Animation Material in Expo (React Native)

This document explains in detail each code block used in animations in Expo React Native. The discussion focuses on four main topics: button animation, list item animation, scroll-based animation, and swipe card gesture animation.

Part 1 – Button Animation (AnimatedButton)

1. Import Library

```
import { useRouter } from 'expo-router';
import React from 'react';
import { StyleSheet, Text, TouchableWithoutFeedback } from 'react-native';
import Animated, { useAnimatedStyle, useSharedValue, withSpring } from 'react-native-reanimated';
```

- useRouter is used for navigation purposes.
- TouchableWithoutFeedback is used as a wrapper for the button.
- useSharedValue is used to store animation values.
- useAnimatedStyle is used to bind animation values to UI styles.
- withSpring is used to create elastic (spring-based) animations.

2. Shared Value

```
const scale = useSharedValue(1);
```

This stores the initial scale value of the button, which is set to 1 (normal size)

3. Animated Style

```
const animatedStyle = useAnimatedStyle(() => ({
    transform: [{ scale: scale.value }],
}));

```

This function reads the scale value and binds it to the transform → scale property. When the scale value changes, the UI updates in real time.

4. Event Animasi

```
const onPressIn = () => {
    scale.value = withSpring(0.85);
};

const onPressOut = () => {
    scale.value = withSpring(1);
};
```

When the button is pressed, its scale is reduced to 0.85, making it appear smaller. When the button is released, the scale returns to 1.

5. Render Button

```
<TouchableWithoutFeedback
    onPressIn={onPressIn}
    onPressOut={onPressOut}
    onPress={() => console.log('Navigate to Profile')}
>
    <Animated.View style={[styles.button, animatedStyle]}>
        <Text style={styles.text}>Go to Profile</Text>
    </Animated.View>
</TouchableWithoutFeedback>
```

Animated.View is used to bind the animation to the button's visual representation.

Part 2 – Animasi List + Scroll Button (Orders)

1. State & Shared Value

```
const translateY = useSharedValue(100);
const [visible, setVisible] = useState(false);
```

translateY determines the vertical position of the button.

visible determines whether the button is displayed or hidden.

2. Scroll Handler

```
const handleScroll = (event: any) => {
  const y = event.nativeEvent.contentOffset.y;

  if (y > 0) {
    setVisible(true);

    if (hideTimer.current) {
      clearTimeout(hideTimer.current);
    }

    hideTimer.current = setTimeout(() => {
      setVisible(false);
    }, 1000);
  }
};
```

When the user scrolls downward, the button becomes visible.

3. Effect Animasi

```
useEffect(() => {
  translateY.value = withTiming(visible ? 0 : 100, {
    duration: 400,
  });
}, [visible]);
```

If visible is true, the button moves upward and appears.

If `visible` is `false`, the button moves downward and hides.

4. Animated Style Button

```
const buttonStyle = useAnimatedStyle(() => ({
  transform: [{ translateY: translateY.value }],
}));
```

This connects the animated value to the button's UI style.

5. Stagger List Animation

```
entering={FadeInRight.delay(i * 100)}
```

Each list item appears one by one using a staggered animation effect.

Part 3 – Gesture Swipe Card

1. Shared Value

```
const translateX = useSharedValue(0);
const translateY = useSharedValue(0);
```

These store the card's position based on the user's gesture.

2. Gesture Logic

```
const panGesture = Gesture.Pan()
    .onUpdate((event) => {
        translateX.value = event.translationX;
        translateY.value = event.translationY;
    })
```

The card moves following the user's finger movement.

3. Swipe Decision

```
.onEnd(() => {
    if (Math.abs(translateX.value) > SWIPE_LIMIT) {
        const to = translateX.value > 0 ? width : -width;

        translateX.value = withSequence(
            withSpring(to),
            withDelay(500, withSpring(0))
        );

        translateY.value = withDelay(500, withSpring(0));
    } else {
        translateX.value = withSpring(0);
        translateY.value = withSpring(0);
    }
});
```

If the swipe distance exceeds a certain threshold, the card is moved out of the screen.

4. Animasi Keluar & Kembali

translateX.value = withSequence(withSpring(to), withDelay(500, withSpring(0)));

After leaving the screen, the card automatically returns to its original position.

5. Rotasi Card

```
const rotate = interpolate(
    translateX.value,
    [-width, 0, width],
    [-15, 0, 15]
);
```

The farther the card is dragged, the more it rotates, creating a natural swipe effect.

Student Assignment

Create a simple application similar to Tinder with the following requirements:

- Display at least three cards in a stack layout.
- The top card can be swiped left and right.
- Left and right swipes must have different behaviors.
- Use react-native-reanimated and react-native-gesture-handler.
- The animations must run smoothly and responsively.

Submit The source code and An application demo (video recording or live demonstration)

https://github.com/deni-stwn/RN-EXPO-ANIMATION