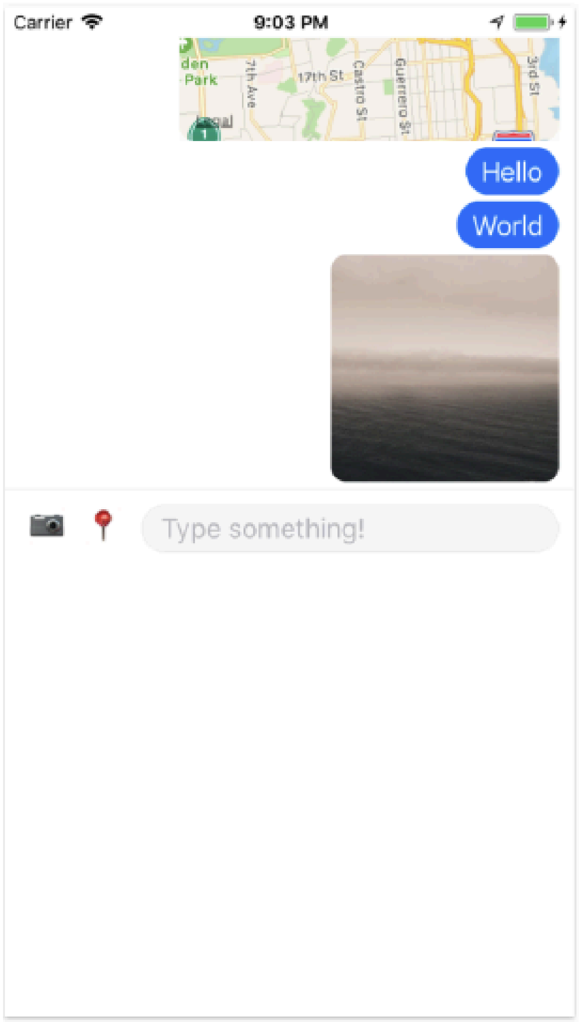
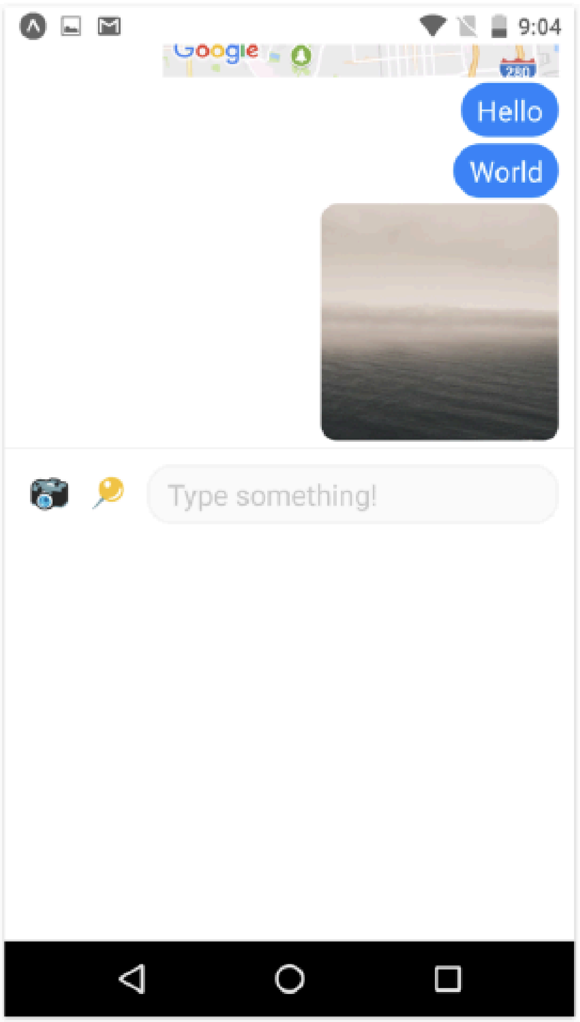
Activity No. 8 Toolbar	
Course Code: CPE 026	**Program:** Computer Engineering
Course Title: Emerging Technologies 3	**Date Performed:** November 2, 2024
Section: CPE41S8	**Date Submitted:** November 3, 2024
Name: Joshua L. Alferos Christian Ed B. Efa	**Instructor:** Engr. Roman RIchard
1. Objective(s)	
This activity aims to introduce students to the use of Toolbar for application development in iOS and Android using React Native.	
2. Intended Learning Outcomes (ILOs)	
After this module, the student should be able to: ● Demonstrate the creation of a toolbar and troubleshoot related errors.	
3. Discussion	
The toolbar will sit above the keyboard and contain an input field for typing messages, along with buttons for switching to an image picker and sending a location.	

iOS

Android





4. Materials and Equipment

To properly perform this activity, the student must have:

- Node LTS
- VS Code
- React Native

5. Procedure

Building the Toolbar

The toolbar is similar to the CommentInput from the Core Components chapter: it maintains the state of a TextInput field internally and uses an onSubmit function prop to tell the parent when the message is ready to send. We'll need a little more control over the focus state of the

TextInput this time, so we'll use an isFocused prop to control the focus state and an onChangeFocus function prop to tell the parent when the state should change. Let's create a new file Toolbar.js in the components directory, and render the top level View which will contain all the elements in the toolbar. We'll add propTypes for the focus state and the various functions which the parent component can pass in:

messaging/components/Toolbar.js
```
import {
  StyleSheet,
  Text,
  TextInput,
  TouchableOpacity,
  View,
} from "react-native";
import PropTypes from "prop-types";
import React from "react";

export default class Toolbar extends React.Component {
  static propTypes = {
    isFocused: PropTypes.bool.isRequired,
    onChangeFocus: PropTypes.func,
    onSubmit: PropTypes.func,
    onPressCamera: PropTypes.func,
    onPressLocation: PropTypes.func,
  };

  static defaultProps = {
    onChangeFocus: () => {},
    onSubmit: () => {},
    onPressCamera: () => {},
    onPressLocation: () => {},
  };

  render() {
    return <View style={styles.toolbar}> {/* ... */} </View>;
  }
}

const styles = StyleSheet.create({
  toolbar: {
    flexDirection: "row",
    alignItems: "center",
    paddingVertical: 10,
    paddingHorizontal: 10,
    paddingLeft: 16,
    backgroundColor: "white",
  },
  // ...
});
```

Let’s add a camera button and a location button and use them to call the onPressCamera and onPressLocation props.

```
// ...
const ToolbarButton = ({ title, onPress }) => (
  <TouchableOpacity onPress={onPress}>
    <Text style={styles.button}>{title}</Text>
  </TouchableOpacity>
);

ToolbarButton.propTypes = {
  title: PropTypes.string.isRequired,
  onPress: PropTypes.func.isRequired,
};

export default class Toolbar extends React.Component {
  // ...
  render() {
    const { onPressCamera, onPressLocation } = this.props;
    return (
      <View style={styles.toolbar}>
        {/* Use emojis for icons instead! */}
        <ToolbarButton title={'C'} onPress={onPressCamera} />
        <ToolbarButton title={'L'} onPress={onPressLocation} />
        {/* ... */}
      </View>
    );
  }
}

const styles = StyleSheet.create({
  // ...
  button: {
    top: -2,
    marginRight: 12,
    fontSize: 20,
    color: 'grey',
  },
  // ...
});
```

We can use emojis here for button icons! They require some positioning tweaks in styles.button, but look decent on both platforms.
Show your modifications by adding buttons to the toolbar in the indicated section of the code.

We define a ToolbarButton component at the top of the file which we can use in the toolbar. This component is fairly small and styled specifically for use in the toolbar, so we leave it in the same file as Toolbar. In terms of coding style, it's common to define small utility components in the same file that they’re used, and move them into separate files later if we want to reuse them. Now let’s render a TextInput. As a reminder from the Core Components chapter: when we style a TextInput, it’s often easier to put styles like border and padding on a wrapper View. Otherwise we tend to run into slight rendering inconsistencies, e.g. borders not rendering.

```
// ...
export default class Toolbar extends React.Component {
  // ...
  state = {
    text: "",
  };

  handleChangeText = (text) => {
    this.setState({ text });
  };

  handleSubmitEditing = () => {
    const { onSubmit } = this.props;
    const { text } = this.state;
    if (!text) return;
    onSubmit(text);
    this.setState({ text: "" });
  };

  render() {
    const { onPressCamera, onPressLocation } = this.props;
    const { text } = this.state;
    return (
      <View style={styles.toolbar}>
        {/* Use emojis for icons instead! */}
        <ToolbarButton title={"C"} onPress={onPressCamera} />
        <ToolbarButton title={"L"} onPress={onPressLocation} />
        <View style={styles.inputContainer}>
          <TextInput
            style={styles.input}
            underlineColorAndroid={"transparent"}
            placeholder={"Type something!"}
            blurOnSubmit={false}
            value={text}
            onChangeText={this.handleChangeText}
            onSubmitEditing={this.handleSubmitEditing}
            // ...
          />
        </View>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  // ...
  inputContainer: {
    flex: 1,
    flexDirection: "row",
    borderWidth: 1,
    borderColor: "rgba(0,0,0,0.04)",
    borderRadius: 16,
    paddingVertical: 4,
    paddingHorizontal: 12,
    backgroundColor: "rgba(0,0,0,0.02)",
  },
  input: {
    flex: 1,
    fontSize: 18,
  },
});
```

When the user presses the return key on the keyboard, we call onSubmit with this value and then reset state.text. Our messaging app doesn't allow sending multiline messages (we're using the return key to submit,so there's no way to insert a newline).

Explain the purpose of using blurOnSubmit={false} in this code.

We'll need to focus and blur the input field at specific times. We need to do this because when the user presses the camera icon, we want to dismiss the keyboard. The built-in React Native APIs for this are imperative: we have to call .focus() and .blur() on an instance of TextInput. We'll contain this complexity in this component, so that App can use an isFocused prop to declare the focus state. In order to do this, we'll use a ref prop.

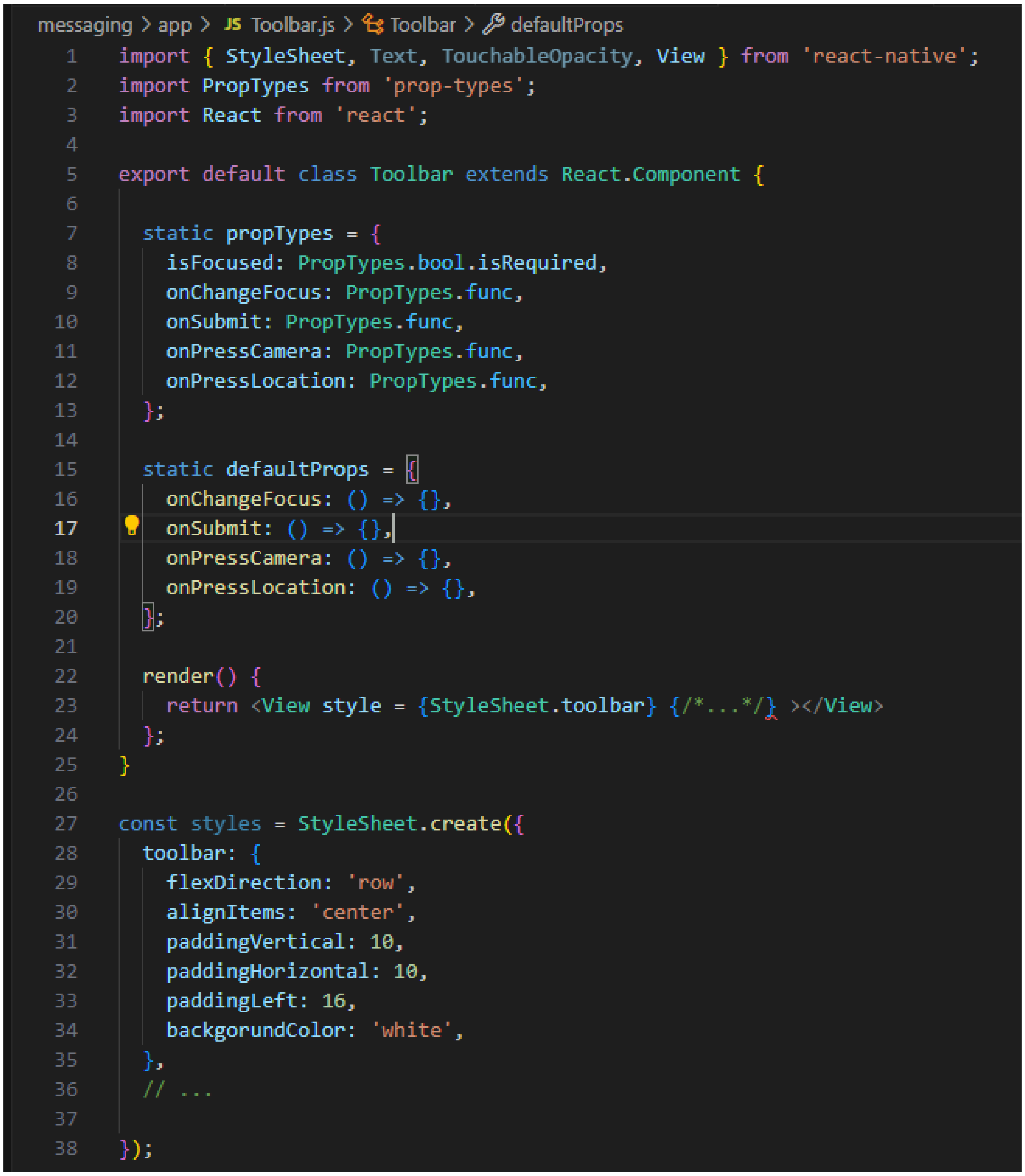
Refs
React lets us access the instance of any component we render using a ref prop. This is a special prop that we can supply a callback – the callback will be called with the instance as a parameter, after the component mounts (and before it unmounts). We can store a reference to the component instance. You can think of a component instance as the "this" when we access this.props or any method that's part of our class. In this case, the TextInput component class has a focus and blur method that can be called from the component instance. We can call these from within the lifecycle of our custom component to control the focus state of the TextInput.

Storing a ref
Let's capture a reference to the TextInput element we render.
We'll store this reference as this.input. We can then use this reference to imperatively focus and blur the input field with this.input.focus() and this.input.blur() when the isFocused prop changes.

```
// ...
export default class Toolbar extends React.Component {
  // ...
  setInputRef = (ref) => {
    this.input = ref;
  };

  componentWillReceiveProps(nextProps) {
    if (nextProps.isFocused !== this.props.isFocused) {
      if (nextProps.isFocused) {
        this.input.focus();
      } else {
        this.input.blur();
      }
    }
  }

  handleFocus = () => {
    const { onChangeFocus } = this.props;
    onChangeFocus(true);
  };

  handleBlur = () => {
    const { onChangeFocus } = this.props;
    onChangeFocus(false);
  };

  // ...
  render() {
    const { onPressCamera, onPressLocation } = this.props;
    // Grab this from state!
    const { text } = this.state;
    return (
      <View style={styles.toolbar}>
        {/* Use emojis for icons instead! */}
        <ToolbarButton title={"C"} onPress={onPressCamera} />
        <ToolbarButton title={"L"} onPress={onPressLocation} />
        <View style={styles.inputContainer}>
          <TextInput
            style={styles.input}
            underlineColorAndroid={"transparent"}
            placeholder={"Type something!"}
            blurOnSubmit={false}
            value={text}
            onChangeText={this.handleChangeText}
            onSubmitEditing={this.handleSubmitEditing}
            // Additional props!
            ref={this.setInputRef}
            onFocus={this.handleFocus}
            onBlur={this.handleBlur}
          />
        </View>
      </View>
    );
  }
```

```
}
```
 	
The onFocus prop of the TextInput will be called when the user taps within the input field, and the onBlur prop will be called when the user taps outside the input field. We use handleFocus and handleBlur to notify the parent of changes to the focus state
 	
What happens whenever the parent passes a different value for the isFocused prop?

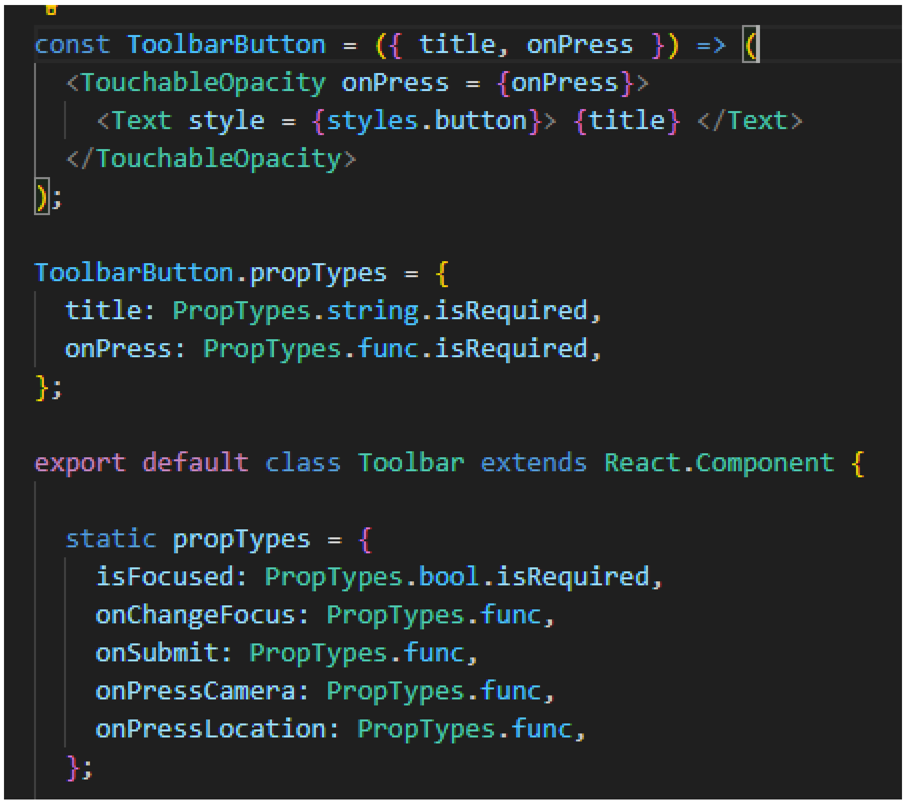
6. Output
ALFEROS **<u>Building the Toolbar</u>** Toolbar.js component



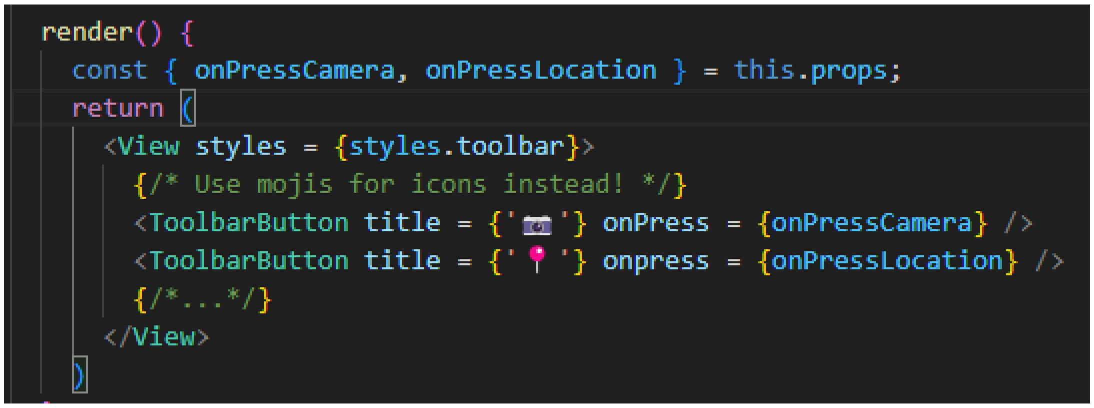
messaging > app > JS Toolbar.js > Toolbar > defaultProps

```javascript
import { StyleSheet, Text, TouchableOpacity, View } from 'react-native';
import PropTypes from 'prop-types';
import React from 'react';

export default class Toolbar extends React.Component {

  static propTypes = {
    isFocused: PropTypes.bool.isRequired,
    onChangeFocus: PropTypes.func,
    onSubmit: PropTypes.func,
    onPressCamera: PropTypes.func,
    onPressLocation: PropTypes.func,
  };

  static defaultProps = {
    onChangeFocus: () => {},
    onSubmit: () => {},
    onPressCamera: () => {},
    onPressLocation: () => {},
  };

  render() {
    return <View style = {StyleSheet.toolbar} {/*...*/} ></View>
  };
}

const styles = StyleSheet.create({
  toolbar: {
    flexDirection: 'row',
    alignItems: 'center',
    paddingVertical: 10,
    paddingHorizontal: 10,
    paddingLeft: 16,
    backgorundColor: 'white',
  },
  // ...

});
```

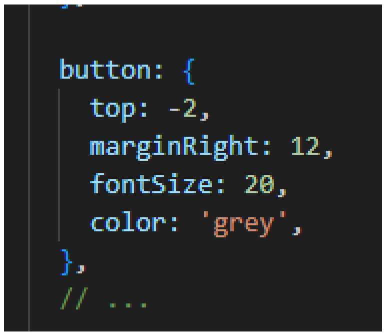
Camera button and location button



```
const ToolbarButton = ({ title, onPress }) => (
  <TouchableOpacity onPress = {onPress}>
    <Text style = {styles.button}> {title} </Text>
  </TouchableOpacity>
);

ToolbarButton.propTypes = {
  title: PropTypes.string.isRequired,
  onPress: PropTypes.func.isRequired,
};

export default class Toolbar extends React.Component {

  static propTypes = {
    isFocused: PropTypes.bool.isRequired,
    onChangeFocus: PropTypes.func,
    onSubmit: PropTypes.func,
    onPressCamera: PropTypes.func,
    onPressLocation: PropTypes.func,
  };
```

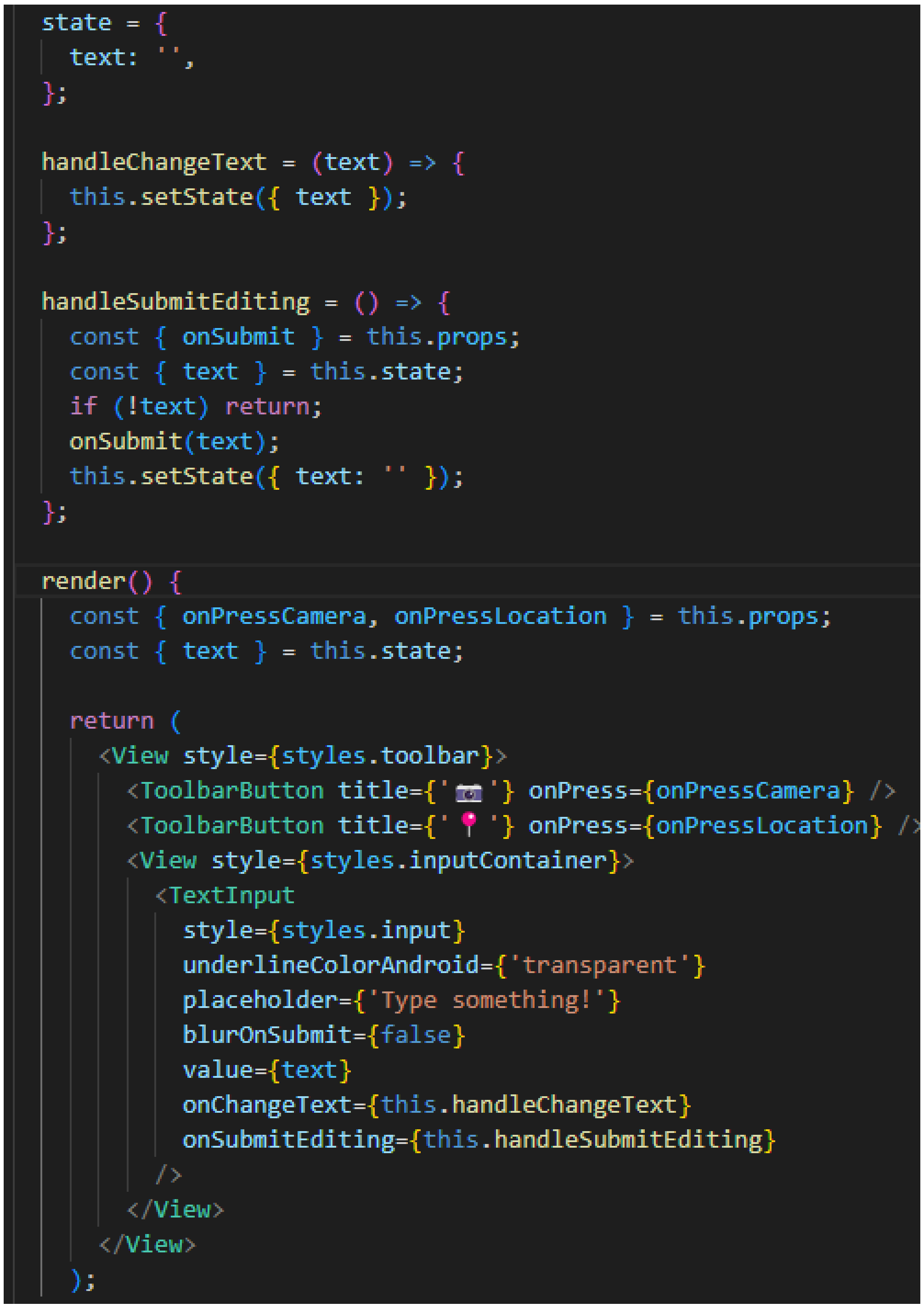


```
render() {
  const { onPressCamera, onPressLocation } = this.props;
  return (
    <View styles = {styles.toolbar}>
      {/* Use mojis for icons instead! */}
      <ToolbarButton title = {'📷'} onPress = {onPressCamera} />
      <ToolbarButton title = {'📍'} onpress = {onPressLocation} />
      {/*...*/}
    </View>
  )
```

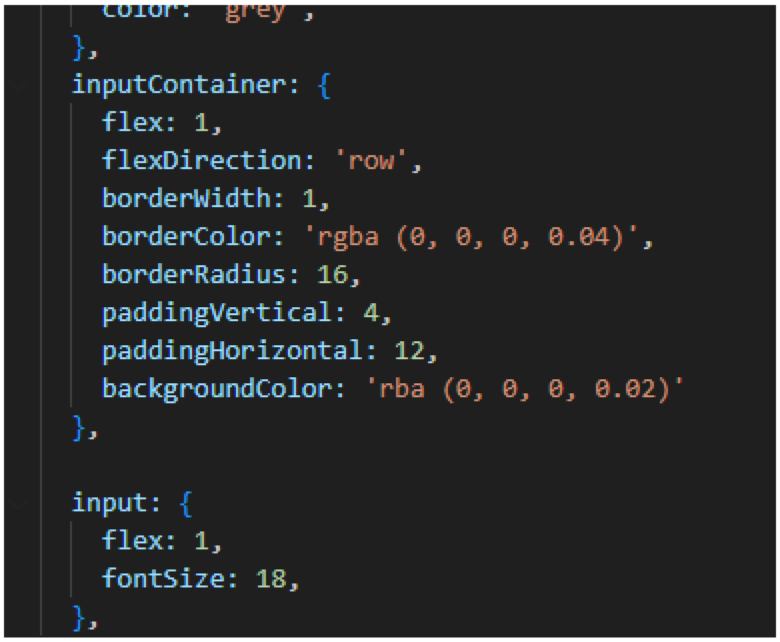


```
button: {
  top: -2,
  marginRight: 12,
  fontSize: 20,
  color: 'grey',
},
// ...
```

Modification



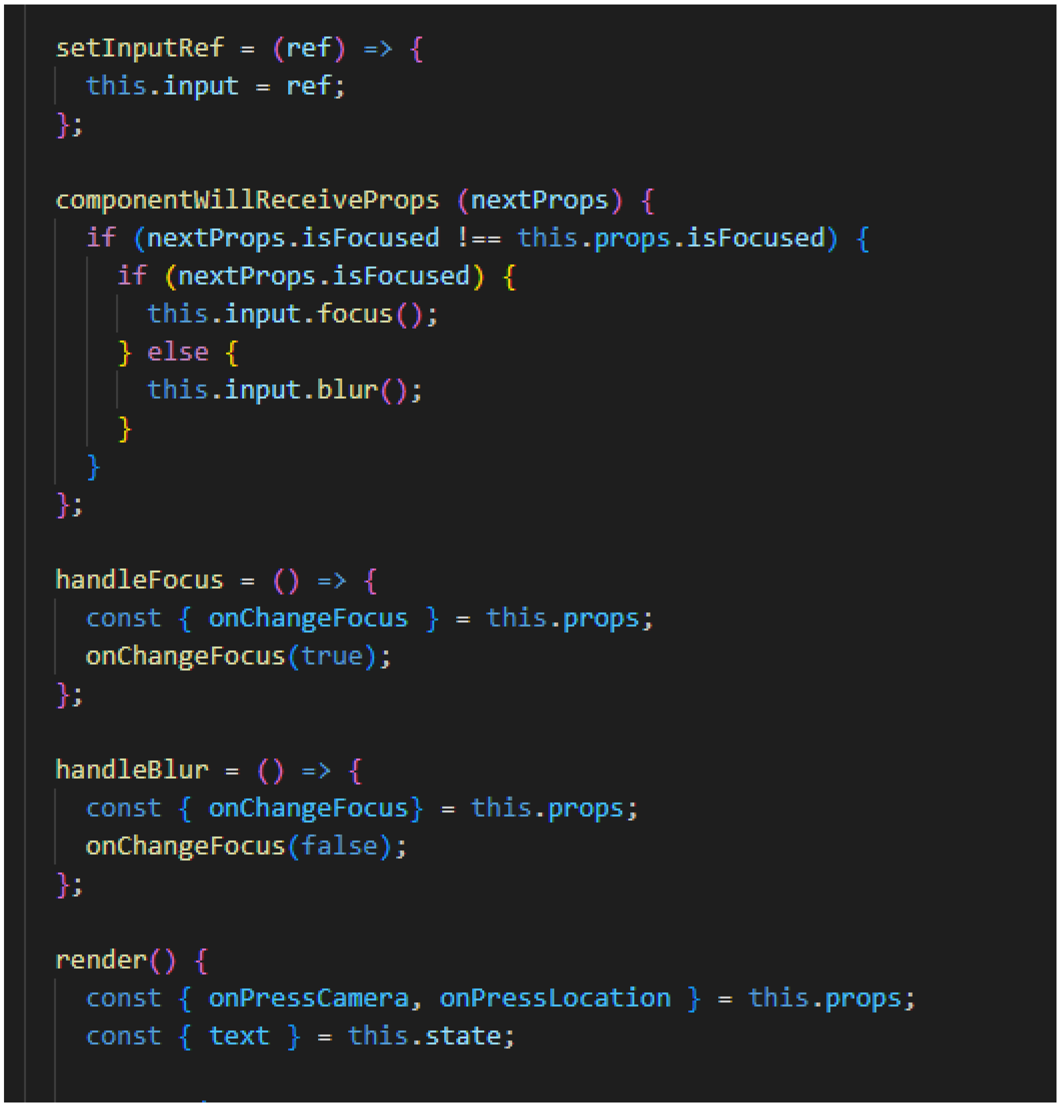
```
state = {
  text: '',
};

handleChangeText = (text) => {
  this.setState({ text });
};

handleSubmitEditing = () => {
  const { onSubmit } = this.props;
  const { text } = this.state;
  if (!text) return;
  onSubmit(text);
  this.setState({ text: '' });
};

render() {
  const { onPressCamera, onPressLocation } = this.props;
  const { text } = this.state;

  return (
    <View style={styles.toolbar}>
      <ToolbarButton title={'📷'} onPress={onPressCamera} />
      <ToolbarButton title={'📍'} onPress={onPressLocation} />
      <View style={styles.inputContainer}>
        <TextInput
          style={styles.input}
          underlineColorAndroid={'transparent'}
          placeholder={'Type something!'}
          blurOnSubmit={false}
          value={text}
          onChangeText={this.handleChangeText}
          onSubmitEditing={this.handleSubmitEditing}
        />
      </View>
    </View>
  );
```



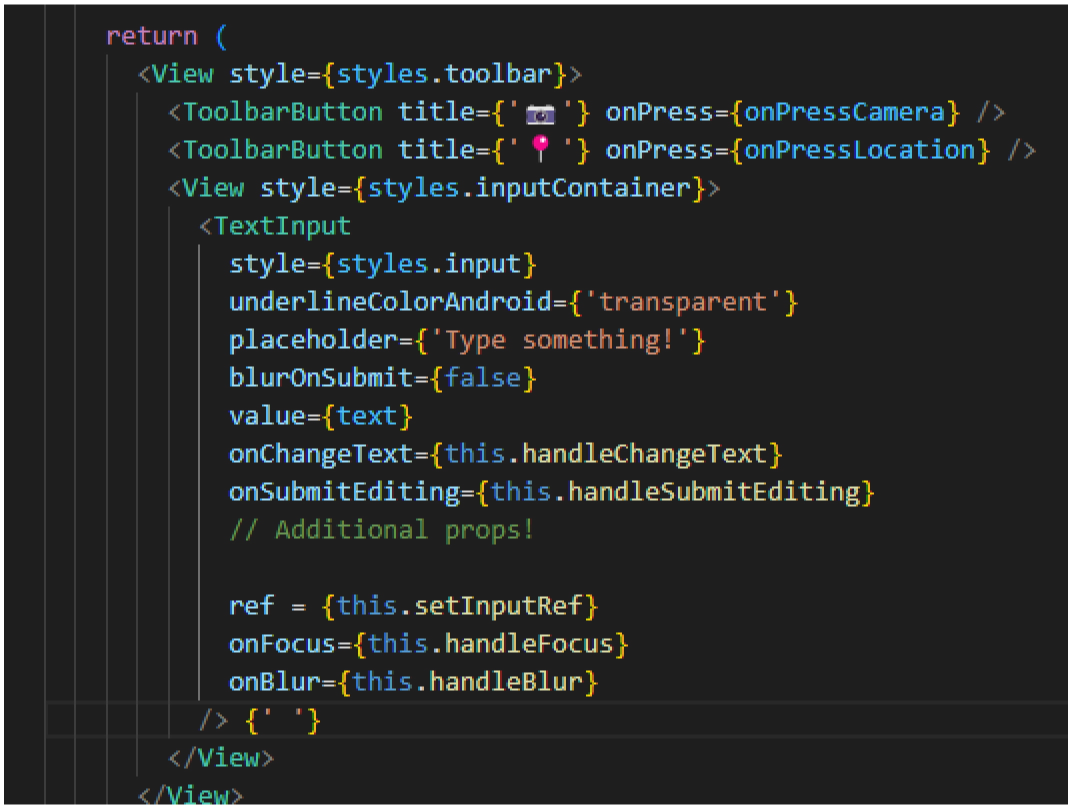
```
    color: 'grey',
  },
  inputContainer: {
    flex: 1,
    flexDirection: 'row',
    borderWidth: 1,
    borderColor: 'rgba (0, 0, 0, 0.04)',
    borderRadius: 16,
    paddingVertical: 4,
    paddingHorizontal: 12,
    backgroundColor: 'rba (0, 0, 0, 0.02)'
  },

  input: {
    flex: 1,
    fontSize: 18,
  },
```

Explain the purpose of using blurOnSubmit={false} in this code.
<u>It ensures that the TextInput stays focused after pressing the submit button, preventing the keyboard from dismissing.</u>

<u>Storing a ref</u>



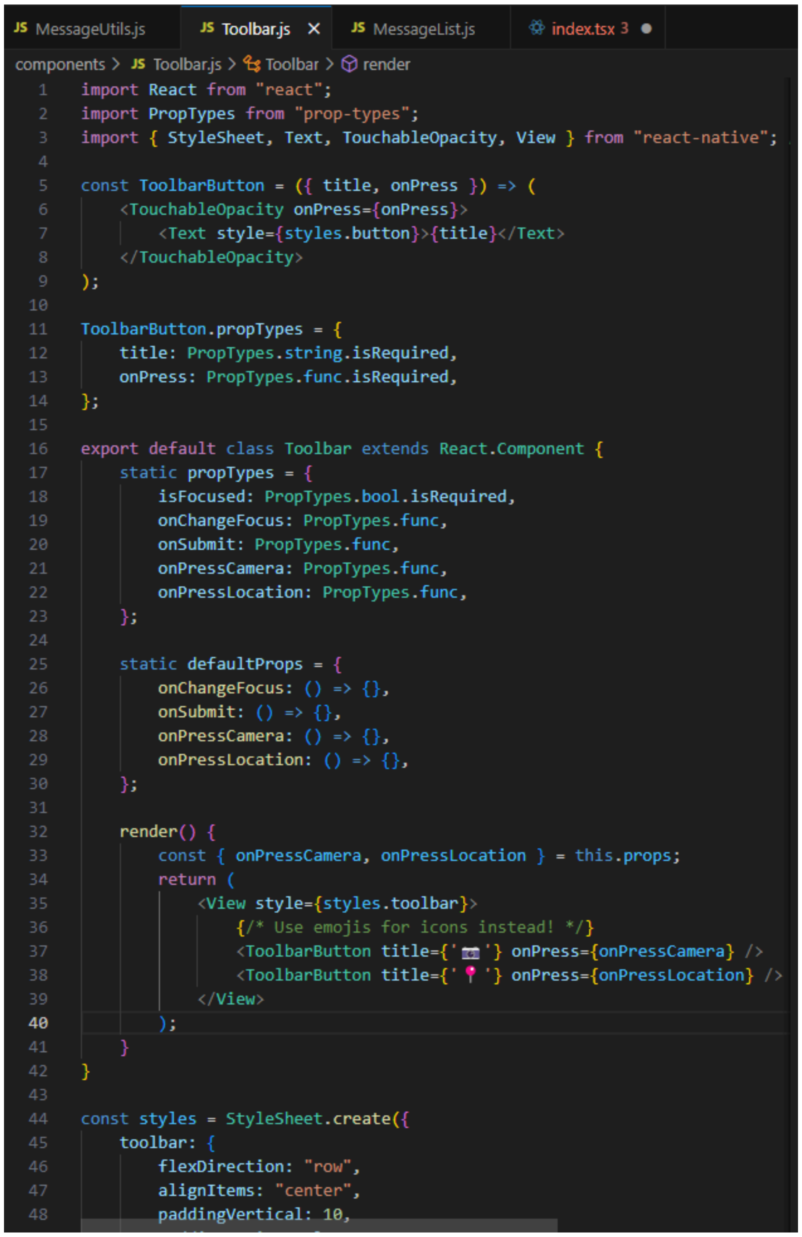
```
setInputRef = (ref) => {
  this.input = ref;
};

componentWillReceiveProps (nextProps) {
  if (nextProps.isFocused !== this.props.isFocused) {
    if (nextProps.isFocused) {
      this.input.focus();
    } else {
      this.input.blur();
    }
  }
};

handleFocus = () => {
  const { onChangeFocus } = this.props;
  onChangeFocus(true);
};

handleBlur = () => {
  const { onChangeFocus} = this.props;
  onChangeFocus(false);
};

render() {
  const { onPressCamera, onPressLocation } = this.props;
  const { text } = this.state;
```



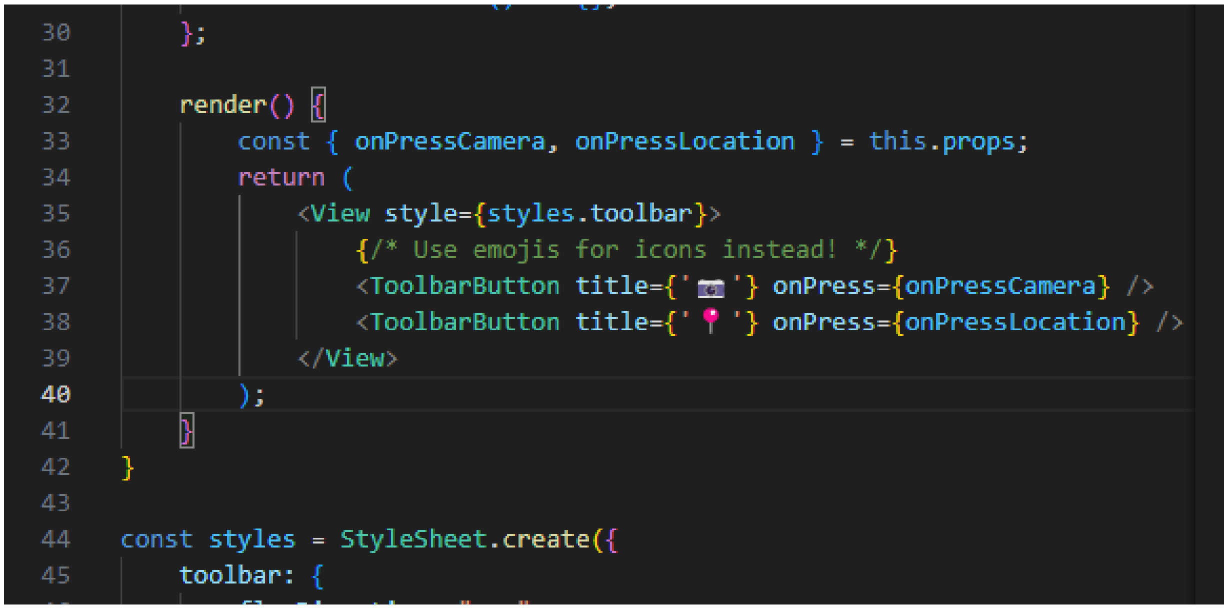
```
return (
  <View style={styles.toolbar}>
    <ToolbarButton title={'📷'} onPress={onPressCamera} />
    <ToolbarButton title={'📍'} onPress={onPressLocation} />
    <View style={styles.inputContainer}>
      <TextInput
        style={styles.input}
        underlineColorAndroid={'transparent'}
        placeholder={'Type something!'}
        blurOnSubmit={false}
        value={text}
        onChangeText={this.handleChangeText}
        onSubmitEditing={this.handleSubmitEditing}
        // Additional props!

        ref = {this.setInputRef}
        onFocus={this.handleFocus}
        onBlur={this.handleBlur}
      /> {' '}
    </View>
  </View>
```

What happens whenever the parent passes a different value for the isFocused prop?
The componentWillReceiveProps will trigger, which checks if the new value is different from the previous one. If isFocused is true, it focuses on the input field, otherwise, it blurs the input field.

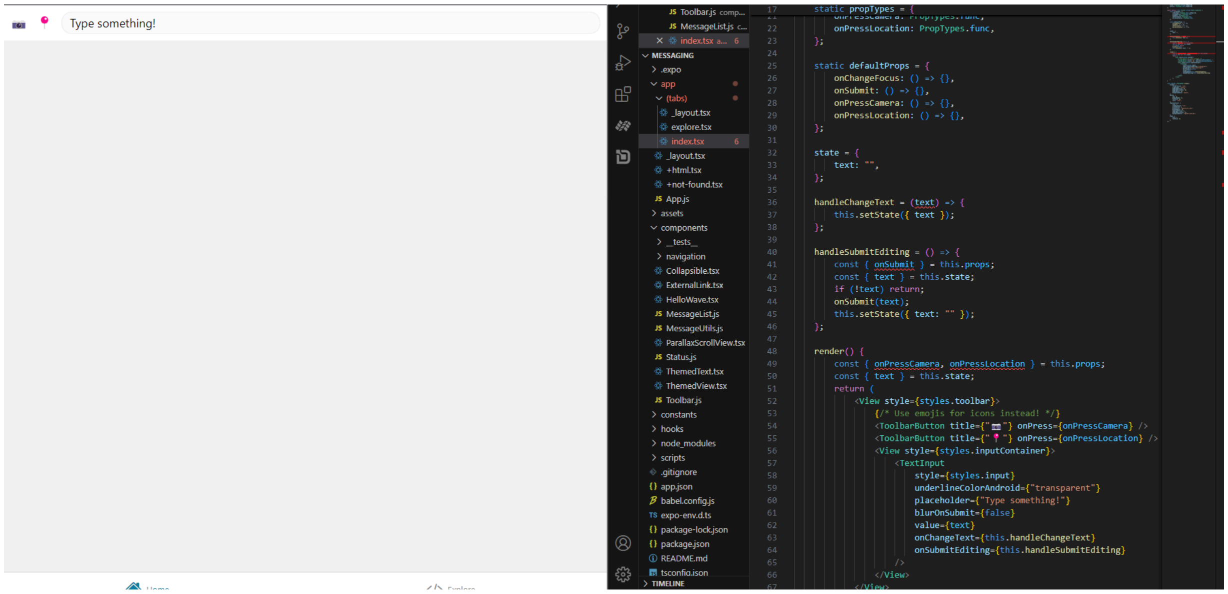
EFA



```
import React from "react";
import PropTypes from "prop-types";
import { StyleSheet, Text, TouchableOpacity, View } from "react-native";

const ToolbarButton = ({ title, onPress }) => (
    <TouchableOpacity onPress={onPress}>
        <Text style={styles.button}>{title}</Text>
    </TouchableOpacity>
);

ToolbarButton.propTypes = {
    title: PropTypes.string.isRequired,
    onPress: PropTypes.func.isRequired,
};

export default class Toolbar extends React.Component {
    static propTypes = {
        isFocused: PropTypes.bool.isRequired,
        onChangeFocus: PropTypes.func,
        onSubmit: PropTypes.func,
        onPressCamera: PropTypes.func,
        onPressLocation: PropTypes.func,
    };

    static defaultProps = {
        onChangeFocus: () => {},
        onSubmit: () => {},
        onPressCamera: () => {},
        onPressLocation: () => {},
    };

    render() {
        const { onPressCamera, onPressLocation } = this.props;
        return (
            <View style={styles.toolbar}>
                {/* Use emojis for icons instead! */}
                <ToolbarButton title={'📷'} onPress={onPressCamera} />
                <ToolbarButton title={'📍'} onPress={onPressLocation} />
            </View>
        );
    }
}

const styles = StyleSheet.create({
    toolbar: {
        flexDirection: "row",
        alignItems: "center",
        paddingVertical: 10,
```

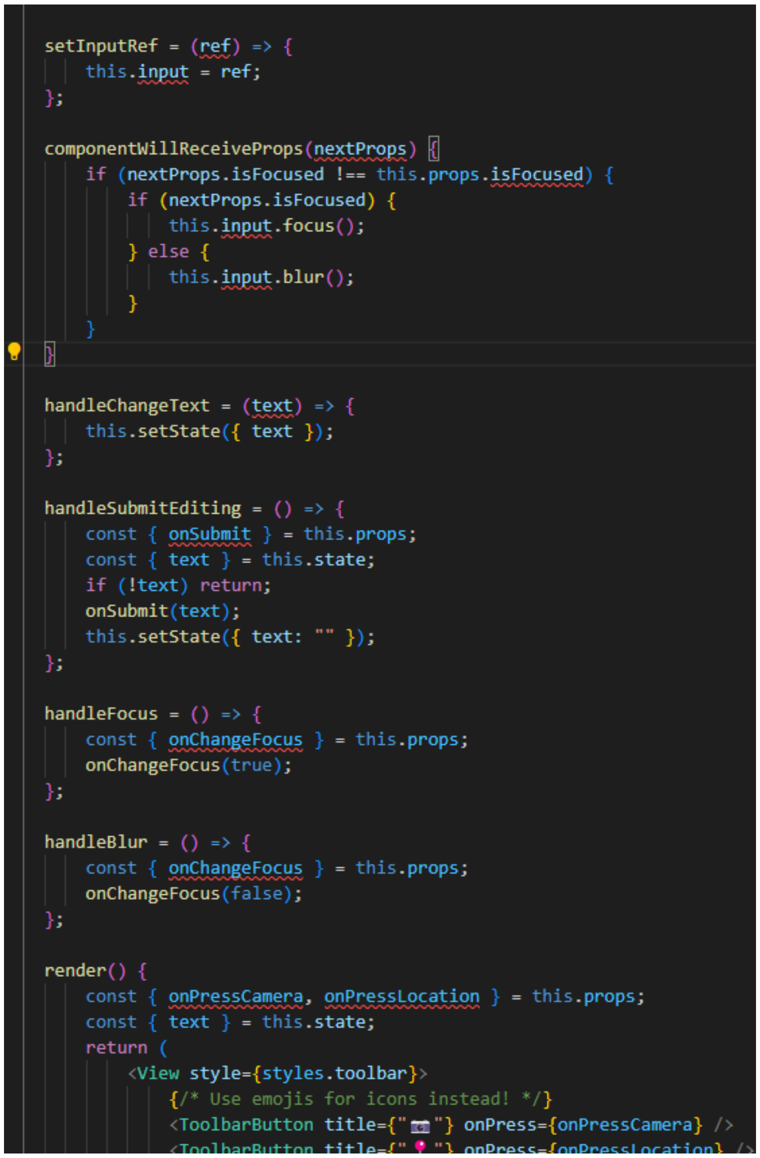


```
    };

    render() {
        const { onPressCamera, onPressLocation } = this.props;
        return (
            <View style={styles.toolbar}>
                {/* Use emojis for icons instead! */}
                <ToolbarButton title={'📷'} onPress={onPressCamera} />
                <ToolbarButton title={'📍'} onPress={onPressLocation} />
            </View>
        );
    }
}

const styles = StyleSheet.create({
    toolbar: {
```



```
    static propTypes = {
        onPressCamera: PropTypes.func,
        onPressLocation: PropTypes.func,
    };

    static defaultProps = {
        onChangeFocus: () => {},
        onSubmit: () => {},
        onPressCamera: () => {},
        onPressLocation: () => {},
    };

    state = {
        text: "",
    };

    handleChangeText = (text) => {
        this.setState({ text });
    };

    handleSubmitEditing = () => {
        const { onSubmit } = this.props;
        const { text } = this.state;
        if (!text) return;
        onSubmit(text);
        this.setState({ text: "" });
    };

    render() {
        const { onPressCamera, onPressLocation } = this.props;
        const { text } = this.state;
        return (
            <View style={styles.toolbar}>
                {/* Use emojis for icons instead! */}
                <ToolbarButton title={"📷"} onPress={onPressCamera} />
                <ToolbarButton title={"📍"} onPress={onPressLocation} />
                <View style={styles.inputContainer}>
                    <TextInput
                        style={styles.input}
                        underlineColorAndroid={"transparent"}
                        placeholder={"Type something!"}
                        blurOnSubmit={false}
                        value={text}
                        onChangeText={this.handleChangeText}
                        onSubmitEditing={this.handleSubmitEditing}
                    />
                </View>
            </View>
```

- blurOnSubmit={false} helps improve the user experience by allowing the TextInput to retain focus after submitting a message, enabling a smoother and more efficient messaging experience.



```
setInputRef = (ref) => {
  this.input = ref;
};

componentWillReceiveProps(nextProps) {
  if (nextProps.isFocused !== this.props.isFocused) {
    if (nextProps.isFocused) {
      this.input.focus();
    } else {
      this.input.blur();
    }
  }
}

handleChangeText = (text) => {
  this.setState({ text });
};

handleSubmitEditing = () => {
  const { onSubmit } = this.props;
  const { text } = this.state;
  if (!text) return;
  onSubmit(text);
  this.setState({ text: "" });
};

handleFocus = () => {
  const { onChangeFocus } = this.props;
  onChangeFocus(true);
};

handleBlur = () => {
  const { onChangeFocus } = this.props;
  onChangeFocus(false);
};

render() {
  const { onPressCamera, onPressLocation } = this.props;
  const { text } = this.state;
  return (
    <View style={styles.toolbar}>
      {/* Use emojis for icons instead! */}
      <ToolbarButton title={"📷"} onPress={onPressCamera} />
      <ToolbarButton title={"📍"} onPress={onPressLocation} />
```

- When the parent updates the isFocused prop, the Toolbar component's componentWillReceiveProps method adjusts the input field's focus accordingly

7. Supplementary Activity

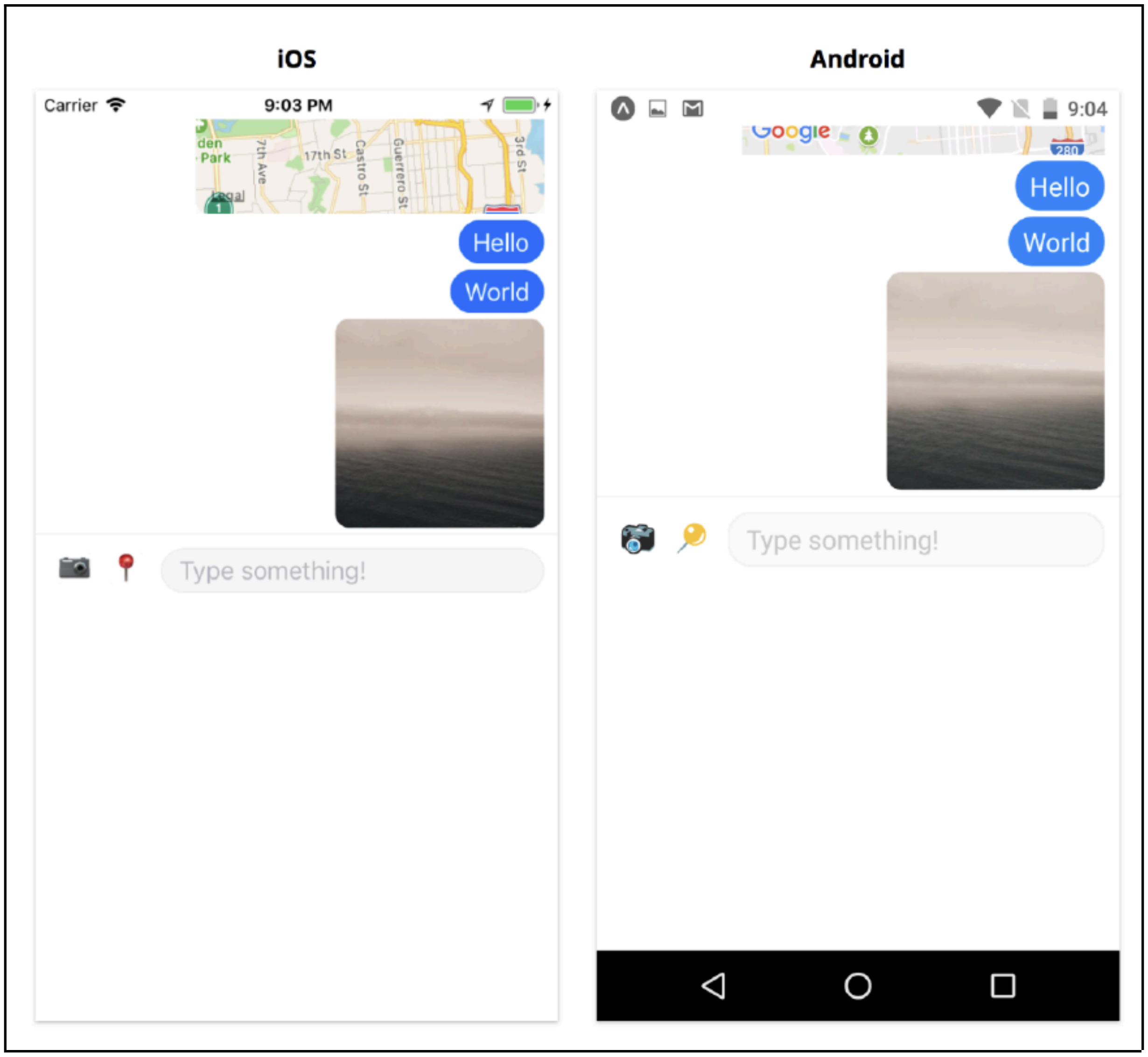
Adding the ToolBar to the App

Let's now render our Toolbar component from App.js. We can handle the onSubmit event to populate our message list with real messages. When we type in the input field and submit the text (by pressing the return key on the keyboard), we can add a new message to the messages in state using our createTextMessage utility function. We'll also add a few callback functions as placeholders.

messaging / App.js:

```
import Toolbar from "./components/Toolbar";
// ...
export default class App extends React.Component {
  state = {
    // ...
    isInputFocused: false,
  };

  handlePressToolbarCamera = () => {
    // ...
  };

  handlePressToolbarLocation = () => {
    // ...
  };

  handleChangeFocus = (isFocused) => {
    this.setState({ isInputFocused: isFocused });
  };

  handleSubmit = (text) => {
    const { messages } = this.state;
    this.setState({
      messages: [createTextMessage(text), ...messages],
    });
  };

  renderToolbar() {
    const { isInputFocused } = this.state;
    return (
      <View style={styles.toolbar}>
        <Toolbar
          isFocused={isInputFocused}
          onSubmit={this.handleSubmit}
          onChangeFocus={this.handleChangeFocus}
          onPressCamera={this.handlePressToolbarCamera}
          onPressLocation={this.handlePressToolbarLocation}
        />
      </View>
    );
  }
  // ...
}
// ...
```

We'll store isInputFocused in this.state to keep track of the focus state of the TextInput in toolbar. After saving App.js, your toolbar should look like this:



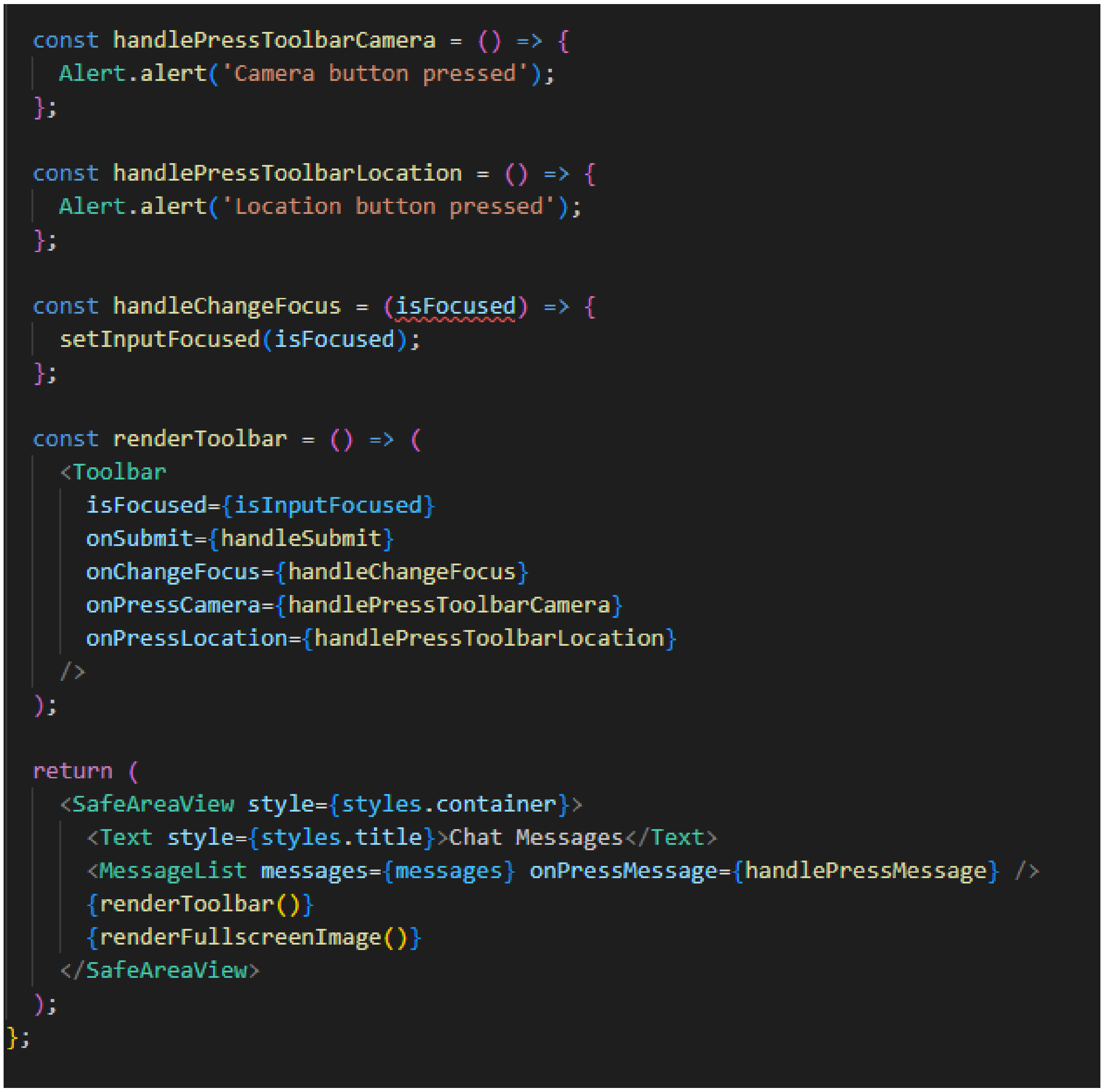
Is this similar to your output? If not, why? Show screenshot.

There's an interesting edge case when we open a fullscreen image preview while the input field is focused – the keyboard stays up even though the input field is no longer visible!

We can address this by updating our handlePressMessage function to also set isInputFocused: false in the component's state. Demonstrate and show a screenshot.

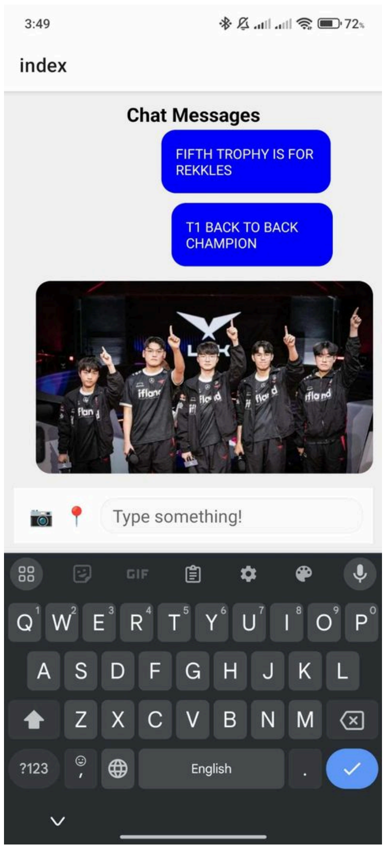
Now the keyboard should be dismissed when we tap an image to preview it fullscreen. **Make sure to demonstrate .**

ALFEROS



```
  const handlePressToolbarCamera = () => {
    Alert.alert('Camera button pressed');
  };

  const handlePressToolbarLocation = () => {
    Alert.alert('Location button pressed');
  };

  const handleChangeFocus = (isFocused) => {
    setInputFocused(isFocused);
  };

  const renderToolbar = () => (
    <Toolbar
      isFocused={isInputFocused}
      onSubmit={handleSubmit}
      onChangeFocus={handleChangeFocus}
      onPressCamera={handlePressToolbarCamera}
      onPressLocation={handlePressToolbarLocation}
    />
  );

  return (
    <SafeAreaView style={styles.container}>
      <Text style={styles.title}>Chat Messages</Text>
      <MessageList messages={messages} onPressMessage={handlePressMessage} />
      {renderToolbar()}
      {renderFullscreenImage()}
    </SafeAreaView>
  );
};
```

Demo

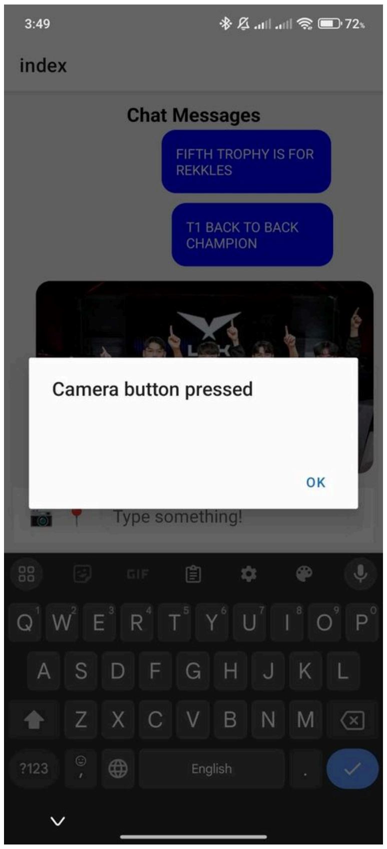

3:49

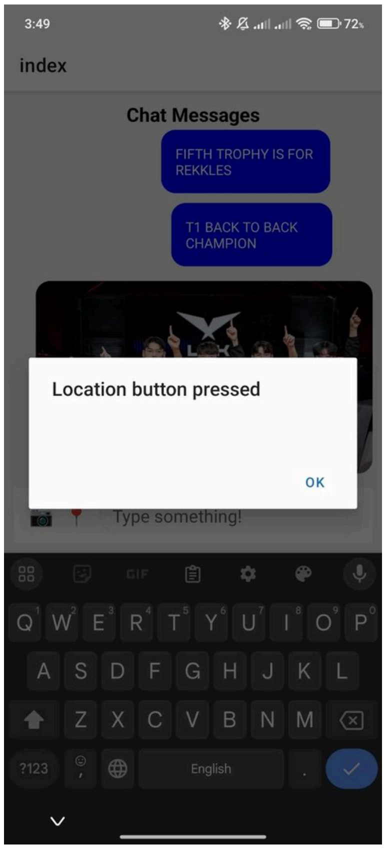
index

Chat Messages

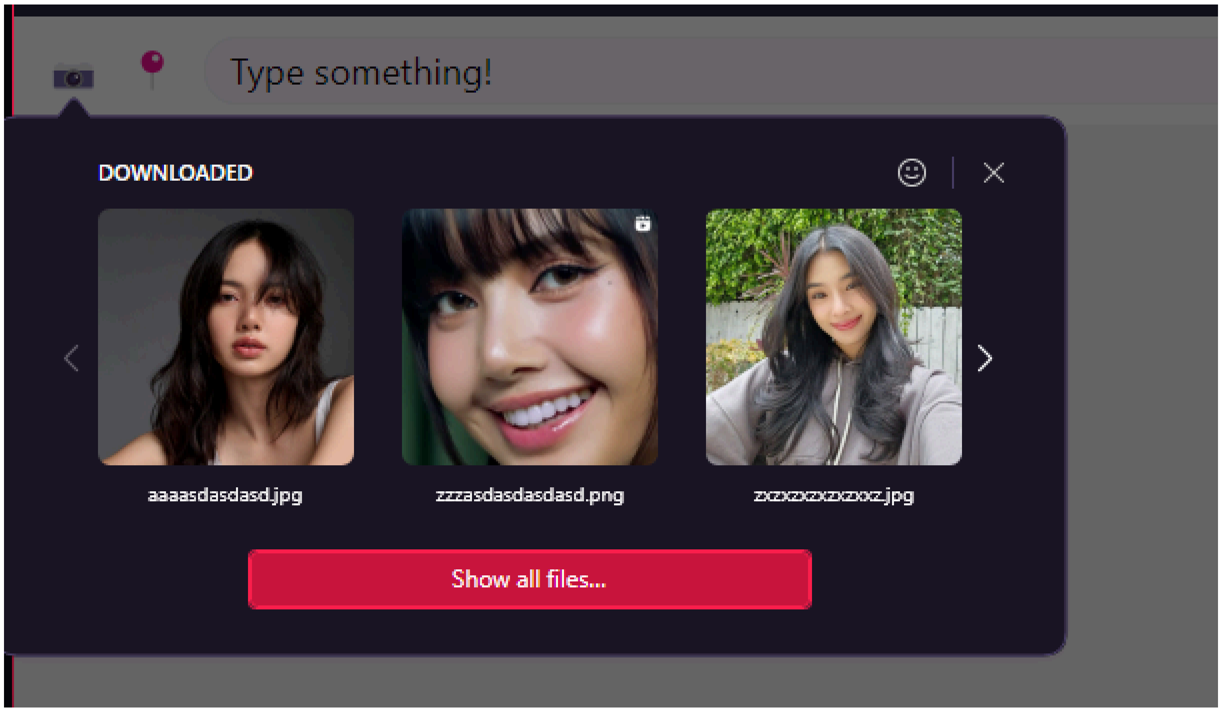
FIFTH TROPHY IS FOR REKKLES

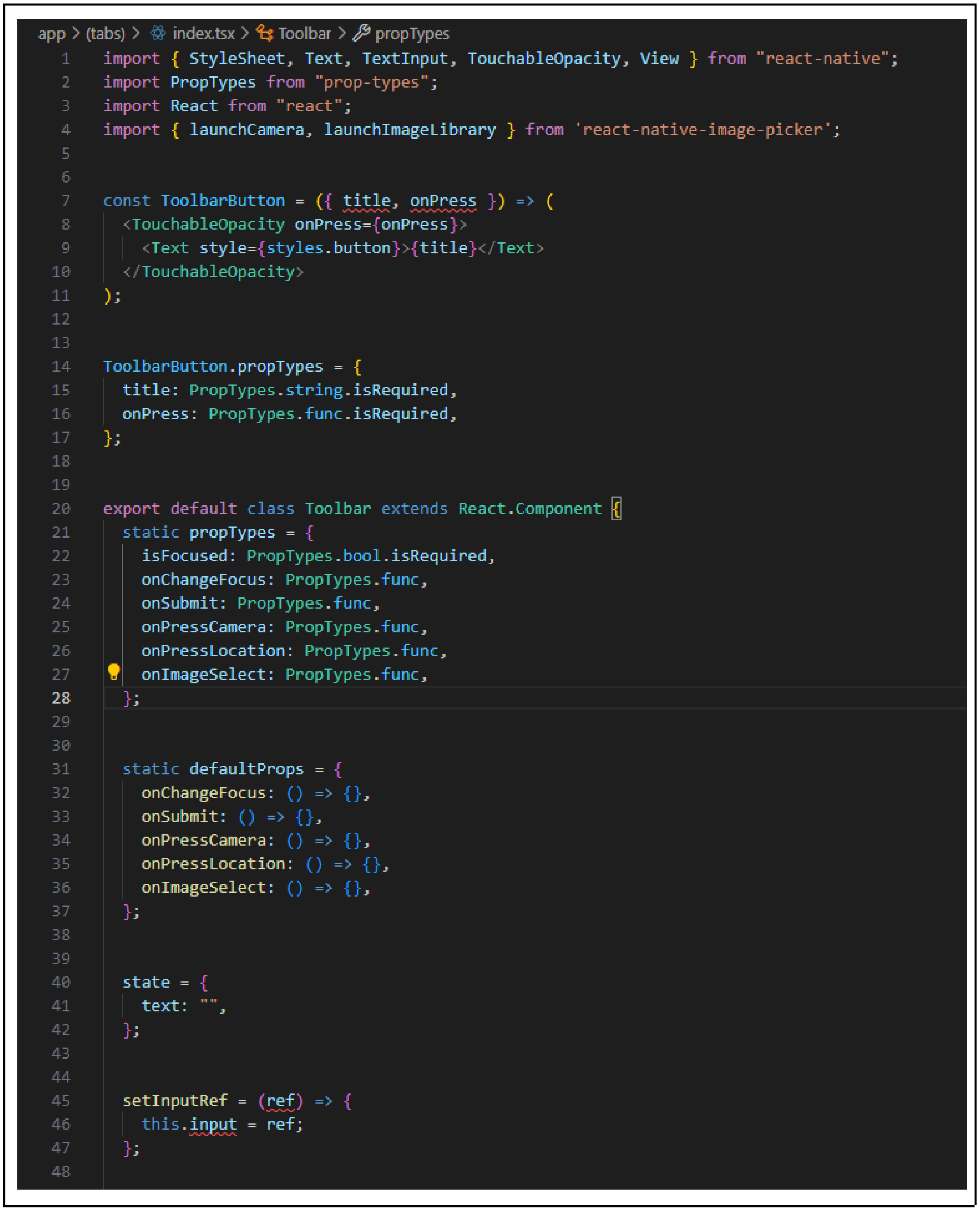
T1 BACK TO BACK CHAMPION


Camera button pressed


Location button pressed

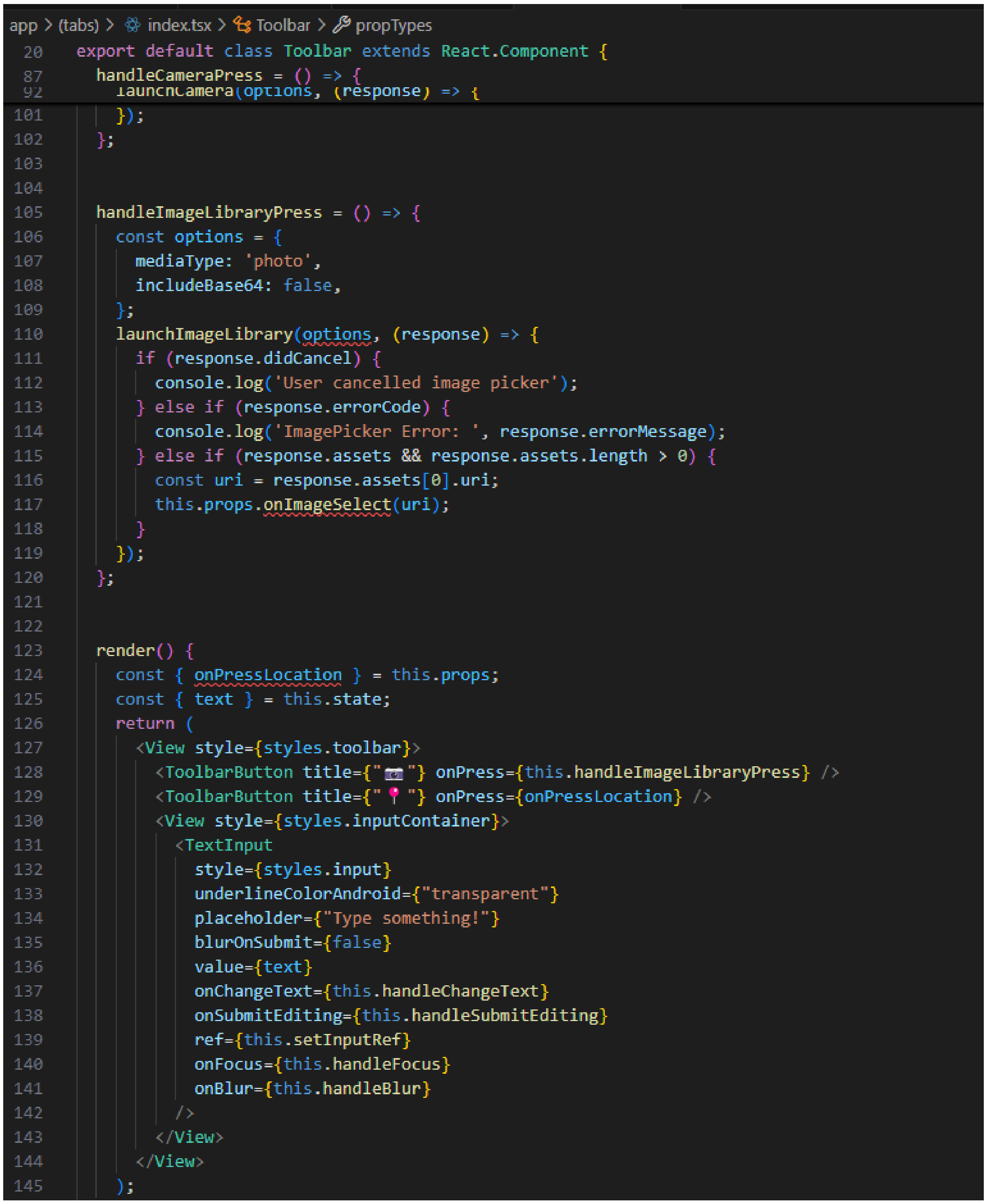
EFA





app > (tabs) > index.tsx > Toolbar > propTypes

```
import { StyleSheet, Text, TextInput, TouchableOpacity, View } from "react-native";
import PropTypes from "prop-types";
import React from "react";
import { launchCamera, launchImageLibrary } from 'react-native-image-picker';


const ToolbarButton = ({ title, onPress }) => (
  <TouchableOpacity onPress={onPress}>
    <Text style={styles.button}>{title}</Text>
  </TouchableOpacity>
);


ToolbarButton.propTypes = {
  title: PropTypes.string.isRequired,
  onPress: PropTypes.func.isRequired,
};


export default class Toolbar extends React.Component {
  static propTypes = {
    isFocused: PropTypes.bool.isRequired,
    onChangeFocus: PropTypes.func,
    onSubmit: PropTypes.func,
    onPressCamera: PropTypes.func,
    onPressLocation: PropTypes.func,
    onImageSelect: PropTypes.func,
  };


  static defaultProps = {
    onChangeFocus: () => {},
    onSubmit: () => {},
    onPressCamera: () => {},
    onPressLocation: () => {},
    onImageSelect: () => {},
  };


  state = {
    text: "",
  };


  setInputRef = (ref) => {
    this.input = ref;
  };

```



```
app > (tabs) > index.tsx > Toolbar > propTypes
export default class Toolbar extends React.Component {
  handleCameraPress = () => {
    launchCamera(options, (response) => {
    });
  };


  handleImageLibraryPress = () => {
    const options = {
      mediaType: 'photo',
      includeBase64: false,
    };
    launchImageLibrary(options, (response) => {
      if (response.didCancel) {
        console.log('User cancelled image picker');
      } else if (response.errorCode) {
        console.log('ImagePicker Error: ', response.errorMessage);
      } else if (response.assets && response.assets.length > 0) {
        const uri = response.assets[0].uri;
        this.props.onImageSelect(uri);
      }
    });
  };


  render() {
    const { onPressLocation } = this.props;
    const { text } = this.state;
    return (
      <View style={styles.toolbar}>
        <ToolbarButton title={"📷"} onPress={this.handleImageLibraryPress} />
        <ToolbarButton title={"📍"} onPress={onPressLocation} />
        <View style={styles.inputContainer}>
          <TextInput
            style={styles.input}
            underlineColorAndroid={"transparent"}
            placeholder={"Type something!"}
            blurOnSubmit={false}
            value={text}
            onChangeText={this.handleChangeText}
            onSubmitEditing={this.handleSubmitEditing}
            ref={this.setInputRef}
            onFocus={this.handleFocus}
            onBlur={this.handleBlur}
          />
        </View>
      </View>
    );
```

- No, since there is a certain problem on designing the app that always the toolbar are on the top and having a hard time finding a way putting it in the center.

8. Conclusion

ALFEROS

- In this activity, we explored the different props that can be used in the Toolbar.js. Props such as isFocused, onSubmit, onChangeFocus, onPressCamera, and onPressLocation are used to receive functions and data from the user. With these, our messaging app can now send text, image, and location via user input, instead of hard coding the messages. Overall, this activity helped us improve our knowledge in creating an interactive and engaging app on React Native.

EFA

- This code demonstrates how to create a customizable toolbar in React Native with buttons for camera and location access, plus a text input field. We addressed potential issues, such as ensuring the correct onPress handlers are assigned and verifying that camera permissions are properly set to troubleshoot any errors with button functionality.

9. Assessment Rubric