**React Theory**

1. What is React? Is a JavaScript Library for building User Interface(Using components). Using components for split the User Interface is important and useful because we can build the building blocks as contained pieces of coded. If something need to change is much more easy to update the code or if we need to use them in different parts of theinterface. React components can be thought as custom HTML elements.
2. A React component is just a function or class that produces HTML to show the user(using JSX) and handles feedback from the user(using events handlers)
3. An app Component:produces JSX and handles user events and RETURNS-> JSX:Set of Instructions to tell React what content we want to show on the screen
4. ReactDOM.render() is a method that allows to render a component as a component in the real DOM. First element is for calling the function, get back JSX and turn it into HTML. Second element is to take the HTML and put into the DOM inside the specified div element.
5. Why React? -UI State becomes difficult to handle with Vanilla JavaScript; -Focus on Business Logic, not on preventing your App from exploding (plus Framework Creators probably write better Code); -Huge Ecosystem, Active Community, High Performance
6. React Alternatives: Angular, Vue, jQuery(not so much)
7. JSX Elements are used to tell React to create a normal HTML element(div, span, h1, table, hr, input” or to tell React to show another component(Field, Translate, Languages).
8. Difference between React(Knows how to wok with components; Called a ‘reconciler’) and ReactDOM(Knows how to take instructions on what we want to show and turn it into HTML; Called a ‘renderer’)
9. useState: -Function for working with React’s ‘state’ system; -State is used to keep track of data that changes over time; -Used to make React update the HTML on the screen
10. Creating a new React app: npx create-react-app my-app
11. Babel can take the new javascript version and transform new syntax and polyfilling missing features to lower versions of javascript(<ES6), so it can run to the old browsers
12. Starting(npm start from react project) and stopping a project(ctrl + C).
13. JSX: -Special dialect of JS(its not HTML!; - Browsers don’t understand JSX code! We write JSX then run tools to turn it into normal JS; -Very similar in form and function to HTML with a couple differences
14. JSX vs HTML: Adding custom styling to an element uses different syntax; -Adding a class to an element uses different syntax(instead of class use className; for->htmlFor); -JSX can reference JS variables({buttonText});
15. HTML <div style=”background-color:red;”></div> ====> JSX <div style={{backgroundColor: “red”}}></div>
16. React ecosystem: Component Nesting(A component can be shown inside of another); - Component Resuability(We want to make components that can be easily reused through out application); Component Configuration(We whould be able to configure a component when it is created)
17. Creating a Reusable, Configurable Component: 1.Identify the JSX that appears to be duplicated; 2. What is the purpose of that block of JSX? Think of a descriptive name for what it does; 3. Create a new file to house this new component -it should have the same name as the component; 4. Create a new component in the new file, paste the JSX into it; 5. Make the new component configurable by using React’s “props” system
18. Props(Properties): System for passing data from a parent component to a child component -> Goal is to customize or configure a child component. <CommentDetail author={authorName}” /> => {props.author}. Other way to pass the props is to use the props.children when passing an entire component as a prop(<ApprovalCard> <CommentDetail author="Sam"…. ->{props.children}).
19. How React Used to Be: -Functional Components->Can produce JSX to show content to the user; -Class Components: Can produce JSX to show content to the user, Can use the Lifecycle Method system to run code at specific points in time. Can use the ‘state’ system to update content on the screen
20. How React is NOW: The Hooks system allwas function components to have the same functionality as the Class components(Can use Hooks to run code at specific points in time; Can use Hooks to access state system and update content on screen)
21. Functional Components:good for simple content; -Class components: good for just about everything else
22. Benefits of Class components: -Easier code organization; -Can use’state’ (another React system)->Easier to handle user input; -Understands lifecycle events->Easier to do things when the app first starts
23. Rules of Class Components: -Must be a JavaScript Class ; -Must extend(subclass) React.Component; -Must define a ‘render’ method that returns some amount of JSX
24. Rules of State: -Only usable with class components(Tehchincally can be used with functional components using the “hooks” system); -You will confuse props with state; -“State” is a JS object that contains data relevant to a component; -Updating “state” on a component causes the component to (almost) instantly rerender; -State must be initialized when a component is created; -State can only be updated using the function”setState”
25. App Lifecycle Walkthrough: 1.JS file leaded by browser. 2.Instance of App component is created. 3.App components “constructor” function gets called. 4.State object is created and assigned to the “this.state” property. 5.We call geolocation service. 6.React calls the components render method. 7.App return JSX, gets rendered to page as HTML…8. We get result of geolocation!. 9.We update our state object with a call to “this.setState”. 10.React sees that we updated the state of a component. 11.React calls our ‘render” method as second time. 12.React method returns some(opdated) JSX. 13.React takes that JSX and updates content on the screen.