06016323 Mobile Device Programming

CHAPTER 12 : ANIMATION (ANIMATED API)

Animations

- เราสามารถอัพเดทขนาด ตำแหน่ง สี และการกำหนดสไตล์ต่างๆ เพื่อทำให้คอมโพเนนต์สามารถเคลื่อนไหวได้ (แอนิเมชัน)
- แอนิเมชันเป็นส่วนสำคัญในการทำให้แอปพลิเคชันที่พัฒนามีความน่าสนใจและน่าใช้งาน
- React Native ได้เตรียม API สำหรับพัฒนาแอนิเมชัน 2 ระบบ
  - Animated API
  - LayoutAnimation API

Performance Challenges

- Calculating new layouts during animation
  - เมื่อมีการเปลี่ยนแอตทริบิวต์สไตล์ ซึ่งอาจมีผลต่อคอมโพเนนต์ ทำให้ต้องมีการประมวลผล layout ของ UI ใหม่
- Re-rendering components
  - เมื่อ state หรือ props ของคอมโพเนนต์เปลี่ยน และต้องทำการ re-render คอมโพเนนต์นั้นใหม่ ซึ่งในกรณีที่เป็นแอนิเมชันอาจใช้เวลาพอควรในการ render
- Communicating between native code and JavaScript
  - เนื่องจากต้องมีการสื่อสารกันระหว่าง native thread และ JavaScript engine ซึ่งอาจทำให้การแสดงแอนิเมชันล่าช้าได้

Animated API

- เป็น API ที่ออกแบบมาเพื่อแสดงภาพเคลื่อนไหวที่น่าสนใจ และรูปแบบการโต้ตอบ (Interaction) ที่หลากหลาย
- API นี้ ช่วยให้เราปรับแอตทริบิวต์ของสไตล์ในคอมโพเนนต์ได้ ทำให้เกิดแอนิเมชันขึ้น
- ใช้เมธอด start/stop เพื่อควบคุมการทำงานของแอนิเมชันตามเวลา
- Animated สามารถจัดการแอนิเมชันบนคอมโพเนนต์ View, Text, Image, ScrollView, FlatList และ SectionList ได้
- เราสามารถกำหนดคอมโพเนนต์ที่เราสร้างเองให้สามารถทำแอนิเมชันได้ ผ่าน Animated.createAnimatedComponent()

แนวคิดการทำแอนิเมชัน – กำหนด Animated.Value()

- กำหนดค่า Animated.Value เก็บค่าเริ่มต้นเพื่อใช้ทำแอนิเมชัน
  - เป็นคลาสที่เก็บ primitive value (เช่น ตัวเลข สตริง เป็นต้น) เพื่อใช้กำหนดสไตล์ของคอมโพเนนต์
  - ค่า Animated.Value ที่ใช้กำหนดสไตล์ของคอมโพเนนต์มีการเปลี่ยนแปลงอย่างต่อเนื่อง ทำให้เกิดแอนิเมชันขึ้น เช่น การทำให้ขยายและย่อขนาดของคอมโพเนนต์ การหมุนคอมโพเนนต์ การขยับคอมโพเนนต์ เป็นต้น
  - การกำหนด Animated.Value
    - กรณีของ Function Component: ใช้ useRef
    - กรณีของ Class Component: ใช้การกำหนด state ใน Class

แนวคิดการทำแอนิเมชัน - กำหนดรูปแบบแอนิเมชัน

- กำหนดรูปแบบการทำแอนิเมชัน โดย Animation.Value จะเปลี่ยนแปลงจาก initial value เป็น final value
  - Animated.timing() : ทำการเปลี่ยนค่า Animation.Value ตามเวลา
  - Animated.spring() : ทำการเคลื่อนไหวแบบสปริง
  - Animated.decay() : กำหนดความเร็วเริ่มต้นแล้วค่อยๆ ลดความเร็วจนหยุดการเคลื่อนที่
- กำหนด Option ที่ใช้ทำแอนิเมชันในรูปแบบต่างๆ เช่น กำหนดเวลาที่ใช้ทำแอนิเมชัน เป็นต้น (มีรายละเอียดหลังจากนี้)

แนวคิดการทำแอนิเมชัน – กำหนดรูปแบบแอนิเมชัน (ต่อ)

- การทำแอนิเมชันเริ่มจากการเรียก start()
  - start() จะรับ callback function ซึ่งจะถูกเรียกเมื่อทำแอนิเมชันเสร็จ โดยจะได้รับ {finished: true}
- กรณีที่แอนิเมชันจบ เนื่องจากมีการเรียก stop() ก่อนที่แอนิเมชันจะทำงานจบจริงๆ กรณีนี้ จะได้รับ {finished: false}
- Animated.timing({}).start(({finish}) => { //...do something... })

แนวคิดการทำแอนิเมชัน - กำหนดคอมโพเนนต์แสดงผล

- การทำแอนิเมชันต้องทำกับคอมโพเนนต์เฉพาะ ที่เป็น animatable component เท่านั้น
  - Animated.Image
  - Animated.ScrollView
  - Animated.Text
  - Animated.View
  - Animated.FlatList
  - Animated.SectionList

แนวคิดการทำแอนิเมชัน - กำหนดสไตล์ของคอมโพเนนต์

- กำหนดสไตล์ของคอมโพเนนต์ให้มีค่าสอดคล้องตาม Animated.Value() ที่กำหนดไว้ข้างต้น โดยแอตทริบิวต์พื้นฐานที่ใช้ทำแอนิเมชัน มีดังนี้
  - opacity : กำหนดความทึบ/ความสว่างของคอมโพเนนต์
  - transform : กำหนดการเปลี่ยนรูปของคอมโพเนนต์
    - scale : ปรับขนาด (ย่อ/ขยาย) คอมโพเนนต์
    - rotate : หมุนคอมโพเนนต์
    - translate : ขยับ/เลื่อนคอมโพเนนต์
- Read more: https://reactnative.dev/docs/transforms

ตัวอย่างการทำแอนิเมชันในรูปแบบต่างๆ

การทำให้คอมโพเนนต์ค่อยๆ ทึบ/สว่าง (opacity)

Animated.timing()

- รูปแบบ : **timing(animatedValue, config)**
- config เป็นอ๊อบเจ็คต์ที่มี option ต่างๆ ดังนี้
  - duration: ความยาวแอนิเมชัน (millisecs) : Default = 500
  - easing: ฟังก์ชันเพื่อกำหนดรูปแบบแอนิเมชัน : Default = Easing.inOut(Easing.ease)
  - delay: ดีเลย์ก่อนเริ่มทำแอนิเมชัน (millisecs) : Default = 0
  - isInteraction: สร้าง interaction handle หรือไม่ : Default = true
  - useNativeDriver: ใช้ native driver หรือไม่ : Default = false

ตัวอย่างโปรแกรม

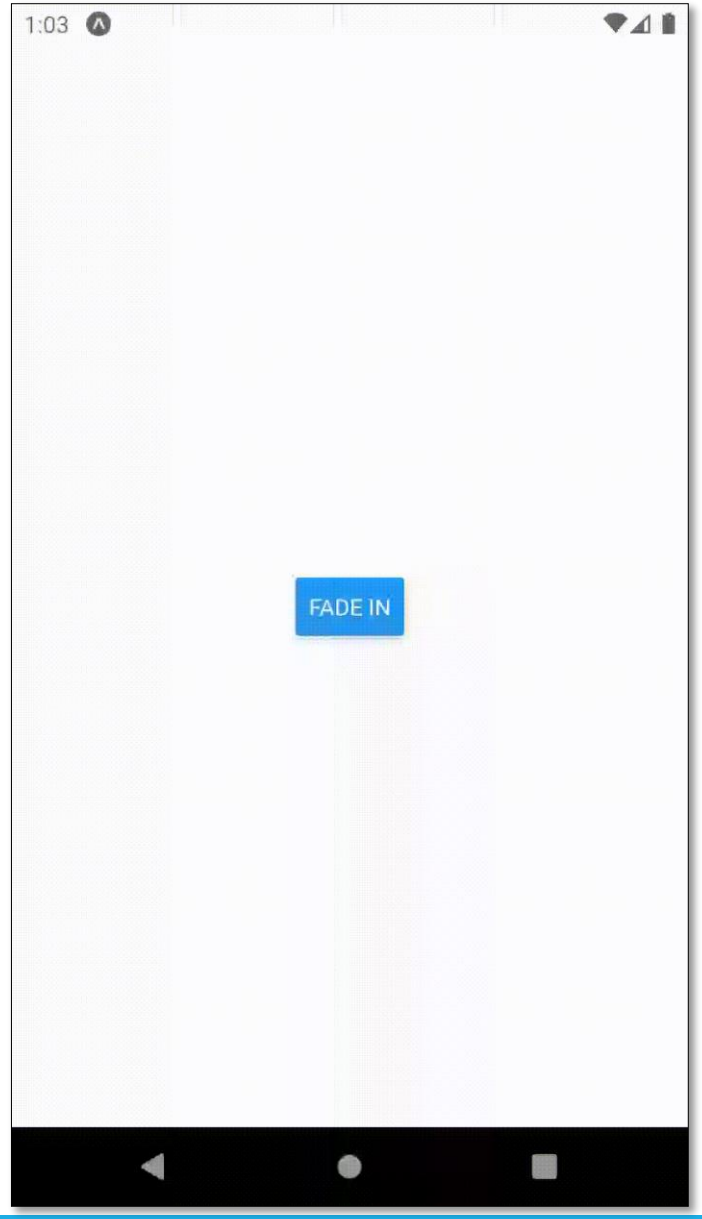
```
import React, { useRef } from "react";
import { Animated, Text, View, StyleSheet, Button } from
"react-native";

const Example01 = (props) => {
  const fadeAnim = useRef(new Animated.Value(0)).current;

  const fadeIn = () => {
    Animated.timing(fadeAnim, {
      toValue: 1,
      duration: 5000,
      useNativeDriver: true,
    }).start(); };
```

```
 return (
    <View style={styles.container}>
      <Animated.View style={ [ styles.fadingContainer,
                                {opacity: fadeAnim} ] }>
        <Text style={styles.fadingText}>Fading View!</Text>
      </Animated.View>
      <Button title="Fade In" onPress={fadeIn} />
    </View>
  );
};
```

ตัวอย่างการทำงานของโปรแกรม



การทำให้คอมโพเนนต์หมุน (transform: rotate)

Interpolate()

- เป็นฟังก์ชันที่รับค่าอินพุต (เป็นช่วง) แล้วทำการ map ออกเป็นเอาท์พุต (เป็นช่วง)
- ค่าช่วงเอาท์พุตที่ได้จากฟังก์ชันนี้ สามารถนำไปใช้กำหนดสไตล์ของคอมโพเนนต์ได้
- ฟังก์ชันที่ใช้ map ค่าอินพุตและเอาท์พุต คือ linear interpolation

```
value.interpolate({
    inputRange: [0, 1],
    outputRange: [0, 100]
})
```

```
value.interpolate({
    inputRange: [0, 0.5, 1]
    outputRange: [0, 100, 0]
})
```

Easing

- เป็นโมดูลที่ใช้กำหนด easing function เพื่อใช้กำหนดลักษณะการเคลื่อนที่ในรูปแบบต่างๆ เช่น
  - bounce : Easing.bounce
  - ease : Easing.ease
  - elastic : Easting.elastic(3)

- Read more: https://reactnative.dev/docs/easing

ตัวอย่างโปรแกรม

```
import React, { useRef } from "react";
import { Animated, View, StyleSheet, Button, Easing } from "react-
native";

const Example02 = (props) => {
  const spinAnim = useRef(new Animated.Value(0)).current;

  const spin = spinAnim.interpolate({
    inputRange: [0, 1],
    outputRange: ["0deg", "360deg"],
  });
  const spinning = () => {
     Animated.timing(spinAnim, {
        toValue: 1,
        duration: 5000,
        easing: Easing.bounce,
 }).start( ()=>{spinAnim.setValue(0)} ); };
```

```
 return (
    <View style={styles.container}>
        <Animated.Image
           style={{ width: 180, height: 150, transform: [{ rotate: spin }] }}
           source={require("../assets/luffy.png")}
         />
        <Button title="Spin" onPress={spinning} />
    </View>
  );
};
```

ตัวอย่างการทำงานของโปรแกรม

การทำให้คอมโพเนนต์เคลื่อนไหวแบบสปริง

Animated.spring()

- รูปแบบ : **spring(animatedValue, config)**
- config เป็นอ๊อบเจ็คต์ที่มี option ต่างๆ ดังนี้
  - friction: ควบคุมการกระเด้ง : Default = 7
  - tension: ควบคุมความเร็ว : Default = 40
  - speed: ควบคุมความเร็ว : Default = 12
  - bounciness: ควบคุมการกระเด้ง : Default = 8

ตัวอย่างโปรแกรม

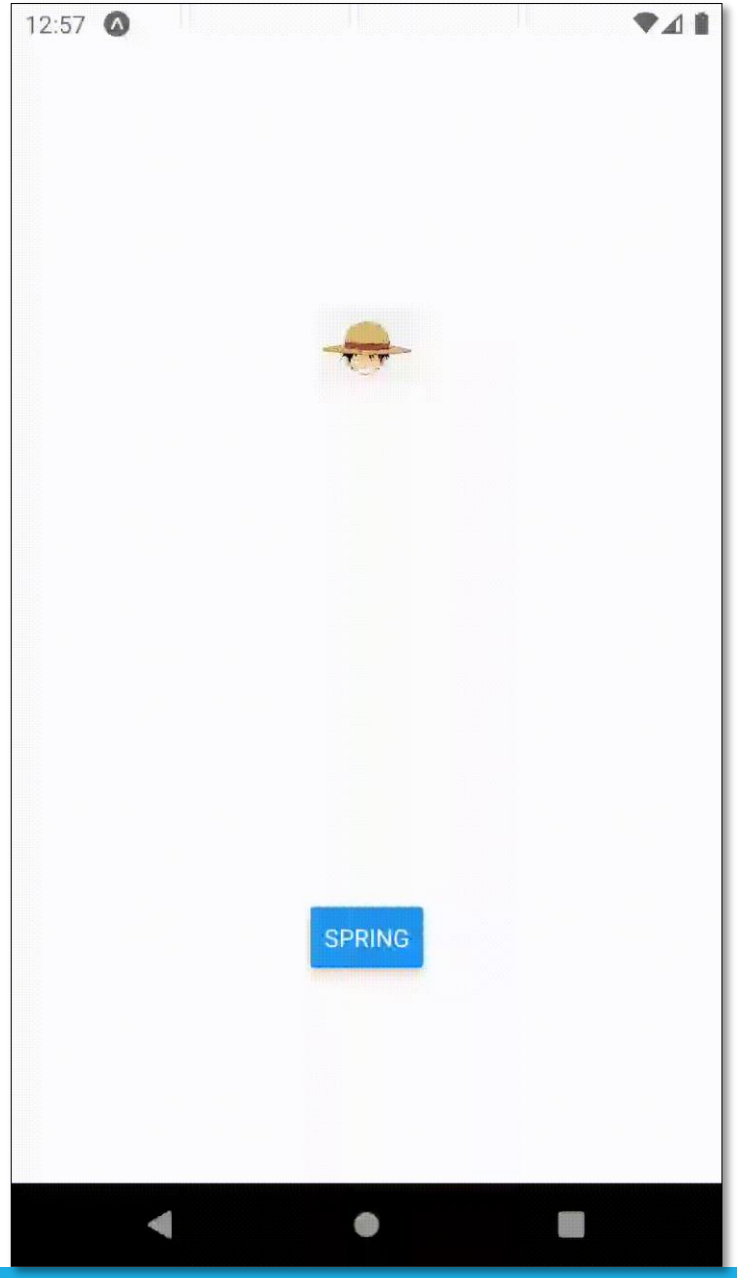
```
import React, { useRef } from "react";
import { Animated, View, StyleSheet, Button} from "react-native";

const Example03 = (props) => {
  const springVal = useRef(new Animated.Value(0.3)).current;

  const spring = () => {
    Animated.spring(springVal, {
      toValue: 1,
      friction: 1,
}).start( ()=> { springVal.setValue(0.3); } ); };
```

```
 return (
    <View style={styles.container}>
      <Animated.Image
        style={{ width: 180, height: 150, transform: [{scale: springVal}]}}
        source={require("../assets/luffy.png")}
      />
      <Button title="Spring" onPress={spring} />
    </View>
  );
};
```

ตัวอย่างการทำงานของโปรแกรม



การรวมแอนิเมชัน

- กรณีที่มีการกำหนดแอนิเมชันหลายส่วน เราสามารถใช้ composition functions เพื่อจัดการการทำแอนิเมชันที่ซับซ้อนได้ โดยการใช้
  - Animated.delay(time) : เริ่มทำแอนิเมชันหลังจากดีเลย์ที่กำหนด
  - Animated.parallel(animations, config?) : ทำแอนิเมชันหลายอันพร้อมๆ กันได้
  - Animated.sequence(animations) : กำหนดลำดับการทำของแอนิเมชันได้ โดยเมื่อแอนิเมชันอันนึงทำเสร็จ แอนิเมชันอีกอันจึงจะทำงานได้
  - Animated.stagger(time, animations) : เริ่มทำแอนิเมชันแต่ละอันตามลำดับ โดยมีการกำหนดดีเลย์ระหว่างแอนิเมชันด้วย

ตัวอย่างโปรแกรม

```
import ... ;

const Example04 = (props) => {
  const animV1 = useRef(...).current;
  const animV2 = useRef(...).current;

  const spin = animV2.interpolate({ ... });
  const animate = () => {
    Animated.parallel( [
      Animated.spring(animV1, { ... }),
      Animated.timing(animV2, { ... }),
    ] ).start( () => { ... } ); };
```

```
return (
   <View style={styles.container}>
     <Animated.Image
       style={{ width: 180, height: 150, transform: [{ scale: animV1 }] }}
       source={require("../assets/luffy.png")}
     />
     <Animated.Image
       style={{ width: 180, height: 150, transform: [{ rotate: spin }] }}
       source={require("../assets/luffy.png")}
     />
     <Button title="Start" onPress={animate} />
   </View>
 );
};
```

ตัวอย่างการทำงานของโปรแกรม