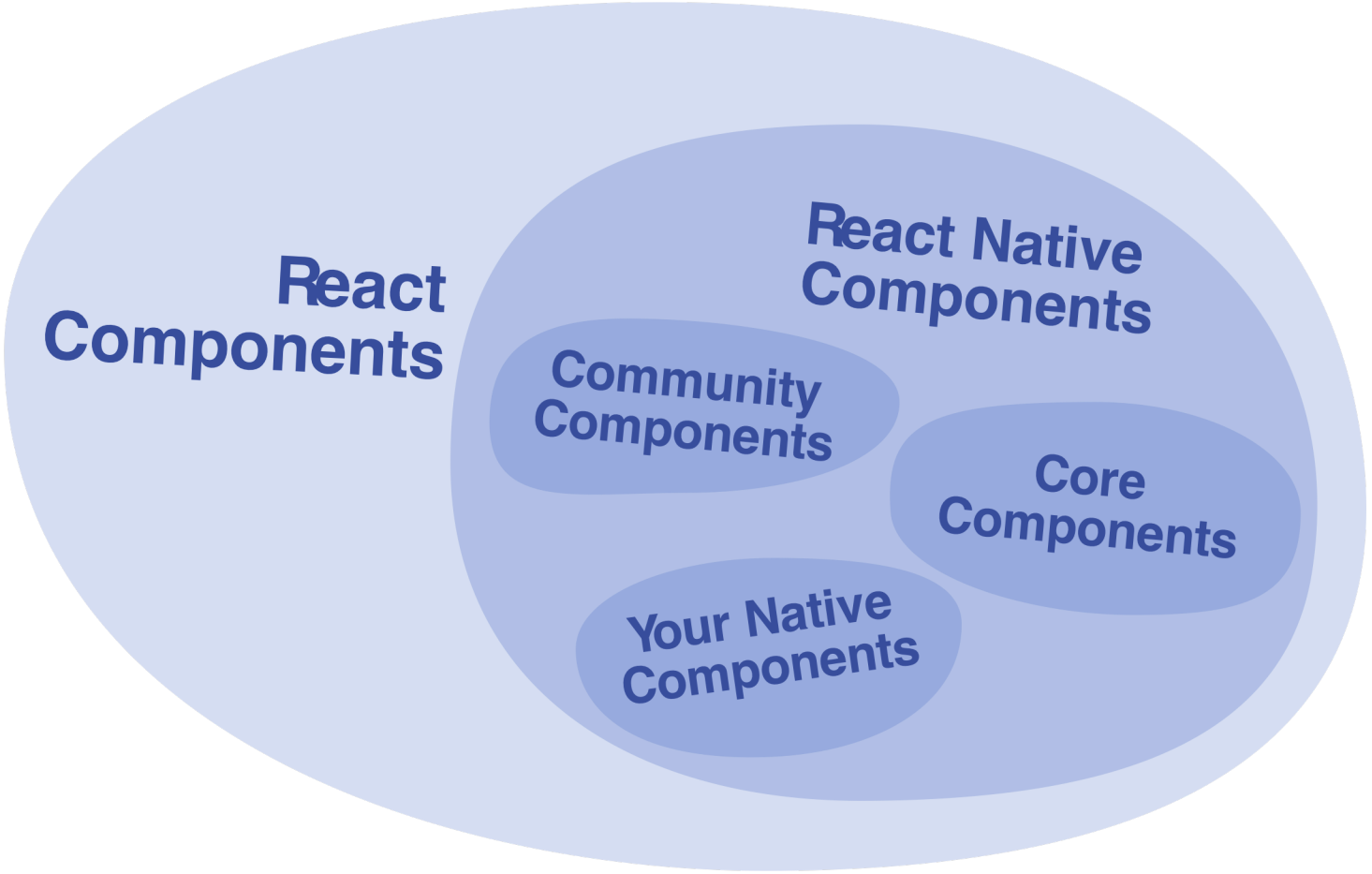
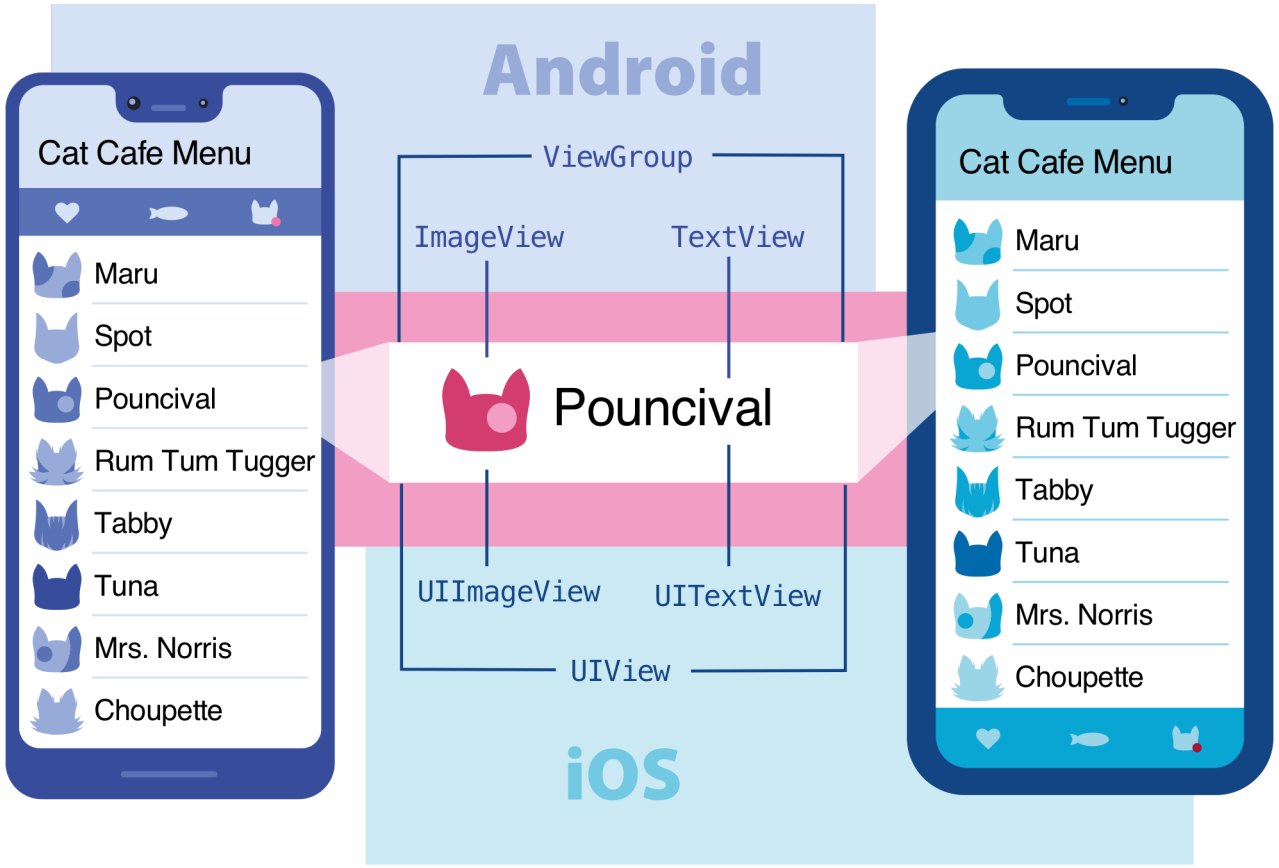
06016323 Mobile Device Programming

CHAPTER 5 : CORE COMPONENTS (PART I)

Core Components and Native Components



Views and mobile development



Native Components

- React Native ทำการสร้างมุมมองในส่วน Android และ iOS ที่สอดคล้องกันสำหรับ Component ต่าง ๆ ที่เลือกใช้ในการพัฒนา เนื่องจาก Component ของ React Native จะสนับสนุนและสามารถทำให้แอพพิเคชันที่พัฒนาบน Android และ iOS แสดงมุมมองเดียวกัน จึงทำให้แอพพิเคชันที่ถูกพัฒนาโดย React Native นั้นมีโครงสร้าง การแสดงผล และทำงานเหมือนกับแอพพิเคชันอื่น ๆ เราเรียก Component ที่สนับสนุนแพลตฟอร์มเหล่านี้ว่า **Native Components.**

Core Components

- React Native สนับสนุนและจัดเตรียม Components ให้กับผู้พัฒนา อยู่หลาย Components ซึ่งเรียกว่า Core Components มีดังนี้
  - Basic Components
  - User Interface
  - List Views
  - iOS-specific
  - Android-specific
  - Others

Basic Components

- View
  - การแสดงผลทางหน้าจอ ซึ่งเป็นส่วนติดต่อกับผู้ใช้ หรือ UI
- Text
  - การแสดงผลในรูปแบบตัวอักษรต่าง ๆ เป็นหลัก
- Image
  - การแสดงผลรูปภาพที่ประกอบในแอพพิเคชัน หรือภาพหลังของแอพพิเคชัน
- TextInput
  - การจัดการรับค่าข้อมูลทางคีย์บอร์ดที่ปรากฏอยู่ในแอพพิเคชัน
- ScrollView
  - จัดการคอนเทรนเนอร์สำหรับเลื่อนจอภาพที่มีการแสดงผลเกินขนาดของจอภาพบนอุปกรณ์เคลื่อนที่ ซึ่งมักใช้คู่กับ component อื่น เช่น View Component
- StyleSheet
  - มีลักษณะการทำงานเหมือน Layer ที่ส่งต่อไปยัง CSS Stylesheets.

View Component

- องค์ประกอบพื้นฐานที่สำคัญที่สุดสำหรับการสร้าง User Interface View คือ คอนเทนเนอร์ที่รองรับเลย์เอาต์ด้วย flexbox, style, some touch handling, และ accessibility View component จะทำการปรับมุมมองที่สร้างให้สามารถเทียบเท่ากับแพลตฟอร์มใดก็ตามที่ React Native กำลังทำงานอยู่ไม่ว่าจะเป็น UIView, , android.view ฯลฯ
- *View* ได้รับการออกแบบให้สามารถซ้อนอยู่ภายในมุมมองอื่น ๆ ได้ และสามารถมีจำนวนที่ซ้อนกันได้ ตั้งแต่ 0 จนถึงหลายประเภทแบบใดก็ได้

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

const ViewBoxesWithColorAndText = () => {
  return (
    <View
      style={{
        flexDirection: "row",
        height: 100,
        padding: 20
      }}
    >
      <View style={{ backgroundColor: "blue", flex: 0.3 }} />
      <View style={{ backgroundColor: "red", flex: 0.5 }} />
      <Text>Hello World!</Text>
    </View>
  );
};

export default ViewBoxesWithColorAndText;
```

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

class App extends Component {
  render() {
    return (
      <View
        style={{
          flexDirection: "row",
          height: 100,
          padding: 20
        }}
      >
        <View style={{ backgroundColor: "blue", flex: 0.3 }} />
        <View style={{ backgroundColor: "red", flex: 0.5 }} />
        <Text>Hello World!</Text>
      </View>
    );
  }
}

export default App;
```

Text Component

- ข้อความรองรับ nesting, styling และ touch ตัวอย่างสำหรับการใช้ Text Component จะแสดงถึงการใช้ nested title และ body text ซึ่งจะสืบทอดจาก fontFamilyfrom styles.baseText ซึ่งจะสามารถเพิ่มเติม Styles ได้ตามที่ต้องการ

ข้อจำกัดของ Text Component

- สำหรับ React Native มีความเข้มงวดมาก: การใส่ข้อความทั้งหมดต้องอยู่ภายใต้คอมโพเนนต์ <Text> คุณไม่สามารถใส่ข้อความได้โดยตรงภายใต้ <View>

```
// BAD: will raise exception, can't have a text
node as child of a
<View> <View>                    </View>
// GOOD
<View> <Text>                    </Text> </View>
```

```
<View>
 <MyAppText>
        Text styled with the default font for the entire application
</MyAppText>
<MyAppHeaderText>
       Text styled as a header
</MyAppHeaderText>
</View>
```

```
class MyAppHeaderText extends Component { render() {
    return (
        <MyAppText>
            <Text style={{ fontSize: 20 }}>
                {this.props.children}
            </Text>
        </MyAppText> ); } }
```

```
<Text style={{ fontWeight: 'bold' }}>
I am bold
<Text style={{ color: 'red' }}>and red</Text>
</Text>
```

Image Component

- คอมโพเนนต์สำหรับการแสดงภาพประเภทต่าง ๆ รวมถึงภาพจากเครือข่าย สื่อหรือรูปภาพแบบคงที่ (static resources) ภาพหรือสื่อแบบชั่วคราว (temporary local images) และภาพจากดิสก์ในเครื่องเช่นม้วนฟิล์ม (camera roll)
- การกำหนดสำหรับระบุที่อยู่ของสื่อหรือรูปภาพ เพื่อทำการดึงและแสดงภาพจากที่จัดเก็บในตัวเครื่อง จะทำการระบุไว้ในโครงร่าง ของ “data:” ในรูปแบบของ URI Scheme ส่วนการระบุ สื่อหรือรูปภาพ จากเครือข่าย จะใช้โครงสร้างเหมือนดังกล่าวข้างต้น

```
class DisplayAnImage extends Component {
  render() {
    return (
      <View style={styles.container}>
        <Image
          style={styles.tinyLogo}
          source={require('@expo/snack-static/react-native-logo.png')}
        />
        <Image
          style={styles.tinyLogo}
          source={{uri: 'https://reactnative.dev/img/tiny_logo.png'}}
        />
        <Image
          style={styles.logo}
          source={{uri:
'
BACARB+2/ab8BEeQNhFi6WSYzYLYudDQYGBgYGBgYGBgYGBgZmcvDqYGBgmhivGQYGBgYGBgYGBgYGBgbmQw+P/eMrC5UTVAAAAABJRU5ErkJggg==='}}
        />
      </View>
    );
  }
}
```

TextInput Component

- คือ ส่วนประกอบพื้นฐานสำหรับการป้อนข้อความลงในแอพพิเคชันผ่านคีย์บอร์ด Props จะทำการจัดเตรียมและกำหนดค่าสำหรับคุณสมบัติต่าง ๆ เช่นการแก้ไขอัตโนมัติ การใช้อักษรตัวพิมพ์ใหญ่อัตโนมัติ ข้อความประเภท placeholder และแป้นพิมพ์ประเภทต่าง ๆ เช่นแป้นพิมพ์ตัวเลข

- การใช้งานขั้นพื้นฐานส่วนมาก คือการเลื่อน TextInput ปรับปรุงข้อความตามเหตุการณ์ ที่อยู่onChangeText เพื่ออ่านอินพุตของผู้ใช้ นอกจากนี้ยังมีเหตุการณ์อื่น ๆ เช่น onSubmitEditing และ onFocus ที่สามารถปรับปรุงได้ตามเหตุการณ์ต่าง ๆ

```
const UselessTextInputMultiline = () => {
  const [value, onChangeText] = React.useState('Useless Multiline Placeholder');

  // If you type something in the text box that is a color, the background will change to that
  // color.
  return (
    <View
      style={{
        backgroundColor: value,
        borderBottomColor: '#000000',
        borderBottomWidth: 1,
      }}>
      <UselessTextInput
        multiline
        numberOfLines={4}
        onChangeText={text => onChangeText(text)}
        value={value}
      />
    </View>
  );
}
```

ScrollView Component

- ScrollView คอมโพเนนท์ เป็นคอมโพเนนท์ ที่ถูกรวบไว้ในกับระบบ ’responder’ ซึ่งเป็นระบบสัมผัส
- คอมโพเนนท์มีข้อจำกัดสำหรับการกำหนดค่าความสูงของ ScrollViews โดยตัวคอมโพเนนท์จะไม่ขึ้นอยู่กับคอนเทนเนอร์ที่มีขอบเขต ดังนั้นการกำหนดความสูงควรกำหนดตามขอบเขตของความสูงของ ScrollViews ไม่ควรกำหนดที่ขึ้นตรงกับ View หรือ ตรวจสอบให้แน่ใจว่าตัว parent tag ครอบคลุมขอบเขตความสูงที่กำหนด นอกจากนี้ไม่ควรลืมที่จะเปลี่ยนค่า {flex: 1} ลงใน stack view หากลืมอาจจะทำให้เกิด errors.
- ยังไม่สนับสนุน การแสดงผลในลักษณะ Blocking จาก responder อื่น ๆ
- <ScrollView> และ <FlatList>

ScrollView

- แท็ก หรือ คำสั่งที่อยู่ภายใต้ ScrollView จะถูก renders พร้อมกันเพียงครั้งเดียว ซึ่งมีข้อเสียด้านประสิทธิภาพของแอพพิเคชัน
- ตัวอย่างเช่น หากแอพพิเคชันมีรายการที่ยาวมากที่ต้องการแสดงผลและมีเนื้อหาหลายหน้าจอ การสร้างคอมโพเนนท์ JS และ Native view สำหรับประมวลผลทุกสิ่งพร้อมกันเพียงครั้งเดียว ซึ่งส่วนใหญ่อาจไม่ได้แสดงด้วยซ้ำ มีผลทำให้การแสดงผลช้าและอาจต้องมีการใช้หน่วยความจำเพิ่มมากขึ้น

```
import React from 'react';
import { StyleSheet, Text, SafeAreaView, ScrollView } from 'react-native';
import Constants from 'expo-constants';

const App = () => {
  return (
    <SafeAreaView style={styles.container}>
      <ScrollView style={styles.scrollView}>
        <Text style={styles.text}>
          Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
          eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
          minim veniam, quis nostrud exercitation ullamco laboris nisi ut
          aliquip ex ea commodo consequat. Duis aute irure dolor in
          reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
          pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
          culpa qui officia deserunt mollit anim id est laborum.
        </Text>
      </ScrollView>
    </SafeAreaView>
  );
}
```

FlatList

- ในกรณีที่ เราต้องการให้การประมวลผลแยก renders สำหรับหลายรายการ หลายคอลัมน์ การโหลดที่มีลักษณะ Infinity หรือคุณสมบัติอื่น ๆ ที่ต้องการประมวลผลในลักษณะนี้
- ดังนั้น FlatList เข้ามามีบทบาท ในกรณีที่ แอพเคชันต้องการแสดงรายการอย่างต่อเนื่อง เมื่อผู้ใช้กำลังจะเพิ่มและลบรายการที่ต้องการเลื่อนออกจากหน้าจอ
- ต้องการประหยัดหน่วยความจำ
- ต้องการประหยัดเวลาในการประมวลผล

```
import React from 'react';
import { SafeAreaView, View, FlatList, StyleSheet, Text, StatusBar } from
'react-native';

const DATA = [
  {
    id: 'bd7acbea-c1b1-46c2-aed5-3ad53abb28ba',
    title: 'First Item',
  },
  {
    id: '3ac68afc-c605-48d3-a4f8-fbd91aa97f63',
    title: 'Second Item',
  },
  {
    id: '58694a0f-3da1-471f-bd96-145571e29d72',
    title: 'Third Item',
  },
];
```

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

const App = () => {
  const renderItem = ({ item }) => (
    <Item title={item.title} />
  );
```

```
return (
    <SafeAreaView style={styles.container}>
      <FlatList
        data={DATA}
        renderItem={renderItem}
        keyExtractor={item => item.id}
      />
    </SafeAreaView>
  );
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: StatusBar.currentHeight || 0,
  },
  item: {
    backgroundColor: '#f9c2ff',
    padding: 20,
    marginVertical: 8,
    marginHorizontal: 16,
  },
  title: {
    fontSize: 32,
  },
});

export default App;
```

StyleSheet Component

- การย้ายสไตล์ออกจากฟังก์ชันการแสดงผลจะทำให้เข้าใจโค้ดได้ง่ายขึ้น
- การตั้งชื่อสไตล์ที่เข้าใจง่ายเป็นวิธีที่ดีและควรทำ ทำให้เราสามารถเพิ่มความหมายให้กับคอมโพเนนท์ระดับล่างในฟังก์ชัน render

Method of StyleSheet Component

- **compose()**

```
static compose(style1: object, style2: object): object | array<object>
```

- **create()**

```
static create(obj: object): object
```

- **flatten()**

```
static flatten(style: array<object>): object
```

- **setStyleAttributePreprocessor()**

```
static setStyleAttributePreprocessor(property: string, process: (propValue: any)
```

Properties of StyleSheet Component

- (position: 'absolute', left: 0, right: 0, top: 0, bottom: 0) -> **absoluteFill**

```
Box1: {
    ...StyleSheet.absoluteFill,
    width: 100,
    height: 100,
    backgroundColor: 'blue'
```

```
box1: {
    position: 'absolute',
    top: 50,
    left: 50,
    width: 80,
    height: 100,
    backgroundColor: ‘blue'
```

Properties of StyleSheet Component

- **absoluteFill and absoluteFillObject**

```
box2: {
  ...StyleSheet.absoluteFill,
  top: 120,
  left: 50,
  width: 100,
  height: 100,
  backgroundColor: 'blue'
},
```

```
box3: {
  ...StyleSheet.absoluteFillObject,
  top: 120,
  left: 120,
  width: 100,
  height: 100,
  backgroundColor: 'green'
```

- **hairlineWidth**

User Interface Component : Button

- Button คอมโพเนนท์ เป็นส่วนประกอบพื้นฐานที่สามารถใช้ได้ในทุกแพลตฟอร์ม
- หาก Button ที่อยู่ในแอพพิเคชันมีปัญหา ในการใช้งาน แนะนำให้เปลี่ยนไปใช้ TouchableOpacity or TouchableWithoutFeedback.

```
<Button
onPress={onPressLearnMore}
title="Learn More"
color="#841584"
accessibilityLabel="Learn more
about this purple button" />
```

Prosperities : Button

- **onPress**
- **title**
- accessibilityLabel
- color
- disabled
- hasTVPreferredFocus (TV)
- nextFocusDown (Android) (TV)
- nextFocusForward (Android) (TV)
- nextFocusLeft (Android) (TV)
- nextFocusRight (Android) (TV)
- nextFocusUp (Android) (TV)
- testID
- touchSoundDisabled (Android)

User Interface Component : Switch

- คอมโพเนนท์ สำหรับการจัดการเรื่อง Boolean
- คอมโพเนนท์ที่มีการควบคุมที่ต้องการการเรียกกลับ (onValueChange) เพื่อปรับปรุงค่าที่อยู่ใน prop เพื่อให้คอมโพเนนท์สามรถสะท้อนหรือส่งต่อการกระทำของผู้ใช้กำหนด หากค่า prop ไม่ได้รับกาปรับปรุงค่า คอมโพเนนท์จะยังคงแสดงผลค่า prop ที่ให้มาแทน (Default)

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

const App = () => {
  const [isEnabled, setIsEnabled] = useState(false);
  const toggleSwitch = () => setIsEnabled(previousState => !previousState);

  return (
    <View style={styles.container}>
      <Switch
        trackColor={{ false: "#767577", true: "#81b0ff" }}
        thumbColor={isEnabled ? "#f5dd4b" : "#f4f3f4"}
        ios_backgroundColor="#3e3e3e"
        onValueChange={toggleSwitch}
        value={isEnabled}
      />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: "center",
    justifyContent: "center"
  }
});

export default App;
```

Prosperities : Switch

- disabled
- ios_backgroundColor
- onChange
- onValueChange
- thumbColor
- trackColor
- value