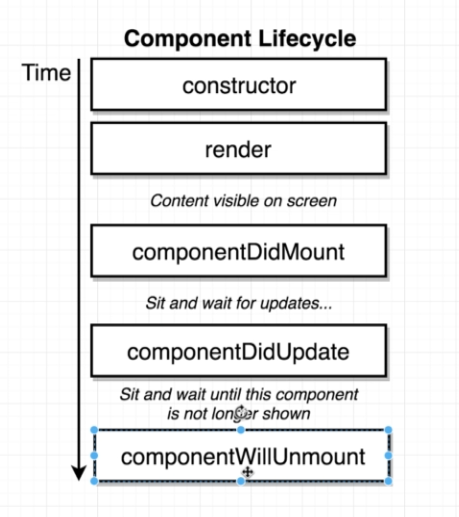
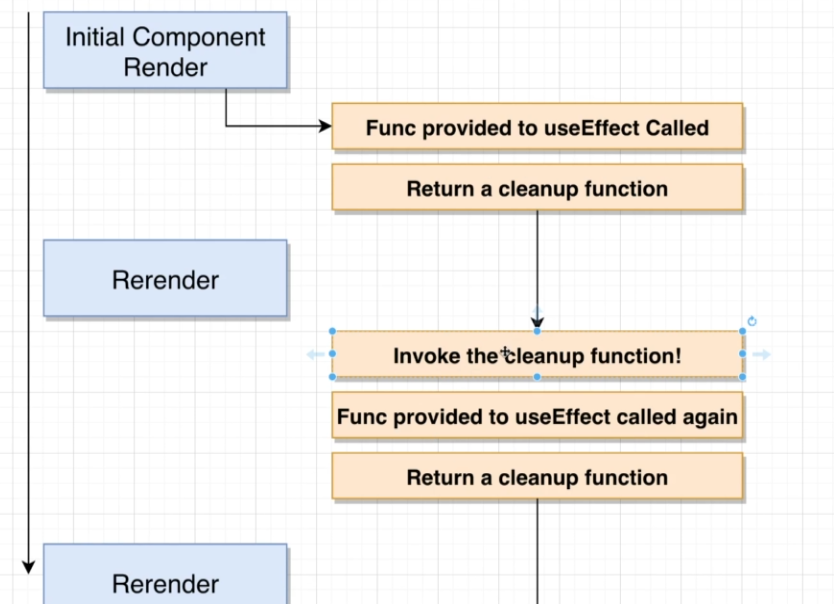
**React Theory**

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.
26. Component Lifecycle method is a method that we can define inside in a class based components. Componenet Lifecycle:1.Constructor-> Good place to do one-time setup; 2.Render(Content visible on screen)->Avoid doing anything besides returning JSX; 3.componentDidMount(Sit and wit for updates…)->Good place to do data-loading; 4.render->componentDidUpdate(Sit and wait until this component is not longer shown)->Good place to do more data-loading when state/props change; 5.componentWillUnmount -> Good place to do cleanup(especially for non-React stuff)…other licycle methods(rarely used):shouldComponentUpdate, getDerivedStateFromProps, getSnapshotBeforeUpdate.



1. We can create default properties when initializing a component with Component.defaultProps={message:”Loading…”}
2. Special functions to pass to specific html elements: onClick= User click on something; onChange(user changes text in an input); onSubmit(User submits a form). This function accepts a reference to a function(a callback function)
3. Control components vs Uncontroll components: Ex. Saving the input value into a state property and overwriting the value back to the input. After that we can control what is shown for the user(like only capitalized text). Uncontrol components means that if we want a value we need to reach the DOM and extract the value from there
4. To access the props from a class component we need to refer it with this.props…. We can pass a prop from child to parent when passing the prop from parent to child, and then calling the method from the child as this.props.onSubmitParent(this.state.term)
5. Using axios instead of fetch for making request to an API
6. For list of elements we should add a key to the root returned element so that when the react render our content the performance will increase
7. React Refs??: gives access to a single DOM element; -We create refs in the constructor, assign them to instance variables, then pass to a particular JSX element as props(<img ref={this.imageRef}
8. **The Hook System-**give function components a lot of functionality: useState->Function that lets you use state in a functional component; -useEffect->Function that lets you use something like lifecycle methods in a functional component; -useRef-> Function that lets you create a “ref” in a function component
9. Hooks are a way to write reusable code, instead of more classic techniques like Inheritance
10. Functions included by react to give more function to the functional components: useState, useEffect, useContext, useReducer, useCallback, useMemo, useRef, useImperativeHandle, useLayoutEffect, usedDebugValue;
11. We can build Custom Hooks that use the react hooks to make the code more reusable
12. Using “useState” (destructuring -> const[activeIndex(piece of state), setActiveindex(Function to change this piece of state)] = useState(null(Initial value for this piece of state))
13. For multiple pieces of state they must be declared one by one, not like in a class component
14. When the function to set the state the function component will be rerendered
15. “useEffect” Allows function components to use something like lifecycle methods. We configure the hook to run some code automatically in one of three scenarios:1.When the component is rendered for the first time only. 2.When the component is rendered for the first time and whenever it rerenders. 3.When the component is rendered for the first time and whenever it rerenders and some piece of data has changed.
16. useEffect(function, secondArgument). SecondArgument: 1.[]->Run at initial render; 2. Nothing->Run at initial render and run after every renderer; 3.[data]->Run at initial render and Run after very rerender if data has changed since last render
17. using async functions to request data in the useEffect: need to use the inner function as async await, using promises or using async/await but with IFFI functions. Can’t use await function for the useEffect function directly
18. XSS attacks in react when using dangerouslySetInnerHTML={{ \_\_html: result.snippet }}(transforming the string into inner html). Use only when the request is done from a trusted API
19. The useEffect can return a function that will be called before the useEffect is called second time
20. When referencing to a piece of state or a prop than you should reference it in the dependency array of the useEffect
21. If using addEventListener for an element that event is called first and after that the wired React element is called(JSX)…so the bubbling up is not longer working as expected
22. useRef allows us to get a direct reference to a DOM element.(get the first element of the component wanted)->const ref=useRef();ref={ref};console.log(ref.current)
23. React routing: User clicks on “list” -> Change the URL, but don’t do a full page refresh! -> Each Route could detect the URL has changed -> Route could update piece of state tracking the current pathname -> Each Route rerenders, showing/hiding component appropriately
24. Custom Hooks: -Best way to create reusable code in a React project (besides components!); -Created by extracting hook-related code out of a function component; -Custom hooks always make use of at least one primitive hook internally; -Each custom hook should have one purpose; -Kind of an art form! (pretty challenging); -Data-fetching is a great thing to try to make reusable.(not refactoring anything related about JSX)
25. Process for creating reusable Hooks: 1.Identify each line of code related to some single purpose; 2.Identify the inputs to that code; 3.Identify the outputs to that code; 4.Extract all of the code into a separate function, receiving the inputs as arguments, and returning the outputs;( “ If you give me a list of inputs I will give you a list of outputs”=”If you give me a default search term, I will give you a way to search for videos and a list of videos”)
26. What is **Redux**? -State management library; -Makes creating complex application easier; -Not required to create a React app!; -Not explicitly designed to work with React(there are ports for other languages to work with redux)
27. Redux Cycle: Action Creator -> Action -> dispatch -> Reducers -> State.