REACT NATIVE

ThS. Dương Hữu Thành,
Khoa CNTT, Đại học Mở TP.HCM,
thanh.dh@ou.edu.vn.

Nội dung chính

- Giới thiệu
- Props & State
- Vòng đời React Component
- Style
- Các điều khiển thông dụng
- React Navigation
- React Animation

Giới thiệu

- React Native là một framework Javascript để xây dựng ứng dụng di động cho iOS và Android.
- Nó sử dụng các native component thay vì sử dụng web component cho ứng dụng.

Giới thiệu

- Đa nền tảng: làm việc cả iOS và Android
- Mã nguồn viết trong React Native sẽ được biên dịch thành native code cho cả hai hệ điều hành.
- Chỉ sử dụng Javascript để phát triển ứng dụng di động.
- Cộng đồng sử dụng lớn.

Ứng dụng đầu tiên

- Cài đặt NodeJS
- Cài công cụ tạo project

```
npx create-expo-app <project-name> --template blank
```

- Thực thi: npx expo start
- Thực thi tuỳ chọn: s (chạy trên expo client)

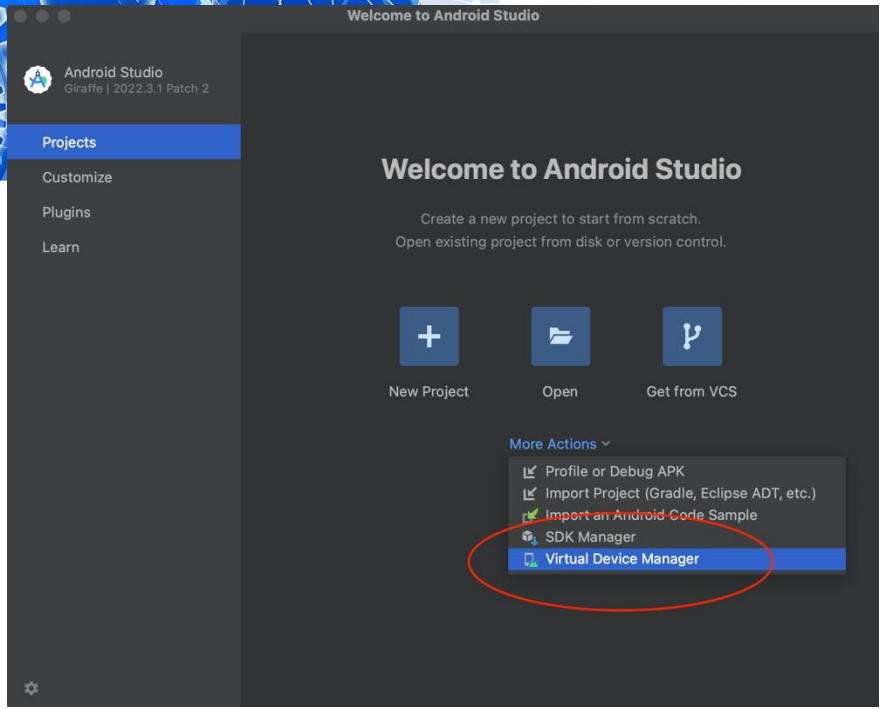
Ứng dụng đầu tiên

- Cài ứng dụng export client trên điện thoại và quét mã QR này để mở ứng dụng.

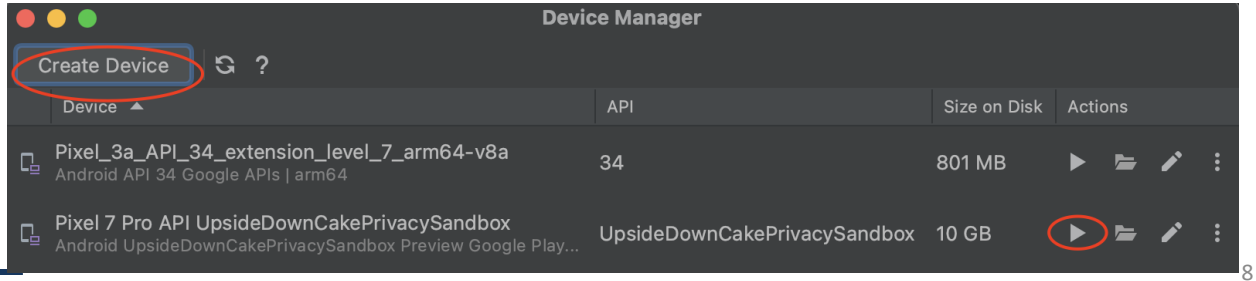
```
› Metro waiting on http://localhost:8081
› Scan the QR code above to open the proje

› Using development build
› Press s | switch to Expo Go

› Press a | open Android
› Press i | open iOS simulator
```

Ứng dụng đầu tiên

- Để chạy trên máy giả lập Android
  - Cài Android Studio
  - Tạo AVD
  - Chạy máy giả lập Android


Welcome to Android Studio
Android Studio
Giraffe | 2022.3.1 Patch 2
Projects
Customize
Plugins
Learn
Welcome to Android Studio
Create a new project to start from scratch.
Open existing project from disk or version control.
New Project
Open
Get from VCS
More Actions
Profile or Debug APK
Import Project (Gradle, Eclipse ADT, etc.)
Import an Android Code Sample
SDK Manager
Virtual Device Manager


10:00
Sat, Oct 14
Play Store
Gmail
Photos
YouTube

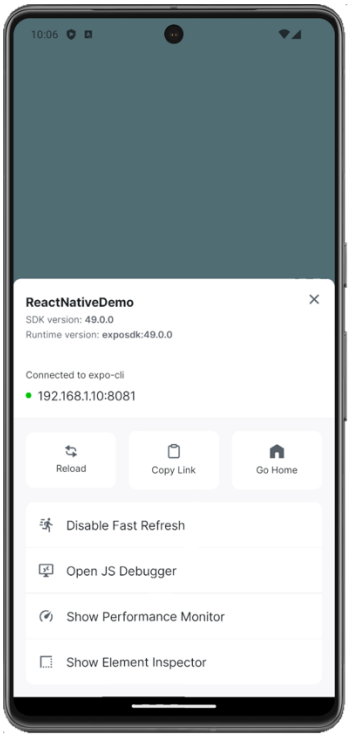

Device Manager
Create Device
Device
API
Size on Disk
Actions
Pixel_3a_API_34_extension_level_7_arm64-v8a
Android API 34 Google APIs | arm64
34
801 MB
Pixel 7 Pro API UpsideDownCakePrivacySandbox
Android UpsideDownCakePrivacySandbox Preview Google Play...
UpsideDownCakePrivacySandbox
10 GB

Ứng dụng đầu tiên

- Chạy lại project lần lượt thực thi lệnh
  - npm start
  - s
  - a

Debugging

- Trên Android: Command + M
- Trên iOS: Command + D



Props

- Props là các đối số được truyền vào các component giống như các thuộc tính.
- Giá trị của props tại component con không được thay đổi.
- Nếu component có constructor thì props phải được truyền vào constructor và truyền cho React. Component bằng phương thức super.

Props

```
export default App = () => {
  const users = ["Duong Huu Thanh", "Duong Le"]
  return (
    <View>
      <Child users={users} />
    </View>
  );
}
```

```
export default Child = (props) => {
  return <View>
    {props.users.map(u => <Text>{u}</Text>)}
  </View>
}
```

State

- Đối tượng state lưu giá trị các thuộc tính của component.
- State chỉ có hiệu lực trong phạm vi component, giá trị state có thể thay đổi bất cứ khi nào bằng phương thức this.setState().
- Khi đối tượng state thay đổi thì component sẽ nạp lại (re-render).
- Truy cập thuộc tính của đối tượng state:
  - this.state.propertyName

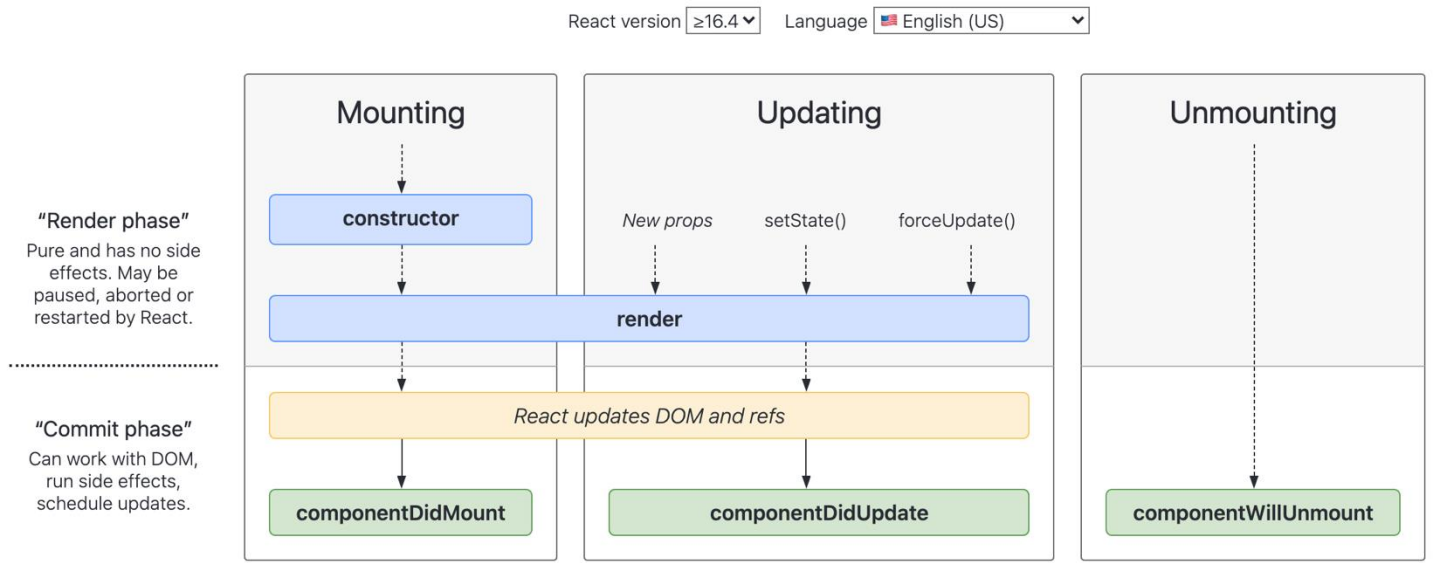
State

```
export default App = () => {
  const [name, setName] = useState("Demo");

  return (
    <View style={styles.container}>
      <TextInput style={styles.input} value={name}
        onChangeText={v => setName(v)}
        placeholder='Enter name...' />
      <Text>Hello {name}</Text>
    </View>
  );
}
```

Vòng đời React Component

- Mỗi component của React có 3 giai đoạn chính
  - Mounting: khi đặt một component vào DOM.
  - Updating: khi component được update. Một component được update khi có thay đổi trong props hoặc state.
  - Unmounting: khi một component được gỡ khỏi DOM.

Vòng đời React Component



Nguồn ảnh: https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

Style

- Tất cả các thành phần có thuộc tính style, thuộc tính name và value tương tự CSS, tên phải dùng lạc đà.
- Nếu style phức tạp có thể sử dụng StyleSheet.create

Style

- React Native thường sử dụng Flexbox để bố cục các thành phần linh hoạt theo kích thước màn hình.
  - flexDirection: column|row
  - justifyContent: center|flex-start|flext-end|space-around|space-between.
  - alignItems: center|flex-start|flex-end|stretched

Style

```
export default App = () => {
  return (
    <View style={myStyles.container}>
      <Text style={{ fontSize: 36, color: "red" }}>
        Hello World!!!
      </Text>
    </View>
  );
}
const myStyles = StyleSheet.create({
  container: {
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "lightblue",
    flex: 1
  }
})
```

Style

- Position
  - relative (mặc định): sử dụng top, left, right, bottom điều chỉnh vị trí, không ảnh hưởng các thành phần khác trong bố cục.
  - absolute

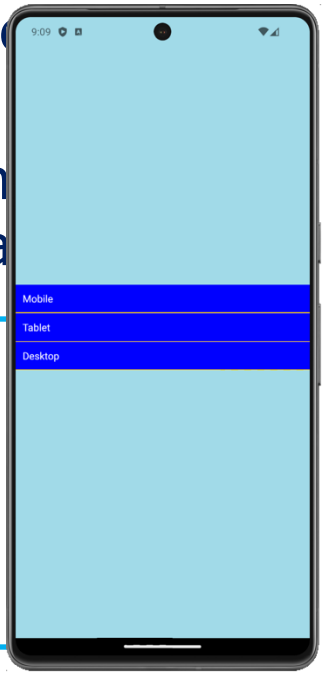
View

- View là thành phần container có thể chứa các thành phần khác.
- View có thể bổ cục với Flexbox, Style.
- ListView

View

- ScrollView hiển thị các thành phần th
dọc hoặc ngang có thể cuộn được.
- Nó phù hợp hiển thị số lượng item nh
để hiển thị số lượng lớn hơn dùng Fla

```
<View style={myStyles.container}>
 <ScrollView>
 {cates.map(c => <Text style={myStyles.item}
key={c.id}>{c.name}</Text>)}
 </ScrollView>
</View>
```



FlatList

- FlatList là thành phần nhiều nội dung có thể cuộn được.
  - keyExtractor chỉ định cách lấy định danh cho item.

```
const cates = [{
  "id": 1,
  "name": "Mobile"
}, {
  "id": 2,
  "name": "Tablet"
}, {
  "id": 3,
  "name": "Desktop"
}]
```

```
<FlatList data={cates} renderItem={(c) => <Text
key={c.item.id}>{c.item.name}</Text>} />
```

```
<FlatList data={cates} renderItem={({ item: { id, name } }) => <Text
key={id}>{name}</Text>} />
```

FlatList

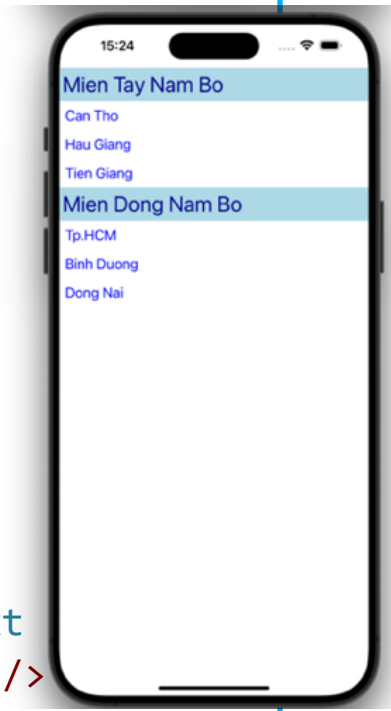
- Một số thuộc tính xử lý lazy loading:
  - onEndReached: chỉ định hàm callback được gọi khi người dùng cuộn đến gần cuối danh sách.
  - onEndReachedThreshold: nhận giá trị từ 0 đến 1 xác định khi nào hàm onEndReached sẽ được gọi, chẳng hạn nếu giá trị của nó là 0.5, tức là hàm trong onEndReached sẽ được gọi khi cuộn 50% cuối của danh sách.

Section List

- SectionList cho phép hiển thị danh sách phức tạp, có cấu trúc rõ ràng, có thể chia thành nhiều vùng hiển thị khác nhau.
  - sections: chỉ định mảng chứa các vùng, mỗi vùng là một đối tượng có hai thuộc tính là title và data.
  - renderItem: chỉ định cách hiển thị mỗi item trong một vùng.
  - renderSectionHeader: chỉ định cách hiển thị tiêu đề mỗi vùng.

Section List

```
const data = [{
  title: "Mien Tay Nam Bo",
  data: ["Can Tho", "Hau Giang", "Tien Giang"]
}, {
  title: "Mien Dong Nam Bo",
  data: ["Tp.HCM", "Binh Duong", "Dong Nai"]
}];
function App() {
  return (
    <SectionList style={{ marginTop: 60 }}
sections={data} renderItem={({item}) => <Text
style={styles.item}>{item}</Text>}
      renderSectionHeader={({section}) => <Text
style={styles.header}>{section.title}</Text>} />
  );
}
```



Text

- Text là thành phần hiển thị văn bản.
- Các thuộc tính quan trọng
  - numberOfLines: số dòng hiển thị văn bản.
  - ellipsizeMode: chỉ định cách hiển thị khi văn bản vượt quá numberOfLines

Touchable

- Touchable cho phép tạo nút bấm theo phong cách riêng và cung cấp hiển thị phản hồi khi thực hiện cử chỉ chạm.
  - Sử dụng TouchableHighlight khi muốn tạo các nút bấm và liên kết đến Web.
  - TouchableOpacity cung cấp phản hồi bằng cách giảm giá trị opacity của nút bấm.
  - TouchableWithoutOpacity cung cấp khả năng xử lý cử chỉ chạm nhưng không hiển thị bất kỳ phản hồi nào.

Image

- Image thành phần hiển thị hình ảnh từ URI chỉ định thông qua thuộc tính source.
- Thuộc tính resizeMode
  - cover
  - contain
  - stretch
  - repeat
  - center

Image

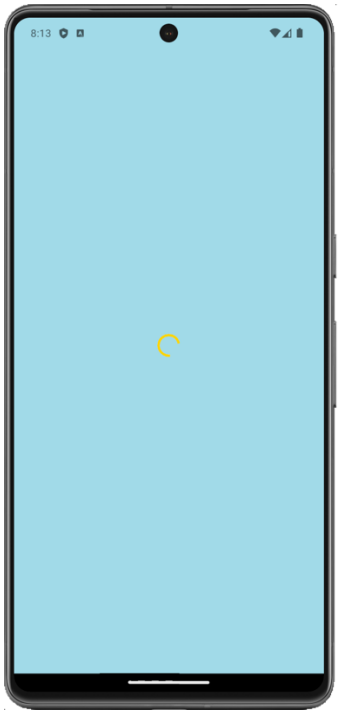
- Để thiết lập ảnh nền cho ứng dụng sử dụng ImageBackground có các thuộc tính giống với Image, các thành phần con bên trong đó sẽ hiển thị chồng trên ảnh nền này

```
<View style={{ flex: 1 }}>
  <ImageBackground style={styles.container}
     source={require("./images/bg.jpeg")}>
    <Text style={styles.title}>REACT NATIVE</Text>
  </ImageBackground>
</View>
```

ActivityIndicator

- Thành phần ActivityIndicator chỉ định trạng thái nạp (loading).
- Nó có các thuộc tính như View và
  - anmating: ẩn hoặc hiện indicator
  - color: màu của spinner
  - size: kích thước spinner (small hoặc large)

```
<ActivityIndicator color="gold"
                   size="large" />
```



TextInput

- TextInput để tạo ô nhập liệu.
- Các thuộc tính
  - value: giá trị đang có trên ô nhập liệu.
  - onChangeText: sự kiện gọi khi văn bản trên ô nhập liệu thay đổi.
  - onSubmitEditting: sự kiện gọi khi người dùng bấm Enter trong ô nhập liệu.

TextInput

- Một số thuộc tính khác
  - autoCapitalize: none|sentences|words|characters
  - autoCorrect: true|false
  - editable: enable hoặc disable ô nhập liệu
  - keyboardType: default|numeric|email-address|phone-pad
  - multiple: ô nhập liệu nhiều dòng
  - placeholder
  - placeholderTextColor
  - returnKeyType: done|go|next|search|send

Button

- <Button />
  - onPress: xử lý sự kiện click.
  - title: văn bản hiển thị cho button.
  - color: màu nền (android) hoặc màu chữ (iOS)
  - disabled: vô hiệu hoá tương tác.
  - assessibilityLabel: hiển thị văn bản trợ năng cho khiếm thị.

Switch

- <Switch />
  - disabled: vô hiệu hoá
  - thumbColor: màu nút chuyển đổi
  - ios_backgroundColor:
  - value: giá trị hiện tại của switch.
  - onValueChange: sự kiện thay đổi trạng thái.

```
export default Test = () => {
  const [status, setStatus] = useState(false);
  return <Switch value={status} onValueChange={setStatus} />
}
```

Alert

- Alert dùng hiển thị một hộp thoại với tiêu đề (title) và nội dung thông điệp (message) được chỉ định.

```
Alert.alert("Welcome",
"Welcome to my team!")
```

Alert

```
Alert.alert("Welcome", "Welcome to my team!", [{
  text: "Response",
  onPress: () => { Alert.alert("Thanks for your response!"); },
  style: "default"
}, {
  text: "Cancel",
  onPress: () => { Alert.alert("Bye!"); },
  style: "destructive"
}])
```

AsyncStorage

- AsyncStorage hệ thống lưu trữ theo dạng key/value, dữ liệu được lưu bất đồng bộ và không được mã hoá.

```
npm install @react-native-async-storage/async-storage
```

- AsyncStorage chỉ làm việc với dữ liệu chuỗi.

AsyncStorage

- Các phương thức quan trọng
  - setItem
  - getItem
  - removeItem
  - mergeItem

React Navigation

- Native Stack Navigator (NSN) của React Navigation cung cấp cách thức để chuyển đổi qua lại giữa các màn hình ứng dụng và quản lý navigation history.

```
npm install @react-navigation/native
npm install react-native-screens
npm install react-native-safe-area-context
```

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

React Navigation

- Ví dụ

```
const Stack = createNativeStackNavigator();
const App = () => {
    return (
        <NavigationContainer>
            <Stack.Navigator initialRouteName="Home">
                <Stack.Screen name="Home" component={Home}
                              options={options} />
                <Stack.Screen name="Demo" component={Demo} />
            </Stack.Navigator>
        </NavigationContainer>
    )
}
```

React Navigation

- Tuỳ chỉnh header
  - headerStyle: thiết lập các style áp dụng cho View bọc header.
  - headerTintColor: thiết lập màu sắc cho tiêu đề trên header và tiêu đề nút quay lui khi chuyển màn hình.
  - headerTitleStyle: tuỳ chỉnh các style như fontFamily, fontWeight hoặc một số thuộc tính khác cho văn bản.

React Navigation

- Chuyển đổi giữa các màn hình

```
const Home = ({ navigation }) => {
    return (
        <View style={styles.container}>
            <Text style={styles.txt}>Trang chủ</Text>
            <TouchableOpacity
                  onPress={() => navigation.navigate('Demo')}>
                <Text>Đến trang demo</Text>
            </TouchableOpacity>
        </View>
    )
}
```

React Navigation

- Để truyền tham số vào route, ta có thể sử dụng thuộc tính initialParams của Screen như sau:

```
<Stack.Screen name="Demo" component={Demo}
              initialParams={params} />
```

- Ta có thể chỉ định tham số khi chuyển màn hình bằng tham số thứ hai của phương thức navigate

```
navigation.navigate('Demo', params);
```

React Navigation

- Ví dụ truyền tham số vào route

```
const Demo = ({ route, navigation }) => {
    const { id, name } = route.params;

    return (
        <View style={styles.container}>
            <Text style={styles.txt}>Trang Demo</Text>
            <Text style={{ fontSize: 30 }}>{id} - {name}</Text>
        </View>
    )
}
```

Tab Navigator

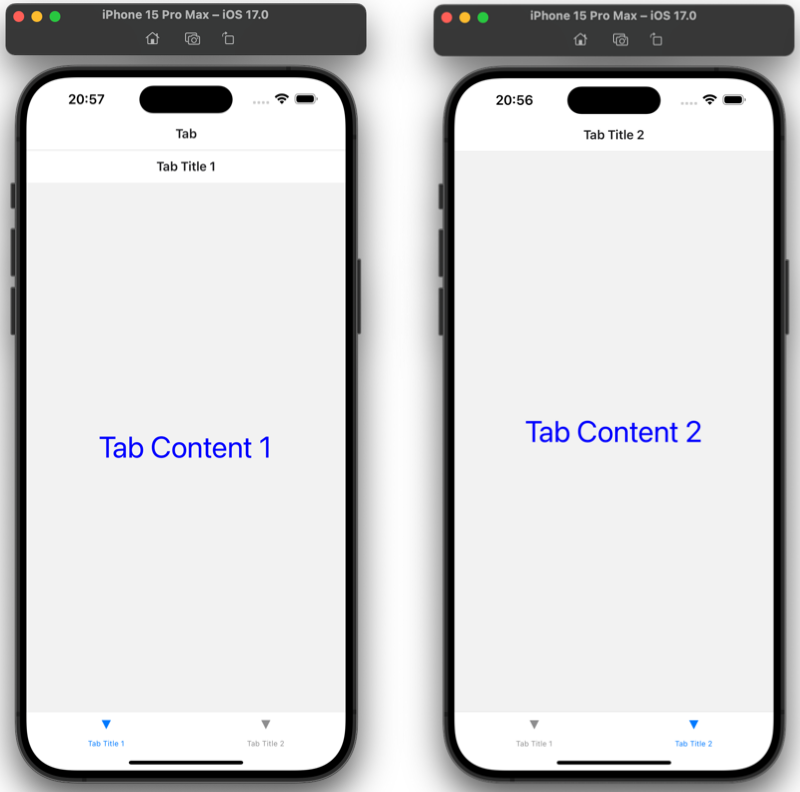
- Để sử dụng được navigator này, ta phải đảm bảm trong project đã cài thư viện "@react-navigation/native" và các dependency liên quan, rồi cài thư viện sau:

```
npm install @react-navigation/bottom-tabs
```

Tab Navigator

```
const Tab = createBottomTabNavigator();

const MainTab = () => {
    return (
        <Tab.Navigator screenOptions={{ title: 'Tab Demo' }}>
            <Tab.Screen name="Tab1" component={Tab1}
                        options={{ title: 'Tab Title 1' }} />
            <Tab.Screen name="Tab2" component={Tab2}
                        options={{ title: 'Tab Title 2' }} />
        </Tab.Navigator>
    )
}
```

Tab Navigator



Drawer Navigator

- Ta phải đảm bảm trong project đã cài thư viện "@react-navigation/native" và các dependency liên quan, rồi cài thư viện sau:

```
npm install @react-navigation/drawer
```

Drawer Navigator

- Ta cần cài đặt một số thư viện cho việc điều hướng này.

```
// Dùng cho expo managed project
npx expo install react-native-gesture-handler
react-native-reanimated

// Dùng cho bare React native project
npm install react-native-gesture-handler react-
native-reanimated
```

Drawer Navigator

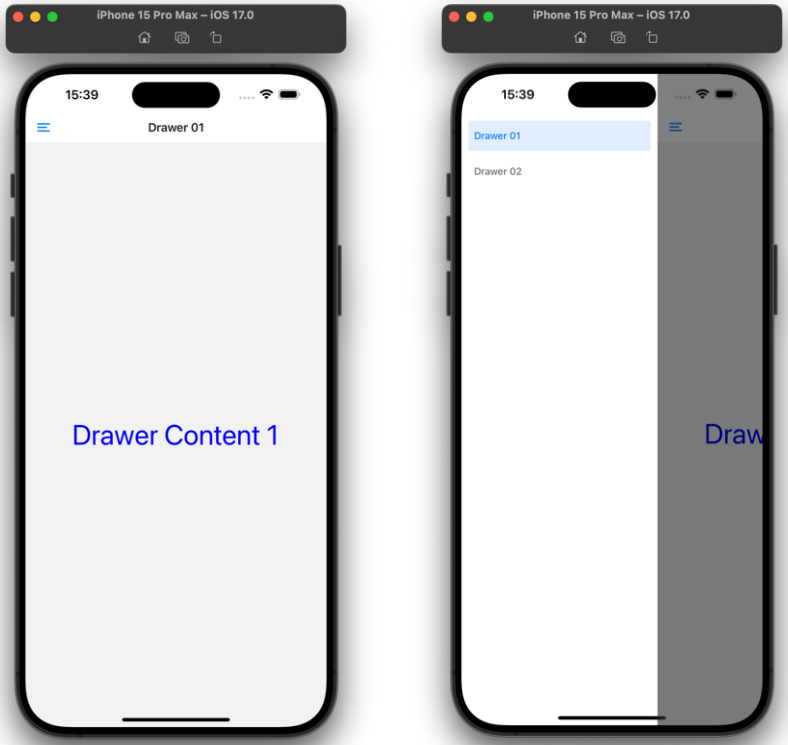
- Để sử dụng phiên bản mới nhất của reanimated, ta cần thêm plugin "react-native-reanimated/plugin" vào babel.config.js như sau:

```
module.exports = function (api) {
  api.cache(true);
  return {
    presets: ['babel-preset-expo'],
    plugins: ['react-native-reanimated/plugin']
  };
};
```

Drawer Navigator

```
const App = () => {
    return (
        <NavigationContainer>
            <Drawer.Navigator>
                <Drawer.Screen name="Drawer 01"
                               component={Drawer1} />
                <Drawer.Screen name="Drawer 02"
                               component={Drawer2} />
            </Drawer.Navigator>
        </NavigationContainer>
    )
}
```

Drawer Navigator



React Animation

- Điều chỉnh liên tục theo thời gian giá trị các thuộc tính liên quan kích thước thành phần, màu sắc, vị trí hoặc một thuộc tính style nào khác.
- React Native cung cấp hai API quan trọng để thực hiện hiệu ứng trong ứng dụng: Animated và LayoutAnimation.

Animated

- Cú pháp

```
Animated.timing({}).start(({finished}) => {

});
```

- Phương thức start() để bắt đầu hiệu ứng.
- Animated hỗ trợ sáu thành phần tạo hiệu ứng, bao gồm: View, Text, Image, ScrollView, FlatList và SectionList.

```
const AnimatedTest = () => {
    const textOpacity = useRef(new Animated.Value(0)).current

    useEffect(() => {
        Animated.timing(textOpacity, {
            toValue: 1,
            duration: 1000,
            useNativeDriver: true
        }).start();
    }, [textOpacity])

    const textStyle = {
        opacity: textOpacity
    }

    return (
        <Animated.View>
            <Text>REACT NATIVE</Text>
        </Animated.View>
    )
}
```

LayoutAnimation

- LayoutAnimation sẽ tạo hiệu ứng trên toàn UI của ứng dụng.
- Nó sẽ tự tạo hiệu ứng khi sự thay đổi layout xảy ra. Chú ý để LayoutAnimation làm việc trên các thiết bị Android, ta cần thực hiện như sau:

```
if (Platform.OS === 'android') {
  if (UIManager.setLayoutAnimationEnabledExperimental) {
    UIManager.setLayoutAnimationEnabledExperimental(true);
  }
}
```

LayoutAnimation

```
const LayoutAnimationTest = () => {
    const [show, setShow] = useState(false);

    const change = () => {
                    LayoutAnimation.configureNext(
                                LayoutAnimation.Presets.linear);
        setShow(prev => !prev);
    }

    return (
    <View style={styles.container}>
        <Button title="Change!" onPress={change} />
        {show && <Text style={styles.text}>Great!</Text>}
    </View>
    )
}
```

Upload Image

- Thư viện expo-image-picker cung cấp cách thức truy cập UI hệ thống để lựa chọn hình ảnh hoặc video từ thư viện của điện thoại hoặc chụp ảnh từ camera.

```
npm install expo-image-picker
```

- Import thư viện vào sử dụng

```
import * as ImagePicker from 'expo-image-picker';
```

Upload Image

```
const pickImage = async () => {
  let { status } =
   await ImgPicker.requestMediaLibraryPermissionsAsync();

  if (status !== 'granted') {
            alert("Permissions denied!");
  } else {
       const result =
            await ImgPicker.launchImageLibraryAsync();
       if (!result.canceled)
           setImage(result.assets[0])
  }
}
```

```
export const ImageUploader = () => {
    const [image, setImage] = useState();
    ...
    return (
    <View style={styles.container}>
        <View style={styles.row}>
            <TouchableOpacity onPress={pickImage}>
                <Text>Choose Image... </Text>
            </TouchableOpacity>
            <TouchableOpacity style={styles.button}
                              onPress={upload}>
                <Text style={{ color: "white" }}>Upload</Text>
            </TouchableOpacity>
        </View>
        {image ? <Image source={{ uri: image.uri }}
                  style={{ width: 100, height: 100 }} /> : ""}
    </View>
    )
}
```

```
const upload = async () => {
    let form = new FormData();
    form.append('image', {
        uri: image.uri,
        name: image.fileName,
        type: image.type
    });

    let res = await fetch('endpoint', {
        method: 'post',
        body: form,
        headers: {
            'Content-Type': 'multipart/form-data'
        }
    });
    let data = await res.json();
    console.info(data);
}
```

Q&A

ThS. Dương Hữu Thành,
Khoa CNTT, Đại học Mở TP.HCM,
thanh.dh@ou.edu.vn.