React Native - Basics - Style

- Dr Nick Hayward

A brief intro to the basics of *Style* in React Native app development.

Contents

Intro

React Native uses JavaScript to define and set styling within an app.

These styles can be set as *props* on all of React Native's core components using the `prop` called `style`.

It is also possible to set a `prop` for style on custom components.

General usage

Similar to CSS usage with standard client-side apps, styles are defined and set for colour, size, background colour, and so on. Property names for these styles are, however, specified using a camelCase pattern. e.g.

```
fontWeight
fontSize
backgroundColor
```

Such styles may be set using a plain JavaScript variable as a container for multiple styles. Using `StyleSheet.create()`, we can pass an object defining multiple custom style properties. These properties include name/value pairs, and the value is set as an object with the defined styles. e.g.

```
const styles = StyleSheet.create({
  headermain: {
    fontWeight: 'bold',
    fontSize: 25,
    color: 'green',
  },
});
```

`style` usage

To add a style to a component, we can set the value of the `style` prop to a standard JavaScript object.

e.g.

```
<Text style={styles.headermain}> Main Header</Text>
```

In this example, we're simply using the property from the `styles` object, which in turn will add the required `style` values for the defined prop.

Platform specific styles

A common requirement for development is the abstraction of code, and likewise styling for an app's UI.

For React Native, we also need to consider both iOS and Android. So, we might use the `Platform` module to add platform detection for iOS or Android.

We can add this directly to a stylesheet, e.g.

```
import { Platform, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  welcome: {
    ...Platform.select({
      ios: {
        fontFamily: 'Arial',
        color: 'cadetblue',
      },
      android: {
        fontFamily: 'Roboto',
        color: 'green',
      },
    }),
    textAlign: 'center',
    margin: 10,
    fontSize: 20,
  },
});
```

In this example `styles`, we can set platform specific styling for font colour and font family. The remaining styles will then be applied for both platforms.

Style inheritance

The documentation for React Native suggests a preferred pattern for setting parent styles, which may then be inherited for children.

This pattern uses nested components with a custom parent defined with abstracted styles. A child component may then inherit such styles or override with specific component-level styles.

e.g.

```
class MyAppText extends Component {
  render() {
    return (
      <Text>
        {this.props.children}
      </Text>
    );
  }
}
```

A parent component is created for an app's rendering of basic text. This will simply return any child text as a default `Text` component. However, we may also create custom styles to add to this new component.

e.g.

```
textdefault: {
```

```
  fontSize: 15,
  color: '#000'
}
```

Usage may then be as follows,

```
<MyAppText style={styles.textdefault}>
  some app text...
  <Text style={styles.welcome}>Welcome to Styles!</Text>
</MyAppText>
```

So, the *child* text in the `MyAppText` component will initially be styled with the `textdefault` styles. We may then override or supplement these styles with specific styles on a given child component.

e.g.

```
welcome: {
  ...Platform.select({
    ios: {
      fontFamily: 'Arial',
      color: 'blue',
    },
    android: {
      fontFamily: 'Roboto',
      color: 'green',
    },
  }),
  fontSize: 25,
  textAlign: 'auto',
  backgroundColor: '#ddd',
}
```

Component sizes

We can also specify sizes for a component, including height and width as rendered in an app's screen.

e.g.

```
<View style={{width: 200, height: 100}} />
```

This allows us to easily create separation and variance in the rendering of our components.

working with Flex dimensions

We can use flex to define how a component may occupy space within a screen.

A standard usage is to simply set a `View` component, for example, to `flex: 1`. This tells the component to fill all of the available space.

However, this usage will also be dependent upon the structure and size of the parent component. It will only be able to expand into space if it's available.

e.g.

```
<View style={{flex: 1}}>
  ...
</View>
```