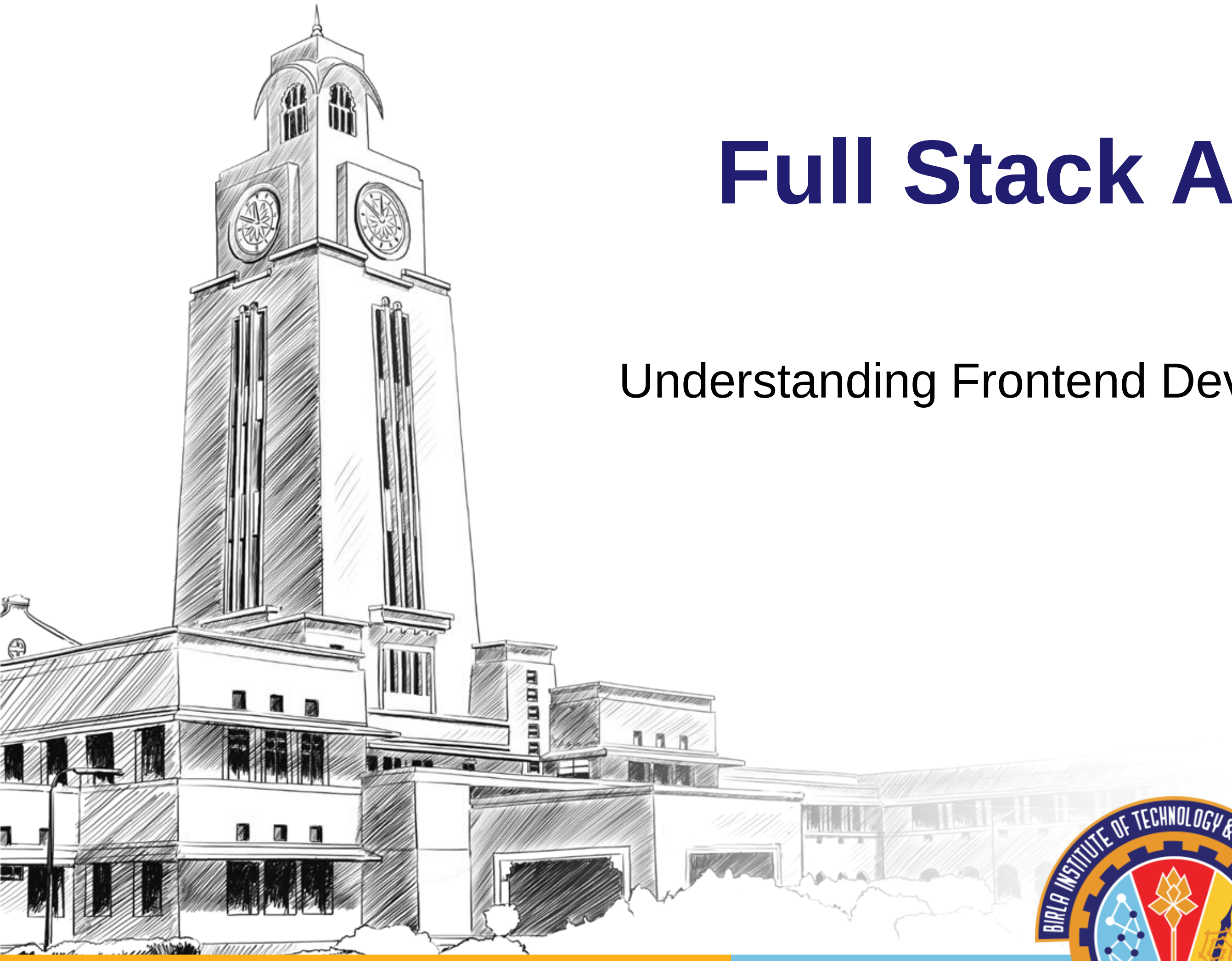
Full Stack Application Development



Understanding Frontend Development

ReactJS

Akshaya Ganesan

ReactJS

React - Introduction

- React is a JavaScript library for rendering user interfaces (UI).
- Open-source
- It is maintained by Meta (formerly Facebook) and a community of individual developers and companies.

Features

- What made React stand out and popular?
  - React adheres to the declarative programming paradigm
  - React lets you break down the UI into reusable components
  - Streamlines the process of building and composing components
  - React uses a virtual DOM
  - It is easy to learn and use

Component based Approach

- React is based on a component-based architecture that allows developers to break down their user interface into small, reusable components.
- This makes it easier to manage and maintain complex UI
- Developers can focus on developing and testing individual components separately.
- Each component consists of well-defined functionality that can be inserted

  into an application without requiring modification of other components

Declarative-What React Simplifies

- Consider the task of adding a element

```
const target = document.getElementById("target");
const wrapper = document.createElement("div");
const headline = document.createElement("h1");

wrapper.id = "welcome";
headline.innerText = "Hello World";

wrapper.appendChild(headline);
target.appendChild(wrapper);
```

```
const { render } = ReactDOM;
const Welcome = () => (
  <div id="welcome">
    <h1>Hello World</h1>
  </div>
);
render(<Welcome />,
document.getElementById("target"));
```

Virtual DOM

- React uses a virtual DOM, which is a lightweight representation of the actual DOM.
- React updates the virtual DOM instead of the actual DOM
- React then compares the virtual DOM with the actual DOM and only updates the necessary parts of the UI

Virtual DOM

- Every time the DOM changes, the browser has to do two intensive operations: repaint and reflow
- Whenever a change is required , React marks that Component as **dirty.**
- React updates the Virtual DOM relative to the components marked as dirty
- React batches much of the changes and performs a unique update to the real DOM.
- Repaint and Reflow the browser must perform to render the changes are executed just once

Important Javascript features

- Data types
- Using var, let and const
- Conditionals and Loops
- Using objects, arrays and functions
- ES6 Arrow functions
- In-built functions such as map(), forEach() and promises.
- Destructuring Arrays and Objects
- Modules

Create react app

- Create React App (CRA) is officially deprecated by the React team, primarily because it has limitations compared to newer, more flexible tools
- Lack of Configurability
- Outdated Technology
- Maintenance Overhead

Alternatives to Create React App

- **Vite**: Known for its speed, Vite is optimized for modern JavaScript and React projects. It provides near-instant startup, fast hot-module replacement (HMR), and excellent configurability. Vite has become very popular for both small and large-scale React applications.
- **Next.js**: Next.js is versatile and can handle single-page applications. It offers built-in routing, API routes, and server-side rendering (SSR) options, making it ideal for production-level React apps.
- Remix is a framework that emphasizes full-stack React applications with a focus on web standards and optimization for faster performance. It’s a good choice for projects where routing and server-side data fetching are important.
- Parcel is a zero-config bundler that’s easy to use and has great performance. It's a viable option for those who prefer minimal setup while still needing good development speed.

Folder Structure

- my-app/
- README.md
- node_modules/
- package.json
- public/
- index.html
- favicon.ico
- src/
- App.css
- App.js
- App.test.js
- index.css
- index.js
- logo.svg

React Elements

- The elements that make up an HTML document become DOM elements when the browser loads HTML and renders the user interface.
- The browser DOM is made up of DOM elements.
- Similarly, the React DOM is made up of React elements.
- A React element is a description of what the actual DOM element should look like.

- Create a React element to represent an h1 using React.createElement:
- `React.createElement("h1", { id: “listitem-0” }, “Web Technologies");`

- During rendering, React will convert this element to an actual DOM element:
- `<h1 id=“listitem-0">Web Technologies</h1>`

ReactDOM

- ReactDOM contains the tools necessary to render React elements in the browser.
- ReactDOM contains the render method.

- `const Litem1 = React.createElement("h1", { id: "listitem-0" }, "Web Technologies");`
- `ReactDOM.render(Litem1, document.getElementById("root"));`

JSX

- const element = <h1 id=“item1”>Web Technologies</h1>;

- const element=React.createElement(‘h1’, {id:’item1’}, ‘Web Technologies’);

- It creates Javascript object.

A JavaScript extension that allows us to define React elements using a tag-based syntax

JSX

- const name = 'John Doe';
- const element = <h1>Hello, {name}</h1>;

- ReactDOM.render(
-   element,
-   document.getElementById('root')
-);

Changes to be noted

- class becomes className
  - Due to the fact that JSX is JavaScript, and class is a reserved word
  - <p className="description">

- for which is translated to htmlFor

- Inline Style : CSS in React
- Instead of accepting a string containing CSS properties, the JSX style attribute only accepts an object
- var divStyle = {
- color: 'white'
- }
- ReactDOM.render(<div style={divStyle}>Hello World!</div>, mountNode)

Mapping Arrays with JSX

- To render multiple JSX elements in React, you can loop through an array with the .map() method and return a single element.

- function courseListItems() {
-   const courses = [“Web Technologies", “Java", “C++"];

-   return courses.map((course) => <li key={course}>{course}</li>);
- }

- add a unique key to identify each list item uniquely

React Fragments

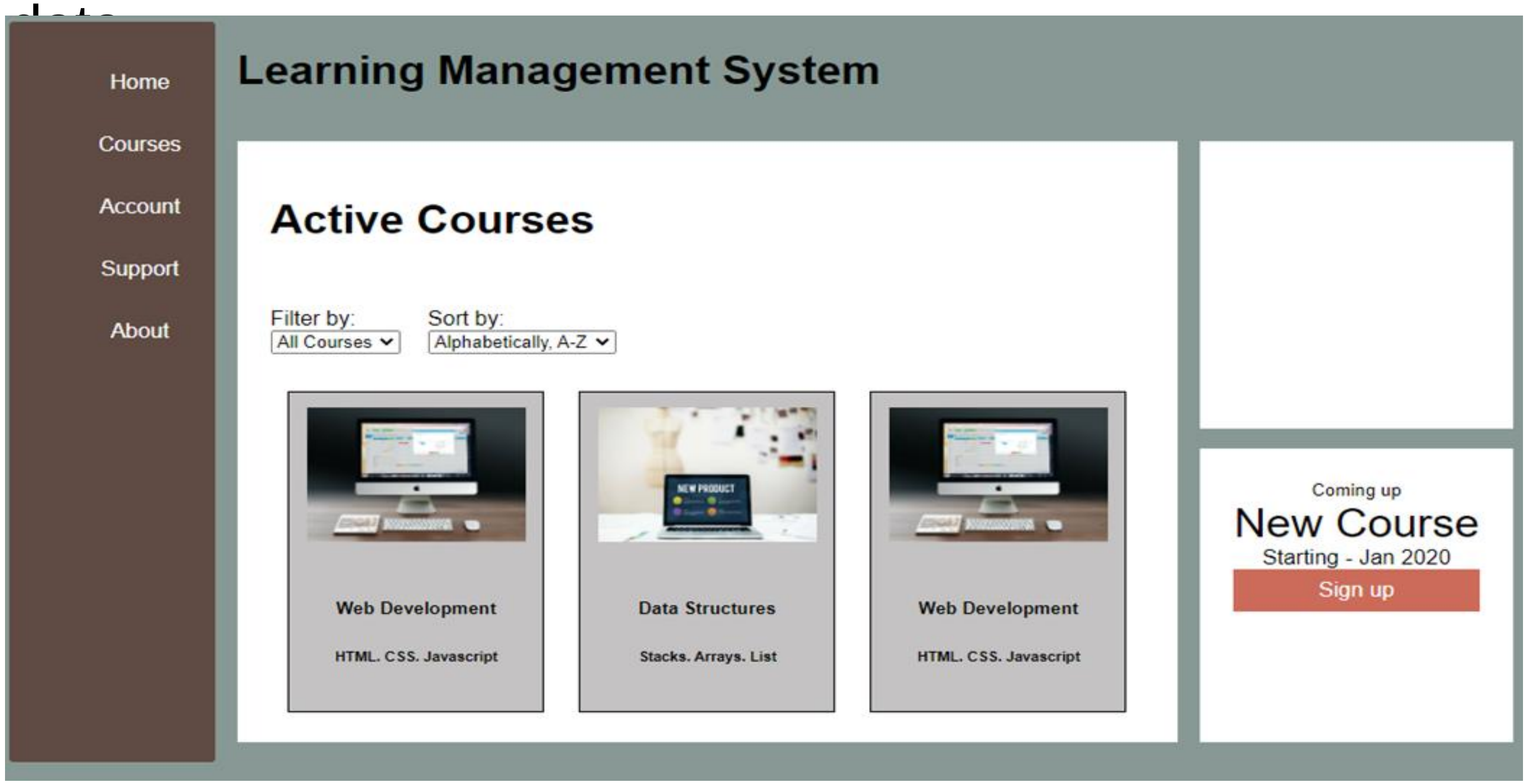
- We render Adjacent elements (two siblings) using a React fragment.

- function listitem({ name }) {
-   return (
-     <h1> {name}</h1>
-     <p>This is the first list item.</p>
-   );
- }

```
function listitem({ name }) {
  return (
    <React.Fragment>
    <h1> {name}</h1>
    <p>This is the first list item.</p>
   <React.Fragment>
  );
}
```

Components

- Components let you split the UI into independent, reusable pieces.
- Components allow us to reuse the same structure with different pieces of



Learning Management System

Active Courses

Filter by: All Courses

Sort by: Alphabetically, A-Z

Web Development
HTML. CSS. Javascript

Data Structures
Stacks. Arrays. List

Web Development
HTML. CSS. Javascript

Coming up
New Course
Starting - Jan 2020

Types

- Functional Components

- function Welcome(props) {
-   return <h1>Hello, {props.name}</h1>;
- }

- Class Components

- class Welcome extends React.Component {
-   render() {
-     return <h1>Hello, {this.props.name}</h1>;
-   }
- }

Rendering a Component

- function Course(props) {
-   return <h1>Course: {props.name} , {props.credits} </h1>;
- }

- const element = <Course name=“Java“ credits=“4” />;
- ReactDOM.render(
-   element,
-   document.getElementById('root')
-);

Props

- Props is how Components get their properties.
- Starting from the top component, every child component gets its props from the parent.
- In a function component, props are available by adding props as the function argument:
- In a class component, props are passed by default.
- They are accessible as this.props in a Component instance.
- When initializing a component, pass the props in a way similar to HTML attributes:

- Props are Read-Only
- Whether you declare a component as a function or a class, it must never modify its own props.

Presentational vs container components

- In React components are often divided into 2 big buckets:
  - presentational components and
  - container components.
- Presentational components are mostly concerned with generating some markup to be
- outputted.
- They don't manage any kind of state, except for state related the presentation.
- Container components are mostly concerned with the "backend" operations.
- They might handle the state of various sub-components.
- They might wrap several presentational components.
- They might interface with Redux.
- Presentational components are concerned with the look,
- container components are concerned with making things work.

State

- React Class components has a built-in state object.
- The state object is where you store property values that belongs to the component.
- When the state object changes, the component re-renders.

State in Class Component

- class JobList extends Component {
-   constructor(props) {
-     super(props);
-     this.state = {cart: [{name:'carservice'}]
- };
-   }
-

Points to be Noted

- Do Not Modify State Directly - Use setState()
  - For example, this will not re-render a component:
  - this.state.comment = 'Hello';

- State Updates May Be Asynchronous
- React may batch multiple setState() calls into a single update for performance.
- Because this.props and this.state may be updated asynchronously, you should not rely on their values for calculating the next state.

- A component may choose to pass its state down as props to its child components

State in Functional components

- Earlier, Functional Components were stateless.
- React Hooks made it possible to have state in Function Components.
- Hooks are a new addition in React 16.8.
- They let you use state without writing a class.
- Hooks allow you to reuse stateful logic without changing your component hierarchy.

Conditional Rendering

- **Conditional rendering** as a term describes the ability to render different UI markup based on certain conditions.
- Conditional rendering in React works the same way conditions work in JavaScript.

- Use JavaScript operators like if or the conditional operator to create elements representing the current state, and let React update the UI to match them.

- If/else
- element variables
- Ternary operator
- Short Circuit Evaluation with &&

State in Functional components

- Earlier, Functional Components were stateless.

- React Hooks made it possible to have state in Function Components.

- Hooks are a new addition in React 16.8.

- Hooks are functions that let you “hook into” React state and lifecycle features from function components.

Benefits

- They let you use state without writing a class.
- It’s hard to reuse stateful logic between components
- Hooks allow you to reuse stateful logic without changing your component hierarchy
- Complex components become hard to understand.
- In many cases it’s not possible to break these components into smaller ones because the stateful logic is all over the place.

Example

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Hooks

- React provides a few built-in Hooks like useState.

- You can also create your own Hooks to reuse stateful behavior between different components.

- Rules of Hooks
- Hooks are JavaScript functions, but they impose two additional rules
- Only call Hooks at the top level. Don’t call Hooks inside loops, conditions, or nested functions.
- Only call Hooks from React function components. Don’t call Hooks from regular JavaScript functions.

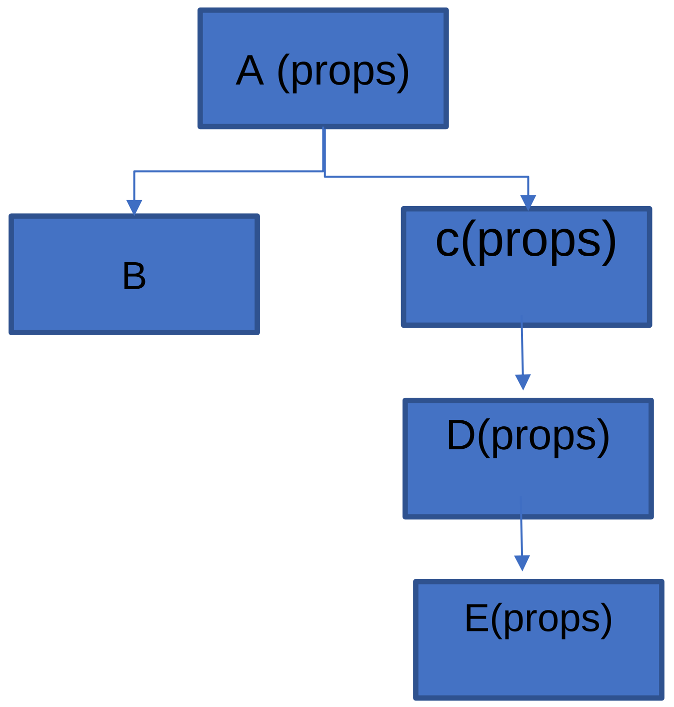
Built in Hook

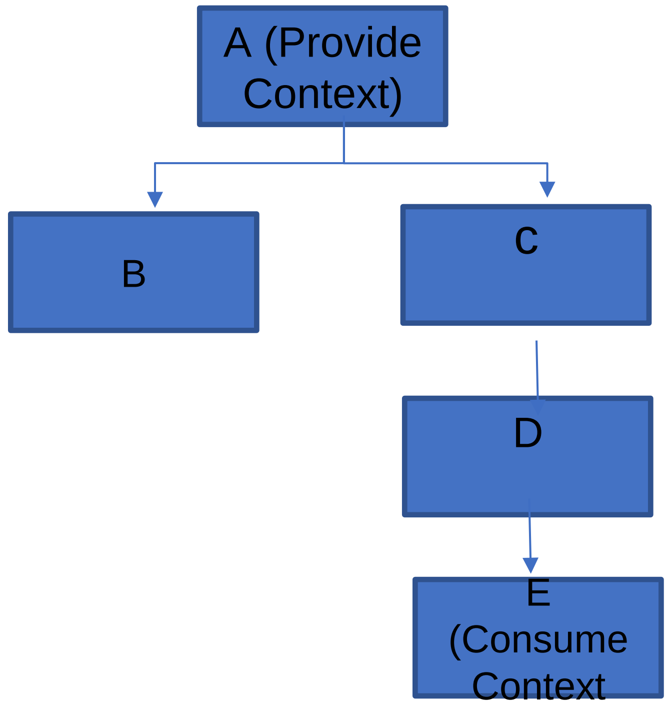
- useState
- const [count, setCount] = useState(0);

- useEffect
- By using this Hook, you tell React that your component needs to do something after render.
- useEffect Hook is componentDidMount, componentDidUpdate, and componentWillUnmount combined.

Context

- In a typical React application, data is passed top-down (parent to child) via props. When you had to pass props several components down your component tree. It results in prop drilling.





React Context

- There are two use cases when to use it:
- When your React component hierarchy grows vertically in size and you want to be able to pass props to child components without bothering components in between.
- When you want to have advanced state management in React. Doing it via React Context allows you to create a shared and global state.

Thank You!