React Native

React Native คือ ?

- React Native คือ JavaScript **Framework** สำหรับสร้าง Cross-Platform Mobile Application
- React Native ใช้ React Library ทำให้ผู้ที่เคยรู้ React อยู่แล้วสามารถพัฒนาโปรแกรมบน React Native ได้อย่างรวดเร็ว
- บทเรียนนี้จะเน้นการรันโปรแกรมบนโทรศัพท์มือถือ **Android**
  - เนื่องจากสามารถใช้คอมพิวเตอร์ windows ในการทดสอบได้
  - โทรศัพท์ Android มีราคาต่ำกว่า iOS

เริ่มสร้าง React Native Project

- ใช้คำสั่งต่อไปนี้

```
npm install -g expo-cli
expo init CashBook
```

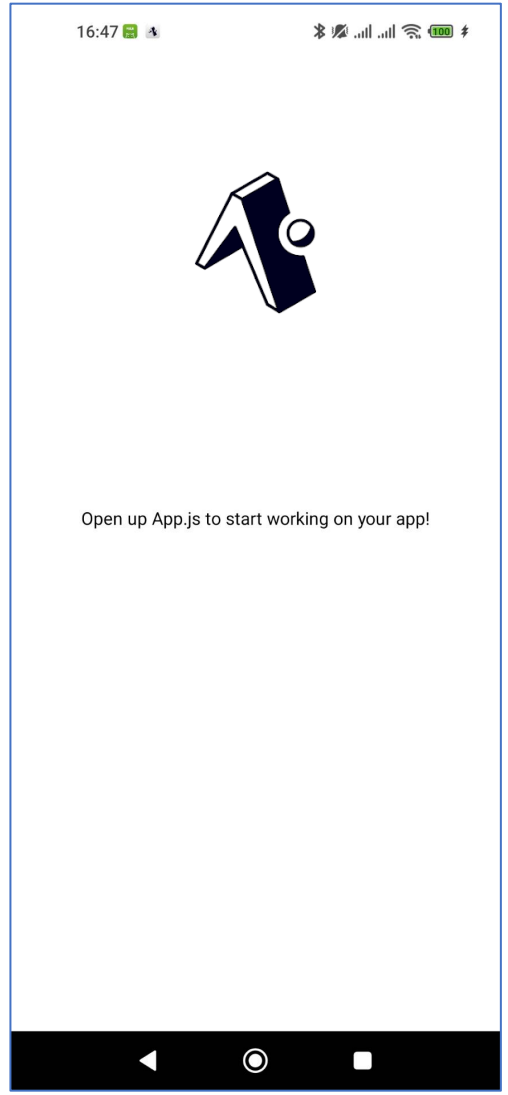
>> เลือก blank

```
cd CashBook
npm start
```

ติดตั้ง App Expo Go บนโทรศัพท์มือถือ จากนั้น scan QRCode ที่แสดงบนหน้าจอ

**โทรศัพท์ต้องอยู่ใน network เดียวกันกับเครื่องที่รันโปรแกรม

```
yarn run v1.22.18
$ expo start --android
Starting project at /Users/localadmin/Desktop/tmp_tutorial/DemoProject
Starting Metro Bundler
Started Metro Bundler
› Opening exp://192.168.1.62:19000 on M2007J3SY

› Metro waiting on exp://192.168.1.62:19000
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

› Press a │ open Android
› Press i │ open iOS simulator
› Press w │ open web

› Press r │ reload app
› Press m │ toggle menu

› Press ? │ show all commands

Logs for your project will appear below. Press Ctrl+C to exit.
Android Bundling complete 1257ms
Android Running app on M2007J3SY
```



ios : expo login -u <USERNAME-OR-EMAIL> -p <PASSWORD>

ทดลองใช้งาน React useState

- สร้างโปรแกรมที่มีปุ่มกด โดยเมื่อกดปุ่มแล้วให้นับจำนวนครั้งที่กด แล้วแสดงผลบนหน้าจอ
- ใช้ useState เหมือน react ปกติ
- import { Button } from ‘react-native’;
- <Button onPress={.....} title="....."/>

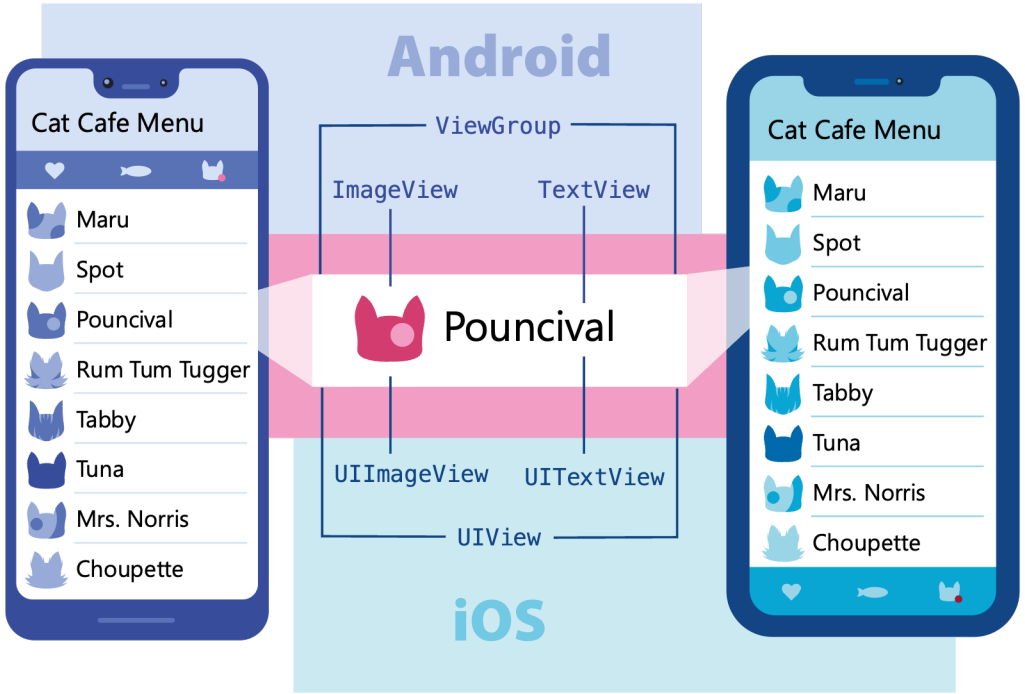
ทดสอบใช้ State

src/App.js

```
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View, Button } from 'react-native';
import { useState } from 'react';

export default function App() {
  const [counter, setCounter] = useState(0)
  return (
    <View style={styles.container}>
      <Text>{counter}</Text>
      <Button onPress={() => setCounter(counter + 1)} title='Counter' />
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

Views !!!!

- React Native ใช้ View tag เป็นส่วนสำหรับแสดงผลทุกอย่าง เช่น ตัวอักษร, ภาพ, กล่องใส่ข้อความ, ปุ่ม
- บาง View จะสามารถใส่ view อื่นๆ ไว้ภายในได้ เช่น ListView, ScrollView



Components

- React Native จะช่วยอำนวยความสะดวกในการเขียนโปรแกรม โดยการเขียนแค่ครั้งเดียว จะสามารถแปลง code เป็น ภาษาอื่นๆ ได้โดยอัตโนมัติ

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrolling `<div>`	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Displays, styles, and nests strings of text and even handles touch events
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Displays different types of images
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	A generic scrolling container that can contain multiple components and views
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Allows the user to enter text

Basic Components

Most apps will end up using one of these basic components.

View

The most fundamental component for building a UI.

Text

A component for displaying text.

Image

A component for displaying images.

TextInput

A component for inputting text into the app via a keyboard.

ScrollView

Provides a scrolling container that can host multiple components and views.

StyleSheet

Provides an abstraction layer similar to CSS stylesheets.

User Interface

These common user interface controls will render on any platform.

Button

A basic button component for handling touches that should render nicely on any platform.

Switch

Renders a boolean input.

List Views

Unlike the more generic `ScrollView`, the following list view components only render elements that are currently showing on the screen. This makes them a performant choice for displaying long lists of data.

FlatList	SectionList
A component for rendering performant scrollable lists.	Like `FlatList`, but for sectioned lists.

Android Components and APIs

Many of the following components provide wrappers for commonly used Android classes.

BackHandler

Detect hardware button presses for back navigation.

DrawerLayoutAndroid

Renders a `DrawerLayout` on Android.

PermissionsAndroid

Provides access to the permissions model introduced in Android M.

ToastAndroid

Create an Android Toast alert.

iOS Components and APIs

Many of the following components provide wrappers for commonly used UIKit classes.

ActionSheetIOS

API to display an iOS action sheet or share sheet.

Others

These components may be useful for certain applications. For an exhaustive list of components and APIs, check out the sidebar to the left (or menu above, if you are on a narrow screen).

ActivityIndicator

Displays a circular loading indicator.

Alert

Launches an alert dialog with the specified title and message.

Animated

A library for creating fluid, powerful animations that are easy to build and maintain.

Dimensions

Provides an interface for getting device dimensions.

KeyboardAvoidingView

Provides a view that moves out of the way of the virtual keyboard automatically.

Linking

Provides a general interface to interact with both incoming and outgoing app links.

Modal

Provides a simple way to present content above an enclosing view.

PixelRatio

Provides access to the device pixel density.

RefreshControl

This component is used inside a `ScrollView` to add pull to refresh functionality.

StatusBar

Component to control the app status bar.

FlatList

```
<FlatList
      data={data}
      renderItem={renderItem}
      keyExtractor={item => item.id}
/>
```

```
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View, Button, FlatList } from 'react-native';
import { useState } from 'react';

export default function App() {
  const [data, setData] = useState([])

  const renderItem = ({ item }) => (
    <View style={styles.item}>
     <Text style={styles.title}>{item.title}</Text>
    </View>
  )

  const randomData = () => {
    return {
      id: data.length + 1,
      title: `Number : ${data.length + 1}`
    }
  }

  return (
    <View style={styles.container}>
      <FlatList
        data={data}
        renderItem={renderItem}
        keyExtractor={item => item.id}
      />
      <Button onPress={() => setData([...data, randomData()])} title= 'Add'
/>

      <StatusBar style="auto" hidden={false}/>
    </View>
  );
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
    margin: 30,
  },
  item: {
    backgroundColor: '#f9c2ff',
    padding: 20,
    marginVertical: 8,
    marginHorizontal: 16,
  },
  title: {
    fontSize: 32,
  },
});
```

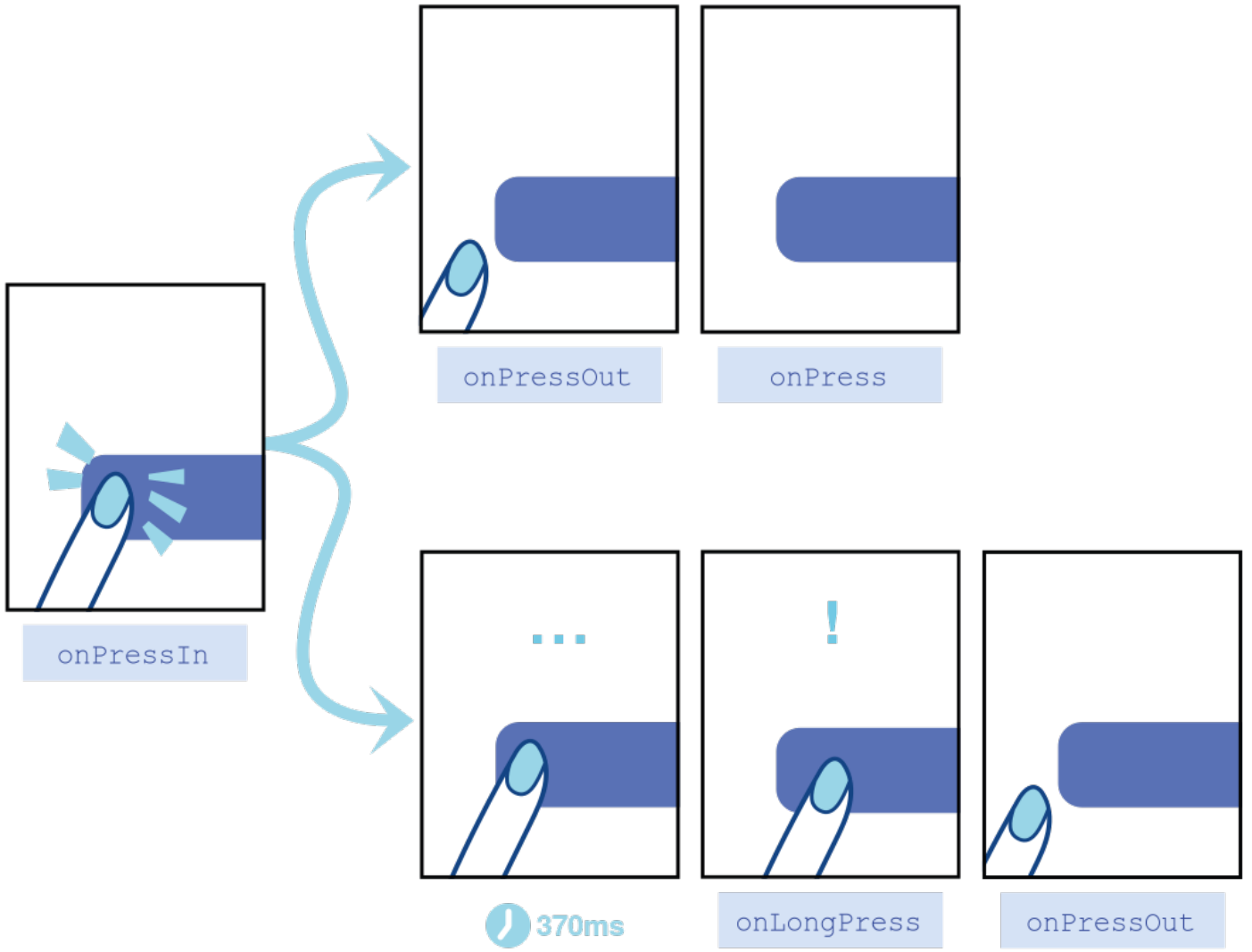
TouchableOpacity & TouchableWithoutFeedback

- ใส่ครอบ Component ใดๆ เพื่อให้ทำการ touch ได้

```
export default function App() {

  const renderItem = ({ item }) => (
    <TouchableOpacity onPress={() => console.log(`${item.id}`)}>
      <View style={styles.item}>
        <Text style={styles.title}>{item.title}</Text>
      </View>
    </TouchableOpacity>
  );

}
```

Pressable

- Component ใหม่ !!
- สำหรับรับการกดปุ่ม
- รองรับ event ที่มากกว่า
- จะไม่มีการกระพริบ



Image

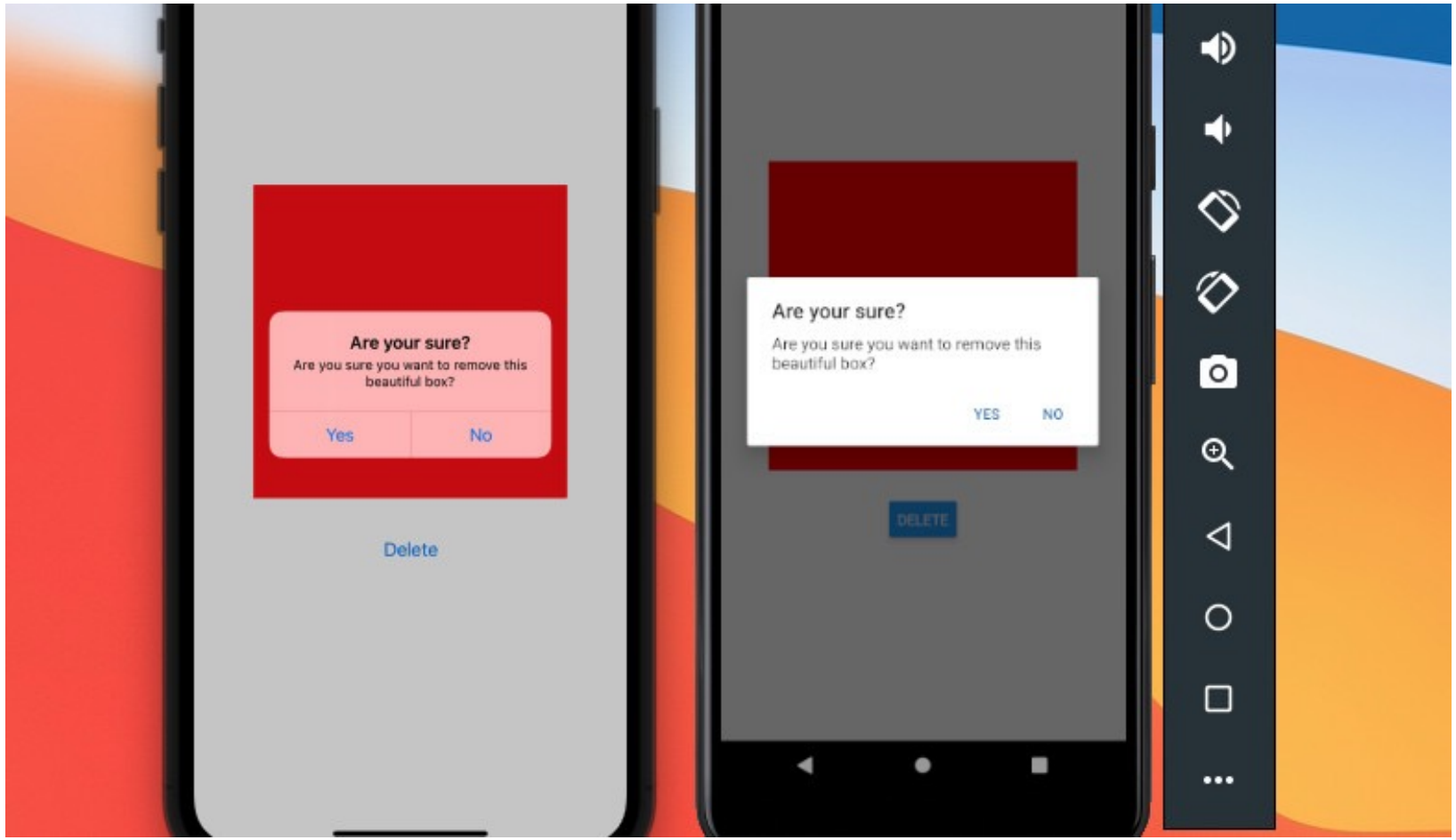
- สำหรับแสดงภาพบนหน้าจอ
- ใช้ property source สำหรับการแสดงผลภาพ

```
const styles = StyleSheet.create({
  tinyLogo: {
    width: 20,
    height: 20
  }
})
```

```
const renderItem = ({ item }) => (
    <TouchableOpacity onPress={() => console.log(`${item.id}`)}>
      <View style={styles.item}>
        <Image
          style={styles.tinyLogo}
          source={{uri:"https://ionicframework.com/docs/icons/logo-react-
icon.png"}} />
        <Text style={styles.title}>{item.title}</Text>
      </View>
    </TouchableOpacity>
  );
```

Alert

- สำหรับแสดง Popup Window เพื่อถามผู้ใช้งานโดยให้เลือกเป็น OK หรือ Cancel




```
Alert.alert(
  title,
  body,
  [buttons]
)
```

```
const showAlert = () => {
    Alert.alert(
      "Are you sure ?",
      `Do you want to add ${data.length + 1} into list ?`,
      [
        {
          text: "Cancel",
        },
        {
          text: "OK",
          onPress: () => setData([...data, randomData()])
        }
    ])
}

return (
    <View style={styles.container}>
      //.............................
      <Button onPress={showAlert} title='Add' />
    </View>
  );
```

Modal

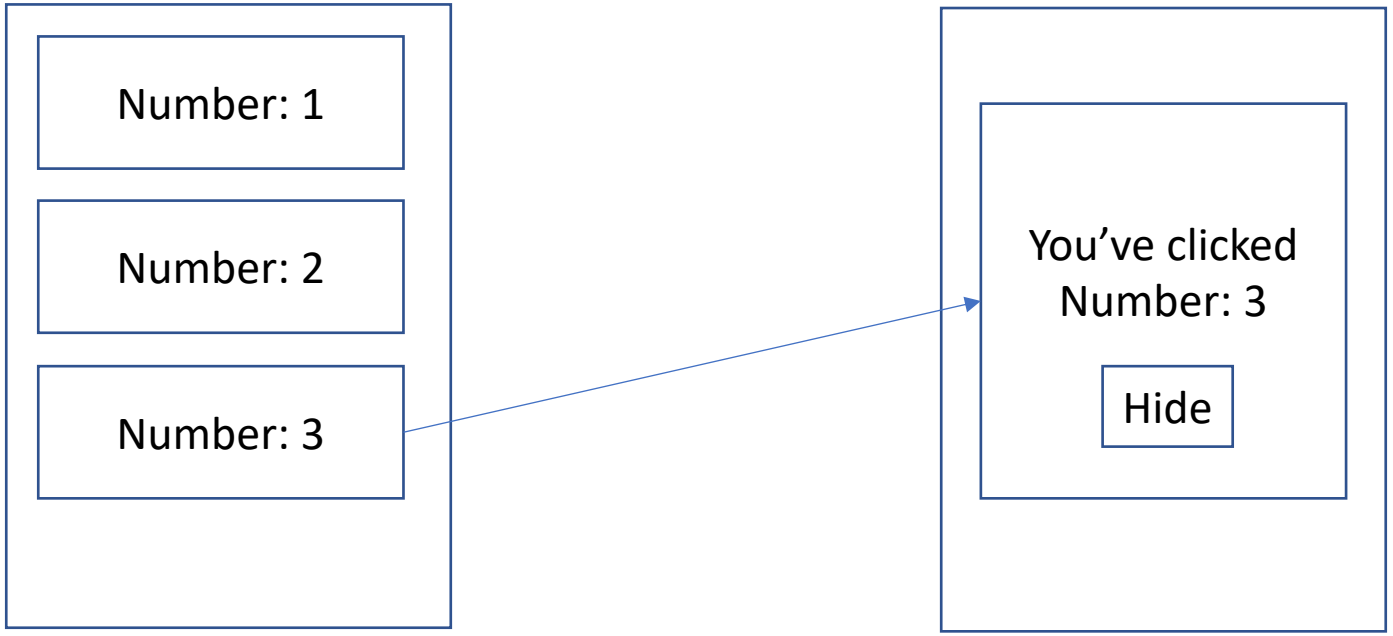
```
const styles = StyleSheet.create({
  centeredView: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    marginTop: 22
  },
})
```

- ใช้สำหรับแสดง Popup Window เช่นกัน แต่จะใช้สามารถ customize ได้มากกว่า

```
const [showModal, setShowModal] = useState(false)
return (
    <View style={styles.container}>
      //.............................
      <Button onPress={() => setShowModal(true)} title='Add' />
      <Modal
          visible={showModal}
          onRequestClose={() => setShowModal(false)}
          transparent={true}
        >
          <View style={{ backgroundColor: "grey", ...styles.centeredView }}>
            <Text>Click outside or back to dismiss.</Text>
            <Button onPress={() => setShowModal(false)} title="Hide"/>
          </View>
        </Modal>
    </View>
  );
```

Modal Excercise

- สร้าง Component ใหม่ชื่อ SampleModal
- ปรับโปรแกรมปัจจุบัน ให้กดที่ List Item แล้วแสดง Modal ขึ้นมา
- ภายใน Modal แสดง content ของ List Item



Styles

Size

- ขนาดของ component โดยมีการตั้งค่าดังต่อไปนี้
  - Fixed Dimensions สำหรับกำหนดขนาดตายตัว ผ่านทาง width และ height
  - Flex Dimensions เป็นการกำหนดขนาดแบบ dynamic โดยจะกำหนดขนาดตามพื้นที่ว่างที่เหลืออยู่ (Fill เต็มพื้นที่ตามสัดส่วน flex)
  - Percentage Dimensions เป็นการกำหนดขนาด เป็นค่า % ของหน้าจอ
- https://reactnative.dev/docs/height-and-width

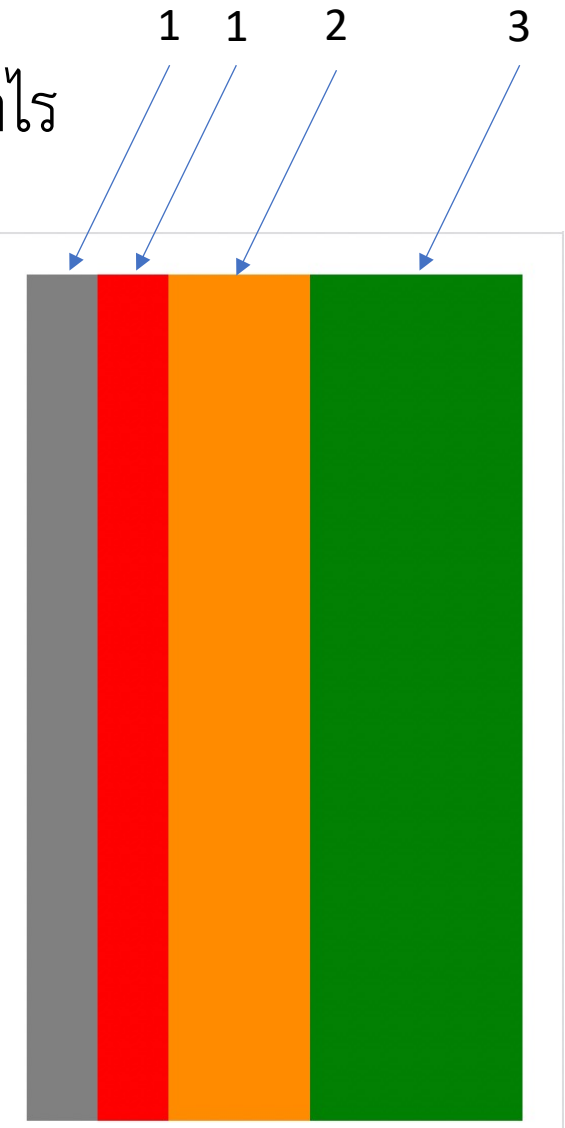
Layout with Flexbox

- Flexbox คือวิธีการบอกตำแหน่งของ component ที่จะแสดงผลบนหน้าจอ
- แนะนำให้ทดลองใน https://reactnative.dev/docs/flexbox

Flex

- สำหรับบอกว่า component ภายในจะมีขนาดเท่าไร

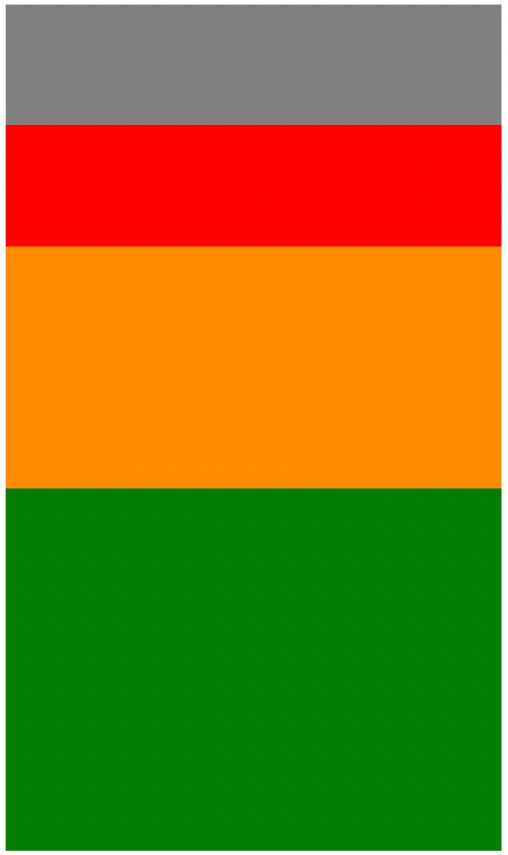
```
import React from "react";
import { StyleSheet, Text, View } from "react-native";

const Flex = () => {
  return (
    <View style={[styles.container, {
      flexDirection: "row"
    }]}>
      <View style={{ flex: 1, backgroundColor: "grey" }} />
      <View style={{ flex: 1, backgroundColor: "red" }} />
      <View style={{ flex: 2, backgroundColor: "darkorange" }} />
      <View style={{ flex: 3, backgroundColor: "green" }} />
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
  },
});

export default Flex;
```



Flex Direction

- กำหนดทิศทาง การวาง component ประกอบไปด้วย

```
import React from "react";
import { StyleSheet, Text, View } from "react-native";

const Flex = () => {
  return (
    <View style={[styles.container, {
      flexDirection: "column"
    }]}>
      <View style={{ flex: 1, backgroundColor: "grey" }} />
      <View style={{ flex: 1, backgroundColor: "red" }} />
      <View style={{ flex: 2, backgroundColor: "darkorange" }} />
      <View style={{ flex: 3, backgroundColor: "green" }} />
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
  },
});

export default Flex;
```



alignItems

- กำหนดตำแหน่งว่าจะวางตำแหน่งไหนของ parent ตามแนวแกน
  - stretch (default) - ยืด component ให้มีขนาดเท่ากับ parent
    - Stretch จะทำงานเมื่อ component ต้องไม่มีขนาดที่ fixed
  - flex-start – วางตำแหน่ง component ไว้ที่ตำแหน่งเริ่มต้นของ parent
  - flex-end - วางตำแหน่ง component ไว้ที่ตำแหน่งสิ้นสุดของ parent
  - center – วางตำแหน่ง component ไว้ตรงกึ่งกลางของ parent
  - baseline – วางตำแหน่ง component ยึดตาม parent

alignItems

- stretch
- flex-start
- flex-end
- center
- baseline

```
import React from "react";
import { StyleSheet, Text, View } from "react-native";

const Flex = () => {
  return (
    <View style={[styles.container, {
        alignItems :"stretch"
    }]}>
      <View
        style={{width:50, height:50, backgroundColor: "powderblue"
}}
      />
      <View
        style={{width:50, height:50, backgroundColor: "skyblue"}}
      />
      <View
        style={{
            width: "auto",
            height: 50,
            minWidth: 50,
            backgroundColor: "steelblue",
          }
        }
      />
    </View>
  );
```

Justify Content

- กำหนดตำแหน่งว่าจะวางตำแหน่งไหนของ parent ตามแนวแกนหลัก
  - flex-start (default) – วางตำแหน่งเดียวกับ parent
  - flex-end – วางไว้ที่ตำแหน่งหลังสุดของ parent
  - center – วางไว้กึ่งกลางของ parent
  - space-between – แบ่ง component ออกเป็นชิ้นย่อย ๆ โดยมีระยะเท่า ๆ กัน
  - space-around – คล้ายกับ space-between แต่จะมีพื้นที่ว่างที่ child ตัวแรกและสุดท้าย
  - space-evenly - แบ่ง component ออกเป็นชิ้นย่อย ๆ โดยมีระยะเท่า ๆ กัน (รวมทั้งพื้นที่ว่างตัวต้นและสุดท้ายด้วย)

Justify Content

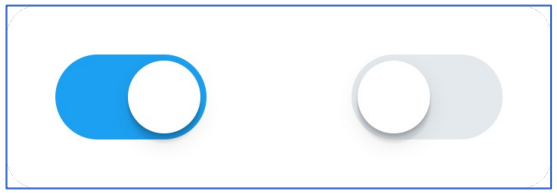
- flex-start
- flex-end
- center
- space-between
- space-around
- space-evenly

```
import React from "react";
import { StyleSheet, Text, View } from "react-native";

const Flex = () => {
  return (
    <View style={[styles.container, {
        justifyContent :"flex-start"
    }]}>
      <View
        style={{width:50, height:50, backgroundColor: "powderblue"
}}
      />
      <View
        style={{width:50, height:50, backgroundColor: "skyblue"}}
      />
      <View
        style={{
            width: "auto",
            height: 50,
            minWidth: 50,
            backgroundColor: "steelblue",
          }
        }
      />
    </View>
  );
```

Forms

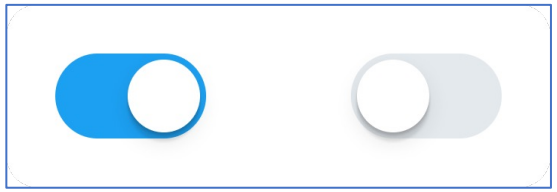
Input

- Switch
  - แสดงปุ่ม toggle สำหรับแสดงค่า true หรือ false



- TextInput
  - สำหรับกรอกข้อมูล

Switch



```
const [isEnabled, setIsEnabled] = useState(false);


<View style={styles.switchView}>
  <Text>รายจ่าย</Text>
  <Switch onValueChange={() => setIsEnabled(!isEnabled)} value={isEnabled}/>
</View>


const styles = StyleSheet.create({
  switchView: {
    flexDirection: "row",
    justifyContent: "center",
    alignItems: "center",
  }
});
```

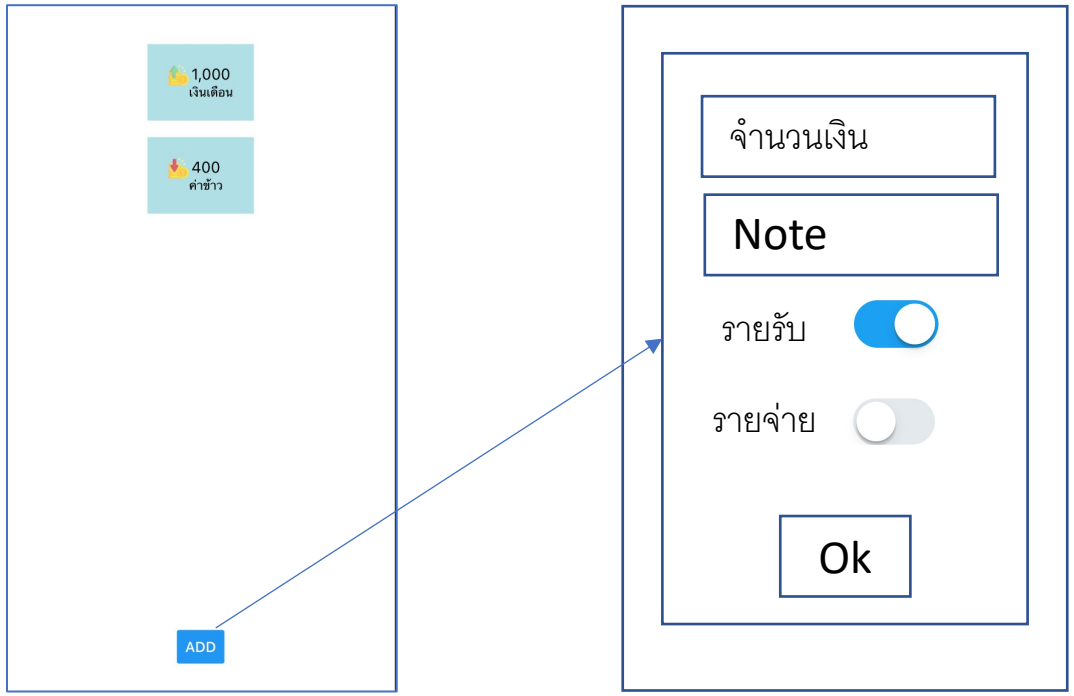
TextInput


```
const [amount, setAmount] = useState(0);
const [note, setNote] = useState(“”);

<TextInput style={styles.input} keyboardType='numeric' placeholder='Amount'
onChangeText={setAmount} value={amount.toString()}/>

<TextInput style={styles.input} placeholder=‘Input’ onChangeText={setNote} value={note} />

const styles = StyleSheet.create({
  input: {
    height: 40,
    width: 200,
    margin: 12,
    borderWidth: 1,
    padding: 10,
  }
});
```

Local Image

```
import expenseImage from './assets/expense.png';

const EXPENSE = Image.resolveAssetSource(expenseImage).uri

<Image source={{uri: EXPENSE }} />
```

CashBook Application



https://reactnative.dev/docs/textinput

https://reactnative.dev/docs/switch

- Mock Data แล้วสร้าง Component ListItem.js เพื่อแสดงผลข้อมูลรายรับรายจ่าย ให้ถูกต้อง
- สร้าง Component ใหม่ชื่อ AddItemModal โดยจะแสดงเมื่อกดปุ่ม Add
- ภายใน Modal แสดงแบบฟอร์ม กรอก จำนวนเงิน, Note ประเภท รายรับ หรือรายจ่าย
- เมื่อกด Ok ให้นำข้อมูล รายรับ รายจ่าย มาแสดงผลใน หน้าหลัก

3rd party library

- https://www.npmjs.com/package/react-native-select-dropdown
- ติดตั้ง package เพิ่มเติมโดยใช้คำสั่ง
  - npm install react-native-select-dropdown
- เปลี่ยน Toggle รายรับ รายจ่าย ให้เป็น Select DropDown

Editable CashBook

- เพิ่มปุ่ม Delete เพื่อลบข้อมูลใน List
  - เมื่อกดให้แจ้งเตือนว่าต้องการลบหรือไม่ และทำการลบข้อมูลออกจาก List
- เพิ่มปุ่ม Edit เพื่อแก้ไขข้อมูลใน List ได้
  - เมื่อกดให้แสดง Popup AddItemModal.js และแก้ไขข้อมูล

HTTP Request

วิธีการส่งข้อมูลไปยัง Server

- Fetch
  - เป็น Library พื้นฐานที่ติดตั้งมาโดยอัตโนมัติ
  - มีข้อเสียคือต้องจัดการมากกว่า เช่น การ convert message หรือการ handle error
- Axios
  - 3rd party library ช่วยในการส่งข้อมูลทำได้ง่ายยิ่งขึ้น

```
npm install axios
```

ตัวอย่างการใช้งาน GET

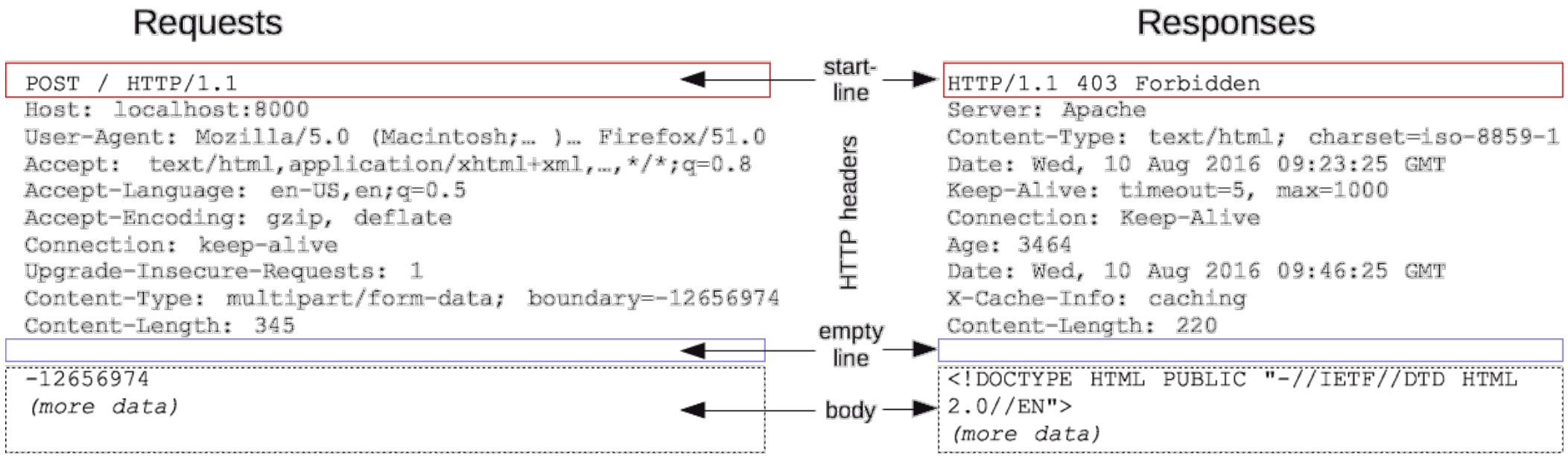
```
import axios from 'axios';
const baseUrl = 'https://reqres.in';

// Passing configuration object to axios
axios({
  method: 'get',
  url: `${baseUrl}/api/users/1`,
}).then((response) => {
  console.log(response.data);
});



// Invoking get method to perform a GET request
axios.get(`${baseUrl}/api/users/1`).then((response) => {
  console.log(response.data);
});
```

การใช้งานผ่าน Async/Await

```
import axios from 'axios';
const baseUrl = 'https://reqres.in';

// Passing configuration object to axios
const fetchUser = async () => {
  const configurationObject = {
    method: 'get',
    url: `${baseUrl}/api/users/1`,
  };
  const response = await axios(configurationObject);
  console.log(response.data);
};

// Invoking get method to perform a GET request
const fetchUser = async () => {
  const url = `${baseUrl}/api/users/1`;
  const response = await axios.get(url);
  console.log(response.data);
};
```

Authentication

- การทำ Authentication ส่วนมากจะทำผ่าน API Token
- API Token จะถูกส่งไปยัง server ทุกครั้งเพื่อยืนยันว่า request ส่งมาจากผู้ใช้งานที่ลงทะเบียนแล้ว



Authentication

```
axios.get('http://<Server Endpoint API>/', {
    headers: {
      'Authorization': `Bearer ${props.token}`
    }
})
```

Navigation

- การเปลี่ยนหน้าแสดงผลของโปรแกรม
- ติดตั้ง package เพิ่มเติม

```
$ npm install @react-navigation/native @react-navigation/native-stack

$ expo install react-native-screens react-native-safe-area-context
```

Navigation Exercise

- สร้างไฟล์ HomeScreen.js แล้ว Copy Code จาก App.js มาทั้งหมด
  - เปลี่ยนชื่อ export default function App()
    เป็น export default function HomeScreen(props)
  - แก้ไข renderItem
    - <TouchableOpacity onPress={() => {props.navigation.navigate('DetailScreen', { item:.item })}}>

Navigation Exercise (2)

- สร้างไฟล์ DetailScreen.js

```
import { useEffect } from 'react';
import { Text, View } from 'react-native';

export default function DetailScreen(props) {
  useEffect(() => {
    console.log(props.route.params.item)
  }, [props])
  return (
    <View>
      <Text>Detail Screen</Text>
      <Text>{props.route.params.item.note}</Text>
    </View>
  )
}
```

Navigation Exercise (3)

- แก้ไขไฟล์ App.js

```
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import HomeScreen from './HomeScreen';
import DetailScreen from "./DetailScreen";

const Stack = createNativeStackNavigator();

export default function App() {

  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="HomeScreen"
          component={HomeScreen}
          options={{ title: 'Home' }}
        />
        <Stack.Screen name="DetailScreen" component={DetailScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

สร้าง Mobile Application จาก API ต่อไปนี้

- https://dash.valorant-api.com/
- https://github.com/jackfperryjr/mog?tab=readme-ov-file#readme
- https://bymykel.github.io/CSGO-API/
- https://tcgdex.dev/rest
- https://www.adsbdb.com/
- https://isrostats.in/apis
- https://disease.sh/docs/#/
- https://hp-api.onrender.com/

สร้าง Mobile Application จาก API ต่อไปนี้

- https://docs.api.jikan.moe/#section/Information
- https://openlibrary.org/developers/api
- https://www.fruityvice.com/doc/index.html
- https://disneyapi.dev/docs/
- https://digimon-api.vercel.app/
- https://github.com/SpEcHiDe/IMDbOT/wiki