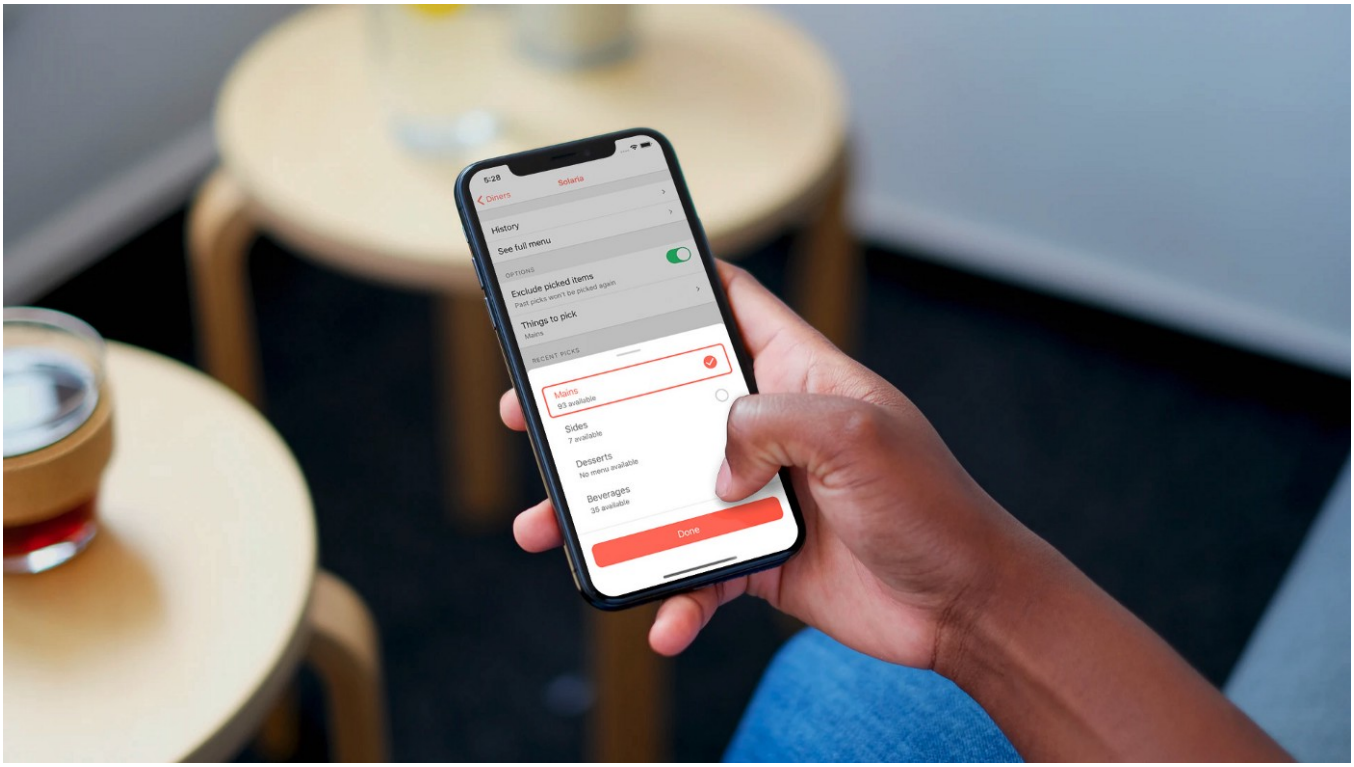
Making the Bottom Sheet Modal using React Native

Endy Hardy

It’s a piece of UI that serves many purpose and implemented in so many ways. It has become one of the best ways to contain secondary information and control to a page, while keeping the cohesion with the page.

The bottom sheet (or as I like to call it — the swipeable bottom modal), is a container that overlays a parent page (like a modal), initializes by popping up from the bottom of the screen (near where your thumb is likely to be), and can be dismissed with a vertical swipe down.

It’s one of those things where I try to avoid reaching for a third-party package given its simple purpose, but it’s also quite tricky to pull off. Here’s how I got it to work with React Native.



What I was aiming for.

Enter /components/BottomSheet.js

First, start by extending React Native's built-in `Modal` component, and creating the overlay:

```
<Modal
  animated
  animationType="fade"
  visible={this.props.visible}
  transparent
  onRequestClose={() => this._handleDismiss()}>
  <View style={styles.overlay}>
    ...
  </View>
</Modal>const styles = StyleSheet.create({
  overlay: {
    backgroundColor: 'rgba(0,0,0,0.2)',
    flex: 1,
    justifyContent: 'flex-end',
  },
});
```

To make the container slide in from the bottom of the screen, create an `Animated.View` component for the container, and set the `top` styling to respond to the animation value.

```
...
<View style={styles.overlay}>
  <Animated.View style={[styles.container, {top}]}>
    {this.props.children}
  </Animated.View>
</View>
...const styles = StyleSheet.create({
  ...
```

```
  container: {
    backgroundColor: 'white',
    paddingTop: 12,
    borderTopRightRadius: 12,
    borderTopLeftRadius: 12,
  },
});
```

Define an animation value in the component state. Then, make two animations for: resetting and sliding the container to the normal position; and dismissing and sliding the container out of view.

```
constructor(props) {
  super(props);
  this.state = {
    panY: new Animated.Value(Dimensions.get('screen').height)
  };
  this._resetPositionAnim = Animated.timing(this.state.panY, {
    toValue: 0,
    duration: 300,
  }
  this._closeAnim = Animated.timing(this.state.panY, {
    toValue: Dimensions.get('screen').height,
    duration: 500,
  }
}
```

Of course, you can use whichever animation type you want. Just make sure the initial value and the value after dismiss will shove the container completely out of view. You may have noticed I have named the animation value as `panY`. That's because we're gonna use it to respond to the swiping gesture.

Interpolate the animation value from the container's `top` position, to cancel limit the output value to zero. This is to prevent the modal from

‘detaching’ from the bottom of the screen when the user swipes too far upwards.

```
render() {
  const top = this.state.panY.interpolate({
    inputRange: [-1, 0, 1],
    outputRange: [0, 0, 1],
  });
  ...
  <Animated.View style={[styles.container, {top}]}>
    {this.props.children}
  </Animated.View>
  ...
}
```

Next, define and attach pan responders to the container. For this, we can create a `view` component that wraps around the animated container. The view will be made to respond to two events: on move (to swipe the modal up and down) and on release (to detect if we should dismiss the modal or return it to its normal position).

```
constructor(props) {
  ...
  this._panResponders = PanResponder.create({
    onStartShouldSetPanResponder: () => true,
    onMoveShouldSetPanResponder: () => false,
    onPanResponderMove: Animated.event([
      null, {dy: this.state.panY}
    ]),
    onPanResponderRelease: (e, gs) => {
      if (gs.dy > 0 && gs.vy > 2) {
        return this._closeAnim.start(() => this.props.onDismis
      }
      return this._resetPositionAnim.start();
    },
```

```
  });
}
```

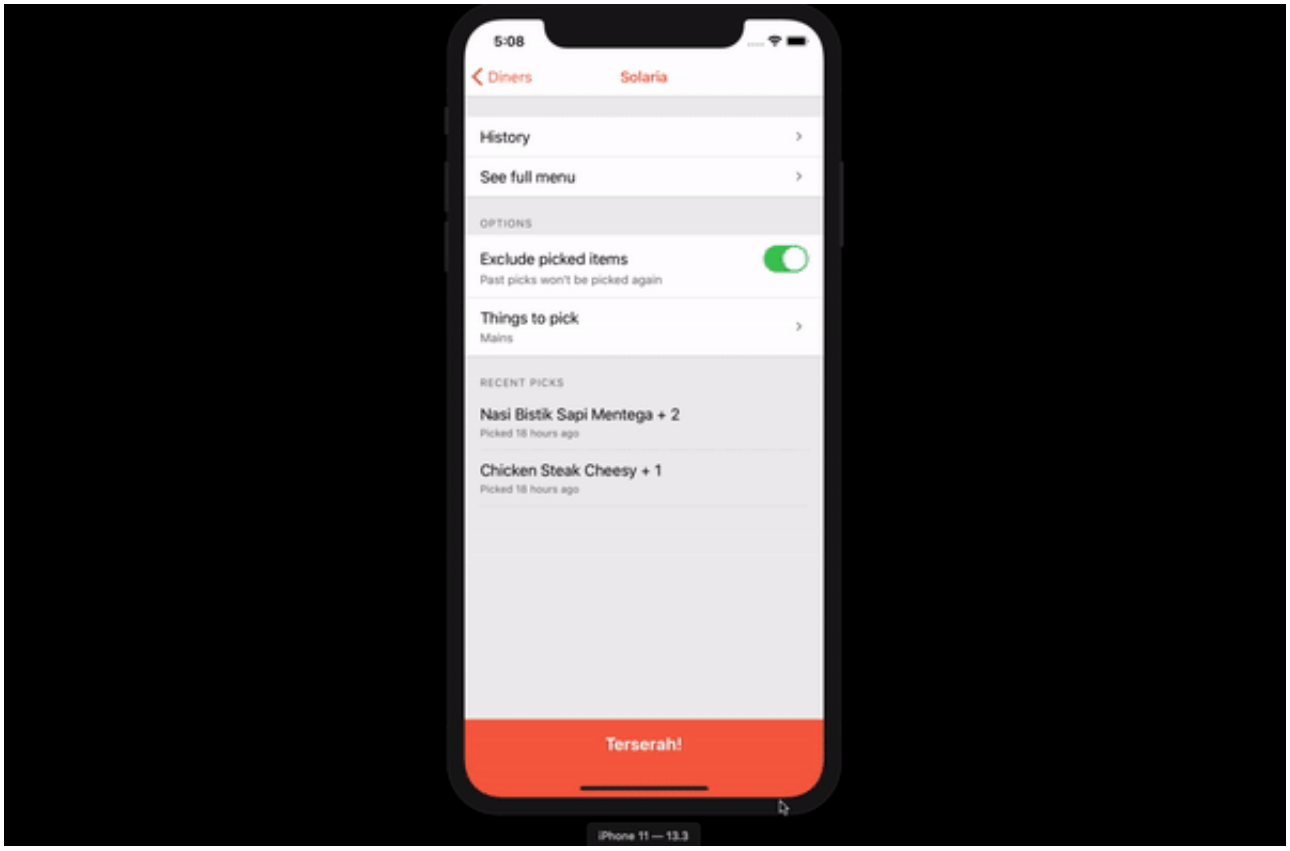
Set `onMoveShouldSetPanResponder` to `false` so it wont take over events for components within the container itself.

On the `onPanResponderMove` event, give an animated event object, which will update the animated value `panY` as the user moves their touch. On the `onPanResponderRelease` event, give a callback that will decide if it should dismiss or reset the modal, based on the direction (`dy`) and speed (`vy`) of the gesture.

Finally, a couple more functions to handle dismissing and initial slide-in:

```
componentDidUpdate(prevProps, prevState) {
  if (
    prevProps.visible !== this.props.visible
    && this.props.visible
  ) {
    this._resetPositionAnim.start();
  }
}_handleDismiss() {
  this._closeAnim.start(() => this.props.onDismiss());
}
```

That’s it!



On a side note, the example doesn't yet account for modal content exceeding the screen height. A scroll view inside the container, coupled with maximum container height might be a way to solve it. Also, to dismiss the modal from its parent component, it should call the `_handleDismiss` function. This can be done by exposing the function using a ref.

Thanks for reading! Check out my other stories about mobile development using React Native!