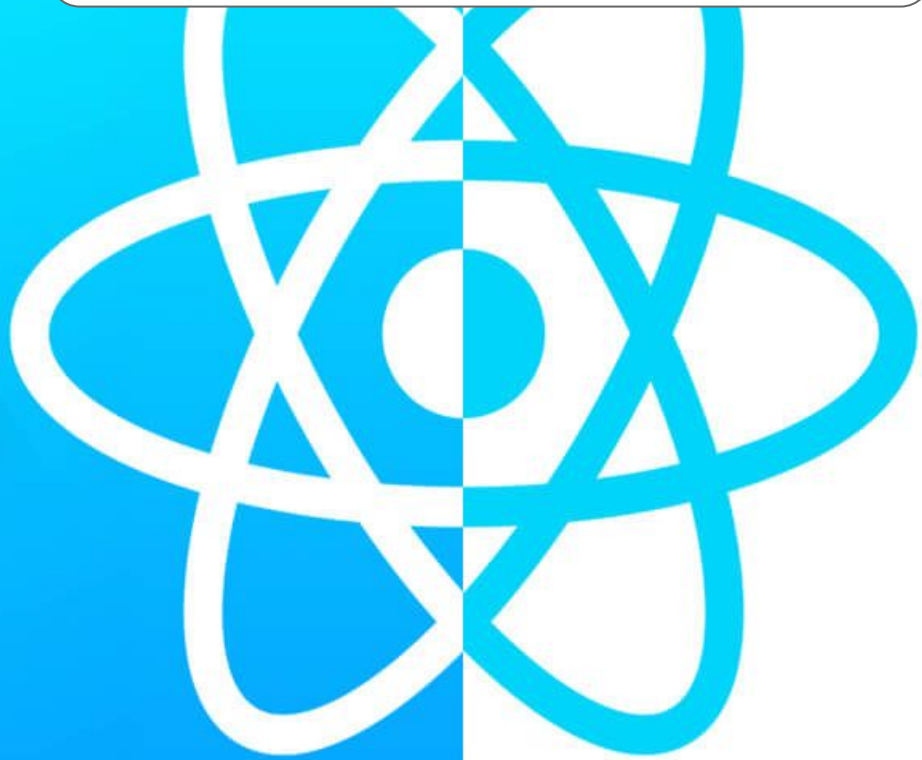
React Native



React Native

React Native is a JavaScript framework for building native mobile apps. It uses the React framework and offers large amount of inbuilt components and APIs.

Getting started

Install expo - https://facebook.github.io/react-native/docs/getting-started

Start a project with expo init prjname

Install expo on your phone.

Npm start and Scan the QR code frpm the terminal with the Expo app (Android) .

You can use -Android studio - https://developer.android.com/studio/run/managing-avds - I recommand that you'll install the latest android version in your virtual phone.

Open debugger

- **iOS Device**: Shake the device a little bit.
- **Android Device:** Shake the device vertically a little bit, or run `adb shell input keyevent 82` in your terminal window if your device is connected via USB.

- **iOS Simulator**: Hit `Ctrl-Cmd-Z` on a Mac in the emulator to simulate the shake gesture, or press `Cmd+D`.
- **Android Emulator:** Either hit `Cmd+M`, or run `adb shell input keyevent 82` in your terminal window.

The React Native View Component

The most fundamental component for building a UI, `View` is a container that supports layout with flexbox, style, some touch handling, and accessibility controls. `View` maps directly to the native view equivalent on whatever platform React Native is running on, whether that is a `UIView`, `<div>`, `android.view`, etc.

`View` is designed to be nested inside other views and can have 0 to many children of any type.

```
class ViewColoredBoxesWithText extends Component {
    render() {
      return (
        <View
          style={{
            flexDirection: 'row',
            height: 100,
            padding: 20,
          }}>
          <View style={{backgroundColor: 'blue', flex: 0.3}} />
          <View style={{backgroundColor: 'red', flex: 0.5}} />
          <Text>Hello World!</Text>
        </View>
      );
    }
```

The React Native Text Component

A React component for displaying text.

`Text` supports nesting, styling, and touch handling.

```
export default class BoldAndBeautiful extends Component {
    render() {
      return (
        <Text style={{fontWeight: 'bold'}}>
          I am bold
          <Text style={{color: 'red'}}>
            and red
          </Text>
        </Text>
      );
    }
  }
```

The `<Text>` element is unique relative to layout: everything inside is no longer using the flexbox layout but using text layout. This means that elements inside of a `<Text>` are no longer rectangles, but wrap when they see the end of the line.

Style your App

You **can**'t use **CSS** in **react**-native apps.
But you **can** use **CSS**-in-JS with styled-components

React Native components accepts specific styles, for example View won't accept a color property. **React Native** styles don't have all the **CSS** properties **you** might be used to having on the web. Media queries aren't built into **React Native**, #istandwithflexbox. **You can**'t pass in a plain JavaScript object into a component

The CSS properties are defined using the camelCase instead of hyphens. For example, you need to use backgroundColor instead of background-color.

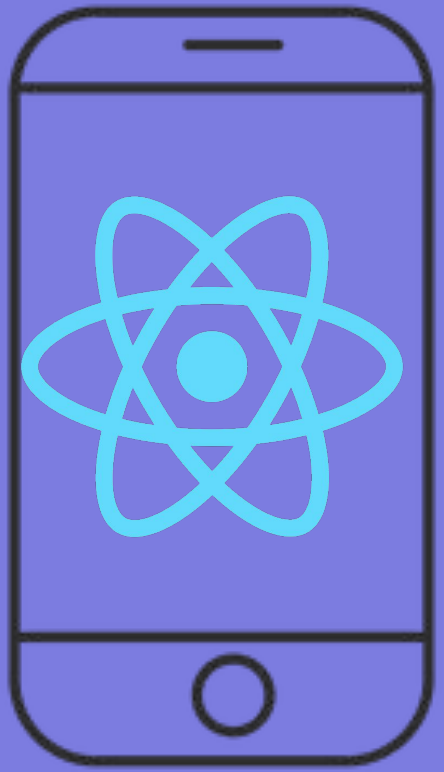
```
<View style={ { flex: 1, justifyContent: "center", alignItems: "center" } }></View>
```

APP.STYLE.JS

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

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  title: {
    fontSize: 32,
  }
});
export default styles
```

```
import styles from '../app.style'
import loadingBg from '../assets/loading.jpg'

export default function App({events}) {

  return (<View style={styles.container} >
            <ImageBackground
                style={styles.backgroundImage}
                source={loadingBg} >
            </ImageBackground>
          </View>)
}
```

Built in components



IMAGES

```
 <Image
     style={{width: 30, height: 30 , margin:10}}
     source={{uri: 'https://facebook.github.io/react-native/img/tiny_logo.png'}}
  />
```

```
import loadingImage from '../assets/loading.jpg'
<ImageBackground style={styles.backgroundImage} source={loadingImage} >
      whohoo
</ImageBackground>
```

Flat List

This is where `FlatList` comes into play. `FlatList` renders items lazily, when they are about to appear, and removes items that scroll way off screen to save memory and processing time.

`FlatList` is also handy if you want to render separators between your items, multiple columns, infinite scroll loading, or any number of other features it supports out of the box.

```
{
   events.map((article, index)=>{
     return <Text key = {index}>
     { article.title }
     </Text>
   })
 }
 <FlatList
   keyExtractor={(item, index) => index.toString()}
   data={events}
   renderItem={({ item }) => <Text>
                                {item.title}
                             </Text>}
 />
```

- Fully cross-platform.
- Optional horizontal mode.
- Configurable viewability callbacks.
- Header support.
- Footer support.
- Separator support.
- Pull to Refresh.
- Scroll loading.
- ScrollToIndex support.
- Multiple column support.

TouchableOpacity

A wrapper for making views respond properly to touches. On press down, the opacity of the wrapped view is decreased, dimming it.

Opacity is controlled by wrapping the children in an Animated.View, which is added to the view hierarchy. Be aware that this can affect layout.

```
<TouchableOpacity onPress={this._onPressButton}>
      <Image
            style={styles.button}
            source={require('./myButton.png')}
      />
</TouchableOpacity>
```

```
<FlatList style={eventStyles.list}
      keyExtractor={(item, index) => index.toString()}
      data={events}
      renderItem={({ item }) => <TouchableOpacity
          style={eventStyles.item}><Text
          style={eventStyles.title}>
          {item.title}</Text></TouchableOpacity>}
    />
```

Buttons

A basic button component that should render nicely on any platform. Supports a minimal level of customization.

If this button doesn't look right for your app, you can build your own button using TouchableOpacity or TouchableNativeFeedback. For inspiration, look at the source code for this button component. Or, take a look at the wide variety of button components built by the community.

```
<Button
          title="Press me"
          color="#f194ff"
          onPress={() => Alert.alert('Button with adjusted color pressed')}
        />
```

Text box

```
export default function EditableItem({ item }) {
    const [value, setValue] = React.useState(‘’);


    return <TouchableOpacity style={eventStyles.item}>
    {/* <TextInput value={value} onChangeText={(text)=>setValue(text)}></TextInput> */}

        <TextInput value={value} onChangeText={setValue}></TextInput>
    </TouchableOpacity>
}
```

And more components

https://facebook.github.io/react-native/docs/components-and-apis#basic-components

Navigation

Navigation

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

Navigation

```
import React, { useEffect, useState } from 'react';
import EventsScreen from './screens/EventsScreen'
import { createAppContainer  } from 'react-navigation'
import { createStackNavigator  } from 'react-navigation-stack'
import { AppLoading } from 'expo';
import { enableScreens } from 'react-native-screens';
import EditScreen from './screens/EditScreen';
enableScreens();

const AppNavigator = createStackNavigator({
    Home: EventsScreen,
    Details: AppLoading,
    Edit: EditScreen,
  },
  {
    initialRouteName: 'Home',
  });

export default createAppContainer(AppNavigator);
```

Navigation

```
function onAddEvent(){
  navigation.navigate('NewEvent',{})
}

return (<View style={styles.container} >
            <TouchableOpacity onPress={onAddEvent} ></TouchableOpacity>
        </View)
```

Navigation

```
function onEditEvent(){
  navigation.navigate('EditEvent',{id:event.id})
}

return (<View style={styles.container} >
            <TouchableOpacity onPress={onEditEvent} ></TouchableOpacity>
        </View)
```

Navigation header styling

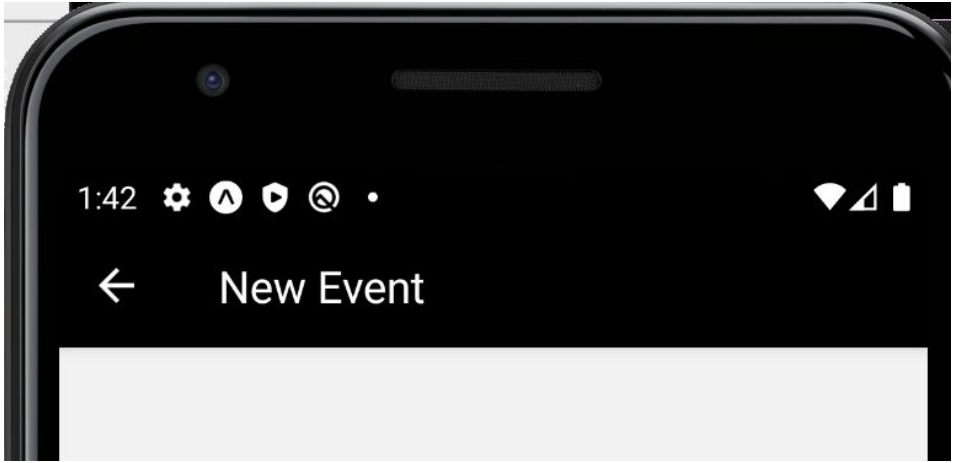
```
class EditScreen extends React.Component{
    static navigationOptions = ({ navigation, navigationOptions }) => {
        const { params } = navigation.state;
        return {
          title: params.id ? 'Edit Event' : 'New Event',
        };
      };

    render(){return <View></View>}
}
```

Navigation header styling - functional component

```
function EditScreen() {
    const [username, setUsername] = useState('');
    return <View style={styles.container} >
        <ImageBackground style={styles.backgroundImage} source={loadingBg} >
            <TextInput placeholder="Enter Email" value={username} onChangeText={setUsername} />
            <TextInput secureTextEntry={true} placeholder="Enter Password"/>
        </ImageBackground>
    </View>
}
EditScreen.navigationOptions = ({ navigation, navigationOptions }) => {
    const { params } = navigation.state;
    return {
        title: params.id ? 'Edit Event' : 'New Event',
    };
};
```

Navigation header styling

```
const AppNavigator = createStackNavigator({
    Home: EventsScreen,
    Details: AppLoading,
    NewEvent: EditScreen,
  }, {
    initialRouteName: 'Home',
    defaultNavigationOptions: {
      headerTintColor: '#fff',
      headerStyle: {
        backgroundColor: '#000',
      }
    },
    navigationOptions: {
      tabBarLabel: 'Home!',
    }
  }
);
```



Navigations life cycle

React Navigation emits events to screen components that subscribe to them. There are four different events that you can subscribe to: `willFocus`, `willBlur`, `didFocus` and `didBlur`. Read more about them in the API reference.

```
useEffect(() => {
  focusEvent = navigation.addListener('didFocus', () => {
    setLoading(true)
    load();
  });

  return () => focusEvent.remove()
}, []);
```

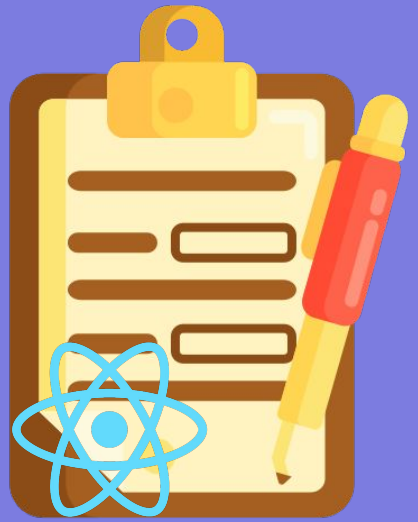
Refresh on reloading screen

```
export default function EventScreen({ navigation }) {
  const key = navigation.getParam("key");
  const [loading, setLoading] = useState(true);
  const [events, setEvents] = useState([]);

  useEffect(() => {
    setLoading(true)
    load();
  }, [key]);

  async function load() {
```

• • •

```
function EditScreen({ navigation }) {
    const id = navigation.getParam("id");

    async function onSaveEvent(){
        await eventService.saveEvent(event);
        navigation.navigate('Home' ,{key:new Date()})
    }
```

Forms



Forms

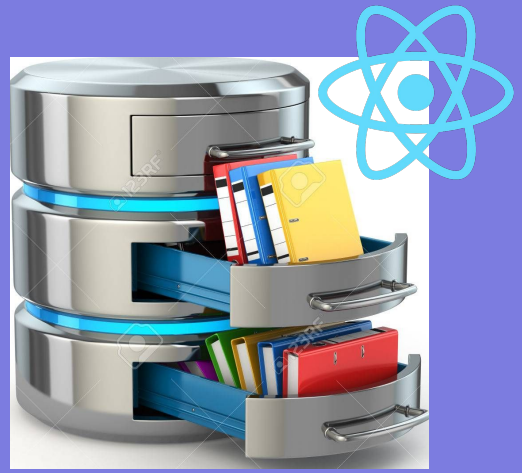
```
const LoginForm = () => {
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');
    return (
      <View>
        <Text> Login Form </Text>
        <View>
          <TextInput placeholder="Enter Email" value={username} onChangeText={setUsername} />
          <TextInput  secureTextEntry={true} placeholder="Enter Password"
          value={password} onChangeText={setPassword} />
        </View>
      </View>
    );
  };
```

User interface components

Render common user interface controls on any platform using the following components. For platform specific components, keep reading.

- Button - A basic button component for handling touches that should render nicely on any platform.
- Picker - Renders the native picker component on Android and iOS.
- Slider - A component used to select a single value from a range of values.
- Switch - Renders a boolean input.

Storage



Async storage

https://facebook.github.io/react-native/docs/asyncstorage

storageService.js :

```
import {AsyncStorage} from 'react-native';

export async function saveToStorage(key,data){
    await AsyncStorage.setItem(key,JSON.stringify(data));
}
export async function getFromStorage(key){
  data =   await AsyncStorage.getItem(key );
  if (!data) return undefined
  return JSON.parse(data);
}
```

`AsyncStorage` is an unencrypted, asynchronous, persistent, key-value storage system that is global to the app. It should be used instead of LocalStorage.