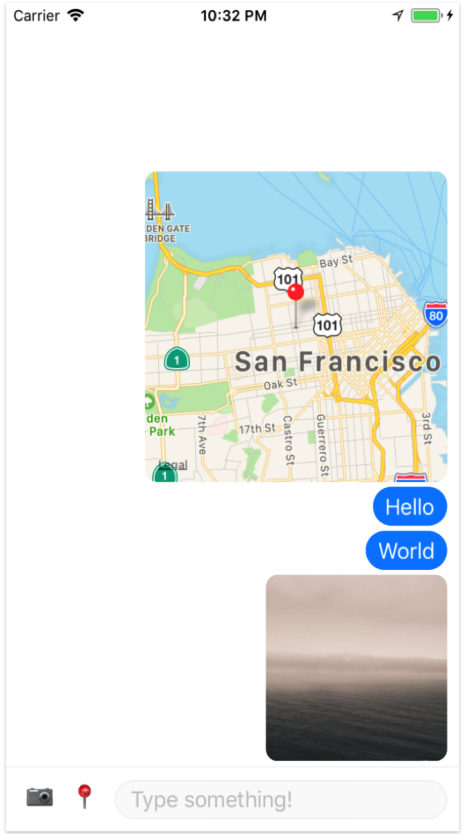
Mobile Device Programming

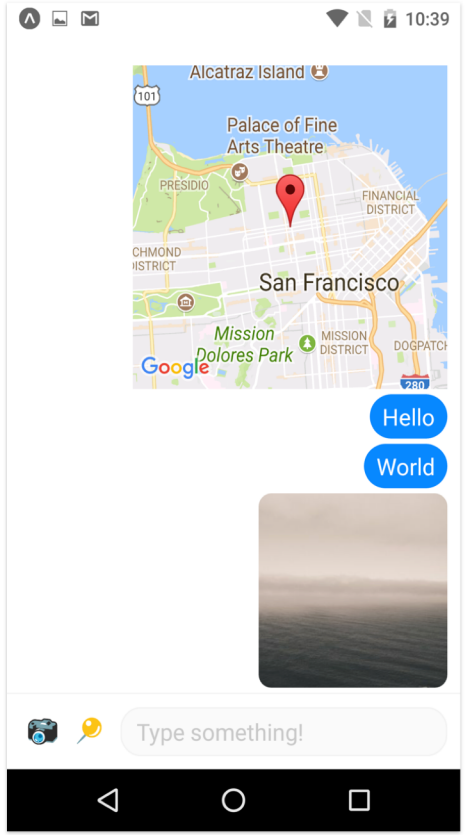
CHAPTER 7 : CALL APIs (Part 2)

Example : Messages Application

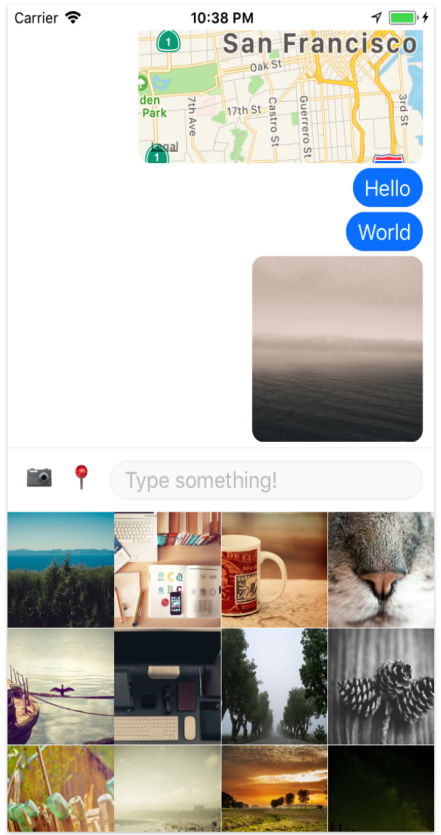
iOS



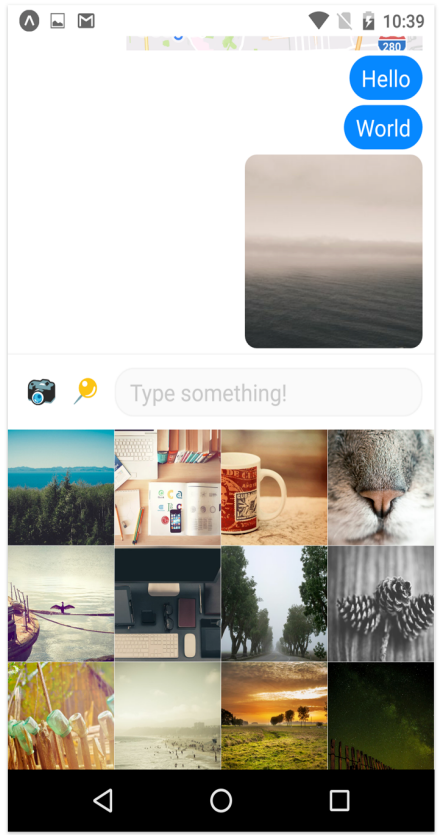
Android



iOS

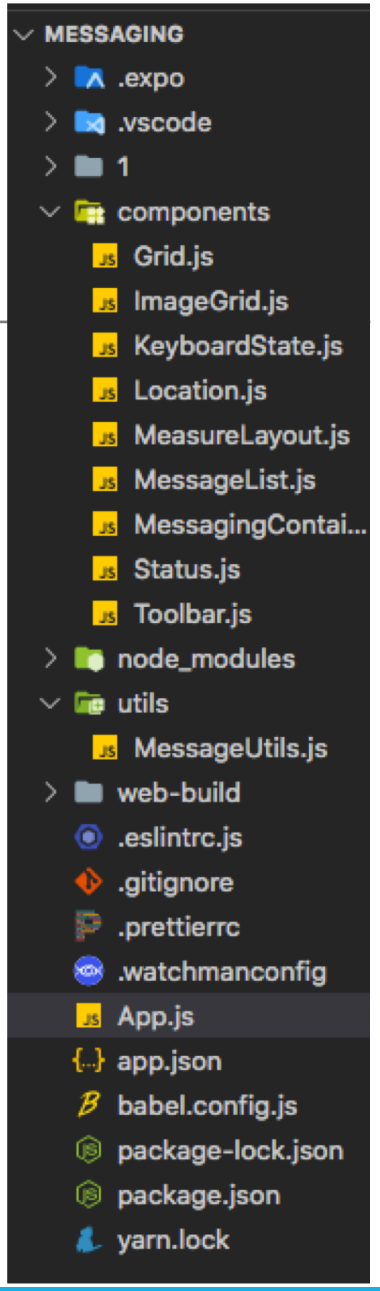


Android

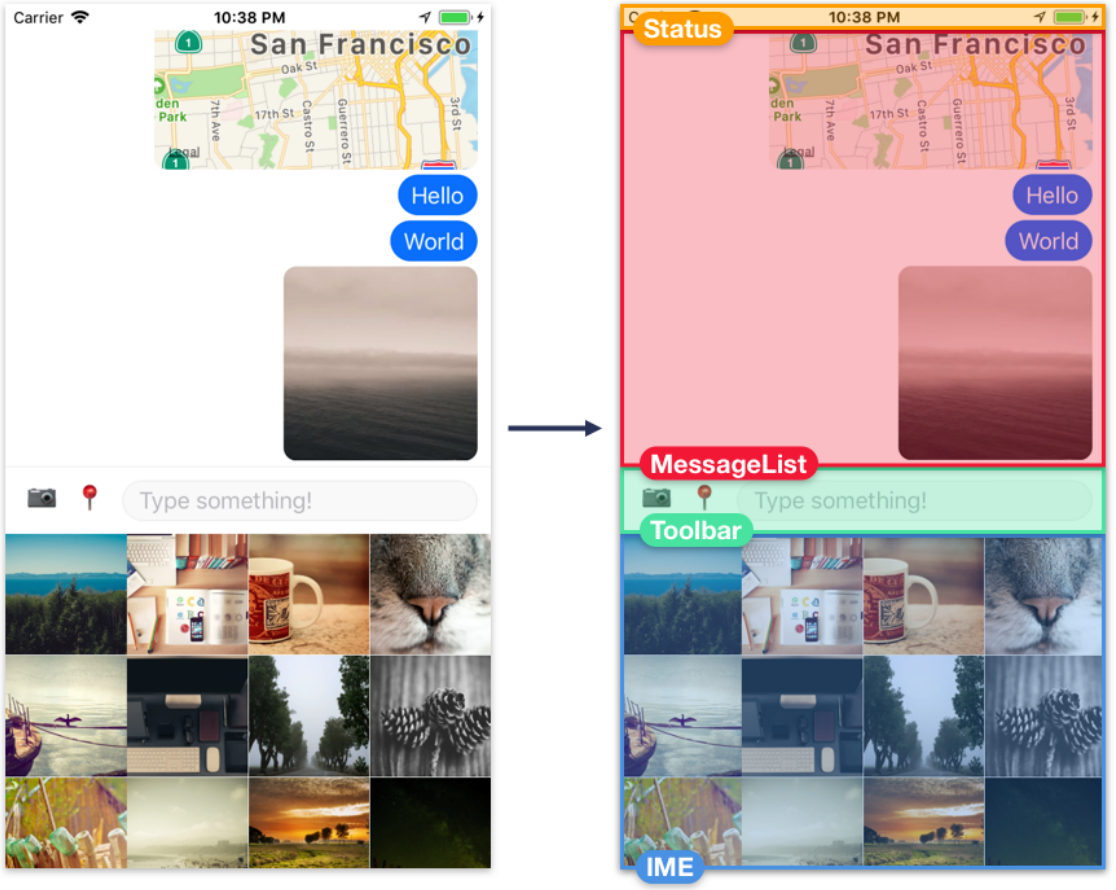


APIs in Messages Application

- 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

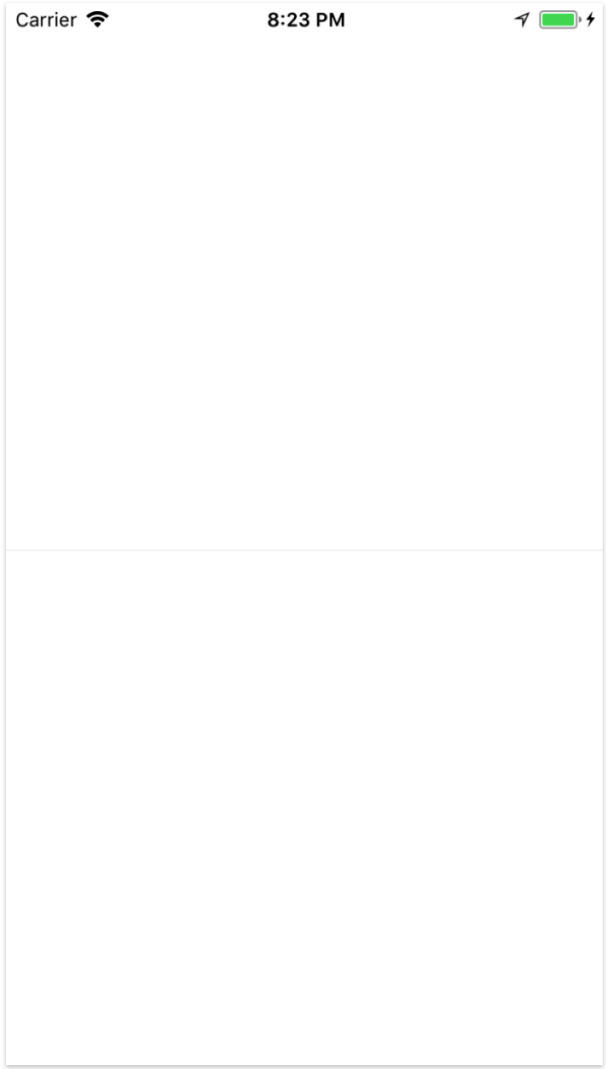


- Status ทำการแสดงข้อมูลพื้นฐานของแอพพิเคชัน เช่น เวลา แบตเตอรี่ ซึ่งจะแสดงในแนวนอน ตำแหน่งบนสุดของจอภาพ แต่ในกรณีนี้จะแสดงสถานะการเชื่อมต่อของเน็ตเวริก์ด้วย
- MessageList ทำหน้าที่ render ข้อความ รูปภาพและแผนที่ที่ส่งเข้ามาในแอพพิเคชัน
- Toolbar ทำหน้าที่สลับการทำงานระหว่าง การส่งข้อความ รูป ตำแหน่งของอุปกรณ์ ขณะที่ผู้ใช้กำลังใช้งานแอพพิเคชัน ในระหว่างที่ทำการพิมพ์ข้อความได้
- Input Method Editor (IME) ทำหน้าที่ render พวก Input method ต่าง ๆ เช่น imageGrid เพื่อทำการจัดการ image component หรือหากผู้ใช้ เรียกใช้คีย์บอร์ด แอพพิเคชันจะทำการจัดการแสดงหรือไม่แสดงคียบร์ดได้

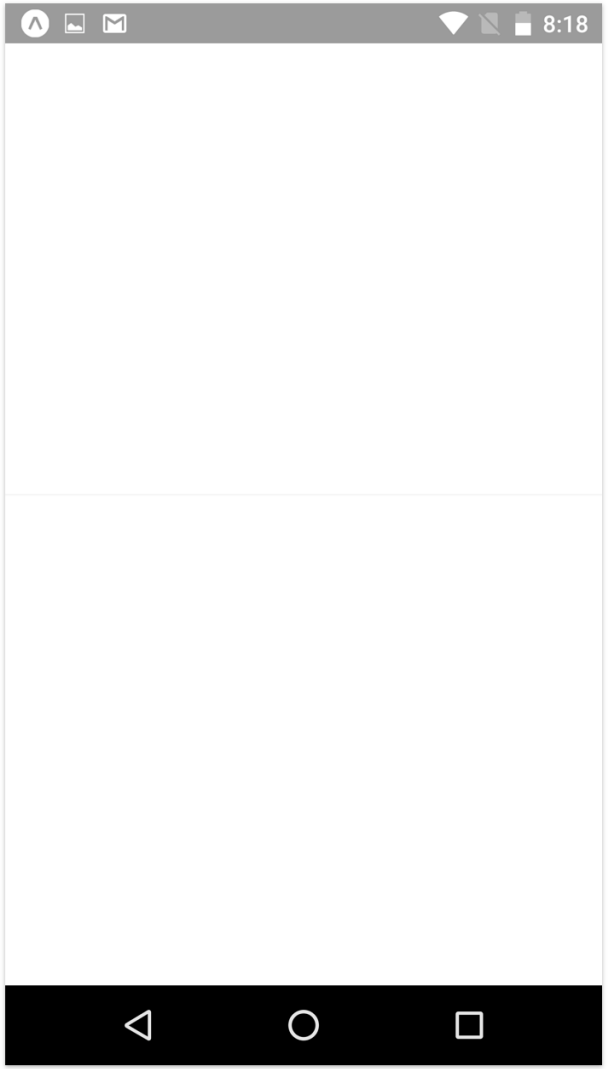
Messaging/App.js

```
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', }, })
```

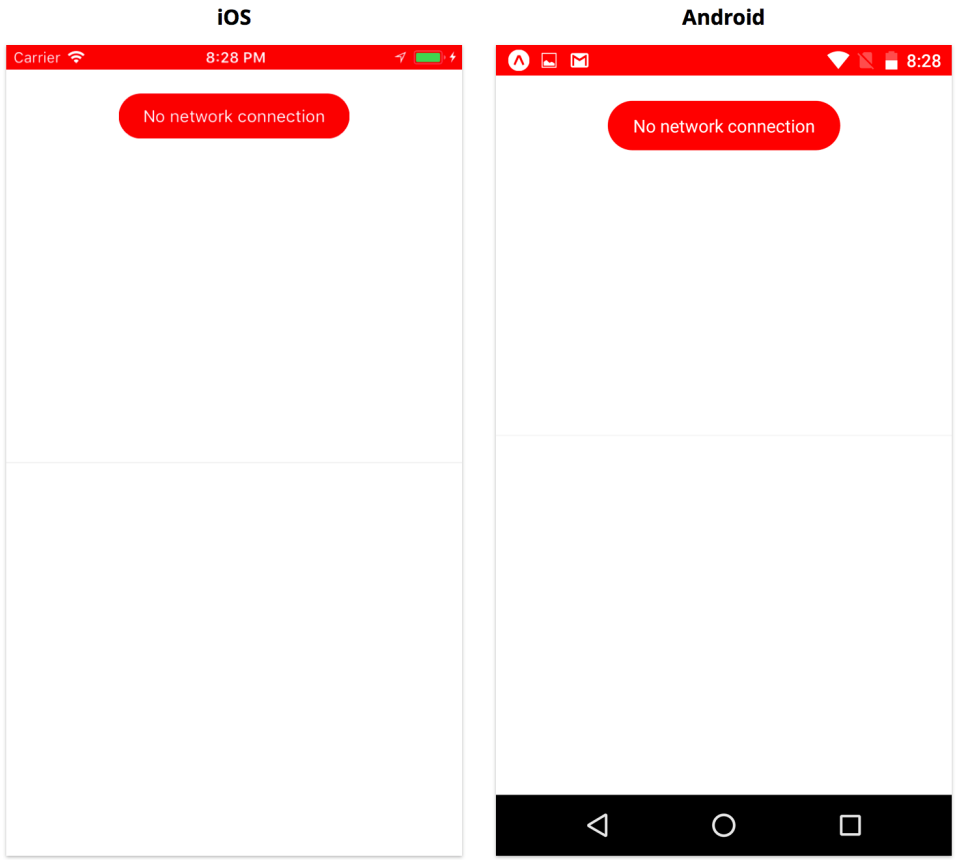
iOS



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

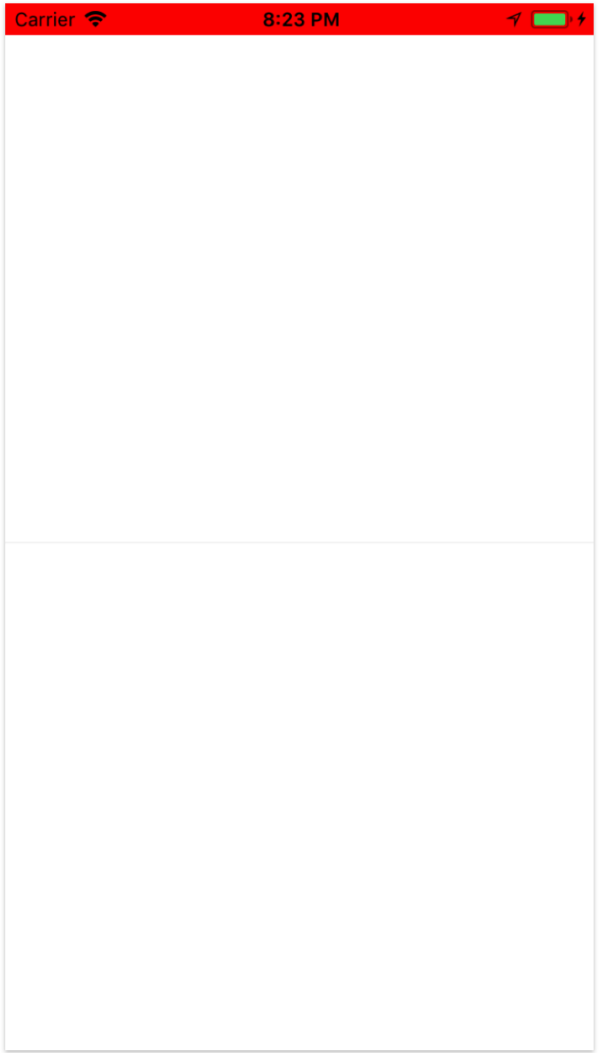
```
import Constants from 'expo-constants';
import NetInfo from '@react-native-community/netinfo';
import {
Platform,
StatusBar,
StyleSheet,
Text,
View,
} from 'react-native';
import React from 'react';
export default class Status extends React.Component {
state = {
isConnected: null,
};
// ...
```

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

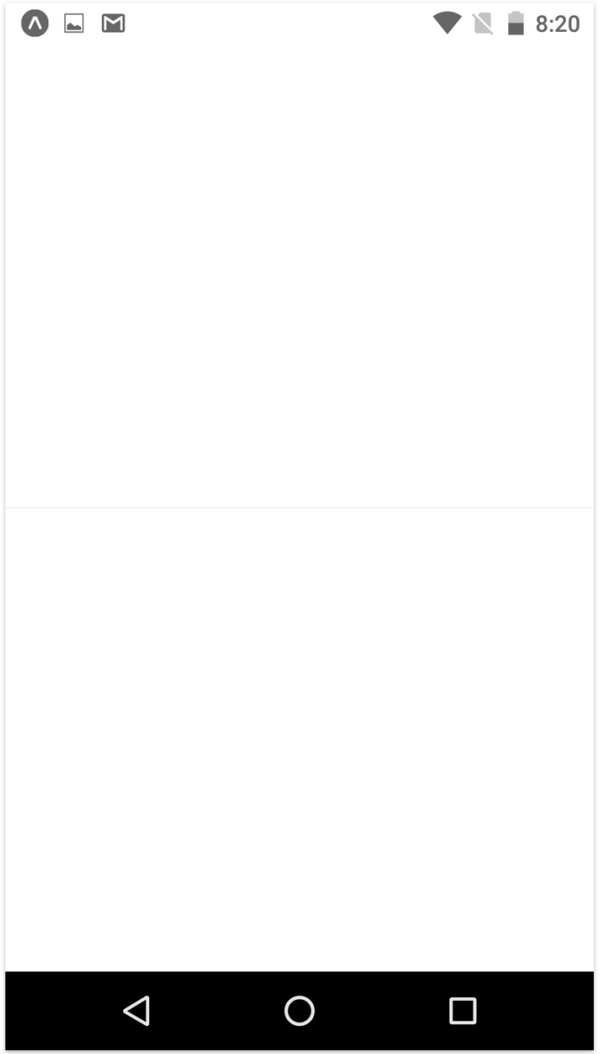
messaging/App.js

```
// ...
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



Android

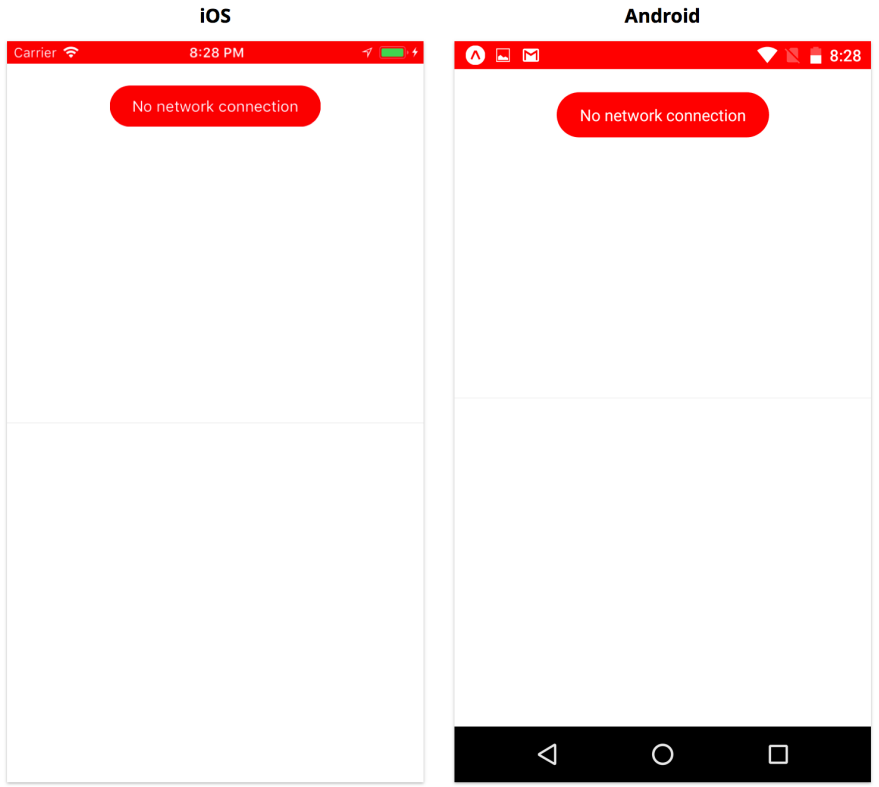


messaging/components/Status.js

```
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

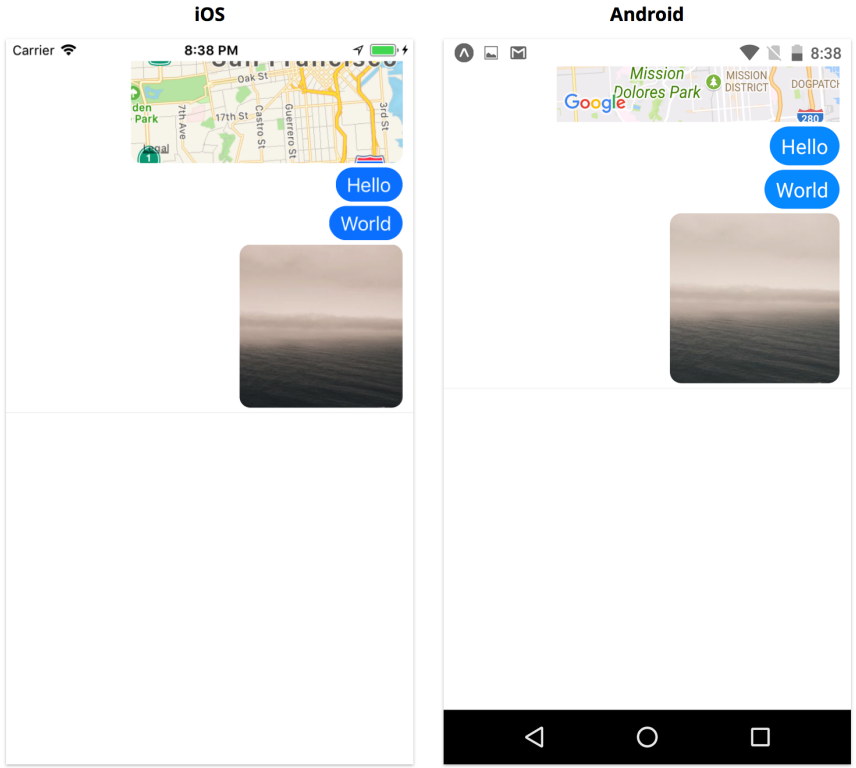
- isConnected in state
- Call NetInfo.fetch() -> connected = true
- Handle -> event listener to NetInfo.NetInfo
  - Call addEventListener

```
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

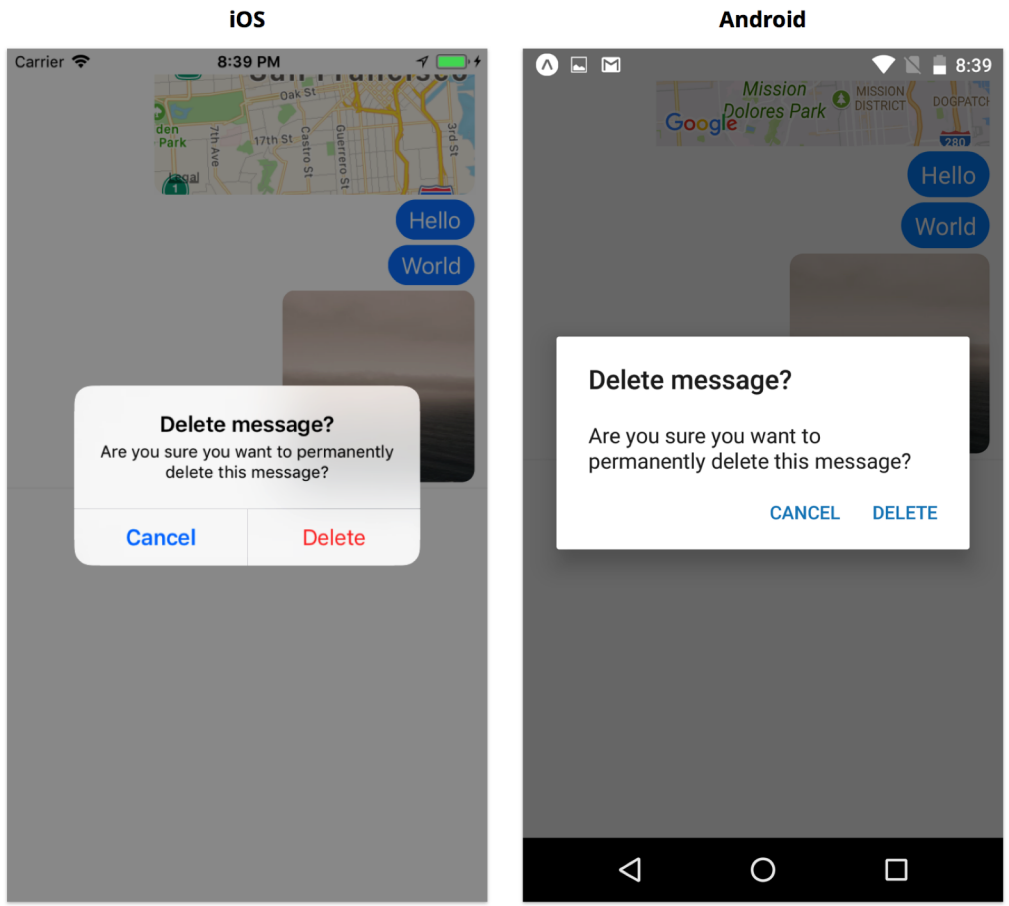
```
import React from 'react';
import PropTypes from 'prop-types’;

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

- 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)

Alert

- The full method signature is Alert.alert(title, message?, buttons?, options?, type?):
  - title - A string, shown in a large bold font, at the top of the dialog
  - message - A string, typically longer, shown in a normal weight font below the title
  - buttons - An array of objects containing text (a string), onPress (a callback function), and optionally a style on iOS (styles can be one of default, cancel, or destructive).
  - options - An object for controlling the dialog dismissal behavior on Android. Tapping outside the dialog will normally exit the dialog. This can be prevented by setting { cancelable: false } or handled specially with { onDismiss: () => {} }.
  - type - Allows text entry on iOS using one of the following options: default, plain-text, secure-text, or login-password.



Alert

messaging/App.js

```
import {
  Alert,
  // ...
} from 'react-native';
```

messaging/App.js

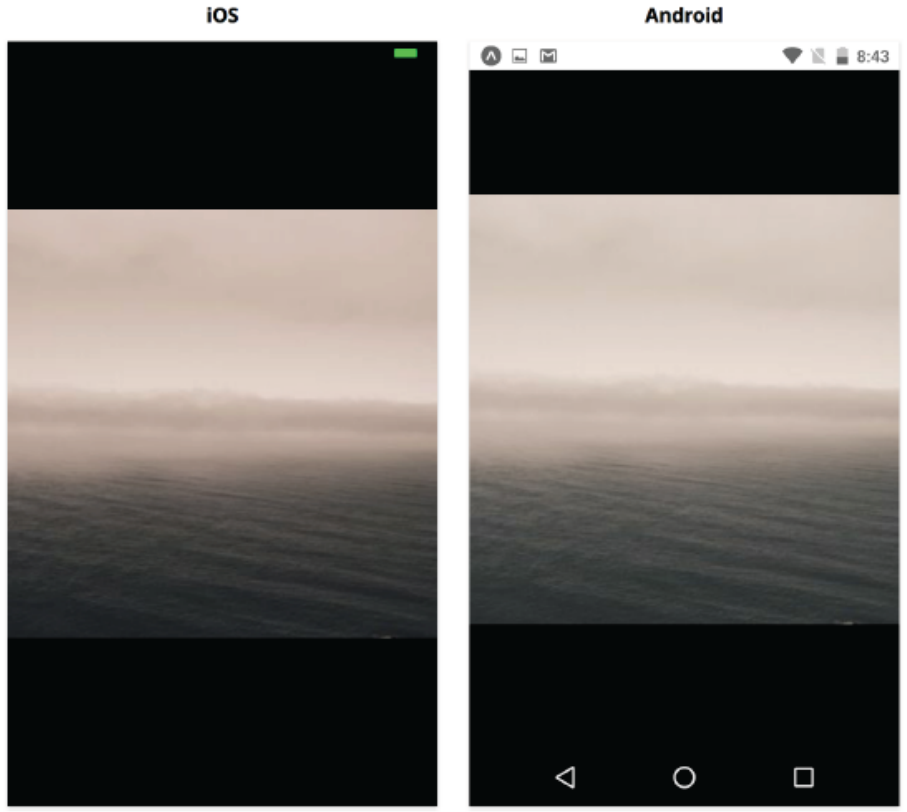
```
// ...

handlePressMessage = ({ id, type }) => {
  switch (type) {
    case 'text':
      Alert.alert(
        'Delete message?',
        'Are you sure you want to permanently delete this message?',
        [
          {
            text: 'Cancel',
            style: 'cancel',
          },
          {
            text: 'Delete',
            style: 'destructive',
            onPress: () => {
```

```
            const { messages } = this.state;
            this.setState({
              messages: messages.filter(
                message => message.id !== id,
              ),
            });
          },
        },
      ],
    );
    break;
  default:
    break;
  }
};

// ...
```

Fullscreen Image

messaging/App.js

```
// ...

state = {
  // ...
  fullscreenImageId: null,
};

dismissFullscreenImage = () => {
  this.setState({ fullscreenImageId: null });
};

// ...
```



Fullscreen Image

messaging/App.js

```
// ...

handlePressMessage = ({ id, type }) => {
  switch (type) {
    case 'text':
      // ...
    case 'image':
      this.setState({ fullscreenImageId: id });
      break;
    default:
      break;
  }
};

// ...
```

TouchableHighlight

- หากต้องการ Render รูปภาพ ต้องใช้ TouchableHighlight สำหรับทำการ refresh ภาพบนหน้าจอ หรือการทำ Overlay Background

messaging/App.js

```
import {
  // ...
  Image,
  TouchableHighlight,
} from 'react-native';
```

messaging/App.js

```
const styles = StyleSheet.create({
  // ...
  fullscreenOverlay: {
    ...StyleSheet.absoluteFillObject,
    backgroundColor: 'black',
    zIndex: 2,
  },
  fullscreenImage: {
    flex: 1,
    resizeMode: 'contain',
  },
});
```

messaging/App.js

```
// ...

renderFullscreenImage = () => {
  const { messages, fullscreenImageId } = this.state;

  if (!fullscreenImageId) return null;

  const image = messages.find(
    message => message.id === fullscreenImageId,
  );

  if (!image) return null;

  const { uri } = image;

  return (
    <TouchableHighlight
      style={styles.fullscreenOverlay}
      onPress={this.dismissFullscreenImage}
    >
      <Image style={styles.fullscreenImage} source={{ uri }} />
    </TouchableHighlight>
  );
};
```

```
  render() {
    return (
      <View style={styles.container}>
        <Status />

        {this.renderMessageList()}
        {this.renderToolbar()}
        {this.renderInputMethodEditor()}
        {this.renderFullscreenImage()}
      </View>
    );
  }
```

BlackHandler (สำหรับ iOS)

messaging/App.js

```
import {
  // ...
  BackHandler,
} from 'react-native';
```

messaging/App.js

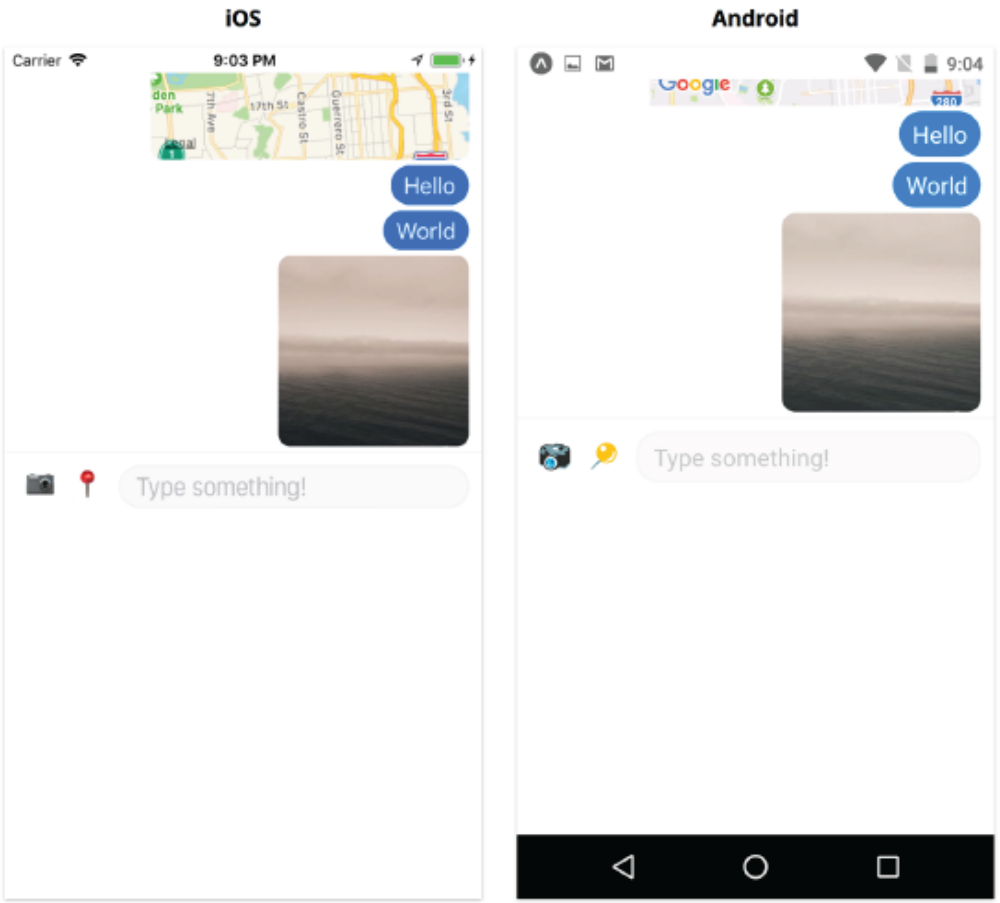
```
// ...

componentDidMount() {
  this.subscription = BackHandler.addEventListener(
    'hardwareBackPress',
    () => {
      const { fullscreenImageId } = this.state;

      if (fullscreenImageId) {
        this.dismissFullscreenImage();
        return true;
      }

      return false;
    },
  );
}

componentWillUnmount() {
  this.subscription.remove();
}

// ...
```

Toolbar

- มีลักษณะคล้ายกับ CommentInput ที่อยู่ใน Core Component (State : TextInput, onSubmit, isFocused, onChangeFocus)

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',
  },
  // ...
});
```

Add Carmera button and location : onPressCamera & 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',
  },
  // ...
});
```

Add Toolbar to App.js

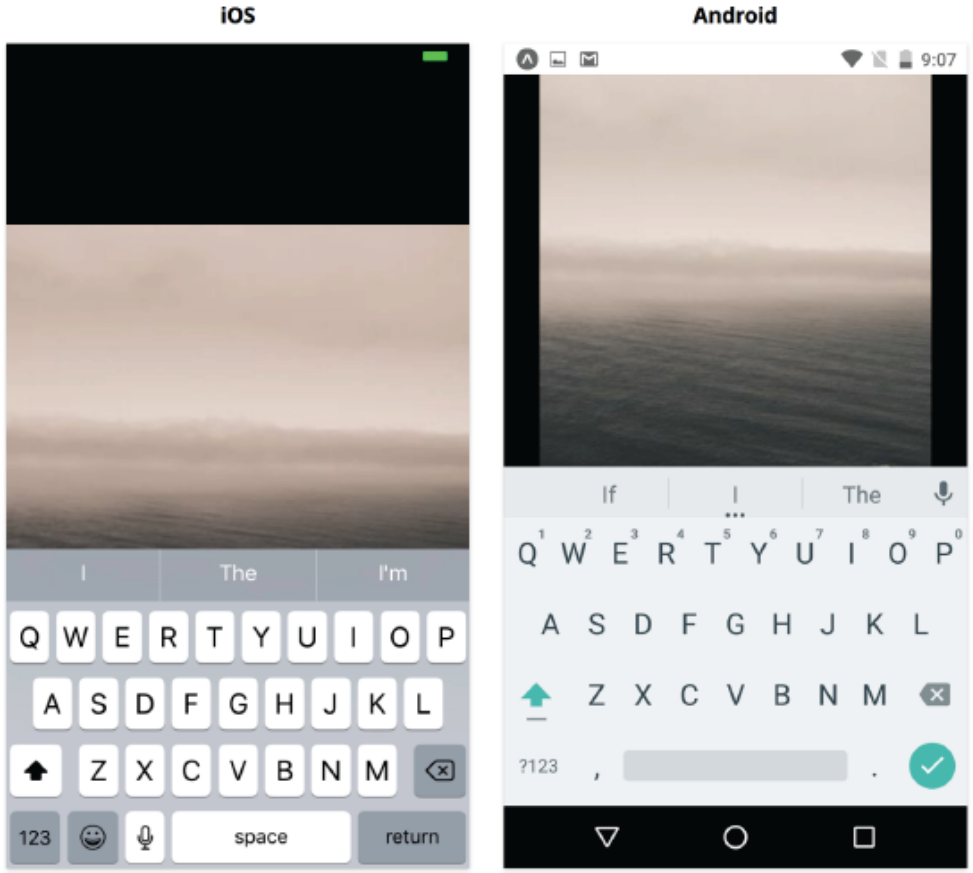
messaging/App.js

```
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>
    );
  }

  // ...

}

// ...
```

Update : handlePressMessage

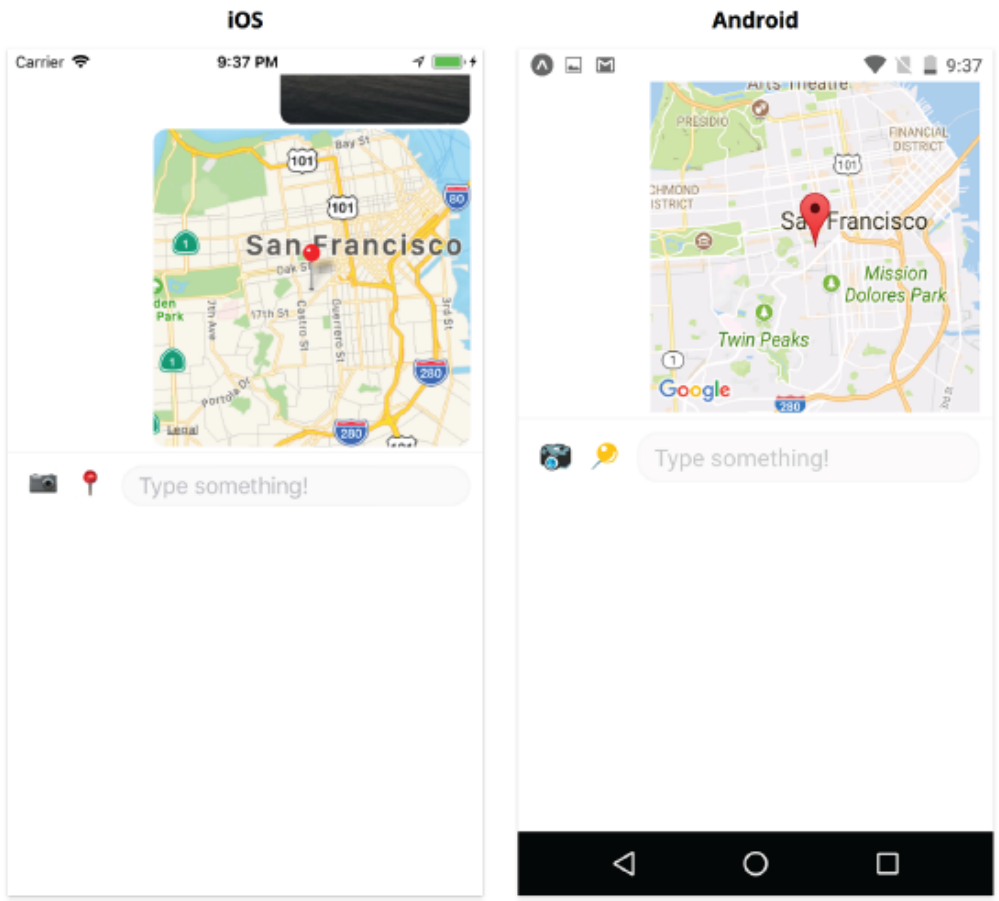
messaging/App.js

```
// ...

handlePressMessage = ({ id, type }) => {
  switch (type) {
    // ...
    case 'image':
      this.setState({
        fullscreenImageId: id,
        isInputFocused: false,
      });
      break;
    default:
      break;
  }
};

// ...
```



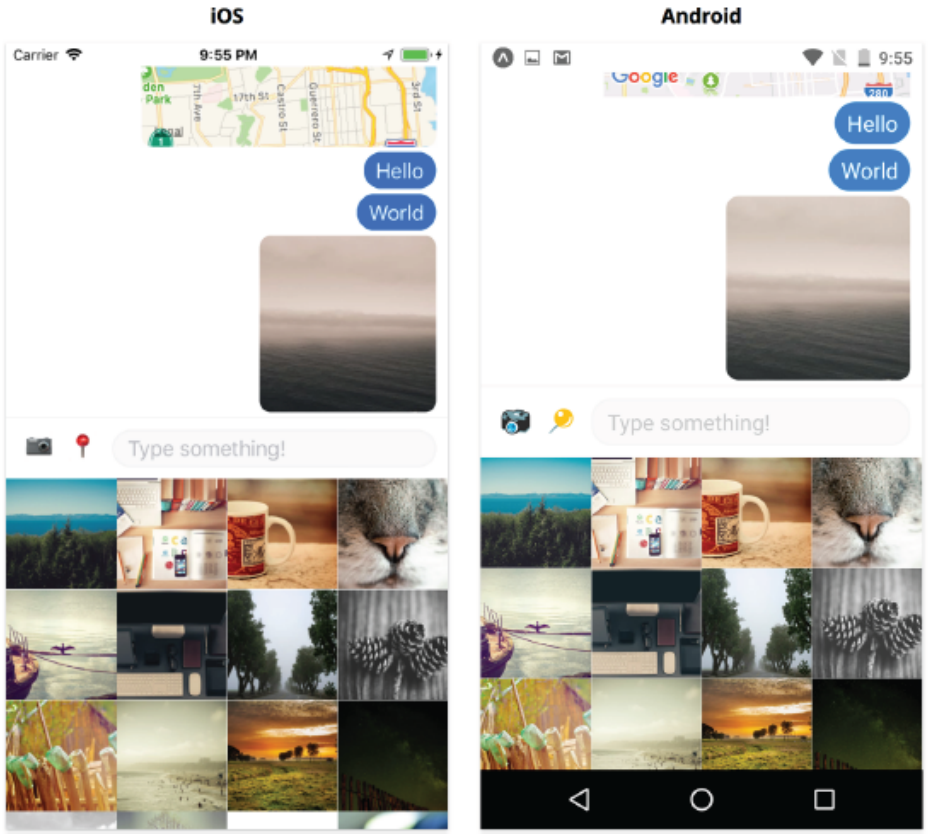
Geolocation : App.js

```
// ...

handlePressToolbarLocation = () => {
  const { messages } = this.state;

  navigator.geolocation.getCurrentPosition((position) => {
    const { coords: { latitude, longitude } } = position;

    this.setState({
      messages: [
        createLocationMessage({
          latitude,
          longitude,
        }),
        ...messages,
      ],
    });
  });
};

// ...
```



Input Method Editor (IME)

- Image picker : CarmeraRoll, Dimensions, PixelRatio
- Building a grid (Grid.js) -> Flastlist -> numColumns prop



messaging/components/Grid.js

```
import { Dimensions, FlatList, PixelRatio, StyleSheet } from 'react-nat\
ive';
import PropTypes from 'prop-types';
import React from 'react';

export default class Grid extends React.Component {
  static propTypes = {
    renderItem: PropTypes.func.isRequired,
    numColumns: PropTypes.number,
    itemMargin: PropTypes.number,
  };

  static defaultProps = {
    numColumns: 4,
    itemMargin: StyleSheet.hairlineWidth,
  };

  renderGridItem = (info) => {
    // ... The interesting stuff happens here!
  };

  render() {
    return (
      <FlatList {...this.props} renderItem={this.renderGridItem} />
    );
  }
}
```

messaging/components/Grid.js

```
renderGridItem = (info) => {
  const { index } = info;
  const { numColumns, itemMargin } = this.props;

  const { width } = Dimensions.get('window');

  const size = PixelRatio.roundToNearestPixel(
    (width - itemMargin * (numColumns - 1)) / numColumns,
  );

  // We don't want to include a `marginLeft` on the first item of a
  // row
  const marginLeft = index % numColumns === 0 ? 0 : itemMargin;

  // We don't want to include a `marginTop` on the first row of the
  // grid
  const marginTop = index < numColumns ? 0 : itemMargin;

  // ...
};
```

messaging/components/Grid.js

```
renderGridItem = (info) => {
  const { renderItem, numColumns, itemMargin } = this.props;

  // ...

  return renderItem({ ...info, size, marginLeft, marginTop });
};
```

Adding images to the grid

messaging/components/ImageGrid.js

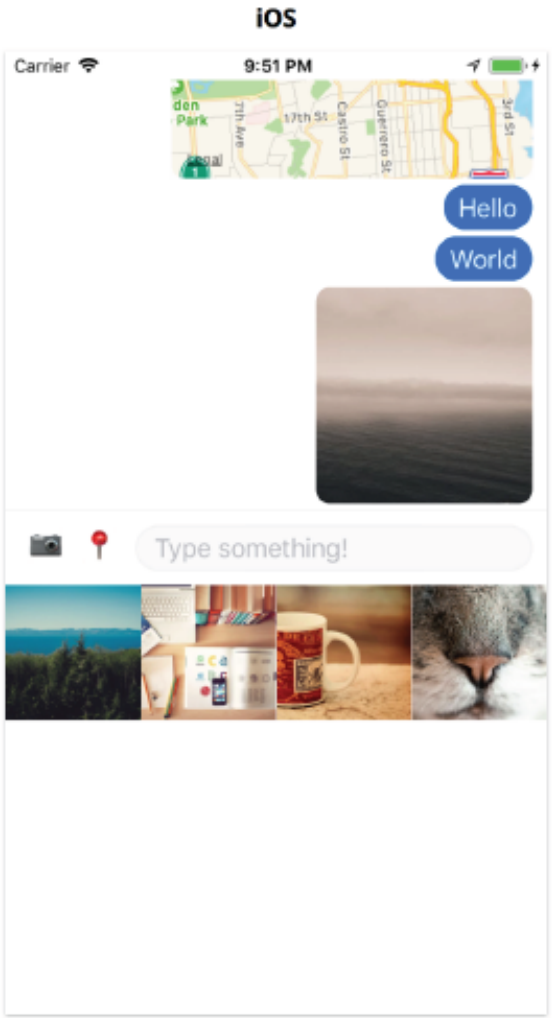
```
import { Image, StyleSheet, TouchableOpacity } from 'react-native';
import CameraRoll from 'expo-cameraroll'
import * as Permissions from 'expo-permissions';
import PropTypes from 'prop-types';
import React from 'react';

import Grid from './Grid';

const keyExtractor = ({ uri }) => uri;
```

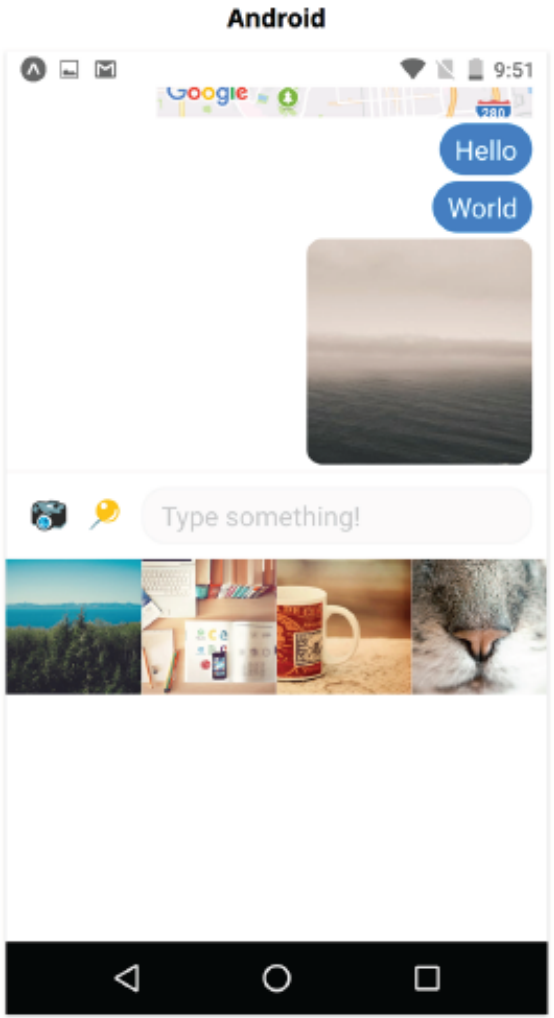
```
export default class ImageGrid extends React.Component {
  static propTypes = {
    onPressImage: PropTypes.func,
  };

  static defaultProps = {
    onPressImage: () => {},
  };

  state = {
    images: [
      { uri: 'https://picsum.photos/600/600?image=10' },
      { uri: 'https://picsum.photos/600/600?image=20' },
      { uri: 'https://picsum.photos/600/600?image=30' },
      { uri: 'https://picsum.photos/600/600?image=40' },
    ],
  };
```

```
renderItem = ({ item: { uri }, size, marginTop, marginLeft }) => {
  const style = {
    width: size,
    height: size,
    marginLeft,
    marginTop,
  };

  return (
    <Image source={{ uri }} style={style} />
  );
};

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

  return (
    <Grid
      data={images}
      renderItem={this.renderItem}
      keyExtractor={keyExtractor}
      // ...
    />
  );
}
}

const styles = StyleSheet.create({
  image: {
    flex: 1,
  },
});
```

messaging/App.js

```
// ...

import ImageGrid from './components/ImageGrid';

export default class App extends React.Component {

  // ...

  renderInputMethodEditor = () => (
    <View style={styles.inputMethodEditor}>
      <ImageGrid />
    </View>
  );

  // ...

}

// ...
```





Loading images from camera roll

- Add -> State.images

```
await Permissions.askAsync(Permissions.CAMERA_ROLL);

if (status !== 'granted') {
  // Denied
} else {
  // Good to go!
}
```

messaging/components/ImageGrid.js

```
// ...

export default class ImageGrid extends React.Component {
  state = {
    images: [],
  };

  componentDidMount() {
    this.getImages();
  }

  getImages = async () => {
    const { status } = await Permissions.askAsync(
      Permissions.CAMERA_ROLL,
    );

    if (status !== 'granted') {
      console.log('Camera roll permission denied');
      return;
    }

    const results = await CameraRoll.getPhotos({
      first: 20,
      assetType: 'Photos',
    });

    const { edges } = results;

    const loadedImages = edges.map(item => item.node.image);

    this.setState({ images: loadedImages });
  };

  // ...
}
```

messaging/components/ImageGrid.js

```
// ...

export default class ImageGrid extends React.Component {
  loading = false;
  cursor = null;

  // ...

  getNextImages = () => {
    if (!this.cursor) return;

    this.getImages(this.cursor);
  };
```

```
  getImages = async (after) => {
    if (this.loading) return;

    this.loading = true;

    const results = await CameraRoll.getPhotos({
      first: 20,
      after,
      assetType: 'Photos',
    });

    const {
      edges,
      page_info: { has_next_page, end_cursor },
    } = results;

    const loadedImages = edges.map(item => item.node.image);
```

```
    this.setState(
      {
        images: this.state.images.concat(loadedImages),
      },
      () => {
        this.loading = false;
        this.cursor = has_next_page ? end_cursor : null;
      },
    );
  };
}
```

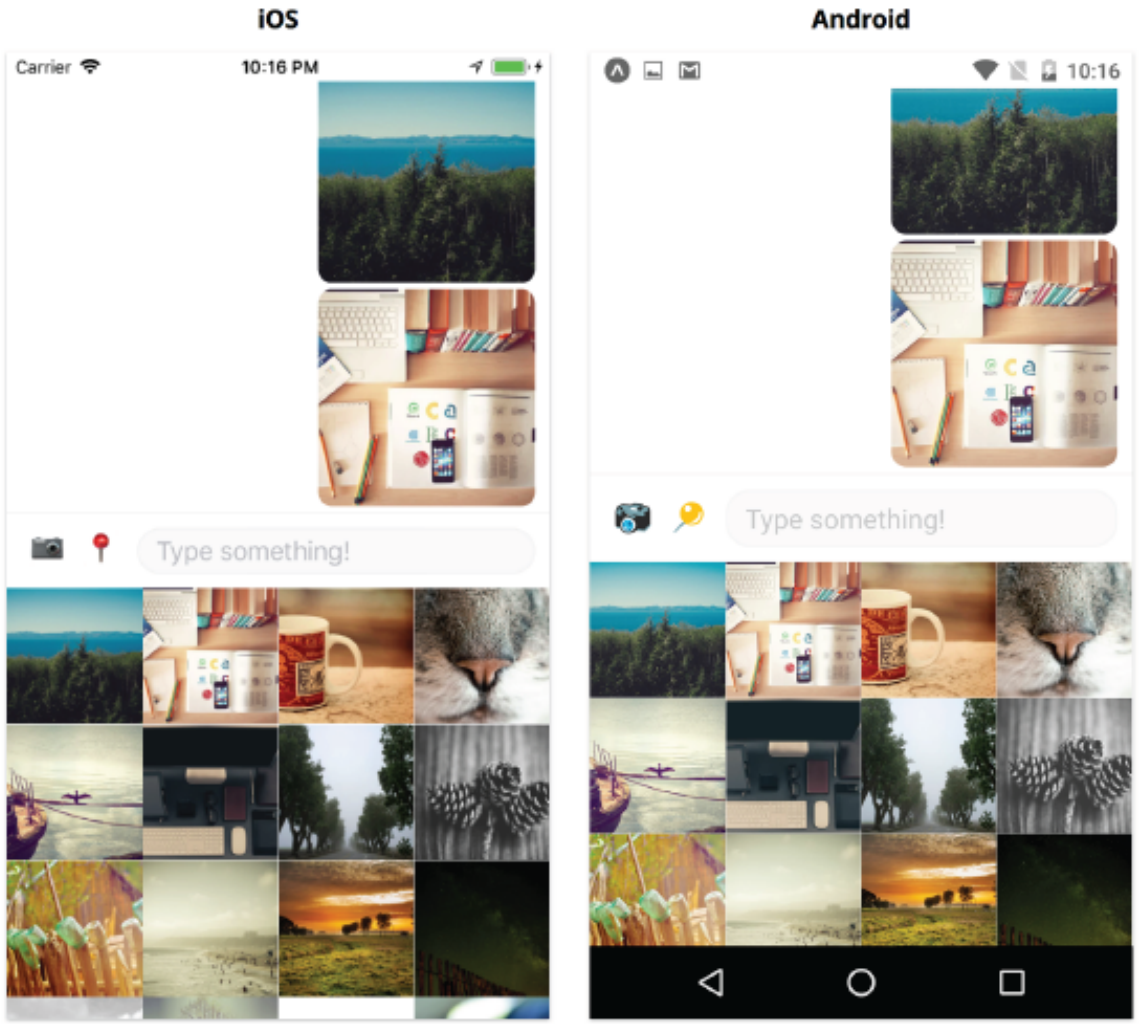
messaging/components/ImageGrid.js

```
// ...

renderItem = ({ item: { uri }, size, marginTop, marginLeft }) => {
  const { onPressImage } = this.props;

  const style = {
    width: size,
    height: size,
    marginLeft,
    marginTop,
  };

  return (
    <TouchableOpacity
      key={uri}
      activeOpacity={0.75}
      onPress={() => onPressImage(uri)}
      style={style}
    >
      <Image source={{ uri }} style={styles.image} />
    </TouchableOpacity>
  );
};

// ...
```

Sending images from ImageGrid

messaging/App.js

```
// ...

export default class App extends React.Component {

  // ...

  handlePressImage = (uri) => {
    const { messages } = this.state;

    this.setState({
      messages: [createImageMessage(uri), ...messages],
    });
  };

  renderInputMethodEditor = () => (
    <View style={styles.inputMethodEditor}>
      <ImageGrid onPressImage={this.handlePressImage} />
    </View>
  );

  // ...

}
```



Advance Messenger Application

- MeasureLayout - This component will measure the available space for our messaging UI
- KeyboardState - This component will keep track of the keyboard's visibility, height, etc
- MessagingContainer - This component will displaying the correct IME (text,

Measuring the available space

messaging/components/MeasureLayout.js

```
import Constants from 'expo-constants';
import { Platform, StyleSheet, View } from 'react-native';
import PropTypes from 'prop-types';
import React from 'react';

export default class MeasureLayout extends React.Component {
  static propTypes = {
    children: PropTypes.func.isRequired,
  };

  state = {
    layout: null,
  };
```

```
  handleLayout = event => {
    const { nativeEvent: { layout } } = event;

    this.setState({
      layout: {
        ...layout,
        y:
          layout.y +
          (Platform.OS === 'android' ? Constants.statusBarHeight : 0),
      },
```

```
    });
  };

  render() {
    const { children } = this.props;
    const { layout } = this.state;

    // Measure the available space with a placeholder view set to
    // flex 1
    if (!layout) {
      return (
        <View onLayout={this.handleLayout} style={styles.container} />
      );
    }

    return children(layout);
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
});
```

Keyboard events: KeyboardState

messaging/components/KeyboardState.js

```
import { Keyboard, Platform } from "react-native";
import PropTypes from 'prop-types';
import React from 'react';

export default class KeyboardState extends React.Component {
  static propTypes = {
    layout: PropTypes.shape({
      x: PropTypes.number.isRequired,
      y: PropTypes.number.isRequired,
      width: PropTypes.number.isRequired,
      height: PropTypes.number.isRequired,
    }).isRequired,
    children: PropTypes.func.isRequired,
  };

  // ...
}
```

Children of KeyboardState component

- contentHeight: The height available for our messaging content.
- keyboardHeight: The height of the keyboard. We keep track of this so we set our image picker to the same size as the keyboard.
- keyboardVisible: Is the keyboard fully visible or fully hidden?
- keyboardWillShow: Is the keyboard animating into view currently? This is only relevant on iOS.
- keyboardWillHide: Is the keyboard animating out of view currently? This is only relevant on iOS, and we'll only use it for fixing visual issues on the iPhone X.
- keyboardAnimationDuration: When we animate our UI to avoid the keyboard, we'll want to use the same animation duration as the keyboard. Let's initialize this with the value 250 (in milliseconds) as an approximation.

messaging/components/KeyboardState.js

```
// ...

const INITIAL_ANIMATION_DURATION = 250;

export default class KeyboardState extends React.Component {
  // ...

  constructor(props) {
    super(props);

    const { layout: { height } } = props;

    this.state = {
      contentHeight: height,
      keyboardHeight: 0,
      keyboardVisible: false,
      keyboardWillShow: false,
      keyboardWillHide: false,
      keyboardAnimationDuration: INITIAL_ANIMATION_DURATION,
    };
  }

  // ...
}
```

4 Keyboard events

- keyboardWillShow (iOS only) - The keyboard is going to appear
- keyboardWillHide (iOS only) - The keyboard is going to disappear
- keyboardDidShow - The keyboard is now fully visible
- keyboardDidHide - The keyboard is now fully hidden

```
// ...

  componentDidMount() {
    if (Platform.OS === 'ios') {
      this.subscriptions = [
        Keyboard.addListener(
          'keyboardWillShow',
          this.keyboardWillShow,
        ),
        Keyboard.addListener(
          'keyboardWillHide',
          this.keyboardWillHide,
        ),
        Keyboard.addListener('keyboardDidShow', this.keyboardDidShow),
        Keyboard.addListener('keyboardDidHide', this.keyboardDidHide),
      ];
    } else {
      this.subscriptions = [
        Keyboard.addListener('keyboardDidHide', this.keyboardDidHide),
        Keyboard.addListener('keyboardDidShow', this.keyboardDidShow),
      ];
    }
  }

  componentWillUnmount() {
    this.subscriptions.forEach(subscription => subscription.remove());
  }

// ...
```

messaging/components/KeyboardState.js

```
// ...

keyboardWillShow = (event) => {
  this.setState({ keyboardWillShow: true });

  // ...
};

keyboardDidShow = () => {
  this.setState({
    keyboardWillShow: false,
    keyboardVisible: true,
  });

  // ...
};

keyboardWillHide = (event) => {
  this.setState({ keyboardWillHide: true });

  // ...
};

keyboardDidHide = () => {
  this.setState({
    keyboardWillHide: false,
    keyboardVisible: false
  });
};

// ...
```