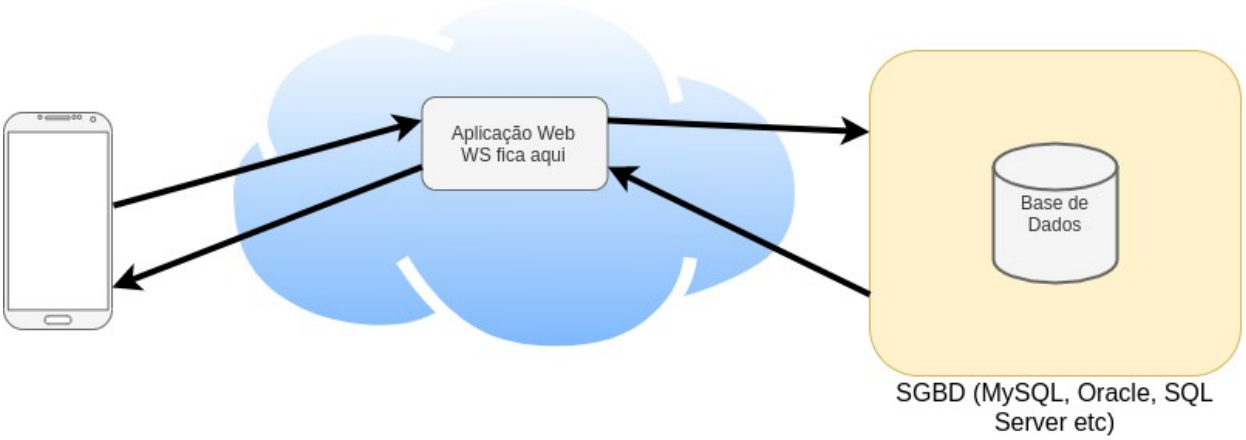
Professor Rodrigo Bossini
https://sites.google.com/site/professorrodrigobossini/

Introdução a React & React Native
Consumo de Web Services

1 Introdução

Em geral, aplicações para dispositivos móveis armazenam seus dados em servidores remotos. O acesso a bases remotas nunca é feito diretamente: a aplicação consome um Web Service (que nada mais é do que um serviço disponibilizado por uma aplicação web) que é responsável por acessar a base de interesse e entregar as informações solicitadas. A Figura 1.1 mostra como essa interação se dá tipicamente.

Figura 1.1



Neste material iremos desenvolver uma aplicação cuja tela principal é exibida na Figura 1.2. Ela consome um Web Service que disponibiliza a previsão do tempo. Em particular, o serviço que utilizaremos disponibiliza previsões em intervalos de três horas para os próximos cinco dias.

Figura 1.2


Sao Paulo
OK
18:00:00 - céu pouco nublado
Min: 18.02° Max: 19.16° Hum: 76%
21:00:00 - nuvens dispersas
Min: 17.43° Max: 17.91° Hum: 72%
00:00:00 - nuvens quebradas
Min: 17.85° Max: 17.99° Hum: 65%
03:00:00 - nuvens quebradas
Min: 16.71° Max: 16.73° Hum: 69%
06:00:00 - nuvens dispersas
Min: 16.16° Max: 16.16° Hum: 71%
09:00:00 - nuvens dispersas
Min: 19.35° Max: 19.35° Hum: 59%
12:00:00 - céu limpo
Min: 24.87° Max: 24.87° Hum: 39%
15:00:00 - céu limpo

2 Desenvolvimento

2.1 (Criando um novo projeto) Crie um novo projeto com o comando a seguir

expo init nome_do_seu_projeto

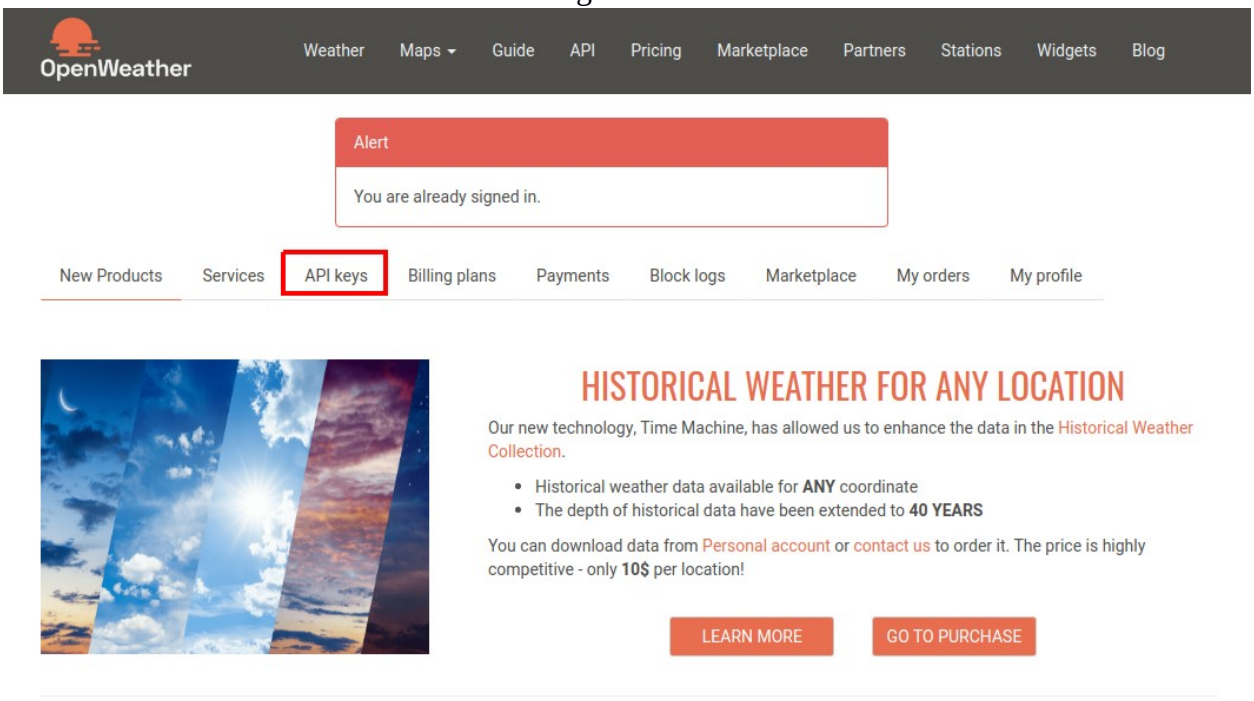
Escolha o template **blank (minimal app as clean as empty canvas)**.

2.2 (Criando uma conta no serviço de previsões) Enquanto o projeto está sendo criado, visite o Link 2.2.1 e crie uma conta para você no serviço de previsões do tempo.

Link 2.2.1
https://openweathermap.org/

Uma vez que tenha criado a conta, faça login e visite a página exibida pela Figura 2.2.1. Note que há uma opção chamada API keys. Clique nela.

Figura 2.2.1



Na tela a seguir, você deverá ver a sua chave. Copie-a e deixe guardada no seu clipboard para o próximo passo.

2.3 (O endpoint e a chave) No arquivo **App.js**, vamos criar duas contantes para armazenar o endereço do web service e a chave. Veja a Listagem 2.3.1.

Listagem 2.3.1

```
const endPoint = "https://api.openweathermap.org/data/2.5/forecast?
lang=pt&units=metric&q=";
const apiKey = //sua chave aqui
```

2.4 (A expressão JSX da tela principal) Note que a tela principal é um tanto simples. Ela Possui um campo para entrada de dados textual e, lado a lado com ele, um botão. Ou seja, precisamos de uma View para abrigá-los cujo flexDirection seja row. A seguir, temos uma FlatList que será alimentada em breve. A Listagem 2.4.1 mostra a expressão JSX e a Listagem 2.4.2 mostra a definição dos objetos de estilo.

Listagem 2.4.1

```
return (
<View style={styles.container}>
<View style={styles.entrada}>
<TextInput
style={styles.nomeCidade}
placeholder="Digite o nome de uma cidade"
/>
<Button
title="Ok"
/>
</View>
<FlatList
/>
</View>
);
}
```

Listagem 2.4.2

```
const styles = StyleSheet.create({
container: {
padding: 40,
flexDirection: 'column',
flex: 1,
backgroundColor: '#fff'
},
nomeCidade: {
padding: 10,
borderBottomColor: '#BB96F3',
borderBottomWidth: 2,
textAlign: 'left',
flexGrow: 0.9
},
entrada: {
flexDirection: 'row',
justifyContent: 'space-between',
marginBottom: 8
}
});
```

2.5 (Capturando a cidade digitada) Quando o usuário digitar algo, iremos capturar esse valor para que ele possa ser usado na consulta ao Web Service. Por isso, o componente terá uma variável em seu estado para armazenar a cidade digitada. Além disso, teremos uma função que será chamada sempre que o usuário digitar algo. Veja a Listagem 2.5.1.

Listagem 2.5.1

```
const [cidade, setCidade] = useState('');
const capturarCidade = (cidade) => {
setCidade(cidade);
}
<TextInput
style={styles.nomeCidade}
placeholder="Digite o nome de uma cidade"
value={cidade}
onChangeText={capturarCidade}
/>
```

2.6 (Estado para a lista de previsões) Uma vez feita a consulta ao Web Service, a aplicação receberá o resultado e o armazenará em uma lista, que também faz parte do componente principal. A FlatList utilizará essa lista para exibir seus itens. A princípio, iremos exibir somente sua representação textual. Veja a Listagem 2.6.1.

Listagem 2.6.1

```
const [previsoes, setPrevisoes] = useState([]);
<FlatList
data={previsoes}
renderItem={
previsao => (
<Text>{JSON.stringify(previsao)}</Text>
)
}
/>
```

2.7 (Função de consumo do Web Service) Quando o botão for tocado, a aplicação deverá consumir o Web Service e alimentar a FlatList adequadamente. A função da Listagem 2.7.1 é responsável pelo consumo do Web Service. Ela faz o seguinte

- Limpa a lista de previsões, para que resultados prévios sejam apagados e fique claro para o usuário que uma nova consulta está sendo realizada.

- Constrói o endereço do Web Service completo (chamamos de **target**), incluindo a cidade digitada pelo usuário e a chave.

- Consome o Webservice usando a função **fetch**.

- A primeira chamada à função **then** converte o objeto recebido para sua representação em JSON.

- A segunda chamada à função **then** configura o estado da aplicação, tomando o cuidado de extrair somente o vetor JSON associado à chave **list** do resultado recebido do Web Service e faz o teclado desaparecer.

Lembre-se que a função precisa ser vinculada ao botão. Veja a Listagem 2.7.1.

Listagem 2.7.1

```
const obtemPrevisoes = () => {
setPrevisoes([]);
const target = endPoint + cidade + "&appid=" + apiKey;
fetch(target)
.then((dados) => dados.json())
.then((dados) => {
setPrevisoes(dados["list"])
Keyboard.dismiss()
});
}

<Button
title="Ok"
onPress={obtemPrevisoes}
/>
```

Faça um **teste** neste momento. A aplicação deveria exibir uma lista de objetos JSON na tela, completamente sem formatação ainda, evidentemente.

2.8 (Novo componente para a exibição de previsões) Agora vamos criar um novo componente cuja finalidade será exibir as previsões. Crie uma nova pasta na raiz do projeto chamada **components**. Dentro dela, crie um arquivo chamado **PrevisaoItem.js**. Sua definição inicial é dada na Listagem 2.8.1.

Listagem 2.8.1

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



const PrevisaoItem = (props) => {
return (
<View />
);
}
const estilos = StyleSheet.create({});
export default PrevisaoItem;
```

2.9 (Usando cartões) Faremos uso de cartões para exibir cada previsão. O componente cartão é exibido na Listagem 2.9.1.

Listagem 2.9.1

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

const Cartao = (props) => {
return (
<View style={{ ...estilos.cartao, ...props.estilos }}>
{props.children}
</View>
);
};

const estilos = StyleSheet.create({
cartao: {
alignItems: 'center',
shadowColor: 'black',
shadowOffset: {
width: 0,
height: 2
},
shadowRadius: 6,
shadowOpacity: 0.32,
backgroundColor: 'white',
elevation: 4,
padding: 12,
borderRadius: 12
}
});


export default Cartao;
```

2.10 (A expressão JSX do componente PrevisaoItem) Cada previsão será exibida como um cartão. Como mostra a Figura 1.2, cada previsão uma figura que fica do lado de um componente visual com duas linhas. Na primeira linha, exibimos o horário da previsão e um texto descritivo. Na segunda linha, exibimos a temperatura mínima, a temperatura máxima e a humidade relativa do ar, todos lado a lado. Começamos adicionando um cartão à raiz do componente, como na Listagem 2.10.1.

Listagem 2.10.1

```
return (
<Cartao estilos={estilos.cartao}>

</Cartao>
);


cartao: {
marginBottom: 5
},
```

- A seguir, definimos uma View cuja finalidade será colocar a imagem e os demais componentes lado a lado. Veja a Listagem 2.10.2.

Listagem 2.10.2

```
return (
<Cartao estilos={estilos.cartao}>
<View style={estilos.tela}>
</View>

</Cartao>
tela: {
flexDirection: 'row',
},
```

- Para exibir uma imagem, faremos uso do componente **Image**, próprio do React Native. Ele se encarrega de fazer o download da imagem, dada a sua URL. Quando fazemos a primeira requisição ao Web Service, ele nos entregar um objeto JSON que inclui, entre todas as outras coisas, o nome de uma figura que pode ser baixada. A seguir, podemos fazer o download dela a partir de um outro endPoint, também definido na documentação oficial. Faça o teste acessando o Link 2.10.1.

Link 2.10.1
https://openweathermap.org/img/wn/01d.png

A definição do componente Image é dada na Listagem 2.10.3. Note que o link da figura deve ser especificado na propriedade **source** (e não **src**). Em breve ajustaremos esse detalhe. Além disso, precisamos especificar as medidas da figura.

Listagem 2.10.3

```
<Cartao estilos={estilos.cartao}>
<View style={estilos.tela}>
<Image
style={estilos.imagem}
source={{ uri: "" }}
/>
</View>

</Cartao>


imagem: {
width: 50,
height: 50
},
```

- A seguir, especificamos uma View que terá como finalidade abrigar as duas linhas. Sua existência é importante pois, por padrão, ela irá colocar seus filhos na vertical, já que o seu valor de **flexDirection** padrão é column. A primeira linha exibe a data e a descrição enquanto a segunda linha exibe temperaturas mínima e máxima e a humidade relativa do ar. Veja a Listagem 2.10.4.

Listagem 2.10.4

```
return (
<Cartao estilos={estilos.cartao}>
<View style={estilos.tela}>
<Image
style={estilos.imagem}
source={{ uri: "" }}
/>
<View>
<View style={estilos.primeiraLinha}>
<Text >data e descrição ficarão aqui</Text>
</View>

<View style={estilos.segundaLinha}>
<Text style={estilos.valor}>Temperatura mínima aqui</Text>
<Text style={estilos.valor}>Temperatura máxima aqui</Text>
<Text style={estilos.valor}>Humidade aqui</Text>
</View>
</View>
</View>
</Cartao>
);
}
primeiraLinha: {
justifyContent: 'center',
flexDirection: 'row'
},
segundaLinha: {
flex: 1,
flexDirection: 'row',
justifyContent: 'center',
marginTop: 4,
borderTopWidth: 1,
borderTopColor: '#DDD'
},
valor: {
marginHorizontal: 2,
}
```

2.11 (Usando o componente PrevisaoItem em App.js) Voltando ao arquivo App.js, especificamos uma expressão JSX que utiliza o componente PrevisaoItem para a exibição feita pela FlatList. Veja a Listagem 2.11.1. Note que entregamos ao componente a previsão que ele deverá renderizar, via **props**.

Listagem 2.11.1

```
<FlatList
data={previsoes}
renderItem={
previsao => (
<PrevisaoItem previsao={previsao} />
)
}
/>
```

2.12 (Extraindo os dados de interesse em PrevisaoItem.js) De volta ao arquivo PrevisaoItem.js, podemos extrair os dados de interesse do objeto recebido. A estrutura do JSON com que estamos lidando é exibida na Listagem 2.12.1. Estamos interessados somente nos seguintes itens:

- **dt**: quantidade de segundos passados desde 01/01/1070 até a data em que a consulta foi feita
- **description**: descrição sucinta das condições climáticas.
- **temp_min**: Temperatura mínima
- **temp_max**: Temperatura máxima
- **humidity**: Humidade relativa do ar
- **icon**: Nome da figura a ser baixada que representa as condições previstas

Listagem 2.12.1

```
"list": [
  {
   "dt": 1578409200,
   "main": {
    "temp": 284.92,
    "feels_like": 281.38,
    "temp_min": 283.58,
    "temp_max": 284.92,
    "pressure": 1020,
    "sea_level": 1020,
    "grnd_level": 1016,
    "humidity": 90,
    "temp_kf": 1.34
   },
   "weather": [
    {
     "id": 804,
     "main": "Clouds",
     "description": "overcast clouds",
     "icon": "04d"
    }
   ],
   "clouds": {
    "all": 100
   },
   "wind": {
    "speed": 5.19,
    "deg": 211
   },
   "sys": {
    "pod": "d"
   },
   "dt_txt": "2020-01-07 15:00:00"
  },
```

A Listagem 2.12.2 mostra como o componente PrevisaoItem utiliza esses dados.

Listagem 2.12.2

```
return (
<Cartao estilos={estilos.cartao}>
<View style={estilos.tela}>
<Image
style={estilos.imagem}
source={{ uri: "https://openweathermap.org/img/wn/" +
props.previsao.item.weather[0].icon + ".png" }}
/>
<View>
<View style={estilos.primeiraLinha}>
<Text >{new Date(props.previsao.item.dt * 1000).toLocaleTimeString()} -
{props.previsao.item.weather[0].description}</Text>
</View>

<View style={estilos.segundaLinha}>
<Text style={estilos.valor}>Min: {props.previsao.item.main.temp_min + "\
u00B0"}</Text>
<Text style={estilos.valor}>Max: {props.previsao.item.main.temp_max +
"\u00B0"}</Text>
<Text style={estilos.valor}>Hum: {props.previsao.item.main.humidity}
%</Text>
</View>
</View>
</View>
</Cartao>
);
```

Referências

React – A JavaScript library for building user interfaces. 2020. Disponível em <https://reactjs.org/> Acesso em abril de 2020.

React Native · A framework for building native apps using React. 2020. Disponível em <https://facebook.github.io/react-native/>. Acesso em abril de 2020.

Current weather and forecast - OpenWeatherMap - 2020. Disponível em <https://openweathermap.org/>. Acesso em abril de 2020.