React Native - Basics - Flexbox Layout

- Dr Nick Hayward

A brief intro to the basics of flex layout in React Native app development.

Contents

Intro

React Native uses the *flexbox* algorithm to specify layout and design for its components, and their children.

An inherent benefit of *flexbox* layouts are their adaptation to multiple screen sizes, aspect ratios, and orientations.

There are many similarities between flexbox usage for CSS and React Native. However, there are also some known differences as well.

To quote the React Native documentation,

> > Flexbox works the same way in React Native as it does in CSS on the web, with a few exceptions. The defaults are different, with flexDirection defaulting to column instead of row, and the flex parameter only supporting a single number.

For React Native, there tends to be three predominant uses, including

- `alignItems`
- `flexDirection`
- `justifyContent`

`flexDirection`

By defining a component's `flexDirection`, we're setting the organisational pattern for its subsequent children. These might be set to a horizontal *row* or a vertical *column*. By default, `flexDirection` will be set to a column.

e.g.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    flexDirection: 'row',
  },
});
```

So, a `View` with the style for `container` will use all of the available screen space, and render its child components in a row pattern. One after another, cascading from row to row.

`justifyContent`

We may then update this style to define how child components start to fill each row, effectively setting their `justifyContent` value. Options include,

- `flex-start`
- `flex-end`
- `space-around`
- `space-between`

e.g.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    flexDirection: 'row',
    justifyContent: 'flex-end'
  },
});
```

`alignItems`

Align items offers a simple, complementary option to `flexDirection`.

So, if the direction for the primary axis, set using `flexDirection`, is *column*, `alignItems` will define the secondary axis as *row*.

Available options include,

- `flex-start`
- `flex-end`
- `center`
- `stretch`

However, there is a caveat to using the `stretch` value. We need to ensure that no fixed dimensions are set for any children of the flex component.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    flexDirection: 'column',
    justifyContent: 'flex-start',
    alignItems: 'stretch',
  },
});
```

more layout options

Further options may be specified as props, which we can add to a given component or stylesheet. Full details can be found at the following URL,

- Layout Props