

Activity No. 6 Introduction to Core APIs	
Course Code: CPE 026	Program: Computer Engineering
Course Title: Emerging Technologies 3 in CpE	Date Performed: 10/23/2024
Section: CPE41S8	Date Submitted: 10/24/2024
Name: Christian Ed B. Efa Joshua L. Alferos	Instructor: Engr. Richard Roman
1. Objective(s)	
This activity aims to introduce the students to the Core APIs used in React Native by building a simple messaging application.	
2. Intended Learning Outcomes (ILOs)	
After this module, the student should be able to: <ul style="list-style-type: none"> • Develop an application using both functional and object-oriented programming in React Native. • Utilize core APIs to enable functionality within the application. 	
3. Discussion	
<u>Building a messaging app</u> Our messaging app should enable people to communicate. Specifically it must allow the users to: <ol style="list-style-type: none"> 1. Send text messages, images, and maps; 2. Choose images from the device camera roll; and 3. View images full screen. For this activity, we will use some of the following APIs to create a simple messaging app. <ul style="list-style-type: none"> • Alert - Displays modal dialog windows for simple user input • BackHandler - Controls the back button on Android • CameraRoll - Returns images and videos stored on the device • Dimensions - Returns the dimensions of the screen • Geolocation - Returns the location of the device, and emits events when the location changes • Keyboard - Emits events when the keyboard appears or disappears • NetInfo - Returns network connectivity information, and emits events when the connectivity changes • PixelRatio - Translates from density-independent pixels to density-dependent pixels (more detail on the later) • StatusBar - Controls the visibility and color of the status bar We'll just be focusing on the UI, so we won't actually send messages, but we could connect	

the UI we build to a backend if we wanted to use it in a production app.

4. Materials and Equipment

To properly perform this activity, the student must have:

- Node.js LTS
- Expo
- Emulator / Simulator / Android or iOS Mobile Device

5. Procedure

Initializing the Project

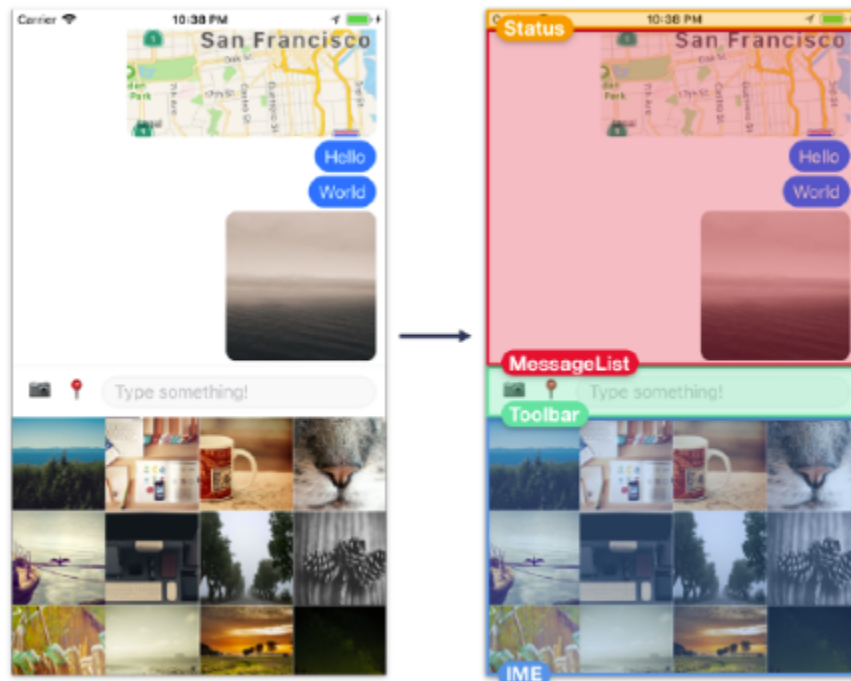
Like we have done with previous activities, we are initializing the project through the following steps:

1. Create a new expo app called messaging.
2. Navigate into the messaging director afterwards.

Section end, screenshot of progress is required for the output section.

The App

The Application should have the following features, observe the app's skeleton below:



The components that make up the skeleton are as follows:

- Status - The device generally renders a status bar, the horizontal strip at the top of the screen that shows time, battery life, etc - but in this case, we'll augment it to show network connectivity more prominently. We'll create our own component, Status, which renders beneath the device's status bar.
- MessageList - This is where we'll render text messages, images, and maps.
- Toolbar - This is where the user can switch between sending text, images, or location, and where the input field for typing messages lives.

- Input Method Editor (IME) - This is where we can render a custom input method, i.e. sending images. We'll build an image picker component, ImageGrid, and use it here. Note that the keyboard is rendered natively by the operating system, so we will trigger the keyboard to appear and disappear at the right times, but we won't render it ourselves.

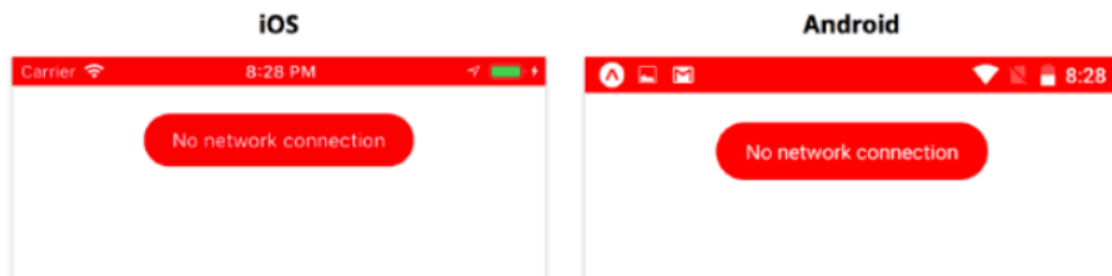
Now, in your App.js, follow the procedures:

1. Create the skeleton of the application using functional React Native, like we have done in the previous activity.
2. Create the stylesheet objects with the following specifications:
 - a. Container with flex 1 and background color white.
 - b. Content with flex 1 and background color white.
 - c. inputMethodEditor with flex 1 and background color white.
 - d. Toolbar with borderTopWidth 1, borderTopColor as 'rgba(0,0,0,0.04)', and background color white.
3. Run the app and preview your progress.

Section end, screenshot of progress is required for the output section.

Network Connectivity Connector

Since we're building a messaging app, network connectivity is relevant at all times. Let's let the user know when they've lost connectivity by turning the status bar red and displaying a short message.



1. Open app.json and add the following code to allow our network status bar to use a solid color. Note: You can ignore this if you're not using Android.

```
"androidStatusBar": {
  "barStyle": "dark-content",
  "backgroundColor": "#FFFFFF"
}
```

2. Create a new status bar component and include imports for Constants from 'expo' and StyleSheet from React Native.
3. We'll have two visual states: one where the user is connected to the network, and one where the user is disconnected. We'll set the color for each state in render, so let's start with the base style for the status bar:

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

const statusHeight = (Platform.OS === 'ios' ? Constants.statusBarHeight : 0)

const styles = StyleSheet.create({
  status: {
    zIndex: 1,
    height: statusHeight,
  }
})
```

The base style status will give the View its height. The View will have the same height regardless of whether this component is in the connected or disconnected state. We use a zIndex of 1 to indicate that this View should be drawn on top of other content – this will be relevant later, since we’re going to render a ScrollView beneath it.

4. We’ll store the network connectivity status in component state as state.info. Network connectivity status can have several different states, but let’s assume for now that if the state.info is "none" then we’re disconnected, and anything else means we’re connected. Let’s try rendering this background View.

```
import { Constants } from 'expo';
import { NetInfo, Platform, StatusBar, StyleSheet, Text, View } from
'react-native';
import React from 'react';

export default class Status extends React.Component{

  state = {
    info: null,
  };

  render() {
    const {info} = this.state;

    const isConnected = info !== 'none';
    const backgroundColor = isConnected ? 'white' : 'red';

    if(Platform.OS === 'ios'){
      return <View style={[styles.status, {backgroundColor}]}></View>
    }

    return null; //Temporary!
  }
}

const statusHeight = (Platform.OS === 'ios' ? Constants.statusBarHeight : 0)

const styles = StyleSheet.create({
  status: {
    zIndex: 1,
    height: statusHeight,
  }
})
```

Notice how we use an array for the View to apply two styles: the status style, and then a style object containing a different background color depending on whether we’re connected to the network or not.

5. Let's save Status.js and import it from App.js so we can see what we have so far. We can now go ahead and render our new Status component from App:

```
import Status from './components/Status';

export default function App() {
  return (
    <View style={styles.container}>
      <Status />
      // ...
    </View>
  );
}
```

We shouldn't see anything yet... but to verify that everything is working, you can temporarily set info: 'none' in the state of Status. If possible, you should test this on iOS. You should have a similar result to the image below:



Section end, you must provide a screenshot of your progress so far. Note all of your observations and lessons learned so far.

Using StatusBar

1. Let's import it from react-native and render it within our View. Place this before the if (Platform.OS === 'ios')

```
const statusBar = (
  <StatusBar
    backgroundColor={backgroundColor}
    barStyle={isConnected ? 'dark-content' : 'light-content'}
    animated={false}
  />
);
```

Here we set barStyle to dark-content if we're connected (black text on our white background) and light-content if we're disconnected (white text on our red background). We set backgroundColor to set the correct background color on Android. We also set animated to false – since we're not animating the background color on iOS, animating the text color won't look very good.

Section end, you must provide a screenshot of your progress so far. All of your observations and lessons learned so far have to be noted.

Message Bubble

Since the red status bar alone doesn't indicate anything about network connectivity, let's also add a short message in a floating bubble at the top of the screen.



1. Create a new messageContainer so that your code looks similar to what is shown below.

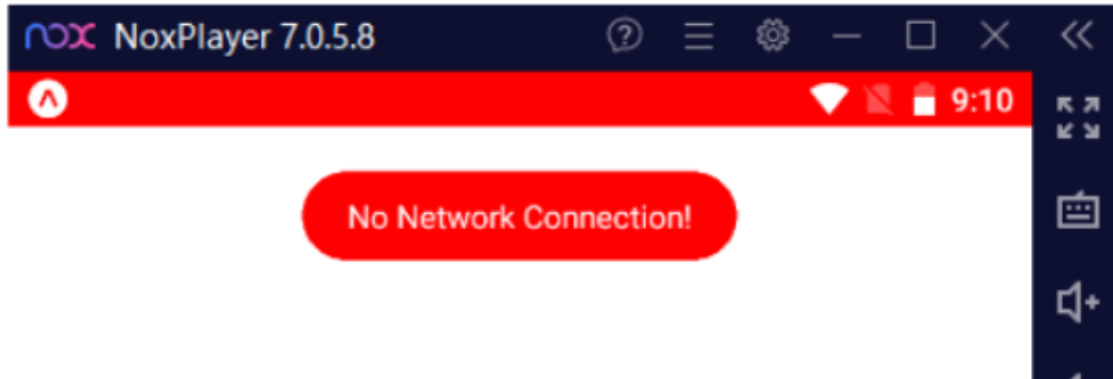
```
import { Constants } from "expo";
import { NetInfo, StatusBar, StyleSheet, Text, View } from "react-native";
import React from "react";

export default class Status extends React.Component {
  // ...
  render() {
    const { info } = this.state;
    const isConnected = info !== "none";
    const backgroundColor = isConnected ? "white" : "red";
    const statusBar = (
      <StatusBar
        backgroundColor={backgroundColor}
        barStyle={isConnected ? "dark-content" : "light-content"}
        animated={false}
      />
    );
    const messageContainer = (
      <View style={styles.messageContainer} pointerEvents={"none"}>
        {statusBar}
        {!!isConnected && (
          <View style={styles.bubble}>
            <Text style={styles.text}>No network connection</Text>
          </View>
        )}
      </View>
    );
    if (Platform.OS === "ios") {
      return (
        <View style={[styles.status, { backgroundColor }]}>
          {messageContainer}
        </View>
      );
    }
    return messageContainer;
  }
}
```

2. Create a StyleSheet object so that you have the following specifications.
 - a. messageContainer has a zIndex of 1, position absolute, top value

- statusHeight + 20, right and left as 0, height as 80, and alignItems to center.
- b. Bubble with paddingHorizontal value of 20, paddingVertical value of 10, borderRadius value of 20, and backgroundColor value of 'red'.
 - c. Text with color white.

Screenshot your current progress, your output must be similar to the image below. Is it similar? Did you encounter any challenges answering this activity?



6. Output

Your output for this activity contributes toward the achievement of ***ILO1: Develop an application using both functional and object-oriented programming in React Native.***

ALFEROS

Initializing the Project

```
PS C:\Users\admin\Documents\Emtech3-HOA> npx create-expo-app messaging
✓ Downloaded and extracted project files.
> npm install
```

```
PS C:\Users\admin\Documents\Emtech3-HOA> cd messaging
PS C:\Users\admin\Documents\Emtech3-HOA\messaging> npx expo start
Starting project at C:\Users\admin\Documents\Emtech3-HOA\messaging
Starting Metro Bundler
```

```
PS C:\Users\admin\Documents\Emtech3-HOA\messaging> npm run reset-project
```

app/_layout.tsx created.

PS C:\Users\admin\Documents\Emtech3-HOA\messaging> npx expo start
Starting project at C:\Users\admin\Documents\Emtech3-HOA\messaging
Starting Metro Bundler



```
🔧 _layout.tsx M ●
messaging > app > 🔧 _layout.tsx > ...
1  import { Stack } from "expo-router";
2
3  export default function RootLayout() {
4    return (
5      <Stack>
6        <Stack.Screen name="index" />
7      </Stack>
8    );
9  }
10
```

The App

Status

```
messaging > app > JS Status.js > ...
1  import React from 'react';
2  import { View, Text, StyleSheet } from 'react-native';
3
4  const Status = () => {
5    return (
6      <View style={styles.container}>
7        <Text style={styles.statusText}>Status: sdsdsd</Text>
8      </View>
9    );
10 };
11
12 const styles = StyleSheet.create({
13   container: {
14     backgroundColor: 'lightgray',
15     padding: 10,
16   },
17   statusText: {
18     fontSize: 16,
19     color: 'black',
20   },
21 });
22
23 export default Status;
24
```

MessageList

```

messaging > app > JS MessageList.js > [⌘] styles
1  import React from 'react';
2  import { FlatList, Text, View, StyleSheet } from 'react-native';
3
4  const messages = [
5    { id: '1', type: 'text', content: 'Hoa 9.1' },
6    { id: '2', type: 'text', content: 'Work in Progress :D' },
7  ];
8
9  const MessageList = () => {
10   const renderItem = ({ item }) => (
11     <Text style={styles.textMessage}>{item.content}</Text>
12   );
13
14   return (
15     <FlatList
16       data={messages}
17       keyExtractor={({ item }) => item.id}
18       renderItem={renderItem}
19     />
20   );
21 };
22
23 const styles = StyleSheet.create([
24   container: {
25     flex: 1,
26     backgroundColor: 'white',
27   },
28   textMessage: {
29     fontSize: 18,
30     padding: 10,
31   },
32 ]);
33
34 export default MessageList;

```

Toolbar

```
messaging > app > JS Toolbar.js > ...
1  import React from 'react';
2  import { View, Button, StyleSheet } from 'react-native';
3
4  const Toolbar = ({ onModeChange }) => {
5    return (
6      <View style={styles.toolbar}>
7        <Button title="Text" onPress={() => onModeChange('text')} />
8        <Button title="Image" onPress={() => onModeChange('image')} />
9        <Button title="Location" onPress={() => onModeChange('location')} />
10     </View>
11   );
12 };
13
14 const styles = StyleSheet.create({
15   toolbar: {
16     borderTopWidth: 1,
17     borderTopColor: 'rgba(0,0,0,0.04)',
18     backgroundColor: 'white',
19     flexDirection: 'row',
20     justifyContent: 'space-around',
21     padding: 10,
22   },
23 });
24
25 export default Toolbar;
```

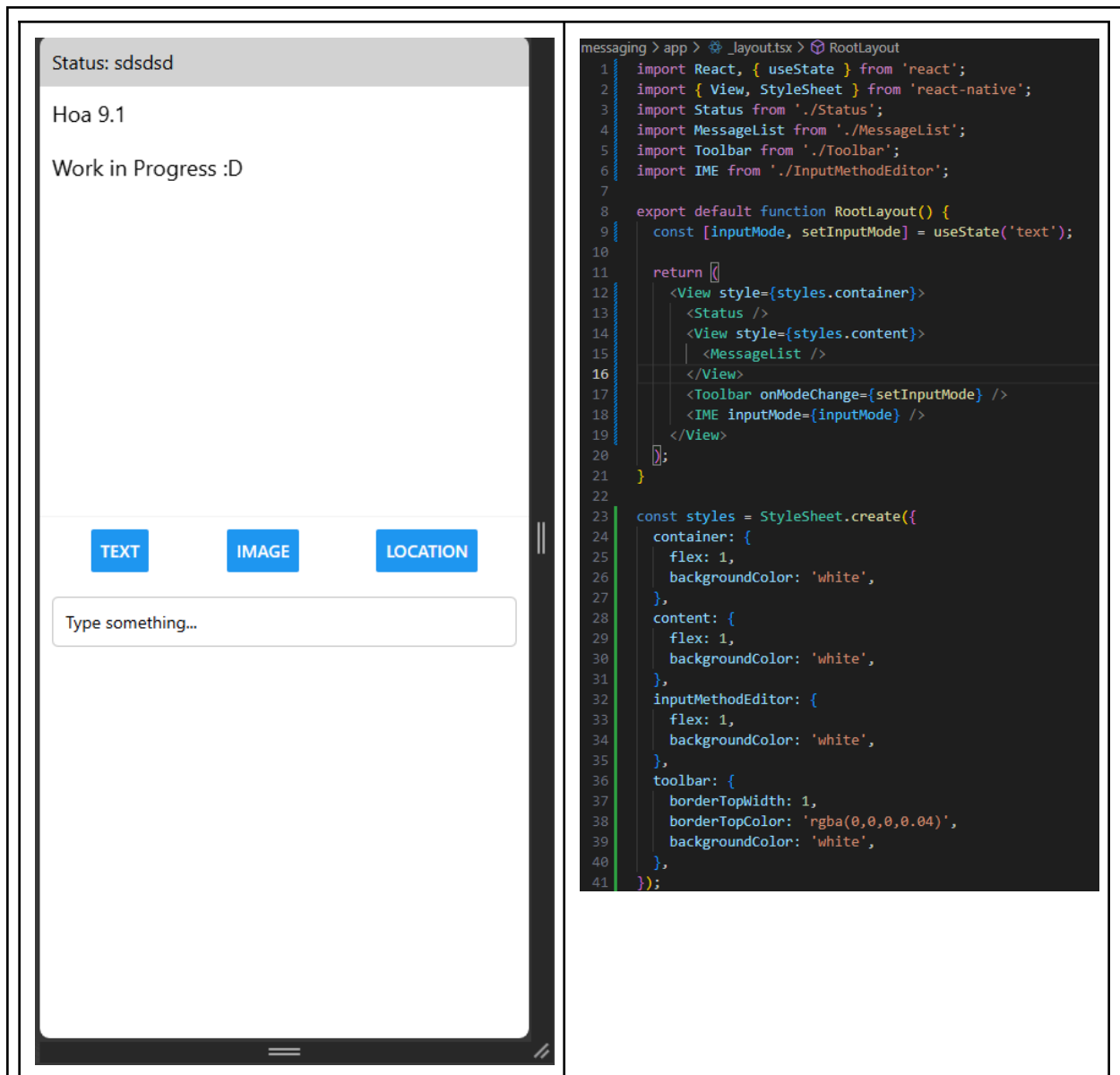
Input Method Editor (IME)

```

messaging > app > JS InputMethodEditor.js > [⌘] styles
1  import React, { useState } from 'react';
2  import { View, TextInput, StyleSheet } from 'react-native';
3
4  const IME = ({ inputMode }) => {
5    const [text, setText] = useState('');
6
7    return (
8      <View style={styles.inputMethodEditor}>
9        {inputMode === 'text' && (
10          <TextInput
11            style={styles.textInput}
12            placeholder="Type something..."
13            value={text}
14            onChangeText={setText}
15          />
16        )}
17      </View>
18    );
19  };
20
21  const styles = StyleSheet.create({
22    inputMethodEditor: {
23      flex: 1,
24      backgroundColor: 'white',
25      padding: 10,
26    },
27    textInput: {
28      borderColor: '#ccc',
29      borderWidth: 1,
30      padding: 10,
31      borderRadius: 5,
32    },
33  });
34
35  export default IME;

```

Layout.tsx



After some tinkering, my code went crazy so I was not able to monitor it step by step. Here is my final output so far..

messaging > app > JS Status.js > ...

```
1  import React, { useState, useEffect } from 'react';
2  import { StyleSheet, Text, View } from 'react-native';
3  import NetInfo from '@react-native-community/netinfo';
4
5  const Status = () => {
6    const [isConnected, setIsConnected] = useState(true);
7
8    useEffect(() => {
9      const unsubscribe = NetInfo.addEventListener(state => {
10        setIsConnected(state.isConnected);
11      });
12
13      return () => unsubscribe();
14    }, []);
15
16    return (
17      <View style={styles.container}>
18        <Text style={[styles.status, isConnected ? styles.online : styles.offline]}>
19          {isConnected ? 'Online' : 'Offline'}
20        </Text>
21      </View>
22    );
23  };
24
25  const styles = StyleSheet.create({
26    container: {
```

index

Online

EFA

Initializing the Project

```
terminal Help  ← →  messaging

indextsx
app > (tabs) > indextsx > App
1  import React from 'react';
2  import { StyleSheet, View } from 'react-native';
3
4  const App = () => {
5    return (
6      <View style={styles.container}>
7        <View style={styles.content}>
8          { /* ... */ }
9        </View>
10       <View style={styles.inputMethodEditor}>
11         { /* ... */ }
12       </View>
13       <View style={styles.toolbar}>
14         { /* ... */ }
15       </View>
16     </View>
17   );
18 };
19

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
node + - - - - -
PS C:\messaging> npx expo start
npx : The term 'npx' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ npx expo start
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (npx:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

PS C:\messaging> npx expo start
Starting project at C:\messaging
> Port 8081 is being used by another process
✓ Use port 8082 instead? ... yes
Starting Metro Bundler
```

The App

```
PS C:\messaging> npx expo install @react-native-community/netinfo
>>
> Installing 1 SDK 51.0.0 compatible native module using npm
> npm install

added 1 package, and audited 1583 packages in 6s

147 packages are looking for funding
  run `npm fund` for details

8 vulnerabilities (5 low, 3 moderate)
```


index.tsx 2 × JS Status.js

app > (tabs) > index.tsx > statusHeight

```
1  import { NetInfo, Platform, StatusBar, StyleSheet, View } from 'react-native';
2  import Constants from 'expo';
3  import React from 'react';
4
5  export default class Status extends React.Component {
6
7      state = {
8          info:null,
9      };
10
11      render() {
12          const { info } = this.state;
13
14          const isConnected = info !== 'none';
15          const backgroundColor = isConnected ? 'white' : 'red';
16
17          if(Platform.OS === 'ios'){
18
19              return
20              <View style={ [styles.status, { backgroundColor } ]}></View>
21
22          }
23          return null;
24      }
25  }
26
27
28
29  const statusHeight = (Platform.OS === 'ios' ? Constants.statusBarHeight : 0);
30
31  const styles = StyleSheet.create({
32      status: {
33          zIndex: 1,
34          height: statusHeight,
35      },
36  });
37
```

components > JS Status.js > ...

```
1 import { Constants } from 'expo'
2 import { StyleSheet } from 'react-native';
3
4 const statusBarHeight = (Platform.OS == 'ios' ? Constants.statusBarHeight : 0)
5
6 const styles = StyleSheet.create({
7   status: {
8     zIndex: 1,
9     height: statusBarHeight,
10  }
11
12
13
14 })
```

Using StatusBar


```
return (
  <View style={styles.container}>
    <StatusBar
      barStyle={isConnected ? 'dark-content' : 'light-content'} |
      backgroundColor={statusBarBackgroundColor}
      animated={false}
    />
  </View>
)
```


4:56



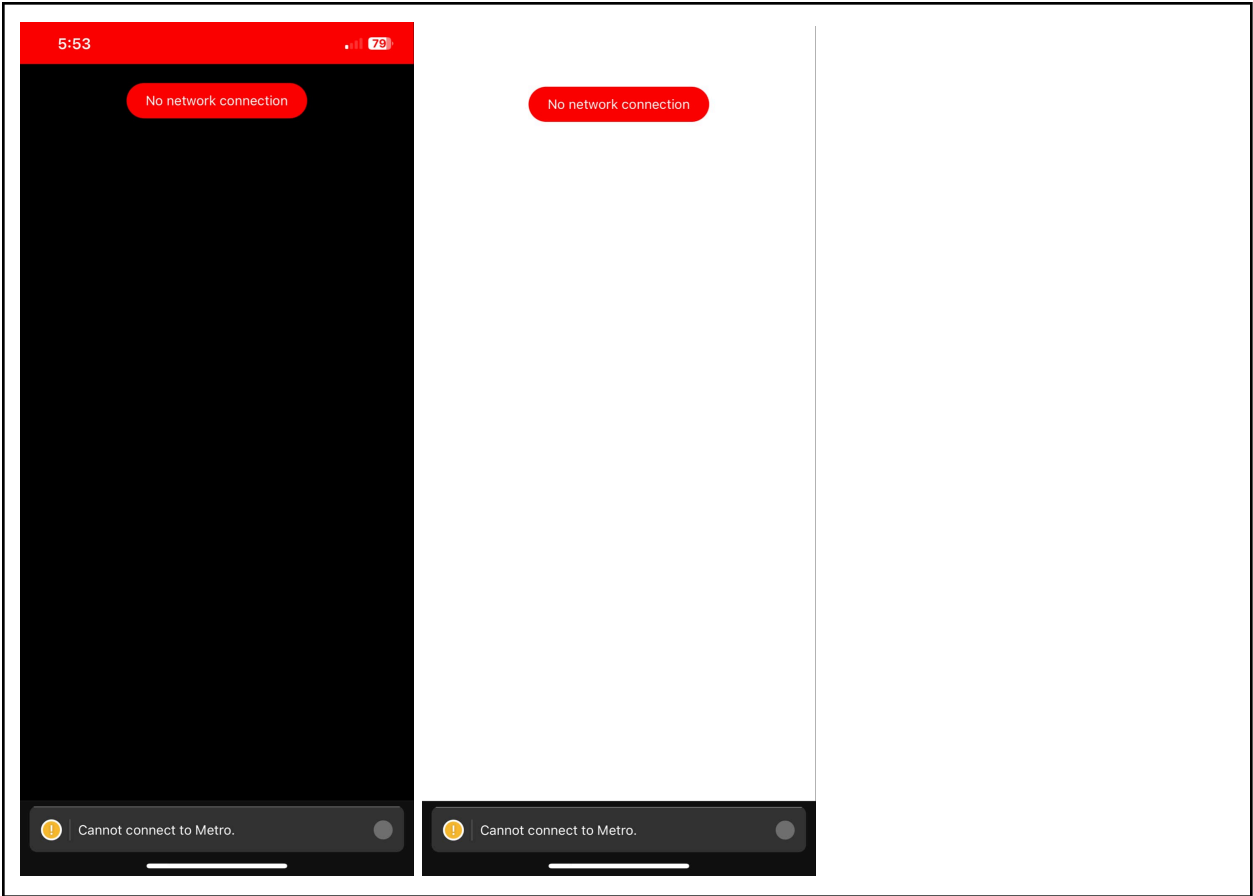
4:56



 Cannot connect to Metro.

 Cannot connect to Metro.

Message Bubble



```
render() {  
  const { info } = this.state;  
  const isConnected = info !== 'none';  
  const backgroundColor = isConnected ? 'white' : 'red';  
  
  const statusBar = (  
    <StatusBar  
      backgroundColor={backgroundColor}  
      barStyle={isConnected ? 'dark-content' : 'light-content'}  
      animated={false}  
    />  
  );  
  
  const messageContainer = (  
    <View style={styles.messageContainer} pointerEvents={'none'}>  
      {statusBar}  
      {!isConnected && (  
        <View style={styles.bubble}>  
          <Text style={styles.text}>No network connection</Text>  
        </View>  
      )}  
    </View>  
  );  
}
```

```

60
61 const styles = StyleSheet.create({
62   status: {
63     zIndex: 1,
64     height: statusHeight,
65   },
66   messageContainer: {
67     zIndex: 1,
68     position: 'absolute',
69     top: statusHeight + 20,
70     right: 0,
71     left: 0,
72     height: 80,
73     alignItems: 'center',
74   },
75   bubble: {
76     paddingHorizontal: 20,
77     paddingVertical: 10,
78     borderRadius: 20,
79     backgroundColor: 'red',
80   },
81   text: {
82     color: 'white',
83   },
84 });
85

```

7. Supplementary Activity

ILO2: Utilize core APIs to enable functionality within the application.

We have info in state, and we have logic to switch between showing a connected and disconnected UI in our render method. Now we need to update this state whenever network connectivity changes

We can do this using the NetInfo APIs. The NetInfo APIs are a good example of React Native core APIs: these provide a uniform interface to the lower level native APIs on iOS and Android. React Native is essentially providing JavaScript bindings and smoothing out platform differences for us.

We can call `NetInfo.getConnectionInfo()` to get the network connectivity status. `NetInfo.getConnectionInfo()` returns a promise which resolves to a string. If the device is connected, the string value will be 'wifi' or 'cellular' If the device isn't connected, the promise will still resolve, but with the value 'none'.

If we wanted to update our UI when the network connection changes, we could continuously poll `NetInfo.getConnectionInfo()` to get the network status – but this would be inefficient. Instead, we can add an event listener to `NetInfo`. `NetInfo` provides the method `addEventListener`, which we can call with a callback function, which it will invoke each time the network status changes.

Here's an example of using `NetInfo.addEventListener`

```
const subscription = NetInfo.addEventListener('connectionChange', (status) => {  
  console.log('Network status changed', status)  
});
```

This example would log a new status each time the network connectivity changes. We can call `subscription.remove()` when we want to stop listening for changes - most of the time, we'll do this when our component unmounts. For our app, we'll use both `NetInfo.getConnectionInfo` and `NetInfo.addEventListener`. First we'll call `NetInfo.getConnectionInfo` when the `Status` component mounts to get the initial network connectivity. Then we'll use `NetInfo.addEventListener` to update our UI when a change occurs.

Tasks:

- Modify `Status.js` so that you can use a Core API (such as `NetInfo`) to retrieve connection information
- Test the connection: It must not display any message when there is a connection and vice-versa.

Hint: Check documentation for `NetInfo` API to verify functions associated and check sample code.

ALFEROS

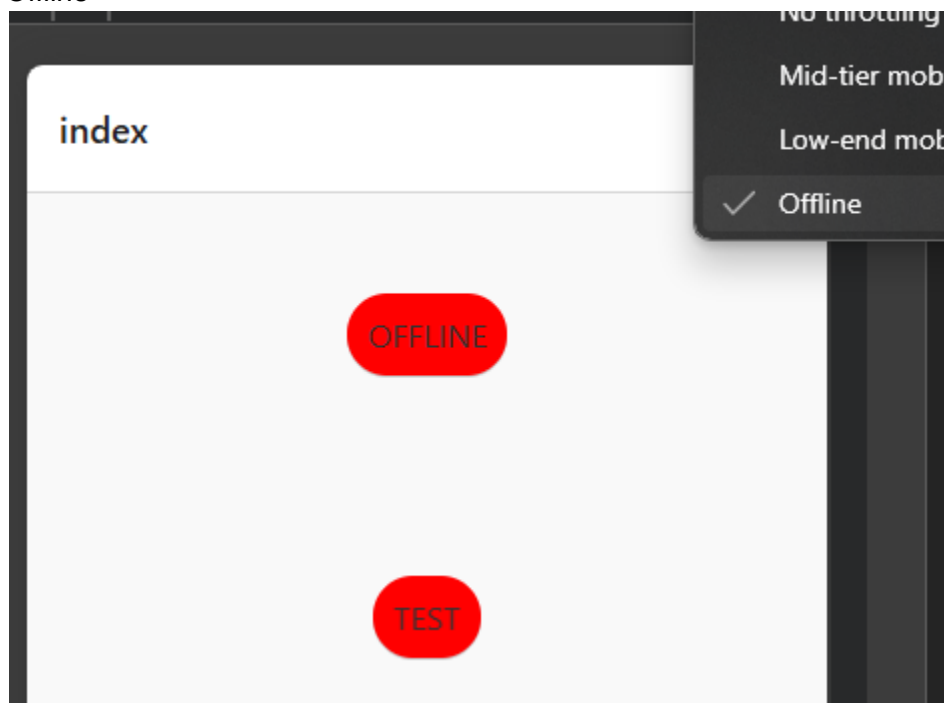
```
// SUPPLE
useEffect(() => {
  const unsubscribe = NetInfo.addEventListener(state => {
    setIsOnline(state.isConnected);
  });

  return () => unsubscribe();
}, []);

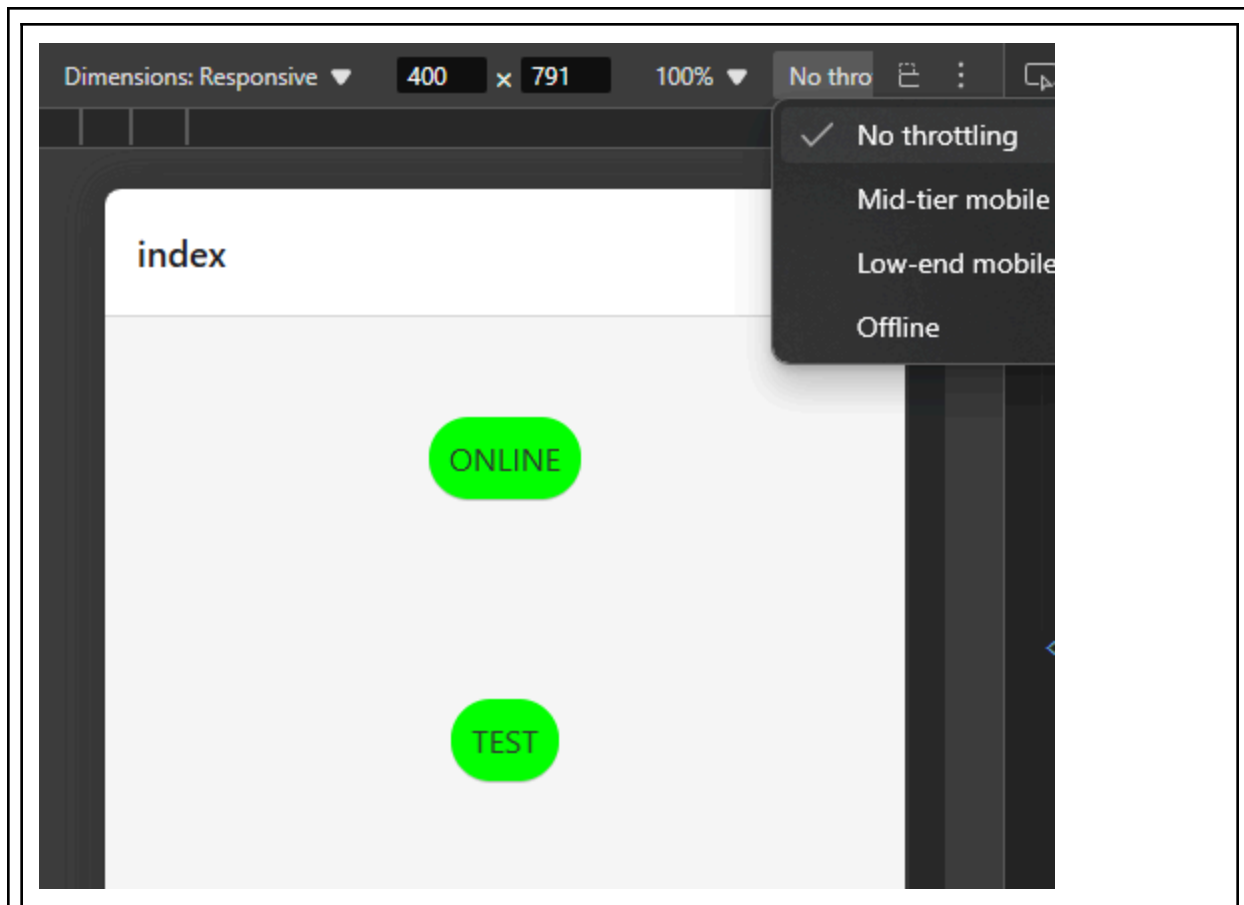
const handleSendMessage = () => {
  if (message.trim() !== '') {
    setMessages((prevMessages) => [...prevMessages, message]);
    setMessage('');
  }
};

const networkStatusMessage = isOnline ? "ONLINE" : "OFFLINE";
```

Offline



Online



EFA

```
app > (tabs) > index.tsx > styles > text
1  import NetInfo from '@react-native-community/netinfo';
2  import Constants from 'expo-constants';
3  import { Platform, StatusBar, StyleSheet, Text, View } from 'react-native';
4  import React from 'react';
5
6  export default class Status extends React.Component {
7    state = {
8      isConnected: true, // Assume connected initially
9    };
10
11    componentDidMount() {
12      NetInfo.fetch().then(state => {
13        this.setState({ isConnected: state.isConnected });
14      });
15
16      this.unsubscribe = NetInfo.addEventListener(state => {
17        this.setState({ isConnected: state.isConnected });
18      });
19    }
20
21    componentWillUnmount() {
22      if (this.unsubscribe) this.unsubscribe();
23    }
24  }
```

```

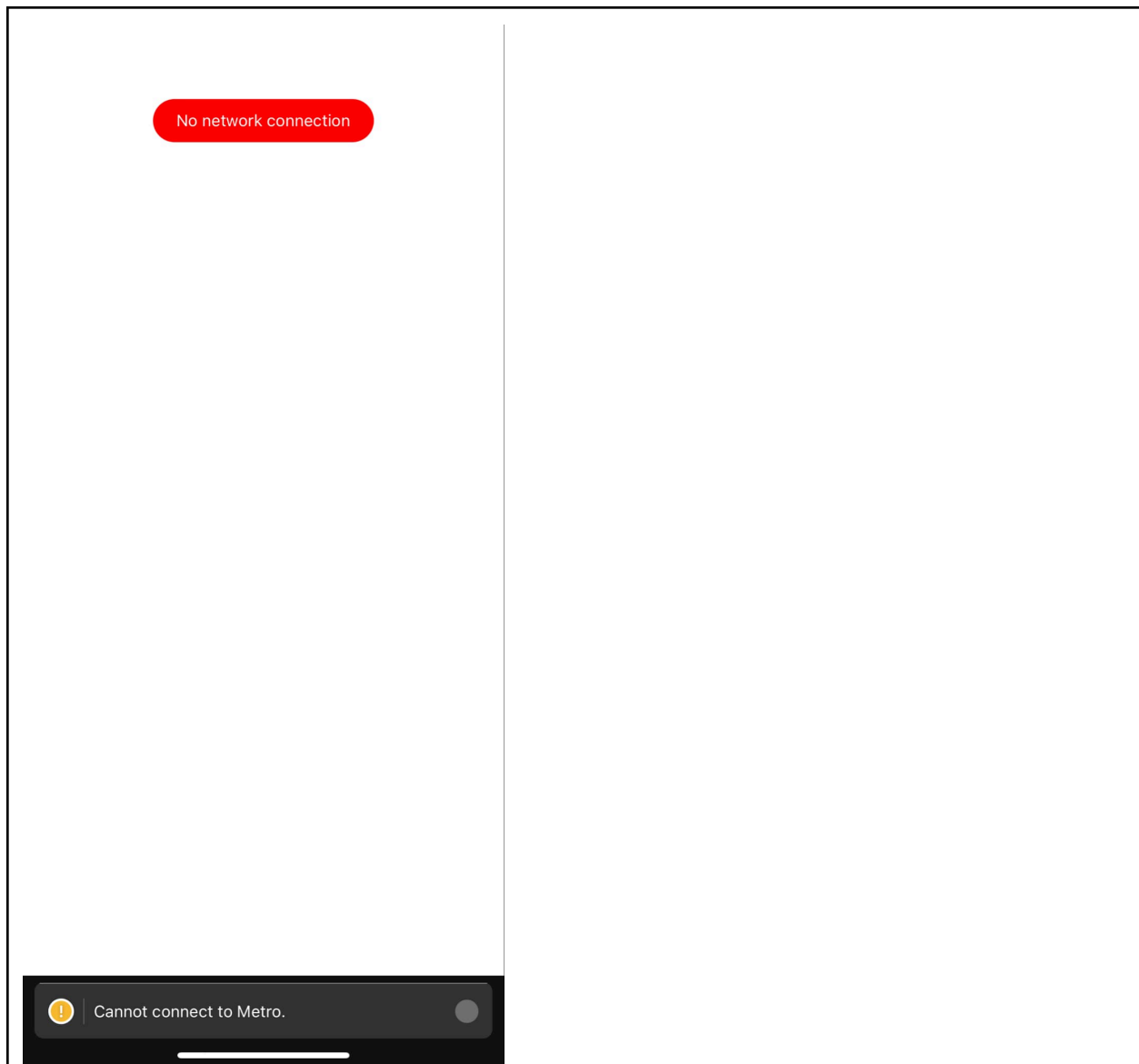
6   export default class Status extends React.Component {
25   render() {
35
36     const messageContainer = !isConnected ? (
37       <View style={styles.messageContainer} pointerEvents="none">
38         <View style={styles.bubble}>
39           <Text style={styles.text}>No network connection</Text>
40         </View>
41       </View>
42     ) : null;
43
44     if (Platform.OS === 'ios') {
45       return (
46         <View style={styles.container}>
47           <View style={styles.status}>
48             {statusBar}
49             {messageContainer}
50           </View>
51         </View>
52       );
53     }
54
55     return (
56       <View style={styles.container}>
57         {statusBar}
58         {messageContainer}
59       </View>
60     );

```

```

65
66   const styles = StyleSheet.create({
67     container: {
68       flex: 1,
69       backgroundColor: 'white',
70     },
71     status: {
72       zIndex: 1,
73       height: statusHeight,
74       backgroundColor: 'white',
75     },
76     messageContainer: {
77       zIndex: 1,
78       position: 'absolute',
79       top: statusHeight + 20,
80       right: 0,
81       left: 0,
82       height: 80,
83       alignItems: 'center',
84     },
85     bubble: {
86       paddingHorizontal: 20,
87       paddingVertical: 10,
88       borderRadius: 20,
89       backgroundColor: 'red',
90     },
91     text: {
92       color: 'white',

```



8. Conclusion

ALFEROS

- In this activity, we explored the use of React Native's core API, specifically the NetInfo API, to manage network connectivity within a messaging application. By implementing features that detect online and offline statuses, we enhanced the user experience, allowing for real-time updates in the UI based on network changes. I had a hard time following the procedure, since the code provided in there are depreciated, so I search for their updated alternatives. Also, I encountered an error, where I was doing smoothly at the start of the procedure, but suddenly my components and main index went bonkers, so I need to recreate another expo and redo all of my work. This lead to my lack of screenshots on the procedure part. Overall, this activity reinforced our knowledge in leveraging core APIs to create applications that uses connectivity status.

EFA

- ## 9. Assessment Rubric

T.I.P. SO 7										
Criteria	Ratings									Pts
<div><div>T.I.P. SO 7.1</div><div>Acquire and apply new knowledge from outside sources</div><div>threshold 4.2 pts</div></div>	6 pts [Excellent] Educational interests and pursuits exist and flourish outside classroom requirements;knowledge and/or experiences are pursued independently and applies knowledge learned into practice	5 pts [Good] Educational interests and pursuits exist and flourish outside classroom requirements;knowledge and/or experiences are pursued independently	4 pts [Satisfactory] Look beyond classroom requirements, showing interest in pursuing knowledge independently	3 pts [Unsatisfactory] Begins to look beyond classroom requirements, showing interest in pursuing knowledge independently	2 pts [Poor] Relies on classroom instruction only	1 pts [Very Poor] No initiative or interest in acquiring new knowledge	6 pts			
<div><div>T.I.P. SO 7.2</div><div>Learn independently</div><div>threshold 4.2 pts</div></div>	6 pts [Excellent] Completes an assigned task independently and practices continuous improvement	5 pts [Good] Completes an assigned task without supervision or guidance	4 pts [Satisfactory] Requires minimal guidance to complete an assigned task	3 pts [Unsatisfactory] Requires detailed or step-by-step instructions to complete a task	2 pts [Poor] Shows little interest to complete a task independently	1 pts [Very Poor] No interest to complete a task independently	6 pts			
<div><div>T.I.P. SO 7.3</div><div>Critical thinking in the broadest context of technological change</div><div>threshold 4.2 pts</div></div>	6 pts [Excellent] Synthesizes and integrates information from a variety of sources; formulates a clear and precise perspective; draws appropriate conclusions	5 pts [Good] Evaluate information from a variety of sources; formulates a clear and precise perspective.	4 pts [Satisfactory] Analyze information from a variety of sources; formulates a clear and precise perspective.	3 pts [Unsatisfactory] Apply the gathered information to formulate the problem	2 pts [Poor] Gather and summarize the information from a variety of sources but failed to formulate the problem	1 pts [Very Poor] Gather information from a variety of sources	6 pts			
<div><div>T.I.P. SO 7.4</div><div>Creativity and adaptability to new and emerging technologies</div><div>threshold 4.2 pts</div></div>	6 pts [Excellent] Ideas are combined in original and creative ways in line with the new and emerging technology trends to solve a problem or address an issue.	5 pts [Good] Ideas are creative and adapt the new knowledge to solve a problem or address an issue	4 pts [Satisfactory] Ideas are creative in solving a problem, or address an issue	3 pts [Unsatisfactory] Shows some creative ways to solve the problem	2 pts [Poor] Shows initiative and attempt to develop creative ideas to solve the problem	1 pts [Very Poor] Ideas are copied or related from the sources consulted	6 pts			
Total Points: 24										