Version: 2.6.0 – 2.12.0

# Quick start

RNGH2 provides much simpler way to add gestures to your app. All you need to do is wrap the view that you want your gesture to work on with `GestureDetector`, define the gesture and pass it to detector. That's all!

To demonstrate how you would use the new API, let's make a simple app where you can drag a ball around. You will need to add `react-native-gesture-handler` (for gestures) and `react-native-reanimated` (for animations) modules.

---

**Step 1**

First let's define styles we will need to make the app:

```
const styles = StyleSheet.create({
  ball: {
    width: 100,
    height: 100,
    borderRadius: 100,
    backgroundColor: 'blue',
    alignSelf: 'center',
  },
});
```

---

**Step 2**

Then we can start writing our `Ball` component:

```
function Ball() {
  return (
    <GestureDetector>
      <Animated.View style={[styles.ball]} />
    </GestureDetector>
  );
}
```

---

**Step 3**

```
const isPressed = useSharedValue(false);
const offset = useSharedValue({ x: 0, y: 0 });
```

We also need to define shared values to keep track of the ball position and create animated styles in order to be able to position the ball on the screen:

```
const animatedStyles = useAnimatedStyle(() => {
  return {
    transform: [
      { translateX: offset.value.x },
      { translateY: offset.value.y },
      { scale: withSpring(isPressed.value ? 1.2
: 1) },
    ],
    backgroundColor: isPressed.value ? 'yellow'
: 'blue',
  };
});
```

Step 4

And add it to the ball's styles:

```
...
return (
  <GestureDetector>
    <Animated.View style={[styles.ball,
animatedStyles]} />
  </GestureDetector>
);
...
```

Step 5

The only thing left is to define the pan gesture and assign it to the detector:

```
const start = useSharedValue({ x: 0, y: 0 });
const gesture = Gesture.Pan()
  .onBegin(() => {
    isPressed.value = true;
  })
  .onUpdate((e) => {
    offset.value = {
      x: e.translationX + start.value.x,
      y: e.translationY + start.value.y,
    };
  })
  .onEnd(() => {
    start.value = {
      x: offset.value.x,
      y: offset.value.y,
    };
  })
  .onFinalize(() => {
    isPressed.value = false;
  });
```

```
...
return (
  <GestureDetector gesture={gesture}>
    <Animated.View style={[styles.ball,
animatedStyles]} />
  </GestureDetector>
);
...
```

---

Note the `start` shared value. We need it to store the position of the ball at the moment we grab it to be able to correctly position it later, because we only have access to translation relative to the starting point of the gesture.

Now you can just add `Ball` component to some view in the app and see the results! (Or you can just check the code here and see it in action in the Example app.)