Customize the App

Congratulations on creating your first You.i RN app! Now, let's open up the app and make some changes.

Folder Structure

As you saw in the first tutorial, `youi-tv init` creates scaffolding for your project. The initial folder structure looks like this:

```
├── README.youi.md          # The getting started instructions you just completed
├── __tests__               # Unit and application tests folder
    └ ...
├── client                  # Files for working with Cloud Solution for Roku
    └ ...
├── index.youi.js           # Main entry point for a You.i React Native project (file must have th
├── node_modules
    └ ...
├── package.json
├── rn-cli.config.js        # Configure the Metro bundler to exclude You.i build files
├── yarn.lock
└── youi                    # You.i C++ files associated with your app
    ├── AE
    |   └── assets          # Unlike Facebook React Native apps, assets are added to this folder
    |       └── drawable
    |           └── default
    |               └── youi_logo_red.png
    ├── CMakeLists.txt
|   ├── Project.cmake
|   ├── SourceList.cmake
|   ├── build.rb            # Invokes CMake to build the application binary
|   ├── generate.rb         # Invokes CMake to generate C++ scaffolding and build files
|   ├── reinstall.rb
|   └── src                 # Project C++ source files
|       ├── App.cpp
|       ├── App.h
|       └── AppFactory.cpp
```

Update Assets and Text

Like many other applications, your You.i React Native (RN) app has an asset pipeline. Let's replace the You.i TV logo image and text in the sample with your choice of content.

Pro Tip: Due to the number of platforms supported by You.i Engine One, not all React Native props can be used in your You.i RN app. Check our complete documentation for details. For example, props for the `Image` component can be found here.

1. Select a PNG or JPG image, like your company logo, to use in your application. For best results, pick an image with 4:3 or 1:1 aspect ratio.
2. Copy your image into `<AppName>/youi/AE/assets/drawable/default`.
3. Open `<AppName>/index.youi.js` in your favorite editor and replace the You.i TV logo image file name with yours.

```
<View style={styles.row}>
  <Image
    style={styles.image}
```

```
      source={{ uri: "res://drawable/default/youi_logo_red.png" }}
    />
  </View>
```

4. When you add new assets, you need to rebuild your project to copy them to the right location in your target platform(s).

Mac Linux Windows

```
youi-tv build -p osx
```

5. If you still have the Metro bundler (`yarn` server) running from the first part of the tutorial, you can launch your app again and the changes are picked up. Otherwise, `yarn start` and launch your app to see your updated image.

   From the `MyApp` folder, start your app.

Mac Linux Windows

```
./youi/build/osx/Debug/MyApp
```

6. As with other React Native apps, you can modify most of it while both the application and Metro bundler (Yarn server) are running. Modify your app's text and watch the application update when you save the file.

Add a React Native Component

Changing the text and image was a trivial exercise, but one that shows you the similarities between Facebook React Native and You.i React Native.

Let's take a quick moment to do another familiar task: add a React Native component to your app.

1. Open `MyApp/index.youi.js` in your editor.
2. Import the `Button` component and add a simple button press handler `onMyButtonPress` above the render function.

> **Pro Tip:** You.i Engine One customizes some common RN components, including `Image` and `Button`, to make them work on other target platforms. These customized components are automatically imported when you create apps with our toolchain.

```
import React, { Component } from "react";
import { AppRegistry, Button, Image, StyleSheet, Text, View } from "react-native";
import { FormFactor } from "@youi/react-native-youi";

export default class YiReactApp extends Component {

  onMyButtonPress = () => {
    console.log("***** You keep pressing my buttons. Stop it! *****");
  };

  render() {
  ...
```

> **Pro Tip:** Use `console.log` to help with debugging. See the output in the terminal window where you started your app.

3. Add a new `View` below the View `Image` is contained within. Give it the style `buttonContainer` (we'll create that in a moment). In this view, add the button with a `title` and an `onPress` handler.

```
  render() {
  ...
  <View style={styles.headerContainer}>
    <View style={styles.imageContainer}
    ...
```

```
      </View>
      <View style={styles.buttonContainer}>
        <Button
          onPress={this.onMyButtonPress}
          title="My button"
        />
      </View>
    </View>
    ...
}
```

4. Like Facebook RN apps, You.i Engine One apps can be made accessible. When VoiceOver is enabled on a device, you can enable screen reading by adding accessibility props to your app.

   Let's add accessibility props for your new component.

```
    <View style={styles.buttonContainer}
      focusable={true}
      accessible={true}
      accessibilityLabel="My button"
      accessibilityHint="Button in your first app"
      accessibilityRole="button"
    >
      <Button
        onPress={this.onMyButtonPress}
        title="My button"
        />
    </View>
```

   Note: Currently, screen reader functionality is fully supported for tvOS and iOS apps. To learn more, see Accessibility in You.i Engine One React Native.

5. Add the `buttonContainer` style to the Stylesheet defined at the bottom of `index.youi.js`. You might want to put it after `image` to keep things in the same order they're used.

```
...
},
buttonContainer: {
  flex: 1,
  justifyContent: "center",
  alignItems: "center"
},
bodyContainer: {
  ...
```

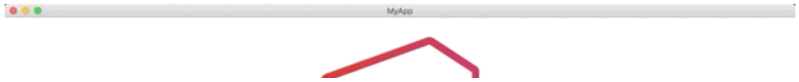
Run your application again to see the button you just added. There's not much to the button yet, but try clicking it. Switch to the terminal window where you launched the app and search for your button handler log message in the console output.

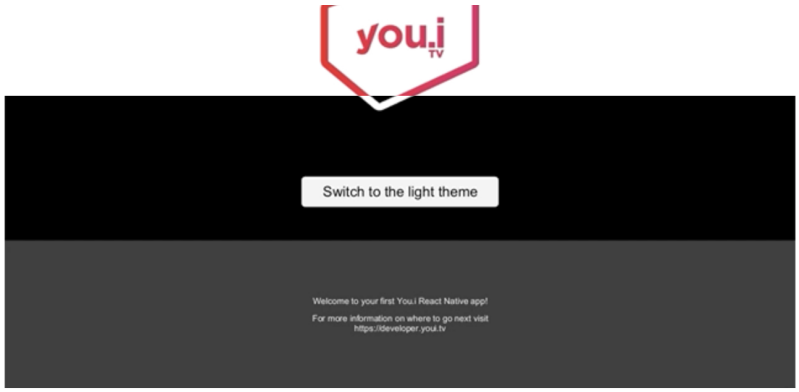
Add a Custom Component

Next, let's add a bit of fun to the application by adding a light and dark mode. To do this, we'll make use of a new custom `Button` component and a couple more styles.

Note: Before getting started, you should restore the You.i TV logo or have a light and dark version of the image you've selected. As a reminder, the file name is `youi_logo_red.png`.

Here's a quick GIF demonstrating what we're going to build.





1. Add a state variable to your component that tracks the current theme state: light or dark. Also, change the `onMyButtonPress` handler to switch the state each time the button is pressed and print a more useful log message.

```
...
export default class YiReactApp extends Component {

state = {
  lightTheme: true
};

onMyButtonPress = () => {
  this.setState({
    lightTheme: !this.state.lightTheme
  });
  console.log('***** Changing theme to: ' + (this.state.lightTheme ? 'dark' : 'light' ) + ' *
};
```

2. In `render`, destructure `lightTheme` from the state.

```
render() {
  const { lightTheme } = this.state;
```

And update button's `title` to be more informative.

```
   return (
   ...
     <Button
       onPress={this.onMyButtonPress}
       title={`Switch to the ${lightTheme ? "dark" : "light"} theme`}
     />
   ...
}
```

Switch back to your app and click the button a few times. With each click, the button text changes.

3. Copy and paste the following custom `Button` component into a new file called `button.js` in the same folder as `index.youi.js`.

```
import React from "react";
import { StyleSheet, Text, TouchableOpacity, View } from "react-native";
import { FocusManager, FormFactor } from "@youi/react-native-youi";

export const Button = ({
  buttonStyle,
  defaultFocus,
  onPress,
  textStyle,
  title
}) => {
```

4. Copy and paste the following style definitions into a new file called `styles.js`.

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

export const darkStyles = StyleSheet.create({
  header: {
    backgroundColor: "#000000"
  },
  button: {
    backgroundColor: "#f1f1f1"
  },
  buttonText: {
    color: "#000000"
```

5. Modify `index.youi.js` to import the custom `Button` component and style sheets you just created in `styles.js`. (Replace the previous `Button` import with this one.)

```
import React, { Component } from "react";
import { AppRegistry, Image, StyleSheet, Text, View } from "react-native";
import { FormFactor } from "@youi/react-native-youi";
import { Button } from "./button.js";
import { lightStyles, darkStyles } from './styles.js';
...
```

6. Update the `Button` component in your `View`.
   Use the custom component's `defaultFocus` prop to give this button focus when the app starts.

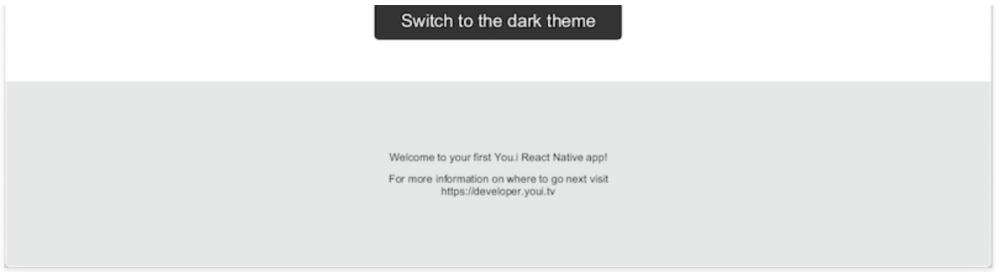
```
render()
...
       <Button
         ...
         defaultFocus={true}
       />
...
```

Add two new props, `textStyle` and `buttonStyle`, that change the button's styling based on the currently selected theme. The full new button component looks like this:

```
render()
...
       <Button
         onPress={this.onMyButtonPress}
         title={`Switch to the ${lightTheme ? "dark" : "light"} theme`}
         textStyle={lightTheme ? lightStyles.buttonText : darkStyles.buttonText}
         buttonStyle={lightTheme ? lightStyles.button : darkStyles.button}
         defaultFocus={true}
       />
```

Pro Tip: If you're still running your app, you may need to press Ctrl-R to refresh the screen. This clears any previous errors (from our interim changes) and reloads the scene.

At this point, your application should look like this. The button is there, but it doesn't do much other than change its text and style when you click it. Next, we'll add in the rest of the dark style.



Pro Tip: You may see stats at the top and bottom of your window (such as frame rate). You can hide these from the Dev Panel. Triple-click in any corner of the window to show the Dev Panel. Click the `Toggle HUD` button to turn off the heads up display (HUD). When you're done, click anywhere outside the Dev Panel to close it.

7. Add a new function to the bottom of `index.youi.js` that provides the name of the stylesheet to use (lightStyles or darkStyles).

```
...
});

function getTheme(lightTheme) {
  return lightTheme ? lightStyles : darkStyles;
}

AppRegistry.registerComponent("YiReactApp", () => YiReactApp);
```

8. Make use of this new function in your main `render()` by tracking it in `themeStyles`.

```
render() {

  const { lightTheme } = this.state;
  const themeStyles = getTheme(lightTheme);

  return (
...
```

9. Update styles for `headerContainer`, `bodyContainer`, `headlineText`, and `bodyText` to use a combination of the original style and `themeStyles.<x>` as shown below.

Note: Styles are applied left to right. We use this to our advantage to define new colors in `lightStyles` and `darkStyles`. These theme colors override colors from `styles` but the rest of the component styling is common.

```
   return (
      <View style={[styles.mainContainer]}>
        <View style={[styles.headerContainer, themeStyles.header]}>
...
      <View style={[styles.bodyContainer, themeStyles.body]} focusable={true} accessible={tru
        <Text
          style={[styles.headlineText, themeStyles.headlineText]}
          accessibilityLabel="Welcome to your first You I React Native app"
        >
          Welcome to your first You.i React Native app!
        </Text>
        <Text
          style={[styles.bodyText, themeStyles.bodyText]}
        >
          For more information on where to go next visit
        </Text>
        <Text
          style={[styles.bodyText, themeStyles.bodyText]}
          accessibilityLabel="https://developer dot you i dot tv"
        >
```

```
            https://developer.youi.tv
        </Text>
      </View>
...
```

Pro Tip: `accessibilityLabel` suppresses the text of the node itself and enables you to control what's read aloud. You'll notice we're using this technique to have the screen reader read the above website address correctly.

10. Lastly, right-click and download this You.i TV dark mode logo, save it to `<AppName>/youi/AE/assets/drawable/default`, and update `index.youi.js` to change the picture dynamically.

    Add two new variables in `render` for the image files, and update the `Image` component's source to choose the image based on the current theme.

```
render() {

   const { lightTheme } = this.state;
   const themeStyles = getTheme(lightTheme);
   const lightImage = "youi_logo_red.png";
   const darkImage = "youi_logo_white.png";

...
   <Image
     style={styles.image}
     source={{ uri: `res://drawable/default/${lightTheme ? lightImage : darkImage}` }}
   />
...
```

    Don't forget to rebuild your application again to get the new image copied into your target folder. Then run your app again to see the full dark mode toggle in action!

Mac Linux Windows

```
youi-tv build -p osx
```

Congratulations! Now you have a working app that uses basic React Native components and a custom component. Sometimes, custom components won't do the trick and you'll need to write a native module.

Follow along with the next step in this tutorial as we write a native module for your app.

Updated: February 18, 2020