CUI Abbottabad

Department of Computer Science

Mobile Application Development

Lecture Topic

Introduction to React Native

Agenda

- Introduction to React Native
- Views, Native Components, Core components
- JSX
- Props
- state

Introduction to React Native

- React Native is an open source framework for building Android and iOS applications using React and the app platform's native capabilities.
- With React Native, you use JavaScript to access your platform's APIs as well as to describe the appearance and behavior of your UI using React components: bundles of reusable, nestable code. You can learn more about React in the next section. But first, let's cover how components work in React Native.

- **Native Components**
- In Android development, you write views in Kotlin or Java; in iOS development, you use Swift or Objective-C. With React Native, you can invoke these views with JavaScript using React components. At runtime, React Native creates the corresponding Android and iOS views for those components. Because React Native components are backed by the same views as Android and iOS, React Native apps look, feel, and perform like any other apps. We call these platform-backed components Native Components.

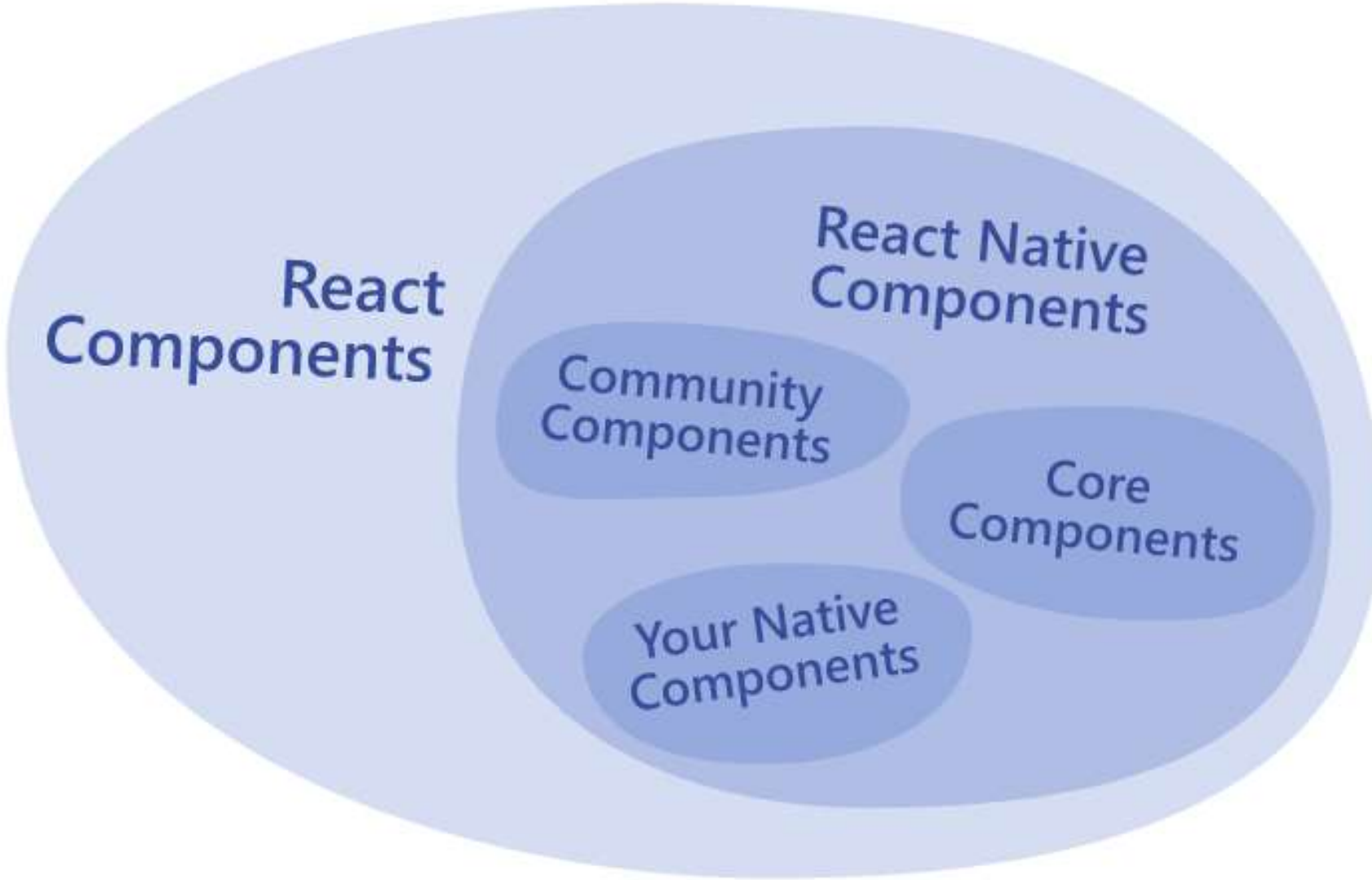
- **Native Components**
- React Native comes with a set of essential, ready-to-use Native Components you can use to start building your app today. These are React Native's Core Components.
- React Native also lets you build your own Native Components for Android and iOS to suit your app's unique needs. We also have a thriving ecosystem of these community-contributed components. Check out Native Directory to find what the community has been creating.
- function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand

- **Native Components**
- React Native also lets you build your own Native Components for Android and iOS to suit your app's unique needs. We also have a thriving ecosystem of these community-contributed components. Check out Native Directory to find what the community has been creating.

Core Components

React Native has many Core Components for everything from controls to activity indicators. You can find them all documented in the API section. You will mostly work with the following Core Components:

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrolling `<div>`	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Displays, styles, and nests strings of text and even handles touch events
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Displays different types of images
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	A generic scrolling container that can contain multiple components and views
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Allows the user to enter text


React
Components
React Native
Components
Community
Components
Core
Components
Your Native
Components

React Fundamentals

- We’re going to cover the core concepts behind React:

- components
- JSX
- props
- state

React Components

- Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.

- Components come in two types, Class components and Function components

- **Function Components**
- A function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand

Your component starts as a function:

```
const Cat = () => {};
```

You can think of components as blueprints. Whatever a function component returns is rendered as a **React element.** React elements let you describe what you want to see on the screen.

- Code example

- **JSX**
- React and React Native use JSX, a syntax that lets you write elements inside JavaScript like so: <Text>Hello, I am your cat!</Text>. The React docs have a comprehensive guide to JSX you can refer to learn even more.
- Because JSX is JavaScript, you can use variables inside it. Here you are declaring a name for the cat, name, and embedding it with curly braces inside <Text>
- Code example
- Any JavaScript expression will work between curly braces, including function calls like {getFullName("Rum", "Tum", "Tugger")}:
- Code example

- **Custom Components**
- You've already met React Native's Core Components. React lets you nest these components inside each other to create new components. These nestable, reusable components are at the heart of the React paradigm.

- For example, you can nest Text and TextInput inside a View below, and React Native will render them together:
- Code example

- **Props**
- Props is short for “properties”. Props let you customize React components. For example, here you pass each <Cat> a different name for Cat to render:
- Code example:

- **State**
- While you can think of props as arguments you use to configure how components render, state is like a component's personal data storage. State is useful for handling data that changes over time or that comes from user interaction. State gives your components memory!
- As a general rule, use props to configure a component when it renders. Use state to keep track of any component data that you expect to change over time.
- The following example takes place in a cat cafe where two hungry cats are waiting to be fed. Their hunger, which we expect to change over time (unlike their names), is stored as state. To feed the cats, press their buttons—which will update their state. Code example

- **Hook**
- You can add state to a component by calling React's useState Hook. A Hook is a kind of function that lets you "hook into" React features. For example, useState is a Hook that lets you add state to function components.
- Calling useState does two things:
- it creates a "state variable" with an initial value—in this case the state variable is isHungry and its initial value is true
- it creates a function to set that state variable's value—setIsHungry
- It doesn't matter what names you use. But it can be handy to think of the pattern as [<getter>, <setter>] = useState(<initialValue>). Code example