

# React Native

Projeto de Interfaces de Dispositivos Móveis

**Entrada de Parâmetros,  
Apresentação de Dados,  
Networking  
Aula 02**

# Introdução

- **Uso de Text Input**
- **Uso de Touches (toques)**
- **Scrolling**
- **List**

# <TextInput>

- É um componente básico para entrada de dados.
- A propriedade **onChangeText** recebe uma função toda vez que o texto é modificado.
- Vejamos o exemplo a seguir:

# Calculadora IMC (Ex. 1)


- Entra com o peso `<TextInput>`
- Entra com a altura `<TextInput>`
- Calcula o IMC ( $\text{peso} / \text{altura} * \text{altura}$ ) em um componente separado, passando a altura e peso como **props** desse componente.

# Calculadora IMC (Ex. 1)

```
import React, { Component } from 'react';
import { AppRegistry, Text, TextInput, View } from 'react-native';

class Calculadora extends Component{
  constructor(props){
    super(props);
  }
  render(){
    if(!this.props.altura || !this.props.peso) return null;
    return (
      <Text style={{padding: 10, fontSize: 20}}>
        IMC: {this.props.peso/(this.props.altura*this.props.altura)}
      </Text>
    )
  }
}
```

Recebe altura e peso do componente pai. São props.



```
export default class CalculadoraApp extends Component {
  constructor(props) {
    super(props);
    this.state = {altura: null, peso: null};
  }

  render() {
    return (
      <View style={{padding: 10}}>
        <TextInput
          style={{height: 40}}
          placeholder="Digite a altura"
          onChangeText={(altura) => this.setState({altura})}
        />
        <TextInput
          style={{height: 40}}
          placeholder="Digite o peso"
          onChangeText={(peso) => this.setState({peso})}
        />
        <Calculadora altura={this.state.altura} peso={this.state.peso} />
      </View>
    );
  }
}
```

```
AppRegistry.registerComponent('AwesomeProject', () => CalculadoraApp);
```

Chama o componente filho “Calculadora”, passado como parâmetro o peso e a altura.

Nesse componente, peso e altura são states (mudam com um determinado evento)!

# onChangeText

- recebe uma função como parâmetro.
  - Ex.: (altura) => `this.setState({altura})`
  - Como queremos mudar uma propriedade do estados, usamo o método `setState`, que também recebe uma função como parâmetro.
  - No caso, estamos passando apenas `{altura}` como parâmetro para `setState`. Essa é uma forma simplificada de escrever:

```
this.setState(  
  ()=>{  
    return {altura:altura}  
  }  
)
```

“altura” vindo do state do componente

“altura” vindo da função de entrada para `onChangeText`.

# onChangeText - setState

- Por que usar o **setState**? Eu poderia fazer o código abaixo:

```
onChangeText={
  (altura) => {
    this.state.altura = altura;
  }
}
```

- Teoricamente, o resultado seria o mesmo?



# onChangeText - setState

- O **setState** infileira mudanças no componente e diz ao React que o componente deve ser **atualizado** (renderizado novamente);
- O **setState** é **assíncrono**, o uso de `this.state`, apesar de parecer funcionar, não garante que é o estado **mais atual** da variável `state`. O correto é usar a variável “`prevState`” e retornar um novo objeto.
- Assinatura: `setState(updater[, callback])`
- Onde `updater` é uma função: `(prevState, props) => stateChange`

# onChangeText - setState

- Exemplo de uso:

```
this.setState((prevState, props) => {  
  return {counter: prevState.counter + props.step};  
})
```

*“Due to the async nature of **setState**, it is **not advisable** to use **this.state** to get the previous state within **setState**. Instead, always rely on the above way. Both **prevState** and **props** received by the **updater** function are **guaranteed** to be up-to-date. The output of the updater is merged with **prevState**.”*

<https://itnext.io/react-setstate-usage-and-gotchas-ac10b4e03d60>

# react-number-format

- <https://www.npmjs.com/package/react-number-format>
- **npm install react-number-format --save**
- No arquivo js:
  - import NumberFormat from 'react-number-format';

```
<NumberFormat  
  value={this.props.peso/(this.props.altura*this.props.altura)}  
  decimalScale='2'  
  displayType='text' />
```

# onChangeText - setState

- Resumindo:

```
onChangeText={
  (h) => {
    this.state.altura = h;
  } //ERRADO
}
```

```
onChangeText={
  (h) =>
    this.setState(
      ()=>{
        this.state.altura = h;
        return this.state;
      }
    ) //ERRADO
}
```

```
onChangeText={
  (h) =>
    this.setState(
      ()=>{
        return {altura:h};
      }
    ) //CORRETO
}
```

```
onChangeText={
  (h) => this.setState({altura:h}) //CORRETO
}
```

```
onChangeText={
  (altura) => this.setState({altura}) //CORRETO
}
```

# Exercício

- Crie um novo componente, chamado **IMCMensagem**, que obedece a seguinte regra:

Resultado	Situação
Abaixo de 17	Muito abaixo do <i>peso</i>
Entre 17 e 18,49	Abaixo do <i>peso</i>
Entre 18,5 e 24,99	<i>Peso</i> normal
Entre 25 e 29,99	Acima do <i>peso</i>
Entre 30 e 34,99	<i>Obesidade</i> I
Entre 35 e 39,99	<i>Obesidade</i> II (severa)
Acima de 40	<i>Obesidade</i> III (mórbida)

- Esse novo componente deve ter um props “imc”, o qual será passado a ele como parâmetro. Daí, ele irá decidir qual mensagem mostrar.

```

class IMCMensagem extends Component{
  constructor(props){
    super(props);
  }
  render(){
    if(!this.props.imc) return null;
    if(this.props.imc<25) return (<Text>Magro</Text>);
    return (<Text>Gordo</Text>);
  }
}

class Calculadora extends Component{
  constructor(props){
    super(props);
  }
  render(){
    if(!this.props.altura || !this.props.peso) return null;
    let imc = this.props.peso/(this.props.altura*this.props.altura);
    return (
      <View>
        <Text style={{padding: 10, fontSize: 20}}>
          IMC:{imc}
        </Text>
        <IMCMensagem imc={imc}/>
      </View>
    );
  }
}

```

**Resposta parcial...**

**Você deve fazer para todos os casos.**

# Tradutor para Pizza (Ex. 2)

```
import React, { Component } from 'react';
import { AppRegistry, Text, TextInput, View } from 'react-native';
```

```
export default class TradutorDePizza extends Component {
  constructor(props) {
    super(props);
    this.state = {text: ""};
  }
```

→ Começa um state com a propriedade “text”, vazia.

```
  render() {
    return (
      <View style={{padding: 10}}>
        <TextInput
          style={{height: 40}}
          placeholder="Digite para traduzir!"
          onChangeText={(text) => this.setState({text})}
          value={this.state.text}
        />
        <Text style={{padding: 10, fontSize: 22}}>
          {this.state.text.split(' ').map((palavra) => palavra && '🍕').join(' ')}
        </Text>
      </View>
    );
  }
}
```

→ Entrada de dados.

O método “split” retorna um array a partir da string de entrada (no caso “text”). O separador usado como parâmetro é o ‘ ‘ (espaço).

Neste array (resultante de ‘text’), é chamado o método map. O método map implementa um laço sobre o array e retorna o array, que pode ter sido modificado pela função de entrada. No caso (palavra) => palavra && '🍕'

A função (palavra) => palavra && '🍕', substitui cada elemento do array (palavra) pelo caracter '🍕'. O && é usado caso “palavra” seja null. Daí não escreve nada.

Por último, sobre esse array de '🍕' resultante, é chamado o método join, que retorna uma string separando os elementos do array por ' ' (espaço).

<https://emojipedia.org/slice-of-pizza/>

```
AppRegistry.registerComponent('ProjetoInicial', () => TradutorDePizza);
```

# Botões

- Botões são componentes básicos de interação os quais são renderizados de forma satisfatória, em qualquer plataforma. O exemplo mínimo de um botão é o seguinte (não funciona no navegador, use o Expo App):

```
<Button
  onPress={() => {
    Alert.alert('You tapped the button!');
  }}
  title="Press Me"
/>
```



# Botões

- **onPress = {<função>}**
  - Recebe uma função a qual irá ser chamada caso o botão seja pressionado. É a ação do botão.
- **title = <título do botão>**
  - Recebe uma string que representa o título do botão.

# Botões

- Crie o seguinte layout:

```
<View estilo1>  
  <View estilo2>  
    <Button/>  
  </View>  
  <View estilo2>  
    <Button/>  
  </View>  
  <View estilo3>  
    <Button/>  
    <Button/>  
  </View>  
</View>
```

#841584



# Botões (outro exemplo)

```
import React, { Component } from 'react';
import { Alert, AppRegistry, Button, StyleSheet, View } from 'react-native';
```

```
export default class ButtonBasics extends Component {
  acaoBotao() {
    Alert.alert("Você apertou o botão!!")
  }
}
```

```
render() {
  return (
    <View style={styles.container}>
      <View style={styles.buttonContainer}>
        <Button
          onPress={this.acaoBotao}
          title="Me Aperte"
        />
      </View>
      <View style={styles.buttonContainer}>
        <Button
          onPress={this.acaoBotao}
          title="Me Aperte"
          color="#841584"
        />
      </View>
    </View>
  );
}
```

```
<View style={styles.alternativeLayoutButtonContainer}>
  <Button
    onPress={this.acaoBotao}
    title="Que legal!"
  />
  <Button
    onPress={this.acaoBotao}
    title="OK!"
    color="#841584"
  />
</View>
</View>
);
}
```

# Botões (outro exemplo)

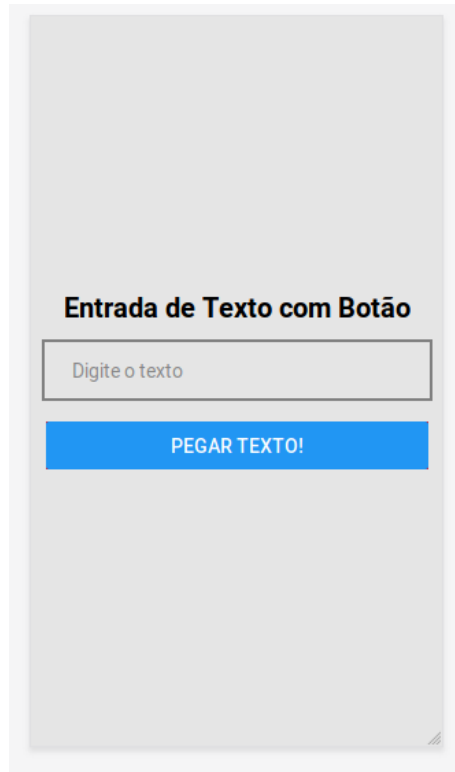
```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
  },
  buttonContainer: {
    margin: 20
  },
  alternativeLayoutButtonContainer: {
    margin: 20,
    flexDirection: 'row',
    justifyContent: 'space-between'
  }
});
```

```
// skip this line if using Create React Native App
AppRegistry.registerComponent('ProjetoInicial', () => ButtonBasics);
```

# Botões e Input

- Iremos implementar agora como recuperar um valor de um TextInput ao clicar em Botão e não apenas enquanto digita o valor (feito anteriormente com a ajuda do **onChangeText**).

# Botões e Input



```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#e5e5e5"
  },
  headerText: {
    fontSize: 20,
    textAlign: "center",
    margin: 10,
    fontWeight: "bold"
  },
  textInput: {
    height: 45,
    width: "95%",
    borderColor: "gray",
    borderWidth: 2,
    paddingLeft: 20
  },
  viewButton: {
    width: "93%",
    margin: 15,
    backgroundColor: "red"
  }
});
```

# Botões e Input

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

export default class BotaoInput extends Component {

  constructor(props) {
    super(props);
    this.state = { text: "" };
  }

  acaoCliqueBotao = ()=>{

  }

  render() {

  }

}

const styles = StyleSheet.create(...)
AppRegistry.registerComponent('ProjetoInicial', () => BotaoInput);
```

**Este é basicamente o template da aplicação:**

**constructor** : inicializa o state

**acaoCliqueBotao**: chamada ao clicar no btn

**render**: renderiza a nossa tela.

# Botões e Input

```
acaoCliqueBotao = ()=>{  
  const {text} = this.state;  
  Alert.alert(text);  
}
```

→ Cria uma variável local inicializada com o valor de text. Depois, mostrar essa variável em um Alert.

```
render() {  
  return (  
    <View style={styles.container}>  
      <Text style={styles.headerText}>  
        Entrada de Texto com Botão  
      </Text>  
      <TextInput  
        style={styles.textInput}  
        onChangeText={(text) => this.setState({ text })}  
        placeholder="Digite o texto"  
      />  
      <View style={styles.viewButton}>  
        <Button  
          title="Pegar Texto!"  
          onPress={this.acaoCliqueBotao}  
        />  
      </View>  
    </View>  
  );  
}
```

→ Modifica o valor de text, dentro de state.

→ chama o método acaoCliqueBotao



# Botões e Input

- Modifique a implementação do IMC para que só calcule o IMC quando for pressionado um botão. Use estilos para deixar o seu layout centralizado.

# Tocáveis

- É possível construir o próprio botão usando os “Touchable” do React Native. Os componentes “Touchable” são capazes de capturar gestos de toque e mostrar algum feedback quando um gesto é reconhecido.
- Esses componentes não tem nenhum estilo padrão. Você deverá de prover esse estilo se quiser ter algo minimamente agradável.

# Tocáveis

- **TouchableHighlight**: pode-se usar em qualquer lugar que queira um botão ou link. O background escurece ao ser pressionado.
- **TouchableNativeFeedback**: somente no Android. Usa características nativas para mostrar o feedback do toque.
- **TouchableOpacity**: reduz a opacidade do botão ao ser tocado, permitindo ver o background ao fundo enquanto o usuário continua pressionando o botão.
- **TouchableWithoutFeedback**: caso não queira nenhum feedback visual, use este tocável.
- Em alguns casos, você talvez queria que quando o usuário pressiona e segura o botão por um long período de tempo. Essas “pressionadas longas” podem ser tratadas passando uma função para o props “**onLongPress**” de qualquer “Touchable”.

```

import React, { Component } from 'react';
import { Alert, AppRegistry, Platform, StyleSheet, Text, TouchableHighlight, TouchableOpacity, TouchableNativeFeedback, TouchableWithoutFeedback,
View } from 'react-native';

export default class Touchables extends Component {
  _onPressButton() {
    Alert.alert("You tapped the button!")
  }

  _onLongPressButton() {
    Alert.alert("You long-pressed the button!")
  }

  render() {
    return (
      <View style={styles.container}>

        <TouchableHighlight onPress={this._onPressButton} underlayColor="white">
          <View style={styles.button}>
            <Text style={styles.buttonText}>TouchableHighlight</Text>
          </View>
        </TouchableHighlight>

        <TouchableOpacity onPress={this._onPressButton}>
          <View style={styles.button}>
            <Text style={styles.buttonText}>TouchableOpacity</Text>
          </View>
        </TouchableOpacity>
      </View>
    );
  }
}

```

```
<TouchableNativeFeedback
  onPress={this._onPressButton}
  background={Platform.OS === 'android' ? TouchableNativeFeedback.SelectableBackground() : ''}>
  <View style={styles.button}>
    <Text style={styles.buttonText}>TouchableNativeFeedback</Text>
  </View>
</TouchableNativeFeedback>
```

```
<TouchableWithoutFeedback
  onPress={this._onPressButton}>
  <View style={styles.button}>
    <Text style={styles.buttonText}>TouchableWithoutFeedback</Text>
  </View>
</TouchableWithoutFeedback>
```

```
<TouchableHighlight onPress={this._onPressButton} onLongPress={this._onLongPressButton} underlayColor="white">
  <View style={styles.button}>
    <Text style={styles.buttonText}>Touchable with Long Press</Text>
  </View>
</TouchableHighlight>
```

```
</View>
);
}
}
```

```
const styles = StyleSheet.create({
  container: {
    paddingTop: 60,
    alignItems: 'center'
  },
  button: {
    marginBottom: 30,
    width: 260,
    alignItems: 'center',
    backgroundColor: '#2196F3'
  },
  buttonText: {
    padding: 20,
    color: 'white'
  }
});
```

// skip this line if using Create React Native App

```
AppRegistry.registerComponent('ProjetoInicial', () => Touchables);
```

# Scrolling

- **ScrollView** é um container genérico que pode agrupar múltiplos componentes e views. Seus itens não necessariamente precisam ser homogêneos e você pode fazer o scroll tanto verticalmente quanto horizontalmente.

```

import React, { Component } from 'react';
import { AppRegistry, ScrollView, Image, Text } from 'react-native';

export default class IScrolledDownAndWhatHappenedNextShockedMe extends Component {
  render() {
    return (
      <ScrollView>
        <Text style={{fontSize:96}}>Scroll me plz</Text>
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Text style={{fontSize:96}}>If you like</Text>
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Text style={{fontSize:96}}>Scrolling down</Text>
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Text style={{fontSize:96}}>What's the best</Text>
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
        <Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
      </ScrollView>
    );
  }
}

```



# Scrolling (cont)

```
<Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
<Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
<Text style={{fontSize:96}}>Framework around?</Text>
<Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
<Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
<Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
<Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
<Image source={{uri: "https://facebook.github.io/react-native/img/favicon.png", width: 64, height: 64}} />
<Text style={{fontSize:80}}>React Native</Text>
</ScrollView>
);
}
}

// skip these lines if using Create React Native App
AppRegistry.registerComponent(
  'ProjetoInicial',
  () => IScrolledDownAndWhatHappenedNextShockedMe);
```

# List Views

- Para mostrar conjuntos de dados, React Native oferece os componentes:
  - **FlatList**: mostra uma lista com scrolling de dados semelhantes, porém mutáveis. Renderiza apenas os elementos que estão aparecendo na tela, e não todos de uma vez.
  - **SectionList**: renderiza conjuntos de dados separados em seções lógicas, possivelmente com cabeçalhos em cada seção.

# FlatList

- Formada pelas props elementares:
  - **data**: o dado propriamente dito. Uma lista de objetos json. Não obrigatoriamente esses objetos precisam ter a propriedade “key.” No entanto, caso não tenha, você deve prover o props `keyExtractor`.
    - `data = { <lista de objetos json> }`
  - **renderItem**: renderiza cada objeto json, de acordo com as propriedades escolhidas pelo programador.
    - `renderItem = { ({item}) => <Código JSX> {item.propriedade} </Código JSX>}`
  - **keyExtractor**: caso você não tenha a propriedade `key` nos objetos json de `data`, você deve implementar essa propriedade.
    - `keyExtractor = {(item: object, index: number) => string;}`
    - Ex.: `keyExtractor = {(item, index) => item + index}`

# FlatList

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

export default class FlatListBasics extends Component {
  render() {
    return (
      <View style={styles.container}>
        <FlatList
          data={[
            {key: 'Devin'},
            {key: 'Jackson'},
            {key: 'James'},
            {key: 'Joel'},
            {key: 'John'},
            {key: 'Jillian'},
            {key: 'Jimmy'},
            {key: 'Julie'},
          ]}
          renderItem={({item}) => <Text style={styles.item}>{item.key}</Text>}
        />
      </View>
    );
  }
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: 22
  },
  item: {
    padding: 10,
    fontSize: 18,
    height: 44,
  },
});
```

```
// skip this line if using Create React Native App
AppRegistry.registerComponent('ProjetoInicial', () =>
  FlatListBasics);
```

# SectionList

- Formada por quatro props elementares:
  - **sections**: o dado a ser renderizado. O tipo é um array de Section (objeto com uma propriedade data do tipo array).
  - **renderItem**: renderizador padrão para todo item em cada seção. Seu tipo é uma função de renderização que retorna JSX (elemento React).
  - **renderSectionHeader**: renderizado no topo de cada seção.
  - **keyExtractor**: usado para extrair uma chave única de um dado item em um índice especificado.

```

import React, { Component } from 'react';
import { AppRegistry, SectionList, StyleSheet, Text, View } from 'react-native';

export default class SectionListBasics extends Component {
  render() {
    return (
      <View style={styles.container}>
        <SectionList
          sections={[
            {title: 'D', data: ['Devin']},
            {title: 'J', data: ['Jackson', 'James', 'Jillian', 'Jimmy', 'Joel', 'John', 'Julie']},
          ]}
          renderItem={
            ({item}) => <Text style={styles.item}>
              {item}
            </Text>
          }
          renderSectionHeader={
            ({section}) => <Text style={styles.sectionHeader}>
              {section.title}
            </Text>
          }
          keyExtractor={
            (item, index) => index+item
          }
        />
      </View>
    );
  }
}

```

# SectionList

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: 22
  },
  sectionHeader: {
    paddingTop: 2,
    paddingLeft: 10,
    paddingRight: 10,
    paddingBottom: 2,
    fontSize: 14,
    fontWeight: 'bold',
    backgroundColor: 'rgba(247,247,247,1.0)',
  },
  item: {
    padding: 10,
    fontSize: 18,
    height: 44,
  },
})
```

```
// skip this line if using Create React Native App
AppRegistry.registerComponent('ProjetoInicial', () => SectionListBasics);
```

# Networking

- React Native provê a **Fetch API** para as suas necessidades de networking ([https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)).
- Fetch vai parecer familiar se você já tiver usado **XMLHttpRequest** ou outras APIs antes.



# Networking - Request

- Você pode simplesmente fazer um **fetch**, passando a URL:

```
fetch('https://mywebsite.com/mydata.json');
```

- Ou ainda, adicionar mais parâmetros, customizando a requisição HTTP:

```
fetch('https://mywebsite.com/endpoint/', {  
  method: 'POST',  
  headers: {  
    Accept: 'application/json',  
    'Content-Type': 'application/json',  
  },  
  body: JSON.stringify({  
    firstParam: 'yourValue',  
    secondParam: 'yourOtherValue',  
  }),  
});
```

# Networking - Response

- A operação de **fetch** é uma operação assíncrona, retornando assim uma **promisse**.

```
function getMoviesFromApiAsync() {  
  return fetch('https://facebook.github.io/react-native/movies.json')  
    .then((response) => response.json())  
    .then((responseJson) => {  
      return responseJson.movies;  
    })  
    .catch((error) => {  
      console.error(error);  
    });  
}
```

# Networking - Response

- Também é possível usar o **await/async**:

```
async function getMoviesFromApi() {  
  try {  
    let response = await fetch(  
      'https://facebook.github.io/react-native/movies.json',  
    );  
    let responseJson = await response.json();  
    return responseJson.movies;  
  } catch (error) {  
    console.error(error);  
  }  
}
```

# Networking – Exemplo (template)

```
import React, { Component } from 'react';
import { AppRegistry, FlatList, ActivityIndicator, Text, View } from 'react-native';

export default class FetchExample extends Component {

  constructor(props){
    super(props);
    this.state = { isLoading: true }
  }

  componentDidMount(){

  }

  render(){
    if(this.state.isLoading){
    }
  }
}

AppRegistry.registerComponent('ProjetoInicial', () => FetchExample);
```

**constructor:** inicializa o props e o state com a propriedade “isLoading”. Usaremos essa propriedade para indicar que os dados ainda estão sendo carregados do servidor.

**componentDidMount:** método do ciclo de vida do componente (ver aula01). É invocado imediatamente após um componente ser montado (inserido na árvore). Inicializações que exijam nós do DOM devem vir aqui. Se precisar carregar data de um endpoint remoto, este é um bom lugar para instanciar sua requisição. Este método é um bom lugar para colocar qualquer subscrição. Se fizer isto, não esqueça de desinscrever no componentWillUnmount().

**render:** finalmente, a renderização. No entanto, vamos antes testar se os dados estão sendo carregados. Caso afirmativo, iremos mostrar um feedback pro usuário, caso contrário, mostraremos os dados em uma Flatlist.

# Networking - Exemplo

```
componentDidMount(){  
  return fetch('https://facebook.github.io/react-native/movies.json')  
    .then((response) => response.json())  
    .then((responseJson) => {  
  
      this.setState({  
        isLoading: false,  
        dataSource: responseJson.movies,  
      });  
  
    })  
    .catch((error) => {  
      console.error(error);  
    });  
}
```

Requisita informação da URL. Retorna uma Promise<Responde>.

Transforma a response em um json. Retorna uma Promise<Any>, com o objeto json dentro.

Se “inscreve” na Promise e lê o json dela, chamando-o de **respondeJson** (podia ser qualquer nome).

Muda o estado via **setState**. **isLoading** passa a ser falso (não está mais carregando) e atualiza a propriedade **dataSource** com o objeto “movies”, que é um lista (vetor) de filmes.

A chamada o setState dentro do componentDidMount irá disparar o método render novamente.

# Networking - Exemplo

```
render(){
```

```
  if(this.state.isLoading){
```

```
    return(
```

```
      <View style={{flex: 1, padding: 20}}>
```

```
        <ActivityIndicator/>
```

```
      </View>
```

```
    )
```

```
  }
```

```
  return(
```

```
    <View style={{flex: 1, paddingTop: 20}}>
```

```
      <FlatList
```

```
        data={this.state.dataSource}
```

```
        renderItem={({item}) => <Text>{item.title}, {item.releaseYear}</Text>}
```

```
        keyExtractor={({id}, index) => id+index}
```

```
      />
```

```
    </View>
```

```
  );
```

```
}
```

Caso **isLoading** ainda esteja True, mostra o componente `<ActivityIndicator/>`.

Caso **isLoading** esteja False, cria uma FlatList, iniciando os dados com `this.state.dataSource`, variável inicializada pela Promise do método anterior.

O **render** vai ser chamado duas vezes: a primeira vez vai ser quando o componente for carregado pela primeira vez. Como `isLoading` ainda estará true, cairá no primeiro if. A segunda vez será quando o `setState` for chamado pelo método **componentDidMount**. O `isLoading` passará a ser false e o render vai cair no segundo if.