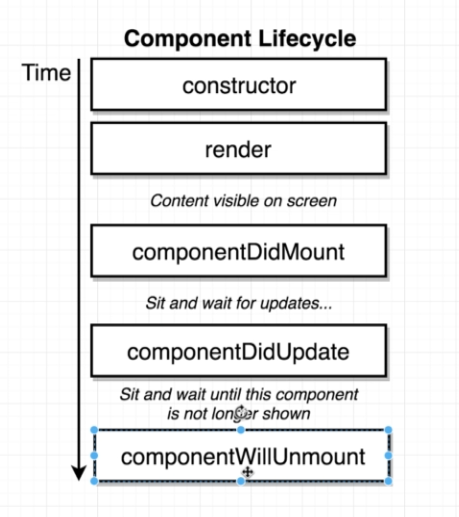
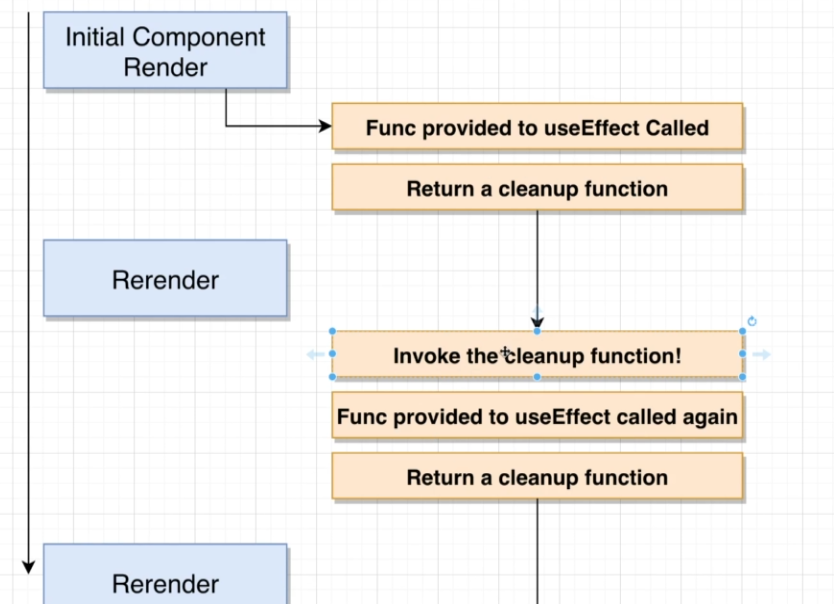
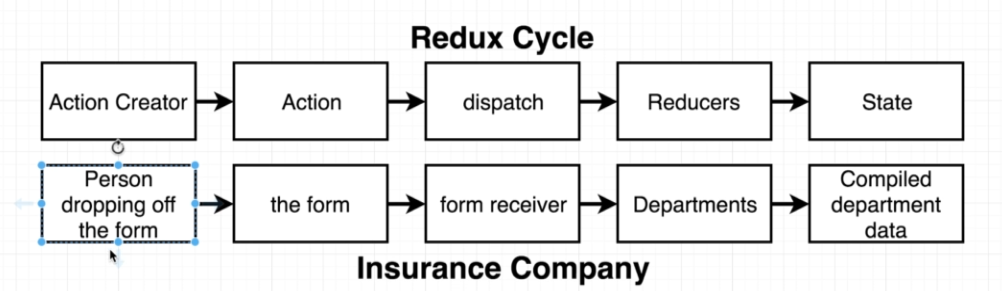
**React Theory**

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.
28. Reducers take 2 parameters: one for oldList of data, and a parameter for changing the oldList(action) and than the reducer returns the new array without modifying the old List
29. To create an action Creator we need to configure the type and the payload(this is optional).
30. The action is the concrete function(object) that is created from an action Creator. All the reducers are added into the global state and dispatch call all of them when calling with the action object.
31. The Redux have 2 properties that are used here: createStore and combineReducers. All the reducers are combined together by combineReducers and after that the store is created by calling the createStore(combinedReducersObject). This way the store will have all the data, and the only way to update the data is working with the store object function dispatch.
32. To change the state we need to use the dispatch function. Other redux functions: const {createStore, combineReducers} =Redux;

//wire up all the reducers

const ourDepartments = combineReducers({…

const store = createStore(ourDepartments);

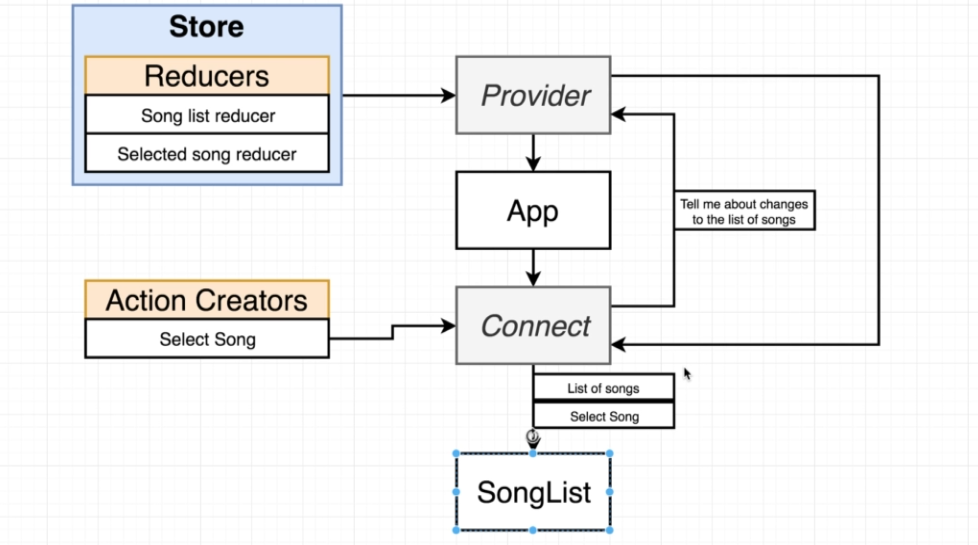
const action = createPolicy("Alex", 20 );

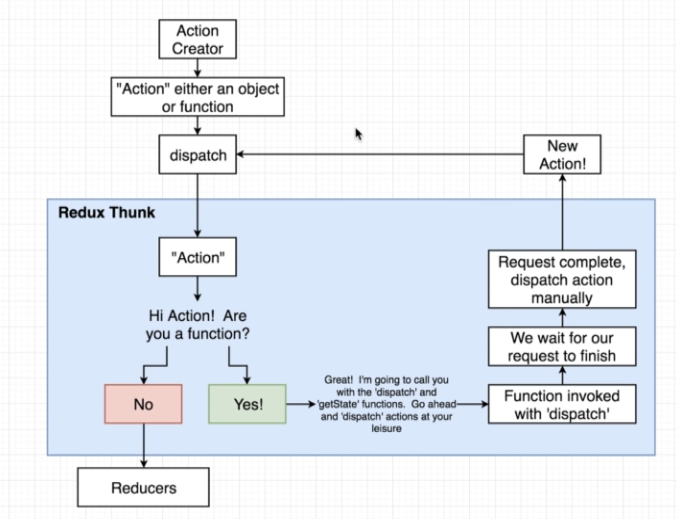
store.dispatch(action);

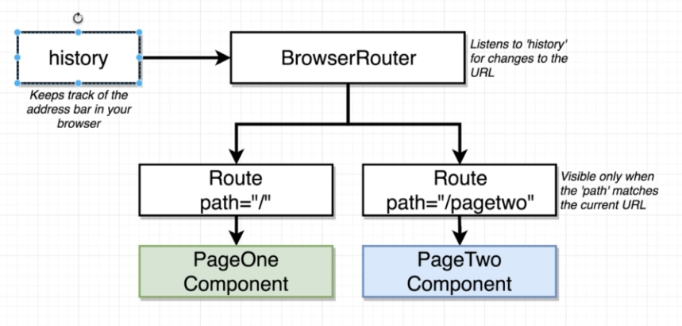
store.dispatch(createPolicy("Jim",30));

console.log(store.getState());

1. npm install --save redux react-redux
2. How React work with Redux? There are 2 components from redux-react library(Provider and Connect. Every time the data saved in reducers is changed, the Provider inform the Connect with the help of context. Every time the action Creator is called to store the data, the Connect component(function, tag) pass down the props so that the component that implements the Connect can use them. The state is defined when using the combineReducers(that makes an object with pairs of keys which are the names of the properties of the state and the values which are functions)

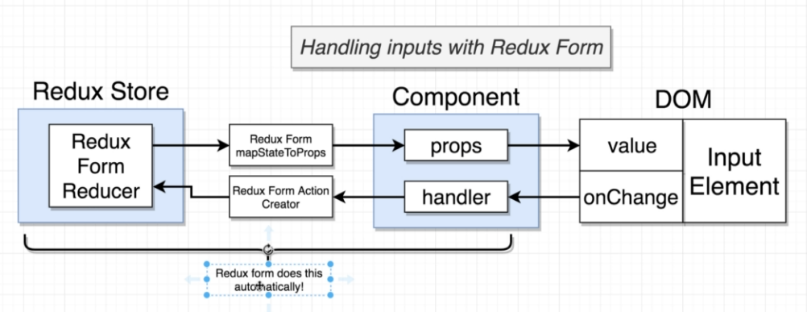


1. Named exports are set by marking the functions with export directly, and they are imported with the curly braces. The default exports are set by exporting default functions and are imported with just a name from a directory.
2. Redux is not magic: need to do a lot of wiring: Redux does not automatically detect action creators being called; Redux dows not automatically detect a function returning an object that is an “action”. The action must be passed into the “connect” component as a function inside the object, and the connect takes the functions inside the object and wrap it up into a bigger function and call the dispatch for use in order to change the state.
3. mapStateToProps function used for connect has two arguments: the global state and the ownProps from the current component;
4. npm install redux react-redux axios redux-thunk: redux(the redux library); react-redux(integration layer between react and redux); axios(helps us make network requests); redux-thunk(Middleware to help us make requests in a redux application: it contains functions that change the store capabilities, it is a good solution for making network requests)
5. General data loading with Redux: -Component gets rendered onto the screen; -Components’ “componentDidMount” lifecycle methods gets called; - We call action creator from “componentDidMount”(Components are generally responsible for fetching data they need by calling an action creator); -Action creator runs code to make an API request; -API responds with data; -Action creator returns an “action” with the fetched data on the “payload” property(Action creators are responsible for making API requests->This is where Redux-Thunk comes into play); -Some reducer sees the action, returns the data off the “payload”; -Because we generated some new state object, redux/react-redux cause our React app to be rerendered;(We get fetched data into a component by generating new state in our redux store, then getting that into our component through mapStateToProps)
6. What’s Wrong with fetchPosts ?Because using async/await function=> Actions creators must return plain JS objects with a type property – we are not!(because of transpilling); By the time our action gets to a reducer, we won’t have fetched out data!
7. Synchrounous action creator: Instantly returns an action with data ready to go; Asynchronous action creator: Takes some amount of time for it to get its data ready to go.(need to use a middleware for asynchronous)
8. Middleware in Redux: -Function that gets called with every action we dispatch. -Has the ability to STOP, MODIFY, or otherwise mess around with actions; -Tons of open source middleware exist; - Most popular use of middleware is for dealing with async actions; -We are going to use a middleware called “Redux-Thunk” to solve our async issues
9. Normal Rules:Action Creators must return action object; 0Actions must have a type property; -Actions can optionally have a “payload”; =🡺 Rules with Redux Thunk: Action Creators can return action objects **OR** Action Creators can return functions!; - If an action object gets returned, it must have a type; -If an action object gets returned, it can optionally have a “payload”
10. Rules of reducers: -Must return any value besides “undefined”; -Produces “state”, or data to be used inside of your app using only previous state and the action(reducers are pure); -Must not return reach “out of itself” to decide what value to return; -MISLEADING>>Must not mutate its input “state” argument => Just kidding! You can mutate it all day and not see any errors! Its easier to tell beginners “don’t mutate state ever” tan to tell them when they can and can’t, but there is a corner case when some problem might appear. If we modify the object but we return the same object than the redux says that that is the same object and no modifications were made, so the old state is returned.
11. For working with arrays and objects in order to perform mutations on them there is a helpful library : Lodash.
12. **React-Routing**: npm install --save react-router-dom. Types of react-router: -react-router(Core navigation lib – we don’t install this manually); -react-router-dom(Navigation for dom-based apps(we want this); -react-router-native(Navigation for react-native apps); -react-router-redux(bindings between Redux and React Router-but it is not necessary or recommended)
13. React routing is used for SINGLE APP PAGES and it means that we trick the user to think he uses multiple pages, when in fact the React Router with the help of Link, which stops an anchor element to trigger the default event. This way the Router than can only hide or show the components needed.
14. For react-routing it is used BrowserRouter that have a history objectat that keeps track of the address bar in your browser. The BrowserRouter listens to “history” for changes to the URL and when there is a change it send the path to the Route and it will be visible only when the “path” matches the current URL. Inside a Route will be wired the component to show. On the same URL we can nest more components. Use “exact” keyword so that the paths will match exactly. Also need to use Link component.



1. Bad Navigation(classic one): -You add a <a/> tag to your application with href=”/pagetwo” and click it; -Your browser makes a request to localhost:3000/pagetwo; -Development server responds with index.html file; -Browser receives index.html file, dumps old HTML file it was showing (including all of your React/Redux state data!!!!!!!!!); - index.html file lists our JS files in script tags – browser downloads and executes these scripts; -Our app starts up.
2. What we want: User wants to navigate to another page in our app; -User click a “Link” tag; -React Router Prevents the browser from navigating to the new page and fetching new index.html file!!!!!!!! ; -URL still changes; -“History” object still sees updated URL, takes URL and sends it to BrowserRouter; -BrowserRouter communicates the URL to Route copmonents; - Route components rerender to show new set of components.
3. Types of Router components: -BrowserRouter(Uses everything after the TLD(.com, .net…) or port as the “path”; -HashRouter(Uses everything after a # as the “path”); -MemoryRouter (Doesn’t use the URL to track navigation)
4. OAuth Authentication: -User authenticates with outside service provider (Google, Linkedin, Facebook); -User authorizes our app to access their information; -Outside provider tells us about the user; -We are trusting the outside provider to correctly handle identification of a user; -OAuth can be used for (1) user identification in our app an (2) our app making action on behalf of user. For permissions there is a list of scopes what you can use.
5. OAuth for Servers: -Results in a “token” that a server can use to make requests on behalf of a user; -Usually used when we have an app that needs to access user data when they are not logged in; -Difficult to setup because we need to store a lot of info about the user;
6. OAuth for JS Browser Apps: -Results in a ‘token” that a browser app can use to make requests on behalf of the user; -Usually used when we have an app that only needs to access user data while they are logged in; -Very easy to set up thanks to Google’s JS lib to automate flow.
7. Steps for OAuth authentication: 1.User click “login with Google” button; 2.We use google’s JS lib to initiate OAuth process; 3. Google’s JS lib makes auth request to Google; -Google displays confirmation screen to user in popup window; -User accepts; -Popup window closes; -Google’s JS lib invokes a callback in our React/Redux app; -Callback provided with “authorization” token and profile info for user.
8. Steps for Setting up Oath: 1.Create a new project at console.developers.google.com; 2. Set up an OAuth confirmation screen; 3.Generate an OAuth Client ID; 4.Install Google`s API library, initialize it with the OAuth Client ID; 5.Make sure the lib gets called any time the user clicks on the “Login with Google” button
9. Localhost:3000?debug\_session=<some\_string> => Saves all data in Redux Store between refreshes of the page. The debugging can be saved in strings and jumps from one to another.
10. Redux form: there is already an reducer that we need to wire it up and we can use it to

store the data.



1. With React-Router, each component needs to be designed to work in isolation ( each component need to fetch its own data!!!!!!!!!!!)
2. React Portal: when we don’t want a component to be render as a direct component we need to use portals to place the component to another root element. ReactDOM.createPortal(JSX, document.querySelector(element))
3. React.Fragment used for inserting sibiling elements into one place. It is like an invisible div and it is invisible to the DOM.(== “<> </>”)
4. Context System. Gets data from a parent component to any nested child component. Just for communication. How it works? -Application loads up in the browser; -We create a context object with a default value of “English”; -App component gets rendered, creates a Provider that wraps UserCreate; -Provider updates the value of the context object to “this.state.language”; -Button and Field reach into context object, see the value from “this.state.language”; -Button and Field render appropriate text to the screen.
5. When setting a provider each provider creates a new pipe for the components that it wraps.
6. For consuming a single value we can use this.context, but if there are multiple context values we need to use ContextPiece.Consumer and return a children that is a function
7. Redux:-distributes data to various components; -Centralizes data in a store; -Provides mechanism for changing data in the store; ==== Context : Distributes data to various components.
8. If we want to use Context in place of Redux: -We need to be able to get data to any component in our hierarchy; -We need to be able to separate our view logic from business logic; -We need to be able to split up business logic (not have a single file with 1000 lines of code)
9. Redux(excellent documentation, well-known design patterns, tremendous amount of open source libs) VS. Context(no need for an extra lib, Hard to build a “store” component with cross cutting concerns).