Version: 3.x

Applying modifiers

Another way of customizing animations in Reanimated is by using animation modifiers. Reanimated comes with three built-in modifiers: `withRepeat`, `withSequence`, and `withDelay`.

Let's build a simple shake animation which uses all the modifiers and learn them along the way. Let's go!

Starting point

In this example, we're going to make an animated box which will shake on a button press after a slight delay.

Let's start by having an `Animated.View` and a `Button` which moves the view to the right on press. To achieve this we can use `useAnimatedStyle` and `withTiming` function to smoothly animate our box `40px` to the right.

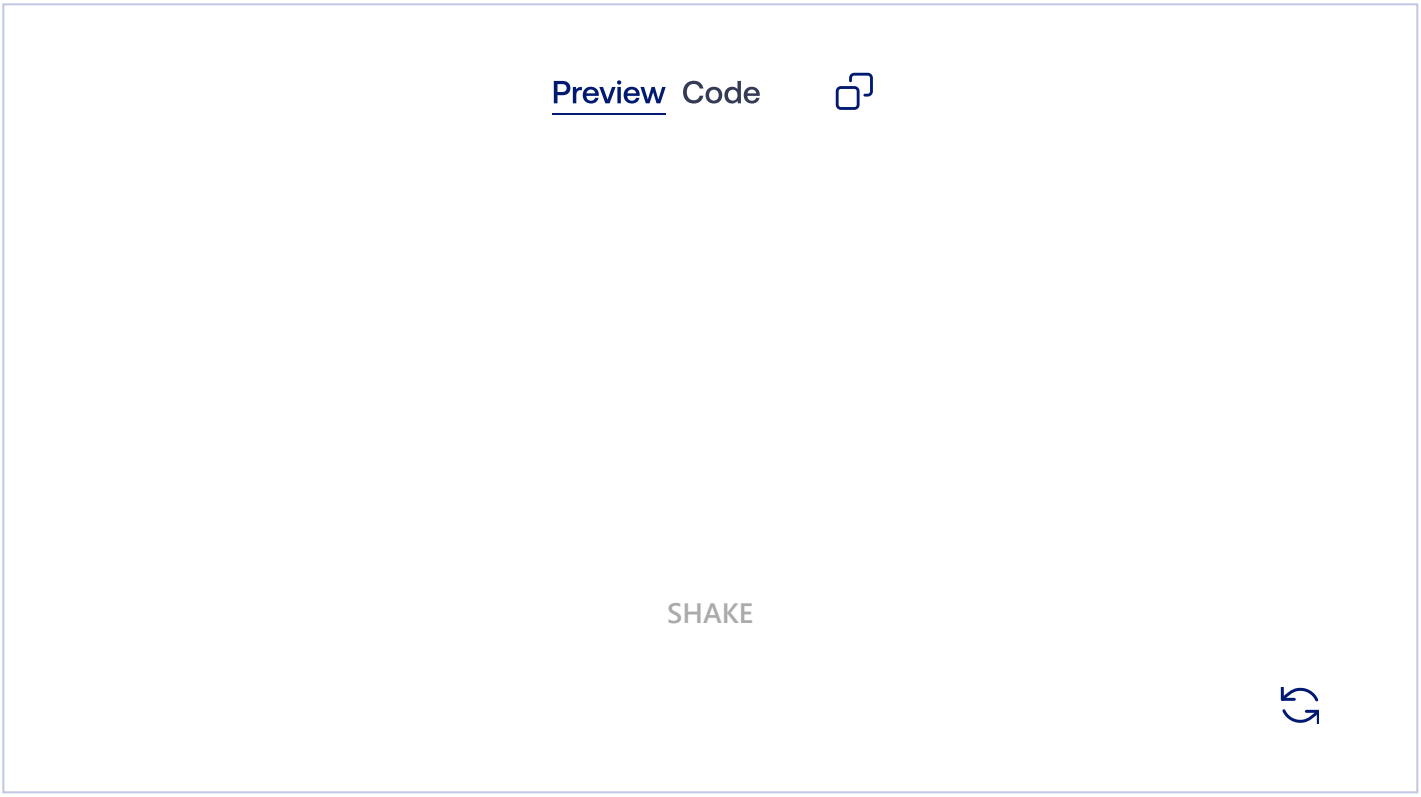
If this sounds new to you, no worries! Start by going through the basics in Your First Animation and come back here when you're ready.

Expand the full code

```
export default function App() {
  const offset = useSharedValue(0);

  const style = useAnimatedStyle(() => ({
    transform: [{ translateX: offset.value }],
  }));

  const OFFSET = 40;

  const handlePress = () => {
    offset.value = withTiming(OFFSET);
  };

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

```
      <Button title="shake" onPress={handlePress} />
    </View>
  );
}
```

Preview Code



Repeating an animation

To implement our desired shake animation we can use `withRepeat` modifier. This modifier lets you repeat a provided animation.

```
import { withRepeat } from 'react-native-reanimated';

function App() {
  sv.value = withRepeat(withTiming(50), 5);
  // ...
}
```

Pass a number to the second parameter of the function to make it repeat a given number of times. You can make it repeat forever by passing a non-positive value (eg. `0` or `-1`). You can make the animation go back and forth by passing `true` to the third (i.e. `reverse`) argument.

To learn more about it you can check out the full `withRepeat` API reference.

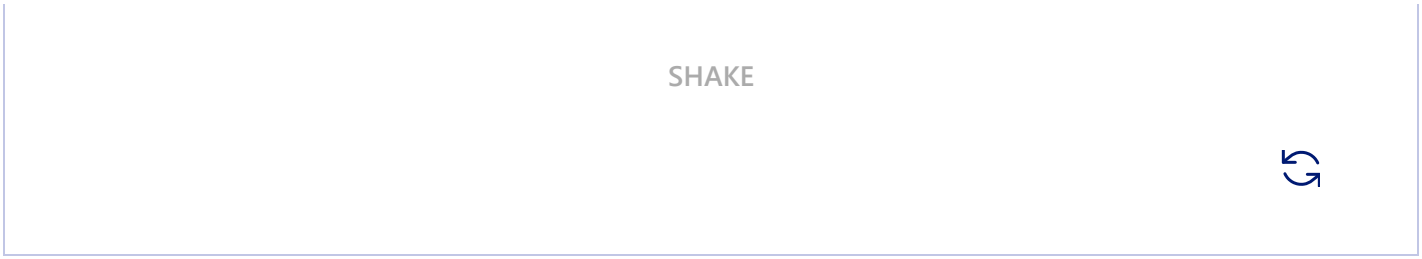
Let's use this function in our example:

Expand the full code

```
export default function App() {
  const offset = useSharedValue(0);

  const style = useAnimatedStyle(() => ({
    transform: [{ translateX: offset.value }],
  }));

  const OFFSET = 40;

  const handlePress = () => {
    offset.value = withRepeat(withTiming(OFFSET), 5, true);
  };

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

When we run this code we can see that the box jiggles left to right between the set offset and a starting position. After the animation finishes the box doesn't come back to its initial place. It's not really how we imagined a shake animation but we'll get there in a second!

Preview Code



Running animations in a sequence

One way to improve our animation is to start it with a left offset and reset it back to starting position after the animation ends. That's a perfect use case for the `withSequence` modifier.

```
import { withSequence } from 'react-native-reanimated';

function App() {
  sv.value = withSequence(withTiming(50), withTiming(0));
  // ...
}
```

This modifier lets you chain animations together. The next animation starts when the previous one ends. You use it by passing animations as arguments in the order you want them to run.

To learn more about it you can check out the full `withSequence` **API reference**.

Coming back to our example - we can utilize `withSequence` to improve our animation. First, we'll sway the box to the left which we'll make it to take half of the duration of one swing. Then, we'll shake the box 5 times and finish up the animation by bringing it to starting position also with half of the animation duration.

Expand the full code

```
const OFFSET = 40;
const TIME = 250;

const handlePress = () => {
  offset.value = withSequence(
    // start from -OFFSET
    withTiming(-OFFSET, { duration: TIME / 2 }),
    // shake between -OFFSET and OFFSET 5 times
    withRepeat(withTiming(OFFSET, { duration: TIME }), 5, true),
```

```
      // go back to 0 at the end
      withTiming(0, { duration: TIME / 2 })
    );
  };
```

Preview Code

Starting the animation with delay

As a cherry on top, we'll add a little bit of suspense by adding a slight delay before the animation begins. For this exact purpose, Reanimated comes with a `withDelay` modifier.

```
import { withDelay } from 'react-native-reanimated';

function App() {
  sv.value = withDelay(500, withTiming(0));
  // ...
}
```

This function as a first parameter takes the duration in milliseconds before the animation starts. The second parameter defines the animation to delay.

To learn more about it you can check out the full `withDelay` API reference.

Expand the full code

```
const OFFSET = 40;
const TIME = 250;
const DELAY = 400;

const handlePress = () => {
  offset.value = withDelay(
    DELAY,
    withSequence(
      // start from -OFFSET
      withTiming(-OFFSET, { duration: TIME / 2 }),
      // shake between -OFFSET and OFFSET 5 times
      withRepeat(withTiming(OFFSET, { duration: TIME }), 5, true),
      // go back to 0 at the end
      withTiming(0, { duration: TIME / 2 })
    )
  );
};
```

Preview Code

SHAKE

Summary

In this section, we learned how to create complex animations by using animation modifiers. To sum up:

- Reanimated comes with three built-in animation modifiers - `withRepeat`, `withSequence`, and `withDelay`.
- `withRepeat` lets you repeat an animation a given number of times or run it indefinitely.
- `withSequence` lets you run animations in a sequence.
- `withDelay` lets you start an animation with a delay.

What's next?

In the next section, we're going to learn about handling `Tap` and `Pan` gestures. Also, we'll get to know the `withDecay` animation function.

Edit this page