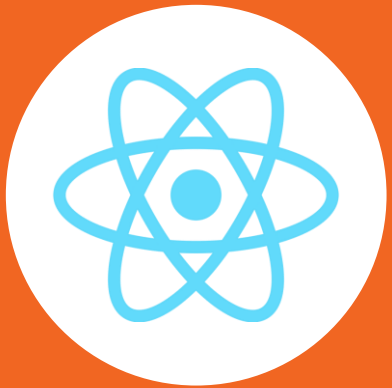
FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 3: ANIMATION TRONG REACT NATIVE

PHẦN 1: GIỚI THIỆU, CÀI ĐẶT THƯ VIỆN REANIMATED

https://caodang.fpt.edu.vn/

Mục tiêu

- Giới thiệu và cài đặt thư viện Reanimated
- Lưu trữ giá trị animation với **useSharedValue()**
- Animation với **withSpring()**

Giới thiệu animation

- **Animation** trong React Native cho phéptạo ra các hiệu ứng chuyển động trong ứng dụng di động của mình. Nó giúp cải thiện trải nghiệm người dùng và tạo ra giao diện tương tác và hấp dẫn hơn.

- React native cũng đã cung cấp sẵn **Animated** để bạn có thể tạo animation cho mình mà không cần cài đặt bất kỳ thư viện nào. Nhưng tạo animation với **Animated** sẽ tạo ra các hiệu ứng không mượt mà, gây chậm hiệu suất ứng dụng và làm giảm trải nghiệm người dùng. Vậy nên ở bài học này chúng ta sẽ sử dụng thư viện **Reanimated,** nó sẽ tạo cho một một animation trơn tru, mạnh mẽ và mang đến cho người dùng trải nghiệm tốt nhất.

Reanimated

- Reanimated là một thư viện React Native cho phép tạo các hiệu ứng chuyển động và tương tác mượt mà chạy trên luồng UI.
- Lý do ra đời thư viện Reanimated
  - Trong các ứng dụng React Native, mã ứng dụng được thực thi bên ngoài luồng chính của ứng dụng. Đây là một trong những yếu tố chính trong kiến trúc của React Native và giúp ngăn ngừa giảm khung hình trong trường hợp luồng JavaScript có một số công việc nặng phải làm. Reanimated nhằm mục đích cung cấp các cách giảm tải animation và logic xử lý sự kiện khỏi luồng JavaScript và vào luồng UI.

Reanimated 3.x có gì mới?

- **Reanimated 3** tập trung vào việc cải thiện tính ổn định và hiệu suất. Phiên bản này sử dụng API Reanimated v2, vẫn giữ các worklet và share value
- Nó đi kèm với việc viết lại toàn bộ cơ chế Shared Value và Layout Animation và Shared element transition
- Bên cạnh nhiều cải tiến và tính năng, **Reanimated 3.x** cũng giới thiệu hỗ trợ cho Kiến trúc mới. Khi ứng dụng của bạn (hoặc thư viện bạn đang sử dụng) sử dụng API Reanimated v1, nó sẽ không hoạt động với Reanimated 3.x.

Bạn có thể truy cập trang document này để biết rõ hơn về chi tiết
https://docs.swmansion.com/react-native-reanimated/docs/

Cài đặt reanimated

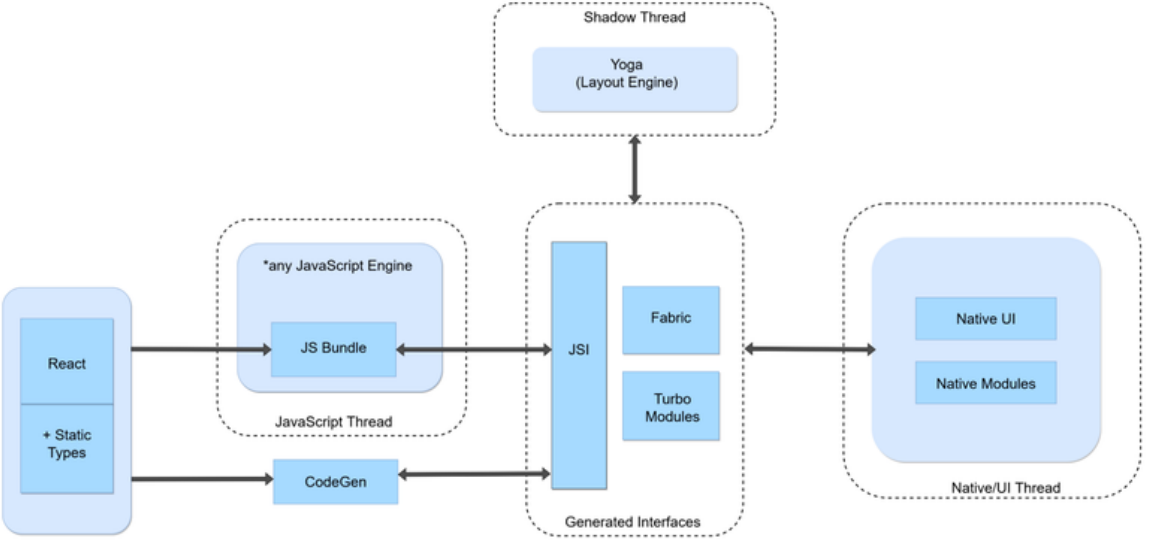
- Cài đặt package

```
npm i react-native-reanimated
```

- Babel plugin
  - Thêm plugin babel của Reanimated vào file **babel.config.js**

```
module.exports = {
  presets: [
    ...
  ],
  plugins: [
    ...
    'react-native-reanimated/plugin',
  ],
};
```

- Trước khi đi vào phần tiếp theo bạn phải hiểu được **Kiến trúc của React Native**



Kiến trúc React Native

3 Thread trong react native

- Khi bạn khởi chạy một React Native app, phần code JS sẽ được package (được gọi là JS Bundle) và được tách riêng ra với phần Native Code (Android/IOS)
- Một ứng dụng React Native sẽ chạy trên 3 thread
  - **JS thread (Javascript thread/ Main thread)**: Được sử dụng bởi JS Engine, dùng để chạy JS bundle
  - **Native/UI thread**: Được sử dụng để khởi chạy các native module, các tiến trình render UI, animations, gesture handle, ...
  - **Shadow thread**: Được sử dụng để tính toán Layout của element trước khi render ra màn hình.

Shared Values

- **Shared Values** là một trong những khái niệm cơ bản đằng sau Reanimated. Nếu bạn đã quen thuộc với API Animated của React Native, bạn có thể so sánh chúng với **Animated.Values**. Chúng phục vụ một mục đích tương tự là mang dữ liệu của animation.
- Một trong những mục tiêu chính của **Shared Values** là cung cấp một khái niệm về bộ nhớ được chia sẻ trong Reanimated.
- Các đối tượng **Shared Values** đóng vai trò là tham chiếu đến các phần dữ liệu được chia sẻ có thể được truy cập và sửa đổi bằng thuộc tính **.value** của chúng.

Shared Values

- Để tạo tham chiếu **Shared Value** sử dụng hook **useSharedValue**:

```
const sharedVal = useSharedValue(3.1415);
```

- Để cập nhật Shared Value từ thread React Native hoặc từ worklet chạy trên UI thread, bạn nên đặt gọi **.value** của Shared Value đó.

```
const changeSharedValue = () => {
  sharedVal.value = Math.random();
};
```

withSpring

- **withSpring** là một hàm trong thư viện Reanimated được sử dụng để chuyển đổi giá trị chuyển động theo dạng một lò xo.
- Cú pháp sử dụng của **withTiming** trong Reanimated như sau:

```
width.value = withSpring(toValue, userConfig, callback);
```

- Tham số

  - **toValue**

    Giá trị dừng lại của animation. Hỗ trợ các giá trị như **number**, **suffixed numbers** ("5.5%", "90deg", "3bananas"), **color**, **objects**, **array**, ...

  - **userConfig** (optional)

    Thiết lập chuyển động của animation.

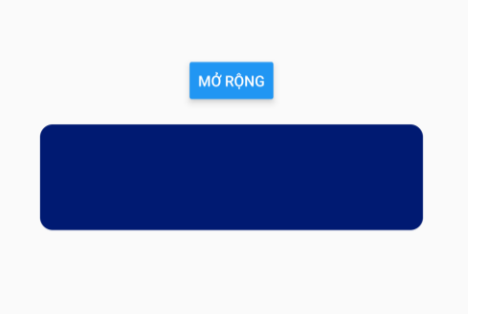
userConfig chấp nhận các tham số truyền vào sau:

```
withSpring(sv.value, {
  mass: 1,
  damping: 10,
  stiffness: 100,
  overshootClamping: false,
  restDisplacementThreshold: 0.01,
  restSpeedThreshold: 2,
  reduceMotion: ReduceMotion.System,
})
```

- **callback** (optional)

  Một hàm được gọi khi hoàn thành animation. Nếu animation bị hủy, callback sẽ nhận được **false** làm đối số; nếu không, nó sẽ nhận được **true**.

- Ví dụ, tạo animation đơn giản đầu tiên



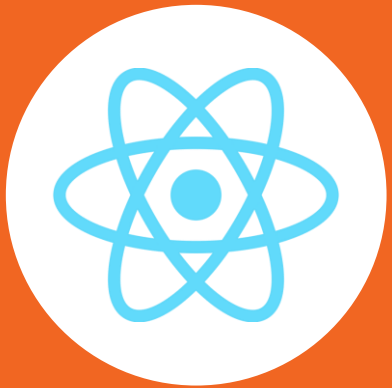
- Kết hợp sử dụng **useSharedValue** và **withSpring** để tạo **animation** cho thẻ View

```
export const UseSharedValueScreen = () => {
  const width = useSharedValue(100);
  const handlePress = () => {
    width.value = withSpring(width.value + 50);
  };

  return (
    <View style={styles.containter}>
      <Button onPress={handlePress} title="Mở rộng" />
      <Animated.View style={[{width}, styles.box]} />
    </View>
)};
```

FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 3: ANIMATION TRONG REACT NATIVE

PHẦN 2: useAnimatedStyle, EVENTS TRONG REANIMATED

Mục tiêu

- Sử dụng useAnimatedStyle để tạo style animation cho component.
- Tạo các animation với reanimated
- Tương tác với các event của con trỏ lên animation

useAnimatedStyle

- **useAnimatedStyle** cho phép bạn tạo một **styles** object, tương tự như **StyleSheet**, có thể tạo animation bằng cách sử dụng **shared values**.
- Các styles sử dụng **useAnimatedStyle** phải được chuyển sang thuộc tính style của thành phần Animated. Styles sẽ được tự động cập nhật khi một **shared value** thay đổi giá trị.

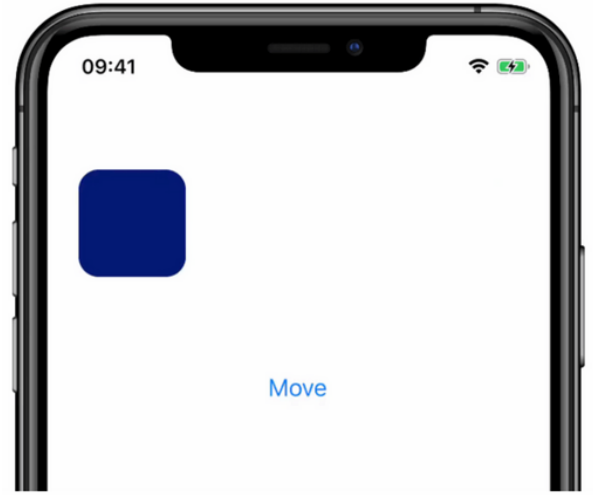
useAnimatedStyle

- Ví dụ

```
function App() {
  const animatedStyles = useAnimatedStyle(() => {
    return {
      opacity: sv.value ? 1 : 0,
    };
  });

  return <Animated.View style={[styles.box, animatedStyles]} />;
}
```

- Xây dựng animation di chuyển trong Reanimated



Trước khi nhấn **Move**

Sau khi nhấn **Move**

- Ví dụ

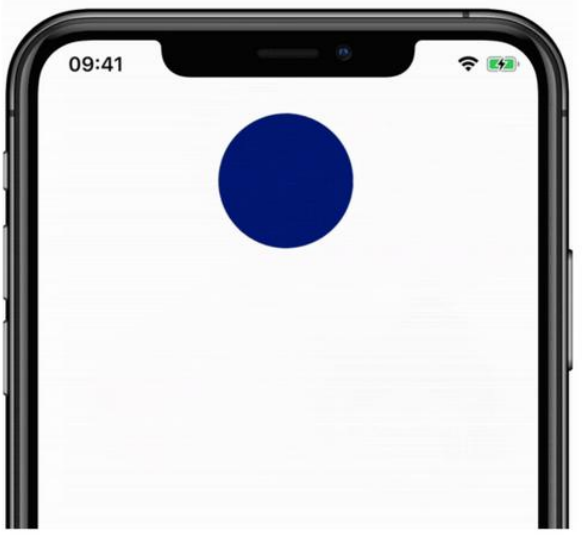
```
const offset = useSharedValue(0);
const animatedStyles = useAnimatedStyle(() => {
  return { transform: [{translateY: offset.value}]};
});

return (
  <View style={styles.containter}>
    <Button
      onPress={() => (offset.value = withSpring(Math.random() * 455))}
      title="Move"
    />
    <Animated.View style={[styles.box, animatedStyles]} />
  </View>
);
```

Events

- Trong thế giới thực, không có gì thay đổi ngay lập tức - luôn có một cái gì đó giữa các thành phần. Khi chúng ta chạm vào một cuốn sách, chúng ta không mong đợi nó mở ngay lập tức trên một trang nhất định. Để làm cho ứng dụng thiết bị di động cảm thấy tự nhiên hơn đối với người dùng, sử dụng animation để làm mượt mà các tương tác của người dùng với giao diện ứng dụng.

- Các sự kiện sử lý với con trỏ như: kéo, thả, nhấn, ...



Element hình tròn sẽ di chuyển theo vị trí con trỏ

Handling gesture events

- Để thao tác các sự kiện liên quan đến con trỏ, các bạn cần cài đặt package **react-native-gesture-handler**
- Bọc dụng của bằng **<GestureHandlerRootView>** thành phần từ thư viện react-native-gesture-handler để sử dụng được các sự kiện con trỏ.

```
import { GestureHandlerRootView } from 'react-native-gesture-handler';

export default function App() {
  return (
    <GestureHandlerRootView style={{ flex: 1 }}>
      {/* content */}
    </GestureHandlerRootView>
  );
}
```

Handling gesture events

- Ví dụ, thao tác với sự kiện con trỏ khi nhấn giữ

Trước khi nhấn

Sau khi nhấn

Handling gesture events

- Tạo nút hình tròn, được bọc bởi **GestureDetector** để bắt sự kiện khi có tác động lên phần tử này.

```
export const GestureHandleScreen = () => {
  const pressed = useSharedValue(false);
  return (
    <GestureDetector gesture={tap}>
      <Animated.View style={[styles.ball, uas]} />
    </GestureDetector>
  );
};
```

Handling gesture events

- Sử dụng **Gesture.Tap()** để bắt được sự kiện nhấn con trỏ

```
const tap = Gesture.Tap()
  .onBegin(() => {
    pressed.value = true;
  })
  .onFinalize(() => {
    pressed.value = false;
  });
```

- **onBegin:** khi bắt đầu nhấn vào
- **onFinalize:** khi thả tha

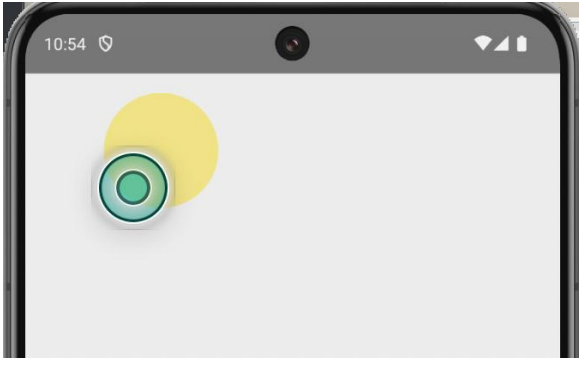
Handling gesture events

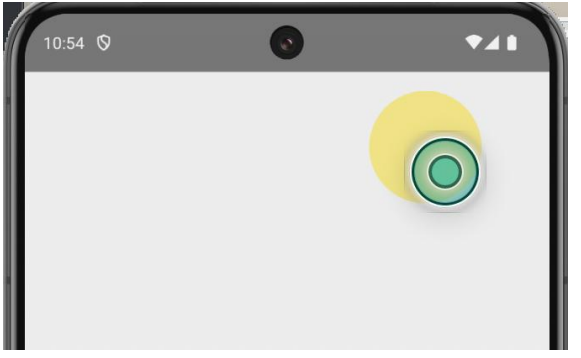
- Style cho hình tròn bằng **useAnimatedStyle**, màu background và độ scale của hình tròn sẽ dựa vào **pressed.value.**

```
const uas = useAnimatedStyle(() => {
  return {
    backgroundColor: pressed.value ? '#FEEF86' : '#001972',
    transform: [{scale: pressed.value ? 1.2 : 1}],
  };
});
```

Handling gesture events

- Sử dụng **Gesture.Pan()** bắt sự kiện di chuyển con trỏ





Handling gesture events

- Thêm **onChange** bắt vị trí của con trỏ hiện tại. Reset lại vị trí **offset** trong hàm **onFinalize,** với **withSpring()**

```
const offset = useSharedValue(0);

const tap = Gesture.Pan()
  .onBegin(() => {
    pressed.value = true;
  })
  .onChange(event => {
    offset.value = event.translationX;
  })
  .onFinalize(() => {
    offset.value = withSpring(0);
    pressed.value = false;
  });
```

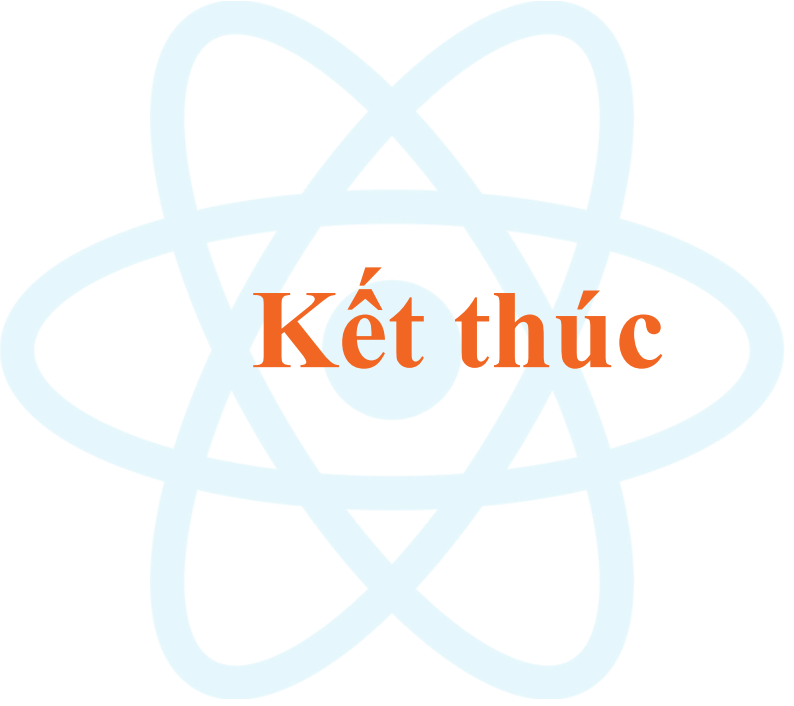
Handling gesture events

- Đưa giá trị **offset.value** vào **translateX** để di chuyển phần tử.

```
const uas = useAnimatedStyle(() => {
  return {
    backgroundColor: pressed.value ? '#FEEF86' : '#001972',
    transform: [{translateX: offset.value}, {scale: pressed.value ? 1.2 : 1}],
  };
});
```

Tổng kết

- Hiểu và sử dụng hiệu quả thư viện Reanimated
- Tối ưu hóa hiệu suất cho animation
- Xây dựng các hiệu ứng animation di chuyển, thay đổi kích thước.
- Tạo animation với các thao tác con trỏ



Kết thúc