Simon Källberg
Ylva Selling

Getting started with React Native

Introduction

In this guide we will be using React native and specifically look at

- **Simple Layout**
- **Callback function**
- **Navigate with different screens**

If you haven't installed react native yet, there is a great guide over at:
https://facebook.github.io/react-native/docs/getting-started

Simple Layout

So you wanna make an app in react native but is struggling with getting your components at the right place. In this layout guide we'll be the most common component in React Native "View" and style it using Flexbox. Flexbox provides a way to distribute space in a container to all items even if we don't know the size of the container and/or the container will change in size.

View

A view is simply a box, and is React natives equivalent to div-elements. A View can contain other elements and also nest other Views. If a View is empty, and no height and width is specified, it will not be visible on the screen as the height and width are zero.

ScrollView

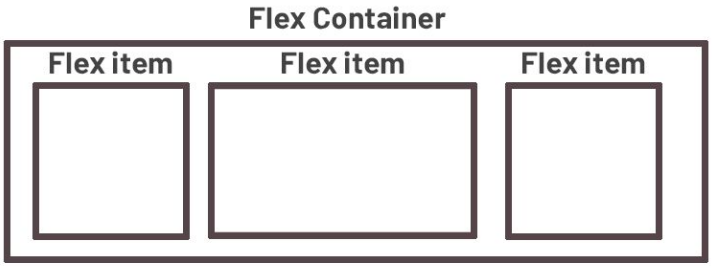
ScrollView is a type of view that enables scrolling and therefore the height of the scrollview can be infinite. If you put items in the view that are too large for the height it will enable a scrollview interface that will display the items in the bottom of the scrollview when you scroll down. As a result of the expandable height of the scrollview, it is not possible to access its height in child components. A child that has 50% height of the parent height will not be seen since the height is undefined.

SafeAreaView

SafeAreaView is another type of view that restricts the view to the areas of the screen which are safe to display items in. That means that the top area of the mobile screen that displays the clock and the like are excluded from the SafeAreaView.

Simon Källberg
Ylva Selling

Flexbox



In the styling of your components you can use Flexbox. Flexbox is a CSS layout mode to decide the position and size of the boxes, also called flex items. A flex item is a child element of the flex container. Flexbox is a single-direction layout concept, meaning that the flex items are either in a horizontal row or a vertical column. To change the direction you simply write:

```
container: {
        flexDirection: ‘row/column’
}
```

Where flexDirection: ‘row’ is used by default. Flexbox is a very flexible styling tool if the size of the window changes the space between the items will adjust automatically or you can choose to wrap the items to the next line if you want using *flex-wrap*.

```
container: {
        flex-wrap: ‘nowrap/wrap/wrap-reverse’
}
```

If there is space in the container there is properties flexbox can handle what you want to do with the free space. If you want your items to be aligned a certain way on the main axis and/or want the items to be distributed a certain way over the free space, you can use: *justify-content.*

```
container: {
        justify-content: ‘flex-start/flex-end/space-between/space-around/space-evenly/center’
}
```

Here flex-start/flex-end just decides if the items should be at the start (left-most) or the end (right-most) or the container. Space-between/space-around/space-evenly decides how the free space in the container is distributed, space-between makes the items have maximum free space between each other whereas space-around makes the items have equal space between each other. Space-evenly makes the items have equal spacing between each other.

There is a lot more to be learned about flexbox but this is the basics to get you working. If you want to learn more and try out all the different attributes we recommend: Flexboxfroggy that is a web-browser game to get comfortable using flexbox.

Simon Källberg
Ylva Selling

Callback functions in React Native

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action. In other programming languages this is similar to a function handle that is passed into a function as an argument. Since React Native uses javascript for it's functions, the callback syntax is the same as in plain vanilla javascript. Here is an example:

```
function greeting(name) {
  alert('Hello ' + name);
}

function processUserInput(callback) {
  var name = prompt('Please enter your name.');
  callback(name);
}

processUserInput(greeting);
```

Listeners in React Native

Listeners are a common part of javascript, where an element is accessed through javascript and then an eventListener is added, to "listen" to if something happens to the element it is attached to. An example in javascript is this:

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

However, in React Native, this is not really a part of the syntax any more. Since React Native uses a class based approach, the declaration of a "listening" function is quite different. Here is an example:

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(state => ({
      isToggleOn: !state.isToggleOn
    }));
  }
```

Simon Källberg
Ylva Selling

```
  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

The yellow marked code is what you need in order to declare a function that “listens” to button clicks. Mainly there are three parts in declaring this type of function:

1. In order to make state and other variables shown, such as props, to the functions it needs to be bound to the class. Therefore there is a “bind” line of code in the constructor.
2. The body of the function, i.e. what the function should execute when called, is declared as a normal javascript function, complete with arguments and return statement if needed.
3. In order to execute the function via a component that is rendered on the screen, it needs to be attached to a button or something similar. Therefore it is added as an “onClick”-event to a button in the “return” part of the “render” function.

When these three steps have been implemented, the functionality of the listener has been implemented in React Native.

Navigate between different screens

In some apps you might want to have different screens to your disposal, which can be similar to the tabs in a web browser. The common way to accomplish this in React Native is by using a StackNavigator. In order to use the StackNavigator, you need to first install the package that contains the stack navigator. This is done with either yarn or npm. These packages are needed for the stack navigator:

```
npm install --save react-navigation
yarn add react-native-gesture-handler
# or with npm
# npm install --save react-native-gesture-handle
yarn add react-navigation-stack
# or with npm
# npm install --save react-navigation-stack
```

Now you have all the components that are needed to create different screens! Here is an example of how you use them in your app:

```
import {createAppContainer} from 'react-navigation';
import {createStackNavigator} from 'react-navigation-stack';
```

Simon Källberg
Ylva Selling

```
const MainNavigator = createStackNavigator({
  Home: {screen: HomeScreen},
  Profile: {screen: ProfileScreen},
});

const App = createAppContainer(MainNavigator);

export default App;
```

Now you need to create the different screens. A screen is simply declared as a component class:

```
class HomeScreen extends React.Component {
  static navigationOptions = {
    title: 'Welcome',
  };
  render() {
    const {navigate} = this.props.navigation;
    return (
      <Button
        title="Go to Jane's profile"
        onPress={() => navigate('Profile', {name: 'Jane'})}
      />
    );
  }
}
```

And now you have an app with many screens!