USING MQTT WITH REACT NATIVE

Windows

DEPENDENCIES

- Node.JS
- Android Studio
- Android SDK

INSTALLING NODE

Open an Administrator Command Prompt (right click Command Prompt and select "Run as Administrator"), then run the following command:
- choco install -y nodejs-lts openjdk11

INSTALLING ANDROID STUDIO

Download and install Android Studio. While on Android Studio installation wizard, make sure the boxes next to all of the following items are checked:

- Android SDK
- Android SDK Platform
- Android Virtual Device
- If you are not already using Hyper-V: Performance (Intel ® HAXM) (See here for AMD or Hyper-V)

Then, click "Next" to install all of these components.

INSTALLING ANDROID SDK

To do that, open Android Studio, click on "More Actions" button and select "SDK Manager".
Select the "SDK Platforms" tab from within the SDK Manager, then check the box next to "Show Package Details" in the bottom right corner. Look for and expand the Android 12 (S) entry, then make sure the following items are checked:

- Android SDK Platform 31
- Intel x86 Atom_64 System Image or Google APIs Intel x86 Atom System Image

Next, select the "SDK Tools" tab and check the box next to "Show Package Details" here as well. Look for and expand the Android SDK Build-Tools entry, then make sure that 31.0.0 is selected.
Finally, click "Apply" to download and install the Android SDK and related build tools.

Windows

CONFIGURE THE ANDROID_HOME ENVIRONMENT VARIABLE

The React Native tools require some environment variables to be set up in order to build apps with native code.

1. Open the Windows Control Panel.
2. Click on User Accounts, then click User Accounts again
3. Click on Change my environment variables
4. Click on New... to create a new ANDROID_HOME user variable that points to the path to your Android SDK.

The SDK is installed, by default, at the following location: %LOCALAPPDATA%\Android\Sdk

You can find the actual location of the SDK in the Android Studio "Settings" dialog, under Appearance & Behavior → System Settings → Android SDK.
Open a new Command Prompt window to ensure the new environment variable is loaded before proceeding to the next step.

1. Open powershell
2. Copy and paste Get-ChildItem -Path Env:\ into powershell
3. Verify ANDROID_HOME has been added.

Add platform-tools to Path:

1. Open the **Windows Control Panel.**
2. Click on **User Accounts,** then click **User Accounts** again
3. Click on **Change my environment variables**
4. Select the **Path** variable.
5. Click **Edit.**
6. Click **New** and add the path to platform-tools to the list.

The default location for this folder is:
%LOCALAPPDATA%\Android\Sdk\platform-tools

CREATING A NEW APPLICATION

```
npx react-native init client-mqtt
```

https://reactnative.dev/docs/environment-setup

Linux

DEPENDENCIES

- Node.JS
- Java Development Kit
- Android Studio
- Android SDK
- Watchman

INSTALLING NODE

Follow the installation instructions for your Linux distribution to install Node 14 or newer.

- https://nodejs.org/en/download/package-manager/

INSTALLING JAVA DEVELOPMENT KIT

React Native currently recommends version 11 of the Java SE Development Kit (JDK). You may encounter problems using higher JDK versions. You may download and install OpenJDK from AdoptOpenJDK or your system packager.

- http://openjdk.java.net/
- https://adoptopenjdk.net/

INSTALLING ANDROID STUDIO

Download and install Android Studio. While on Android Studio installation wizard, make sure the boxes next to all of the following items are checked:

- Android SDK
- Android SDK Platform
- Android Virtual Device

Then, click "Next" to install all of these components.

INSTALLING ANDROID SDK

Android Studio installs the latest Android SDK by default. Building a React Native app with native code, however, requires the Android 12 (S) SDK in particular. Additional Android SDKs can be installed through the SDK Manager in Android Studio.
To do that, open Android Studio, click on "Configure" button and select "SDK Manager".

Linux

INSTALLING ANDROID SDK

Select the "SDK Platforms" tab from within the SDK Manager, then check the box next to "Show Package Details" in the bottom right corner. Look for and expand the Android 12 (S) entry, then make sure the following items are checked:

- Android SDK Platform 31
- Intel x86 Atom_64 System Image or Google APIs Intel x86 Atom System Image

Next, select the "SDK Tools" tab and check the box next to "Show Package Details" here as well. Look for and expand the "Android SDK Build-Tools" entry, then make sure that 31.0.0 is selected.
Finally, click "Apply" to download and install the Android SDK and related build tools.

Configure the ANDROID_SDK_ROOT environment variable:

The React Native tools require some environment variables to be set up in order to build apps with native code.
Add the following lines to your $HOME/.bash_profile or $HOME/.bashrc (if you are using zsh then ~/.zprofile or ~/.zshrc) config file:

```
export ANDROID_SDK_ROOT=$HOME/Library/Android/Sdk
export PATH=$PATH:$ANDROID_SDK_ROOT/emulator
export PATH=$PATH:$ANDROID_SDK_ROOT/platform-tools
```

Type source $HOME/.bash_profile for bash or source $HOME/.zprofile to load the config into your current shell. Verify that ANDROID_SDK_ROOT has been set by running echo $ANDROID_SDK_ROOT and the appropriate directories have been added to your path by running echo $PATH.

WATCHMAN

Follow the Watchman installation guide to compile and install Watchman from source.

- https://facebook.github.io/watchman/docs/install/#buildinstall

CREATING A NEW APPLICATION

```
npx react-native init client-mqtt
```

macOS

DEPENDENCIES

- Node & Watchman
- Java Development Kit
- Android Studio
- Android SDK

INSTALLING NODE & WATCHMAN

```
brew install node
brew install watchman
```

INSTALLING JAVA DEVELOPMENT KIT

```
brew tap homebrew/cask-versions
brew install --cask zulu11
```

INSTALLING ANDROID STUDIO

Download and install Android Studio. While on Android Studio installation wizard, make sure the boxes next to all of the following items are checked:

- Android SDK
- Android SDK Platform
- Android Virtual Device

Then, click "Next" to install all of these components.

INSTALLING ANDROID SDK

Android Studio installs the latest Android SDK by default. Building a React Native app with native code, however, requires the Android 12 (S) SDK in particular. Additional Android SDKs can be installed through the SDK Manager in Android Studio.
To do that, open Android Studio, click on "Configure" button and select "SDK Manager"
Select the "SDK Platforms" tab from within the SDK Manager, then check the box next to "Show Package Details" in the bottom right corner. Look for and expand the Android 12 (S) entry, then make sure the following items are checked:
.

macOS

INSTALLING ANDROID SDK

- Android SDK Platform 31
- Intel x86 Atom_64 System Image or Google APIs Intel x86 Atom System Image or (for Apple M1 Silicon) Google APIs ARM 64 v8a System Image

Next, select the "SDK Tools" tab and check the box next to "Show Package Details" here as well. Look for and expand the "Android SDK Build-Tools" entry, then make sure that 31.0.0 is selected.
Finally, click "Apply" to download and install the Android SDK and related build tools.

Configure the ANDROID_SDK_ROOT environment variable:
The React Native tools require some environment variables to be set up in order to build apps with native code.
Add the following lines to your $HOME/.bash_profile or $HOME/.bashrc (if you are using zsh then ~/.zprofile or ~/.zshrc) config file:

```
export ANDROID_SDK_ROOT=$HOME/Library/Android/sdk
export PATH=$PATH:$ANDROID_SDK_ROOT/emulator
export PATH=$PATH:$ANDROID_SDK_ROOT/platform-tools
```

Type source $HOME/.bash_profile for bash or source $HOME/.zprofile to load the config into your current shell. Verify that ANDROID_SDK_ROOT has been set by running echo $ANDROID_SDK_ROOT and the appropriate directories have been added to your path by running echo $PATH.

CREATING A NEW APPLICATION

```
npx react-native init client-mqtt
```

https://reactnative.dev/docs/environment-setup

Project Setup

After creating the project, you can navigate to **client-mqtt** folder and run:

```
npx react-native start
```

In another terminal, run:

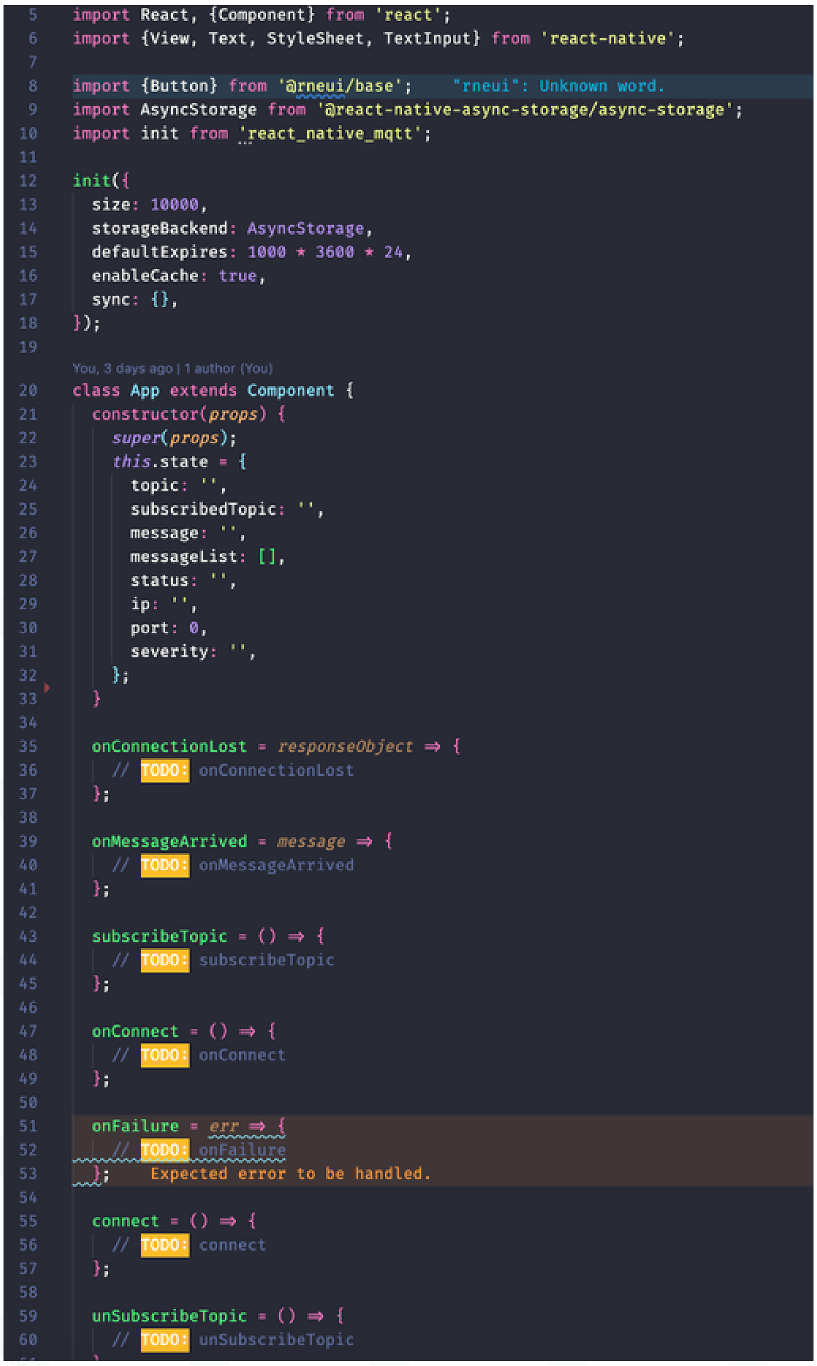
```
npx react-native run-android
```

Creating project

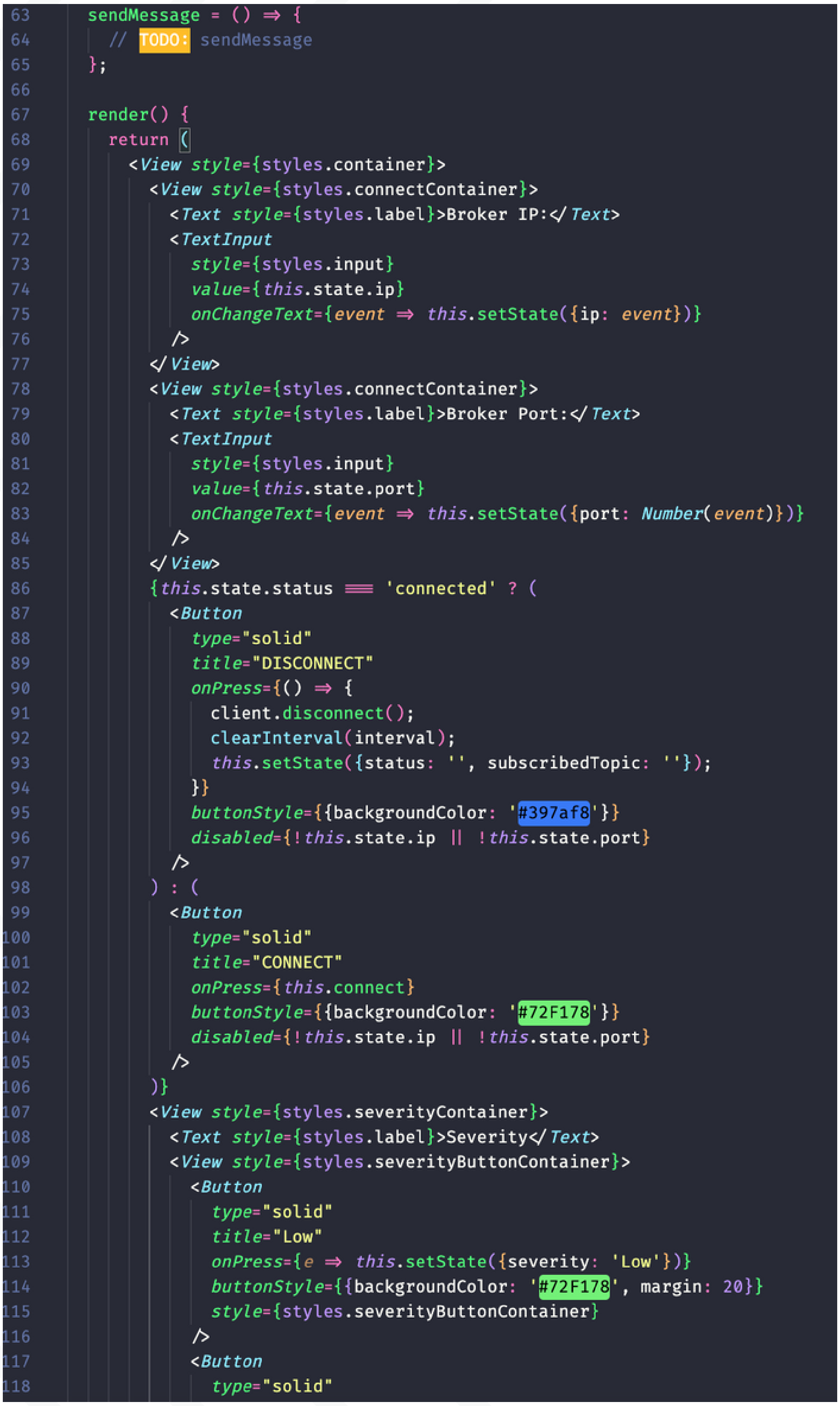
Our application is going to have only one screens:

1- Config form to IP address, broker port and severity level.

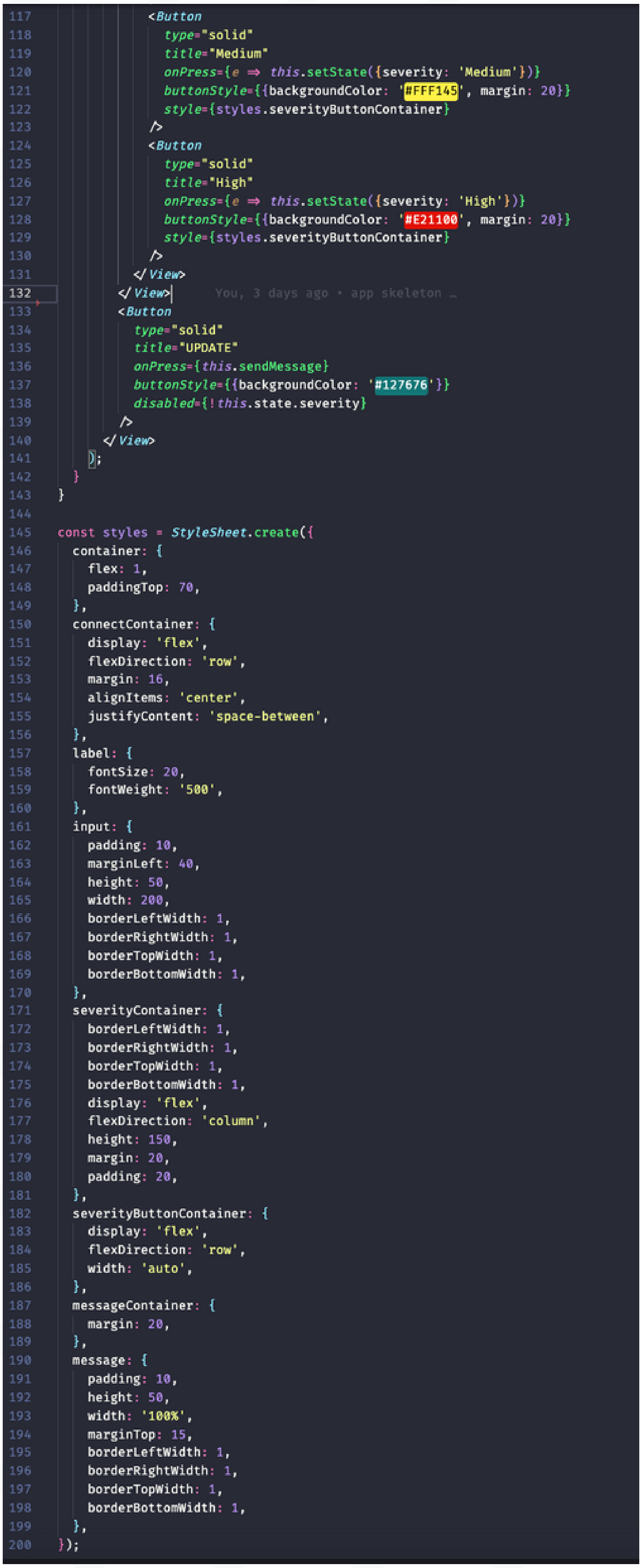
Now we can updatee the file called **App.js** with the following code:



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

import {Button} from '@rneui/base';    "rneui": Unknown word.
import AsyncStorage from '@react-native-async-storage/async-storage';
import init from 'react_native_mqtt';

init({
  size: 10000,
  storageBackend: AsyncStorage,
  defaultExpires: 1000 * 3600 * 24,
  enableCache: true,
  sync: {},
});

You, 3 days ago | 1 author (You)
class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      topic: '',
      subscribedTopic: '',
      message: '',
      messageList: [],
      status: '',
      ip: '',
      port: 0,
      severity: '',
    };
  }

  onConnectionLost = responseObject => {
    // TODO: onConnectionLost
  };

  onMessageArrived = message => {
    // TODO: onMessageArrived
  };

  subscribeTopic = () => {
    // TODO: subscribeTopic
  };

  onConnect = () => {
    // TODO: onConnect
  };

  onFailure = err => {
    // TODO: onFailure
  };    Expected error to be handled.

  connect = () => {
    // TODO: connect
  };

  unSubscribeTopic = () => {
    // TODO: unSubscribeTopic
```



```
  sendMessage = () => {
    // TODO: sendMessage
  };

  render() {
    return (
      <View style={styles.container}>
        <View style={styles.connectContainer}>
          <Text style={styles.label}>Broker IP:</Text>
          <TextInput
            style={styles.input}
            value={this.state.ip}
            onChangeText={event => this.setState({ip: event})}
          />
        </View>
        <View style={styles.connectContainer}>
          <Text style={styles.label}>Broker Port:</Text>
          <TextInput
            style={styles.input}
            value={this.state.port}
            onChangeText={event => this.setState({port: Number(event)})}
          />
        </View>
        {this.state.status === 'connected' ? (
          <Button
            type="solid"
            title="DISCONNECT"
            onPress={() => {
              client.disconnect();
              clearInterval(interval);
              this.setState({status: '', subscribedTopic: ''});
            }}
            buttonStyle={{backgroundColor: '#397af8'}}
            disabled={!this.state.ip || !this.state.port}
          />
        ) : (
          <Button
            type="solid"
            title="CONNECT"
            onPress={this.connect}
            buttonStyle={{backgroundColor: '#72F178'}}
            disabled={!this.state.ip || !this.state.port}
          />
        )}
        <View style={styles.severityContainer}>
          <Text style={styles.label}>Severity</Text>
          <View style={styles.severityButtonContainer}>
            <Button
              type="solid"
              title="Low"
              onPress={e => this.setState({severity: 'Low'})}
              buttonStyle={{backgroundColor: '#72F178', margin: 20}}
              style={styles.severityButtonContainer}
            />
            <Button
              type="solid"
```

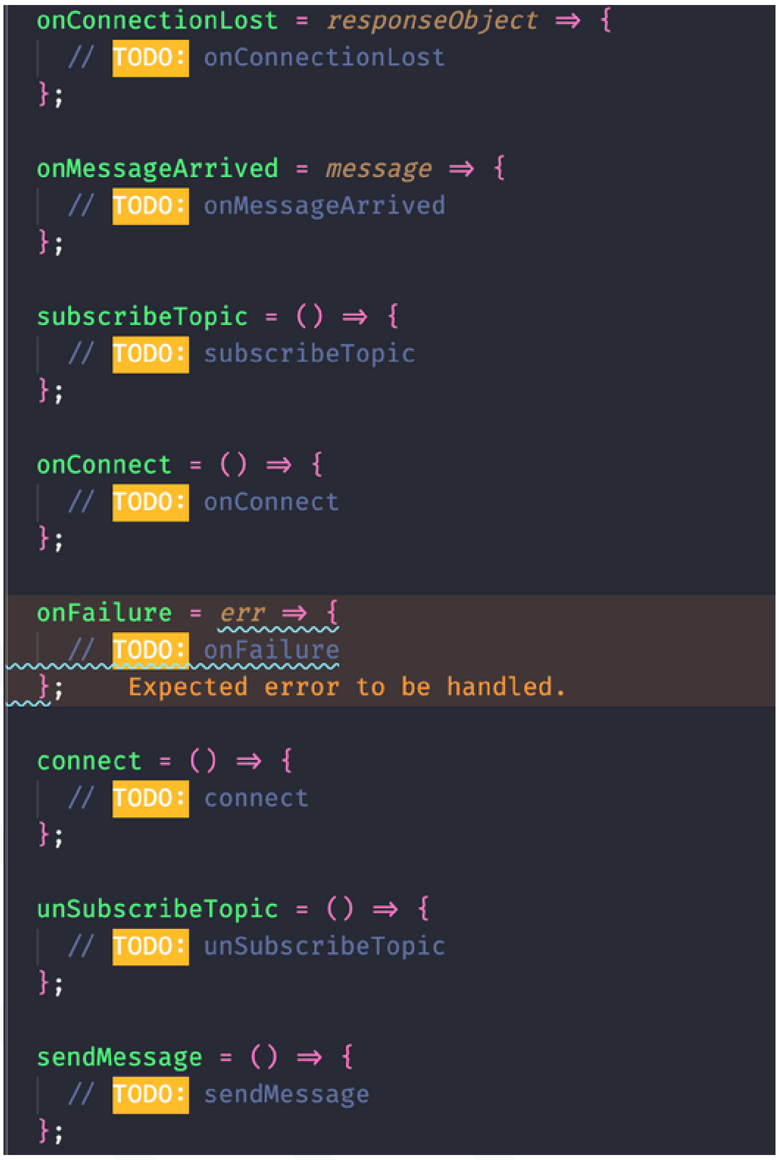


```
            <Button
              type="solid"
              title="Medium"
              onPress={e => this.setState({severity: 'Medium'})}
              buttonStyle={{backgroundColor: '#FFF145', margin: 20}}
              style={styles.severityButtonContainer}
            />
            <Button
              type="solid"
              title="High"
              onPress={e => this.setState({severity: 'High'})}
              buttonStyle={{backgroundColor: '#E21100', margin: 20}}
              style={styles.severityButtonContainer}
            />
          </View>
        </View>        You, 3 days ago • app skeleton …
        <Button
          type="solid"
          title="UPDATE"
          onPress={this.sendMessage}
          buttonStyle={{backgroundColor: '#127676'}}
          disabled={!this.state.severity}
        />
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: 70,
  },
  connectContainer: {
    display: 'flex',
    flexDirection: 'row',
    margin: 16,
    alignItems: 'center',
    justifyContent: 'space-between',
  },
  label: {
    fontSize: 20,
    fontWeight: '500',
  },
  input: {
    padding: 10,
    marginLeft: 40,
    height: 50,
    width: 200,
    borderLeftWidth: 1,
    borderRightWidth: 1,
    borderTopWidth: 1,
    borderBottomWidth: 1,
  },
  severityContainer: {
    borderLeftWidth: 1,
    borderRightWidth: 1,
    borderTopWidth: 1,
    borderBottomWidth: 1,
    display: 'flex',
    flexDirection: 'column',
    height: 150,
    margin: 20,
    padding: 20,
  },
  severityButtonContainer: {
    display: 'flex',
    flexDirection: 'row',
    width: 'auto',
  },
  messageContainer: {
    margin: 20,
  },
  message: {
    padding: 10,
    height: 50,
    width: '100%',
    marginTop: 15,
    borderLeftWidth: 1,
    borderRightWidth: 1,
    borderTopWidth: 1,
    borderBottomWidth: 1,
  },
});
```

On lines 23 through 31, we use the defined states that we are going to use to store our data.
For example, if we type **43.174.34.226** on the IP address input, the value of our state **ip** is going to be changed to **43.174.34.226.**

On lines 145 through 200, a **styles** variable that is the style sheet of our components. It is where we are going to style our text input and buttons.

In this file we have 8 methods to test our MQTT connection: **onConnectionLost**, **onMessageArrived**, **subscribeTopic**, **onConnect**, **onFailure**, **connect**, **unSubscribeTopic** and **sendMessage**.



```
onConnectionLost = responseObject ⇒ {
  // TODO: onConnectionLost
};

onMessageArrived = message ⇒ {
  // TODO: onMessageArrived
};

subscribeTopic = () ⇒ {
  // TODO: subscribeTopic
};

onConnect = () ⇒ {
  // TODO: onConnect
};

onFailure = err ⇒ {
  // TODO: onFailure
};    Expected error to be handled.

connect = () ⇒ {
  // TODO: connect
};

unSubscribeTopic = () ⇒ {
  // TODO: unSubscribeTopic
};

sendMessage = () ⇒ {
  // TODO: sendMessage
};
```

Running project

Now that we have our MQTT client working and our screen created, we can run our project by:

npm install

cd ios && pod install (in case you want to run with iOS)

npx react-native start (start metro bundler)

npx react-native run-ios (run with iOS)

npx react-native run-android (run with android)