

CS 5520

MOBILE APPLICATION DEVELOPMENT

Week 2

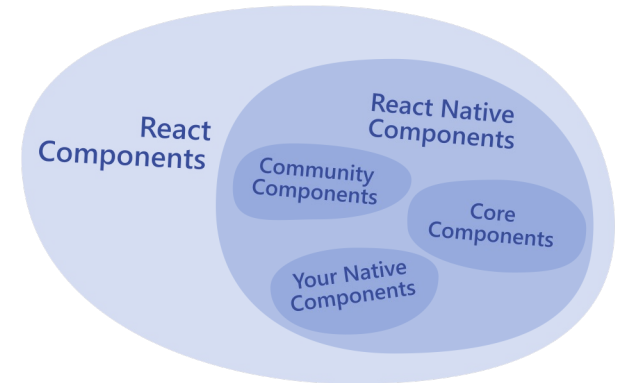
NEDA CHANGIZI

Today's Learning Outcomes

- To learn what are the properties (props) in React
- To explore the states available in React
- To become familiar with some React Native core components

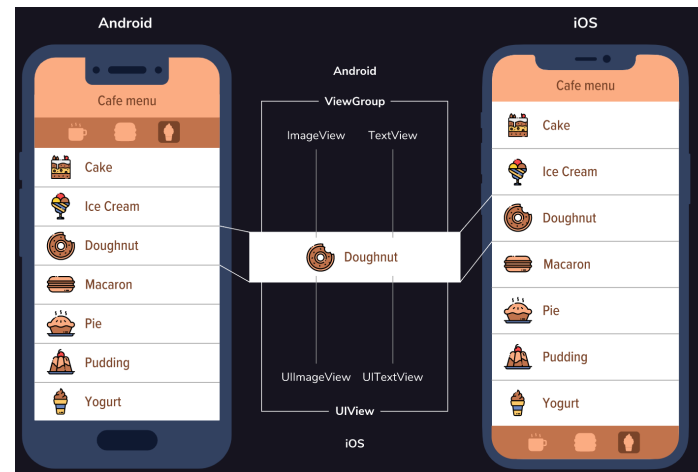
React Native Components

- All React Native apps are made of components
 - Small reusable pieces of your app, all working together



<https://reactnative.dev/docs/intro-react-native-components>

- React Native components are rendered into a counterpart native components.
 - Do all core components look the same on all platforms?
 - [Apple Design Guidelines](#)
 - [Android Design Guidelines](#)



Core Components

- Essential, ready-to-use Native Components. Most apps will end up using one of these basic components:

View The most fundamental component for building a UI.	Text A component for displaying text.
Image A component for displaying images.	TextInput A component for inputting text into the app via a keyboard.
ScrollView Provides a scrolling container that can host multiple components and views.	StyleSheet Provides an abstraction layer similar to CSS stylesheets.

- Which React Native core components are being used in App.js?
 - Note the import statement from 'react-native' at the top
- React Native uses the same API structure as React components. Let's do a quick intro on React.

Componentizing!

Whatever a function component returns is rendered as a React element

```
const Cat = () => {  
  return (  
    <Text>Hello, I am your cat!</Text>  
  );  
}  
  
export default Cat;
```

Function component

```
class Cat extends Component {  
  render() {  
    return (  
      <Text>Hello, I am your cat!</Text>  
    );  
  }  
}  
  
export default Cat;
```

Class component

- [Export and import cheatsheet](#)

What is JSX?

- React and React Native use JSX (JavaScript as XML)
 - Saved in a .js file
 - JSX elements can go anywhere JavaScript expressions can go

a JSX expression must
have exactly one
outermost element

```
<View style={styles.container}>  
  <Text>Open up App.js to start working on your app!</Text>  
  <StatusBar style="auto" />  
</View>
```

JSX elements can
have attributes.

Activity – JSX

- In App.js, inside the App() function, before the return statement, add a const variable name with the value being the name of your app. Update the <Text> component to use the name variable.
 - Hint: curly braces create a portal into JS functionality in your JSX!
 - You can do JS function calls inside {} in JSX



Open up App.js to start working on The Awesome App!!

Props

- Props (short for properties): arbitrary input to React components.
 - Can be used to customize a component, pass data to components, etc.
 - A mix of the two mental models above: HTML attributes and function parameters.

```
const Cafe = () => {  
  return (  
    <View>  
      <Cat name="Maru" />  
    </View>  
  );  
}
```

```
const Cat = (props) => {  
  return (  
    <View>  
      <Text>Hello, I am {props.name}!</Text>  
    </View>  
  );  
}
```

- What props can you identify in App.js?

Activity – Props

- Create a new folder called "components" and make a new file called Header.js in that folder.
 - Add a `<View>` component with a `<Text>` component inside it.
 - What happens if you write some text without wrapping it in `<Text>`?
 - Import the new `<Header>` component in App.js and replace the `<Text>` component
 - The `<App>` component is a parent component and the `<Header>` component is a child component
- Pass the application name to Header component as a variable using props.
 - Hint: [props](#): All JSX attributes and children of a user-defined component are passed to it as a single object.
 - [React Props Cheatsheet: 10 Patterns You Should Know](#)
 - [Object destructuring](#)
 - [Typechecking With PropTypes](#)

State hook

- State is like a component's personal data storage, it gives your component memory!
- `useState` takes the initial value for a state variable and return an array with the current state value and a function that lets you update it.

`[<getter>, <setter>] = useState(<initialValue>)`

array destructuring

```
const [age, setAge] = useState(42);  
const [fruit, setFruit] = useState('banana');  
const [todos, setTodos] = useState([ { text: 'Learn Hooks' } ]);
```

- The initial state argument is only used during the first render.
- During the next renders, `useState` gives us the current state.

- Setting a state variable causes the component to re-render!

<TextInput> Component

- TextInput's props:
 - Provide configurations such as auto-correction, placeholder, keyboard types, etc.
 - Event handlers such as onChangeText, OnKeyPress, OnFocus, etc.
 - The value to this kind of props should be a function

```
<TextInput
  style={styles.input}
  onChangeText={onChangeNumber}
  value={number}
  placeholder="useless placeholder"
  keyboardType="numeric"
/>
```

- Controlled component: An input form element whose value is controlled by React's state variable
 - Keep input's value in sync with a state variable

Activity - TextInput

- Add a `<Text>` and a `<TextInput>` in App.js. Create a state variables to keep track of the value of `<TextInput>` element:
 - `value` attributes of the `<TextInput>` should be equal to the related state variable
 - [`onChangeText`](#) attribute of the `<TextInput>` should be a function that receives the new text and call the set state function with the received value
 - Show the text that user enters in the `<Text>` component.

<Button>

- Button has 2 required props. What are they?
- Move the <TextInput> to a new component called Input.js.
- We need to be able to pass the text the user has entered back to App.js.
 - Child component can receive and call a callback function from the parent
- Define a function in App.js and pass it as a prop to Input.js
- Add a Button called Confirm in Input.js and as its onPress prop to call the function that is passed to the component and pass the entered text in it.
- Clear the state variable that's tracking the text when the button is pressed

<Modal>

- Let's wrap the code in Input.js in a <Modal> component.
 - You need to add a container styling to the View in Input.js
- By default, the Modal is always visible.
- Define a state variable in App.js to keep track of modal's visibility.
- Pass the state variable to Input.js and use it as Modal's [visible](#) prop
- Add a Button in App.js that would make the Modal visible when it's pressed.
- Update the function passed to Input component to also make the modal invisible when user has added a goal
- Add a Cancel button to the component to dismiss the modal

<Image> Component

- Add an <Image> component in the Input component. Practice setting the Source prop with two different methods:
 - From a URL: <https://cdn-icons-png.flaticon.com/512/2617/2617812.png>
 - A local resource: save the above image in your codebase
- Set the style prop to width:100 and height:100

Debugging

- Different kinds of errors that might happen in your app:
 - Syntax error
 - Logical errors
 - Styling, layout, UX errors
- How to Debug?
 - Read the error messages!
 - `console.log()`
 - Chrome Debugger + Breakpoints
 - Bring up the developer menu on the simulator/device and select "Debug Remote JS"
 - On the browser page that opens navigate to Source tab in developer tools and set breakpoints
 - <https://www.npmjs.com/package/react-devtools>