ReactJs

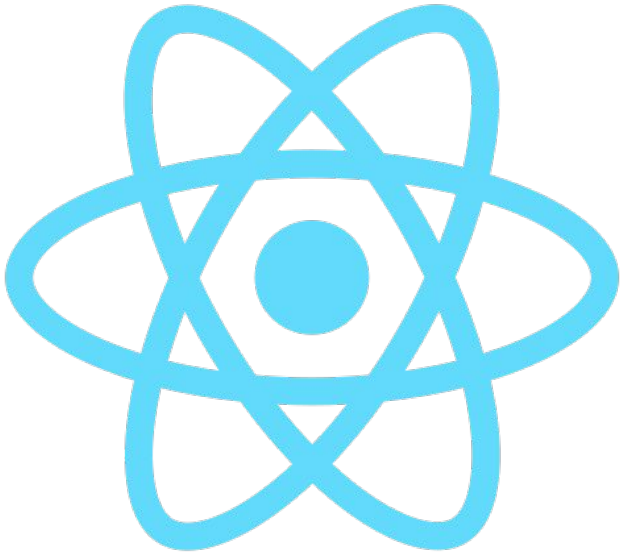
Prerequisite

- HTML5
- CSS
- Basic Javascript -ECMAScript 6

Ref:

- https://www.w3schools.com/html/default.asp
- https://www.w3schools.com/css/default.asp
- https://www.w3schools.com/js/default.asp

What is ReactJS?

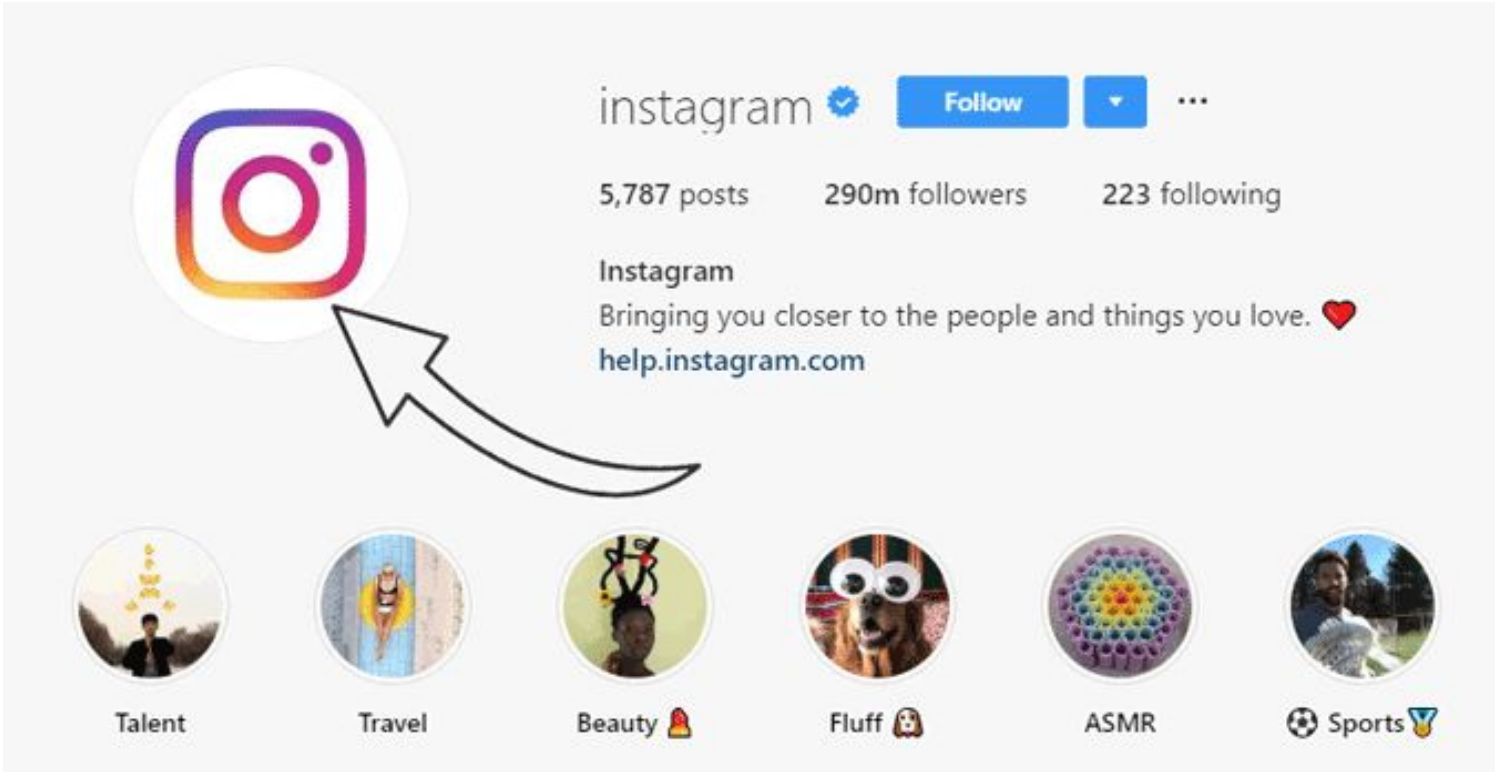


What is ReactJS?

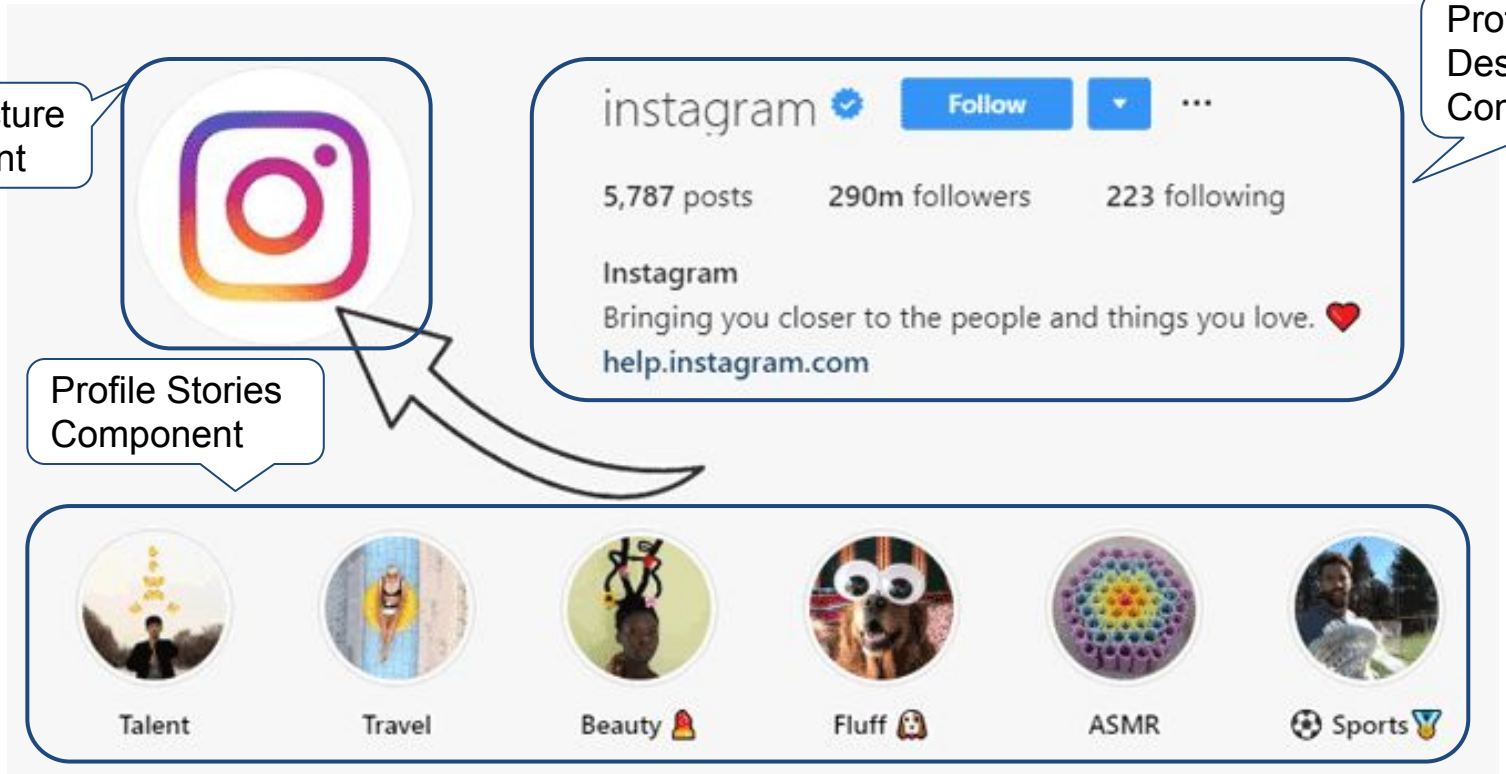
- ReactJs is a JavaScript library for building fast and interactive user interfaces for the web as well as mobile Applications
- **!! ReactJs is not a framework !!**
- It is an open-source, reusable component-based front-end library.
- ReactJs was created by **Jordan Walke,** who was a software Engineer at **Facebook.**
- ReactJs is managed by Facebook now.

Lets see how react works in real-time !

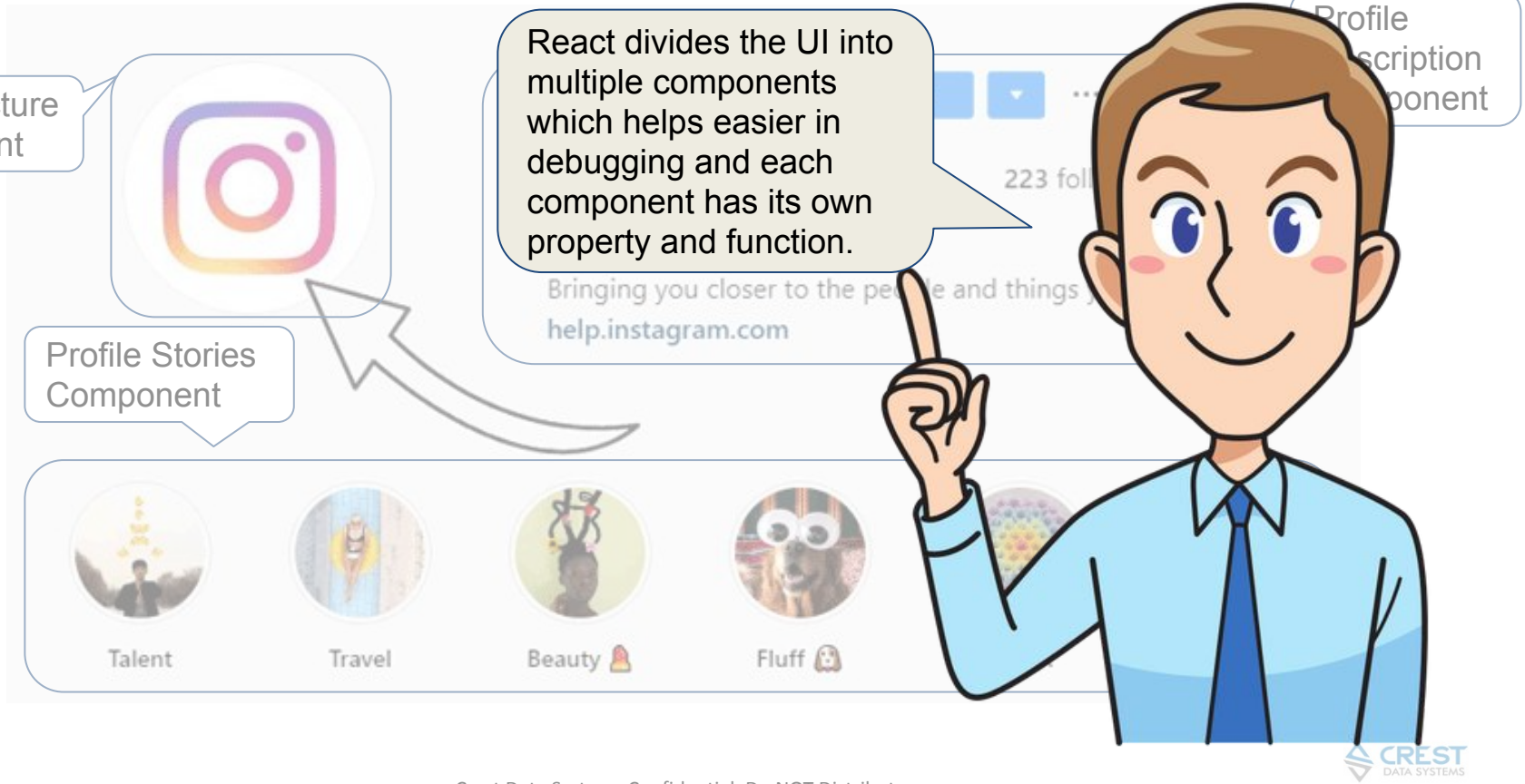
Instagram, AWS are great examples for ReactJs.



Instagram, AWS are great examples for ReactJs.



Instagram, AWS are great examples for ReactJs.



Key Features of ReactJs

- JSX - JavaScript Syntax Extension
- Virtual DOM
- Performance
- One-way data binding
- Extensions
- Debugging

Day 1

Agenda Day - 1

- Overview of Tools:
  - Babel
  - Webpack
  - Npm
  - Npx
- React App Setup:
  - Using create-react-App
  - From Scratch

Overview of Tools

- Babel
  - Babel is a JavaScript compiler, a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.
  - Ref: https://babeljs.io/
- Webpack
  - At its core, webpack is a static module bundler for modern JavaScript applications. When webpack processes your application, it internally builds a dependency graph which maps every module your project needs and generates one or more bundles.
  - Ref: https://webpack.js.org/

Overview of Tools

- Npm & Npx
  - https://nodejs.org/en/download/
  - https://www.geeksforgeeks.org/what-are-the-differences-between-npm-and-npx/

Ref:

- https://reactjs.org/tutorial/tutorial.html#what-is-react
- https://reactjs.org/docs/getting-started.html

React App Setup

- Using create-react-app (CLI) - https://reactjs.org/docs/create-a-new-react-app.html.
- From Scratch - https://www.codementor.io/@rajjeet/step-by-step-create-a-react-project-from-scratch-11s9skvnxv

Hands On Day - 1

- Environment setup for React application.
- Create a React application using npx and create-react-app from scratch.
- Run in dev mode and create builds with different webpack and configurations.
- Run in production mode and test builds with different webpack and configurations.

Day 2

Agenda Day - 2

- Hello World program
- Introducing JSX
- Rendering Elements

- Hello World program
  - https://reactjs.org/docs/hello-world.html

- Introducing JSX
  - JSX stands for JavaScript XML.
  - JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods.
  - After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.
  - This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions:
  - Ref: https://reactjs.org/docs/introducing-jsx.html

- Rendering Elements
  - https://reactjs.org/docs/rendering-elements.html
- Elements are the smallest building blocks of React apps.
- Here in the example `*element*` will be the render by `*ReactDOM.render()*` method.
- We call this HTML, a “root” DOM node because everything inside it will be managed by React DOM.
- React DOM compares the element and its children to the previous one, and only applies the DOM updates necessary to bring the DOM to the desired state.

```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(
    element,
    document.getElementById('root')
  );
}

setInterval(tick, 1000);

```

```
<div id="root"></div>
```

Hands On Day - 2

- Build a login form in App.js using React elements and JSX.

Day 3

Agenda Day - 3

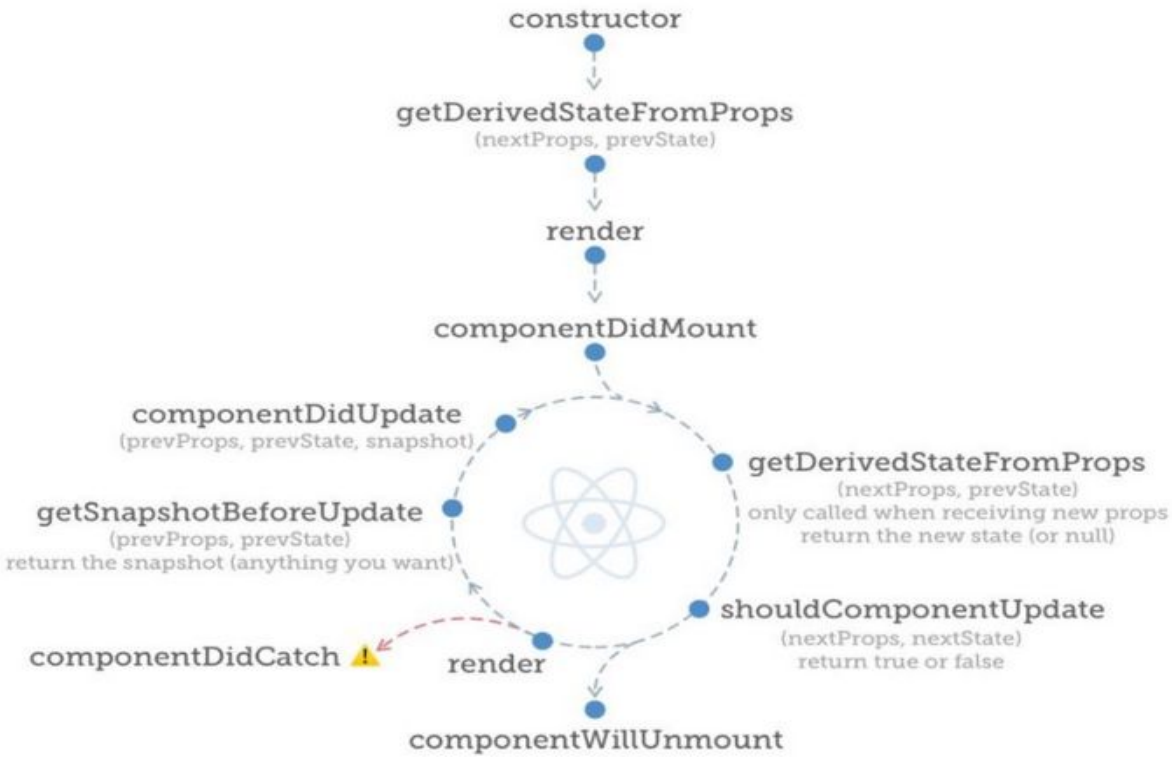
- States’ Life cycle
- Components and Props

- States’ Life cycle methods
  - Mounting
    - These methods are called in the following order when an instance of a component is being created and inserted into the DOM:
    - constructor()
    - static getDerivedStateFromProps()
    - render()
    - componentDidMount()

  - Ref: https://reactjs.org/docs/react-component.html

- States’ Life cycle methods
  - Updating
    - An update can be caused by changes to props or state. These methods are called in the following order when a component is being re-rendered:
    - static getDerivedStateFromProps()
    - shouldComponentUpdate()
    - render()
    - getSnapshotBeforeUpdate()
    - componentDidUpdate()
  - Unmounting
    - This method is called when a component is being removed from the DOM:
    - componentWillUnmount()
  - Ref: https://reactjs.org/docs/react-component.html

- States’ Life cycle methods



- Components and Props
  - Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. This page provides an introduction to the idea of components. You can find a detailed component API reference here.

  - Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.
  - Ref: https://reactjs.org/docs/components-and-props.html

- Components and Props

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState({
      date: new Date()
    });
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

ReactDOM.render(
  <Clock />,
  document.getElementById('root')
);
```

Hands On Day - 3

- Build a login form in App.js using state management.
- Build a calculator application.

Day 4

Agenda Day - 4

- React Hooks
- Handling Events

- React Hooks
  - Basic Hooks
    - useState
    - useEffect
    - useContext
  - Additional Hooks
    - useReducer
    - useCallback
    - useMemo
    - useRef
    - useLayoutEffect

  - Ref: https://reactjs.org/docs/hooks-intro.html

- Handling Events
  - Handling events with React elements is very similar to handling events on DOM elements. There are some syntax differences:
  - React events are named using camelCase, rather than lowercase.
  - With JSX you pass a function as the event handler, rather than a string.
  - Ref: https://reactjs.org/docs/handling-events.html

Hands On Day - 4

- Build a calculator application with hooks.

Day 5

Agenda Day - 5

- Conditional Rendering
- Lists and Keys
- Forms

- Conditional Rendering:
  - In React, you can create distinct components that encapsulate behavior you need. Then, you can render only some of them, depending on the state of your application.

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

ReactDOM.render(
  // Try changing to isLoggedIn={true}:
  <Greeting isLoggedIn={false} />,
  document.getElementById('root')
);
```

Ref: https://reactjs.org/docs/conditional-rendering.html

- Lists
  - Lists in react are the same as Arrays in javascript, Here we loop through the numbers array using the JavaScript map() function. We return a <li> element for each item. Finally, we assign the resulting array of elements to list-Items

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);
```

- Keys
  - Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity:

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li key={number.toString()}>
    {number}
  </li>
);
```

Ref: https://reactjs.org/docs/lists-and-keys.html

- Forms
  - In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input. In React, mutable state is typically kept in the state property of components, and only updated with setState().

Ref: https://reactjs.org/docs/forms.html

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Hands On Day - 5

- Create a To-do list.
- Build a registration page using react forms.

Day 6

Agenda Day - 6

- Lifting State Up
- Composition
- Inheritance
- Fragments

Lifting State Up, Composition vs Inheritance, Fragments

- Lifting State Up
  - Sharing state is accomplished by moving it up to the closest common ancestor of the components that need it. This is called “lifting state up”.

- Ref: https://reactjs.org/docs/lifting-state-up.html

Lifting State Up, Composition vs Inheritance, Fragments

- Composition vs Inheritance
  - React has a powerful composition model, and React community recommend using composition instead of inheritance to reuse code between components.

- Ref: https://reactjs.org/docs/composition-vs-inheritance.html
- https://akash-mihul.medium.com/inheritance-in-react-components-caaf9d20f249

Composition with component in props

```
function SplitPane(props) {
  return (
    <div className="SplitPane">
      <div className="SplitPane-left">
        {props.left}
      </div>
      <div className="SplitPane-right">
        {props.right}
      </div>
    </div>
  );
}

function App() {
  return (
    <SplitPane
      left={
        <Contacts />
      }
      right={
        <Chat />
      } />
  );
}
```

Lifting State Up, Composition vs Inheritance, Fragments

- Composition vs Inheritance
  - React has a powerful composition model, and React community recommend using composition instead of inheritance to reuse code between components.

- Ref: https://reactjs.org/docs/composition-vs-inheritance.html
- https://akash-mihul.medium.com/inheritance-in-react-components-caaf9d20f249

Composition with value in props

```
function Dialog(props) {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        {props.title}
      </h1>
      <p className="Dialog-message">
        {props.message}
      </p>
    </FancyBorder>
  );
}

function WelcomeDialog() {
  return (
    <Dialog
      title="Welcome"
      message="Thank you for visiting our spacecraft!" />
  );
}
```

Lifting State Up, Composition vs Inheritance, Fragments

- Fragments
  - A common pattern in React is for a component to return multiple elements. Fragments let you group a list of children without adding extra nodes to the DOM.

```
render() {
  return (
    <React.Fragment>
      <ChildA />
      <ChildB />
      <ChildC />
    </React.Fragment>
  );
}
```

- Ref: https://reactjs.org/docs/fragments.html

Hands On Day - 6

- Build a unit conversion application.
  - Square yard to Square feet.
  - KG to Litres/Gallons.
  - Miles to Kilometers.
  - And vice versa.

Day 7

Agenda Day - 7

- Type-checking with PropTypes
- Higher Order Components
- Uncontrolled Components

Type-checking with PropTypes, Higher Order Components, Uncontrolled Components

- Type-Checking with PropTypes
  - ‘prop-types’ library is used to implement type-checking
  - This is used to check type of the props passes to the component.
  - This is used to restrict the other data types into the props.
  - Also it is useful for other developer to understand the developed code.

```
import PropTypes from 'prop-types';

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

Greeting.propTypes = {
  name: PropTypes.string
};
```

- Ref: https://reactjs.org/docs/typechecking-with-proptypes.html

Type-checking with PropTypes, Higher Order Components, Uncontrolled Components

- HOC-Higher order components
  - A higher-order component (HOC) is an advanced technique in React for reusing component logic. HOCs are not part of the React API, per se. They are a pattern that emerges from React's compositional nature.
  - A higher-order component is a function that takes a component and returns a new component.
- Ref:- https://reactjs.org/docs/higher-order-components.html
- https://flexiple.com/react/introduction-to-higher-order-components-in-react-by-example/

```
// Take in a component as argument WrappedComponent
function simpleHOC(WrappedComponent) {
  // And return a new anonymous component
  return class extends React.Component{
    render() {
      return <WrappedComponent {...this.props}/>;
    }
  }
}
```

```
// Create a new component
const NewComponent = simpleHOC(Hello);

// NewComponent can be used exactly like any component
// In this case, NewComponent is functionally the same as Hello
<NewComponent/>
```

Type-checking with PropTypes, Higher Order Components, Uncontrolled Components

- Uncontrolled Components
  - To write an uncontrolled component, instead of writing an event handler for every state update, you can use a ref to get form values from the DOM.

- Ref: https://reactjs.org/docs/uncontrolled-components.html

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.handleSubmit = this.handleSubmit.bind(this);
    this.input = React.createRef();
  }

  handleSubmit(event) {
    alert('A name was submitted: ' +
this.input.current.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" ref={this.input} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Hands On Day - 7

- Create a Notification application that loads JSON from a static file and use props to load the notification JSON data.

Day 8

Agenda Day - 8

- React Router

React Router

- https://reactrouter.com/docs/en/v6/getting-started/overview
- https://reactrouter.com/docs/en/v6/getting-started/concepts
- https://reactrouter.com/docs/en/v6/examples/lazy-loading
- https://reactrouter.com/docs/en/v6/examples/search-params
- https://reactrouter.com/docs/en/v6/examples/ssr

Hands On Day - 8

- Create SPA with navigation menu, on hitting each menu redirect to that page location.
- Reference demo:
- https://codesandbox.io/s/nn8x24vm60?from-embed

Day 9

Agenda Day - 9

- Introduction to Redux
- Redux Store

Introduction to Redux, Redux Store

- https://devdocs.io/redux-basics/

Hands On Day - 9

- Create a login page that redirects to the dashboard when the user is logged in else redirect back to the login page if trying to access another URL.

Day 10

Agenda Day - 10

- Redux Store (Cont.)
- Redux Devtools

Redux Store, Redux Dev Tools

- https://devdocs.io/redux-basics/
- https://codeburst.io/redux-devtools-for-dummies-74566c597d7

Hands On Day - 10

- Create Todo list using Redux.
  - Reference: https://redux.js.org/basics/example

Day 11

Agenda Day - 11

- Advance ReactJS:
- Ref and the DOMS & Forwarding Ref
- React developer tools

Ref and the DOMS & Forwarding Ref and React developer tools

- Ref and the DOMS
  - Refs provide a way to access DOM nodes or React elements created in the render method.
  - There are a few cases where you need to imperatively modify a child outside of the typical data flow of using props. At that time, Ref comes in the picture.

- Ref: https://reactjs.org/docs/refs-and-the-dom.html

Creating a ref

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.myRef = React.createRef();
  }
  render() {
    return <div ref={this.myRef} />;
  }
}
```

Ref and the DOMS & Forwarding Ref and React developer tools

- Forwarding Refs
  - Ref forwarding is a technique for automatically passing a ref through a component to one of its children.
  - React components hide their implementation details, including their rendered output.
  - In the example given here, other components using FancyButton usually will not need to obtain a ref to the inner button DOM element.
- Ref: https://reactjs.org/docs/forwarding-refs.html

Ref Forwarding

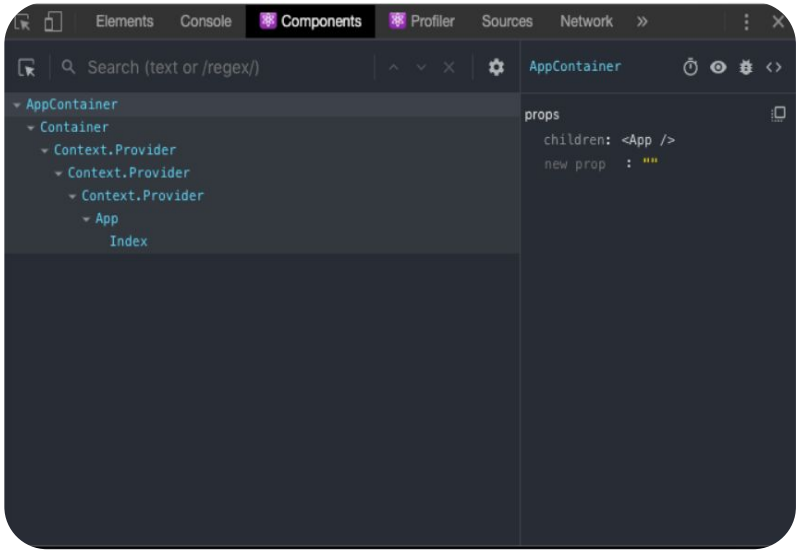
```
const FancyButton = React.forwardRef((props, ref) => (
  <button ref={ref} className="FancyButton">
    {props.children}
  </button>
));

// You can now get a ref directly to the DOM button:
const ref = React.createRef();
<FancyButton ref={ref}>Click me!</FancyButton>;
```

React developer tools

- React developer tools
  - Available for both Chrome and Firefox, the React Developer Tools are an essential instrument you can use to inspect a React application.
  - They provide an inspector that reveals the React components tree that builds your page, and for each component you can go and check the props, the state, hooks, and lots more.

Developer Tool



- Ref: https://flaviocopes.com/react-developer-tools/

Hands On Day - 11

- Implement Ref, Forwarding and test react dev tools.

Day 12

Agenda Day - 12

- Advance ReactJS
- Error Boundaries
- Optimizing Performance

Error Boundaries and Optimizing Performance

- Error Boundaries
  - A JavaScript error in a part of the UI shouldn’t break the whole app. To solve this problem for React users, React 16 introduces a new concept of an “error boundary”.
  - Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed.
- Ref: https://reactjs.org/docs/error-boundaries.html

Error Boundary

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI.
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    // You can also log the error to an error reporting service
    logErrorToMyService(error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

Error Boundaries and Optimizing Performance

- Optimizing Performance
  - Internally, React uses several clever techniques to minimize the number of costly DOM operations required to update the UI.
  - For many applications, using React will lead to a fast user interface without doing much work to specifically optimize for performance. Nevertheless, there are several ways you can speed up your React application.
  - Some of the techniques are:
    - Using the Production Build
    - Using Single-File Builds and Plugins like Webpack, Browserify etc.
    - Avoiding Reconciliation
    - Virtualizing Long Lists, etc.

- Ref: https://reactjs.org/docs/optimizing-performance.html

Hands On Day - 12

- Implement error boundaries and various scenarios to optimize the performance of app.

Day 13

Agenda Day - 13

- Final Exercise & Axios

Final Exercise & Axios

- Axios
  - Many projects on the web need to interface with a REST API at some stage in their development. Axios is a lightweight HTTP client based on the $http service within Angular.js v1.x and is similar to the native JavaScript Fetch API.
  - Axios is promise-based, which gives you the ability to take advantage of JavaScript's async and await for more readable asynchronous code.
  - You can also intercept and cancel requests, and there's built-in client-side protection against cross-site request forgery.

- Ref: https://www.digitalocean.com/community/tutorials/react-axios-react

Final Exercise & Axios

GET Request

```
import React from 'react';

import axios from 'axios';

export default class PersonList extends React.Component {
  state = {
    persons: []
  }

  componentDidMount() {
    axios.get(`https://jsonplaceholder.typicode.com/users`)
      .then(res => {
        const persons = res.data;
        this.setState({ persons });
      })
  }

  render() {
    return (
      <ul>
        { this.state.persons.map(person => <li>{person.name}</li>)}
      </ul>
    )
  }
}
```

POST Request

```
  axios.post(`https://jsonplaceholder.typicode.com/users`, { user })
    .then(res => {
      console.log(res);
      console.log(res.data);
    })
}
```

DELETE Request

```
axios.delete(`https://jsonplaceholder.typicode.com/users/${this.state.id}`)
  .then(res => {
    console.log(res);
    console.log(res.data);
  })
```

Using Async & Await

```
handleSubmit = async event => {
  event.preventDefault();

  //
  const response = await API.delete(`users/${this.state.id}`);

  console.log(response);
  console.log(response.data);
};
```

Hands On Day - 13

- Final Exercise
  - Open API for reference.
  - https://api.nasa.gov/
  - https://jsonplaceholder.typicode.com/
  - Ex. 1 https://jsonplaceholder.typicode.com/todos/1
  - Ex. 2 https://jsonplaceholder.typicode.com/todos

Day 14

Agenda Day - 14

- ESLint - requirement & configuration

Configuring ESLint

ESLint is designed to be completely configurable, meaning you can turn off every rule and run only with basic syntax validation, or mix and match the bundled rules and your custom rules to make ESLint perfect for your project. There are two primary ways to configure ESLint:

- **Configuration Comments** - use JavaScript comments to embed configuration information directly into a file.
- **Configuration Files** - use a JavaScript, JSON or YAML file to specify configuration information for an entire directory and all of its subdirectories. This can be in the form of an `.eslintrc.*` file or an `eslintConfig` field in a `package.json` file, both of which ESLint will look for and read automatically, or you can specify a configuration file on the command line.
- https://eslint.org/docs/user-guide/getting-started
- https://eslint.org/docs/user-guide/configuring/

Hands On Day - 14

- Configure ESLint to the app and fix all warnings

Day 15

Agenda Day - 15

- Final Exercise & TypeScript

Basic foundation for type scripting and how type scripting can be integrated with react?

- Basic foundation for type scripting
  - TypeScript stands in an unusual relationship to JavaScript. TypeScript offers all of JavaScript's features, and an additional layer on top of these: TypeScript's type system.
  - For example, JavaScript provides language primitives like string, number, and object, but it doesn't check that you've consistently assigned these. TypeScript does.
  - This means that your existing working JavaScript code is also TypeScript code. The main benefit of TypeScript is that it can highlight unexpected behavior in your code, lowering the chance of bugs.

- Ref: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

Basic foundation for type scripting and how type scripting can be integrated with react?

- How type scripting can be integrated with react?
  - To start a new Create React App project with TypeScript, you can run any of the following commands:
    - npx create-react-app my-app --template typescript
    - yarn create react-app my-app --template typescript
  - To add TypeScript to an existing Create React App project, first install it using any of the following commands:
    - npm install --save typescript @types/node @types/react @types/react-dom @types/jest
    - yarn add typescript @types/node @types/react @types/react-dom @types/jest
  - Next, rename any file to be a TypeScript file (e.g. src/index.js to src/index.tsx) and restart your development server!
- Ref: https://create-react-app.dev/docs/adding-typescript/

Basic foundation for type scripting and how type scripting can be integrated with react? (Additional Resources)

- https://www.typescriptlang.org/play/index.html?jsx=2&esModuleInterop=true&e=196#example/typescript-with-react
- https://github.com/typescript-cheatsheets/react-typescript-cheatsheet#reacttypescript-cheatsheets

Hands On Day - 15

- Create a React App + Typescript Project.

THANK YOU!

info@crestdatasys.com
https://www.crestdatasys.com/

CREST DATA SYSTEMS