Getting started with React Native

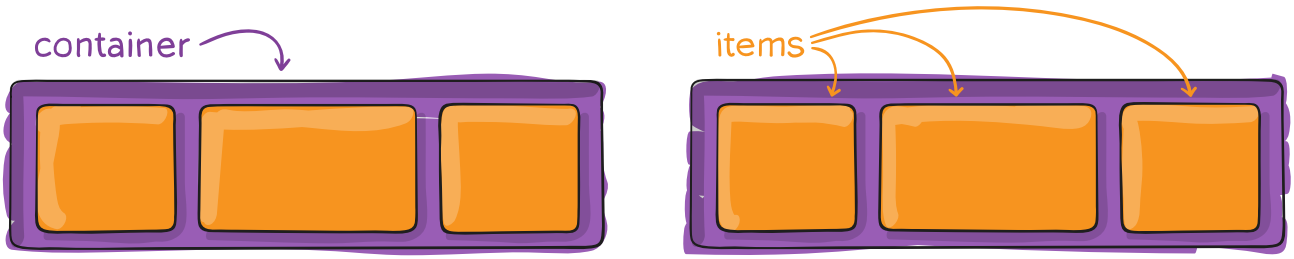
Layouts - Interaction & Callbacks - Navigation

Prequisite: Install and setup React Native for your system here

- Layouts

Layout in React Native is primarly composed of the components View and ScrollView.

A View is similar to a div-element in HTML but uses **flexbox** to position elements inside it by default. In the following image consider the "container" as a View component and the "items" as elements inside the view. Images from css-tricks.com.



To customize the view you use the **"style"** prop of the View. To specify how the view positions elements inside it certain **properties** can be used. Example of these properties are "flexDirection", "justifyContent" and "alignItems".

The following code is an example usage of a View. The View positions the elements inside itself on a row, horizontally across the screen, and centers them both horizontally and vertically inside the View.

```
import {View} from "react-native";

return (
    <View
        style={{
            flexDirection: "row",
            justifyContent: "center",
            alignItems: "center",
        }}>
        {/* Elements inside the view are centered */}
    </View>
);
```

There are other propeties as well: "flex-grow", "flex-shrink", "flex-wrap", the list goes on. To read about all the properties that can be used with flexbox we recommend you visit css-tricks.com which is a excellent resource to learn everything about flexbox.

A ScrollView is great component that is recommended to use when you have many elements or long lists in your app and need to be able to scroll through them. If the elements inside the ScrollView don't fit inside the

size of a single screen, the ScrollView will still render the rest of the items and make the container **scrollable**.

The following code is an example usage of a ScrollView. As you can see it is very easy to use, just place your list of elements inside it and it will take care of the rest.

```
import {ScrollView} from "react-native";

    return (
        <ScrollView>
            { /* Long list of elements can now be scrolled through. */ }
        </ScroView>;
    );
```

These two containers, View and ScrollView, are enough to create all sorts of complex layouts in your app. Use these together and nest them together to accomplish your desired layout.

A Text-component is a simple component used to **display text** in your app. The component is pretty straight forward to use, just put the text you want to display inside the component. Text components can be nested and also styled with the "style"-prop.

The following is an example of nested Text-components and also of customized font weight.

```
import {Text} from "react-native";

    return (
        <Text>
        This is regular text
            <Text style={{fontWeight: "bold"}}> and this is bold.</Text>
        </Text>
);
```

The following example creates a common layout: A View which will serve as a header-menu with different tabs at the top and beneath it a ScrollView which will contain a list of content.

```
const App = () => {
    return (
        <View>
            <View
                style={{
                    flexDirection: "row", //Display items on a row
                    justifyContent: "space-between", //Distribute the element
evenly
                    alignItems: "center", //Center the items on the horizontal
line
                }}>

                <Text title="Home"/>
                <Text title="About"/>
```

```
                <Text title="Contact"/>

            </View>

            <ScrollView> {/* Scrollable content */}

                <ListOfElments1 />
                <ListOfElments1 />
                <ListOfElments1 />

            </ScrollView>

        <View>
    );
}

export default App;
```

Styling and StyleSheets

We can use the "style"-prop in our component to customize our component, just as we can in **HTML** and **CSS**. However, writing the styles inline as a prop makes sharing and updating styles quite ineffective since the prop needs to be updated everywhere if you want a uniform design across your app.

This is where StyleSheets shine. Create and define your styles seperately and apply them to the elements that you want to have certain styling. By moving your style from the rendering your code will be **easier to read** and styles easier to **share and update™™ between elements.

See the simple example below.

```
import {StyleSheet} from "react-native";

const App = () => (
  <View style={styles.container}>
     <Text style={styles.text}>I'm using StyleSheets</Text>
     <Text style={styles.text}>Me too</Text>
  </View>
);

const styles = StyleSheet.create({
  container: {
    backgroundColor: "gray"
  },
  text: {
    color: "white",
    textAlign: "center",
    fontWeight: "bold"
  }
});
```

There you go! Now you got the basics down and can start using the different views to create your layout and also customize the look for each element. We also recommend that you dive into the documentation for these components to learn more about what you can do with them.

- Interaction & Callbacks

Interaction and Callbacks are an essential part of an application. Interactions and callbacks can be very complex but lets start with a simple example.

The following code renders a basic Button-component. The Button has an **"onPress"-property** which accepts a function which the button will call when it is pressed, this is the callback function of the button.

```
import {Button} from "react-native";

const callbackFunction = () => {
    //Do stuff related to the button here
    alert("Button pressed");
};

<Button
    title="Example button"
    onPress={() => callbackFunction()}
/>
```

An application often want to keep track of certain interactions the user has made to update the UI accordingly. Let start keeping track of the **state** of the application with the help of the React hook **useState()**. If you are unfamiliar with React hooks look into the documentation to get the basics down.

The following example shows a normal Button-component which will increase a counter that is updated and displayed to the user.

```
import React, {useState} from "react";
import {Button} from "react-native";

//State hook to keep track of the counter - we initialize the value to zero
const [counter, setCounter] = useState(0);

return (
    <>
        <Text>{counter}</Text> //Display the current counter
        <Button
            title="Increase counter!"

            //Use our state as the callback function for the button
            //and increase the counter by 1 for each click
            onPress={() => setCounter(counter + 1)}
        />
    </>
);
```

State is also commonly used to hide or show certain elements by conditionally rendering elements:

```
    import React, {useState} from "react";
    import {Button} from "react-native";

    //State hook to that will decide if
    const [showText, setShowText] = useState(false);

    return (
        <>
            <Button
                title="Toggle text view"
                //Use our state as the callback function for the button
                //and toggle the showText to true or false depending on its
current state
                onPress={() => setShowText(!showText)}
            />
            {showText && <Text>I'm hideable!</Text> /* This Text is only rendered
when "showText" is true */}
        </>
    );
```

Great, now you can handle interactions and callbacks in your app and make it feel ***responsive*** and ***alive***!

- Navigation

Now that you are familiar the basics of React Native, it is time to go into the basics of Navigation within a React Native app. The easiest way to get started with navigation is to use the React Navigation which is a navigation library that comes with built-in navigators that are easy to use and is customizable by the user.

Installation

```
# Using npm
npm install @react-navigation/native

# Using yarn
yarn add @react-navigation/native
```

Then we need to install the following dependencies:

Installing dependencies into an Expo managed project

```
expo install react-native-gesture-handler react-native-reanimated react-native-
screens react-native-safe-area-context @react-native-community/masked-view
```

Installing dependencies into a bare React Native project

```
# Using npm
npm install react-native-reanimated react-native-gesture-handler react-native-
screens react-native-safe-area-context @react-native-community/masked-view

# Using yarn
yarn add react-native-reanimated react-native-gesture-handler react-native-screens
react-native-safe-area-context @react-native-community/masked-view
```

Using React Navigation

Now we are ready to start using the newly added React Navigation library. First we need to import the **NavigationContainer** into our project and then wrap our whole app within the **NavigationContainer**. We also import the **createStackNavigator** which we are going to need when we create our different Stack screens. This need to be done in the root of our App, usually in the App.js file.

```
import {NavigationContainer} from "@react-navigation/native";
import {createStackNavigator} from "@react-navigation/stack";
```

We can then wrap the whole app in our **NavigationContainer**.

```
return <NavigationContainer>{/* Your app code here */}</NavigationContainer>;
```

Now it is time to use the **createStackNavigator** function that contains two functions: a **Navigator** and a **Screen** element. To write our different screens, we first need to inititate the **StackNavigator** like this:

```
const Stack = createStackNavigator();
```

We can then start to add **Stack Screens** inside our **NavigationContainer**. The **Stack Screens** is wrapped in a **Stack.Navigator** container. We provide each screen with a name and a component that is going to belong a specific screen.

```
<Stack.Navigator initialRouteName="Home">
    <Stack.Screen name="Home" component={HomeScreen} />
    <Stack.Screen name="ItemDetail" component={ItemDetail} />
</Stack.Navigator>
```

In our **Stack.Navigator** we also added the *initalRouteName*="*Home*" to configure so that our **HomeScreen** component is the first one to render when the app is started. The second screen **ItemDetail** could for example be a screen we want to navigate to when something in the **HomeScreen** is clicked.

The different screens also take options as props that we as users can choose to modify the behaviour and look of our different screens. For example we could add a background color to a screen so that the default background color i set for the whole screen. We can also disable the default header being shown when the screen renders.

```
<Stack.Screen
    name="Home"
    component={HomeScreen}
    options={{headerShown: false, cardStyle: {backgroundColor: "#001313"}}}
/>
```

Our App.js code would now look like this:

```
import React from "react";
import {NavigationContainer} from "@react-navigation/native";
import {createStackNavigator} from "@react-navigation/stack";

import HomeScreen from "./components/HomeScreen";
import ItemDetail from "./components/ItemDetail";

const Stack = createStackNavigator();

const App = () => {
    return (
        <NavigationContainer>
            <Stack.Navigator initialRouteName="Home">
                <Stack.Screen
                    name="Home"
                    component={HomeScreen}
                    options={{headerShown: false, cardStyle: {backgroundColor:
"#001313"}}}
                />
                <Stack.Screen
                    name="ItemDetail"
                    component={ItemDetail}
                    options={{headerShown: false, cardStyle: {backgroundColor:
"#001313"}}}
                />
            </Stack.Navigator>
        </NavigationContainer>
    );
};

export default App;
```

The next step is to use the *navigation* prop to be able to navigate between our two screens.
In our component **HomeScreen.js** we need to specify *navigation* as a passed down prop from our parent component **App.js**.

```
const HomeScreen = ({navigation}) => {
    return {/* The code of the HomeScreen component*/}
};
export default HomeScreen;
```

We can now use the *navigation* prop to navigate between the screens. A basic example could look like this:

```
const HomeScreen = ({navigation}) => {

    return (
        <View style={{flex: 1, width: "100%"}}> //View which wraps all of our
content.
            <View
                style={{
                    width: "90%" // fills out 90% of the parent view width
                    flex: 0.8 // fill out on the height 80%
                    flexDirection: "column", //Display items in a column
                    justifyContent: "space-between", //Distribute the elements
evenly in the available space
                    alignItems: "center", //Center the items on the vertical
                }}>

                <Button
                    title="Go to ItemDetails button"
                    onPress={() => navigation.navigate('ItemDetails')}
                />
            </View>
        <View>
    );
}

export default HomeScreen;
```

In the **Button** we use a callback function that uses the *navigation.navigate* prop were we specifiy that when the button is pressed, the app should navigate to our other screen, which we named **ItemDetails**.

To go back we could either create a button in the **ItemDetails** component that does the same thing, or we could use the **navigation.goBack()** function to revisit the screen we navigated from.

```
return (
    <>
        <Button title="Go back Home button" onPress={() =>
navigation.navigate('Home')}/>
        {/* or*/}
        <Button title="Go back button" onPress={() => navigation.goBack()}/>
    </>
);
```

Additionally, we could pass along props from our active component to the on we navigate to, this is very usefull when we need to handle states send data between different components.

```
const Home = ({navigation}) => {
    let item = {name: "Babu Frik", price: "129 kr", inStock: true};
    return (
        <>
            <Button title="Go to ItemDetails button" onPress={() =>
navigation.navigate('ItemDetails', item)}/>

        </>
    );
}

export default Home;
```

In the code above, we pass in the object *item* as an additional prop into the **ItemDetails** component, which we then can access in the component the same way as the *navigation* prop.

This was some of the basics of using the **React Navigation** library. For more details you should visit the React Navigation documentation to see all the features of the library.