

Mobile Device Programming

CHAPTER 6 : CALL APIs

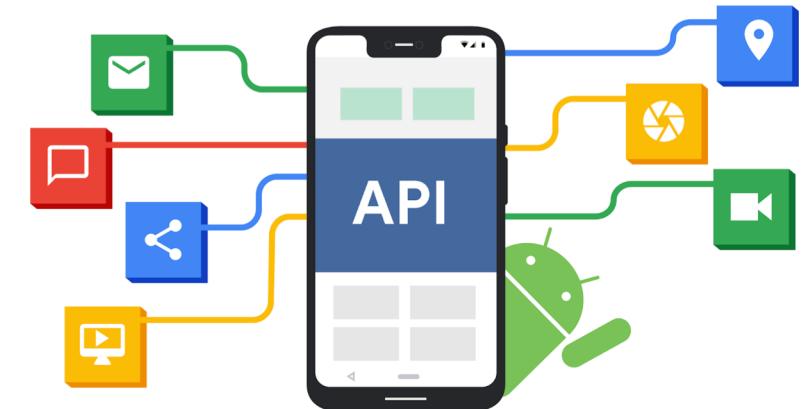
Application Program Interface (APIs)

- ❖ คือ คำสั่ง (Code) ที่อนุญาตให้ software program สามารถสื่อสารระหว่างกันได้ ถ้าจะพูดในภาษาคนเขียน program แล้ว API เป็นช่องทางสำหรับขอใช้บริการคำสั่ง จาก operation system (OS) หรือ application อื่น ๆ ซึ่งมันใช้งานโดยติดตั้ง function และเรียกใช้งานตาม document ที่เขียนไว้



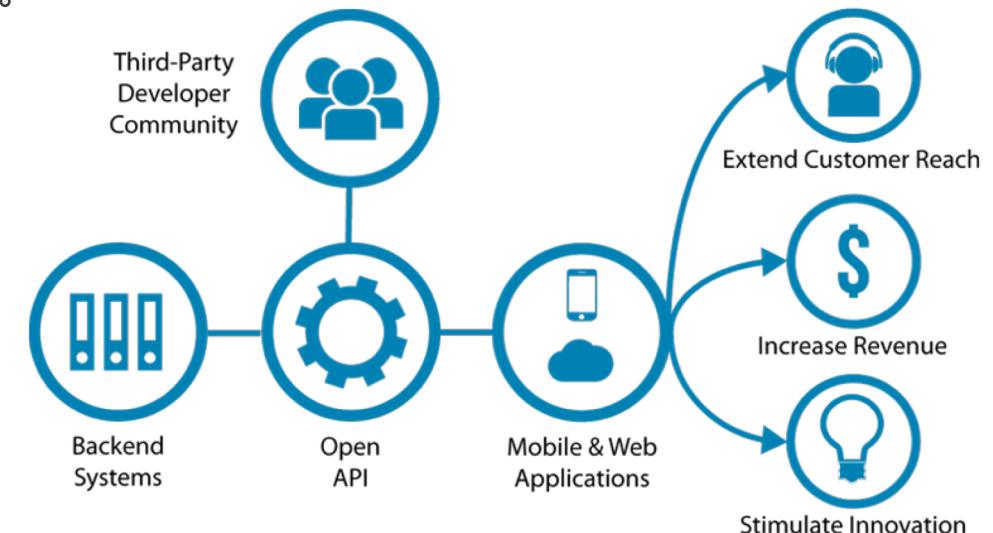
ส่วนประกอบของ APIs

- ❖ ข้อกำหนดที่จะอธิบายการแลกเปลี่ยนข้อมูลระหว่าง program ซึ่งทำออกมาในลักษณะ document เพื่อบอกว่า request/response ต้องเป็นอย่างไร
- ❖ Software ที่เขียนขึ้นตามข้อกำหนด และทำการเผยแพร่ออกไปให้ใช้งาน



หลักการใช้งาน APIs

- ❖ API ถูกใช้งานใน application เพื่อสื่อสารกับ user โดยไม่จำเป็นต้องมีความรู้ บริษัทใหญ่ ๆ หลายบริษัทมีการเปิด API ให้ภายนอกเข้ามาใช้งาน เช่น Facebook, google, twitter ผู้พัฒนาระบบที่สนใจ สามารถนำเอา API เหล่านี้ไปต่อยอด ซึ่งทางบริษัทก็สามารถขยายฐานลูกค้า ออกไปได้อีก รูปแบบการนำเอา API ไปใช้งานมีดังนี้



ตัวอย่าง APIs ที่ยอดนิยม

- ❖ Google Maps API
- ❖ YouTube APIs
- ❖ Flickr API
- ❖ Twitter APIs
- ❖ Amazon Product Advertising API



หลักการพื้นฐานการใช้งาน APIs

- ❖ JavaScript และ ES6
- ❖ fetch

First Step

- ❖ Create States
- ❖ Boolean state
- ❖ array

```
constructor(props) {  
    super(props);  
    this.state = {  
        loading: true,  
        dataSource: []  
    };  
}
```

Second Step

- ❖ Use componentDidMount method
- ❖ Use Fetch เพื่อเรียก URL ที่ต้องการใช้งานข้อมูล และ APIs

```
componentDidMount(){
    fetch("https://jsonplaceholder.typicode.com/users")
    .then(response => response.json())
    .then((responseJson)=> {
        this.setState({
            loading: false,
            dataSource: responseJson
        })
    })
    .catch(error=>console.log(error)) //to catch the
errors if any
}
```

Final Step

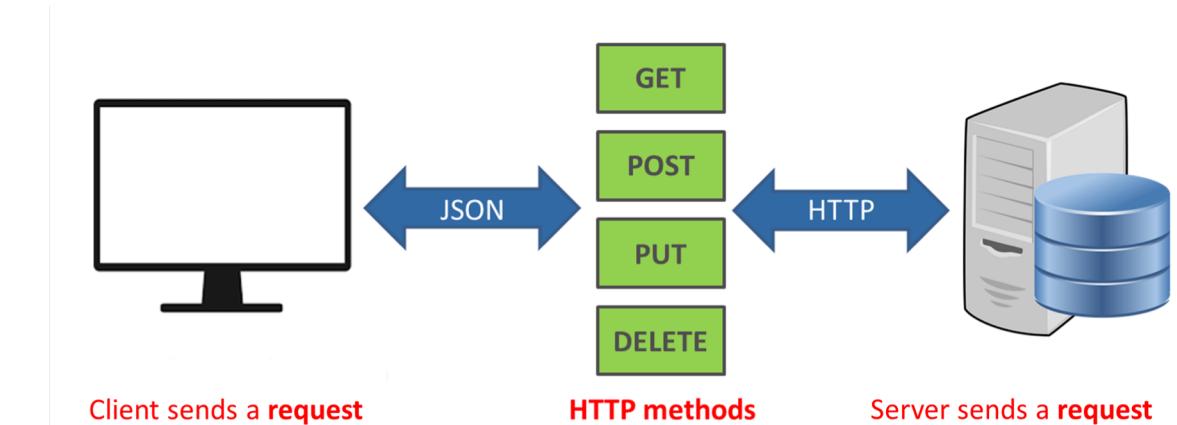
- ❖ Use FlatList component
- ❖ implement in render function

```
render(){
  if(this.state.loading){
    return(
      <View style={styles.loader}>
        <ActivityIndicator size="large" color="#0c9"/>
      </View>
    )
    return(
      <View style={styles.container}>
        <FlatList
          data= {this.state.dataSource}
          ItemSeparatorComponent = {this.FlatListItemSeparator}

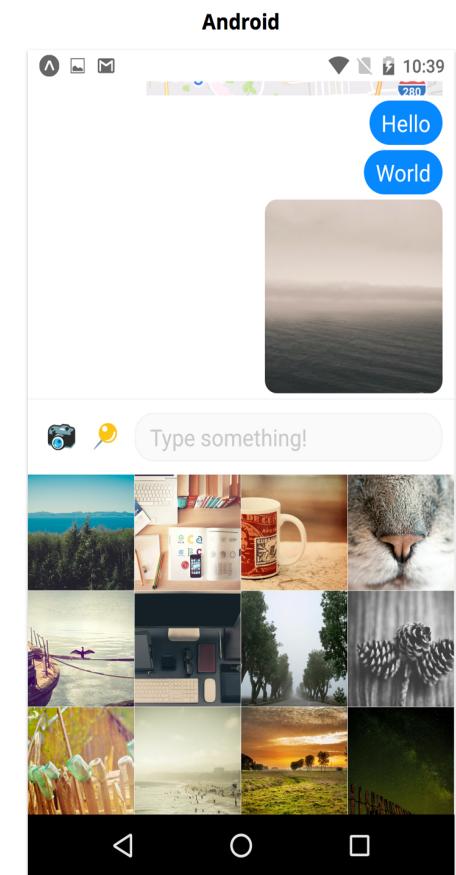
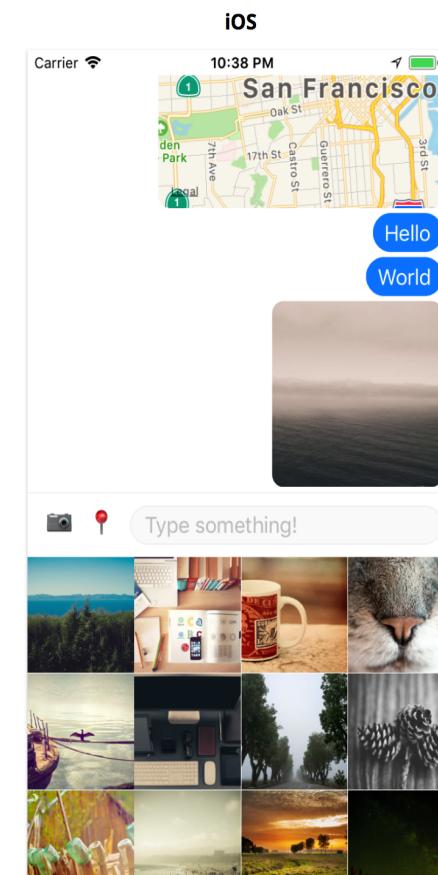
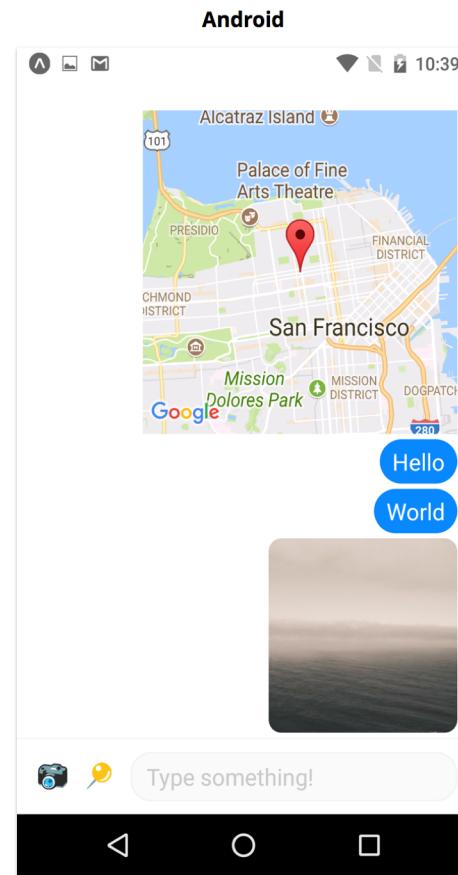
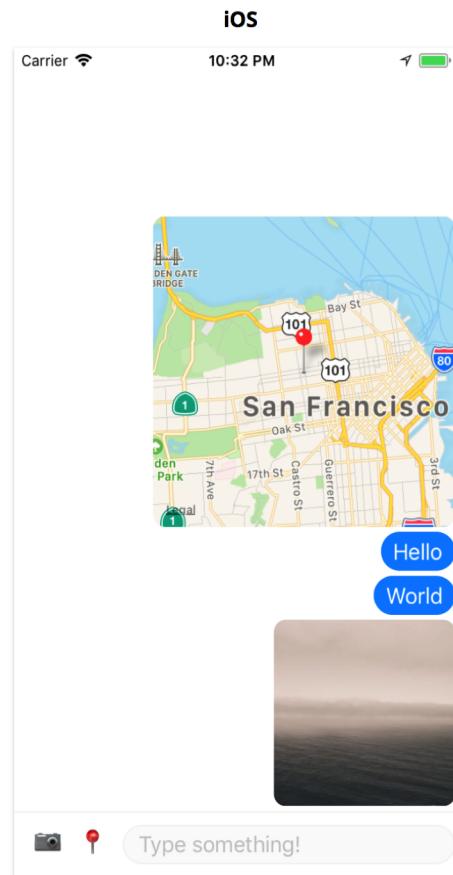
          renderItem= {item=> this.renderItem(item)}
          keyExtractor= {item=>item.id.toString()}
        />
      </View>
    )
  }
}
```

Networking

- ❖ Fetch API
- ❖ XMLHttpRequest
- ❖ WebSocket (full-duplex)
- ❖ Making requests



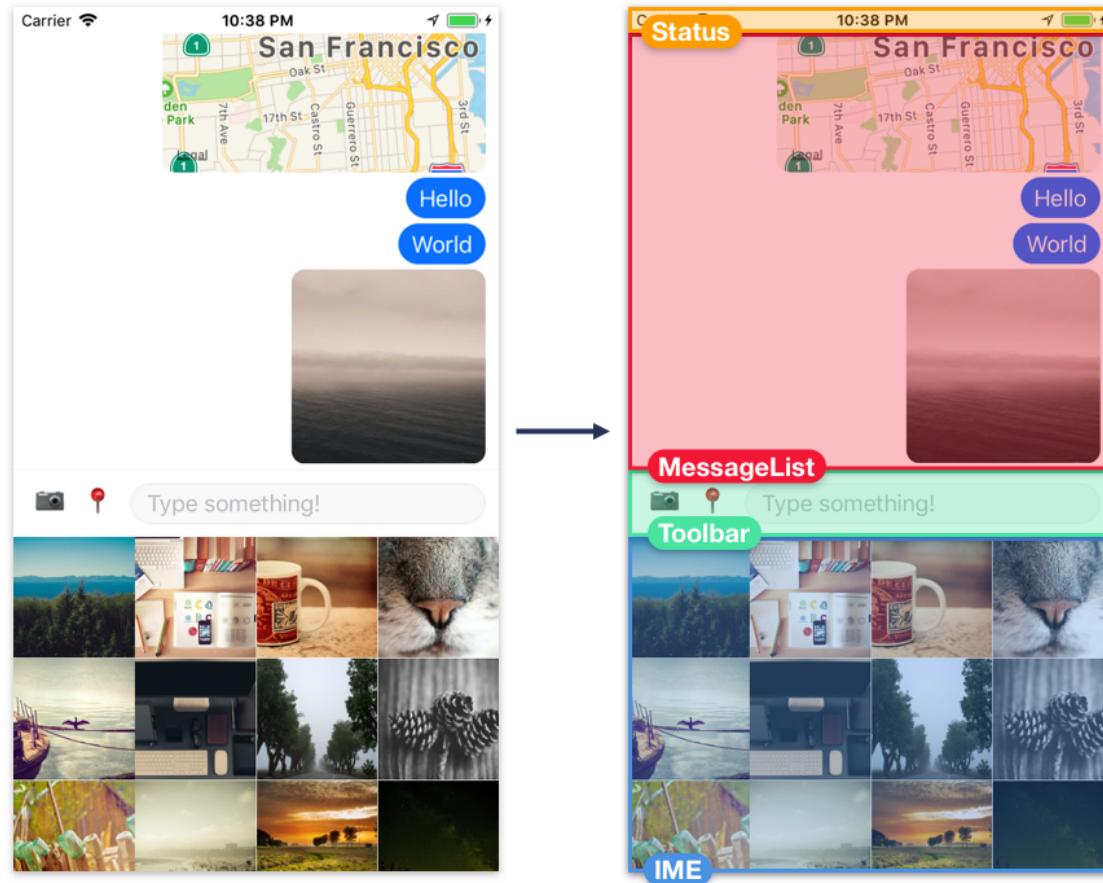
Example : Messages Application



APIs in Messages Application

- ❖ **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 this later)
- ❖ **StatusBar** - Controls the visibility and color of the status bar

โครงสร้างของ Application



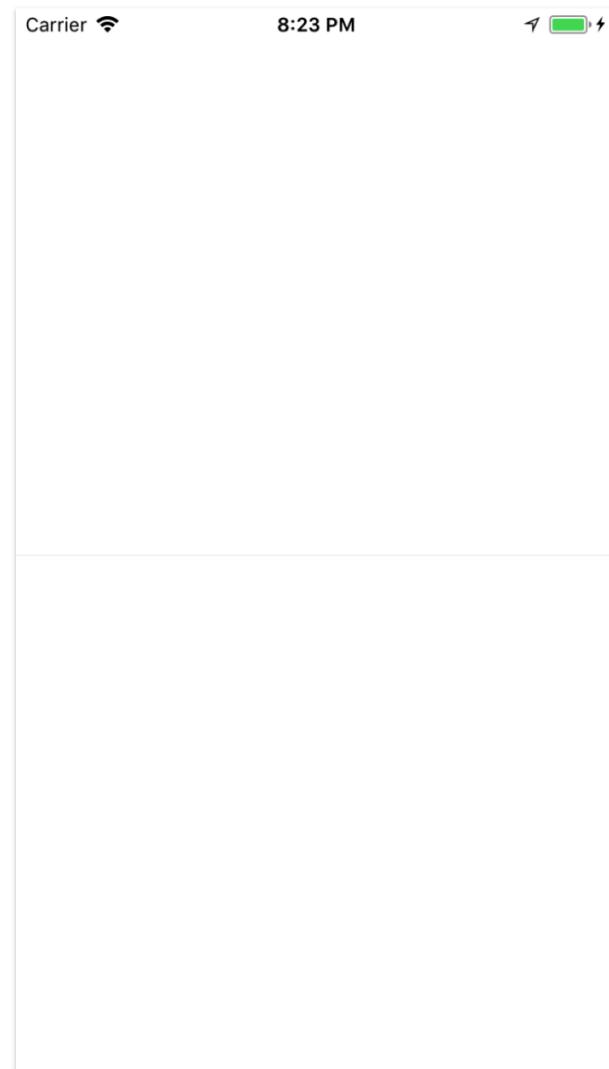
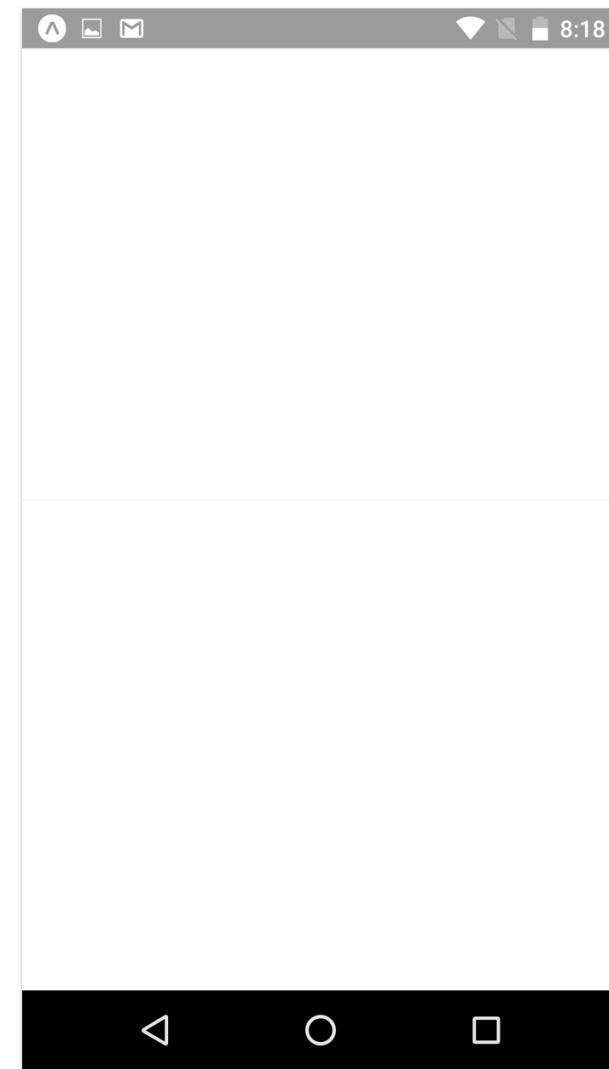
- ❖ **Status** ทำการแสดงข้อมูลพื้นฐานของแอพพิเคชัน เช่น เวลา แบตเตอรี่ ซึ่งจะแสดงในแนวนอน ตำแหน่งบนสุดของจอกาฟ แต่ในกรณีนี้จะแสดงสถานะการเชื่อมต่อของเน็ตเวิร์กด้วย
- ❖ **MessageList** ทำหน้าที่ render ข้อความ รูปภาพและแผนที่ที่ส่งเข้ามาในแอพพิเคชัน
- ❖ **Toolbar** ทำหน้าที่สลับการทำงานระหว่าง การส่งข้อความ รูป ตำแหน่งของอุปกรณ์ ขณะที่ผู้ใช้กำลังใช้งานแอพพิเคชัน ในระหว่างที่ทำการพิมพ์ข้อความได้
- ❖ **Input Method Editor (IME)** ทำหน้าที่ render พวก Input method ต่าง ๆ เช่น imageGrid เพื่อทำการจัดการ image component หรือหากผู้ใช้เรียกใช้คีย์บอร์ด แอพพิเคชันจะทำการจัดการแสดงหรือไม่แสดงคีย์บอร์ดได้

Messaging/App.js

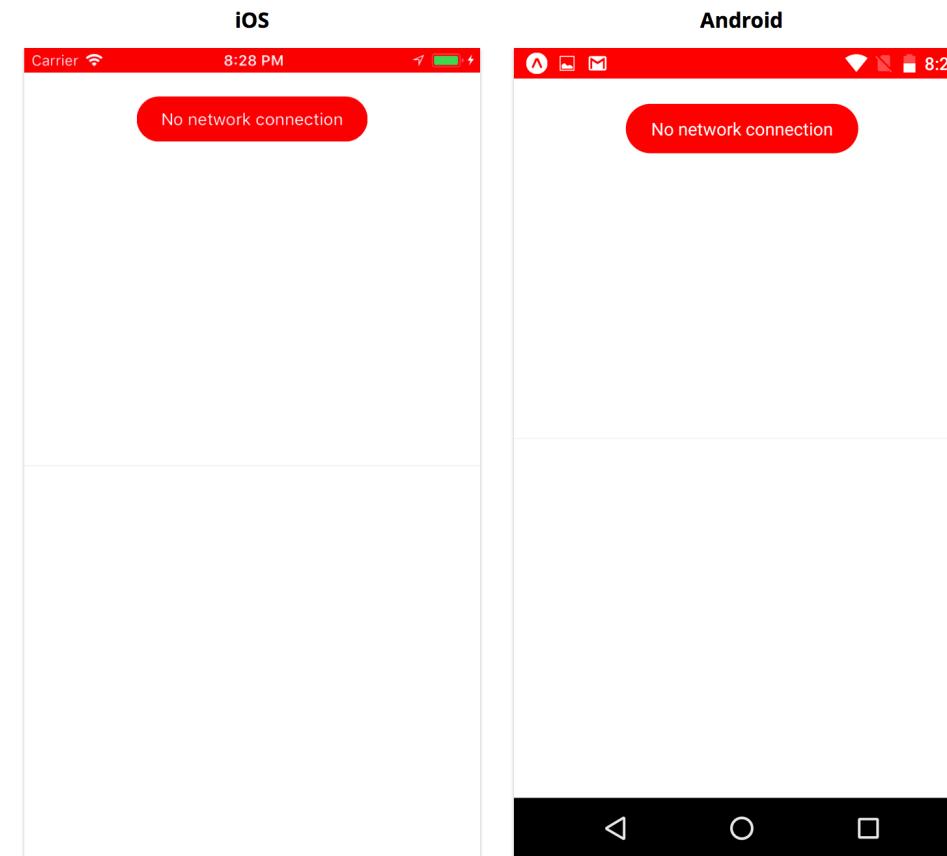
```
import { StyleSheet, View } from 'react-native';
import React from 'react';
export default class App extends React.Component {
  renderMessageList() {
    return (
      <View style={styles.content}></View>
    );
  }
  renderInputMethodEditor() {
    return (
      <View style={styles.inputMethodEditor}></View>
    );
  }
  renderToolbar() {
    return (
      <View style={styles.toolbar}></View>
    );
  }
  render() {
    return (
      <View style={styles.container}>
        {this.renderMessageList()}
        {this.renderToolbar()}
        {this.renderInputMethodEditor()}
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'white',
  },
  content: {
    flex: 1,
    backgroundColor: 'white',
  },
  inputMethodEditor: {
    flex: 1,
    backgroundColor: 'white',
  },
  toolbar: {
    borderTopWidth: 1,
    borderTopColor: 'rgba(0,0,0,0.04)',
    backgroundColor: 'white', }, })
});
```



iOS**Android**

Network connectivity indicator



StatusBar API : Component to control the app status bar.

❖ Props

- ❖ animated
- ❖ backgroundColor (**Android**)
- ❖ barStyle (**Android**) default = enum('default',
'light-content', 'dark-content')
- ❖ hidden
- ❖ networkActivityIndicatorVisible (**iOS**)
- ❖ showHideTransition
- ❖ translucent

❖ Methods

- ❖ proStackEntry()
- ❖ pushStackEntry()
- ❖ replaceStackEntry()
- ❖ setBackgroundColor() (**Android**)
- ❖ setBarStyle()
- ❖ setHidden()
- ❖ setNetworkActivityIndicatorVisible() (**iOS**)
- ❖ setTranslucent() (**Android**)

messaging/app.json

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

Status Styles

- ❖ background styles -> create view
- ❖ two visual states :
 - ❖ connect to network
 - ❖ disconnected

messaging/components/Status.js

```
import Constants from 'expo-constants';
import { StyleSheet } from 'react-native';
// ...
const statusHeight =
(Platform.OS === 'ios' ? Constants.statusBarHeight : 0);
const styles = StyleSheet.create({
  status: {
    zIndex: 1,
    height: statusHeight,
  },
  // ...
});
```

messaging/components/Status.js

```
import Constants from 'expo-constants';
import NetInfo from '@react-native-community/netinfo';
import {
Platform,
StatusBar,
StyleSheet,
Text,
View,
} from 'react-native';
import React from 'react';
export default class Status extends React.Component {
state = {
isConnected: null,
};
// ...
```

```
render() {
const { isConnected } = this.state;
const backgroundColor = isConnected ? 'white' : 'red';
if (Platform.OS === 'ios') {
return (
<View style={[styles.status, { backgroundColor }]}></View>
);
}
return null; // Temporary!
}
}
// ...
```

```
messaging/App.js
// ...
import Status from './components/Status';
export default class App extends React.Component {
// ...
render() {
  return (
    <View style={styles.container}>
      <Status />
      {this.renderMessageList()}
      {this.renderToolbar()}
      {this.renderInputMethodEditor()}
    </View>
  );
}
// ...
}
// ...
```

iOS

Carrier  8:23 PM   

Android

     8:20

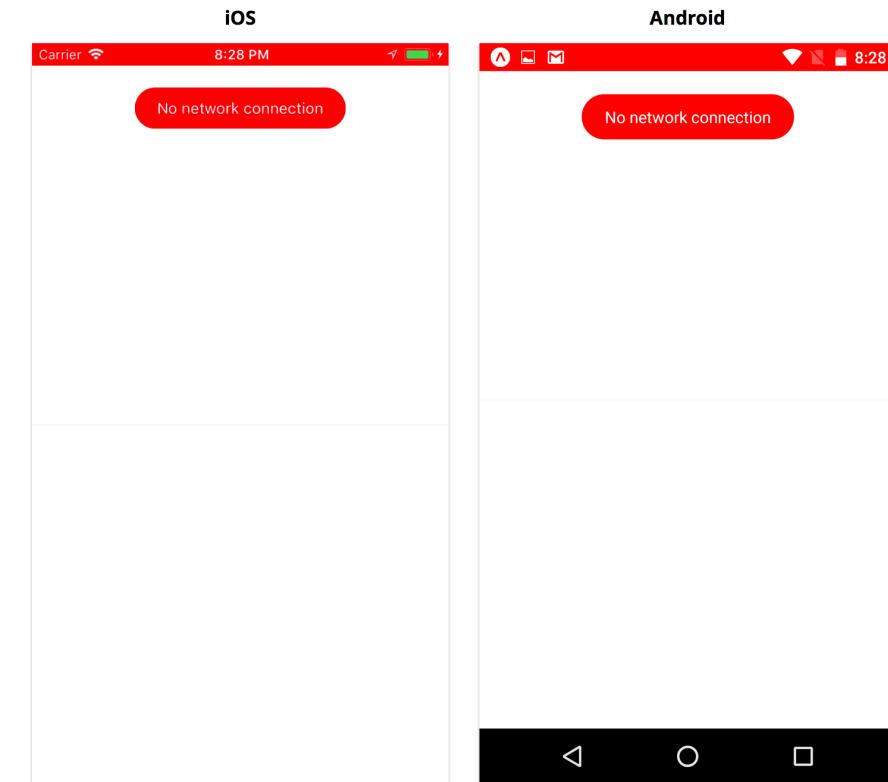
messaging/components/Status.js

```
import Constants from 'expo-constants';
import { StatusBar, StyleSheet, View } from 'react-native';
import React from 'react';
export default class Status extends React.Component {
state = {
isConnected: true,
};
// ...
render() {
const { isConnected } = this.state;
const backgroundColor = isConnected ? 'white' : 'red';
}
```

```
const statusBar = (
<StatusBar
backgroundColor={backgroundColor}
barStyle={isConnected ? 'dark-content' : 'light-content'}
animated={false}
/>
);
if (Platform.OS === 'ios') {
return (
<View style={[styles.status, { backgroundColor }]}>
{messageContainer}
</View>
);
}
return null; // Temporary!
}
}
```

Message bubble

- ❖ the red status bar alone doesn't indicate anything about network connectivity
- ❖ short message in a floating bubble at the top of the screen.



messaging/components/Status.js

```
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;
}
```

```
const styles = StyleSheet.create({
  // ...
  messageContainer: {
    zIndex: 1,
    position: 'absolute',
    top: statusHeight + 20,
    right: 0,
    left: 0,
    height: 80,
    alignItems: 'center',
  },
  bubble: {
    paddingHorizontal: 20,
    paddingVertical: 10,
    borderRadius: 20,
    backgroundColor: 'red',
  },
  text: {
    color: 'white',
  },
});
```

NetInfo

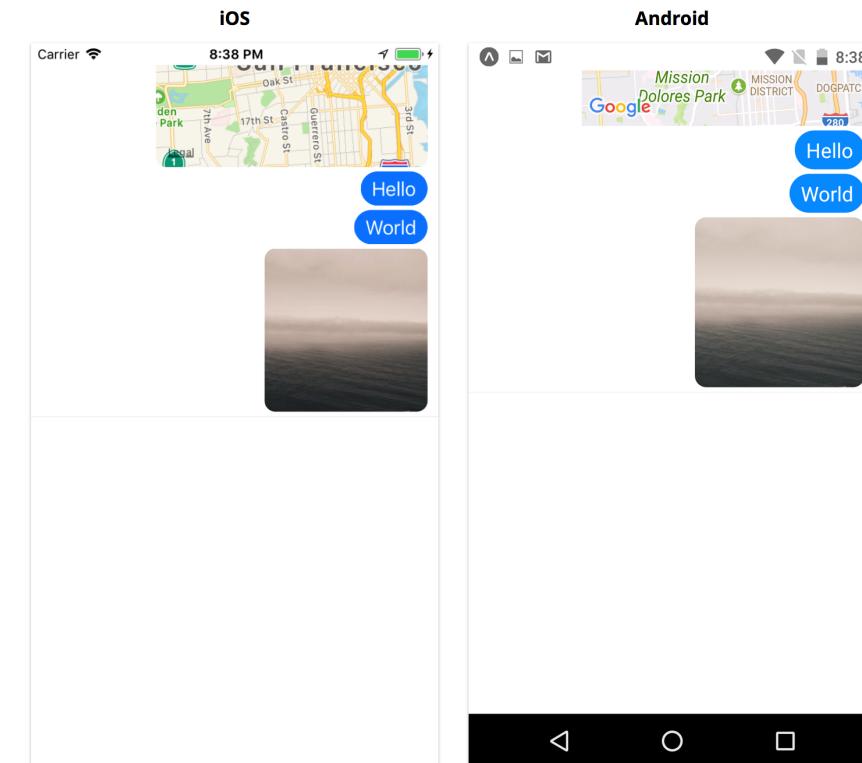
- ❖ isConnected in state
- ❖ Call NetInfo.fetch() -> connected = true
- ❖ Handle -> event listener to NetInfo.NetInfo
- ❖ Call addEventListener

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

```
// ...  
  
async componentDidMount() {  
    this.subscription =  
    NetInfo.addEventListener(this.handleChange);  
  
    const { isConnected } = await NetInfo.fetch();  
  
    this.setState({ isConnected });  
}  
  
componentWillUnmount() {  
    this.subscription()  
}  
  
handleChange = ({ isConnected }) => {  
    this.setState({ isConnected });  
};  
  
//
```

The message list : MessageUtils

- ❖ The message list will display a vertically scrolling list of text messages, image messages, and location messages.
- ❖ Use FlatList component
 - ❖ create ./utils/MessageUtils.js
 - ❖ Use PropTypes.shape
- ❖ Final, exporting creatTextMessage, CreateImageMessage, and createLocataionMessage -> FlatList



messaging/utils/MessageUtils.js

```
import PropTypes from 'prop-types';
```

```
export const MessageShape = PropTypes.shape({
    id: PropTypes.number.isRequired,
    type: PropTypes.oneOf(['text', 'image', 'location']),
    text: PropTypes.string,
    uri: PropTypes.string,
    coordinate: PropTypes.shape({
        latitude: PropTypes.number.isRequired,
        longitude: PropTypes.number.isRequired,
    }),
});
```

messaging/utils/MessageUtils.js

```
let messageId = 0;

function getNextId() {
    messageId += 1;
    return messageId;
}

export function createTextMessage(text) {
    return {
        type: 'text',
        id: getNextId(),
        text,
    };
}
```

```
export function createImageMessage(uri) {
    return {
        type: 'image',
        id: getNextId(),
        uri,
    };
}
```

```
export function createLocationMessage(coordinate) {
    return {
        type: 'location',
        id: getNextId(),
        coordinate,
    };
}
```

The message list : MessageList (incomplete)

- ❖ Create MessageList.js
- ❖ Add to App.js

```
messaging/components/MessageList.js
import React from 'react';
import PropTypes from 'prop-types';

import { MessageShape } from '../utils/MessageUtils';
const keyExtractor = item => item.id.toString();

export default class MessageList extends React.Component {
    static propTypes = {
        messages: PropTypes.arrayOf(MessageShape).isRequired,
        onPressMessage: PropTypes.func,
    };

    static defaultProps = {
        onPressMessage: () => {},
    };
    // ...
}
```

Alert API

- ❖ use the Alert.alert API to present the user with a native dialog window for making this choice.
- ❖ present a dialog with two choices: delete and cancel.
- ❖ Methods (Alert.alert(title, message?, buttons?, options?, type?))
 - ❖ title
 - ❖ message
 - ❖ buttons -> iOS (default, cancel, or destructive)
 - ❖ options
 - ❖ type ->iOS (default, plain-text, secure-text, or login-password)