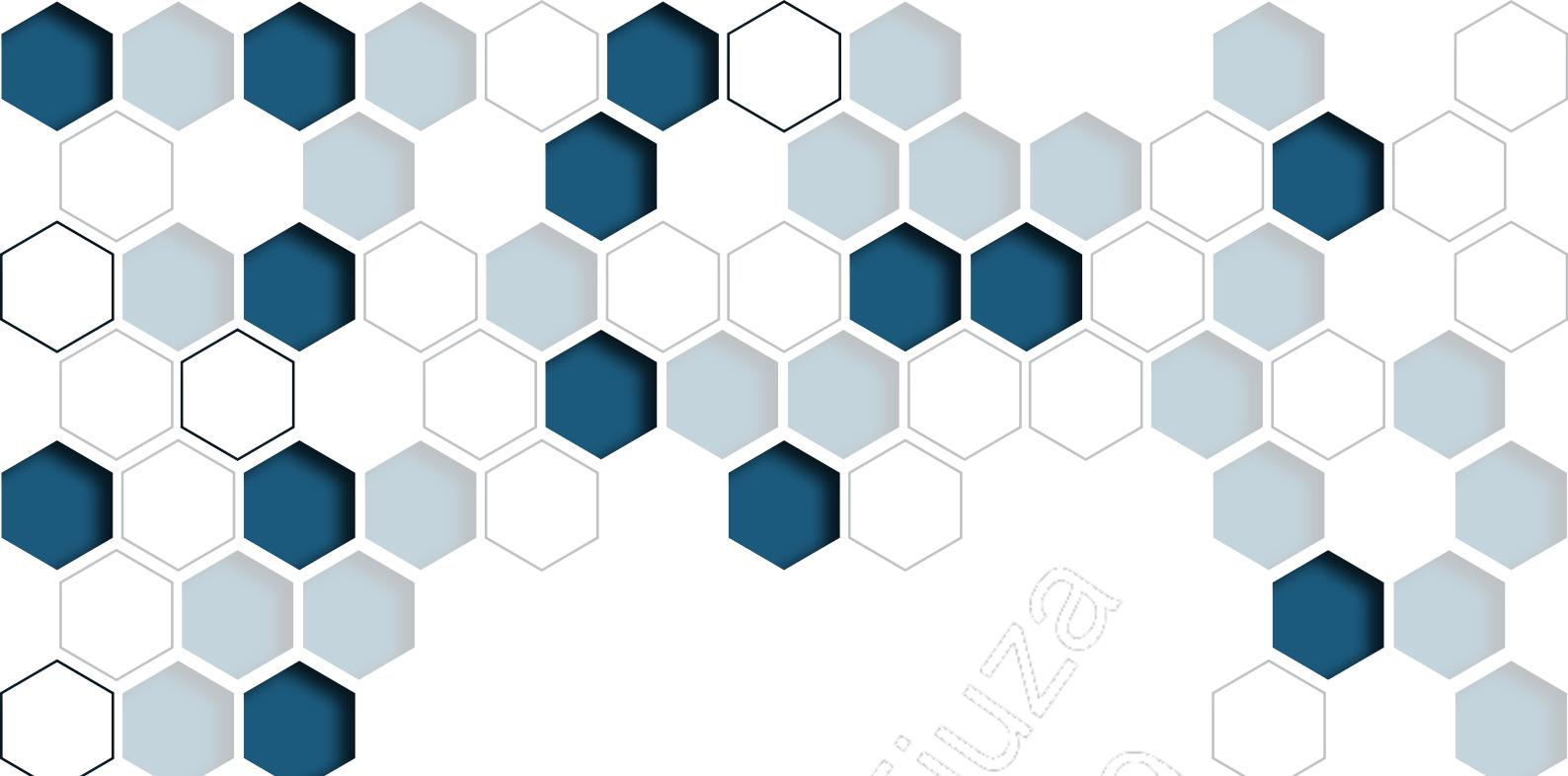
# Desenvolvimento de Componentes Web e Mobile com React.Js, Redux e React Native



Editora  
**IMPACTA**







# Desenvolvimento de Componentes Web e Mobile com React.Js, Redux e React Native



Editora  
**IMPACTA**



## Créditos

---

Copyright © Monte Everest Participações e Empreendimentos Ltda.

Todos os direitos autorais reservados. Este manual não pode ser copiado, fotocopiado, reproduzido, traduzido ou convertido em qualquer forma eletrônica, ou legível por qualquer meio, em parte ou no todo, sem a aprovação prévia, por escrito, da Monte Everest Participações e Empreendimentos Ltda., estando o contrafator sujeito a responder por crime de Violação de Direito Autoral, conforme o art.184 do Código Penal Brasileiro, além de responder por Perdas e Danos. Todos os logotipos e marcas utilizados neste material pertencem às suas respectivas empresas.

*"As marcas registradas e os nomes comerciais citados nesta obra, mesmo que não sejam assim identificados, pertencem aos seus respectivos proprietários nos termos das leis, convenções e diretrizes nacionais e internacionais."*

# Desenvolvimento de Componentes Web e Mobile com React.Js, Redux e React Native

## Coordenação Geral

Marcia M. Rosa

## Coordenação Editorial

Henrique Thomaz Bruscagin

## Autoria

Emilio Celso de Souza

## Revisão Ortográfica e Gramatical

Fernanda Monteiro Laneri

## Diagramação

Bruno de Oliveira Santos

**Edição nº 1 | 1851\_1**

Setembro/2018

# Sumário

---

<b>Conteúdo de Referência .....</b>	<b>07</b>
<b>Introdução .....</b>	<b>11</b>
1. Visão geral do Node.js e do npm .....	13
2. Acesso a dados .....	23
3. Conceitos do React.js .....	37
4. Desenvolvimento Web .....	63
5. Redux .....	67
6. React Native .....	77
<b>Projeto .....</b>	<b>85</b>
1. Preparando o ambiente .....	87
2. Criação do banco de dados e do Web service .....	89
3. Configuração do webpack .....	97
4. Aplicação do React.js .....	103
5. Aplicação do Redux .....	131
6. Aplicação do React Native .....	143
<b>Mãos à obra! .....</b>	<b>155</b>
1. Criando um projeto com React.js .....	157
2. Criando um projeto com Redux .....	159
3. Criando um projeto com React Native .....	161
<b>Anotações .....</b>	<b>163</b>



# Conteúdo Programático

## Conteúdo programático - 1/6

### 1 – Visão geral do Node.js e do npm

- Definição
- Iniciando com o Node.js
- Iniciando com o npm
- Definição de módulos
- Definição de servidor Web
- Conceitos do ExpressJS
- O arquivo package.json
- Instalação de dependências

Trainamentos  
**IMPACTA**

## Conteúdo programático - 2/6

### 2 – Acesso a dados

- Acesso a dados com MongoDB
- Usando o Mongoose
- Criação de serviços REST

Trainamentos  
**IMPACTA**

## Conteúdo programático - 3/6

### 3 – Conceitos do React.js

- Programação com ECMAScript usando Babel
- Configuração do webpack
- Definição e configuração do React.js
- Desenvolvimento de componentes
- Atributos dos componentes
- Estado e propriedades dos componentes
- Rotas
- Ciclo de vida dos componentes



FINANCIAMENTO

## Conteúdo programático - 4/6

### 4 – Desenvolvimento Web

- Formulários
- Componentes de formulários



DESENVOLVIMENTO  
WEB

## Conteúdo programático - 5/6

### 5 – Redux

- Implementação e configuração do Redux
- Integração do React com o Redux
- O Middleware Redux: Redux-Promise, Redux-Multi e Redux-Thunk

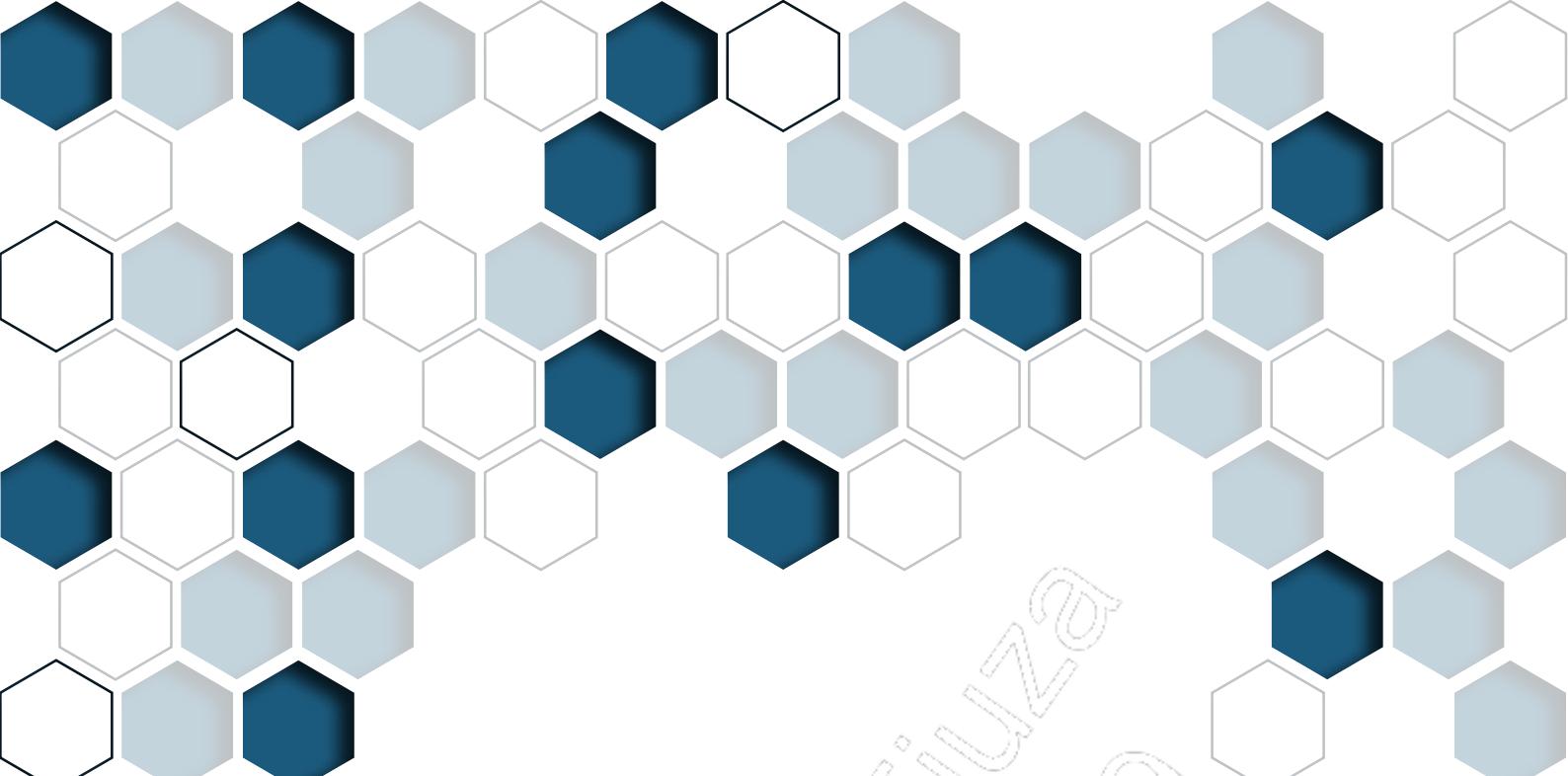
Trainamentos  
**IMPACTA**

## Conteúdo programático - 6/6

### 6 – React Native

- Conceitos do React Native
- Uso do aplicativo Expo
- Desenvolvimento de aplicações
- Aplicação na plataforma nativa
- Aplicação no simulador

Trainamentos  
**IMPACTA**



# Introdução



Wainess & 95°  
442-1950.

## Introdução

Neste treinamento, aplicaremos, de forma prática, os conceitos associados ao desenvolvimento de componentes com o framework **React.js**.

Esses componentes podem ser utilizados em aplicações Web ou em aplicativos mobile. Para aplicativos, usamos uma vertente do React.js conhecida como **React Native**.

Conheceremos, também, os fundamentos do **Redux**, uma ferramenta do React.js que possibilita concentrar grande parte das informações em um único local, melhorando sua performance e manutenção.



## Introdução

Ao longo do curso, estudaremos cada uma dessas ferramentas e como elas interagem para compor uma aplicação real.

Os códigos desenvolvidos ao longo deste curso podem ser escritos em qualquer editor de textos, mas utilizaremos o **Visual Studio Code** (também conhecido como **VSCode**).

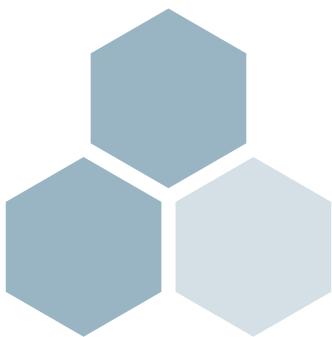
Podemos obtê-lo em <https://code.visualstudio.com/>.





1

# Visão geral do Node.js e do npm



## Definição

Em poucas palavras, **Node.js** (ou **NodeJS**) é uma plataforma de software que permite a criação de um Web server e o desenvolvimento de aplicações Web capazes de serem executadas pelo servidor criado pelo próprio Node.js.

O Node.js é composto por módulos, e um dos módulos disponíveis é o módulo HTTP. Esse módulo permite a criação do servidor, sem a necessidade de ferramentas adicionais, como é o caso do PHP, por exemplo, que necessita do servidor Apache, ou do ASP.NET, que é executado sobre o IIS.

## Iniciando com o Node.js

O primeiro passo para trabalhar com o Node.js é obtê-lo e instalá-lo:

- Acesse o link <<https://nodejs.org/en/>>;
- Obtenha a versão mais recente e instale-a.
- Para testar, execute no prompt de comandos:

```
node -v
```



## Iniciando com o npm

- Verifique o número da versão;
- Juntamente com o Node.js, é instalado o **npm** (node package manager). Para testá-lo, execute:

```
npm -v
```



## Definição de módulos no Node.js

A grande força do Node.js está nos módulos. É por meio deles que podemos criar novas aplicações, como aplicações Web com ExpressJS ou aplicações SPA com Angular 4.

O padrão utilizado no carregamento dos módulos é chamado **CommonJS**.

Um módulo é, na verdade, um código JavaScript. Para exemplificar, vamos definir três arquivos: dois representando os módulos e o terceiro representando a aplicação, utilizando o módulo em questão.



## Definição de módulos no Node.js

- Arquivo mod1.js:

```
module.exports = function (x) {  
    console.log(x);  
}
```

- Arquivo mod2.js:

```
exports.mensagem = function (x) {  
    console.log(x);  
}
```

IMPACTA

## Definição de módulos no Node.js

Carregando os módulos:

- Arquivo app.js:

```
var m1 = require('./mod1');  
var m2 = require('./mod2');  
  
m1('Carregando uma única função modular');  
m2.mensagem('Carregando objeto com funções modulares');
```

O símbolo ./ indica o mesmo diretório. Para testar, execute:

```
node app.js
```

IMPACTA

## Definição de servidor Web

Da mesma forma que podemos definir nossos próprios módulos, existem diversos módulos prontos para serem usados. Para ilustrar, vamos utilizar dois módulos: **http** e **fs** (fs se refere a file system):



## Definição de servidor Web

- Arquivo `app_server.js`:

```
var http = require('http');
var requisicao = function (request, response) {
    response.writeHead(200, { "Content-Type": "text/html" });
    response.write("<h1>Texto a ser exibido no browser</h1>");
    response.end();
}

var server = http.createServer(requisicao);
var resultado = function () {
    console.log('Servidor em funcionamento!');
}
server.listen(3000, resultado);
```



## Definição de servidor Web

Vamos analisar o código anterior:

1. Obtemos uma referência ao módulo http:

```
var http = require('http');
```

2. Definimos uma função com dois parâmetros (requisição e resposta):

```
var requisicao = function (request, response) {
  response.writeHead(200, { "Content-Type": "text/html" });
  response.write("<h1>Texto a ser exibido no browser</h1>");
  response.end();
}
```

## Definição de servidor Web

3. Definimos uma variável responsável pela conexão com o servidor.

Observe que a função `createServer()` espera uma função callback contendo dois parâmetros, que são configurados adequadamente:

```
var server = http.createServer(requisicao);
```

4. Essa função é opcional, porém importante. Ela será executada quando a conexão for estabelecida e o servidor estiver no ar:

```
var resultado = function () {
  console.log('Servidor em funcionamento!');
}
```

5. Finalmente, definimos o servidor atendendo a requisições na porta 3000:

```
server.listen(3000, resultado);
```

## Definição de servidor Web

Para executar:

1. No prompt, execute:

```
node app_server.js
```

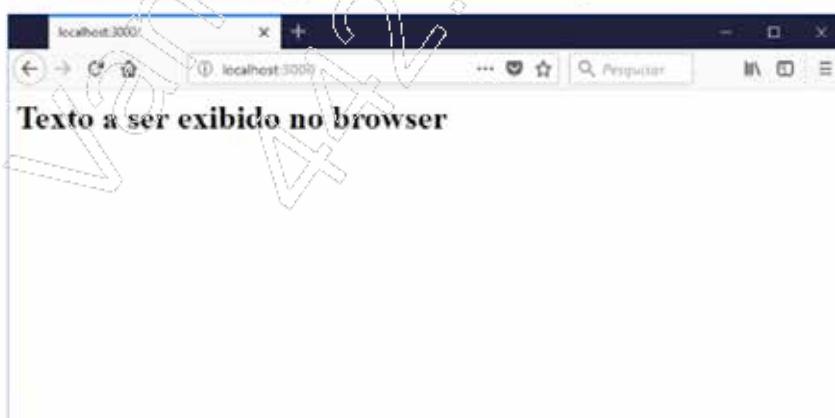
2. Quando surgir a mensagem **Servidor em Funcionamento**, o servidor já estará no ar. Abra um browser da sua escolha e digite a url:

```
localhost:3000
```

Treinamentos  
**IMPACTA**

## Definição de servidor Web

O resultado deverá ser algo semelhante ao apresentado abaixo:



Treinamentos  
**IMPACTA**

## Definição de servidor Web

O exemplo anterior considera um conteúdo HTML escrito na própria função referenciada pela variável **requisicao**. Se for necessária uma página mais elaborada, esse procedimento se tornará inviável.

## Conceitos do ExpressJS

**Express** (ou **ExpressJS**) é um framework para desenvolvimento Web baseado no **Node.js**.

Podemos dizer que, com exceção do banco de dados MongoDB, é possível obtermos uma aplicação completa baseada na arquitetura **MVC** usando o **Node** e o **Express**.

O Express, assim como qualquer outro framework baseado no **Node.js**, é composto por módulos. Cada módulo especifica uma funcionalidade da aplicação. Os módulos são incluídos na aplicação por meio do gerenciador de módulos, o **npm**, e o npm instala os módulos com base nas informações do arquivo **package.json**.

## O arquivo package.json

A seguir, apresentaremos um exemplo do arquivo package.json:

```
{  
  "name": "app-name",  
  "version": "0.0.1",  
  "private": true,  
  "dependencies": {  
    "body-parser": "~1.15.1",  
    "cookie-parser": "~1.4.3",  
    "debug": "~2.2.0",  
    "ejs": "~2.4.1",  
    "express": "~4.13.4",  
    "morgan": "~1.7.0",  
    "serve-favicon": "~2.3.0",  
  }  
}
```

Metadados da aplicação

Dependências



## O arquivo package.json

Observe que, na parte de dependências, que é o que o Node instala de fato no projeto, existem versões com símbolos especiais na frente do número da versão, como ~ ou ^. Esses símbolos possuem significados especiais, que serão descritos a seguir:



## O arquivo package.json

**versão:** exatamente a versão indicada  
**~versão:** aproximadamente  
**>= versão:** igual ou maior que  
**> versão:** maior que  
**<= versão:** igual ou menor que  
**< versão:** menor que  
**^versão:** compatível com  
**1.2.x:** 1.2.0, 1.2.1, 1.2.2,..., mas não 1.3.0

A documentação do npm pode ser obtida em <<https://docs.npmjs.com/>>.

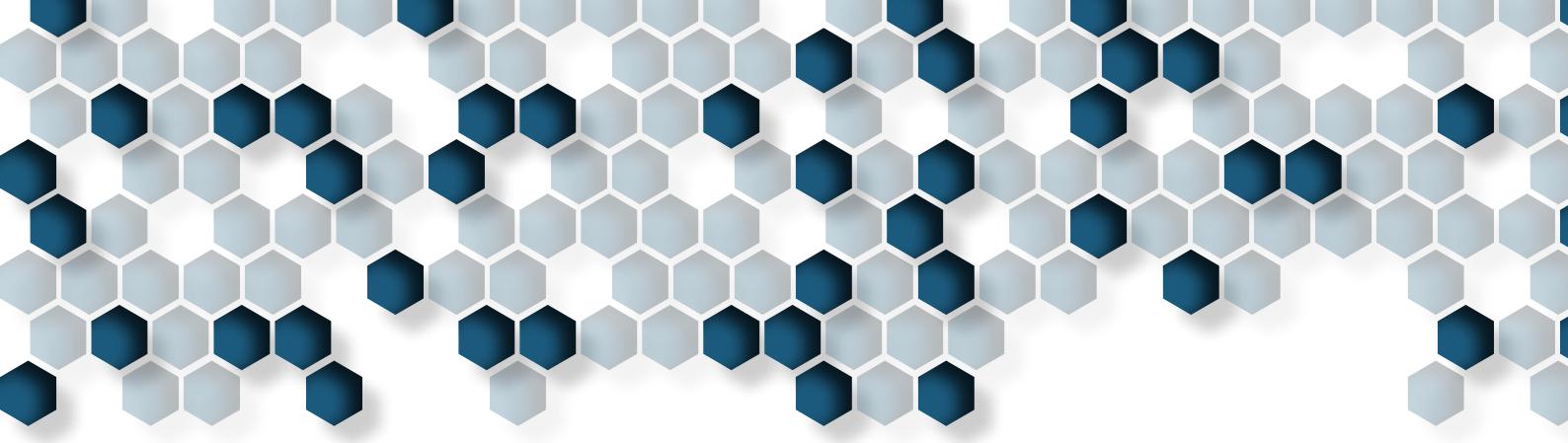
## Instalação de dependências

O arquivo **package.json** fica localizado na raiz da pasta do projeto. A instalação das dependências requeridas neste arquivo é feita por meio do comando:

`npm install`

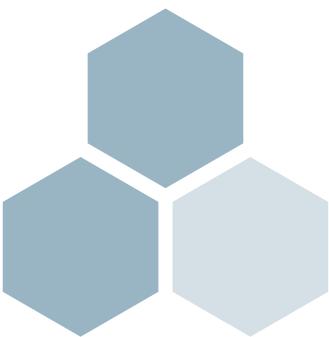
Esse procedimento instrui o npm a baixar todas as dependências e instalá-las em uma pasta chamada **node\_modules**.

**Elaborar o projeto 01**



2

# Acesso a dados



Editora  
**IMPACTA**

## Acesso a dados com MongoDB

MongoDB é baseado em **documento** (e não composto por tabelas, como é o caso de bancos relacionais).

Este formato não considera linhas e colunas; apenas linhas. Cada linha é um elemento BSON (Binary JSON – um JSON serializado) .

IMPACTA

## Acesso a dados com MongoDB

O primeiro passo é instalar o banco de dados. Ele pode ser obtido em <https://www.mongodb.com/>.

Por padrão (no Windows), os bancos de dados ficam armazenados na pasta C:\data\db. Devemos criar essa pasta antes de iniciar.

É possível definir outra pasta, desde que, ao iniciar o serviço, a especifiquemos.

Para iniciar o serviço, devemos digitar o comando abaixo na linha de comandos:

mongod

IMPACTA

## Instalando o banco de dados

Para iniciar o servidor referenciando uma pasta diferente, devemos executar o comando:

```
mongod --dbpath <caminho escolhido>
```

Exemplo:

```
mongod --dbpath D:\Projetos\mongodb\dados
```

Para o comando acima funcionar corretamente, é necessário que exista a pasta **D:\Projetos\mongodb\dados**.

Por padrão, o MongoDB é executado na porta **27017**.



## Criando e acessando o banco de dados

Como servidor em execução, abrimos outra instância do terminal e executamos o comando:

```
mongo
```

Após essa execução, teremos um terminal cliente para executar instruções pertinentes ao banco de dados, como criar e manipular bancos de dados. O terminal do MongoDB possui o símbolo **>**.

Vamos executar algumas instruções e testar o banco de dados instalado.



## Criando e acessando o banco de dados

1. Exibindo todos os bancos de dados:

```
>show databases  
OU  
>show dbs
```

2. Criando um banco de dados: Na verdade, nós executamos o mesmo comando para criar e para tornar um banco de dados ativo. Se ele não existir, será criado:

```
>use db exemplo
```

O banco de dados **db exemplo** se tornará ativo, mas não existe de fato.

Trilamentos  
**IMPACTA**

## Criando e acessando o banco de dados

3. Listando as coleções:

```
>show collections
```

Nada será mostrado, uma vez que o banco de dados ativo não possui nenhum documento. Quando criarmos um documento ou coleção, este será criado.

4. Inserindo um novo documento no banco:

```
>db.clientes.insert({ nome: "Impacta", telefone : "3254-2200" })
```

Após a execução dessa instrução, o banco de dados é criado, juntamente com a coleção **clientes**. Execute **show collections** para visualizar!

Trilamentos  
**IMPACTA**

## Criando e acessando o banco de dados

5. Listando os documentos:

```
>db.clientes.find()
```

Exibe os documentos contidos na coleção `clientes`.

6. Listando os documentos de forma estruturada:

```
>db.clientes.find().pretty()
```

7. Atualizando um documento:

```
>db.clientes.update({nome: "Impacta"},{url: "impacta.com.br"})
```



## Criando e acessando o banco de dados

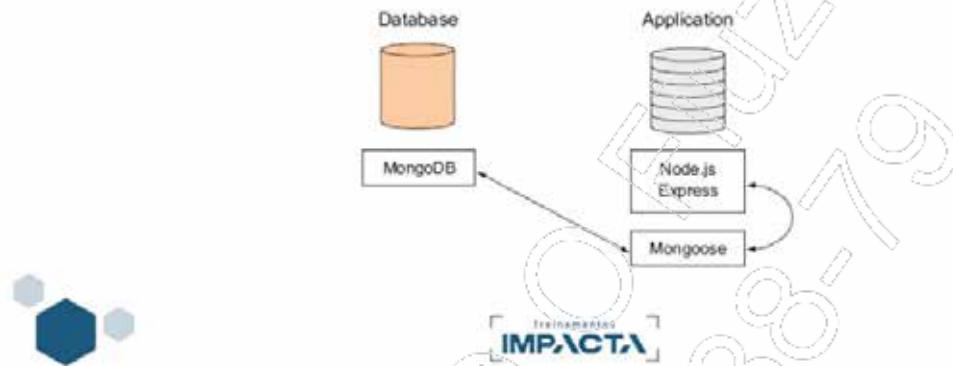
Substitui o atributo `nome` (quando encontrado) pelo atributo `url`.

Esses são alguns exemplos de utilização do banco de dados. Nos próximos passos veremos como manipular o banco de dados por meio de uma aplicação baseada no Express.



## Usando o Mongoose

O Mongoose é um módulo do Node.js que facilita as tarefas de acesso ao banco de dados. Em vez de acessá-lo diretamente, o Mongoose representa uma camada que abstrai toda a complexidade pertinente ao banco. Seu mecanismo é representado na imagem abaixo:



## Usando o Mongoose

Tendo o MongoDB instalado e o serviço iniciado, o Mongoose é capaz de criar o banco de dados, caso não exista. Não há necessidade de interferir diretamente no banco de dados.

Para instalar o Mongoose na aplicação, usamos o comando:

```
npm install --save mongoose
```

Lembrando que o atributo **--save** inclui o módulo no arquivo **package.json** para futuras instalações, quando necessário.

## Usando o Mongoose com Models

O elemento central no desenvolvimento de um banco de dados é modelo, ou **Schema**, um tipo especial de código JavaScript onde definimos os elementos no banco de dados.

A forma geral para definição de um elemento no documento é:

```
elemento: {type: Tipo [, parametros adicionais]}
```

Exemplo:

```
nome: {type: String}
```



## Usando o Mongoose com Models

Os tipos usados na definição de um schema são:

**String:** Qualquer cadeia de caracteres no padrão UTF-8.

**Number:** Qualquer tipo numérico. Não temos como especificar se é inteiro ou double, apenas Number.

**Date:** Retornado como ISODate no MongoDB.

**Boolean:** Retorna True ou False.



## Usando o Mongoose com Models

Os tipos usados na definição de um schema são:

**Buffer:** Representa informações binárias, como imagens, por exemplo.

**Mixed:** Qualquer tipo de dado.

**Array:** Uma coleção de elementos

**ObjectId:** Qualquer identificador diferente de `_id`. Normalmente usado para referenciar o valor do `_id` em outros documentos.



## Usando o Mongoose com Models

O exemplo a seguir ilustra a definição de um schema:

```
var cliente = new mongoose.Schema({  
    nome: String,  
    endereco: String,  
    avaliacao: Number,  
    atividades: [String]  
});
```

Ou



## Usando o Mongoose com Models

```
var Schema = require('mongoose').Schema;

var cliente = Schema({
  nome: String,
  endereco: String,
  avaliacao: Number,
  atividades: [String] //array de String
});
```



## Usando o Mongoose com Models

É possível adicionarmos subdocumentos a um documento. Exemplo:

```
var schemaAtividade = new mongoose.Schema({
  descricao: { type: String },
  duracao: Number
});

var cliente = Schema({
  nome: { type: String, required: true },
  endereco: String,
  avaliacao: { type: Number, "default": 0, min: 0, max: 5, },
  atividades: [schemaAtividade] //array de SchemaAtividade
});
```



## Usando o Mongoose com Models

Na definição de subdocumentos, é importante que a definição do subdocumento ocorra antes do documento principal que o utiliza.

Trilamentos  
**IMPACTA**

## Criando serviços REST

Para definirmos serviços REST, devemos utilizar o módulo **express**, disponibilizado a partir do npm. É conveniente que o projeto pertinente aos serviços REST não contemplem os recursos do React, pois são componentes diferentes.

Podemos considerar um arquivo chamado **db.js** com o conteúdo:

```
const mongoose = require("mongoose");
mongoose.Promise = global.Promise;
module.exports = mongoose.connect('mongodb://localhost:27017/escola');
```

Trilamentos  
**IMPACTA**

## Criando serviços REST

Definir, também, o componente responsável por colocar o servidor proprietário do serviço no ar, baseado no express (arquivo **server.js**):

```
const port = 3200;

const bodyParser = require('body-parser');
const express = require('express');
const server = express(); //novo servidor
const allowCors = require('./cors');

server.use(bodyParser.urlencoded({ extended: true }));
server.use(bodyParser.json());
server.use(allowCors);
```



## Criando serviços REST

```
server.listen(port, function () {
  console.log(`servidor no ar, na porta ${port}`);
});

module.exports = server;
```



## Criando serviços REST

Levando em conta que o modelo considerado seja o Cliente apontado anteriormente, vamos definir as opções de verbos HTTP responsáveis pelo acesso ao serviço. Este código será escrito no arquivo **service.js**:

```
Clientes.methods(['get', 'post', 'put', 'delete']);  
Clientes.updateOptions({ new: true, runValidators: true });  
  
module.exports = Clientes;
```

## Criando serviços REST

O próximo passo é definir as rotas disponíveis para acesso ao modelo por meio do serviço. Vamos apresentar nosso código no arquivo **rotas.js**:

```
const express = require('express');  
  
module.exports = function (server) {  
  const router = express.Router();  
  server.use('/ws', router);  
  
  const Clientes = require('./clientes');  
  
  Clientes.register(router, '/clientes');  
};
```

## Criando serviços REST

Teremos oportunidade de aplicar esses conceitos durante o desenvolvimento do projeto.

Elaborar o projeto 02



treinamentos  
**IMPACTA**

Vanessa D'Fiuna  
442.950.6379

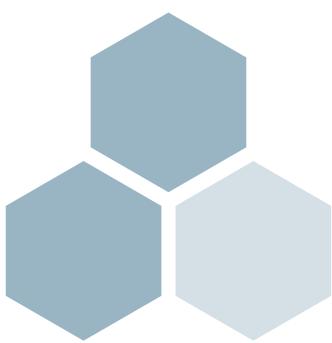




3

# Conceitos do React.js

Valueless  
442.950.  
Fluza  
19



## Introdução

O React, também chamado **React.js**, é uma biblioteca front-end destinada ao desenvolvimento de componentes para Web ou para dispositivos móveis. Ela foi desenvolvida pelo Facebook e disponibilizada para a comunidade.

O propósito é criar componentes reutilizáveis, cujo conteúdo é baseado no HTML e no CSS. No caso do React Native, existe uma biblioteca complementar responsável por gerar elementos para diferentes plataformas, como Android e iOS.

## Programação com ES6 usando Babel

ES6 é também chamado de ECMAScript 6 ou ECMAScript 2015.

O ES6 é a mais nova versão do JavaScript. O nome oficial do JavaScript é ES6 ou ES2015, como alguns gostam de chamar.

Ao longo do curso, usaremos diversas instruções utilizando o ES6.

Mais informações a respeito da sintaxe do ES6 podem ser encontradas em <[https://www.w3schools.com/js/js\\_es6.asp](https://www.w3schools.com/js/js_es6.asp)>.



## Programação com ES6 usando Babel

O React.js faz uso extensivo do ES6, e seus componentes são executados por meio do Node.js. O problema é que o Node.js não suporta (ainda) todas as funcionalidades do ES6.

O componente Babel (Babel.js) é um compilador baseado no JavaScript que permite transformar um código com as especificações do ES6 em uma estrutura compatível com o Node.js.

Existem diversos componentes que podem ser usados no projeto. Os módulos do Babel são adicionados ao projeto por meio do npm.

Veremos sua utilização no desenvolvimento do projeto.



## Configuração do webpack

webpack é um módulo responsável por agregar módulos do JavaScript, bem com suas dependências, em um arquivo JavaScript possível de ser interpretado pelo browser.

Em outras palavras, o webpack realiza um “bundle” com os componentes desenvolvidos, especialmente no projeto React.

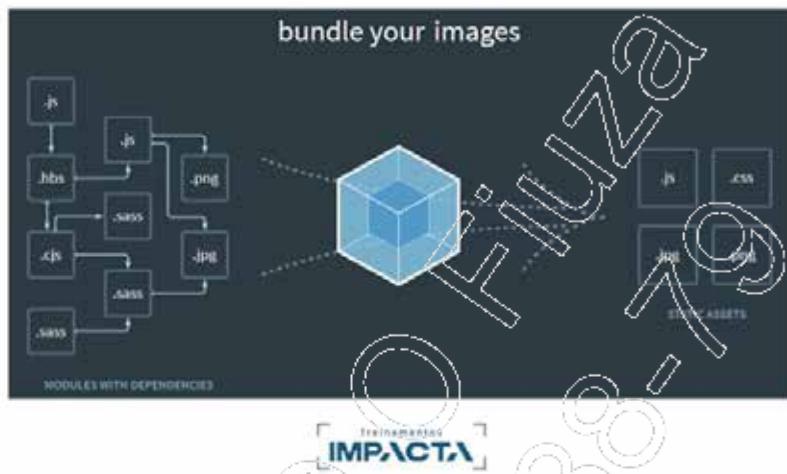
No site a seguir podemos ver o funcionamento do webpack:

<<https://webpack.js.org>>.



## Configuração do webpack

Um esboço obtido neste link é apresentado a seguir:



## Configuração do webpack

Em um projeto React.js, as configurações do webpack são realizadas no arquivo `webpack.config.js`.

A instalação do webpack também é realizada por meio do `npm`. Os principais módulos são:

`webpack`  
`webpack-dev-server`  
`webpack-cli`

## Configuração do webpack

Podemos obter mais informações a respeito dos elementos de configuração do webpack no link <<https://webpack.js.org/concepts/>>.

Assim como o Babel, realizaremos algumas configurações do webpack ao longo da elaboração dos exercícios e do projeto.



## Definição e configuração do React.js

Vamos agora conhecer um projeto baseado no React, desde suas configurações até a execução de um componente.

Em uma pasta destinada ao projeto de exemplo (**cursoReact**, por exemplo), vamos executar a instrução:

```
npm init -yes
```

Esse comando cria o arquivo **package.json** na pasta. O parâmetro **-yes** é usado para apresentar a resposta “sim” a todas as perguntas durante a criação do arquivo.



## Definição e configuração do React.js

Vamos abrir o VSCode apontando para esta pasta.

Precisamos instalar as dependências do Babel e do webpack, além das dependências do próprio React. No prompt de comando (onde criamos o arquivo package.json), execute os comandos:

- **Babel:**

```
npm i -g babel  
npm i -g babel-cli  
npm i --save babel-core babel-loader babel-preset-react  
babel-preset-es2015
```



## Definição e configuração do React.js

No prompt de comandos, entre na pasta que você escolheu para o projeto (se ainda não o fez). Instale as dependências do webpack:

- **webpack:**

```
npm i --save webpack  
npm i --save webpack-dev-server  
npm i --save webpack-cli
```



## Definição e configuração do React.js

Instale as dependências do React:

- React:

```
npm i --save react react-dom
```



FUNÇÃO



DESENVOLVIMENTO



DESENVOLVIMENTO

## Definição e configuração do React.js

Esses são os módulos necessários, mas não todos que poderemos precisar em todos os projetos. Os módulos são adicionados sob demanda.

No arquivo `package.json`, faça a alteração no objeto `scripts`:

```
"start": "webpack-dev-server --hot"
```

Esse script será usado para iniciar a aplicação no prompt de comandos. Mas, para isso, devemos definir o arquivo `webpack.config.js`.



## Definição e configuração do React.js

Este arquivo deverá ser criado na raiz do projeto. Seu conteúdo deverá ser:

```
var config = {  
    entry: './main.js',  
    output: {  
        path: '/public',  
        filename: 'index.js',  
    },  
    devServer: {  
        inline: true,  
        port: 8080  
    },  
};
```

 IMPACTA

## Definição e configuração do React.js

```
module: {  
    rules: [  
        {  
            test: /\.jsx?$/,  
            exclude: /node_modules/,  
            loader: 'babel-loader',  
            query: {  
                presets: ['es2015', 'react']  
            }  
        }  
    ]  
}  
}  
module.exports = config;
```

 IMPACTA

## Definição e configuração do React.js

É importante constantemente consultar a documentação pois o conteúdo desse arquivo pode sofrer alterações na medida em que as suas versões evoluem.

Crie a pasta **public** no projeto, representando o local de saída do webpack, direcionado para o arquivo **index.js**.



## Definição e configuração do React.js

Crie os seguintes arquivos (na raiz do projeto):

- **main.js**;
- **index.html**;
- **App.jsx**.



## Definição e configuração do React.js

- index.html

Representa a página a ser executada de fato. Observe o elemento div com id=app no seu conteúdo. É nesse elemento que o React incluirá seu resultado, durante a execução:

## Definição e configuração do React.js

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>React App</title>
</head>
<body>
    <div id="app"></div>
    <script src="index.js"></script>
</body>
</html>
```

## Definição e configuração do React.js

- App.jsx

É o componente React que desenvolvemos. Um componente com conteúdo HTML normalmente recebe a extensão `.jsx`, embora possa ser `.js` também. É sempre recomendado seguirmos a convenção sugerida.

Este arquivo é o responsável por criar o elemento UI a ser reaproveitado.

## Definição e configuração do React.js

```
import React, { Component } from 'react';

export default class App extends Component{
  render() {
    return (
      <div>
        Introdução ao React.
      </div>
    )
  }
}
```



## Definição e configuração do React.js

- **main.js**

É o arquivo indicado como ponto de entrada no webpack. Ele renderizará no elemento **div** do **index.html** o conteúdo do componente **App.jsx**:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App />, document.getElementById('app'));
```

Observe que o componente **App.jsx** entra como uma tag a ser traduzida pelo React.

 IMPACTA

## Definição e configuração do React.js

Para verificar o resultado, execute o comando **npm start** no prompt de comandos e observe a compilação por meio do webpack.

Se nenhum erro for apresentado, abra um browser e digite:  
**localhost:8080**

**Elaborar o projeto 03**

 IMPACTA

## Definição de componentes

Criar um componente no React equivale a criar uma classe com certas funcionalidades. Como vimos no primeiro exemplo, a classe **App** é um componente, cujo retorno é um elemento `div` com um texto.

Vamos fazer uma abordagem a respeito da definição do componente e das formas como eles podem ser acessados.

## Definição de componentes – Exemplo 1

Considere o componente a seguir, definido no arquivo **Comp1.jsx**:

```
import React, { Component } from 'react';

export default class Componente1 extends Component {
  render() {
    return <h1>Primeiro Componente</h1>
  }
}
```



## Definição de componentes – Exemplo 1

Sua utilização em main.js, sem remover o que já existe lá, fica:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';
import Comp01 from './Comp1.jsx';

ReactDOM.render(
  <div>
    <App />
    <Comp01 />
  </div>
, document.getElementById('app'));
```

## Definição de componentes – Exemplo 1

Todo componente deve ser exportado para ser utilizado no seu destino (no nosso exemplo, no arquivo **main.js**).

Observe que usamos **export default**, o que indica que, independente do nome da classe, podemos importá-lo atribuindo o nome que desejarmos:

```
export default class Componente1
```

- Em **main.js**:

```
import Comp01 from './Comp1.jsx';
```

## Definição de componentes – Exemplo 1

Comp01 não é nem o nome do arquivo, nem da classe, mas o nome que usamos no nosso destino. Isso foi possível porque usamos o comando **default** na definição da classe. É comum no React desenvolvemos um arquivo para cada componente, e, em cada arquivo, pode haver diversos itens, como constantes, funções, entre outros, mas o que está sendo disponibilizado para consumo é o componente principal, exportado de forma default.



## Definição de componentes – Exemplo 1

Quando uma classe representa um componente, ela deve estender a classe **Component** e obrigatoriamente deve implementar a função **render()**.

Esta função é a responsável por entregar o componente propriamente dito.

E se a classe não utilizar o comando **default**?



## Definição de componentes – Exemplo 2

Considere agora a classe a seguir:

```
import React, { Component } from 'react';

export class Componente2 extends Component {
    render() {
        return <h1>Segundo Componente</h1>
    }
}
```

Ela não utiliza o comando **default**. Neste caso, sua utilização ocorre da seguinte forma:

## Definição de componentes – Exemplo 2

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';
import Comp01 from './Comp1.jsx';
import { Componente2 } from './Comp2.jsx';

ReactDOM.render(
    <div>
        <App />
        <Comp01 />
        <Componente2 />
    </div>
, document.getElementById('app'));
```

## Definição de componentes – Exemplo 2

Ou seja, é necessário usar o próprio nome definido para a classe, entre chaves.

Outro ponto a considerarmos é que, quando mais de um componente é renderizado no destino, devemos envolvê-los por um componente mais abrangente. No nosso exemplo, usando o elemento `div`:

## Definição de componentes – Exemplo 2

```
ReactDOM.render(  
  <div>  
    <App />  
    <Comp01 />  
    <Componente2 />  
  </div>  
, document.getElementById('app'));
```

A função `ReactDOM.render()` renderiza, na verdade, um único elemento, cujo conteúdo pode conter outros elementos filhos.



## Atributos dos componentes

Quando for necessário usarmos atributos nos componentes que devem ser renderizados, algumas alterações devem ser consideradas.

A utilização de classes CSS, por exemplo. Considere o arquivo **estilos.css** a seguir, definido na raiz do projeto:

```
.fonte {  
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;  
    color: blue;  
}
```

## Atributos dos componentes

O local adequado para que ele seja referenciado é no arquivo **index.html**:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <title>React App</title>  
    <link href="estilos.css" rel="stylesheet">  
</head>  
<body>  
    <div id="app"></div>  
    <script src="index.js"></script>  
</body>  
</html>
```

## Atributos dos componentes

A utilização desta classe no componente `App.jsx`, por exemplo, deve usar o atributo `className` em vez de `class`, uma vez que a palavra `class` tem outro significado no React:

```
import React, { Component } from 'react';

export default class App extends Component{
    render() {
        return (
            <div className="fonte">
                Introdução ao React.
            </div>
        )
    }
}
```



## Conteúdo dinâmico

É possível utilizarmos conteúdo dinâmico nos componentes, inclusive expressões condicionais com base no operador ternário. Exemplo:

```
export default class App extends Component{
    render() {
        var curso = 'React';
        return (
            <div className="fonte">
                Introdução ao {curso == 'React' ? 'React.js' :
                    'Javascript'}.
            </div>
        )
    }
}
```



## Estilos para componentes

Para definirmos estilos individualmente para os componentes, podemos criar localmente objetos com as propriedades dos estilos. Para medidas, não colocamos unidades como px, por exemplo. O React considera este valor:

```
export class Componente3 extends Component {  
    render() {  
  
        var estilos = {  
            fontSize: 50,  
            color: '#FF0000'  
        }  
        return <h1 style={estilos}>Informações do componente</h1>  
    }  
}
```

## Estado e propriedade dos componentes

Uma das tarefas mais importantes dos componentes é armazenar e transmitir informações, especialmente quando essas informações são provenientes de fontes externas. O armazenamento das informações se dá por meio de uma propriedade chamada **state**, definida no construtor da classe, e a recuperação de valores trocados entre os componentes por meio do objeto **props**.

Vamos considerar o exemplo a seguir:

## Estado e propriedade dos componentes

```
export default class EstadoComponente extends Component {  
  constructor() {  
    super();  
  
    this.state = {  
      escola: 'Impacta',  
      cursos: ['React', 'MEAN', 'Excel']  
    }  
  }  
}
```



FIUNA  
TODA  
MUNDO  
CRIAR  
IDEIAS

## Estado e propriedade dos componentes

```
render() {  
  return(  
    <div>  
      <h1>Escola: {this.state.escola}</h1>  
      <h2>Cursos:</h2>  
      <div>  
        {this.state.cursos.map((curso, i) =>  
          <Lista key={i} info={curso} />)  
        </div>  
      </div>  
    );  
  }  
}
```



VINHEDO  
SANTOS  
SÃO PAULO  
BRAZIL  
IMPACTA

## Estado e propriedade dos componentes

```
class Lista extends React.Component {  
    render() {  
        return (  
            <ul>  
                <li>{this.props.info}</li>  
            </ul>  
        )  
    }  
}
```



## Estado e propriedade dos componentes

Pelo fato da classe estender **Component**, a propriedade **state** é herdada na nossa classe, assim como **props**.

O construtor define o estado como sendo um objeto com duas propriedades: **escola** e **cursos**.

Essas propriedades podem ser acessadas de qualquer ponto por meio da instrução **this.state.escola** ou **this.state.cursos**.



## Estado e propriedade dos componentes

- Na parte da instrução...

```
<Lista key={i} info={curso} />
```

...o atributo **info** é recebido na classe **Lista** por meio de **props**:

```
class Lista extends React.Component {  
    render() {  
        return (  
            <ul>  
                <li>{this.props.info}</li>  
            </ul>  
        );  
    }  
}
```

## Rotas

O mecanismo de rotas é o meio pelo qual conseguimos acessar outros componentes a partir do componente atual. Normalmente uma rota é acionada por meio de links.

Para trabalharmos com rotas, devemos seguir estas etapas:

- Inclua o módulo **react-router-dom**:

```
npm install --save react-router-dom
```



## Rotas

- Considerando que queiramos uma rota apontando para /exemplo, devemos implementá-la da seguinte forma:

```
export default props => (  
  
  <Switch>  
    <Route path='/exemplo' component={Exemplo} />  
  </Switch>  
)
```

Trilamentos IMPACTA

## Rotas

- No arquivo que apresenta a rota (app.jsx, por exemplo), inclua os elementos:

```
<Link to='/exemplo'>Exemplos</Link>
```

Trilamentos IMPACTA

## Métodos do ciclo de vida do React.js

Durante a execução de um componente, é imprescindível que certas operações ocorram no momento propício, sem que tenhamos que chamar o método explicitamente.

Os métodos do ciclo de vida do React.js servem para esta finalidade. Os principais métodos disponíveis são:

- **componentWillMount**: Executado antes da renderização do componente.
- **componentDidMount**: Executado após a primeira renderização somente no lado do cliente. É neste método que requisições AJAX ou atualizações de estados de componentes devem ocorrer.



## Métodos do ciclo de vida do React.js

- **componentWillReceiveProps**: Executado assim que props é executado e antes da próxima renderização do componente.
- **shouldComponentUpdate**: Determina se o componente será ou não atualizado. Seu retorno é `true` ou `false`.
- **componentWillUpdate**: Chamado imediatamente antes da renderização.
- **componentDidUpdate**: Chamado imediatamente após a renderização.
- **componentWillUnmount**: Chamado após o componente ser removido do DOM.



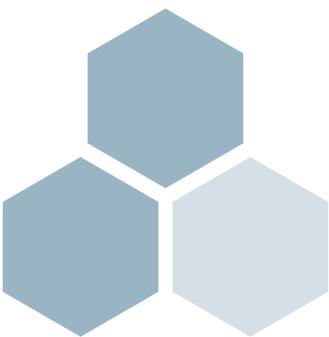




4

# Desenvolvimento Web

Varrioss@ 442.950.679  
Fiuza 79



## Formulários

Muitos componentes devem apresentar recursos para interagir com o usuário, por meio de campos de entrada, de seleção, ou botões para executar alguma ação.

Neste sentido, permitir que um componente ofereça essas funcionalidades é parte fundamental do seu desenvolvimento.

O que torna um componente React atraente é sua capacidade de se integrar com outras partes da aplicação por meio do objeto **props**.

Vamos apresentar os recursos usados no desenvolvimento de componentes contendo elementos de formulários.

 IMPACTA

## Componentes de formulários

Consideremos o exemplo a seguir:

```
export default class App extends Component{  
  constructor(props){  
    super(props);  
    this.state = {  
      texto : 'Text Inicial'  
    }  
    this.textoChange = this.textoChange.bind(this);  
  }  
  
  textoChange(e) {  
    this.setState({ texto: e.target.value})  
  }  
}
```

 IMPACTA

## Componentes de formulários

```
render() {  
    return (  
        <div>  
            <input type="text" value={this.state.texto}  
                onChange={this.textoChange} />  
            <h4>State: {this.state.nome}</h4>  
        </div>  
    )  
}
```



## Componentes de formulários

Quando um texto é informado na caixa de textos, a função **textoChange** é chamada, atualizando o estado do componente, no sentido de alterar a propriedade **texto** presente no **state**.

A propriedade alterada é vinculada ao conteúdo da caixa de textos, e seu valor é exibido na tela.

O uso da instrução...

```
this.textoChange = this.textoChange.bind(this);
```

...é necessário para garantir que, mesmo desacoplando uma função de um objeto, o seu comportamento continue o mesmo.



## Componentes de formulários

Elaborar o projeto 04

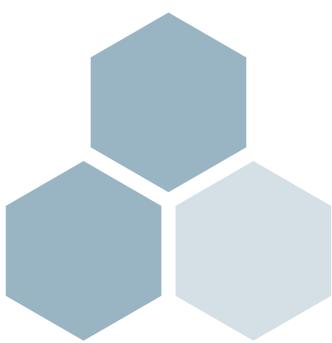




5

# Redux

Variables  
442.950.638-79  
FIUZA



## Implementação e configuração do Redux

Redux é uma ferramenta que permite o gerenciamento do estado dos dados em conformidade com o estado dos componentes.

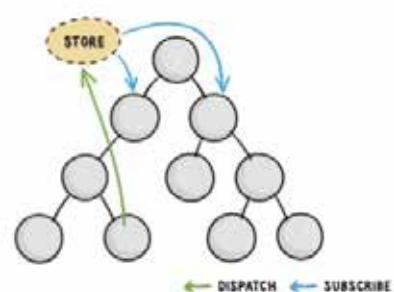
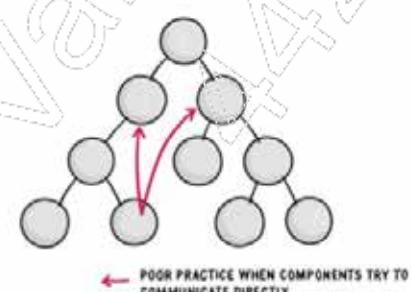
O objetivo do Redux é concentrar em um único local, conhecido como **store**, todos os estados de todos os componentes da aplicação, em vez de permitir que os componentes troquem informações entre si.

O Redux é útil em projetos com muitos componentes, onde o gerenciamento de estado entre eles se torna complexo.

## Implementação e configuração do Redux

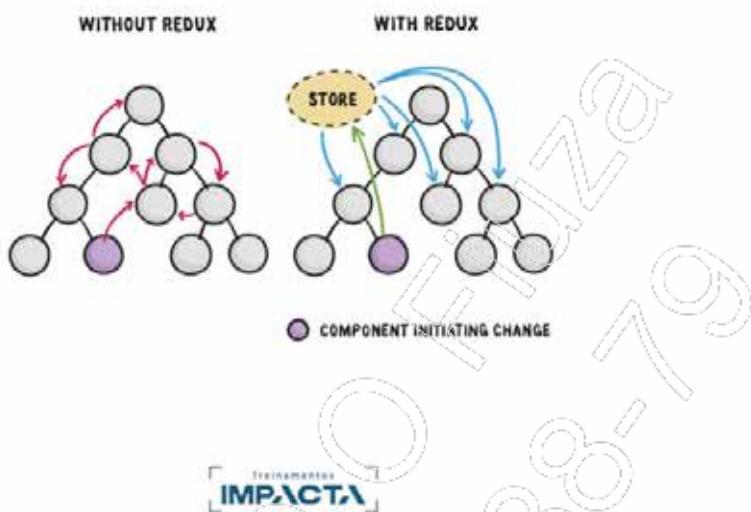
As imagens a seguir resumem a utilização do Redux:

Fonte: <https://css-tricks.com/learning-redux/>



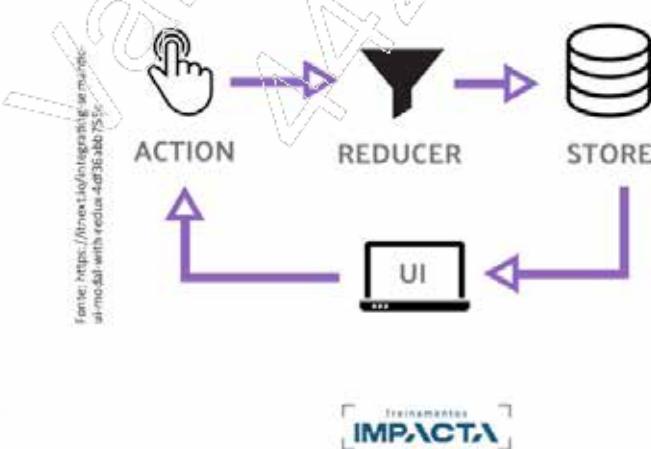
## Implementação e configuração do Redux

Fonte: <https://css-tricks.com/learning-react-with-redux/>



## Implementação e configuração do Redux

O fluxo de implementação de uma aplicação baseada em Redux consiste nos elementos apresentados na imagem:



## Implementação e configuração do Redux

Elementos de uma aplicação Redux:

- **Actions:** Fontes de informações enviadas da aplicação para o Store, por meio de funções conhecidas como **Action Creators**.
- **Reducers:** Responsáveis por receberem as informações e encaminharem para a Store.
- **Store:** Local onde são centralizados os estados da aplicação. A Store é determinada como Single source of truth (Única fonte de verdade).
- **React-redux:** Conexão usada na aplicação responsável por conectar os dados da aplicação ao store por meio do Redux.



## Integração do React com o Redux

Vamos apresentar um exemplo que elucida a implementação do Redux:

- **actions.js:** Analisa o valor da solicitação (type) e cria um objeto com o valor proveniente do evento.

```
export function mudarValor(e) {
  return {
    type: 'VALOR_ALTERADO',
    info: e.target.value
  }
}
```



## Integração do React com o Redux

Vamos apresentar um exemplo que elucida a implementação do Redux:

- **reducers.js**: Retorna o valor gerado no action, repassando-o para o componente.

```
const INITIAL_STATE = { valor: 'inicio' }

export default function (state = INITIAL_STATE, action) {
    switch (action.type) {
        case 'VALOR_ALTERADO':
            return { valor: action.info };
        default:
            return state;
    }
}
```

## Integração do React com o Redux

Vamos apresentar um exemplo que elucida a implementação do Redux:

- **index.jsx**: É o elemento que renderizará o conteúdo na Web. Ele toma o valor da propriedade **counter** do componente que gerou a interface gráfica (UI):

## Integração do React com o Redux

```
import reducer from './reducer';

const reducers = combineReducers({
  counter: reducer
})

ReactDOM.render(
  <Provider store={createStore(reducers)}>
    <Contador />
  </Provider>
, document.getElementById('app'))
```

IMPACTA

## Integração do React com o Redux

Vamos apresentar um exemplo que elucida a implementação do Redux:

- **contador.jsx**: Componente a partir do qual o estado enviado para o store por meio do props é gerado:

IMPACTA

## Integração do React com o Redux

```
import { connect } from "net";

const Contador = props => (
  <div>
    <input onChange={props.mudarValor}
           value={props.counter.valor} />
  </div>
)
```



## Integração do React com o Redux

```
const mapStateToProps = state => ({
  counter: state.counter
});
const mapDispatchToProps = dispatch =>
  bindActionCreators({ mudarValor }, dispatch);

export default connect(mapStateToProps,
  mapDispatchToProps)(Contador);
```



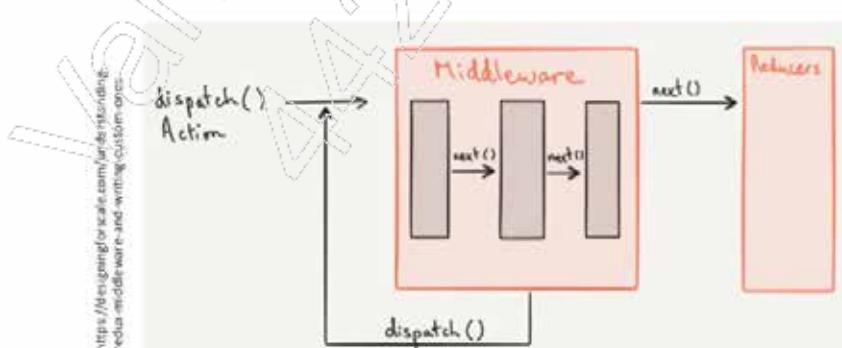
## Middlewares

Middlewares são funções chamadas no meio de um processo. Em outras palavras, são funções usadas para interceptar o fluxo natural de uma aplicação durante sua execução. No Redux temos três middlewares importantes:

- **redux-promise**: Intercepta vínculos executados assincronamente, durante o dispatch na conexão react-redux.
- **redux-multi**: Permite realizar o dispatch de múltiplos actions a partir de um action-creator.
- **redux-thunk**: Permite escrever action creator que retornem funções em vez de valores.

## Middlewares

A imagem a seguir ilustra o uso de middlewares:



## Middlewares

Elaborar o projeto 05



Editora  
**IMPACTA**





6

# React Native

Variables  
442:950.69  
Flujo  
19



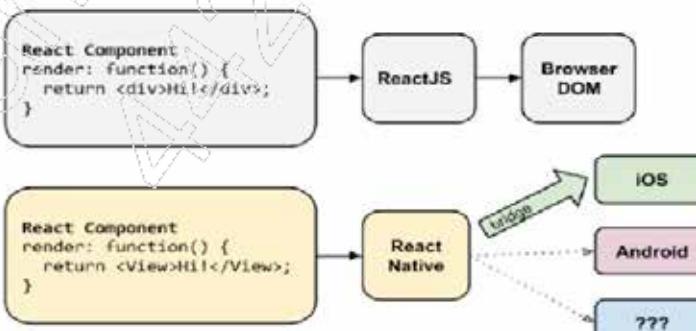
## Conceitos do React Native

Em janeiro de 2015, a equipe React anunciou um novo projeto: **React Native**. Esta nova abordagem usa a plataforma React, com destinos para aplicações nativas como Android e iOS, por meio da implementação de um tipo de "ponte" entre o JavaScript e a plataforma host.

O processo consiste em escrever aplicativos com a sintaxe do React, com seus componentes sendo renderizados para seus equivalentes na plataforma especificada.

Para exemplificar, considere a figura adiante:

## Conceitos do React Native



## Conceitos do React Native

Assim como escrevemos componentes HTML para serem renderizados por diferentes browsers, ou mesmo por uma aplicação desenvolvida em React, os componentes do React Native utilizam bibliotecas de componentes próprias para renderizar elementos nativos da plataforma.

Os componentes, bem como sua manipulação, estão em constantes alterações, e, por causa disso, é extremamente importante tomar como base a documentação oficial, disponível em <<http://facebook.github.io/react-native/>>.



## Uso do aplicativo Expo

No site da documentação do Facebook para o React Native, podemos encontrar informações a respeito de um aplicativo baseado no projeto Expo (mais detalhes em <<https://expo.io/>>). Ele disponibiliza um aplicativo (obtido na loja de aplicativos a partir do seu dispositivo) para testarmos a aplicação.

Para tanto, é necessário iniciarmos no terminal a aplicação por meio de um server, e o sincronismo com o aplicativo é obtido por meio do acesso ao server, desde que estejam na mesma rede.

No desenvolvimento do projeto, poderemos ver seu funcionamento.



## Desenvolvimento de aplicações

Podemos criar um projeto React Native de duas formas:

- Com a plataforma nativa
- Sem a plataforma nativa

IMPACTA

## Aplicação na plataforma nativa

Neste caso, é necessário que o ambiente de execução esteja devidamente configurado para gerar a aplicação. Por exemplo, para o Android, devemos ter o Android SDK devidamente instalado e configurado, o compilador do Java (geralmente na versão compatível com o SDK).

Os detalhes de criação do projeto são encontrados em <<https://facebook.github.io/react-native/docs/getting-started.html>>.

IMPACTA

## Aplicação na plataforma nativa

Necessitamos dos comandos:

```
npm install -g react-native-cli
```

E para criar um projeto:

```
react-native init nome_projeto
```



## Aplicação na plataforma nativa

A execução, como já mencionado, depende do ambiente de desenvolvimento configurado, pois os aplicativos são criados na plataforma real, sem uma camada WebView como geralmente acontece com outros frameworks híbridos.



## Aplicação no simulador Expo

Este é o modo mais tranquilo de criarmos e rodarmos uma aplicação, se o propósito é testá-la e só posteriormente transferi-la para o dispositivo real. É esta abordagem que usaremos nesta aula.

Primeiro, instalamos o gerador de aplicativo:

```
npm install -g create-react-native-app
```

Para criar o projeto, executamos o comando:

```
create-react-native-app appReactNative
```

## Aplicação no simulador Expo

Quando o projeto estiver criado, é importante testar o ambiente. Para testá-lo, vamos executar, na pasta do projeto:

```
npm start
```

Quando a aplicação for iniciada, um QR Code será gerado, o que permitirá, por meio do aplicativo Expo, sua leitura e, consequentemente, seu sincronismo.

Um exemplo de execução é mostrado a seguir:

## Aplicação no simulador Expo



## Aplicação no simulador Expo

Se houver discrepância no IP da rede, vale a pena executar o comando no prompt:

```
set REACT_NATIVE_PACKAGER_HOSTNAME=127.0.0.1
```

É claro que devemos incluir nosso próprio host no comando acima.

Uma vez criado o projeto, vamos analisar algumas partes.

## Aplicação no simulador Expo

Elaborar o projeto 06



## Laboratórios

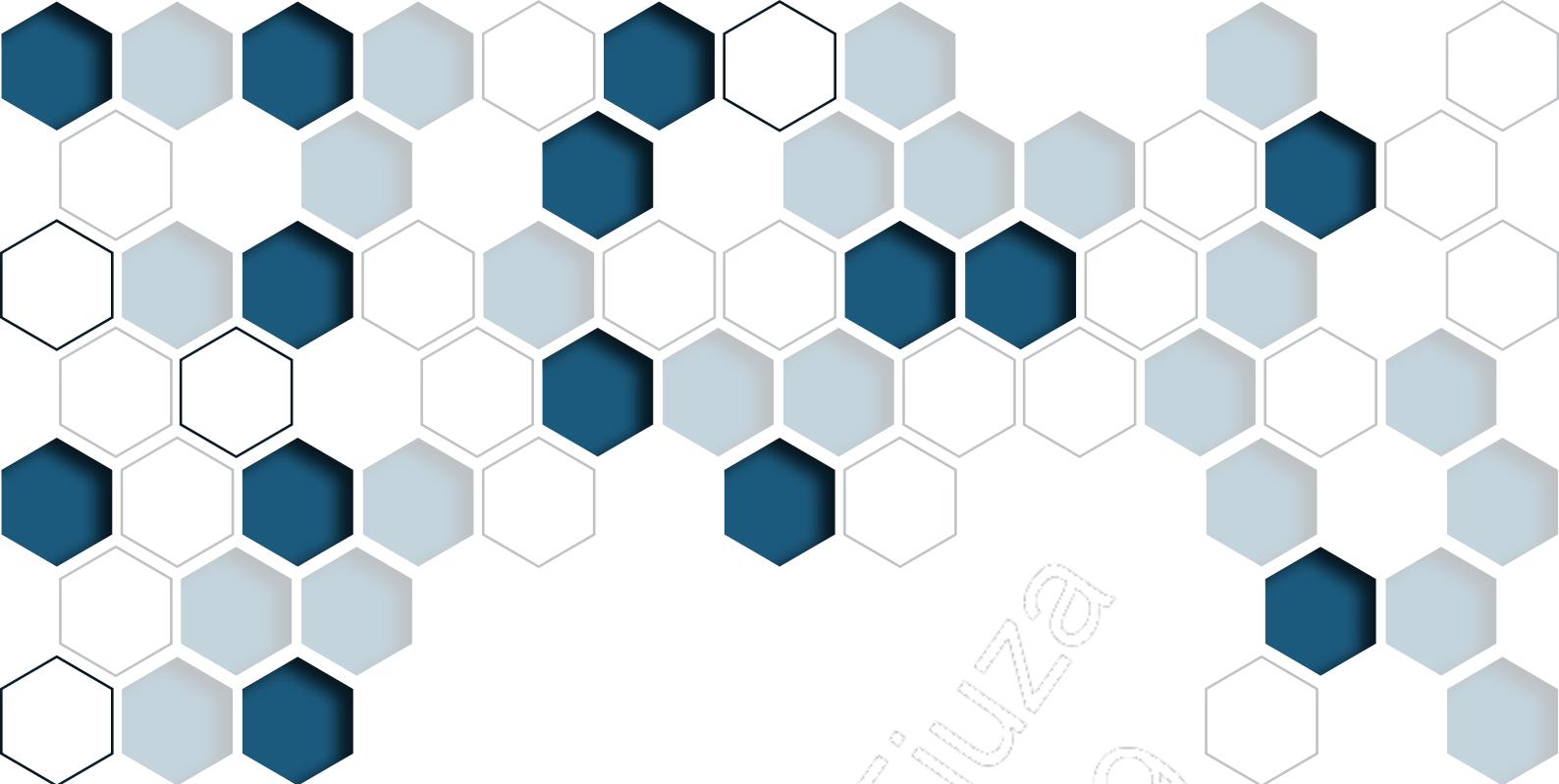
Elaborar os laboratórios:

Laboratório 01

Laboratório 02

Laboratório 03





# Projeto



Editora  
**IMPACTA**

## Apresentando o projeto

No decorrer deste curso, desenvolveremos um projeto contemplando os recursos do React, do Redux e do React Native. Neste projeto, utilizaremos o banco de dados MongoDB como fonte de dados para um Web service, que também será elaborado como parte integrante do projeto.

O projeto consiste em um sistema para cadastro e consulta de cursos para uma escola. Será possível, também, entrar em contato com a escola e solicitar mais informações. O objetivo é aplicar os conceitos estudados durante o treinamento.

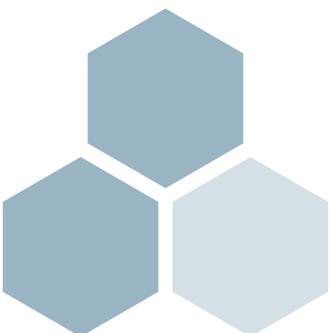
1. No projeto haverá um menu de opções para acesso a página inicial, cadastro e consulta de cursos, e contatos;
2. Um usuário devidamente autenticado poderá realizar o cadastro de cursos;
3. A visualização da lista de cursos será realizada por quaisquer alunos, assim como a possibilidade de entrar em contato com a escola;
4. A listagem dos cursos também poderá ser visualizada por meio de um aplicativo para celulares. Neste caso, apenas a visualização estará disponível, não o cadastro.

A aplicação deverá ter uma aparência agradável, com boa usabilidade e facilidade de navegação. O projeto será desenvolvido com o VSCode (Visual Studio Code), previamente instalado.

1

# Preparando o ambiente

Atividade



## A – Definindo a estrutura inicial para o projeto

1. Em um local de sua preferência (unidade C, pasta Documentos etc.), defina uma pasta chamada **cursoReact** e, dentro dela, duas novas pastas chamadas **cliente** e **server**;

A partir deste ponto, podemos adicionar, alterar ou remover pastas e arquivos para nossa aplicação.

2. Instale o Node.js e o banco de dados MongoDB (se ainda não estiverem instalados);
3. Adicione os diretórios de instalação no path do sistema operacional. As pastas serão parecidas com estas:

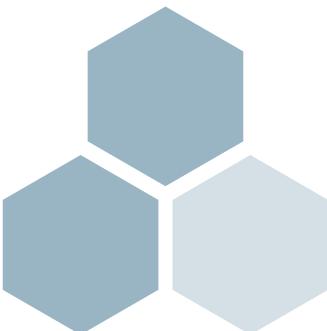
- **C:\Program Files\MongoDB\Server\3.6\bin;**
- **C:\Program Files\nodejs**



2

## Criação do banco de dados e do Web service

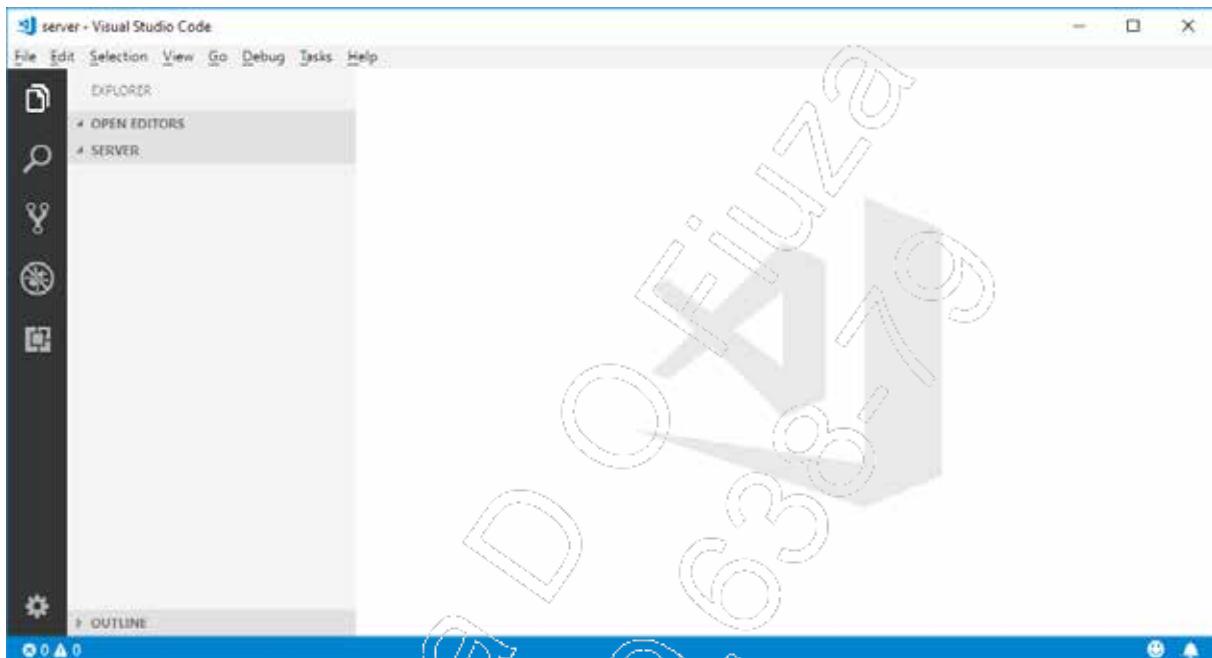
Atividade



## A – Criando o projeto para representar o servidor (server)

Definiremos um projeto que representará nosso servidor de aplicações, fornecendo um Web service a ser consumido pela nossa aplicação React. Vamos então seguir os passos adiante:

1. Abra o VSCode apontando para a pasta **server**, criada no laboratório anterior:



## B – Adicionando os módulos necessários

1. No prompt de comandos, ou no PowerShell disponibilizado pelo VSCode, aponte para a pasta **server** (a mesma aberta no VSCode);
2. Execute os comandos baseados no Node.js:

```
npm init -yes (ou npm init -y)
```

3. Os comandos anteriores criarão o arquivo **package.json** para este projeto. Abra-o no VSCode e realize as alterações sugeridas a seguir:

```
{
  "name": "controle-cursos",
  "version": "1.0.0",
  "description": "",
  "main": "src/loader.js",
  "scripts": {
    "dev": "nodemon",
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

# Criação do banco de dados e do Web service

4. Instale as dependências para o nosso projeto, no prompt de comandos:

```
npm i --save-dev express  
npm i --save-dev mongoose  
npm i --save-dev body-parser  
npm i --save-dev node-restful
```

5. Instale a dependência do nodemon:

```
npm i --save nodemon
```

6. No VSCode, crie uma pasta chamada **src**, usada para conter todo o código-fonte;

7. Nessa pasta, crie o arquivo **loader.js** (vide **package.json**). Esse será nosso arquivo inicial da aplicação;

8. Abaixo de **src**, crie a pasta **config**;

9. Nessa nova pasta, crie o arquivo **server.js** (configurações do servidor);

10. No arquivo **loader.js**, inclua a instrução:

```
require('./config/server');
```

11. No arquivo **server.js**, escreva o código adiante:

```
const port = 3200;  
  
//middlewares - singletons: commons js  
const bodyParser = require('body-parser');  
const express = require('express');  
  
const server = express(); //novo servidor  
  
//para toda requisição que chegar, use o bodyparser para  
//interpretar chegadas no formato urlencoded  
server.use(bodyParser.urlencoded({ extended: true }))  
  
//considera o formato json no corpo da requisição  
server.use(bodyParser.json());  
  
server.listen(port, function () {  
    //template string (observe a crase)  
    console.log(`servidor no ar, na porta ${port}`);  
});
```

12. Crie o arquivo **db.js** na pasta **config** (vamos configurar o mongodb):

```
const mongoose = require("mongoose");

mongoose.Promise = global.Promise; //para evitar warnings
module.exports = mongoose.connect('mongodb://localhost:27017/escola');
```

13. Faça a referência a este arquivo dentro de **loader.js**:

```
require('./config/server');
require('./config/db');
```

Para desenvolver os artefatos do banco de dados, vamos criar o modelo ODM (Object Document Model):

14. Na pasta **src**, crie uma pasta chamada **ws** e, dentro dela, uma nova pasta chamada **cursos** (essa pasta conterá o conteúdo da nossa api);

15. Nessa última pasta, crie o arquivo **cursos.js**:

```
//criando o schema
const restful = require('node-restful');
const mongoose = restful.mongoose; //referência ao mongoose do restful

//definindo o schema curso
const cursoSchema = new mongoose.Schema({
    codigo: { type: Number, required: true },
    descricao: { type: String, required: true },
    cargaHoraria: { type: Number, required: true, min: 4 },
    preco: { type: Number, min: 0 },
    categoria: {type: String, uppercase: true,
        enum:[ 'INFORMATICA', 'ENGENHARIA', 'ADMINISTRACAO', 'REDES' ] }
});

module.exports = restful.model('curso', cursoSchema);
```

16. Na pasta **cursos**, crie o arquivo **cursosService.js**:

```
//serviços rest
const Cursos = require('./cursos');

Cursos.methods(['get', 'post', 'put', 'delete']);
Cursos.updateOptions({ new: true, runValidators: true });

module.exports = Cursos;
```

# Criação do banco de dados e do Web service

17. Na pasta **ws**, crie a pasta **contatos**. Nessa pasta, crie o arquivo (model) **contatos.js**:

```
//criando o schema
const restful = require('node-restful');
const mongoose = restful.mongoose; //referencia ao mongoose do restful

//definindo o schema contato
const contatoSchema = new mongoose.Schema({
  data: { type: Date },
  nome: { type: String },
  email: {type: String},
  assunto: { type: String }
});

module.exports = restful.model('contato', contatoSchema);
```

18. Na mesma pasta, crie o arquivo **contatosService.js**:

```
//serviços rest
const Contatos = require('./contatos');

Contatos.methods(['get', 'post', 'put', 'delete']);
Contatos.updateOptions({ new: true, runValidators: true });

module.exports = Contatos;
```

Vamos, agora, definir as rotas para cada tipo de serviço.

19. Na pasta **config**, crie o arquivo **routes.js**:

```
const express = require('express');

module.exports = function (server) {
  //definir a URL base para todas as rotas
  const router = express.Router();
  server.use('/ws', router);

  //rotas relacionadas às operações com cursos e contatos
  const Cursos = require('../ws/cursos/cursosService');
  const Contatos = require('../ws/contatos/contatosService');

  Cursos.register(router, '/cursos');
  Contatos.register(router, '/contatos');
};
```

20. De volta a **loader.js**, acrescente a lista indicada (observe a referência ao server):

```
const server = require('./config/server');
require('./config/db');
require('./config/routes')(server);
```

21. Para que o server seja visível, devemos exportá-lo no arquivo **server.js**. Vamos fazê-lo:

```
const port = 3200;

//middlewares - singletons: commons js
const bodyParser = require('body-parser');
const express = require('express');

const server = express(); //novo servidor

//para toda requisição que chegar, use o bodyparser para interpretar chegadas
//no formato urlencoded
server.use(bodyParser.urlencoded({ extended: true }));
server.use(bodyParser.json()); //considera o formato json no corpo da requisição

server.listen(port, function () {
    console.log(`servidor no ar, na porta ${port}`); //template string (observe
    //a crase)
});

module.exports = server;
```

22. Vamos, agora, habilitar o CORS (Cross-origin resource sharing) para permitir que nosso Web service seja acessível por todas as origens, mesmo as diferentes do servidor onde o serviço está disponível. Na pasta **config**, inclua o arquivo **cors.js**:

```
module.exports = (request, response, next) => {
    response.header("Access-Control-Allow-Origin", "*");
    response.header("Access-Control-Allow-Methods", "GET,POST,PUT,DELETE");
    response.header("Access-Control-Allow-Headers", "Origin,
    X-Requested-With, Content-Type, Accept");
    next(); //necessário para dar continuidade ao processo de requisição
};
```

# Criação do banco de dados e do Web service

23. No **server.js**, acrescente a referência ao CORS:

```
const port = 3200;

const bodyParser = require('body-parser');
const express = require('express');

const server = express(); //novo servidor

const allowCors = require('./cors');

server.use(bodyParser.urlencoded({ extended: true }));
server.use(bodyParser.json());

server.use(allowCors);

server.listen(port, function () {
  console.log(`servidor no ar, na porta ${port}`);
});

module.exports = server;
```

24. No prompt de comandos, execute a instrução:

```
npm run dev
```

25. No browser, acesse:

```
localhost:3200/ws/cursos  
localhost:3200/ws/contatos
```





3

# Configuração do webpack



Atividade



## A – Instalando todas as dependências do cliente

Nesta fase do projeto, desenvolveremos a parte de configurações do webpack, mas vamos baixar as principais dependências para a continuação do desenvolvimento.

1. Na pasta **cursoReact**, inclua uma nova pasta chamada **client**;
2. No prompt de comandos, acesse essa pasta e, em seguida, execute o comando:

```
npm init -yes
```

3. Adicione as dependências em modo dev: **webpack**, **webpack-dev-server** e **webpack-cli**:

```
npm i --save-dev webpack webpack-dev-server webpack-cli
```

4. Adicione, também, as dependências do Babel:

```
npm i --save-dev babel-core babel-loader babel-plugin-react-html-attrs babel-plugin-transform-object-rest-spread babel-preset-es2015 babel-preset-react
```

5. Para o uso do CSS, vamos adicionar mais estas dependências:

```
npm i --save-dev mini-css-extract-plugin css-loader style-loader file-loader
```

6. Adicione as dependências do **bootstrap** e **font-awesome** (neste caso, consideraremos uma versão fixa, pois sua utilização pode passar por variações):

```
npm i --save-dev bootstrap@4.0.0 font-awesome@4.7.0
```

7. Adicione as dependências para uso do React:

```
npm i --save-dev react react-dom react-router-dom axios
```

8. Abra o VSCode na pasta **client** que você criou;

# Configuração do webpack

9. Na raiz, crie um arquivo chamado **webpack.config.js**:

```
const webpack = require("webpack");
const extract_twp = require("mini-css-extract-plugin");

module.exports = {
  entry: "./src/index.jsx",
  output: {
    path: __dirname + "/public",
    filename: "./app.js",
    publicPath: '/' //importante para as rotas
  },
  devServer: {
    port: 3000,
    contentBase: "./public",
    historyApiFallback: true //importante para as rotas
  },
  resolve: {
    extensions: ['.js', '.jsx'], //não pode ter espaços em branco
  },
  plugins: [
    new extract_twp({
      filename: "app.css"
    })
  ],
  module: {
    rules: [{ //mudou de loaders para rules
      test: /\.jsx?$/,
      loader: 'babel-loader',
      exclude: /node_modules/,
      query: {
        presets: ["es2015", "react"],
        plugins: ["transform-object-rest-spread"]
      }
    },
    {
      test: /\.css$/,
      use: [ "style-loader", "css-loader" ]
    },
    {
      test: /\.woff|\.woff2|\.ttf|\.eot|\.svg*.*$/,
      loader: "file-loader"
    }]
  },
  mode: "development"
}
```

## 10. Altere o arquivo `package.json`:

```
{  
  "name": "client",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "dev": "webpack-dev-server --progress --colors --inline --hot",  
    "production": "webpack --progress -p"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "devDependencies": {  
    "axios": "^0.18.0",  
    "babel-core": "^6.26.3",  
    "babel-loader": "^7.1.5",  
    "babel-plugin-react-html-attrs": "^2.1.0",  
    "babel-plugin-transform-object-rest-spread": "^6.26.0",  
    "babel-preset-es2015": "^6.24.1",  
    "babel-preset-react": "^6.24.1",  
    "bootstrap": "^4.0.0",  
    "css-loader": "^1.0.0",  
    "file-loader": "^1.1.11",  
    "font-awesome": "^4.7.0",  
    "mini-css-extract-plugin": "^0.4.1",  
    "react": "^16.4.2",  
    "react-dom": "^16.4.2",  
    "react-router": "^4.3.1",  
    "style-loader": "^0.22.1",  
    "webpack": "^4.16.5",  
    "webpack-cli": "^3.1.0",  
    "webpack-dev-server": "^3.1.5"  
  }  
}
```

## 11. Crie uma pasta chamada `/public`:

## 12. Nessa pasta, crie o arquivo `index.html` com o seguinte conteúdo:

```
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <meta http-equiv="X-UA-Compatible" content="ie=edge">  
    <title>Escola</title>  
  </head>  
  <body>  
    <div id="app" class="container"></div>  
    <script src="app.js"></script>  
  </body>  
</html>
```

# Configuração do webpack

13. Observe as referências ao arquivo **app.js**, definido no arquivo de configurações do webpack. Crie a pasta **src** (raiz de todos os componentes da aplicação);

14. Nessa pasta, crie outra pasta chamada **main** e, nela, o arquivo **app.jsx**:

```
import styles from '../../node_modules/bootstrap/dist/css/bootstrap.min.css';

import React from 'react';

export default props => (
  <div className={styles.container}>
    <h1>Cadastro de Cursos</h1>
  </div>
)
```

15. Na pasta **src**, inclua o arquivo **index.jsx** (ponto de partida da aplicação):

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './main/app';

ReactDOM.render(<App/>, document.getElementById('app'));
```

16. Execute a aplicação até este ponto, usando a instrução:

```
npm run dev
```



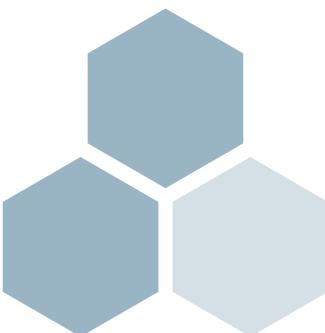


4

# Aplicação do React.js



Atividade



## A – Criando os componentes principais da aplicação

Agora vamos iniciar o desenvolvimento da nossa aplicação propriamente dita.

1. Crie uma pasta chamada **curso** abaixo de **src**. Dentro dela, um arquivo chamado **curso.jsx**:

```
import React, { Component } from 'react';

export default class Curso extends Component {
    render() {
        return (
            <div>
                <h1>Classe Curso</h1>
            </div>
        )
    }
}
```

2. No arquivo **app.jsx**, adicione a referência à classe **Curso** e use-o como componente. Acrescente, também, a referência à API **font-awesome**.

```
import '../../../../../node_modules/bootstrap/dist/css/bootstrap.min.css';
import '../../../../../node_modules/font-awesome/css/font-awesome.min.css';

import React from 'react';
import Curso from '../curso/curso';

export default props =>
    <Curso />
```

3. Vamos, agora, acrescentar os componentes que farão parte da nossa aplicação de fato. Além do cadastro de cursos, teremos o cadastro de contatos, além do menu de opções. Para tanto, criaremos os componentes **contato** e **menu**. Na pasta **src**, crie uma nova pasta chamada **contato** e, dentro dela, o arquivo **contato.jsx**, com o seguinte conteúdo:

```
import React, { Component } from 'react';

export default class Contato extends Component {
    render() {
        return (
            <div className="container">
                <h1>Classe Contato</h1>
            </div>
        )
    }
}
```

4. Para podermos visualizar esses componentes, no arquivo **app.jsx** inclua as referências a eles:

```
import '../../../../../node_modules/bootstrap/dist/css/bootstrap.min.css';
import '../../../../../node_modules/font-awesome/css/font-awesome.min.css';

import React from 'react';
import Curso from '../curso/curso';
import Contato from '../contato/contato';

export default props => (
  <div>
    <Curso />
    <Contato />
  </div>
)
```

5. Visualize a execução;

6. Criaremos o menu de opções agora. Acrescente a pasta **menu** abaixo de **src**. Nessa pasta, acrescente o arquivo **menu.jsx**. Use o conteúdo do bootstrap, conforme documentação (usamos a versão 4 do bootstrap). Verifique os links sugeridos:

```
import React from 'react';

export default props => (
  <nav className="navbar navbar-expand-lg navbar-light bg-light">
    <a className="navbar-brand" href="#">
      ABC Courses
    </a>

    <button className="navbar-toggler" type="button"
      data-toggle="collapse"
      data-target="#navbarContent"
      aria-controls="navbarContent"
      aria-expanded="false"
      aria-label="Toggle navigation">
      <span className="navbar-toggler-icon"></span>
    </button>

    <div className="collapse navbar-collapse" id="navbarContent">
      <ul className="navbar-nav mr-auto">
        <li className="nav-item">
          <a className="nav-link" href="#/cursos">Cursos</a>
        </li>

        <li className="nav-item">
          <a className="nav-link" href="#/contato">Contato</a>
        </li>
      </ul>
    </div>
  </nav>
)
```

7. Acrescente esse componente em **app.jsx**:

```
import '../../../../../node_modules/bootstrap/dist/css/bootstrap.min.css';

import React from 'react';
import Curso from '../curso/curso';
import Contato from '../contato/contato';
import Menu from '../menu/menu';

export default props =>
  <div>
    <Menu />
    <Curso />
    <Contato />
  </div>
)
```

8. Para que o menu ocupe toda a largura do browser, faça as alterações sugeridas:

- **index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Escola</title>
</head>
<body>
  <div id="app"></div>
  <script src="app.js"></script>
</body>
</html>
```

- **curso.jsx**

```
import React, { Component } from 'react';

export default class Curso extends Component {
  render() {
    return (
      <div className="container">
        <h1>Classe Curso</h1>
      </div>
    )
  }
}
```

Transferimos a classe container do bootstrap para os componentes, exceto para o menu. Esse procedimento foi necessário para que o menu ocupasse toda a largura da tela, como já sugerido anteriormente.

9. Para que o menu seja responsivo, inclua as referências ao jQuery e ao Popper.js (necessário para o bootstrap 4.0):

```
npm i --save-dev query popper.js
```

10. Após essa instalação, altere o arquivo **app.jsx**:

```
import '../../../../../node_modules/bootstrap/dist/css/bootstrap.min.css';
import '../../../../../node_modules/font-awesome/css/font-awesome.min.css';

import '../../../../../node_modules/jquery/dist/jquery.min';
import '../../../../../node_modules/popper.js/dist/umd/popper.min';
import '../../../../../node_modules/bootstrap/dist/js/bootstrap.min';

import React from 'react';
import Curso from '../curso/curso';
import Contato from '../contato/contato';
import Menu from '../menu/menu';

export default props => (
  <div>
    <Menu />
    <Curso />
    <Contato />
  </div>
)
```

## B – Definindo as rotas

Vamos realizar severas alterações de forma a incluir as rotas na nossa aplicação.

1. Primeiro, na pasta **main**, vamos criar um arquivo chamado **rotas.jsx**, com o seguinte conteúdo:

```
import React from 'react';
import { BrowserRouter, Route, Switch } from 'react-router-dom';

import Curso from '../curso/curso';
import Contato from '../contato/contato';

export default props => (
  <Switch>
    <Route path='/cursos' component={Curso} />
    <Route path='/contato' component={Contato} />
    <Route path='*' component={Curso} />
  </Switch>
)
```

2. Em seguida, no arquivo **app.jsx**, substituímos os componentes abaixo do menu pela rota:

```
import '../../../../../node_modules/bootstrap/dist/css/bootstrap.min.css';
import '../../../../../node_modules/jquery/dist/jquery.min';
import '../../../../../node_modules/popper.js/dist/umd/popper.min';
import '../../../../../node_modules/bootstrap/dist/js/bootstrap.min';

import React from 'react';
import Menu from '../menu/menu';
import Rotas from '../main/rotas';

export default props =>
  <div>
    <Menu />
    <Rotas />
  </div>
)
```

3. Alteramos, agora, o conteúdo do componente inicial, **index.jsx**:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './main/app';
import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
), document.getElementById('app'));
```

4. Para refletir no menu de opções, vamos alterar **menu.jsx**. Observe as alterações realizadas:

```
import React from 'react';
import { Link } from 'react-router-dom';

export default props => (
  <nav className="navbar navbar-expand-lg navbar-light bg-light">
    <Link className="navbar-brand" to="/">ABC Courses</Link>

    <button className="navbar-toggler" type="button"
      data-toggle="collapse"
      data-target="#navbarContent"
      aria-controls="navbarContent"
      aria-expanded="false"
      aria-label="Toggle navigation">
      <span className="navbar-toggler-icon"></span>
    </button>

    <div className="collapse navbar-collapse" id="navbarContent">
      <ul className="navbar-nav mr-auto">
        <li className="nav-item">
          <Link className="nav-link" to="/cursos">Cursos</Link>
        </li>

        <li className="nav-item">
          <Link className="nav-link" to="/contato">
            Contato
          </Link>
        </li>
      </ul>
    </div>
  </nav>
)
```

5. Execute a aplicação e verifique a naveabilidade a partir do menu. Chame os links pela URL também.

### C – Personalizando o título de cada página

Vamos, agora, criar um título e um subtítulo para cada página, agrupando-os em um único componente.

1. Crie um arquivo chamado **cabecalho.jsx** na pasta **menu**. Seu conteúdo deve ser:

```
import React from 'react';

export default props =>
  <header className="pb-2 mt-4 mb-2 border-bottom">
    <h2><strong>{props.titulo}</strong> -
    <small>{props.subtitulo}</small></h2>
  </header>
)
```

# Desenvolvimento de Componentes Web e Mobile com React.Js, Redux e React Native

2. Altere os arquivos **curso.jsx** e **contato.jsx** para incluir este cabeçalho:

- **curso.jsx**

```
import React, { Component } from 'react';
import Cabecalho from '../menu/cabecalho';

export default class Curso extends Component {
    render() {
        return (
            <div className="container">
                <Cabecalho titulo="Cursos"
                           subtítulo="cadastro de cursos" />
            </div>
        )
    }
}
```

- **contato.jsx**

```
import React, { Component } from 'react';
import Cabecalho from '../menu/cabecalho';

export default class Contato extends Component {
    render() {
        return (
            <div className="container">
                <Cabecalho titulo="Contato"
                           subtítulo="entre em contato conosco" />
            </div>
        )
    }
}
```

3. Veja o resultado na execução. Se quiser alterar esses valores, pode fazê-lo para ajustar às suas necessidades.

## D – Construindo a base para o formulário e listagem de cursos

Vamos, agora, definir três componentes:

- Formulário para cadastro de cursos;
- Listagem de cursos;
- União do formulário com a listagem.

O objetivo é deixar o cadastro e a listagem separados de forma que possam ser reutilizados em outras partes da aplicação.

1. Defina o cadastro de cursos no arquivo **cursoForm.jsx**:

```
import React from 'react';

export default props => (
  <div className="border-right pl-3 pr-3">
    <h3 className="border-bottom">Formulario</h3>
  </div>
)
```

2. Defina a listagem no arquivo **cursoList.jsx**:

```
import React from 'react';

export default props => (
  <div>
    <h3>Lista de Cursos</h3>
  </div>
)
```

3. Defina o componente para cadastro de cursos, responsável pela união dos dois componentes, no arquivo **cursoCadastro.jsx**:

```
import React from 'react';
import CursoForm from './cursoForm';
import CursoList from './cursoList';

export default props => (
  <div className="row border-bottom">
    <div className="col-md-6">
      <CursoForm />
    </div>
    <div className="col-md-6">
      <CursoList />
    </div>
  </div>
)
```

4. Vamos, agora, atualizar o componente **curso.jsx**:

```
import React, { Component } from 'react';
import Cabecalho from '../menu/cabecalho';
import CursoCadastro from './cursoCadastro';

export default class Curso extends Component {
    render() {
        return (
            <div className="container">
                <Cabecalho titulo="Cursos"
                           subtítulo="cadastro de cursos" />
                <CursoCadastro />
            </div>
        )
    }
}
```

5. Verifique o resultado;

6. Atualize o formulário – arquivo **cursoForm.jsx**:

```
import React from 'react';

export default props => (
    <div className="border-right pl-3 pr-3">
        <h3 className="border-bottom">Formulario</h3>
        <form>
            <div className="form-group row">
                <label for="codigo"
                      className="col-sm-3 col-form-label">
                    Código:
                </label>
                <div className="col-sm-5 col-6">
                    <input type="number"
                           className="form-control" id="codigo" />
                </div>
            </div>

            <div className="form-group row">
                <label for="descrição"
                      className="col-sm-3 col-form-label">
                    Descrição:
                </label>
                <div className="col-sm-9">
                    <input type="text"
                           className="form-control" id="descricao" />
                </div>
            </div>
        </form>
    </div>
)
```

```
<div className="form-group row">
    <label for="cargaHoraria"
        className="col-sm-3 col-form-label">
        Carga Horária:</label>
    <div className="col-sm-5 col-6">
        <input type="number"
            className="form-control" id="cargaHoraria" />
    </div>
</div>

<div className="form-group row">
    <label for="preco"
        className="col-sm-3 col-form-label">
        Preço:</label>
    <div className="col-sm-5 col-6">
        <input type="text"
            className="form-control" id="preco" />
    </div>
</div>

<div className="form-group row">
    <label for="categoria"
        className="col-sm-3 col-form-label">Categoria:</label>
    <div className="col-sm-6 col-6">
        <select className="form-control" id="categoria" >
            <option>INFORMATICA</option>
            <option>ENGENHARIA</option>
            <option>ADMINISTRACAO</option>
            <option>REDES</option>
        </select>
    </div>
</div>

<div className="form-group row">
    <button
        className="btn btn-primary ml-3 mb-3">
        Adicionar
    </button>
</div>

</form>
</div>
)
```

7. Verifique o formulário. Mude a resolução para verificar a responsividade.

## E – Adicionando evento ao botão adicionar do formulário

1. Realize as seguintes alterações no arquivo **cursoCadastro.jsx** (vamos transformá-lo em uma classe para adicionarmos eventos necessários. Esta classe, como chama o formulário e a lista, concentrará a lógica de acesso ao Web service):

```
import React, { Component } from 'react';
import CursoForm from './cursoForm';
import CursoList from './cursoList';
import axios from 'axios';

const URL = "http://localhost:3200/ws/cursos";

export default class CursoCadastro extends Component {
    constructor(props) {
        super(props);
        this.state = {
            codigo: '',
            descricao: '',
            cargaHoraria: '',
            preco: '',
            categoria: 'ENGENHARIA',
            cursos: []
        };
        //configura os eventos para o contexto atual
        this.adicionarCurso = this.adicionarCurso.bind(this);

        this.codigoInput = this.codigoInput.bind(this);
        this.descricaoInput = this.descricaoInput.bind(this);
        this.chInput = this.chInput.bind(this);
        this.precoInput = this.precoInput.bind(this);
        this.categoriaInput = this.categoriaInput.bind(this);

        this.listar();
    }

    //evento para adicionar um curso
    adicionarCurso(e) {
        e.preventDefault();

        const codigo = this.state.codigo;
        const descricao = this.state.descricao;
        const cargaHoraria = this.state.cargaHoraria;
        const preco = this.state.preco;
        const categoria = this.state.categoria;

        axios.post(URL, { codigo, descricao, cargaHoraria, preco, categoria })
            .then(response =>
            {
                alert("Curso adicionado");
                this.listar();
            });
    }
}
```

```

//eventos para ler os campos de entrada
codigoInput(e) {
    this.setState({ ...this.state, codigo: e.target.value })
}
descricaoInput(e) {
    this.setState({ ...this.state, descricao: e.target.value })
}
chInput(e) {
    this.setState({ ...this.state, cargaHoraria: e.target.value })
}
precoInput(e) {
    this.setState({ ...this.state, preco: e.target.value })
}
categoriaInput(e) {
    this.setState({ ...this.state, categoria: e.target.value })
}
//

listar() {
    axios.get(URL).then(response => this.setState({...this.state, cursos:
response.data}))
}

render() {
    return (
        <div className="row border-bottom">
            <div className="col-md-6">
                <CursoForm
                    codigo={this.state.codigo}
                    descricao={this.state.descricao}
                    cargaHoraria={this.state.cargaHoraria}
                    preco={this.state.preco}
                    categoria={this.state.categoria}

                    codigoInput={this.codigoInput}
                    descricaoInput={this.descricaoInput}
                    chInput={this.chInput}
                    precoInput={this.precoInput}
                    categoriaInput={this.categoriaInput}

                    adicionarCurso={this.adicionarCurso}
                />
            </div>
            <div className="col-md-6">
                <CursoList cursos={this.state.cursos} />
            </div>
        </div>
    )
}
}

```

2. Altere o arquivo **cursoForm.jsx** para receber as propriedades por meio do componente **props**:

```
import React from 'react';

export default props => {

    return (
        <div className="border-right pl-3 pr-3">
            <h3 className="border-bottom">Formulario</h3>
            <form>
                <div className="form-group row">
                    <label htmlFor="codigo" className="col-sm-3 col-form-label">Código:</label>
                    <div className="col-sm-5 col-6">
                        <input type="number" className="form-control" id="codigo"
                            value={props.codigo}
                            onChange={props.codigoInput} />
                    </div>
                </div>

                <div className="form-group row">
                    <label htmlFor="descricao" className="col-sm-3 col-form-label">Descrição:</label>
                    <div className="col-sm-9">
                        <input type="text" className="form-control" id="descricao"
                            value={props.descricao}
                            onChange={props.descricaoInput} />
                    </div>
                </div>

                <div className="form-group row">
                    <label htmlFor="cargaHoraria" className="col-sm-3 col-form-label">Carga Horária:</label>
                    <div className="col-sm-5 col-6">
                        <input type="number" className="form-control"
                            id="cargaHoraria"
                            value={props.cargaHoraria}
                            onChange={props.chInput} />
                    </div>
                </div>

                <div className="form-group row">
                    <label htmlFor="preco" className="col-sm-3 col-form-
```

```
label">Preço:</label>
    <div className="col-sm-5 col-6">
        <input type="text" className="form-control" id="preco"
            value={props.preco}
            onChange={props.precoInput} />
    </div>
</div>

<div className="form-group row">
    <label htmlFor="categoria" className="col-sm-3 col-form-label">Categoria:</label>
    <div className="col-sm-6 col-6">
        <select className="form-control" id="categoria"
            onChange={props.categoriaInput}
            value={props.categoria}>
            <option>INFORMATICA</option>
            <option>ENGENHARIA</option>
            <option>ADMINISTRACAO</option>
            <option>REDES</option>
        </select>
    </div>
</div>

<div className="form-group row">
    <button className="btn btn-primary ml-3 mb-3"
        onClick={props.adicionarCurso}>Adicionar</button>
</div>
</div>
</div>
)
}
```

3. Altere o arquivo **cursoList.jsx** para exibir a lista de cursos, proveniente de **cursoCadastro.jsx**:

```
import React from 'react';

export default props => {

  const exibirLinhas = () => {

    //retorna a lista de props se existir
    const cursos = props.cursos || [];

    return cursos.map(curso => (
      <tr key={curso._id}>
        <td>{curso.codigo}</td>
        <td>{curso.descricao}</td>
      </tr>
    )));
  }

  return(
    <div>
      <h3>Lista de Cursos</h3>
      <table className="table table-striped">
        <thead>
          <tr>
            <th> Código </th>
            <th> Descrição </th>
          </tr>
        </thead>
        <tbody>
          {exibirLinhas()}
        </tbody>
      </table>
    </div>
  )
}
```

4. Visualize a aplicação até o momento. Realize alguns cadastros e visualize o resultado (é importante que o projeto **server** esteja em funcionamento!);

5. Vamos, agora, incluir o recurso para remover um curso cadastrado. No arquivo **cursoList.jsx**, adicione uma coluna e, nesta coluna, um botão para remover o curso (observe o evento incluído no botão):

```
import React from 'react';

export default props => {

  const exibirLinhas = () => {
    const cursos = props.cursos || [];
    //retorna a lista de props se existir

    return cursos.map(curso => (
      <tr key={curso._id}>
        <td>{curso.codigo}</td>
        <td>{curso.descricao}</td>
        <td><button className="btn btn-danger"
          onClick={() => props.removerCurso(curso)}>
            <i className="fa fa-trash-o"></i>
          </button>
        </td>
      </tr>
    )));
  }

  return(
    <div>
      <h3>Lista de Cursos</h3>
      <table className="table table-striped">
        <thead>
          <tr>
            <th>Código</th>
            <th>Descrição</th>
            <th></th>
          </tr>
        </thead>
        <tbody>
          {exibirLinhas()}
        </tbody>
      </table>
    </div>
  )
}
```

6. No arquivo **cursoCadastro.jsx**, faça as alterações para refletir o funcionamento do novo ícone:

```
import React, { Component } from 'react';
import CursoForm from './cursoForm';
import CursoList from './cursoList';
import axios from 'axios';

const URL = "http://localhost:3200/ws/cursos";

export default class CursoCadastro extends Component {
    constructor(props) {
        super(props);
        this.state = {
            codigo: '',
            descricao: '',
            cargaHoraria: '',
            preco: '',
            categoria: 'ENGENHARIA',
            cursos: []
        };
        //configura os eventos para o contexto atual
        this.adicionarCurso = this.adicionarCurso.bind(this);
        this.removerCurso = this.removerCurso.bind(this);

        this.codigoInput = this.codigoInput.bind(this);
        this.descricaoInput = this.descricaoInput.bind(this);
        this.chInput = this.chInput.bind(this);
        this.precoInput = this.precoInput.bind(this);
        this.categoriaInput = this.categoriaInput.bind(this);

        this.listar();
    }
    //
    //evento para adicionar um curso
    adicionarCurso(e) {
        e.preventDefault();

        const codigo = this.state.codigo;
        const descricao = this.state.descricao;
        const cargaHoraria = this.state.cargaHoraria;
        const preco = this.state.preco;
        const categoria = this.state.categoria;

        axios.post(URL, { codigo, descricao, cargaHoraria, preco, categoria })
            .then(response =>
            {
                alert("Curso adicionado");
                this.listar();
            });
    }
}
```

```

//evento para remover um curso
removerCurso(curso) {
    if (confirm("Tem certeza que deseja remover este curso?")) {
        axios.delete(` ${URL}/${curso._id}`)
            .then(response => this.listar());
    }
}

//eventos para ler os campos de entrada
codigoInput(e) {
    this.setState({ ...this.state, codigo: e.target.value })
}
descricaoInput(e) {
    this.setState({ ...this.state, descricao: e.target.value })
}
chInput(e) {
    this.setState({ ...this.state, cargaHoraria: e.target.value })
}
precoInput(e) {
    this.setState({ ...this.state, preco: e.target.value })
}
categoriaInput(e) {
    this.setState({ ...this.state, categoria: e.target.value })
}
//
listar() {
    axios.get(URL).then(response => this.setState({...this.state, cursos: response.data}))
}

render() {
    return (
        <div className="row border-bottom ">
            <div className="col-md-6">
                <CursoForm
                    codigo={this.state.codigo}
                    descricao={this.state.descricao}
                    cargaHoraria={this.state.cargaHoraria}
                    preco={this.state.preco}
                    categoria={this.state.categoria}

                    codigoInput={this.codigoInput}
                    descricaoInput={this.descricaoInput}
                    chInput={this.chInput}
                    precoInput={this.precoInput}
                    categoriaInput={this.categoriaInput}

                    adicionarCurso={this.adicionarCurso}
                />
            </div>
            <div className="col-md-6">
                <CursoList cursos={this.state.cursos}>
                    removerCurso={this.removerCurso} />
                </div>
            </div>
        )
    }
}

```

7. Teste a nova funcionalidade;

8. Para consultar um curso e exibir seus dados no formulário, vamos realizar as seguintes alterações:

- **cursoList.jsx**

```
import React from 'react';

export default props => {

  const exibirLinhas = () => {
    const cursos = props.cursos || [];
    //retorna a lista de props se existir

    return cursos.map(curso => (
      <tr key={curso._id}>
        <td>{curso.codigo}</td>
        <td>{curso.descricao}</td>
        <td>
          <button className="btn btn-success"
            onClick={() => props.consultarCurso(curso)}>
            <i className="fa fa-check"></i>
          </button>
          <button className="btn btn-danger"
            onClick={() => props.removerCurso(curso)}>
            <i className="fa fa-trash-o"></i>
          </button>
        </td>
      </tr>
    )));
  }

  return(
    <div>
      <h3>Lista de Cursos</h3>
      <table className="table table-striped">
        <thead>
          <tr>
            <th>Código</th>
            <th>Descrição</th>
            <th></th>
          </tr>
        </thead>
        <tbody>
          {exibirLinhas()}
        </tbody>
      </table>
    </div>
  )
}
```

- **cursoCadastro.jsx**

```
import React, { Component } from 'react';
import CursoForm from './cursoForm';
import CursoList from './cursoList';
import axios from 'axios';

const URL = "http://localhost:3200/ws/cursos";

export default class CursoCadastro extends Component {
    constructor(props) {
        super(props);
        this.state = {
            codigo: '',
            descricao: '',
            cargaHoraria: '',
            preco: '',
            categoria: 'ENGENHARIA',
            cursos: []
        };

        //configura os eventos para o contexto atual
        this.adicionarCurso = this.adicionarCurso.bind(this);
        this.removerCurso = this.removerCurso.bind(this);
        this.consultarCurso = this.consultarCurso.bind(this);

        this.codigoInput = this.codigoInput.bind(this);
        this.descricaoInput = this.descricaoInput.bind(this);
        this.chInput = this.chInput.bind(this);
        this.precoInput = this.precoInput.bind(this);
        this.categoriaInput = this.categoriaInput.bind(this);

        this.listar();
    }

    //evento para adicionar um curso
    adicionarCurso(e) {
        e.preventDefault();

        const codigo = this.state.codigo;
        const descricao = this.state.descricao;
        const cargaHoraria = this.state.cargaHoraria;
        const preco = this.state.preco;
        const categoria = this.state.categoria;

        axios.post(URL, { codigo, descricao, cargaHoraria, preco, categoria })
            .then(response =>
            {
                alert("Curso adicionado");
                this.listar();
            });
    }
}
```

```
//evento para remover um curso
removerCurso(curso) {
    if (confirm("Tem certeza que deseja remover este curso?")) {
        axios.delete(` ${URL}/${curso._id}`)
            .then(response => this.listar());
    }
}

consultarCurso(curso) {
    axios.get(` ${URL}/${curso._id}`)
        .then(response => {
            console.log(response.data);
            this.setState(
            {
                ...this.state,
                codigo: response.data.codigo,
                descricao: response.data.descricao,
                cargaHoraria: response.data.cargaHoraria,
                preco: response.data.preco,
                categoria: response.data.categoria
            });
        });
}

//eventos para ler os campos de entrada
codigoInput(e) {
    this.setState({ ...this.state, codigo: e.target.value })
}
descricaoInput(e) {
    this.setState({ ...this.state, descricao: e.target.value })
}
chInput(e) {
    this.setState({ ...this.state, cargaHoraria: e.target.value })
}
precoInput(e) {
    this.setState({ ...this.state, preco: e.target.value })
}
categoriaInput(e) {
    this.setState({ ...this.state, categoria: e.target.value })
}
//listar()
listar() {
    axios.get(URL).then(response => this.setState({...this.state, cursos: response.data}))
}

render() {
    return (
        <div className="row border-bottom">
            <div className="col-md-6">
                <CursoForm
                    codigo={this.state.codigo}
                    descricao={this.state.descricao}
                    cargaHoraria={this.state.cargaHoraria}
                    preco={this.state.preco}
                    categoria={this.state.categoria}>

```

```

        codigoInput={this.codigoInput}
        descricaoInput={this.descricaoInput}
        chInput={this.chInput}
        precoInput={this.precoInput}
        categoriaInput={this.categoriaInput}

        adicionarCurso={this.adicionarCurso}
    />
</div>
<div className="col-md-6">
    <CursoList
        cursos={this.state.cursos}
        removerCurso={this.removerCurso}
        consultarCurso={this.consultarCurso}>
    />
</div>
</div>
)
}
}
}

```

9. Precisamos preparar o formulário para um novo curso, limpando os campos. Para tanto, devemos adicionar um botão no formulário que permita realizar esta tarefa:

- **cursoForm.jsx**

```

import React from 'react';

export default props => {

    return (
        <div className="border-right pl-3 pr-3">
            <h3 className="border-bottom">Formulario</h3>
            <form>
                <div className="form-group row">
                    <label htmlFor="codigo"
                        className="col-sm-3 col-form-label">Código:</label>
                    <div className="col-sm-5 col-6">
                        <input type="number"
                            className="form-control" id="codigo"
                            value={props.codigo}
                            onChange={props.codigoInput} />
                    </div>
                </div>

                <div className="form-group row">
                    <label htmlFor="descricao"
                        className="col-sm-3 col-form-label">Descrição:</label>
                    <div className="col-sm-9">
                        <input type="text"
                            className="form-control" id="descricao"
                            value={props.descricao}
                            onChange={props.descricaoInput} />
                    </div>
                </div>
            </form>
        </div>
    )
}

```

```
<div className="form-group row">
    <label htmlFor="cargaHoraria"
    className="col-sm-3 col-form-label">
        Carga Horária:</label>
    <div className="col-sm-5 col-6">
        <input type="number"
        className="form-control" id="cargaHoraria"
        value={props.cargaHoraria}
        onChange={props.chInput} />
    </div>
</div>

<div className="form-group row">
    <label htmlFor="preco"
    className="col-sm-3 col-form-label">Preço:</label>
    <div className="col-sm-5 col-6">
        <input type="text"
        className="form-control" id="preco"
        value={props.preco}
        onChange={props.precoInput} />
    </div>
</div>

<div className="form-group row">
    <label htmlFor="categoria"
    className="col-sm-3 col-form-label">Categoria:</label>
    <div className="col-sm-6 col-6">
        <select className="form-control" id="categoria"
        onChange={props.categoriaInput}
        value={props.categoria}>
            <option>INFORMATICA</option>
            <option>ENGENHARIA</option>
            <option>ADMINISTRACAO</option>
            <option>REDES</option>
        </select>
    </div>
</div>

<div className="form-group row">
    <button className="btn btn-primary ml-3 mb-3"
    onClick={props.novoCurso}>Novo</button>

```

```
        <button className="btn btn-primary ml-3 mb-3"
        onClick={props.adicionarCurso}>Adicionar</button>
    </div>
</form>
</div>
)
```

}

126

Editora  
**IMPACTA**

- **cursoCadastro.jsx**

```
import React, { Component } from 'react';
import CursoForm from './cursoForm';
import CursoList from './cursoList';
import axios from 'axios';

const URL = "http://localhost:3200/ws/cursos";

export default class CursoCadastro extends Component {
    constructor(props) {
        super(props);
        this.state = {
            codigo: '',
            descricao: '',
            cargaHoraria: '',
            preco: '',
            categoria: 'ENGENHARIA',
            cursos: []
        };
        //configura os eventos para o contexto atual
        this.adicionarCurso = this.adicionarCurso.bind(this);
        this.removerCurso = this.removerCurso.bind(this);
        this.consultarCurso = this.consultarCurso.bind(this);
        this.novoCurso = this.novoCurso.bind(this);

        this.codigoInput = this.codigoInput.bind(this);
        this.descricaoInput = this.descricaoInput.bind(this);
        this.chInput = this.chInput.bind(this);
        this.precoInput = this.precoInput.bind(this);
        this.categoriaInput = this.categoriaInput.bind(this);
    }
    //evento para adicionar um curso
    adicionarCurso(e) {
        e.preventDefault();

        const codigo = this.state.codigo;
        const descricao = this.state.descricao;
        const cargaHoraria = this.state.cargaHoraria;
        const preco = this.state.preco;
        const categoria = this.state.categoria;

        axios.post(URL, { codigo, descricao, cargaHoraria, preco, categoria })
            .then(response =>
            {
                alert("Curso adicionado");
                this.listar();
            });
    }
}
```

```
//evento para remover um curso
removerCurso(curso) {
    if (confirm("Tem certeza que deseja remover este curso?")) {
        axios.delete(` ${URL}/${curso._id}`)
            .then(response => this.listar());
    }
}

consultarCurso(curso) {
    axios.get(` ${URL}/${curso._id}`)
        .then(response => {
            console.log(response.data);
            this.setState(
                {
                    ...this.state,
                    codigo: response.data.codigo,
                    descricao: response.data.descricao,
                    cargaHoraria: response.data.cargaHoraria,
                    preco: response.data.preco,
                    categoria: response.data.categoria
                });
        });
}

novoCurso(e) {
    e.preventDefault();
    this.setState({
        ...this.state, codigo: '',
        descricao: '',
        cargaHoraria: '',
        preco: '',
        categoria: 'INFORMATICA',
    });
}

//eventos para ler os campos de entrada
codigoInput(e) {
    this.setState({ ...this.state, codigo: e.target.value })
}
descricaoInput(e) {
    this.setState({ ...this.state, descricao: e.target.value })
}
chInput(e) {
    this.setState({ ...this.state, cargaHoraria: e.target.value })
}
precoInput(e) {
    this.setState({ ...this.state, preco: e.target.value })
}
categoriaInput(e) {
    this.setState({ ...this.state, categoria: e.target.value })
}
//
```

```
listar() {
    axios.get(URL).then(response => this.setState({...this.state, cursos:
response.data}))
}

render() {
    return (
        <div className="row border-bottom">
            <div className="col-md-6">
                <CursoForm
                    codigo={this.state.codigo}
                    descricao={this.state.descricao}
                    cargaHoraria={this.state.cargaHoraria}
                    preco={this.state.preco}
                    categoria={this.state.categoria}

                    codigoInput={this.codigoInput}
                    descricaoInput={this.descricaoInput}
                    chInput={this.chInput}
                    precoInput={this.precoInput}
                    categoriaInput={this.categoriaInput}

                    adicionarCurso={this.adicionarCurso}
                    novoCurso={this.novoCurso}
                />
            </div>
            <div className="col-md-6">
                <CursoList
                    cursos={this.state.cursos}
                    removerCurso={this.removerCurso}
                    consultarCurso={this.consultarCurso}
                />
            </div>
        )
    }
}
```

10. Ao adicionar um novo curso, limpe os campos logo em seguida, além de atualizar a lista. Para tanto, altere apenas a função **adicionarCurso()**:

```
//evento para adicionar um curso
adicionarCurso(e) {
  e.preventDefault();

  const codigo = this.state.codigo;
  const descricao = this.state.descricao;
  const cargaHoraria = this.state.cargaHoraria;
  const preco = this.state.preco;
  const categoria = this.state.categoria;

  axios.post(URL, { codigo, descricao, cargaHoraria, preco, categoria })
    .then(response =>
    {
      alert("Curso adicionado");
      this.listar();
      this.novoCurso(e);
    });
}
```

11. Teste a aplicação.



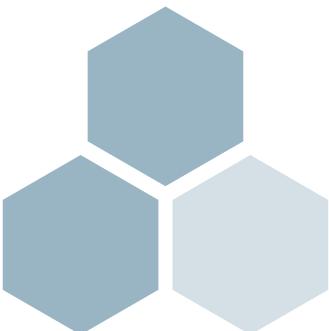


5

# Aplicação do Redux



Atividade



## A – Criando o cadastro de contatos baseado no Redux

A parte referente à inclusão de contatos será realizada pelo Redux. Os tópicos seguintes nos guiarão nesta tarefa.

1. Baixe as dependências do Redux:

```
npm i --save-dev redux react-redux
```

2. Crie uma pasta chamada **reducers** e, nesta pasta, crie o arquivo **reducers.js** (seu conteúdo será criado provisoriamente, apenas para termos um ponto de partida):

```
import { combineReducers } from "redux";

const reducers = combineReducers({
    contato: () => ({
        data: '2018-05-20',
        nome: 'Jose',
        email: 'jose@impacta.com.br',
        assunto: 'Desejo saber mais informações'
    })
});

export default reducers;
```

3. No arquivo **index.jsx**, faça as alterações sugeridas:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './main/app';
import { BrowserRouter } from 'react-router-dom';

import { createStore } from "redux";
import { Provider } from "react-redux";
import reducers from './reducers/reducers';

const store = createStore(reducers);

ReactDOM.render(
    <BrowserRouter>
        <Provider store={store}>
            <App />
        </Provider>
    </BrowserRouter>
), document.getElementById('app'));
```

Esse processo não deve interferir na execução da aplicação, pois o que fizemos foi estabelecer a ponte react-redux, mas nenhuma alteração foi realizada ainda.

4. Para testar a funcionalidade, defina os arquivos **contato.jsx** e **contatoForm.jsx**:

- **contato.jsx**

```
import React, { Component } from 'react';
import Cabecalho from '../menu/cabecalho';
import ContatoForm from './contatoForm';

export default class Contato extends Component {
  constructor(props) {
    super(props);
    this.state = {
      data: '',
      nome: '',
      email: '',
      assunto: ''
    };
  }

  render() {
    return (
      <div className="container">
        <Cabecalho titulo="Contato"
        subtitulo="entre em contato conosco" />
        <ContatoForm
          data={this.state.data}
          nome={this.state.nome}
          email={this.state.email}
          assunto={this.state.assunto}>
        </ContatoForm>
      </div>
    );
  }
}
```

- **contatoForm.jsx**

```
import React from "react";
import { connect } from "react-redux";

const ContatoForm = props => {

    return (
        <div>
            <h3 className="border-bottom">Formulario</h3>
            <form>
                <div className="form-group row">
                    <label htmlFor="data"
                        className="col-sm-3 col-form-label">Data:</label>
                    <div className="col-sm-5 col-6">
                        <input type="date"
                            className="form-control" id="data"
                            value={props.data} />
                    </div>
                </div>

                <div className="form-group row">
                    <label htmlFor="nome"
                        className="col-sm-3 col-form-label">Nome:</label>
                    <div className="col-sm-9">
                        <input type="text"
                            className="form-control" id="nome"
                            value={props.nome} />
                    </div>
                </div>

                <div className="form-group row">
                    <label htmlFor="email"
                        className="col-sm-3 col-form-label">Email:</label>
                    <div className="col-sm-9">
                        <input type="email"
                            className="form-control" id="email"
                            value={props.email} />
                    </div>
                </div>

                <div className="form-group row">
                    <label htmlFor="assunto"
                        className="col-sm-3 col-form-label">Assunto:</label>
                    <div className="col-sm-9">
                        <textarea className="form-control"
                            id="assunto" rows="5"
                            value={props.assunto} />
                    </div>
                </div>

                <div className="form-group row">
```

```
        <button className="btn btn-primary ml-3 mb-3">
          Adicionar
        </button>
      </div>
    </form>
  </div>
)
}
//padrão decorador
const mapStateToProps = state => ({
  data: state.contato.data,
  nome: state.contato.nome,
  email: state.contato.email,
  assunto: state.contato.assunto
});

export default connect(mapStateToProps)(ContatoForm);
```

Conseguimos visualizar os dados fixos que definimos no arquivo **reducers.js**, mas não conseguimos alterar nada porque o estado é fixo. Nas próximas etapas, vamos aplicar completamente o Redux.

5. Crie uma pasta chamada **actions** e, nesta pasta, um novo arquivo chamado **contatoActions.js**. Neste arquivo, teremos todos os nossos actions a serem disparados pelos eventos:

```
//para os campos de entrada
export const dataInput = e => ({
  type: 'DATA_INPUT',
  info: e.target.value
});

export const nomeInput = e => ({
  type: 'NOME_INPUT',
  info: e.target.value
});

export const emailInput = e => ({
  type: 'EMAIL_INPUT',
  info: e.target.value
});

export const assuntoInput = e => ({
  type: 'ASSUNTO_INPUT',
  info: e.target.value
});
```

6. Na pasta **reducers**, crie o arquivo **contatoReducer.js**. Ele serve para evoluir os estados de acordo com o evento disparado por cada campo de entrada. Usará também, o action para listar os cursos:

```
const INITIAL_STATE = {
  data: '',
  nome: '',
  email: '',
  assunto: ''
}

export default (state = INITIAL_STATE, action) => {
  switch (action.type) {
    case 'DATA_INPUT': return { ...state, data: action.info }
    case 'NOME_INPUT': return { ...state, nome: action.info }
    case 'EMAIL_INPUT': return { ...state, email: action.info }
    case 'ASSUNTO_INPUT': return { ...state, assunto: action.info }

    default: return state;
  }
}
```

7. Vamos tornar este componente como sendo um componente de classe. Observe as alterações no arquivo **contatoForm.jsx**:

```
import React, { Component } from "react";
import { connect } from "react-redux";

import { bindActionCreators } from 'redux';
import {
  dataInput,
  nomeInput,
  emailInput,
  cursoInput,
  assuntoInput,
  buscarCursos
} from '../actions/contatoActions';

class ContatoForm extends Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div>
        <h3 className="border-bottom">Formulario</h3>
        <form>
          <div className="form-group row">
            <label htmlFor="data" className="col-sm-3 col-form-label">Data:</label>
```

```
<div className="col-sm-5 col-6">
    <input type="date"
        className="form-control" id="data"
        value={this.props.data}
        onChange={this.props.dataInput} />
</div>
</div>

<div className="form-group row">
    <label htmlFor="nome"
        className="col-sm-3 col-form-label">Nome:</label>
    <div className="col-sm-9">
        <input type="text"
            className="form-control" id="nome"
            value={this.props.nome}
            onChange={this.props.nomeInput} />
    </div>
</div>

<div className="form-group row">
    <label htmlFor="email"
        className="col-sm-3 col-form-label">Email:</label>
    <div className="col-sm-9">
        <input type="email"
            className="form-control" id="email"
            value={this.props.email}
            onChange={this.props.emailInput} />
    </div>
</div>

<div className="form-group row">
    <label htmlFor="assunto"
        className="col-sm-3 col-form-label">Assunto:</label>
    <div className="col-sm-9">
        <textarea
            className="form-control" id="assunto" rows="5"
            value={this.props.assunto}
            onChange={this.props.assuntoInput} />
    </div>
</div>

<div className="form-group row">
    <button className="btn btn-primary ml-3 mb-3">
        Adicionar
    </button>
</div>
</form>
</div>
)
}
}
```

```
//padrão decorator
const mapStateToProps = state => ({
    data: state.contato.data,
    nome: state.contato.nome,
    email: state.contato.email,
    assunto: state.contato.assunto
});

const mapDispatchToProps = dispatch => bindActionCreators({
    dataInput,
    nomeInput,
    emailInput,
    assuntoInput}, dispatch)

export default connect(mapStateToProps, mapDispatchToProps)(ContatoForm);
```

8. Para adicionar um novo contato, vamos realizar algumas alterações nos arquivos: **contatoActions.js**, **contatoReducer.js** e **contatoForm.jsx**:

- **contatoActions.js**

```
import axios from 'axios';
const URL = "http://localhost:3200/ws/contatos";

//para os campos de entrada
export const dataInput = e => ({
    type: 'DATA_INPUT',
    info: e.target.value
});

export const nomeInput = e => ({
    type: 'NOME_INPUT',
    info: e.target.value
});

export const emailInput = e => ({
    type: 'EMAIL_INPUT',
    info: e.target.value
});

export const cursoInput = e => ({
    type: 'CURSO_INPUT',
    info: e.target.value
});

export const assuntoInput = e => ({
    type: 'ASSUNTO_INPUT',
    info: e.target.value
});
```

```
//para os eventos dos botões
```

```
export const adicionarContato = (data, nome, email, assunto) => {
    const request = axios.post(URL, { data, nome, email, assunto });
    return {
        type: 'ADICIONAR_CONTATO',
        info: request
    }
};
```

- **contatoReducer.js**

```
const INITIAL_STATE = {
    data: '',
    nome: '',
    email: '',
    assunto: ''
}

export default (state = INITIAL_STATE, action) => {
    switch (action.type) {
        case 'DATA_INPUT': return { ...state, data: action.info }
        case 'NOME_INPUT': return { ...state, nome: action.info }
        case 'EMAIL_INPUT': return { ...state, email: action.info }
        case 'ASSUNTO_INPUT': return { ...state, assunto: action.info }

        case 'ADICIONAR_CONTATO': return { ...state, data: '', nome: '',
            email: '', assunto: '' }

        default: return state;
    }
}
```

- **contatoForm.jsx**

```
import React, { Component } from "react";
import { connect } from "react-redux";

import { bindActionCreators } from 'redux';
import {
    dataInput,
    nomeInput,
    emailInput,
    assuntoInput,
    adicionarContato
} from '../actions/contatoActions';

class ContatoForm extends Component {
    constructor(props) {
        super(props);
        this.adicionar = this.adicionar.bind(this);
    }
}
```

# Desenvolvimento de Componentes Web e Mobile com React.Js, Redux e React Native

```
adicionar(e) {
    //destructuring
    const { adicionarContato, data, nome, email, assunto } = this.props;

    e.preventDefault();
    adicionarContato(data, nome, email, assunto);
    alert('Contato adicionado com sucesso');
}

render() {

    //novo no es6 - destructuring
    const { data, nome, email, assunto, dataInput, nomeInput,
emailInput, assuntoInput } = this.props;

    return (
        <div>
            <h3 className="border-bottom">Formulario</h3>
            <form>
                <div className="form-group row">
                    <label htmlFor="data" className="col-sm-3 col-form-label">Data:</label>
                    <div className="col-sm-5 col-6">
                        <input type="date" className="form-control" id="data"
value={data} onChange={dataInput} />
                    </div>
                </div>

                <div className="form-group row">
                    <label htmlFor="nome" className="col-sm-3 col-form-label">Nome:</label>
                    <div className="col-sm-9">
                        <input type="text" className="form-control" id="nome"
value={nome} onChange={nomeInput} />
                    </div>
                </div>

                <div className="form-group row">
                    <label htmlFor="email" className="col-sm-3 col-form-label">Email:</label>
                    <div className="col-sm-9">
                        <input type="email" className="form-control" id="email"
value={email} onChange={emailInput} />
                    </div>
                </div>

                <div className="form-group row">
                    <label htmlFor="assunto" className="col-sm-3 col-form-label">Assunto:</label>
                    <div className="col-sm-9">
                        <textarea className="form-control" id="assunto"
rows="5"
value={assunto} onChange={assuntoInput} />
                    </div>
                </div>
            </form>
        </div>
    )
}
```

```
        <div className="form-group row">  
  
            <button className="btn btn-primary ml-3 mb-3"  
                   onClick={this.adicionar}>Adicionar</button>  
        </div>  
    </form>  
  </div>  
)  
}  
  
//padrão decorador  
const mapStateToProps = state => ({  
  data: state.contato.data,  
  nome: state.contato.nome,  
  email: state.contato.email,  
  assunto: state.contato.assunto  
});  
  
const mapDispatchToProps = dispatch => bindActionCreators({  
  dataInput,  
  nomeInput,  
  emailInput,  
  assuntoInput,  
  adicionarContato  
, dispatch)  
  
export default connect(mapStateToProps, mapDispatchToProps)(ContatoForm);
```

9. Teste o novo cadastro. Visualize os contatos no browser ou usando o Postman.



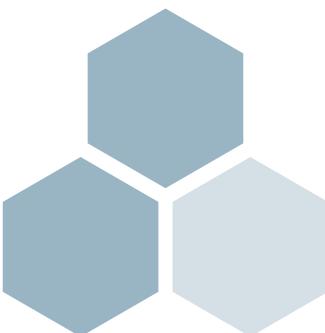


6

# Aplicação do React Native



Atividade



## A – Criando uma aplicação para ser executada de forma emulada

Este é o modo mais tranquilo de criarmos e rodarmos uma aplicação, se o propósito é testá-la e só posteriormente transferi-la para o dispositivo real. É esta abordagem que usaremos nesta aula.

1. Vamos instalar o componente gerador do projeto. Execute o comando:

```
npm install -g create-react-native-app
```

2. Entre na pasta por meio do prompt de comandos. Execute a instrução:

```
create-react-native-app appReactNative
```

3. Será adicionada uma pasta chamada contendo o conteúdo do projeto. Vamos, antes de mais nada, testar o ambiente. Execute o comando:

```
npm start
```

4. Quando a aplicação for iniciada, um QR Code será gerado, o que permitirá, por meio do aplicativo , sua leitura e, consequentemente, seu sincronismo. Um exemplo de execução é mostrado a seguir:



5. Se houver discrepância no IP da rede, vale a pena executar o comando no prompt:

```
set REACT_NATIVE_PACKAGER_HOSTNAME=127.0.0.1
```

Certifique-se que o host anterior seja o que você está utilizando.

## B – Desenvolvendo a aplicação

1. Abra o VSCode na pasta do novo projeto () ;
2. Abra o arquivo gerado na raiz chamado . Analise seu conteúdo e compare com o resultado apresentado no emulador:

```
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

export default class App extends React.Component {
  render() {
    return (
      <View style={styles.container}>
        <Text>Open up App.js to start working on your app!</Text>
        <Text>Changes you make will automatically reload.</Text>
        <Text>Shake your phone to open the developer menu.</Text>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center'
  },
});
```

3. Nossa aplicativo deverá permitir acesso a duas telas (páginas) distintas. Para isso, definiremos rotas apontando para essas telas. Inclua o componente :

```
npm i react-navigation --save
```

4. Altere o conteúdo de :

```
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

import { createStackNavigator } from 'react-navigation';

class PaginaInicial extends React.Component {
  render() {
    return (
      <View style={styles.container}>
        <Text style={styles.texto}>Página inicial</Text>
      </View>
    );
  }
}
```

# Desenvolvimento de Componentes Web e Mobile com React.Js, Redux e React Native

```
const RootStack = createStackNavigator({
  Home: {
    screen: PaginaInicial
  }
});

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  texto: {
    flex: 1,
    fontSize: 20
  }
});

export default class App extends React.Component {
  render() {
    return <RootStack />
  }
}
```

5. Vamos, agora, criar uma naveabilidade entre telas. Para tanto, crie uma pasta no projeto chamada . Nessa pasta, inclua os arquivos e . No arquivo , adicione este conteúdo:

```
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

export default class ListaCursos extends React.Component {
  render() {
    return (
      <View>
        <Text style={styles.texto}>Lista de Cursos</Text>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  texto: {
    alignItems: "center",
    fontSize: 30,
    margin: 20,
    color: '#FF0000'
  }
});
```

6. Retorne ao arquivo , importe esse novo componente e faça as alterações sugeridas:

```
import React from 'react';
import { StyleSheet, Text, View, Button } from 'react-native';

import { createStackNavigator } from 'react-navigation';
import ListaCursos from './routes/cursos';

class PaginaInicial extends React.Component {
  render() {
    return (
      <View>
        <View style={styles.container}>
          <Text style={styles.texto}>Página inicial</Text>
        </View>
        <View style={styles.botao}>
          <Button
            title="Lista de Cursos"
            onPress={() => this.props.navigation.navigate('Cursos')}
          />
        </View>
      );
    }
}

const RootStack = createStackNavigator({
  Home: {
    screen: PaginaInicial
  },
  Cursos: {
    screen: ListaCursos
  }
});

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  texto: {
    flex: 1,
    fontSize: 20
  },
  botao: {
    width: '100%',
    height: '30%',
    justifyContent: 'center',
    alignItems: 'center'
  }
});

export default class App extends React.Component {
  render() {
    return <RootStack />
  }
}
```

## C – Acessando serviços REST

O nosso componente será responsável por acessar o nosso Web service e listar os cursos disponíveis. A lista de cursos será exibida em uma lista.

1. Altere o arquivo para refletir essa nova funcionalidade:

```
import React from 'react';
import { StyleSheet, Text, View, SafeAreaView, FlatList } from 'react-native';

export default class ListaCursos extends React.Component {
    state = {
        text: '',
        data: []
    }

    //função do ciclo de vida do react: executa logo após a carga do componente
    //https://reactjs.org/docs/react-component.html
    async componentDidMount() {
        let response = await
            fetch('http://192.168.0.14:3200/ws/cursos');
        let responseJson = await response.json();
        this.setState({ data: responseJson, isLoading: false });
    }

    render() {
        return (
            <View>
                <Text style={styles.texto}>Lista de Cursos</Text>

                <View>
                    <SafeAreaView>
                        <FlatList
                            data={this.state.data}
                            renderItem={({ item }) => {
                                return (
                                    <View style={styles.item}>
                                        <Text style={styles.conteudo}>
                                            {item.descricao}</Text>
                                    </View>
                                )
                            }}
                            keyExtractor={(item, index) =>
                                index.toString()}
                        />
                    </SafeAreaView>
                </View>
            </View>
        );
    }
}
```

```
const styles = StyleSheet.create({
  conteudo: {
    fontSize: 15,
    color: "#333333"
  },
  texto: {
    alignItems: "center",
    fontSize: 30,
    margin: 20,
    color: '#FF0000',
  },
  item: {
    alignItems: "center",
    backgroundColor: "#dcda44",
    flexGrow: 1,
    margin: 4,
    padding: 20
  }
});
```

2. Neste ponto, a lista de cursos deve estar visível. O objetivo, agora, é permitir que, quando o usuário tocar no nome do curso, seus detalhes apareçam em uma caixa de mensagens. Para isso, realize as seguintes alterações em :

```
import React from 'react';
import { StyleSheet, Text, View, SafeAreaView, FlatList, TouchableOpacity } from 'react-native';

export default class ListaCursos extends React.Component {
  state = {
    text: '',
    data: []
  }

  //função do ciclo de vida do react: executa logo após a carga do componente
  //https://reactjs.org/docs/react-component.html
  async componentDidMount() {
    let response = await
      fetch('http://192.168.0.14:3200/ws/cursos');
    let responseJson = await response.json();
    this.setState({ data: responseJson, isLoading: false });
  }

  exibirDetalhes(item) {
    alert(
      "Código: " + item.codigo +
      "\nDescrição: " + item.descricao +
      "\nCH: " + item.cargaHoraria +
      "\nPreço: " + item.preco +
      "\nCategoria: " + item.categoria);
  }
}
```

# Desenvolvimento de Componentes Web e Mobile com React.Js, Redux e React Native

```
render() {
    return (
        <View>
            <Text style={styles.texto}>Lista de Cursos</Text>

            <View>
                <SafeAreaView>
                    <FlatList
                        data={this.state.data}
                        renderItem={({ item }) => {
                            return (
                                <TouchableOpacity onPress={() =>
                                    this.exibirDetalhes(item)}>
                                    <View style={styles.item}>
                                        <Text
                                            style={styles.conteudo}>
                                                {item.descricao}</Text>
                                        </View>
                                </TouchableOpacity>
                            )
                        }
                    keyExtractor={(item, index) =>
                        index.toString()}
                />
            </SafeAreaView>
        </View>
    );
}

const styles = StyleSheet.create({
    conteudo: {
        fontSize: 15,
        color: "#333333"
    },
    texto: {
        alignItems: "center",
        fontSize: 30,
        margin: 20,
        color: '#FF0000',
    },
    item: {
        alignItems: "center",
        backgroundColor: "#dcda44",
        flexGrow: 1,
        margin: 4,
        padding: 20
    }
});
```

3. Agora vamos acrescentar os recursos para exibir a lista de contatos. Vamos definir o seguinte conteúdo para o arquivo :

```
import React from 'react';
import { StyleSheet, Text, View, FlatList, SafeAreaView } from 'react-native';

export default class ListaContatos extends React.Component {

    state = {
        text: '',
        data: []
    }

    async componentDidMount() {
        let response = await
            fetch('http://192.168.0.14:3200/ws/contatos');
        let responseJson = await response.json();
        this.setState({ data: responseJson, isLoading: false });
    }

    render() {
        return (
            <View>
                <Text style={styles.texto}>Lista de Contatos</Text>

                <View>
                    <SafeAreaView>
                        <FlatList
                            data={this.state.data}
                            renderItem={({ item }) => {
                                return (
                                    <View style={styles.item}>
                                        <Text style={styles.conteudo}>
                                            {item.data}</Text>
                                        <Text style={styles.conteudo}>
                                            {item.nome}</Text>
                                        <Text style={styles.conteudo}>
                                            {item.email}</Text>
                                        <Text style={styles.conteudo}>
                                            {item.assunto}</Text>
                                    </View>
                                )
                            }
                            keyExtractor={(item, index) =>
                                index.toString()
                            }
                        />
                    </SafeAreaView>
                </View>
            );
    }
}
```

```
const styles = StyleSheet.create({
  conteudo: {
    margin: 10,
    borderWidth: 1,
    backgroundColor: 'white',
    fontSize: 20
  },
  texto: {
    alignItems: "center",
    fontSize: 30,
    margin: 20,
    color: '#FF0000',
  },
  item: {
    alignItems: "center",
    backgroundColor: "#dcda44",
    flexGrow: 1,
    margin: 4,
    padding: 20
  },
  conteudo: {
    color: "#333333"
  }
});
```

4. Agora vamos realizar as alterações sugeridas no arquivo :

```
import React from 'react';
import { StyleSheet, Text, View, Button } from 'react-native';

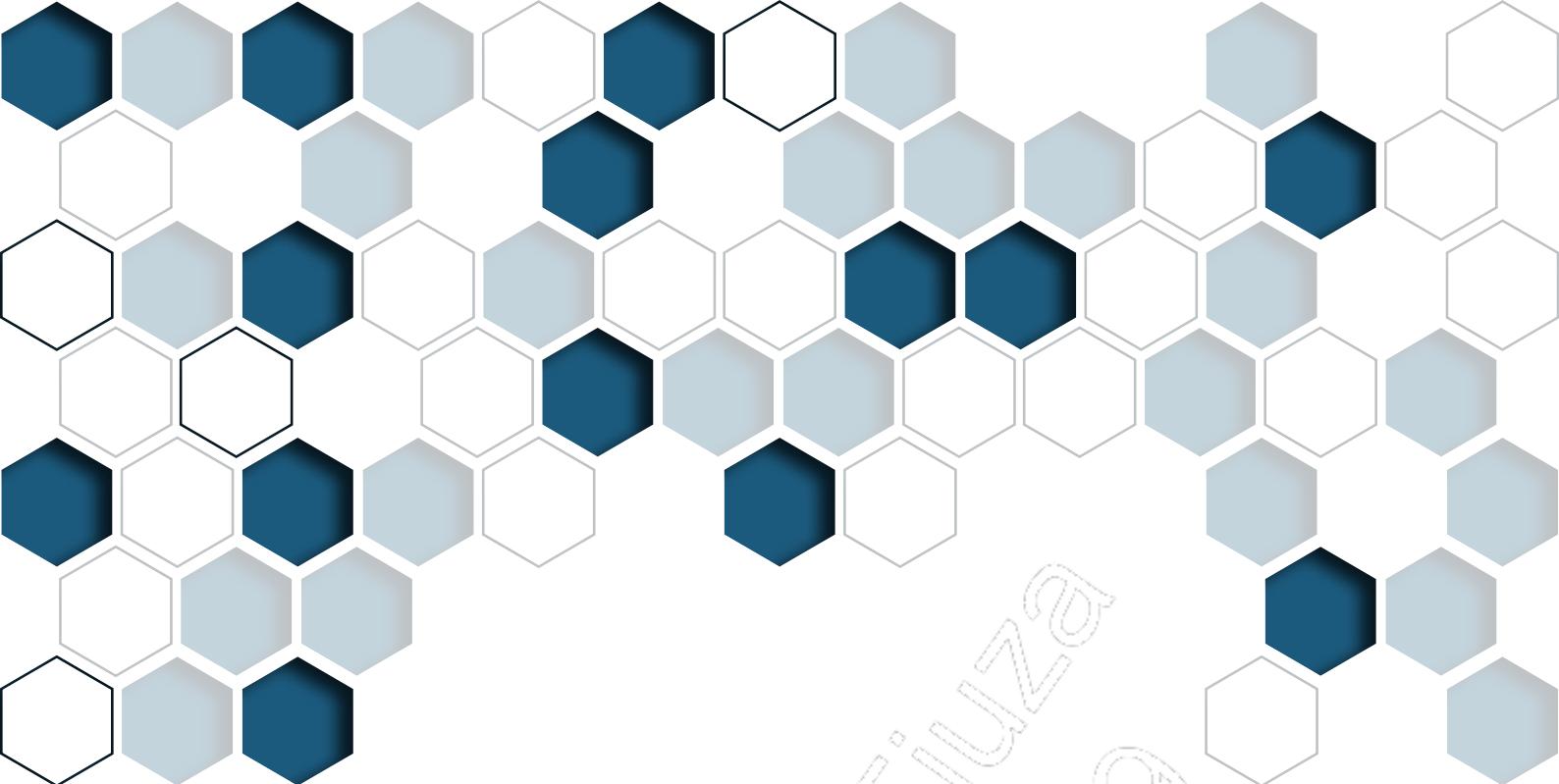
import { createStackNavigator } from 'react-navigation';
import ListaCursos from './routes/cursos';
import ListaContatos from './routes/contatos';

class PaginaInicial extends React.Component {
  render() {
    return (
      <View>
        <View style={styles.conteudo}>
          <Text style={styles.texto}>Página inicial</Text>
        </View>
        <View style={styles.botao}>
          <Button
            title="Lista de Cursos"
            onPress={() => this.props.navigation.navigate('Cursos')} />
        </View>
        <View style={styles.botao}>
          <Button
            title="Lista de Contatos"
            onPress={() => this.props.navigation.navigate('Contatos')} />
        </View>
      </View>
    );
  }
}
```

```
const RootStack = createStackNavigator({  
  Home: {  
    screen: PaginaInicial  
  },  
  Cursos: {  
    screen: ListaCursos  
  },  
  Contatos: {  
    screen: ListaContatos  
  }  
});  
  
const styles = StyleSheet.create({  
  container: {  
    flex: 1,  
    backgroundColor: '#fff',  
    alignItems: 'center',  
    justifyContent: 'center',  
  },  
  texto: {  
    flex: 1,  
    fontSize: 20  
  },  
  botao: {  
    width: '100%',  
    height: '30%',  
    justifyContent: 'center',  
    alignItems: 'center'  
  }  
});  
  
export default class App extends React.Component {  
  render() {  
    return <RootStack />  
  }  
}
```

5. Para finalizar, realize as alterações nos estilos ou no formato da apresentação que julgar necessários.

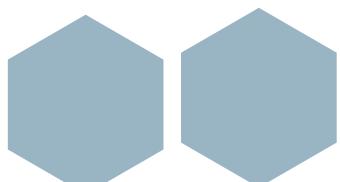




# Mãos à obra!



Editora  
**IMPACTA**







# Criando um projeto com React.js

Mãos à obra!



## A – Criando uma agenda de contatos com React.js, consumindo um serviço baseado no ExpressJS e no MongoDB

O nosso projeto conterá a seguinte funcionalidade:

- Uma página inicial apresenta um menu com a opção **Compromisso**;
- Acessando **Compromisso**, o usuário deve ser direcionado para um formulário contendo campos de entrada para inserir um novo compromisso (os dados devem ser: DATA, DESCRIÇÃO, RESPONSÁVEL e TELEFONE) e para a lista de todos os compromissos;
- As informações do registro deverão ser armazenadas em um banco de dados **MongoDB**.

Para o desenvolvimento deste laboratório, siga as etapas apresentadas adiante:

1. Crie uma pasta chamada **Laboratorios**;
2. Nessa pasta, crie uma subpasta chamada **Lab\_REST**. Essa pasta conterá o Web service a ser manipulado pela aplicação;
3. Crie, também, uma subpasta chamada **Lab01\_React**, abaixo de **Laboratorios**;
4. Abra a pasta **Lab\_REST** no VSCode;
5. No prompt de comandos, acesse essa pasta;
6. Na pasta **Lab-REST**, crie o serviço para permitir o armazenamento e o consumo de um compromisso. As informações para elaboração do modelo são: DATA, DESCRIÇÃO, RESPONSÁVEL e TELEFONE;
7. Usando o Postman, inclua alguns compromissos e teste a consulta;
8. Agora, abra a pasta **Lab01\_React** no VSCode;
9. Seguindo o roteiro apresentado nos projetos, elabore uma aplicação contendo: uma página inicial, o componente para cadastro de um novo compromisso e outro para a listagem de compromissos. Crie as rotas adequadamente para cada situação.



2

# Criando um projeto com Redux

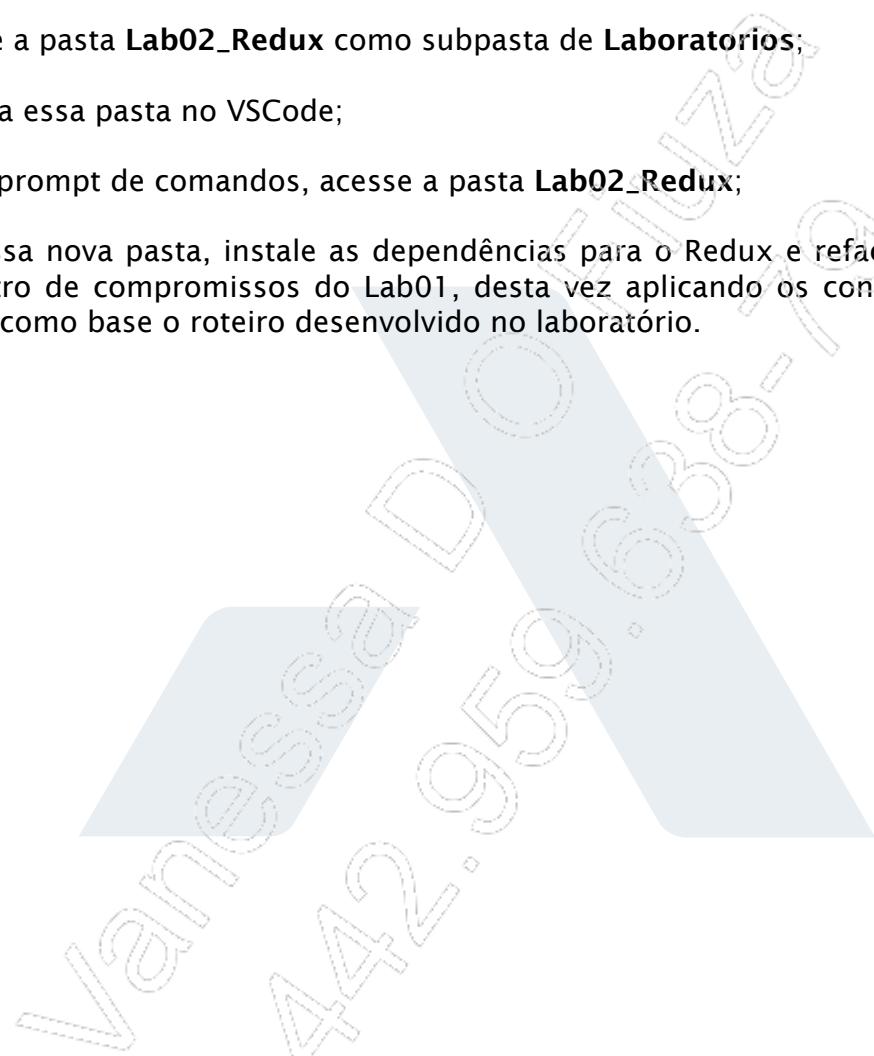


Mãos à obra!

Vanessa Fiuza  
1979



## A – Criando a aplicação de cadastro de compromissos com Redux

1. Crie a pasta **Lab02\_Redux** como subpasta de **Laboratorios**;
  2. Abra essa pasta no VSCode;
  3. No prompt de comandos, acesse a pasta **Lab02\_Redux**;
  4. Nessa nova pasta, instale as dependências para o Redux e refaça a aplicação de cadastro de compromissos do Lab01, desta vez aplicando os conceitos do Redux. Tome como base o roteiro desenvolvido no laboratório.
- 



3

# Criando um projeto com React Native



Mãos à obra!



## A – Criando a aplicação de cadastro de compromissos com React Native

1. Crie a pasta **Lab03\_ReactNative** como subpasta de **Laboratorios**;
  2. Abra essa pasta no VSCode;
  3. No prompt de comandos, acesse a pasta **Lab03\_ReactNative**;
  4. Nessa nova pasta, instale as dependências para o React Native;
  5. Elabore a aplicação de cadastro de compromissos dos iabs Lab01 e Lab02, desta vez aplicando os conceitos do React Native;
  6. Nessa nova aplicação, exiba a lista de compromissos.
- 