

Instituto Federal de Educação, Ciência e Tecnologia de São Paulo
Campus Cubatão

Discentes: EDGAR DOS SANTOS RAMOS – 1790323

BRAYAN FREIRE DOS SANTOS - 1691007

GABRIEL GERALDINO DE SOUZA – 1790242

IGOR DA COSTA B. LAURENTINO – 1790579

JOSUÉ FONSECA – 1790331

JÚLIA DOS SANTOS SIMÕES - 1590308

LUCAS MERCEDES PALOMANES - 1790358

MATHEUS MOURA SILVA - 169118X

MICHEL W. HAAGEN SANTOS – 1690566

SAMUEL TEIXEIRA LIMA – 1690922

VICTOR ZOGHAIB CONDE VENTURA – 1790455

YVIS FREIRE SILVA SANTOS - 1790099

REACT NATIVE

Curso: Técnico de Informática Integrado ao Ensino Médio – CTII 417

Docente: Mauricio Neves Asenjo

Disciplina: Projetos de Sistemas

Cubatão/SP
2020

História do React Native

Em 2012, Mark Zuckerberg, CEO do Facebook, empresa dona das maiores redes sociais do mundo, como Whatsapp, Instagram e o próprio Facebook, comentou, “O maior erro que cometemos como empresa foi apostar demais em HTML5 em oposição ao nativo”, prometendo que a companhia entregaria uma melhor experiência em dispositivos móveis em breve.

Cumprindo a promessa de Zuckerberg, em 2013, o Facebook lançou um projeto interno chamado React.js, uma espécie de Hackathon, competição em que são desenvolvidos softwares, que viria a ser responsável pela criação do framework React Native. Durante a React.js Con. em janeiro de 2015, ocorreu a primeira prévia pública do framework e em março de 2015, durante a conferência anual de desenvolvimento do Facebook, o F8, foi anunciado que o framework estava disponível de forma aberta no GitHub, plataforma de hospedagem de código-fonte.

Ainda durante o Facebook F8 de 2015, foi anunciado que duas gigantes da tecnologia, Microsoft e Samsung, iriam trazer o React Native para o Windows e Tizen, sistema operacional de Smart TVs. Com tamanho apoio, a busca pela plataforma ultrapassou, em setembro de 2016, a busca por desenvolvimento IOS e Android, de acordo com o Google Trends, mantendo sua posição até os dias atuais.



Atualmente, aplicativos extremamente utilizados como Facebook, Instagram, UberEATS, Airbnb e Nubank são desenvolvidos em React Native, isso ocorre pois a programação é facilitada devido a versatilidade da plataforma, que propicia a geração de versões nativas para IOS e Android com o mesmo código-fonte, o que poupa muitas horas e custos das desenvolvedoras. Além disso, o framework utiliza uma das linguagens de programação dinâmicas de alto nível mais populares da atualidade, o JavaScript, combinando assim os benefícios da linguagem com a biblioteca React.js.

As IDEs mais utilizadas no desenvolvimento em React Native

1. Atom

- **URL:** atom.io
- **Github:** atom
- **Documentação:**
 - Configuração;
 - Atom com React Native.
- **Plataformas:** Windows, Mac, Linux
- **Licença:** Open Source
- **Recursos:**
 - Edição multiplataforma;
 - Gerenciador de pacotes integrado;
 - Preenchimento automático inteligente;
 - Navegador do sistema de arquivos;
 - Vários painéis;
 - Encontre e substitua.
- **Pacotes:**
 - **atom-react-native-css:** É um pacote para estilizar componentes React-Native com suporte integrado para SASS / SCSS. React-native-css transforma CSS / SASS válidos no subconjunto de CSS do Facebook;
 - **react-native-snippets:** É um pacote para os snippets React Native para Atom e Nuclide;
 - **zenchat-snippets:** É uma coleção de snippets para react-native, redux e ES6;
 - **atom-xcode:** This package bridges the gap between Mac Xcode and atom. Once installed, the iOS simulator can be controlled from within the atom itself;
 - **language-babel:** This package includes Language grammar for all versions of JavaScript including ES2016 and ESNext, JSX syntax as used by Facebook React, Atom's etch and others.

2. Visual Studio Code

- **URL:** code.visualstudio.com
- **Github:** Microsoft / vscode
- **Licença:** Free-source Code
- **Documentação:**
 - Configuração;
 - Desenvolva aplicativos React Native no Visual Studio Code.
- **Plataformas:** Windows, Mac, Linux
- **Recursos:**
 - Comandos Git integrados;
 - Extensível e personalizável.
- **Extensões:**
 - **Ferramentas ReactNative:** Essa extensão fornece um ambiente de desenvolvimento para projetos React Native. Você pode depurar seu código, executar react-native comandos rapidamente a partir da paleta de comandos e usar o IntelliSense para navegar por objetos, funções e parâmetros para APIs do React Native.

3. Editor Vim

- **URL:** vim.org
- **Github:** [vim / vim](https://github.com/vim/vim)
- **Documentação:**
 - Vim Docs
 - Configure o Vim para React-JSON
- **Licença:** Open Source
- **Plataformas:** Mac, Linux
- **Recursos:**
 - Árvore de desfazer persistente e de vários níveis;
 - Sistema extenso de plugins;
 - Suporte para centenas de linguagens de programação e formatos de arquivo;
 - Pesquisa e substituição poderosa;
 - Integra-se com muitas ferramentas.
- **Plug-ins:**
 - **vim-jsx:** Syntax highlighting and indenting for JSX;
 - **vim-react-snippets:** A set of snippets for Vim to work with Facebook's React library;
 - **vim-babel:** A set of snippets for Vim to work with Facebook's React library.

4. Sublime Text

- **URL:** sublimetext.com
- **Github:** [SublimeText](https://github.com/SublimeText)
- **Documentação:**
 - Configuração
- **Plataformas:** Windows, Mac, Linux
- **Licença:** Sublime Text pode ser baixado e avaliado gratuitamente; no entanto, uma licença deve ser adquirida para uso por um longo período.
- **Recursos:**
 - Vá para qualquer coisa;
 - Múltiplas seleções;
 - Paleta de Comandos;
 - Modo livre de distração;
 - Edição Dividida;
 - Mudança de projeto instantânea;
 - API Plugin;
 - Personalize qualquer coisa;
 - Plataforma Cruzada.
- **Pacotes:**
 - **react-native-snippets:** É uma coleção de Sublime Text Snippets para react-native;
 - **babel-sublime:** Definições de sintaxe para ES6 JavaScript com extensões React JSX.

4. Editor GNU Emacs

- **URL:** gnu.org/software/emacs/

- **Documentação:**

- Documentação oficial;
- Configuração inicial para React Native.

- **Licença:** Gratuito sob licença GPL

- **Plataformas:** Windows, Mac, Linux

- **Recursos:**

- Modos de edição com reconhecimento de conteúdo, incluindo coloração de sintaxe, para muitos tipos de arquivo;

- Documentação integrada completa, incluindo um tutorial para novos usuários;
- Suporte total ao Unicode para quase todos os scripts humanos.
- Altamente personalizável, usando código Emacs Lisp ou uma interface gráfica;
- Um sistema de empacotamento para baixar e instalar extensões.

- **Extensões:**

- **web-mode.el:** É um modo principal autônomo do emacs para edição de modelos da web. É compatível com várias linguagens, incluindo JSX (React).

6. Editor Spacemacs

- **URL:** spacemacs.org
- **Github:** syl20bnr / spacemacs
- **Documentação:**
 - Documentação oficial
- **Licença:** Open Source
- **Plataformas:** Windows, Mac, Linux
- **Recursos:**
 - As combinações de teclas são organizadas usando prefixos mnemônicos;
 - Detectável - exibição inovadora em tempo real das combinações de teclas disponíveis.;
 - Funcionalidades semelhantes têm a mesma ligação de chave em todos os lugares;
 - Sistema de consulta simples para encontrar rapidamente as camadas disponíveis, pacotes e muito mais;
 - A configuração voltada para a comunidade fornece pacotes com curadoria ajustados por usuários avançados e os bugs são corrigidos rapidamente.
- **Extensões:**
 - **Camada React:** camada de configuração pronta para ES6 e JSX para React. Ele reconhecerá automaticamente os arquivos .jsx e .react.js. Uma camada de pacote para integração do React.;

7. Webstorm

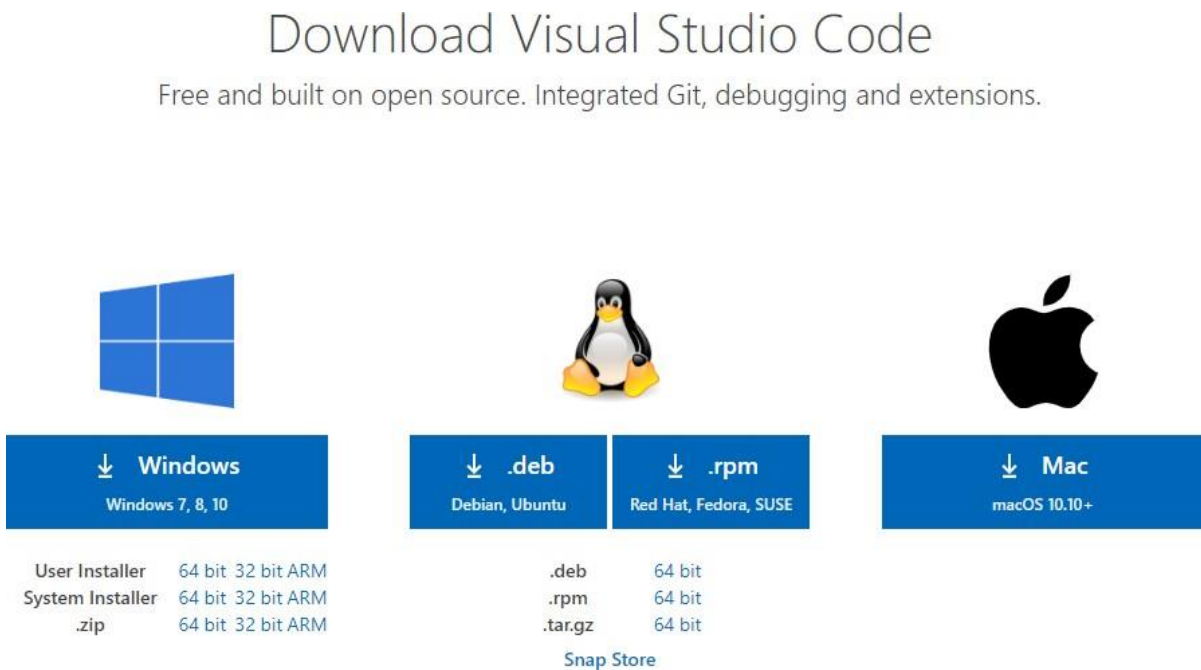
- **URL:** jetbrains.com/webstorm/
- **Documentação:**
 - Documentação oficial
 - Usando ferramentas externas
- **Licença:** Paga (US\$ 129,00 para usuário único / primeiro ano)
- **Plataformas:** Windows, Mac, Linux
- **Recursos:**
 - Assistência de codificação inteligente;
 - Suporte para as tecnologias mais recentes;
 - Sistema de controle de versão;
 - Integração Perfeita de Ferramentas;
 - Depuração, rastreamento e teste;
 - Terminal integrado.
- **Plug-ins:**
 - javascript-jsx.tmbundle:** Pacote Textmate para JSX (React). Atualmente suporta realce de sintaxe.

Passo a Passo da instalação da IDE

A IDE (Integrated Development Environment) escolhida para programação com o framework React Native foi o Visual Studio Code, sendo necessário também o download e instalação do software node.js para o funcionamento do React Native e do software Expo-CLI em um dispositivo móvel e no computador.


Processo de download e instalação do Visual Studio Code:

1. Entrar no [site](#) de download da IDE e escolha a opção relacionada a seu sistema operacional;




Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.

 **↓ Windows**
Windows 7, 8, 10


User Installer	64 bit	32 bit	ARM
System Installer	64 bit	32 bit	ARM
.zip	64 bit	32 bit	ARM

 **↓ .deb**
Debian, Ubuntu

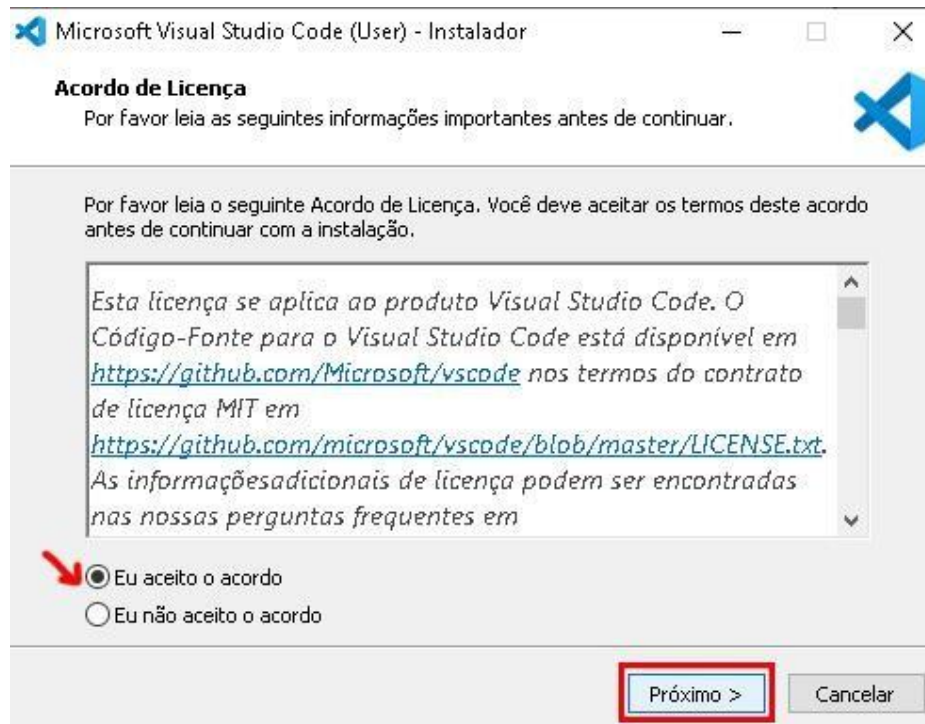
↓ .rpm
Red Hat, Fedora, SUSE

.deb	64 bit
.rpm	64 bit
.tar.gz	64 bit

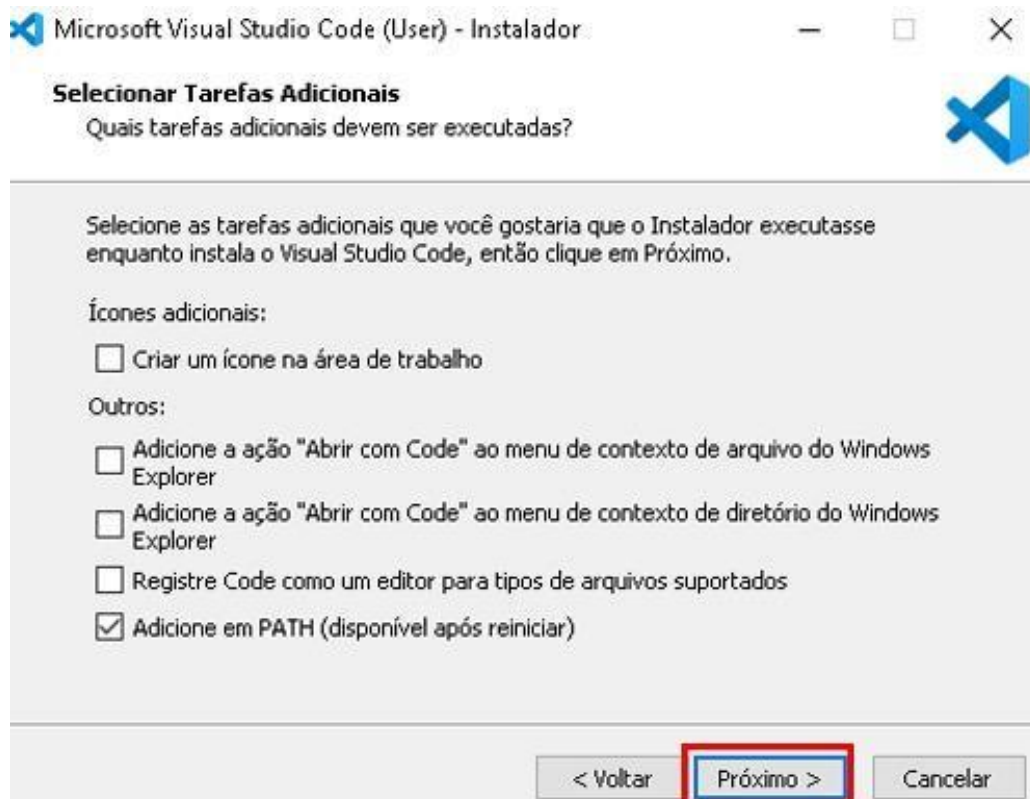
[Snap Store](#)

 **↓ Mac**
macOS 10.10+

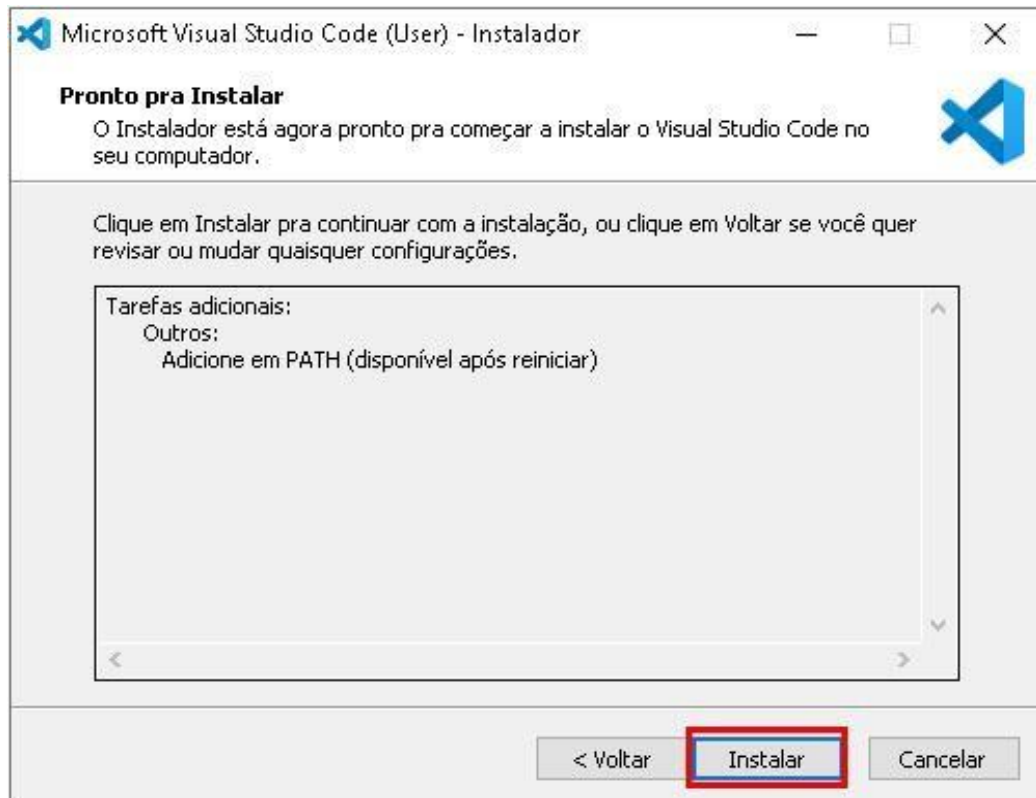
2. Abra o executável, aceite o Acordo de Licença e clique em **Próximo**;



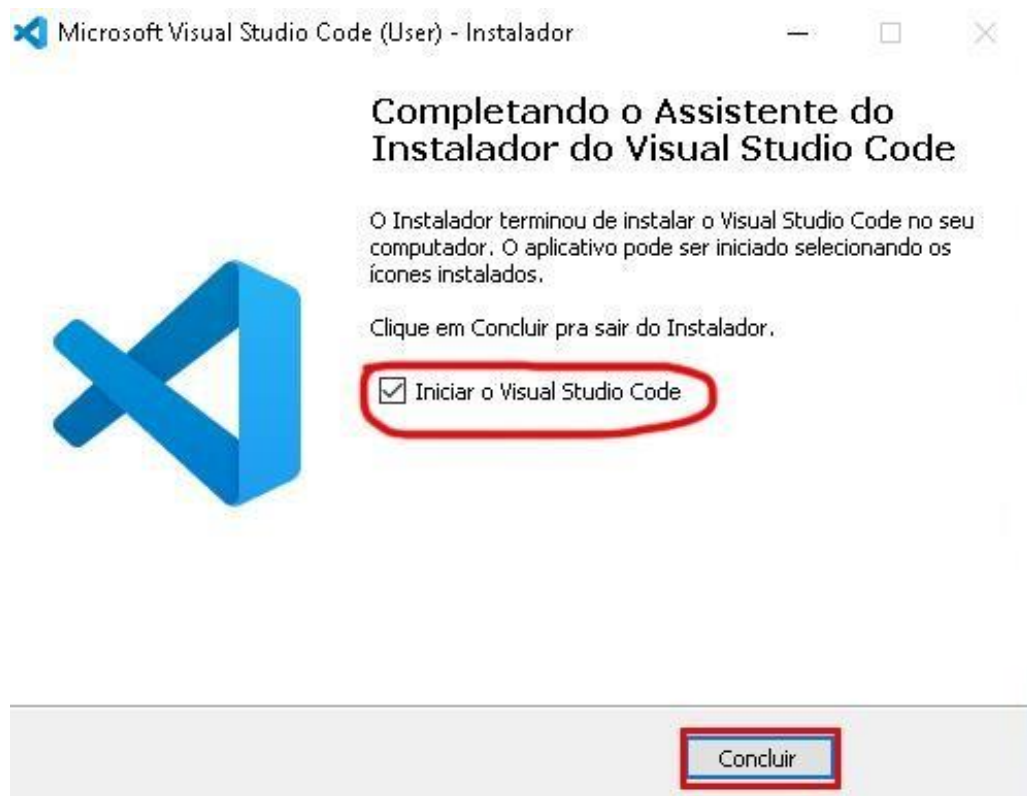
3. Clique em **Próximo**;



4. Clique em **Instalar**;



5. Marque a caixa para iniciar o Visual Studio Code e clique em **concluir**.






Processo de download e instalação do Node.js:

1. Entre no [site](#) de download do *software* e escolha a opção relacionada a seu sistema operacional.

Downloads

Versão LTS Mais Recente: 12.18.4 (includes npm 6.14.6)

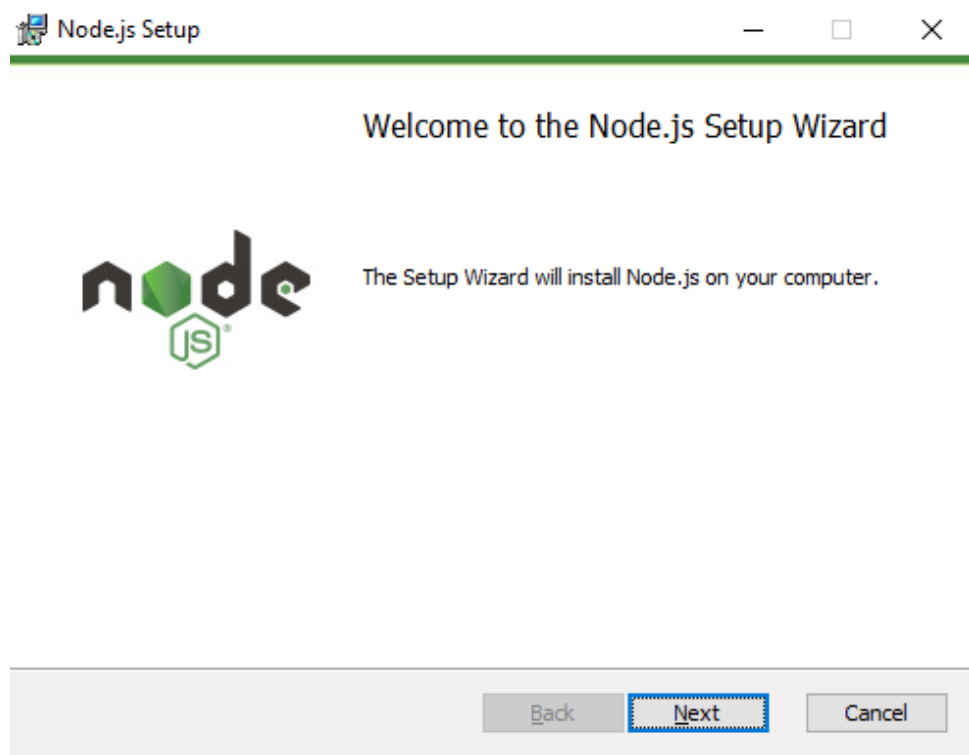
Baixe o código fonte do Node.js ou um instalador pré-compilado para o seu sistema, e comece a desenvolver hoje.

LTS Recomendado Para a Maioria dos Usuários	Atual Recursos Mais Recentes	
 Instalador Windows <small>node-v12.18.4-x04.msi</small>	 Instalador macOS <small>node-v12.18.4.pkg</small>	 Código fonte <small>node-v12.18.4.tar.gz</small>

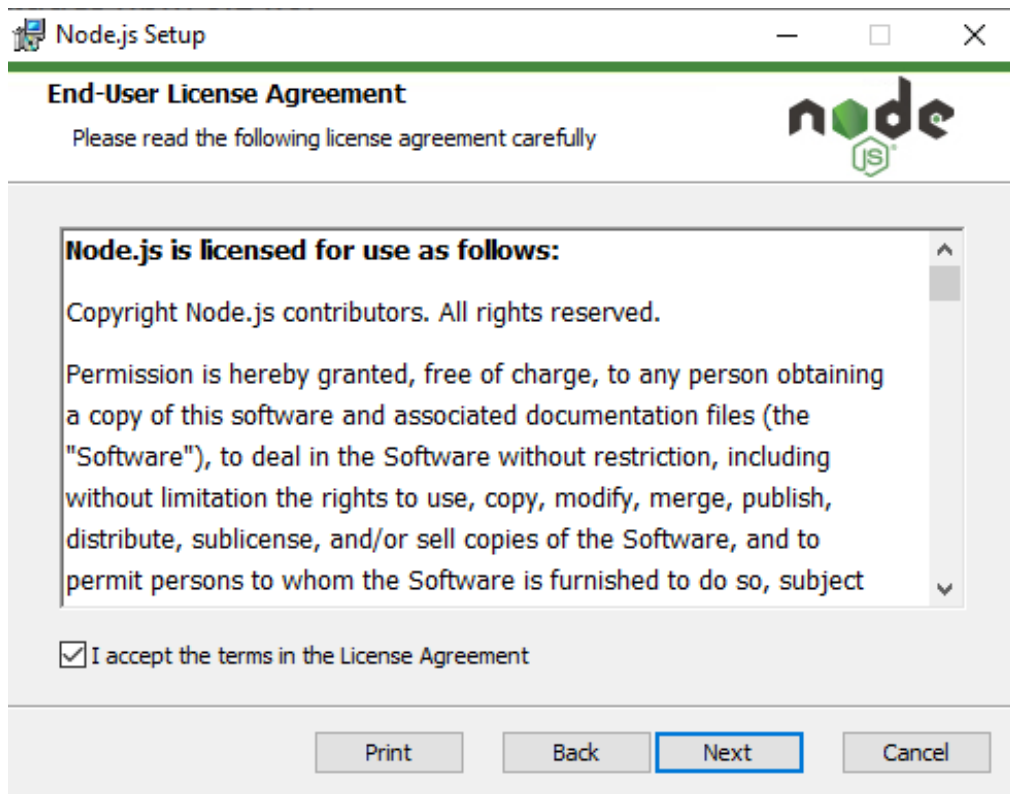
Instalador Windows (.msi)
Binário para Windows (.zip)
Instalador macOS (.pkg)
Binário para macOS (.tar.gz)
Binários para Linux (x64)
Binários para Linux (ARM)
Código fonte

32-bit	64-bit
32-bit	64-bit
64-bit	
64-bit	
64-bit	
ARMv7	ARMv8
node-v12.18.4.tar.gz	

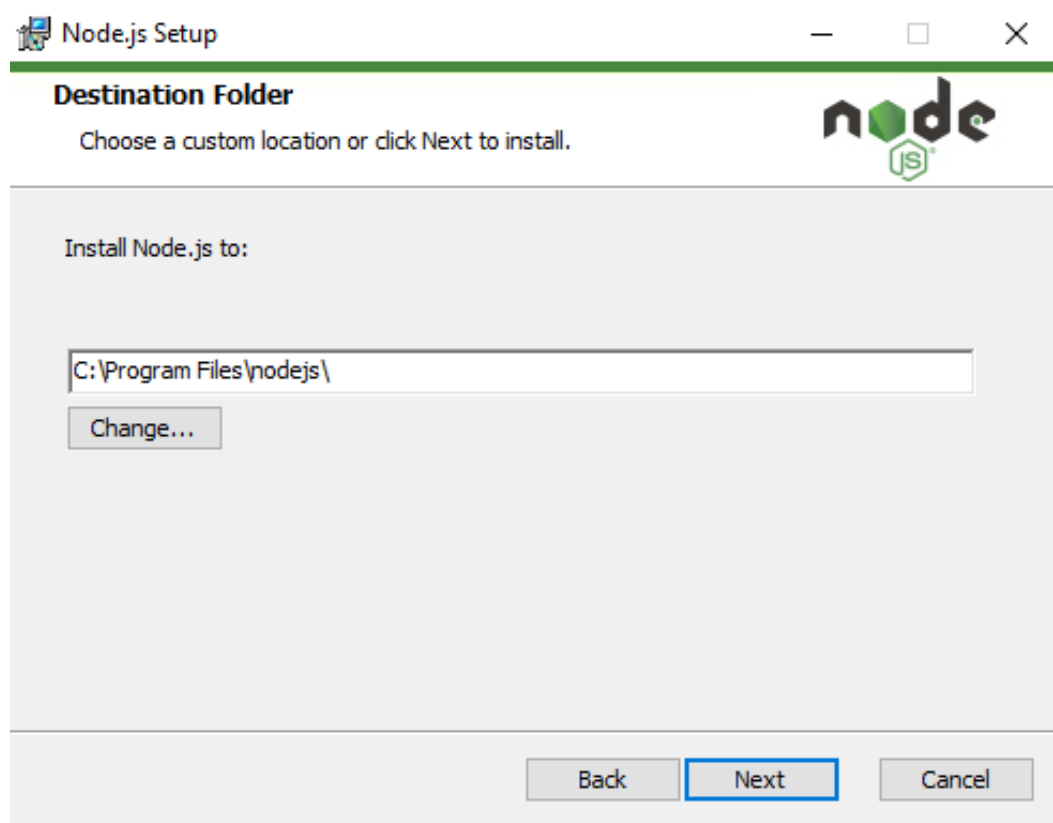
2. Após baixar o instalador, clique em **Next**;



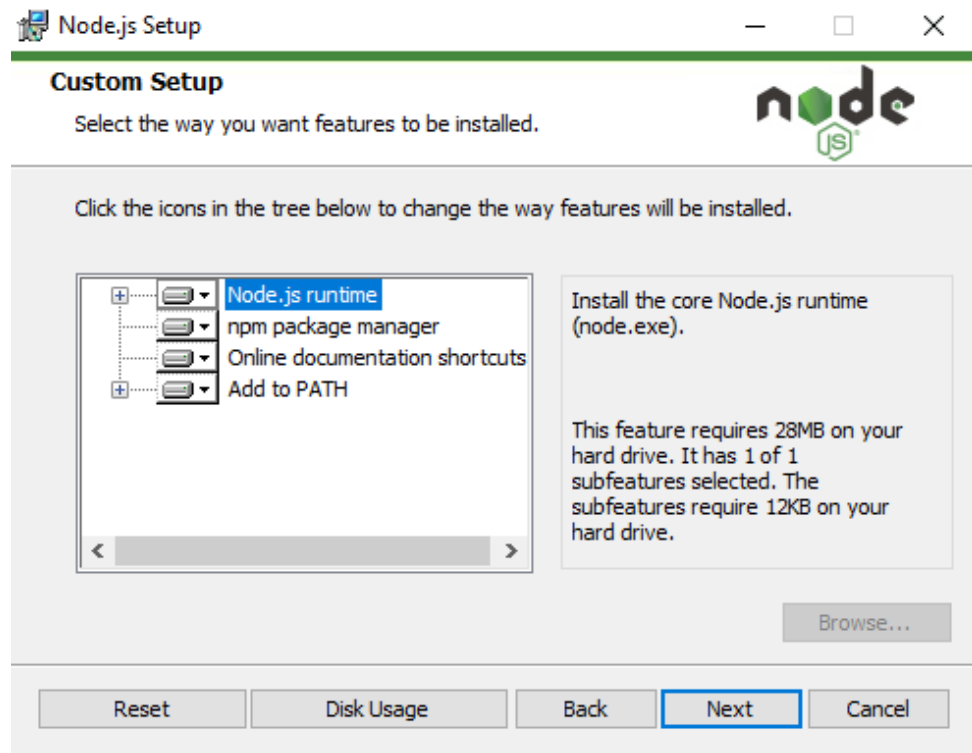
3. Leia, aceite os termos de uso e clique em **Next**;



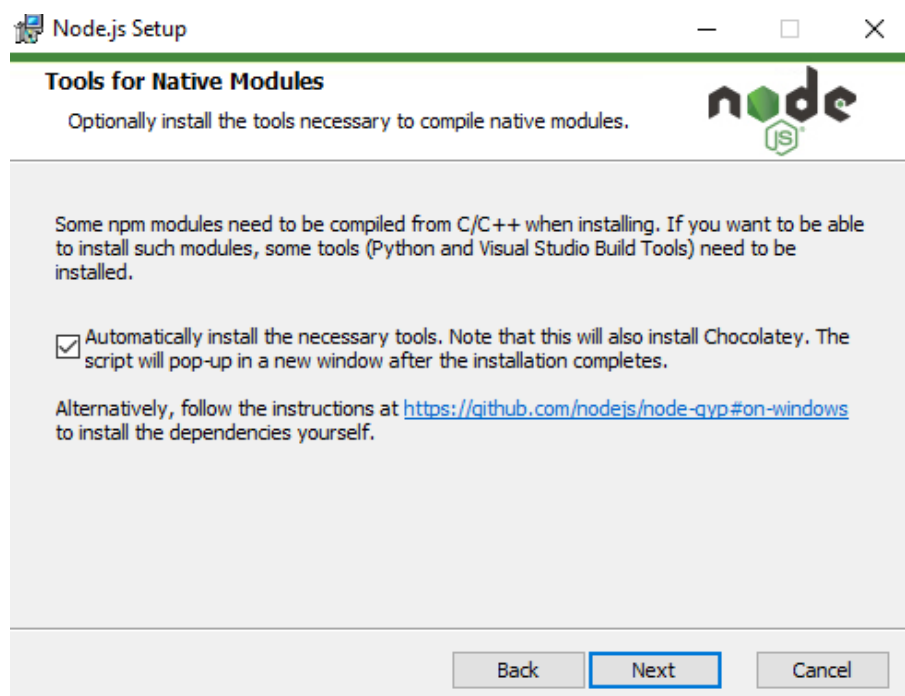
4. Escolha o diretório de instalação e clique em **Next**,



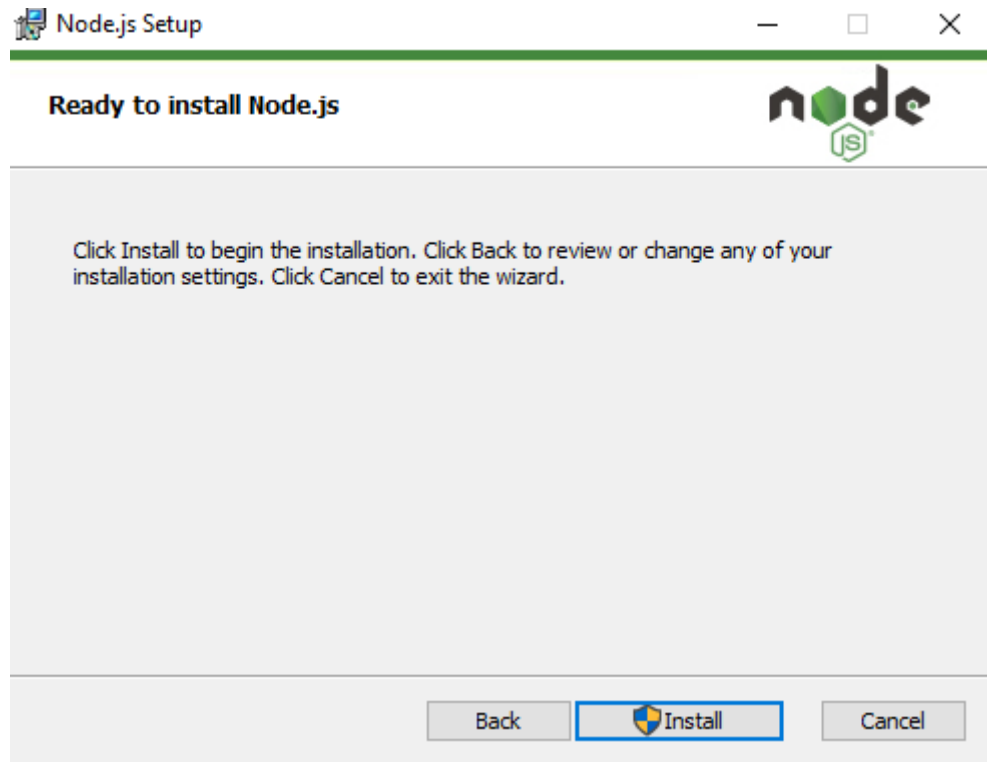
5. Clique em **Next**,



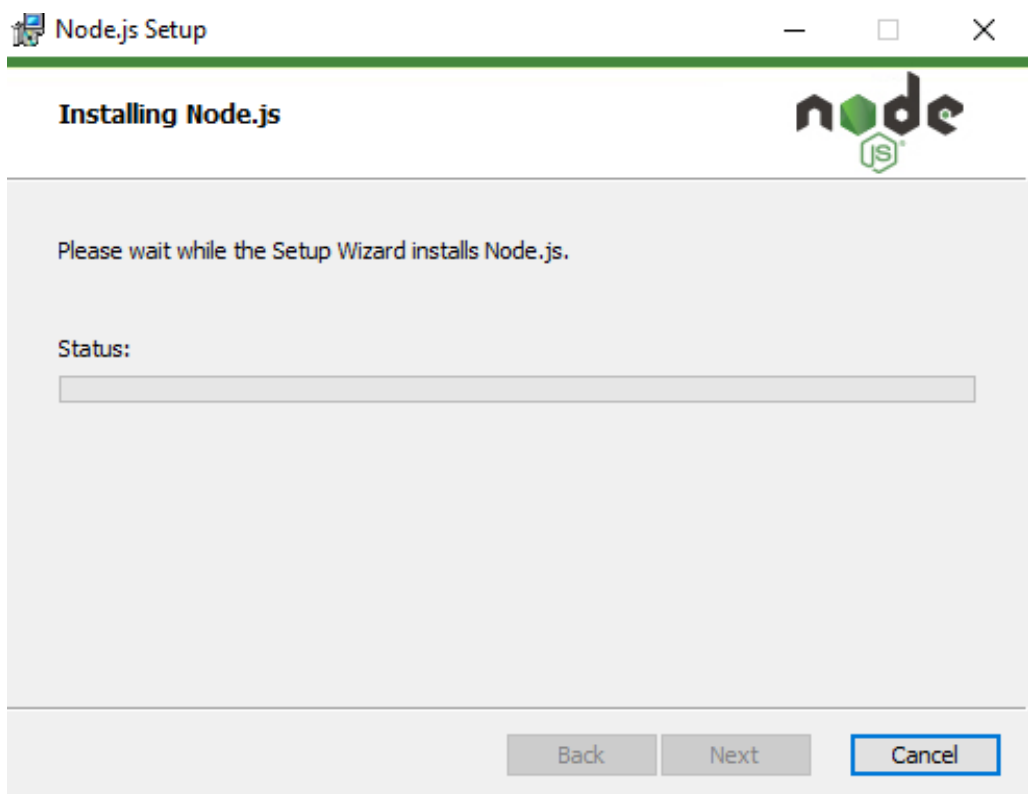
6. Marque a **caixa de seleção** para instalar as ferramentas necessárias para instalar o gerenciador de pacotes Chocolatey e clique em **Next**,



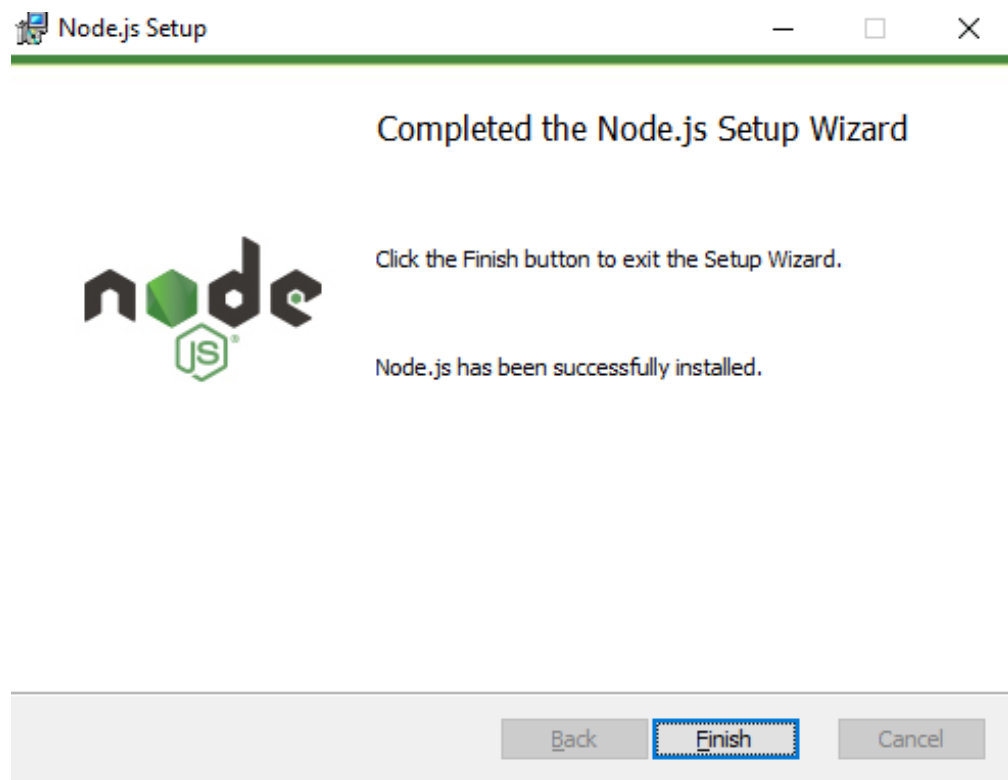
7. Clique em **Install** e clique em **Sim** quando o Windows solicitar permissão de administrador para prosseguir com a instalação;



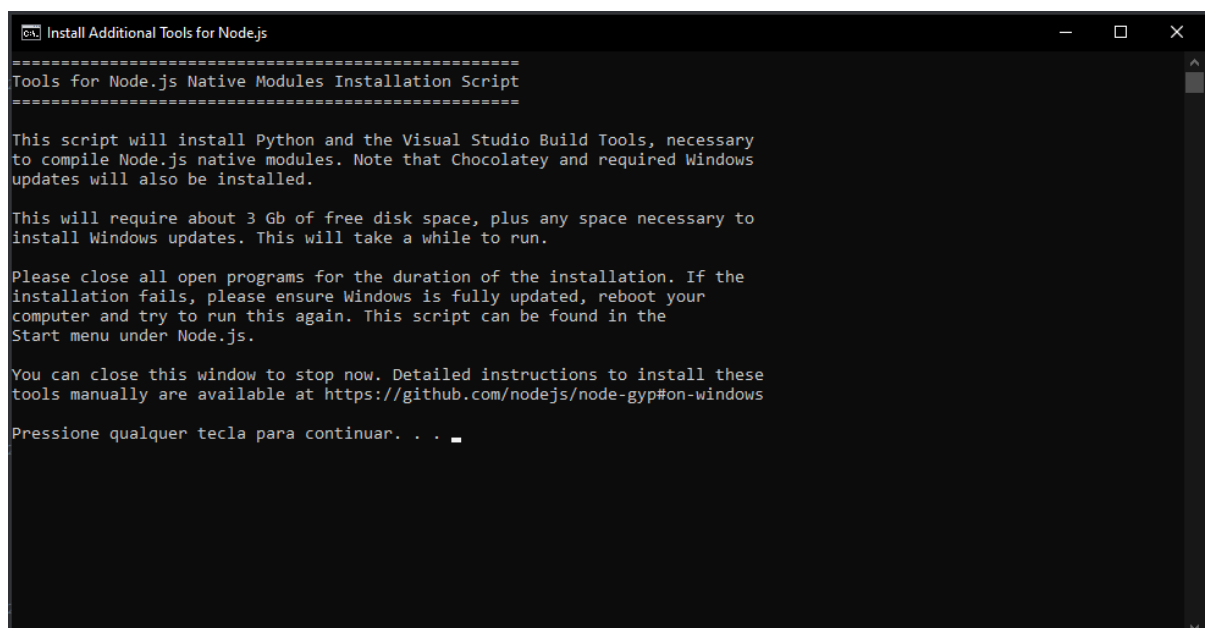
8. Aguarde a barra de progresso ficar completa.



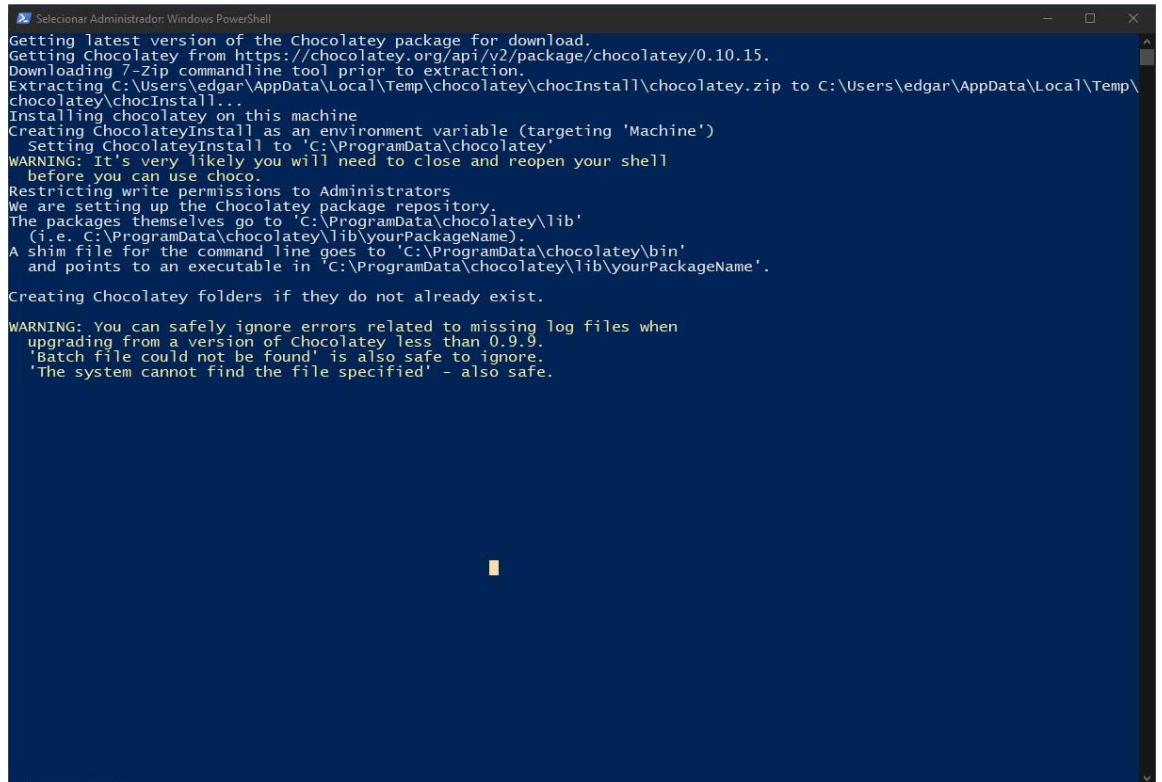
9. Clique em **Finish**;



10. No Prompt de Comando, pressione **qualquer tecla** para continuar;

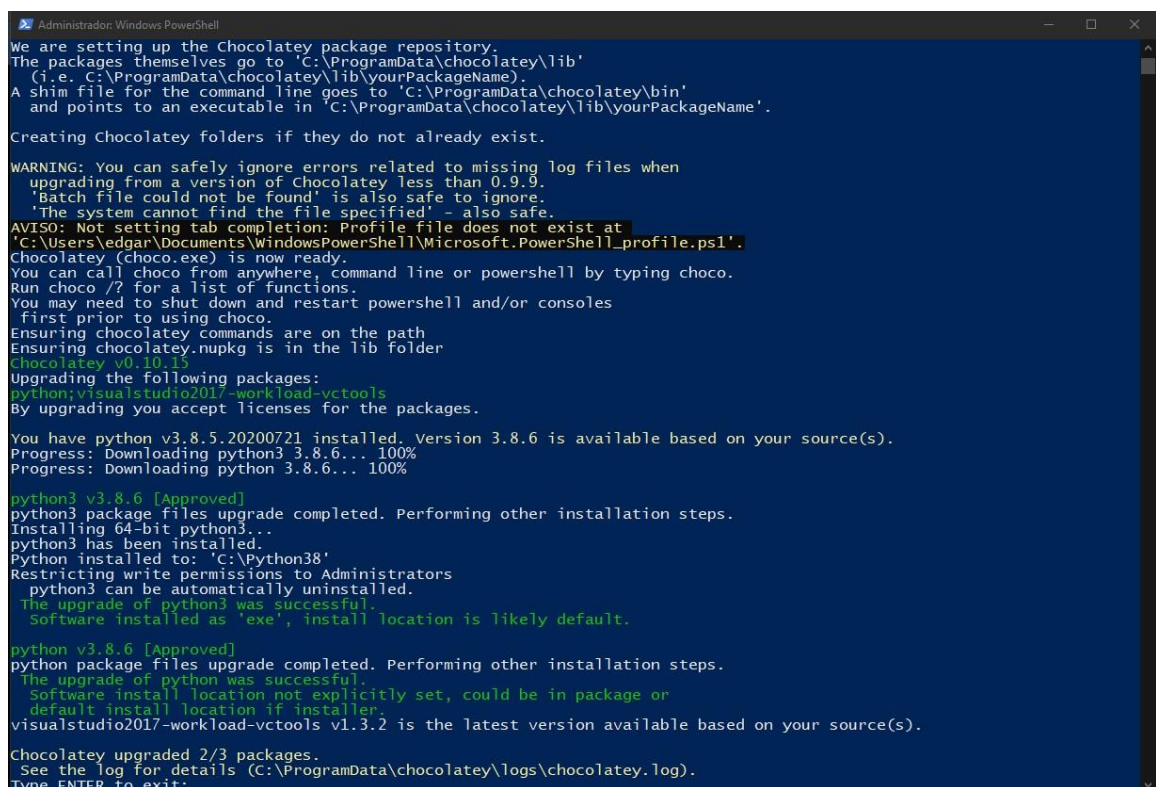


11. Quando o Windows Power Shell for aberto, aguarde a conclusão do progresso de instalação do Chocolatey;



```
Selecionar Administrador: Windows PowerShell
Getting latest version of the Chocolatey package for download.
Getting Chocolatey from https://chocolatey.org/api/v2/package/chocolatey/0.10.15.
Downloading 7-Zip commandline tool prior to extraction.
Extracting C:\Users\edgar\AppData\Local\Temp\chocolatey\chocInstall\chocolatey.zip to C:\Users\edgar\AppData\Local\Temp\chocolatey\chocInstall...
Installing chocolatey on this machine
Creating ChocolateyInstall as an environment variable (targeting 'Machine')
Setting ChocolateyInstall to 'C:\ProgramData\chocolatey'
WARNING: It's very likely you will need to close and reopen your shell
before you can use choco.
Restricting write permissions to Administrators
We are setting up the Chocolatey package repository.
The packages themselves go to 'C:\ProgramData\chocolatey\lib'
(i.e. C:\ProgramData\chocolatey\lib\yourPackageName).
A shim file for the command line goes to 'C:\ProgramData\chocolatey\bin'
and points to an executable in 'C:\ProgramData\chocolatey\lib\yourPackageName'.
Creating Chocolatey folders if they do not already exist.
WARNING: You can safely ignore errors related to missing log files when
upgrading from a version of Chocolatey less than 0.9.9.
'Batch file could not be found' is also safe to ignore.
'The system cannot find the file specified' - also safe.
```

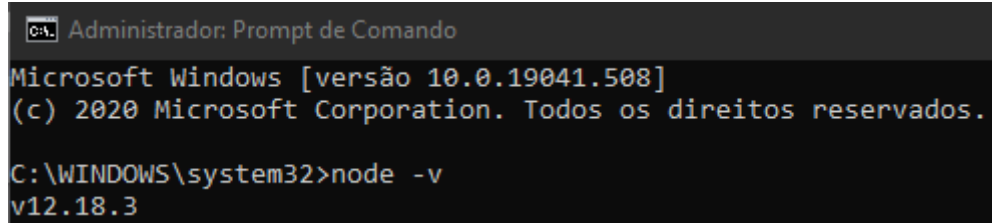
12. Ao fim do processo, aperte a tecla **ENTER**.



```
Administrador: Windows PowerShell
We are setting up the Chocolatey package repository.
The packages themselves go to 'C:\ProgramData\chocolatey\lib'
(i.e. C:\ProgramData\chocolatey\lib\yourPackageName).
A shim file for the command line goes to 'C:\ProgramData\chocolatey\bin'
and points to an executable in 'C:\ProgramData\chocolatey\lib\yourPackageName'.
Creating Chocolatey folders if they do not already exist.
WARNING: You can safely ignore errors related to missing log files when
upgrading from a version of Chocolatey less than 0.9.9.
'Batch file could not be found' is also safe to ignore.
'The system cannot find the file specified' - also safe.
AVISO: Not setting tab completion: Profile file does not exist at
'C:\Users\edgar\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1'.
Chocolatey (choco.exe) is now ready.
You can call choco from anywhere, command line or powershell by typing choco.
Run choco /? for a list of functions.
You may need to shut down and restart powershell and/or consoles
first prior to using choco.
Ensuring chocolatey commands are on the path
Ensuring chocolatey.nupkg is in the lib folder
chocolatey v0.10.15
Upgrading the following packages:
python;visualstudio2017-workload-vctools
By upgrading you accept licenses for the packages.
You have python v3.8.5.20200721 installed. Version 3.8.6 is available based on your source(s).
Progress: Downloading python3 3.8.6... 100%
Progress: Downloading python 3.8.6... 100%
python3 v3.8.6 [Approved]
python3 package files upgrade completed. Performing other installation steps.
Installing 64-bit python3...
python3 has been installed.
Python installed to: 'C:\Python38'
Restricting write permissions to Administrators
python3 can be automatically uninstalled.
The upgrade of python3 was successful.
Software installed as 'exe', install location is likely default.
python v3.8.6 [Approved]
python package files upgrade completed. Performing other installation steps.
The upgrade of python was successful.
Software install location not explicitly set, could be in package or
default install location if installer.
visualstudio2017-workload-vctools v1.3.2 is the latest version available based on your source(s).
Chocolatey upgraded 2/3 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
Type ENTER to exit:
```

Verificando se o node.js e o Chocolatey foram instalados corretamente:

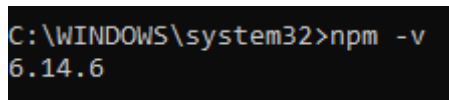
13. Execute o Prompt de Comando como administrador e digite o comando **node -v**;



```
Administrador: Prompt de Comando
Microsoft Windows [versão 10.0.19041.508]
(c) 2020 Microsoft Corporation. Todos os direitos reservados.

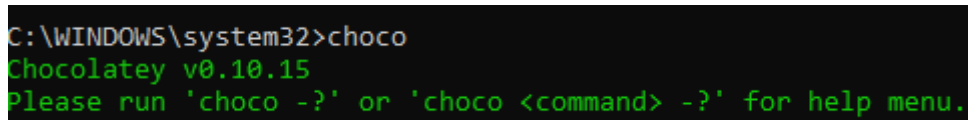
C:\WINDOWS\system32>node -v
v12.18.3
```

14. Digite o comando **npm -v**;



```
C:\WINDOWS\system32>npm -v
6.14.6
```

15. Digite o comando **choco**.



```
C:\WINDOWS\system32>choco
Chocolatey v0.10.15
Please run 'choco -?' or 'choco <command> -?' for help menu.
```

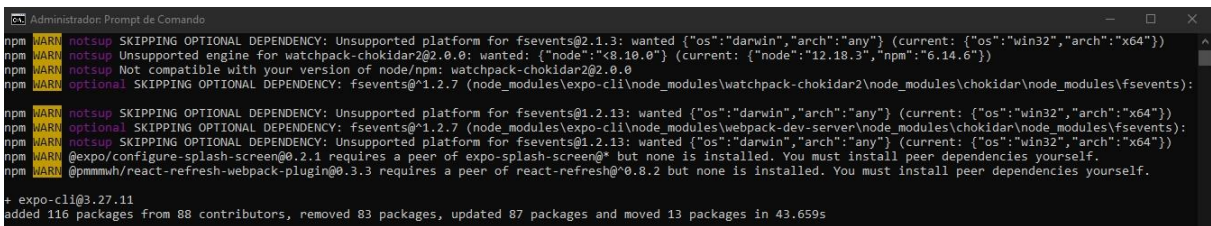
Caso as versões dos *softwares* forem apresentadas, como nas imagens acima, eles terão sido instalados corretamente.

Processo de instalação do Expo-CLI no computador:

1. Insira o comando `cd \` para ir para o diretório raiz e o comando `npm install -g expo-cli` para instalar o expo-cli no computador;

```
C:\WINDOWS\system32>cd\  
C:\>npm install -g expo-cli
```

2. A instalação estará concluída quando a versão do Expo-CLI for apresentada.



```
Administrador Prompt de Comando  
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})  
npm WARN notsup Unsupported engine for watchpack-chokidar2@2.0.0: wanted {"node":"<8.10.0"} (current: {"node":"12.18.3","npm":"6.14.6"})  
npm WARN notsup Not compatible with your version of node/npm: watchpack-chokidar2@2.0.0  
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\expo-cli\node_modules\watchpack-chokidar2\node_modules\chokidar\node_modules\fsevents):  
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})  
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\expo-cli\node_modules\webpack-dev-server\node_modules\chokidar\node_modules\fsevents):  
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})  
npm WARN @expo/configure-splash-screen@0.2.1 requires a peer of expo-splash-screen@* but none is installed. You must install peer dependencies yourself.  
npm WARN @pmmmwh/react-refresh-webpack-plugin@0.3.3 requires a peer of react-refresh@0.8.2 but none is installed. You must install peer dependencies yourself.  
+ expo-cli@3.27.11  
added 116 packages from 88 contributors, removed 83 packages, updated 87 packages and moved 13 packages in 43.659s
```

Processo de instalação do Expo-CLI no dispositivo móvel:

- Abra a loja de aplicativos do celular ([App Store](#) ou [Play Store](#)), pesquise pelo aplicativo **Expo** e instale em seu dispositivo.



Criando e iniciando o projeto React Native com Expo-CLI:

1. Execute o Prompt de Comando como administrador, vá para a pasta raiz com o comando `cd \`, insira o comando `expo init [nome do projeto]` e, apertando **ENTER**, escolha a opção **blank** para gerar um projeto em branco. Quando a mensagem “*Your Project is ready!*” aparecer, seu projeto estará criado;

```
Administrador: Prompt de Comando - expo init exemploIDE
C:\>expo init nomeAplicativo

? Choose a template: (Use arrow keys)
  ----- Managed workflow -----
> blank a minimal app as clean as an empty canvas
  blank (TypeScript) same as blank but with TypeScript configuration
  tabs (TypeScript) several example screens and tabs using react-navigation
                        and TypeScript

? Choose a template: expo-template-blank

[?] Using Yarn to install packages. You can pass --npm to use npm instead.

[✓] Downloaded and extracted project files.
[✓] Installed JavaScript dependencies.

[?] Your project is ready!

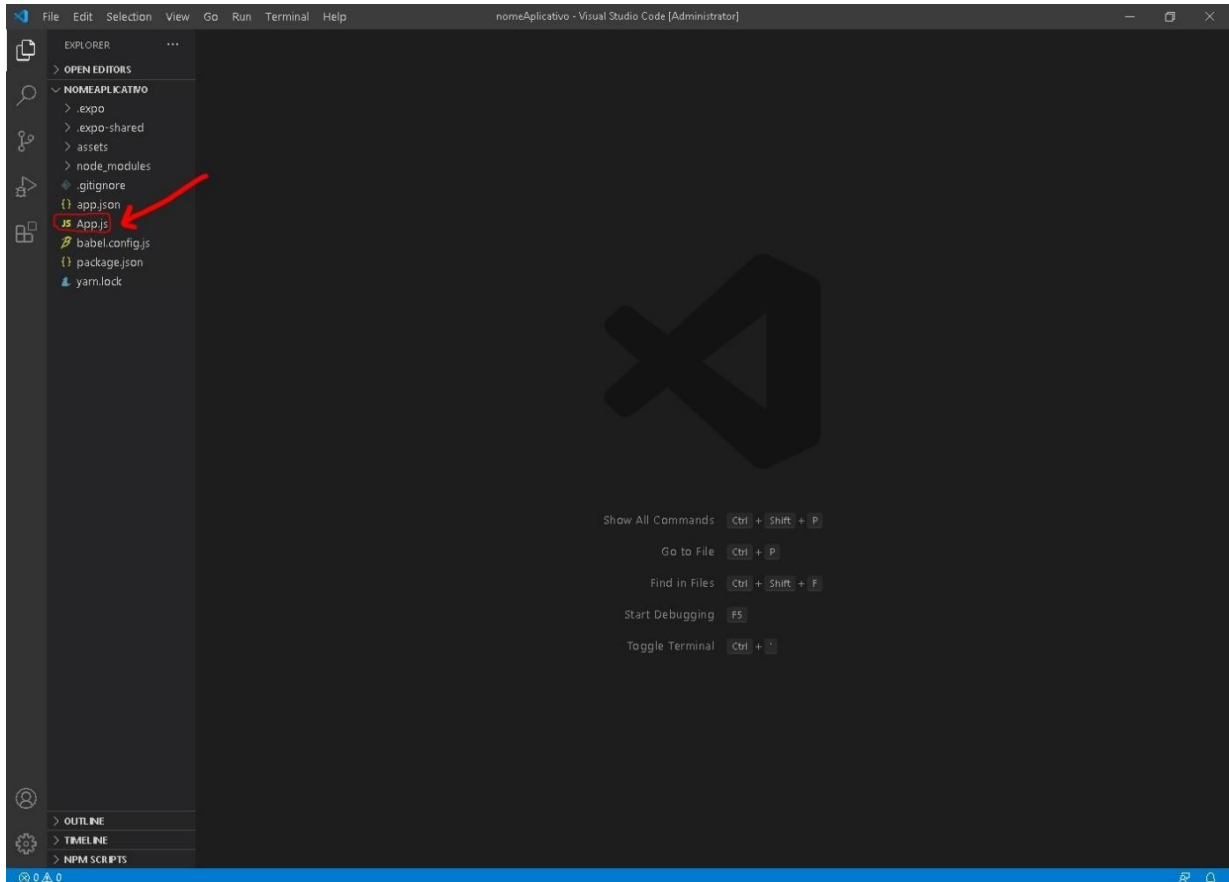
To run your project, navigate to the directory and run one of the following yarn
commands.
```

2. Para abrir o projeto diretamente na IDE (Visual Studio Code), entre na pasta do projeto com `cd [nome do projeto]` e insira o comando `code .`;

```
ca. Administrador: Prompt de Comando
C:\>cd nomeAplicativo

C:\nomeAplicativo>code .
```


3. O Visual Studio Code será aberto, clique no **arquivo App.js** para editar o projeto;

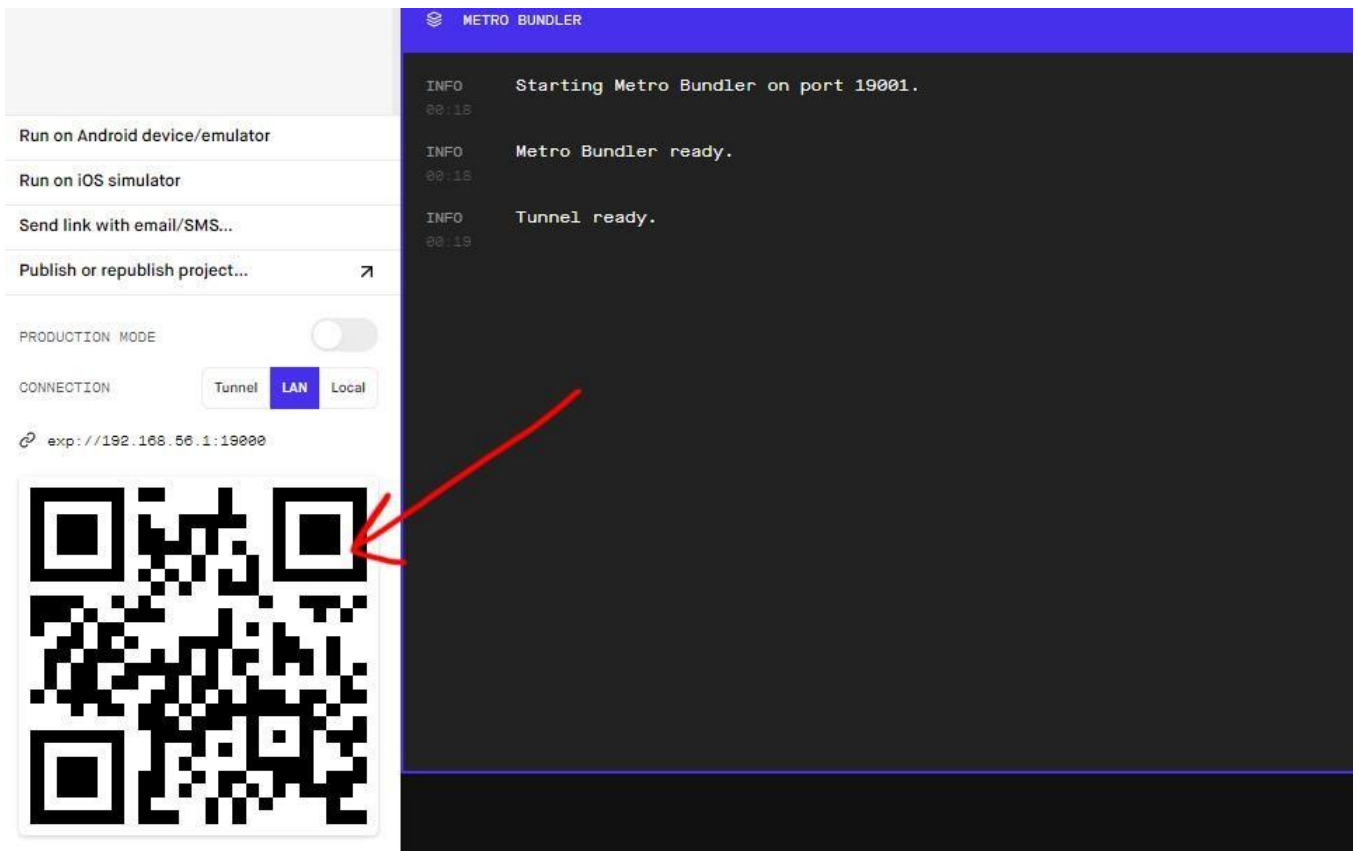


4. Volte para o Prompt de Comando, entre na pasta do projeto digitando **cd [nome do projeto]** e insira o comando **expo start**;

Administrador: Prompt de Comando

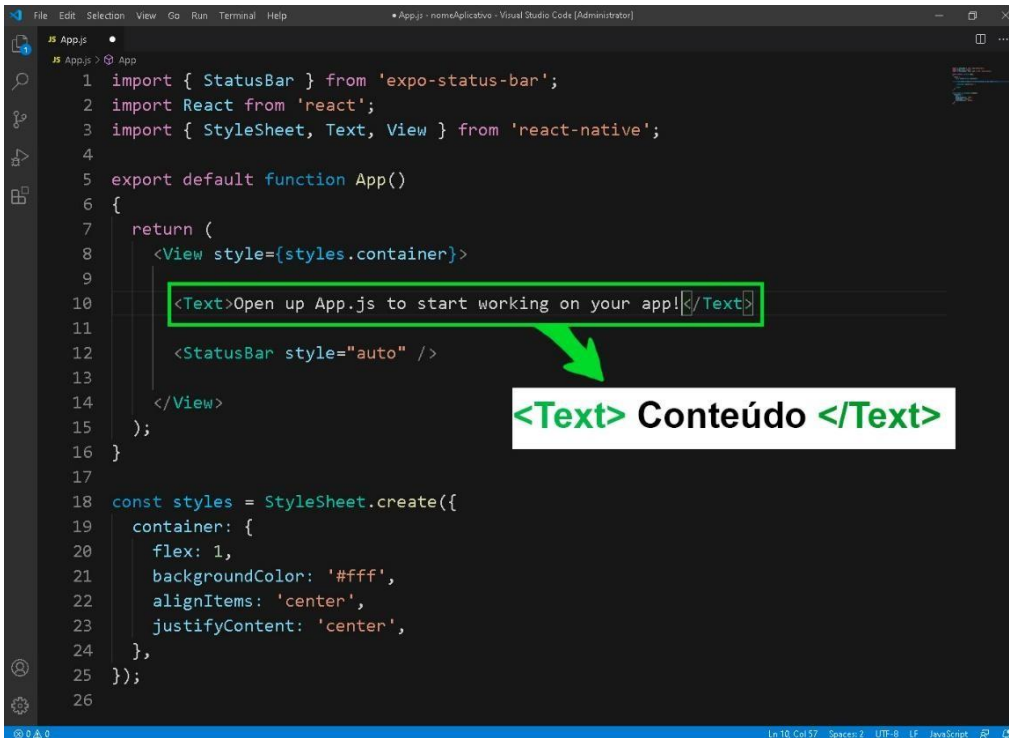
```
C:\>cd nomeAplicativo  
C:\nomeAplicativo>expo start
```

5. Escaneei o **Qr Code** que será aberto no navegador.



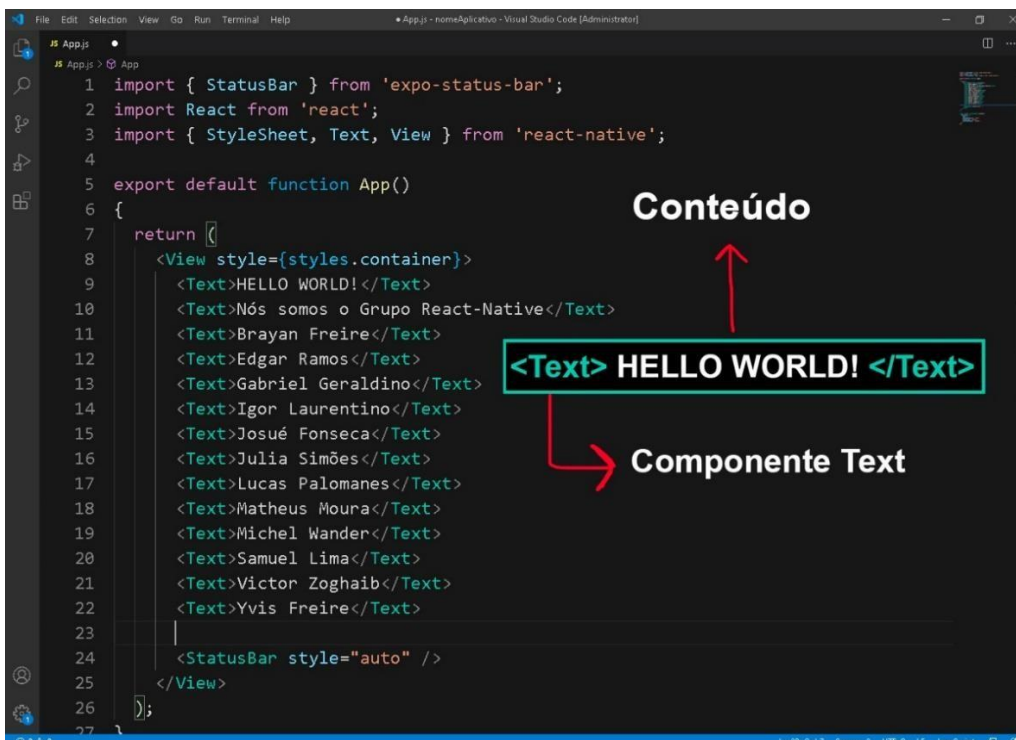
Programando o “Hello World:

1. No Visual Studio Code, edite o conteúdo do componente `<Text>` e escreva **“Hello World”**. Depois de editar o código, lembre-se de apertar **CTRL + S** para salvar o conteúdo e, dessa forma, atualizar a aplicação no dispositivo móvel;



```
1 import { StatusBar } from 'expo-status-bar';
2 import React from 'react';
3 import { StyleSheet, Text, View } from 'react-native';
4
5 export default function App()
6 {
7   return (
8     <View style={styles.container}>
9       <Text>Open up App.js to start working on your app!</Text>
10      <StatusBar style="auto" />
11    </View>
12  );
13 }
14
15 const styles = StyleSheet.create({
16   container: {
17     flex: 1,
18     backgroundColor: '#fff',
19     alignItems: 'center',
20     justifyContent: 'center',
21   },
22 });
```

2. Exemplo do código do projeto e da execução em um dispositivo móvel.



```
1 import { StatusBar } from 'expo-status-bar';
2 import React from 'react';
3 import { StyleSheet, Text, View } from 'react-native';
4
5 export default function App()
6 {
7   return (
8     <View style={styles.container}>
9       <Text>HELLO WORLD!</Text>
10      <Text>Nós somos o Grupo React-Native</Text>
11      <Text>Brayan Freire</Text>
12      <Text>Edgar Ramos</Text>
13      <Text>Gabriel Geraldino</Text>
14      <Text>Igor Laurentino</Text>
15      <Text>Josué Fonseca</Text>
16      <Text>Julia Simões</Text>
17      <Text>Lucas Palomanes</Text>
18      <Text>Matheus Moura</Text>
19      <Text>Michel Wander</Text>
20      <Text>Samuel Lima</Text>
21      <Text>Victor Zoghaib</Text>
22      <Text>Yvis Freire</Text>
23
24      <StatusBar style="auto" />
25    </View>
26  );
27 }
```


React Native é um framework para desenvolvimento de aplicações móveis, capaz de criar aplicações tanto para dispositivos Android quanto para dispositivos iOS, utilizando apenas a linguagem Javascript. Todo o código escrito no React Native é convertido em código nativo do sistema operacional, elevando consideravelmente o desempenho.

Para entender melhor como este framework mobile funciona é necessário dar certa atenção a alguns conceitos como o JSX, componentes e “state”. Antes do passo a passo do projeto para o cálculo da área do triângulo, será analisado o código padrão presente na criação de um projeto no Expo, como uma forma de entender e fixar alguns desses conceitos fundamentais. Veja no exemplo abaixo:

```
import { StatusBar } from 'expo-status-bar';
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Text>Open up App.js to start working on your app!</Text>
      <StatusBar style="auto" />
    </View>
  );
}

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

Inicialmente, é possível observar a importação de alguns objetos. Dentre os vários componentes importados acima, falaremos um pouco sobre o JSX, que vem junto à importação do “react”.

Ao observar o trecho de código ao lado do “return”, que entre está parênteses, vemos essa sintaxe formada por tags, muito semelhante ao HTML. Esta sintaxe é chama de JSX, ou Javascript XML. O React (framework web usado como base para o React Native) faz uso do JSX para utilizar as tags HTML e através delas criar outros componentes, porém, o React Native utiliza tags nativas, como por exemplo <View>, <Text>, <TextInput>, <Image>, entre outros.

Observe a função App sendo exportada, esta função retorna uma View (semelhante a uma div em HTML) contendo um componente de texto e a StatusBar (componente do expo que facilita a manipulação da barra de status). Esta função é nada mais nada menos que um componente do React, com essa mesma sintaxe podemos criar nossos próprios componentes, importa-los, e utilizá-los como for necessário.

Finalmente, temos o objeto “styles” que contém os parâmetros para estilização dos componentes, muito semelhante ao CSS no desenvolvimento web. No entanto ao invés de se utilizar um arquivo CSS separado, é criado um objeto contendo toda estilização do componente.

Aplicativo 1 - Cálculo da área do triângulo

No estado atual do código temos somente uma tela em branco. Para este projeto precisaremos ter entrada de dados para a base e para a altura do triângulo, em seguida, após o cálculo, será exibido o resultado. Veja abaixo o resultado atual do código:

13:13 2,0KB/s

Open up App.js to start working on your app!



Primeiramente, será necessário inserir um componente para receber dados do usuário. O componente que será usado é o `<TextInput>`, que define uma caixa de texto. Veja os passos abaixo.

Importe a função “`useState`” da biblioteca do React e os componentes “`Button`” e “`TextInput`” da biblioteca do React Native.

```
import { StatusBar } from 'expo-status-bar';
import React, { useState } from 'react';
import { StyleSheet, Text, View, Button, TextInput } from 'react-native';
```

Em seguida, crie as variáveis que serão usadas no decorrer da aplicação. Para tal, utilizaremos a função `useState`, que é usado para manipular de forma dinâmica o “state” (ou estado) da aplicação, ou seja, seus dados. A função “`useState`” declara uma variável de estado, recebendo seu valor inicial como parâmetro, e retorna um “array” contendo o estado e a função para alterá-lo. Convencionalmente, declara-se duas constantes, que desconstroem o “array” retornado, sendo uma para acessar o valor em si, como visto abaixo na constante “`base`”, e outra para receber a função que irá alterar este valor, também visto abaixo na função “`setBase`”.

```
const [base, setBase] = useState('');
const [altura, setAltura] = useState('');
const [area, setArea] = useState('');
```

Crie dois `<TextInput>`, um para inserir o valor da base e outro para inserir o valor da altura. Além disso, altere o texto padrão que veio junto ao projeto para algo que condiga com o proposito da aplicação, mas este passo é totalmente opcional.

```
return (
  <View style={styles.container}>
    <Text>Insira os dados abaixo para calcular a área do triângulo.</Text>
```

```
<TextInput
  placeholder='Base'
  style={{height: 40, textAlign: 'center'}}
  keyboardType={'numeric'}
  onChangeText={base => setBase(base)}
/>
<TextInput
  placeholder='Altura'
  style={{height: 40, textAlign: 'center'}}
  keyboardType={'numeric'}
  onChangeText={altura => setAltura(altura)}
/>
```

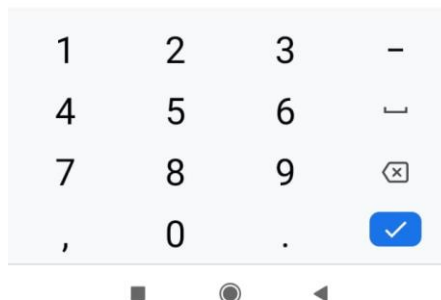

Na declaração acima, criamos dois “TextInput”, definindo seus respectivos nomes, uma altura, e o tipo de teclado a ser utilizado na inserção, que neste caso é o numérico, para evitar eventuais erros. Observe a propriedade “onChangeText” e seu valor, ela define que a cada vez que houver uma alteração no campo, o valor da base ou altura será alterado. Veja o resultado na imagem seguinte.

16:20 17,4KB/s

Insira os dados abaixo para calcular a área do triângulo.

Base

Altura



Terminando o projeto, temos a declaração do botão que irá tratar estes dados e o componente de texto que exibirá o resultado após o cálculo.

Para executar um código Javascript no JSX, é necessário colocá-lo entre chaves, como ocorre no código abaixo que está dentro do componente de texto. O trecho verifica se o valor da área existe ou se é diferente de uma “string” vazia, em caso positivo, será exibida a área, do contrario, nada será exibido.

```
<Button
  title='Calcular'
  onPress={calcularArea}
/>
<Text>{area ? `Resultado: ${area}` : ''}</Text>
```

Observando as propriedades desse botão, além da propriedade que define o título do mesmo, temos o evento “onPress”, que executa uma função sempre que o botão for pressionado. Neste caso, o botão executa a função “calcularArea”, mostrada abaixo.

```
function calcularArea () {
  if (base > 0 && altura > 0) {
    setArea((parseFloat(base) * parseFloat(altura)) /2);
  } else {
    setArea('');
  }
}
```

A função verifica se os valores são maiores que zero, em caso positivo, executa a função “setArea” que recebe como parâmetro o cálculo da área do triângulo, atribuindo o resultado à constante “area”.

Insira os dados abaixo para calcular a área do triângulo.

Base

Altura

CALCULAR

Após seguir todos os passos acima, chegamos ao fim do projeto. Segue abaixo um exemplo do funcionamento do aplicativo e, em seguida, todo código do projeto:

Insira os dados abaixo para calcular a área do triângulo.

5

4

CALCULAR

Resultado: 10

```
import { StatusBar } from 'expo-status-bar';
import React, { useState } from 'react';
import { StyleSheet, Text, View, Button, TextInput } from 'react-native';

export default function App() {

  const [base, setBase] = useState('');
  const [altura, setAltura] = useState('');
  const [area, setArea] = useState('');

  function calcularArea () {
    if (base > 0 && altura > 0) {
      setArea((parseFloat(base) * parseFloat(altura)) /2);
    } else {
      setArea('');
    }
  }

  return (
    <View style={styles.container}>
      <Text>Insira os dados abaixo para calcular a área do triângulo.</Text>
      <TextInput
        placeholder='Base'
        style={{height: 40, textAlign: 'center'}}
        keyboardType={'numeric'}
        onChangeText={base => setBase(base)}
      />
      <TextInput
        placeholder='Altura'
        style={{height: 40, textAlign: 'center'}}
        keyboardType={'numeric'}
        onChangeText={altura => setAltura(altura)}
      />
      <Button
        title='Calcular'
        onPress={calcularArea}
      />
      <Text>{area ? `Resultado: ${area}` : ''}</Text>
    <StatusBar style="auto" />
  )
}
```

```
</View>
);
}

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

Aplicativo 2 - GPS

O recurso que utilizaremos nessa apostila, será o recurso nativo que os próprios aparelhos possuem hoje em dia: O GPS.

- Crie um novo projeto
- abra-o no Visual Studio Code
- abra o App.js

Após isso, inicie a emulação pelo “expo” com o comando “expo start”, e comece a emulação pelo app do expo no celular.

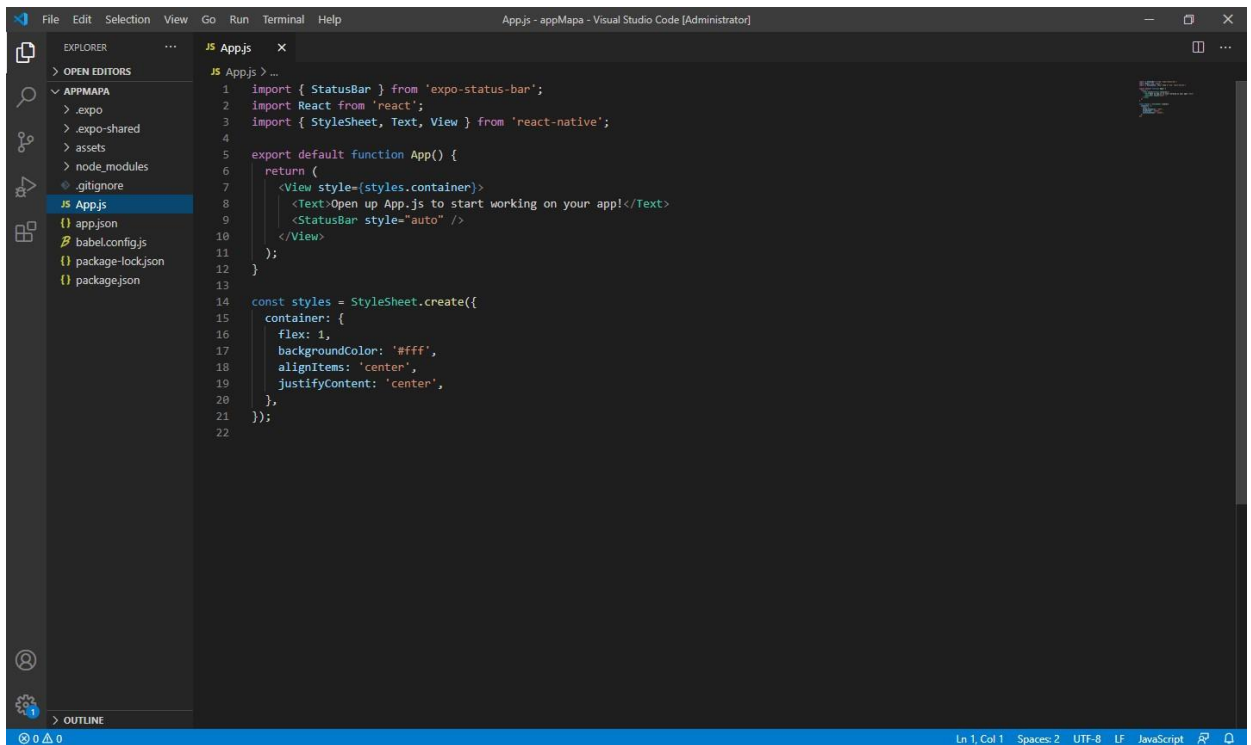


Open up App.js to start working on your app!



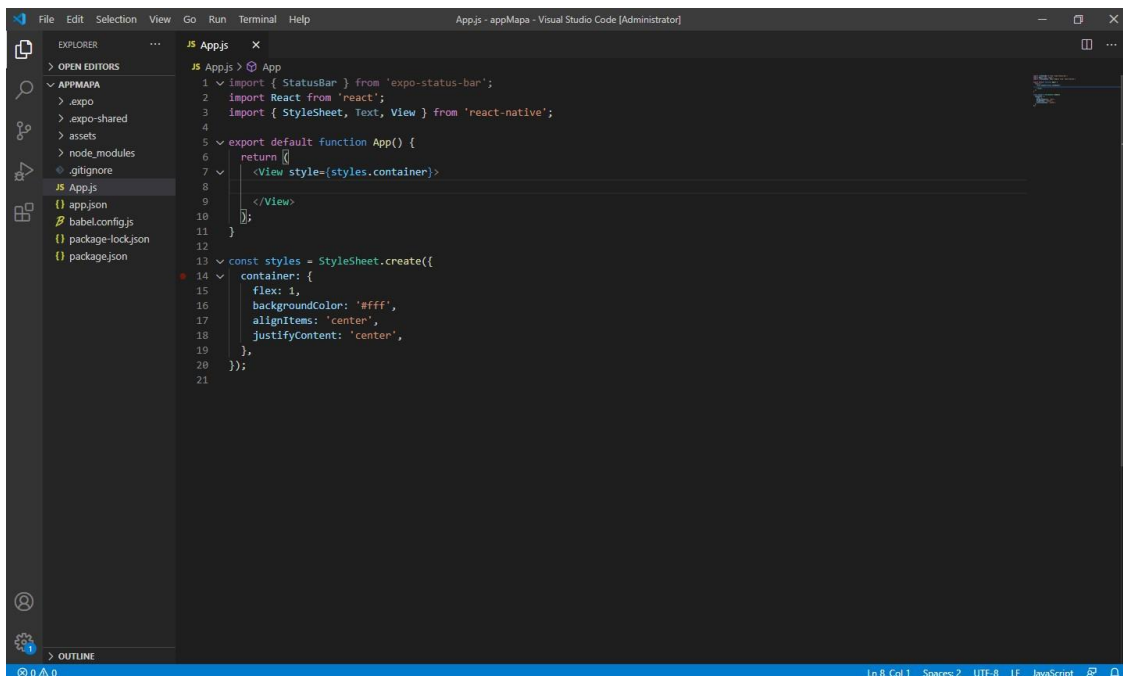
A tela acima será exibida no celular;

No Visual Studio Code será possível ver esta:



```
1 import { StatusBar } from 'expo-status-bar';
2 import React from 'react';
3 import { StyleSheet, Text, View } from 'react-native';
4
5 export default function App() {
6   return (
7     <View style={styles.container}>
8       <Text>Open up App.js to start working on your app!</Text>
9       <StatusBar style="auto" />
10    </View>
11  );
12
13
14 const styles = StyleSheet.create({
15   container: {
16     flex: 1,
17     backgroundColor: '#fff',
18     alignItems: 'center',
19     justifyContent: 'center',
20   },
21 });
22
```

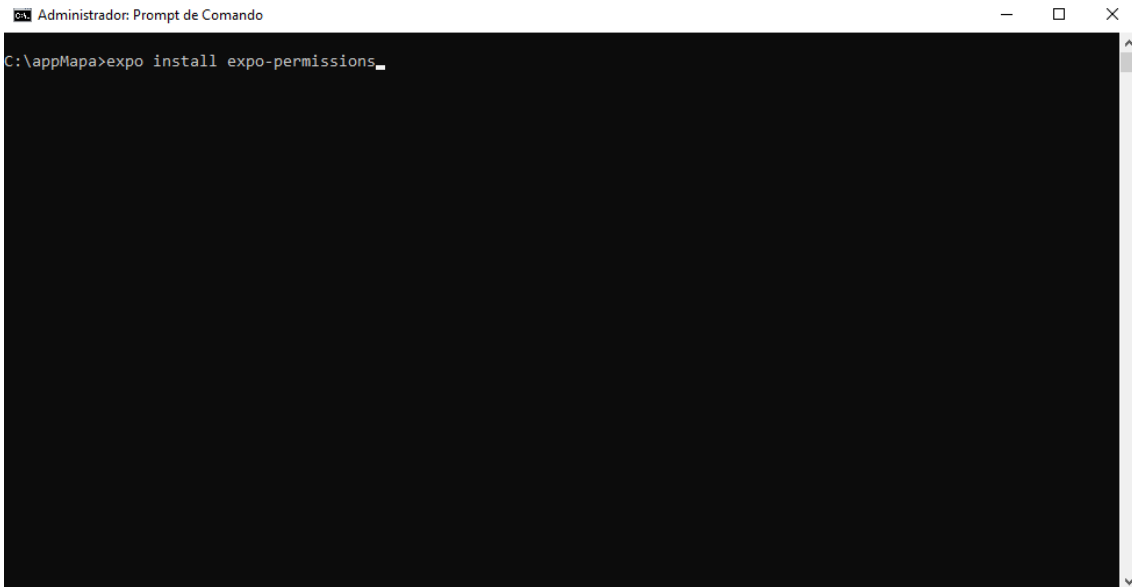
Apague todo o conteúdo dentro da tag <View>.



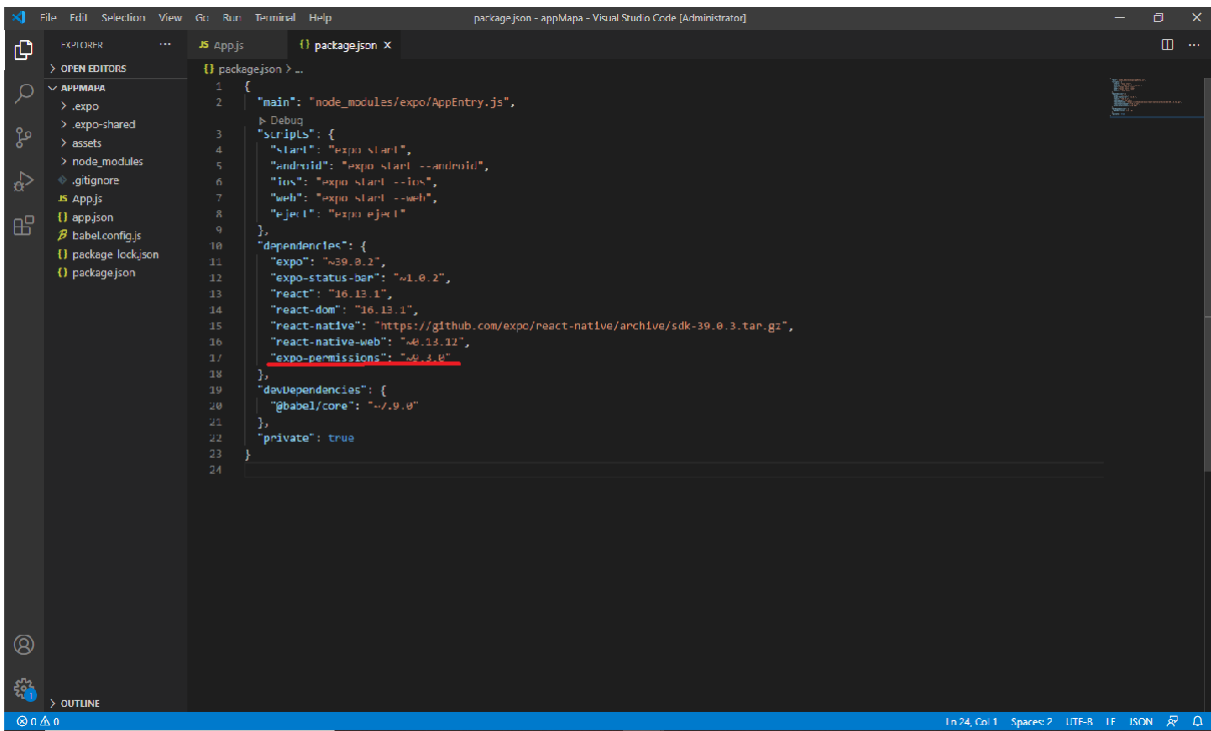
```
1 import { StatusBar } from 'expo-status-bar';
2 import React from 'react';
3 import { StyleSheet, Text, View } from 'react-native';
4
5 export default function App() {
6   return (
7     <View style={styles.container}>
8
9     </View>
10  );
11
12
13 const styles = StyleSheet.create({
14   container: {
15     flex: 1,
16     backgroundColor: '#fff',
17     alignItems: 'center',
18     justifyContent: 'center',
19   },
20 });
21
```

Então como iremos criar um aplicativo que se utiliza da localização atual do GPS do aparelho e exibe-a em um mapa, precisaremos primeiro de acesso ao GPS do celular. Então para fazer isso, precisaremos da permissão para acessar o GPS.

Instalaremos uma biblioteca chamada “expo-permissions”, indo até o prompt de comandos e digitando “expo install expo-permissions”.



Pode-se observar no “package.json” a presença da biblioteca expo-permissions seguida da sua versão em “dependencies”.



Então, voltando a App.js, vamos importar a biblioteca permissions adicionando a linha:

```
import * as Permissions from 'expo-permissions';
```

Vamos também apagar a function App e mudar para:

```
export default class App extends Component {
```

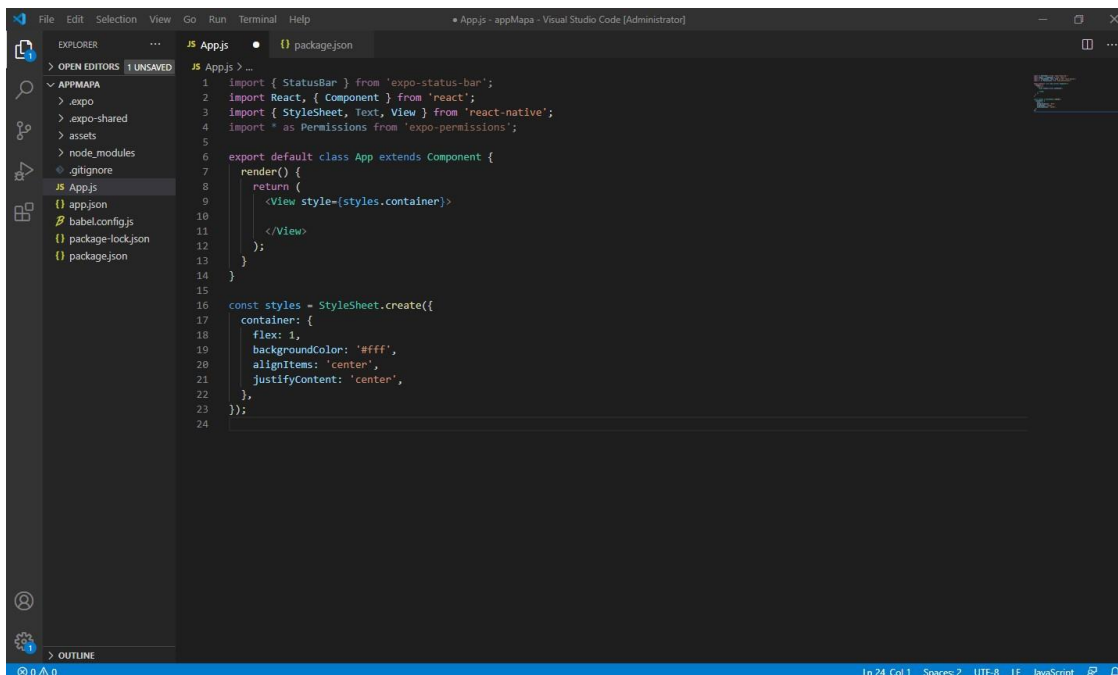
Será necessário escrever o comando “render” antes do return, que nos permitirá usar mais componentes na tela, e vamos também importar a biblioteca Componente, alterando a linha de importação do React:

```
import React from 'react';
```

para:

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

Resultando em:



```
1 import { StatusBar } from 'expo-status-bar';
2 import React, { Component } from 'react';
3 import { StyleSheet, Text, View } from 'react-native';
4 import * as Permissions from 'expo-permissions';
5
6 export default class App extends Component {
7   render() {
8     return (
9       <View style={styles.container}>
10
11         </View>
12     );
13   }
14 }
15
16 const styles = StyleSheet.create({
17   container: {
18     flex: 1,
19     backgroundColor: '#fff',
20     alignItems: 'center',
21     justifyContent: 'center',
22   },
23 });
24
```

Então vamos começar com a parte de pedir a permissão para o GPS.

Primeiro vamos criar uma função para dentro dela, colocar todo o processo de pedir permissão e então conseguir a localização. Vamos chama-la de getLocation. E então defini-la antes da função “render()” digitando a seguinte linha:

```
_getLocation = async () => {
}
```

Como se trata de uma função assíncrona. Definimos ela usando async e arrow function(=>). Adicione o comando:

```
let { status } = await Permissions.askAsync(Permissions.LOCATION);
```

Após solicitada a permissão de acesso ao GPS, precisamos conferir se o acesso foi garantido. Vamos fazer um “if else” para checar:

```
if (status !== 'granted') {
} else {}
```

Criaremos fora da função um state com os objetos: errorMessage e loaded

```
state = {
  places:
  [
    {
      errorMessage:
      null, loaded:
      false,
```

Ficando assim:

```
JS App.js • {} package.json
JS App.js > App > _getLocation
1 import { StatusBar } from 'expo-status-bar';
2 import React, { Component } from 'react';
3 import { StyleSheet, Text, View } from 'react-native';
4 import * as Permissions from 'expo-permissions';
5
6 export default class App extends Component {
7
8   state = {
9     places: [
10      {
11        errorMessage: null,
12        loaded: false,
13      }
14    ]
15  }
16
17  componentDidMount() {
18    this._getLocation();
19  }
20
21  _getLocation = async () => {
22    let { status } = await Permissions.askAsync(Permissions.LOCATION);
23    if (status !== 'granted') {
24
25    } else {
26
27    }
28  }
29
30  render() {
31    return (
32      <View style={styles.container}>
33
34      </View>
35    );
36  }
37 }
38
39 const styles = StyleSheet.create({
40   container: {
```


Para poder começar a ter um resultado em tempo real na aplicação, iremos criar uma outra função chama `componentDidMount()`. Ela é uma função que é executada assim que o app é carregado, e dentro dela vamos mandar executar nossa função `_getLocation`, com o código:

```
componentDidMount() {  
  this._getLocation();  
}
```

Salvando o Código você já deve ter a resposta na tela do seu celular no emulador; O aplicativo pedindo a permissão de acesso ao GPS:



Porem permitindo ou não ele ainda não vai ter resultado.

```
this.setState({
  errorMessage: 'Permissão para acessar a localização do GPS
negada.', loaded: true
});
```

Seguindo então, dentro do if que criamos vamos passar o que deve ser feito caso a permissão seja negada. Atribuiremos valores aos objetos do state que criamos antes:

E então passamos um alert para exibir a mensagem

Agora precisaremos instalar a biblioteca “expo-location” na nossa aplicação para poder começar a trabalhar com o GPS, então da mesma forma que instalamos o expo- permissions, vá até o CMD e digite: npm install expo-location

Então no nosso aplicativo iremos importar a biblioteca expo-location com a linha de comando:

```
import * as Location from 'expo-location';
```

Dentro do else ficara todo o processo de capturar a localização:

```
let location = await Location.getCurrentPositionAsync({ enableHighAccuracy: true });
this.setState({ location, loaded: true, errorMessage: null });
```

adicionando essas duas linhas dentro do else já teremos a captura de um objeto com os dados providos pelo GPS, mas para isso, precisamos primeiro criar um state chamado location, e definir o seu valor inicial como null.

Então, dentro de state, adicionamos location: null

```
state = {
  places: [
    {
      errorMessage:
null,      loaded:
false,    location:
null
    }
  ]
}
```

Agora que temos um objeto com toda a informação da localização, precisaremos dissecá-lo.

Para conseguir a latitude e longitude criaremos uma constante onde iremos atribuir os valores de latitude e longitude:

```
const { latitude, longitude } = this.state.location.coords;
```

A partir da latitude e longitude iremos obter o endereço, mas primeiramente precisaremos criar um novo state para armazenar o objeto do endereço que será devolvido, então no state adicionamos locationAddress, com valor inicial nulo:

```
state = {
  places: [
    {
      errorMessage:
      null, loaded: false,
      location:      null,
      locationAddress: null,
    }
  ]
}
```

Assim dentro do else poderemos adicionar a linha de código que preencherá esse state

```
let locationAddress = await Location.reverseGeocodeAsync({ latitude:
  gitude: longitude, useGoogleMaps: true });
```

O comando “reverseGeocodeAsync” retorna um objeto contendo todo o endereço de certa latitude e longitude, por isso precisaremos passar tais parâmetros dentro dele, e então passaremos as consts criadas anteriormente, que nelas foram armazenadas a latitude e longitude. E agora é só dar um setState para armazenar o objeto no state.

```
this.setState({ locationAddress });
```

E agora para dissecar esse objeto primeiro definimos seu valor em uma variável constante:

```
const address = this.state.locationAddress;
```

e como o objeto do endereço está contido em um array, precisaremos remove-lo. Para isso definimos cada variável que queremos tirar como apenas um array e passaremos como atribuição de valor o array todo, e usaremos um debugger:

```
const [{street, subregion, region, country, postalCode, district}] = address;
debugger
```

cada const contém o mesmo nome que cada objeto dentro de address, então cada um vai pegar o seu respectivo valor, agora só precisamos criar um state para cada um desses:

```
street: null,
subregion: null,
```

```
        region:
    null,    country:
    null, postalCode:
    null,
```

com os states inseridos teremos o seguinte:

```
state = {
  places: [
    {
      errorMessage:
    null, loaded: false,
    location:      null,
    locationAddress: null,
    street:        null,
    subregion:     null,
    region:        null,
    country:       null,
    postalCode:    null,
    district: null
    }
  ]
}
```

E agora é só atribuir os valores que pegamos com um setState:

```
    this.setState({ street: street, subregion: subregion, region: region,
country:
```

e então fechamos o else, tendo o seguinte código:

```
    else {
      let location = await Location.getCurrentPositionAsync({ enableHighAccuracy: true });
      this.setState({ location, loaded: true, errorMessage:
null }); const { latitude, longitude } =
this.state.location.coords;
      let locationAddress = await Location.reverseGeocodeAsync({ latitude: latitude, longitude: longitude, useGoogleMaps: true });
      this.setState({ locationAddress });
      const address = this.state.locationAddress;
      const [{street, subregion, region, country, postalCode, district}]
= add
      ress;
      debugger
```

Com a parte lógica pronta, vamos para a parte gráfica e a exibição da localização em um mapa. Primeiramente precisaremos de uma biblioteca de Mapas, para instalar essa biblioteca digitamos no CMD: `npm install react-native-maps`

E então importar ela com o comando:

```
import MapView from 'react-native-maps';
```

Agora partindo para a render, dentro dela iremos fazer alguns if's para checar se o mapa foi carregado e se houve algum erro, e então dentro desses if's iremos executar o return:

```
if(this.state.loaded) {
  if(this.state.errorMessage) {
    return (
      <View style={styles.container}>
        <Text>{JSON.stringify(this.state.errorMessage)}</Text>
      </View>
    );
  } else if(this.state.location) {
    return (
      <View style={styles.container}>
        </View>
      );
    }
  } else {
    return (
      <View style={styles.container}>
        <Text>Espere...</Text>
      </View>
    );
  }
}
```

Então dentro do return que ficou apenas com uma View vazia será onde o mapa vai ser carregado. Iremos adicionar a linha:

```
<StatusBar style='auto' />
```

Para personalizar a barra superior do celular enquanto o app roda. E logo em seguida adiciono a tag do mapa:

```
<MapView>
  </MapView>
```

Na parte de Styles, adicionaremos o seguinte style para o mapa:

```
mapStyle: {  
  position:  
'absolute', top: 0,  
  left: 0,  
  right: 0,  
  bottom: 0,
```

E então na tag MapView adicionamos o comando:

```
style={styles.mapStyle}
```

para que o style que criamos seja usado no mapa, ficando a tag assim:

```
<MapView  
  style={styles.mapStyle}  
>
```

Agora já é possível ver o mapa no aplicativo, iremos agora fazer com que ele mostre sua localização. Porém antes de fazer com que mostre sua localização, iremos impedir que o usuário possa mover, rodar, dar zoom etc, no mapa, e faremos isso adicionando os seguintes comandos dentro da tag MapView:

Para desativar a rotação:

```
rotateEnabled={false}
```

Para desativar o scroll:

```
scrollEnabled={false}
```

Para desativar o zoom:

```
zoomEnabled={false}
```

Para desativar a exibição de pontos de interesse do mapa (por exemplo, shoppings, escolas, padarias, mercados etc):

```
showsPointsOfInterest={false}
```

Para desativar a exibição de prédios:

```
showsBuildings={false}
```

E vamos adicionar uma ref na tag também:

```
ref={map => this.mapStyle = map}
```

É esperado que o código fique assim:

```
<MapView
  ref={map => this.mapStyle = map}
  style={styles.mapStyle}
  rotateEnabled={false}
  scrollEnabled={false}
  zoomEnabled={false}
  showsPointsOfInterest={false}
  showsBuildings={false}
 />
```

E você poderá observar na emulação do aplicativo que não conseguirá mexer no mapa, ele ficará estático. Precisamos agora adicionar a localização que o mapa irá mostrar, para isso usaremos o `region`, adicionando essa linha dentro da tag `MapView`:

```
      region={{
        latitude:
latitude, longitude:
longitude,
latitudeDelta: 0.0142,
longitudeDelta: 0.0231,
```

`latitudeDelta` e `longitudeDelta` são valores que determinarão o zoom do mapa, e os valores de `latitude` e `longitude` são variáveis que conterão os valores que pegamos anteriormente, e para atribuir tais valores nelas, iremos criar uma `const` com seus nomes e atribuir os valores como mostrado abaixo:

```
const { latitude, longitude } = this.state.location.coords;
```

esse código é adicionado antes do `return`, ficando o código assim:

```
else if(this.state.location) {
  const { latitude, longitude } = this.state.location.coords;

  return (
    <View style={styles.container}>
      <StatusBar style='auto' />
      <MapView
        ref={map => this.mapStyle
= map} region={{
          latitude:
latitude, longitude:
longitude,
latitudeDelta: 0.0142,
longitudeDelta: 0.0231,
        }}
        style={styles.map
```

```

        showsBuildings={false}
      >

    </MapView>
  </View>

```

Pode-se observar que no app o mapa já exibe a sua localização, porém sem nenhuma indicação de onde você realmente está. Para adicionar essa marcação iremos adicionar os seguintes comandos:

```

    { this.state.places.map(place => (
      <MapView.Marker />
    ))

```

A `MapView.Marker` será a responsável para adicionar a marcação, porém ela precisa de algumas outras informações para adicionar esta marcação. Então dentro da tag `MapView.Marker` adicionaremos um `ref`:

```

    ref={mark => place.mark = mark}

```

e uma `key`:

```

    key={place.id}

```

e então as coordenadas da marcação:

```

    coordinate={{
      latitude: latitude,
      longitude: longitude,
    }}

```

E adicionaremos uma caixa de texto em cima da marcação, onde você pode adicionar alguma informação, para tal adicionamos a linha abaixo dentro da tag `MapView.marker`:

```

    title={place.title}
    description={place.description}

```

Você perceberá que não vai aparecer nada, pois temos que adicionar nos `states` o `title` e a `description`, e atribuindo o valor em texto para o que você vai querer exibir, e adicionaremos também um `id` que será igual à 1:

```

    id: 1,
    title: 'Você está aqui!',
    description: 'Essa é sua localiza

```


E a tag `MapView.Marker` estará assim:

```
{ this.state.places.map(place => (  
  <MapView.Marker  
    ref={mark => place.mark =  
mark}      title={place.title}  
description={place.description}  
key={place.id}  
coordinate={{  
  latitude: latitude,  
  longitude: longitude,  
  }}  
  />  
)
```

Voltando a tag `MapView`, adicionaremos o comando `onMapReady`, que irá executar uma função assim que o mapa terminar de ser montado:

```
onMapReady={this._mapReady}
```

E fora da render iremos adicionar a seguinte função:

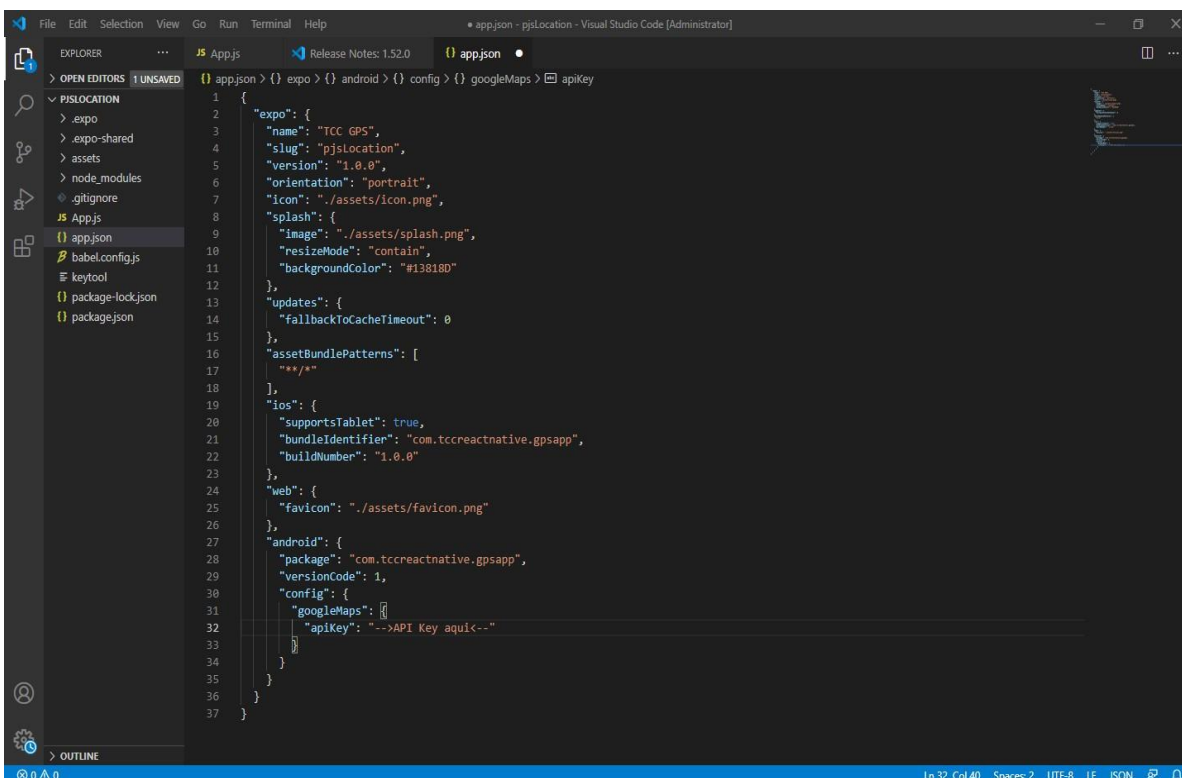
```
_mapReady = () => {  
  this.state.places[0].mark.showCallout();  
};
```

Que irá mover o nosso mapa e a marker para as coordenadas corretas caso ele não esteja.

Para usar o mapa da API do google é necessário gerar uma chave que o google disponibiliza. Para gerar essa chave é necessário ter uma conta do google, ir até o site: <https://console.cloud.google.com/home/dashboard>, ao logar com sua conta do google, clique em “Criar”, para criar um novo projeto, dê um nome a ele e então após finalizar a criação você será redirecionado para o dashboard do seu projeto. No dashboard clique em “APIs e Serviços>Biblioteca>Maps SDK for Android”, agora clique em Ativar, vá até “APIs e Serviços>Credenciais>Criar Credenciais”, após criar as credenciais copie sua API key, vá até app.json no seu App, e em “android”: adicione a linha:

```
"config": {  
  "googleMaps": {  
    "apiKey": "-->API Key aqui<--"  
  }  
}
```

Portanto, temos como resultado:



Agora já temos o mapa funcionando e mostrando a localização, iremos agora adicionar uma telinha onde iremos colocar as informações de endereço que coletamos anteriormente. Para começar iremos importar o ScrollView, então dentro dos imports do react-native adicione ScrollView:

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

e adiciona a tag ScrollView depois da MapView:

```
<ScrollView></ScrollView>
```

Dentro da tag ScrollView iremos adicionar um style para ela e definir que ela é uma ScrollView horizontal:

```
<ScrollView
  style={styles.placesConta
iner} horizontal
></ScrollView>
```

E nos styles adicionaremos o seguinte style para a ScrollView:

```
placesContainer: {
  width: '100%',
  maxHeight: 200,
}
```

Dentro da Scroll iremos definir uma View com um style definido:

```
<ScrollView
  style={styles.placesCont
ainer} horizontal
>
  <View style={styles.place}></View>
```

E iremos adicionar o seguinte style:

```
place: {
  width: width - 40,
  maxHeight: 200,
  backgroundColor: '#fff',
  marginHorizontal: 20,
}
```

E mudaremos o style container para:

```
    contai  
ner: {
```

```

    alignItems: 'flex-
end',    justifyContent:
'flex-end',
    backgroundColor: '#fff'

```

Observe que, no style place, temos um “width” como valor do width, isso é uma variável que contém o tamanho total da tela, e iremos definir ela assim.

Primeiramente, após a importação da ScrollView, importaremos Dimensions:

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

Então, logo abaixo a área das importações definiremos uma variável global:

```
const { height, width } = Dimensions.get('window');
```

A biblioteca “Dimensions” serve para pegar as dimensões da tela.

Então dentro dessa View iremos adicionar o texto contendo as informações:

```

<Text>{ street + ", " + district + ", " + subregion }</Text>
<Text>{ region + ", " + country + ", " + postalCode + "\n"}</Text>

<Text>Latitude: { latitude }</Text>

```

E antes da return iremos definir as consts que darão valor as variáveis chamadas aqui:

```
const { street, subregion, region, country, postalCode, district } = this.state;
```

O Código final da ScrollView ficará assim:

```

<ScrollView
  style={styles.placesContainer} horizontal
>
  <View style={styles.place}>
    <Text>{ street + ", " + district + ", " + subregion }</Text>
    <Text>{ region + ", " + country + ", " + postalCode + "\n"}</Text>
    <Text>Latitude: { latitude }</Text>
    <Text>Longitude: { longitude }</Text>
  </View>

```

E o programa está pronto, o código final do programa ficará assim:

```

import { StatusBar } from 'expo-status-
bar'; import React, { Component } from
'react';

```

```

import * as Permissions from 'expo-permissions';
import * as Location from 'expo-location';
import MapView from 'react-native-maps';

const { height, width } = Dimensions.get('window');

export default class App extends Component {

  state = {
    places: [
      {
        id: 1,
        title: 'Você está aqui!',
        description: 'Essa é sua localização atual',
        errorMessage: null,
        loaded: false,
        location: null,
        locationAddress: null,
        street: null,
        subregion: null,
        region: null,
        country: null,
        postalCode: null,
        district: null
      }
    ]
  }

  componentDidMount() {
    this._getLocation();
  }

  _getLocation = async () => {
    let { status } = await Permissions.askAsync(Permissions.LOCATION);
    if (status !== 'granted') {
      this.setState({
        errorMessage: 'Permissão para acessar a localização do GPS negada.',
        loaded: true
      });
      alert(this.state.errorMessage);
    } else {
      let location = await Location.getCurrentPositionAsync({ enableHighAccuracy: true });
      this.setState({ location, loaded: true, errorMessage: null });
      const { latitude, longitude } = this.state.location.coords;
      let locationAddress = await Location.reverseGeocodeAsync({ latitude: latitude, longitude: longitude, useGoogleMaps: true });
      this.setState({ locationAddress });
      const address = this.state.locationAddress;
    }
  }
}

```

```

        const [{street, subregion, region, country, postalCode, district}] = address;
        debugger
        this.setState({ street: street, subregion: subregion, region: region, country: country, postalCode: postalCode, district: district });
    }
}

_mapReady = () => {
    this.state.places[0].mark.showCallout();
}

render() {

    if(this.state.loaded) {
        if(this.state.errorMessage) {
            return (
                <View style={styles.container}>
                    <Text>{JSON.stringify(this.state.errorMessage)}</Text>
                </View>
            );
        } else if(this.state.location) {
            const { latitude, longitude } = this.state.location.coords;
            const { street, subregion, region, country, postalCode, district } = this.state;

            return (
                <View style={styles.container}>
                    <StatusBar style='auto' />
                    <MapView
                        ref={map => this.mapStyle = map}
                        region={{
                            latitude: latitude,
                            longitude: longitude,
                            latitudeDelta: 0.0142,
                            longitudeDelta: 0.0231,
                        }}
                        style={styles.mapStyle}
                        rotateEnabled={false}
                        scrollEnabled={false}
                        zoomEnabled={false}
                        showsPointsOfInterest={false}
                        showsBuildings={false}
                        onMapReady={this._mapReady}
                    >
                        { this.state.places.map(place => (
                            <MapView.Marker
                                ref={mark => place.mark = mark}
                                title={place.title}

```

```

        description={place.description}
        key={place.id}
        coordinate={{
            latitude: latitude,
            longitude: longitude,
        }}
    />
    )))
</MapView>

<ScrollView
    style={styles.placesContainer}
    horizontal>
    <View style={styles.place}>
        <Text>{ street + ", " + district + ", " + subregion }</Text>
        <Text>{ region + ", " + country + ", " + postalCode + "\n"}<
/Text>
        <Text>Latitude: { latitude }</Text>
        <Text>Longitude: { longitude }</Text>
    </View>
    </ScrollView>
</View>
    );
}

} else {
    return (
        <View style={styles.container}>
            <Text>Espere...</Text>
        </View>
    );
}
}
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        alignItems: 'flex-end',
        justifyContent: 'flex-end',
        backgroundColor: '#fff'
    },
    mapStyle: {
        position: 'absolute',
        top: 0,
        left: 0,
        right: 0,
        bottom: 0,
    },
},

```



```
placesContainer: {  
  width: '100%',  
  maxHeight: 200,  
  },  
  
  place: {  
    width: width - 40,  
    maxHeight: 200,  
    backgroundColor: '#fff',  
    marginHorizontal: 20,  
  },  
}
```

Aplicativo 3 - Contatos

Após acessarmos os recursos nativos de GPS no celular, agora aprenderemos como acessar alguns dados presentes no aparelho, como os contatos salvos em hardware fazendo uma simples busca por um contato que esteja salvo.

Para começar, declaramos os seguintes estados em nosso App.json

```
state = {
  contacts: [],
  campos: [
    {
      pesquisa: null,
    }
  ],
  contactsFilter: [],
  switchValue: false,
}
```

Usaremos essas variáveis futuramente para armazenar os contatos presentes no celular, e também para a pesquisa do nosso programa.

Utilizando a função:

```
componentDidMount(){
  this._showContacts();
}
```

Faremos a primeira varredura dos contatos assim que carregarmos a página.

No terminal instale a biblioteca “expo-permissions” através do comando “npm install expo-permissions”. E então na área das importações, importe a biblioteca instalada como abaixo:

```
import * as Permissions from 'expo-permissions';
```

Agora, precisaremos pegar a permissão do celular. Usaremos as seguintes linhas de código

```
_showContacts = async (inputValue) => {  
  const permission = await Permissions.askAsync(Permissions.CONTACTS)  
  
  if(permission.status !== 'granted'){  
    return;  
  }  
}
```

Iremos criar dentro da função showContatcs a constante permissão seguida daquela linha de código que acessa o hardware do celular e solicita a permissão (um bloco de texto aparecerá similar aos diversos aplicativos que usamos diariamente).

No terminal instale a biblioteca “expo-contacts” através do comando “npm install expo-contacts”. E então na área das importações, importe a biblioteca instalada como abaixo:

```
import * as Contacts from 'expo-contacts';
```

Com a seguinte constante, conseguiremos o número do contato que estamos buscando

```
const contacts = await Contacts.getContactsAsync({  
  fields: [  
    Contacts.PHONE_NUMBERS  
  ],  
});
```

Agora faremos um If para testar a quantidade de contatos(não podendo ser igual ou menor que 0) e caso haja contatos, faremos a pesquisa.

```

if(contacts.total > 0) {
  this.setState({
    contacts: contacts.data
  })

  this.setState({ pesquisa: inputValue });
  input = this.state.pesquisa;

  let filtrado = this.state.contacts.filter(this._acharCorrespondencia);
  this.setState({contactsFilter: filtrado});
}
}

```

Podemos também antes de testar com o IF, criar o método que utilizaremos para a pesquisa, que consiste em:

```

_acharCorrespondencia(contato) {

  let termo = input;
  var regex = new RegExp(termo, 'gi');

  if(regex.test(contato.name))
    return true;

  if(contato.phoneNumbers)
    return contato.phoneNumbers.filter(n => regex.test (n.number)).length > 0;

  return false;
}

```

No return da render do app precisaremos de dois componentes para que o app funcione, um TextInput, onde será digitado o nome que for desejado pesquisar, e uma ScrollView, que é basicamente uma View que conforme se expande tem a opção de scroll, e uma função “map” que servirá para exibir os contatos.

Primeiro precisamos importar esses componentes como mostrado abaixo:

```

import { StyleSheet, Text, View, ScrollView, TextInput,
Pressable } from 'react-native';

```

E então adicione sua TextInput:

```
<TextInput
  style={styles.textInput}
  placeholder='Pesquisar'
  onChangeText={(inputValue) =>
    this._showContacts(inputValue)}
/>
```

E então a ScrollView com o map, para que ele exiba um a um os itens dentro do array que armazena os contatos:

```
<ScrollView
  style={styles.placesContainer}
  vertical
  >
  <View style={placeColor}>
    {
      this.state.contactsFilter.map(contact =>
        <Pressable
          style={({ pressed }) => [
            {
              backgroundColor: pressed
                ? onPressedColor
                : pressableColor
            },
            buttonColor
          ]}
          key={contact.id}
          onPress={() =>
            navigation.navigate('ContactInfos',
              { contact: contact, switchValue: this.state.switchValue })}
        >
        {{{ pressed }} => (
          <Text style={textColor}>
            {pressed ? contact.name : contact.name}
          </Text>
        )
      )
    }
  </View>
</ScrollView>
```

Veja que estamos usando um navigation no onPress, e esse navigation passa alguns parâmetros que são as informações do usuário que serão enviadas para a outra página ao clicar no botão. Para isso no terminal instale a biblioteca “@react-navigation/native”, usando o comando “npm install @react- navigation/native”, e importe a biblioteca como abaixo:

```
import { useNavigation } from '@react-navigation/native';
```

Adicione também a função:

```
function(props) {  
  const navigation = useNavigation();  
  return <MyBackButton {...props} navigation={navigation} />
```

E dentro da sua render declare a seguinte constante:

```
const { navigation } = this.props;
```

Agora definiremos as rotas do navigation para poder criar a segunda tela onde mostraremos as informações do contato. Para isso crie um novo arquivo dentro do projeto chamado “Navigation.js”. Agora precisaremos instalar uma biblioteca chamada “@react-navigation/stack”, no terminal digite “npm install @react- navigation/stack”. E agora iremos importar o que precisamos para criar as rotas do navigation:

```
import React, { Component } from 'react';  
import { NavigationContainer } from '@react-  
navigation/native'; import { createStackNavigator } from
```

Logo após os importes declare a seguinte constante, para criar o StackNavigator:

```
const Stack = createStackNavigator();
```

Agora criamos nossa class:

```
export default class StackNavigator extends Component {}
```

E então dentro dela vamos inserir a tag <NavigationContainer> dentro dessa uma <Stack.Navigator> e dentro do Stack.Navigator

colocaremos as telas da nossa rota, mas primeiro precisamos importar a tela, então importe com a seguinte linha:

```
import App from './App';
```

Adicione agora a tela dentro do Stack.Navigator usando o fluxo abaixo:

```
<Stack.Screen
  name="App"
  component={App}
  options={{
    headerShown: false
  }}
/
```

Agora criaremos outra tela com o nome ContactInfos.js dentro do nosso projeto, e iremos logo adiciona-la na nossa rota, para isso importamos ela primeiro:

```
import ContactInfos from './ContactInfos';
```

E então adicionamos a tela na nossa rota:

```
<Stack.Screen
  name="ContactInfos"
  component={ContactInfos}
  options={{
    title: '',
  }}
/
```


No final vai ficar assim:

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

import { NavigationContainer } from '@react-
navigation/native'; import { createStackNavigator } from
 '@react-navigation/stack'; import App from './App';
import ContactInfos from
 './ContactInfos'; const Stack =
createStackNavigator();

export default class StackNavigator extends Component {
  render() {
    return(
      <NavigationContainer>
        <Stack.Navigator>
          <Stack.Screen
            name="App"
            component={App}
            options={{
              headerShown: false
            }}
          />
          <Stack.Screen
            name="ContactInfos"
            component={ContactInfos}
            options={{
              title: '',
            }}
          />
        </Stack.Navigator>
      </NavigationContainer>
    );
  }
}
```

Nota: Como “App.js” é a primeira página da lista da rota, ela será a primeira a ser exibida, porém você pode adicionar um “initialRoute” dentro do NavigationContainer para definir a primeira tela da rota. Outro detalhe importante, o seu programa está como padrão para executar a “App.js”, então você entra em “node_modules>expo>AppEntry.js” e dentro de AppEntry você muda o import que está importando a tela App.js para importar a tela Navigation.js:

```
import App from '../..//Navigation';
```

Agora, iremos fazer a tela para exibir as informações do usuário. Na tela ContactInfos.js iremos fazer os seguintes imports:

```
import { StatusBar } from 'expo-status-bar'; import React, { Component } from 'react'; import { StyleSheet, Text, View, ScrollView } from 'react-
```

Declarar a class:

```
export default class App extends Component {}
```

Dentro dela iremos adicionar a seguinte função:

```
function(props) {  
  const navigation = useNavigation();  
  return <MyBackButton {...props} />  
}
```

Para podermos usar o Navigation.

dentro da render vamos declarar as seguintes constantes:

```
const { route } = this.props;  
const contact = route.params.contact;  
const phoneNumbers = route.params.phoneNumbers;
```

A route serve para resgatarmos os parâmetros que enviamos da pagina anterior para essa, e na constante contact armazenamos o contato que estava nesse parâmetro.

Agora declaramos o “return() {}”, e nele iremos exibir as informações do usuário:

```
<View style={container}>
  <StatusBar style={statusBarColor}/>
  <View style={styles.content}>
    <View style={contactNameView}>
      <Text
style={contactName}>{contact.name}</Text>
    </View>
    <Text style={title}> Números: </Text>
    <ScrollView
      style={styles.placesContai
ner} vertical
    >
      <View style={styles.place}>
        {
          phoneNumbers != null ?
          phoneNumbers.map(number =>
            <View style={numberView}
key={number.id}>
              <Text
                style={numberText
              }
            >
              {number.number}
            </Text>
          </View>
        }
      </View>
    </View>
  </View>
</View>
```

Usaremos o map pois caso a pessoa tenha mais de um número de telefone registrado no mesmo contato, e o map vai mostrar todos, lembrando que o map sempre leva uma key, e no caso usamos o id do número, veja também a presença de um if ternário, que checa se phoneNumbers é nulo, caso ele seja nulo, ele não exibirá nada, pois se tentar exibir dá erro, e caso não seja nulo ele mostrará o nosso map.

Aplicativo 4 - CRUD

Crie um novo projeto expo, crie um novo projeto no firebase e após criado no menu lateral que fica na esquerda clique em “Criação>Cloud Firestore”, na tela que aparecer clique em “Criar banco de dados” selecione “Iniciar no modo de teste” e clique em Avançar, selecione “southamerica east1” em Local do Cloud Firestore e então clique em Ativar. Agora no seu projeto crie as seguintes pastas “src>config” e dentro de config crie um arquivo chamado “firebase.js”. Na página do firebase clique em “Visão geral do projeto”, agora iremos conectar o firebase com nosso projeto, por estarmos usando expo, clique no símbolo com a descrição “web” para adicionar um aplicativo, de um nome ao app, deixe a checkbox desmarcada e clique em Registrar app, após feito isso o firebase vai gerar um código, copie da linha “var firebaseConfig = {” até a linha “firebase.initializeApp(firebaseConfig);” e cole no firebase.js, agora precisamos instalar a biblioteca do firebase, no terminal digite “npm install firebase”, volte à página do firebase e clique em Continuar no console e então clique em Cloud Firestore no menu lateral e na tela seguinte clique em Iniciar coleção, uma coleção é um conjunto de documentos que contém dados, por exemplo uma coleção usuários iria guardar um documento para cada usuário onde dentro dele existirá as informações de usuário, como forma de exemplo iremos criar uma coleção chamada produtos, para fazer um programa que serviria para armazenar produtos de uma loja ou algo do tipo, então nomeie o Código de coleção como produtos (você pode dar o nome que quiser e armazenar o que quiser, isto é apenas um exemplo), clique em Próxima, clique em Código automático para gerar um Código do documento, e agora precisamos definir os campos do documento, eu vou defini-los como “nome” do tipo string, “descricao” do tipo string e “valor” do tipo number, e vou preencher o valor de cada campo para inicializar o banco com algum dado já existente nele, você pode preencher os valores com o nome, descrição e valor que você quiser, após isso clique em Salvar, você pode então clicar em Adicionar documento para adicionar mais um documento, e o procedimento é mesmo do anterior.

No seu projeto, em firebase.js adicione as seguintes linhas no início para importar o firebase e o storage:

```
import * as firebase from 'firebase';  
import 'firebase/storage';
```

precisamos exportar o database, então no final adicione:

```
export const database = firebase.firestore();
```

Em App.js mude o function para:

```
export default class App extends Component {
```

e coloque o return dentro de uma “render() {}”.

Se preferir crie um arquivo chamado styles.js, e coloque os styles lá da seguinte forma:

```
import { StyleSheet } from 'react-native';

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

export default styles;
```

e em App.js importe os styles:

```
import styles from './styles.js';
```

ainda em App.js importe o database:

```
import { database } from './src/config/firebase';
```

defina o state:

```
state = {
  produtos: null,
  produtosFilter: null,
  nome: null,
  descricao: null,
  valor: null,
}
```

Crie um arquivo chamado “Navigation.js”, instale no terminal as bibliotecas “@react-navigation/native” e “@react-navigation/stack” usando o comando “npm install [nome da biblioteca]” (Obs. pode ser que faltem algumas bibliotecas das quais essas duas dependem, na hora de iniciar a emulação vai dar erro caso estejam faltando e o nome delas será indicado, instale-as), então em Navigation.js faça as seguintes importações:

```
import React, { Component } from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import App from './App';
```

defina a seguinte variável:

```
const Stack = createStackNavigator();
```

e então defina as rotas do navigation:

```
export default class StackNavigator extends Component {
  render() {
    return (
      <NavigationContainer>
        <Stack.Navigator>
          <Stack.Screen
            name="App"
            component={App}
            options={{
              title: '',
              headerStyle: {
                backgroundColor: '#2D363D',
              },
              headerTintColor: '#fff'
            }}
            initialParams={{
              page: 'inicial'
            }}
          />
        </Stack.Navigator>
      </NavigationContainer>
    );
  }
}
```

Entre em `node_modules>expo>AppEntry.js` e mude a linha 3 para:

```
import App from '../Navigation';
```

vamos usar a seguinte render para montar a aplicação:

```
render() {
  LogBox.ignoreLogs(['Setting a timer for a long ', ['Can\'t
perform a React state']]);
  const nome = this.state.nome;
  const descricao = this.state.descricao;
  const valor = this.state.valor;
  const { navigation } = this.props;
  const { route } = this.props;
  const page = route.params.page;

  const produto = route.params.produto;
  const action = route.params.action;

  switch(page) {
    case 'inicial':
      return (
        <View style={styles.container}>

          <View style={styles.buttonView}>
            <Pressable
              style={({ pressed }) => [
                {
                  backgroundColor: pressed
                    ? '#43515C'
                    : '#2D363D'
                },
                styles.button
              ]}
              onPress={() => {
                navigation.push('App', { page: 'criarAtualizar',
action: 'create' })
              }}
            >
              <AntDesign name="plus" size={30} color='#fff' />
            </Pressable>
          </View>

          <ScrollView>
```

```

<StatusBar style="auto" />
<View style={styles.titleView}>
  <Text style={styles.title}>Lista de Produtos</Text>
</View>
<TextInput
  style={styles.textInput}
  placeholder="Pesquisar"
  onChangeText={(text) => this.read(text)}
/>
<ScrollView style={styles.scrollView}>
  {
    this.state.produtosFilter == null
    ? null
    : this.state.produtosFilter.map(produto =>
      <View style={styles.produtoView}
        key={produto.id}>
        <Pressable
          style={({ pressed }) => [
            {
              backgroundColor: pressed
                ? '#43515C'
                : '#2D363D'
            },
            styles.produtos
          ]}
          onPress={() => {
            navigation.push('App', {
              page: 'produtoInfo',
              produto: produto
            })
          }}
        >
        <Text
          style={styles.produtoNome}>{produto.nome}</Text>
          <Text style={styles.produtoDescricao}
            numberOfLines={1}>{produto.descricao}</Text>
          <Text style={styles.produtoValor}>R$
            {produto.valor}</Text>
        </Pressable>
        <Ionicons onPress={() =>
          this.delete(produto.id)} name="trash" size={30} color='#db0000' style={{
          marginLeft: '7%' }}/>
      </View>
    )
  }
</ScrollView>

```



```

        )
    }
    </ScrollView>
</ScrollView>
</View>
);

case 'criarAtualizar':
return(
    <View style={styles.container}>
        <ScrollView>
            <View style={styles.titleView}>
                <Text style={styles.title}>
                    {
                        action == 'create'
                        ? 'Criar Produto'
                        : action == 'update'
                        ? 'Atualizar Produto'
                        : null
                    }
                </Text>
            </View>
            <TextInput style={styles.textInput}
placeholder="Nome" value={nome} onChangeText={({text) => this.setState({
nome: text })})/>
            <TextInput style={styles.textInput}
placeholder="Descrição" value={descricao} onChangeText={({text) =>
this.setState({ descricao: text })})/>
            <TextInput style={styles.textInput}
placeholder="Valor" value={valor} onChangeText={({text) =>
this.setState({ valor: text })})/>
            <Pressable
                onPress={() => {
                    if(action == 'create') {
                        this.create()
                    }else if(action == 'update') {
                        this.update(produto.id)
                    }
                }}
                style={({ pressed }) => [
                    {
                        backgroundColor: pressed
                            ? '#43515C'

```

```

        : '#2D363D'
      },
      styles.botao
    ]}
  >
  <Text style={styles.botaoText}>
    {
      action == 'create'
        ? 'Criar'
        : action == 'update'
        ? 'Atualizar'
        : null
    }
  </Text>
</Pressable>
</ScrollView>
</View>
);

case 'produtoInfo':
  return(
    <View style={styles.container}>
      <ScrollView>
        <View style={styles.titleView}>
          <Text style={styles.title}>{produto.nome}</Text>
        </View>
        <View style={styles.content}>
          <Text
style={styles.produtoText}>{produto.descricao}</Text>
          <Text
style={styles.produtoText}>R$
{produto.valor}</Text>
        </View>
        <Pressable
style={({ pressed }) => [
          {
            backgroundColor: pressed
              ? '#43515C'
              : '#2D363D'
          },
          styles.botao
        ]}
onPress={() => {
  navigation.goBack();

```

```

        navigation.push('App', {
            page: 'criarAtualizar',
            action: 'update',
            produto: produto
        })
    })
}
<
    <Text style={styles.botaoText}>Editar</Text>
</Pressable>
</ScrollView>
</View>
);
}
}

```

Com isso, todas as páginas que vamos utilizar estarão prontas, estilize a página da maneira que achar melhor, podendo usar como base os styles já definidos nos componentes.

Primeiro começaremos com o Read, crie uma função assíncrona chamada “read”:

```
read = async (text) => {}
```

dentro da função, definimos uma variável para armazenar o primeiro valor do nosso read e exibi-lo:

```
let filtrado;
```

E então, adicione o código:

```
database.collection('produtos').onSnapshot((query) => {
  const list = [];
  query.forEach((doc) => {
    list.push({ ...doc.data(), id: doc.id});
  })

  this.setState({ produtos: list })
  input = text;

  filtrado = list.filter(this.acharCorrespondencia);
  this.setState({produtosFilter: filtrado});
})
```

Toda vez que alguma mudança acontecer no banco de dados “database.collection” irá disparar, passando como parâmetro ‘produtos’ que é o nome do documento do nosso banco de dados, então ele define uma constante “list” com um valor de array, e então para cada item “query” que ele encontrar em produtos, ele irá empurrar o valor “data” que é um objeto produto, e o id desse objeto para dentro da const list. Logo após setamos o state “produtos” com o valor de list, captura o valor do texto digitado, define a variável filtrado como um filtro de list, e seta o state produtosFilter como filtrado.

Adicione o `componentDidMount` que vai definir se a página criar/atualizar, vai criar ou atualizar um produto e executar o `read` pela primeira vez:

```
componentDidMount() {
  const { route } = this.props;
  const produto = route.params.produto;
  this.read();

  const action = route.params.action;

  if(action === 'update') {
    this.setState({ nome: produto.nome, descricao:
produto.descricao, valor: produto.valor })
  }
}
```

adicione a seguinte function para que possa ser usado o navigation:

```
function(props) {
  const navigation = useNavigation();
  return <MyBackButton {...props} navigation={navigation} />
}
```

Faça os seguintes imports:

```
import { StatusBar } from 'expo-status-bar';
import React, { Component } from 'react';
import { Text, View, TextInput, Pressable, LogBox, ScrollView } from
'react-native';
import styles from './styles.js';
import { database } from './src/config/firebase';
import { Ionicons, AntDesign } from '@expo/vector-icons';
import { useNavigation } from '@react-navigation/native';
```

Para filtrar os produtos quando pesquisados usaremos regex criando a seguinte função:

```
acharCorrespondencia(produto) {
```

```

let termo = input;
var regex = new RegExp(termo, 'gi');

if(regex.test(produto.nome))
    return true;

if(regex.test(produto.descricao))
    return true;

return false;
}

```

Para ser possível a deleção de produtos adicionamos a seguinte função:

```

delete = async (id) => {
    database.collection('produtos').doc(id).delete()
}

```

E então para update usaremos a função:

```

update = async (id) => {
    const { navigation } = this.props;
    const nome = this.state.nome;
    const descricao = this.state.descricao;
    const valor = this.state.valor;
    database.collection('produtos').doc(id).update({
        nome,
        descricao,
        valor
    })
    navigation.goBack();
}

```

e para criação:

```

create = async () => {
    const nome = this.state.nome;
    const descricao = this.state.descricao;

```

```
const valor = this.state.valor;
const { navigation } = this.props;

database.collection('produtos').add({
  nome,
  descricao,
  valor
});

this.setState({ nome: null, descricao: null, valor: null })
navigation.goBack();
}
```

Cada uma delas puxa os dados do produto e utiliza-os no `database.collection('produtos')` que recebe um artigo que indica sua função e então passa as informações necessárias dentro dele.

Bibliografia

Cimpanu, Catalin. **“Facebook’s React Native Framework Gets Windows and Tizen Support”**. Softpedia News. Disponível em:

<<https://bit.ly/338YxhF>>. Acesso em 03/09/2020.

“Comparação IOS, Android e React Native”. Google Trends. Disponível em: <<https://bit.ly/3588s9P>>. Acesso em 03/09/2020.

Shoutem. **“A brief history of React Native”**. Medium.

Disponível em: <<https://bit.ly/3hW9Orr>>. Acesso em 03/09/2020.

“O que é React Native? O futuro do desenvolvimento híbrido”.

Codificar. Disponível em: <<https://bit.ly/2Drglvn>>. Acesso em 03/09/2020.

“Xamarin vs Ionic vs React Native vs NativeScript: desenvolvimento multiplataforma”. Codificar. Disponível em: <<https://bit.ly/2EXLw25>>.

Acesso em 03/09/2020.

Basrai, Murtaza. **“Top React Native Editors for Mobile Development in 2020”**. icycle Technologies. Disponível em: <<https://bit.ly/34jN9QB>>.

Acesso em 30/09/2020.

“Download Node.js”. Node.js. Disponível em: <<https://bit.ly/3n8dGZE>>.

Acesso em 26/09/2020.

“Expo Client”. App Store. Disponível em: <<https://apple.co/2G7nkev>>.

Acesso em 26/09/2020.

“Expo”. Google Play. Disponível em: <<https://bit.ly/3l8skyj>>. Acesso em 26/09/2020.

“DownloadVisualStudio Code”. Visual Studio Code.
Disponível em: <<https://bit.ly/3n9j7rm>>. Acesso em 26/09/2020.