React Native Animations: Beginner to Advanced

Subtitle: Learn how to create smooth and interactive animations to enhance the user experience in your apps.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Animations make mobile apps engaging and responsive. React Native provides multiple ways to animate components, including the **Animated API**, **LayoutAnimation**, and libraries like **React Native Reanimated**. This guide covers beginner-friendly techniques to add movement and interactivity to your apps.

Step 1: Using the Animated API

The `Animated` API allows you to animate properties like **opacity, position, and scale**.

Example: Fade-in a component:

```
import React, { useRef, useEffect } from 'react';
import { Animated, View, Text } from 'react-native';

export default function App() {
  const fadeAnim = useRef(new Animated.Value(0)).current;

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

  return (
    <Animated.View style={{ opacity: fadeAnim, padding: 20 }}>
      <Text style={{ fontSize: 24 }}>Hello React Native Animations</Text>
    </Animated.View>
  );
}
```

Exercise: Animate a **button's opacity** when pressed

Exercise: Animate a **button's opacity** when pressed.

Step 2: Animating Position

Move components smoothly using `Animated.Value` and `Animated.timing`.

Example: Slide a box horizontally:

```
const slideAnim = useRef(new Animated.Value(0)).current;

useEffect(() => {
  Animated.timing(slideAnim, {
    toValue: 200,
    duration: 1000,
    useNativeDriver: true,
  }).start();
}, [slideAnim]);

<Animated.View style={{ transform: [{ translateX: slideAnim }] }}>
  <View style={{ width: 50, height: 50, backgroundColor: 'red' }} />
</Animated.View>
```

Exercise: Animate a square vertically up and down in a loop.

Step 3: Scaling & Rotation

- Scale components: `transform: [{ scale: scaleAnim }]`
- Rotate components: `transform: [{ rotate: rotateAnim.interpolate({ inputRange: [0,1], outputRange: ['0deg','360deg'] }) }]`

Example: Scale a box on button press:

```
<Animated.View style={{ transform: [{ scale: scaleAnim }] }}>
  <View style={{ width: 50, height: 50, backgroundColor: 'blue' }} />
</Animated.View>
```

Exercise: Create a **pulsing animation** for a notification icon.

Step 4: LayoutAnimation for Simple Transitions

`LayoutAnimation` lets you animate **layout changes automatically** without manually updating Animated values.

Example:

```
import { LayoutAnimation, UIManager, Button, View } from 'react-native';

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

const [expanded, setExpanded] = useState(false);

const toggle = () => {
  LayoutAnimation.configureNext(LayoutAnimation.Presets.easeInEaseOut);
  setExpanded(!expanded);
};
```

Exercise: Create a **collapsible menu** using LayoutAnimation.

Step 5: Best Practices

- Prefer **useNativeDriver: true** for smoother animations.
- Avoid animating **layout-heavy properties** like width/height for performance.
- Combine **Animated API and gesture libraries** for interactive animations.
- Keep animations **subtle and purposeful** — avoid overwhelming users.

Animations make apps feel alive and professional. Mastering React Native animations helps you create **visually appealing and responsive interfaces**.

Visit **haas.dev** for more React Native guides, tutorials, and project examples.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
