DEVS WORKSHOP 1

Learning Objectives

- Understand what React/React native is
- Understand the benefits of React/React Native
- Understand what expo is and why we would use it
- Understand components
- Understand the fundamental concepts of properties and state

What is React?

React is a JavaScript library for building user interfaces.

The three aspects of React which make it special are:

1) **Declarative** - React makes it painless to create interactive UIs. React will efficiently update and render just the right components when your data changes.

2) **Component-Based** – React allows us to build encapsulated components that manage their own state. We can then piece all these encapsulated components together to form complex UIs.

3) **Learn Once, Write Anywhere** – React doesn't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code

What is React Native?

React Native is an open-source mobile application framework created by Facebook. It is used to develop applications for Android, iOS and UWP.

React Native is like React, but it uses native components instead of web components as building blocks. So, to understand the basic structure of a React Native app, you need to understand some of the basic React concepts (such as JSX, components, state, and props).

Expo

Expo is a toolchain built around React Native to help you quickly start an app. It provides a set of tools that simplify the development and testing of React Native app.

Expo tools allow you to start a project without installing and configuring Xcode or Android Studio.

Expo CLI sets up a development environment on your local machine and you can be writing a React Native app within minutes.

Components

When developing apps with React/React native, anything you see on the screen is some sort of component.

A component can be pretty simple - the only thing that's required is a render function which returns some JSX to render.

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

Below, we have a simple component that displays the text “Hello, world!”.

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

export default class HelloWorldApp extends Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }
        <Text>Hello, world!</Text>
      </View>
    );
  }
}
```

Props

Most components can be customized when they are created, with different parameters. These creation parameters are called props.

For example, one basic React Native component is the **Image**. When you create an image, you can use a prop named **source** to control what image it shows. See below for an example of this, in the render function of Bananas, we pass in a uri as the prop **source** to the **Image** component.

```
import React, { Component } from 'react';
import { AppRegistry, Image } from 'react-native';

export default class Bananas extends Component {
  render() {
    let pic = {
      uri: 'https://upload.wikimedia.org/wikipedia/commons/d/de/Bananavarieties
    };
    return (
      <Image source={pic} style={{width: 193, height: 110}}/>
    );
  }
}
```

State

There are two types of data that control a component: props and state. props are set by the parent and they are fixed throughout the lifetime of a component. For data that is going to change, we must use state.

React native state is defined as a component property. It has a key – value pair. The key is used to access the value of a state and setState() method is use to set a new value of a key.

When the setState() method is called, the react native component is re-rendered to update changes in UI.

Take a look at the code snippet below to get an understanding of how state works.

```
class Form extends React.Component {

  constructor (props) {
     super(props)
     this.state = {
       input: ''
     }
  }

handleChangeInput = (text) => {
    this.setState({ input: text })
  }

  render () {
    const { input } = this.state

    return (
       <View>
          <TextInput style={{height: 40, borderColor: 'gray', borderWidth: 1}}
            onChangeText={this.handleChangeInput}
            value={input}
          />
        </View>
      )
    }
 }
```

In the above code snippet you can see a Form class with an input state. It renders a text input which accepts the user's input. Once the user inputs the text, the onChangeText is triggered which in turn calls setState on input.

The setState() method triggers a re-rendering of the component again, and the UI is now updated with the user's latest input. This simple example illustrates how state within a component can be updated and its usage.