**REACT**

General:

* Focuses solely on providing rendering and event handling functionality
* Influenced by functional programming
* One-way data flow
* Virtual DOM
* Vanilla JS for templating
* import ReactDOM from ‘react-dom’; is needed to be able to access the render() function
* import React from ‘react’; is required to use JSX

Components:

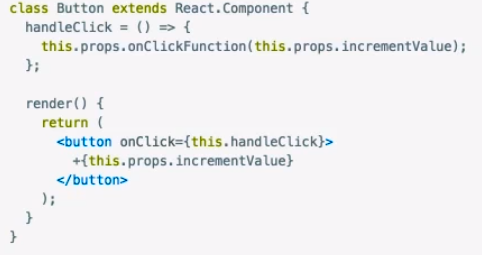
* A React app is a set of reusable components
  + Components take input and produce a description of the user interface in the form of a React element
  + The ReactDOM library allows you to render those React elements in the browser efficiently
* The input for a component is a set of properties you can access inside the component with the props object
* Components can also have a set of state elements that can be accessed with the state object
  + Only in class-based components
  + Can be changed in that component by using React’s setState method
* Every time a component changes its state, React re-renders it
* The props of a component cannot be changed by the component, but the whole component can be re-rendered with different props by the component’s parent
* The syntax to mount a React component in a browser is ReactDOM.render(…)
  + Takes two arguments
    - The component to render
    - An HTML element to hold the React-rendered markup
* The fundamental unit of an application
* Each component corresponds to an element in the DOM
  + The component is responsible for rendering the content of that element and for handling any events that occur within it
* Components can be nested inside other components
* Component lifecycle:
  + Mounting: constructor > componentWillMount > render > componentDidMount
  + Updating: componentWillReceiveProps > shouldComponentUpdate > componentWillUpdate > render > componentDidUpdate
* React components can validate the props they are passed by defining a prop-types object
  + prop-types is defined in a separate npm package: install with npm: install prop-types, then import PropTypes from ‘prop-types’;
  + prop-types specifies the allowed values for each prop key
  + This is runtime validation
  + Other validators can check the data type of a prop, whether a prop is supplied, whether a prop is an instance of a particular class, custom validation function, etc.

JSX and the Virtual DOM:

* HTML is described in JavaScript
  + This produces Virtual DOM notes
  + Special React syntax (JSX) is intended to be closer to HTML
* Once the Virtual DOM is described in JSX, you can pre-transform it before shipping it to the browser
* Babel is commonly used to compile JSX into JavaScript
  + The TypeScript compiler can do this without Babel
* JSX is easier to read, but a script that includes JSX is not valid JavaScript which means it can’t be
  + Minified
  + Processed through a linter
  + Debugged in the browser
  + Formatted with a JavaScript syntax highlighter
* JSX spread attributes: const props = { a: 4, b: 2 }; const element = <Sum {…props} />;

Function and Class Components:

* Function components can be defined with simple functions that receive a props object and return a React element
* Class components can be defined by extending the ReactComponent class and defining a render function inside of that
  + The render function returns a React element
* Function components have a lot more advantages over class components, so you should always use them when you can
* Only use a class-based component when you need to manage a state object (usually done on a top-level component) or when you need personalized event handlers
  + I.e. if you need to invoke a function handler that uses a prop on the component

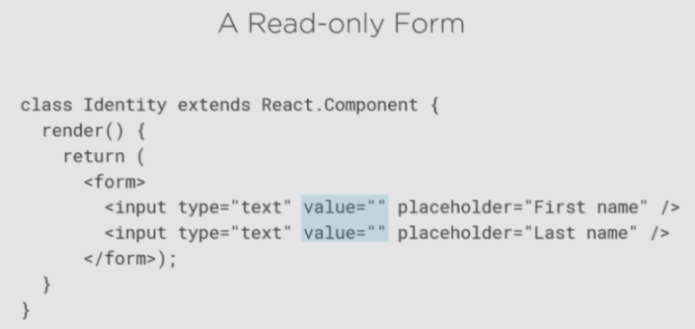


React Events:

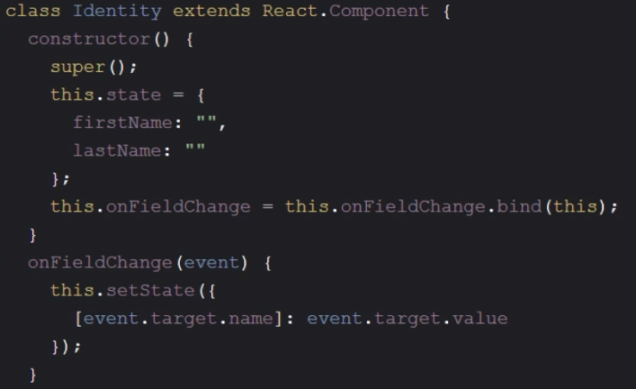
* React also comes with normalized events that work across all browsers in the standard way
  + Ex: onClick event handler
* Every React event function receives an event argument. This event object is a wrapper around the native JavaScript event object
  + All the methods available on the native event object are available here
* preventDefault allows you to handle the event yourself and prevent the browser from doing what it would normally do
  + Most commonly used to prevent forms from submitting so you don’t get an unwanted page reload
  + Syntax for a checkbox that will not check:
    - function Nocheckbox() { return <input type=”checkbox” onClick={ (e) => { e.preventDefault(); }} />; }

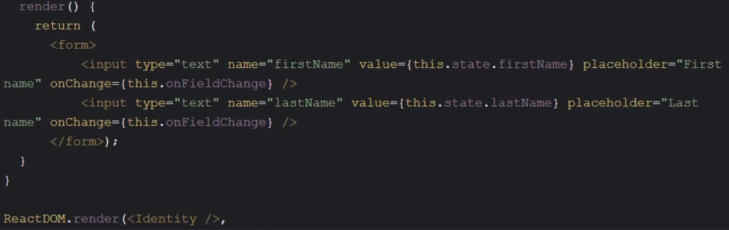
Forms:

* When React renders a form input, it must preserve React’s rendering semantics which guarantees that the rendered UI is a direct translation of the user interface model
  + This has the effect of preventing user input (form elements are read-only)



* To allow a user to provide input into a form, you need to:
  + Add state to the component containing the form
  + Bind the value of the inputs to the component state
  + Add onChange event handlers to catch the user input and update the component state





* Use a form library for form validation (or a mix with some custom implementation)

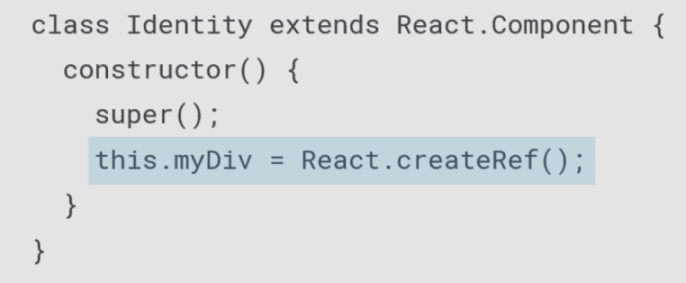
Client-side Routing:

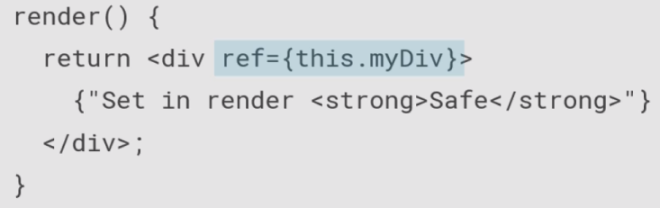
* “React Router” is the most popular client-side router for React
  + Provides conditional rendering based on routes
  + Supports path updates



Refs:

* React has a special property called ref that can be used to get a reference to an element that you would otherwise retrieve with the GetElementById(…) method
  + This property takes in a function that will be executed when the input element is mounted to the DOM
    - That function receives a reference to the element
  + Ex: <input type=”text” ref={(input) => this.userNameInput = input} … />
  + ref breaks React’s declarative rendering paradigm which makes it both useful and a tool of last resort





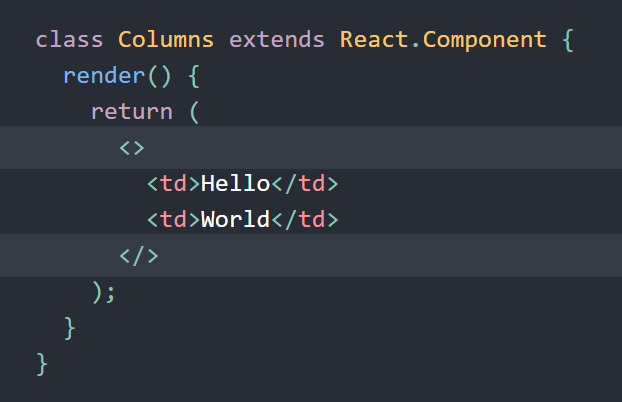
* Controlled elements work with input elements and control their values directly through React itself rather than reading it from the DOM
  + Often labeled as “controlled components”
  + Has more advantages over the simple ref property
  + Avoid using ref – try to use state instead
  + Requires slightly more code than ref but is still relatively simple
  + Ex: state = {username: ‘’} instead of ref={(input) => this.userNameInput = input}
    - Use this state element as the value of the input element
    - Ex: value={this.state.userName}
    - This immediately creates a controlled element
    - To complete this controlled element (at least in this example), you define an onChange event for it to customize its now-controlled behavior:
      * onChange={(event) => this.setState({ userName: event.target.value })}
    - This way, when you type into the text box, React will be aware of this element state change, and it will reflect that change back to the element itself because it’s a regular React state change

Fragments:

* Lets you group a list of children without adding extra nodes to the DOM (lets you return multiple elements from render)

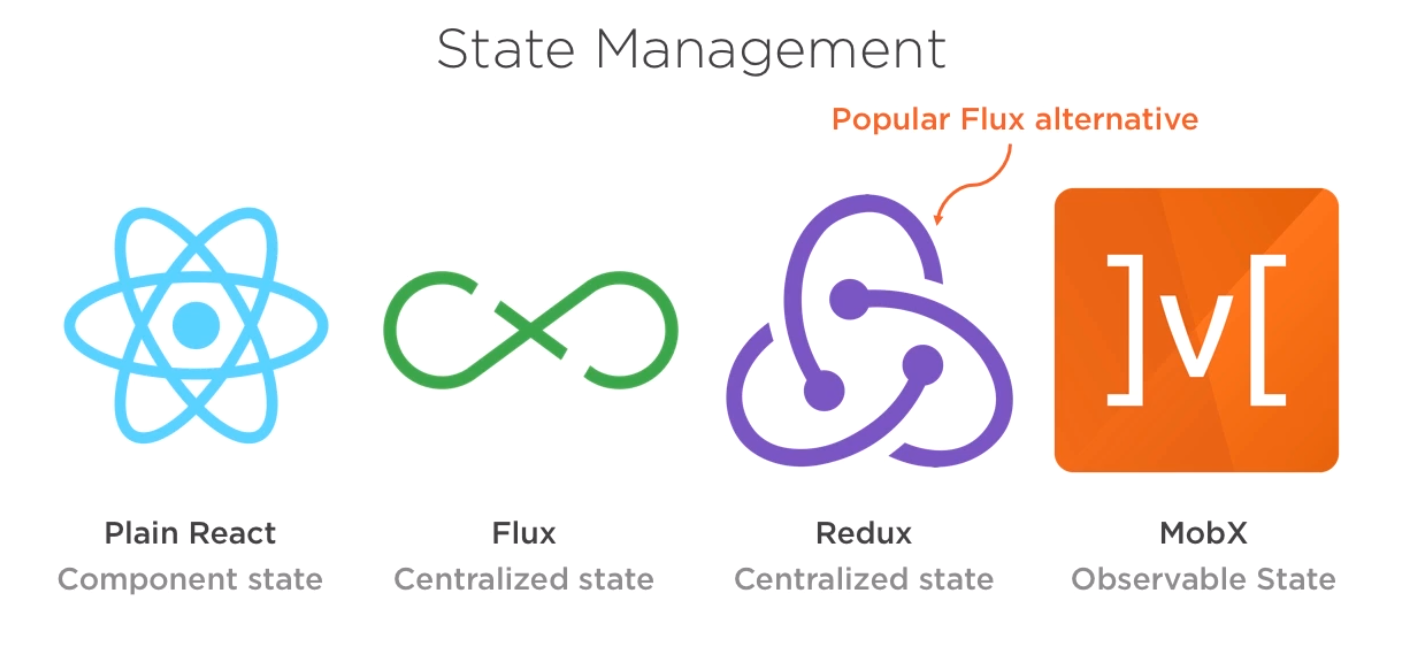


* New short syntax (but not supported all popular tools yet):

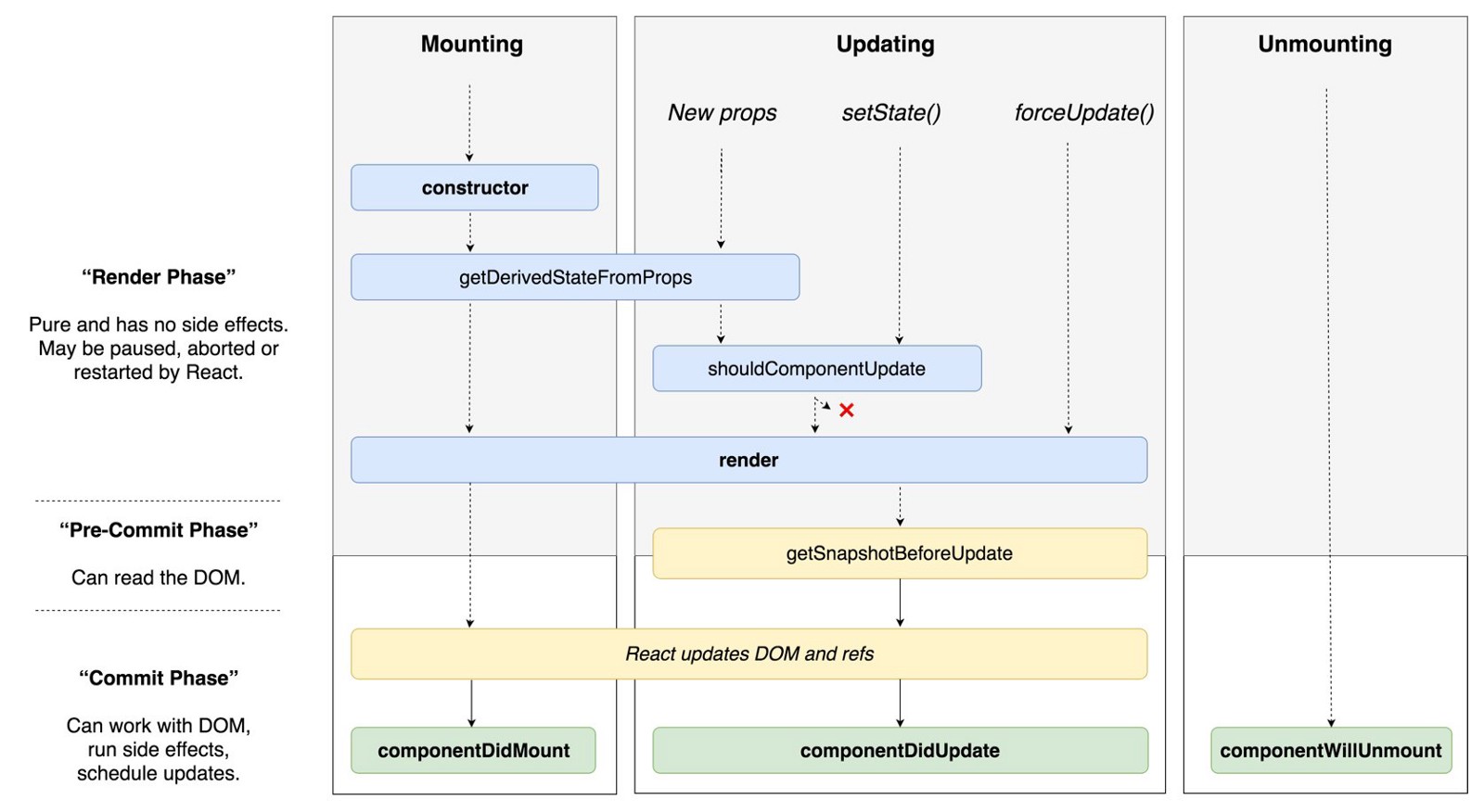


State Container:

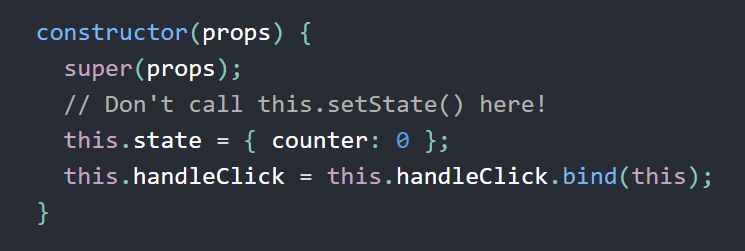
* A state container is a component that takes care of holding the model and controlling updates to that model
* It is not necessary to have a state container
* Redux is a popular, good-quality state container
  + Redux terminology:
    - Reducer – the function that converts the current state and intent to a new updated state (it “reduces” the stream of intents to a single object – the application state at a moment in time)
    - Actions – the events that trigger state changes, aka intents in the context of model-view-intent architecture
* Custom state containers implement three main methods:
  + getState() – returns the current application state object held by the state container
  + dispatch() – applies an intent to the application state producing a new application state
  + subscribe() – registers a callback to be called when the application state changes (when an intent is passed to the dispatch method)



Component Lifecycle:

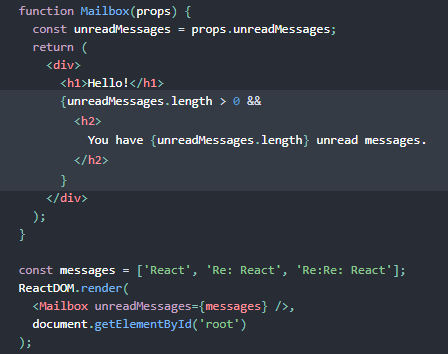


* Error handling - These methods are called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component:
  + static getDerivedStateFromError()
  + componentDidCache()
* AJAX calls should be made in the componentDidMount lifecycle method
  + This is so you can use setState to update your component when the data is retrieved
* If you don’t initialize state and you don’t bind methods, you don’t need to implement a constructor for your React component
  + When implementing the constructor for a React.Component subclass, you should call super(props) before any other statement. Otherwise, this.props will be undefined in the constructor, which can lead to bugs
  + Typically, constructors are only used for two purposes:
    - Initializing local state by assigning an object to this.state
    - Binding event handler methods to an instance
  + Do not call setState() in the constructor
    - If your component needs to use local state, assign the initial state to this.state directly in the constructor
      * The constructor is the only place where you should assign this.state directly. In all other methods, you need to use this.setState() instead

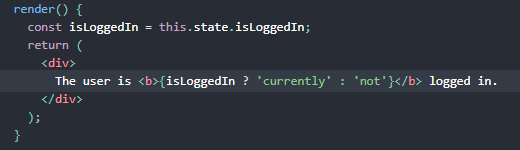


Conditionally Rendering Elements:

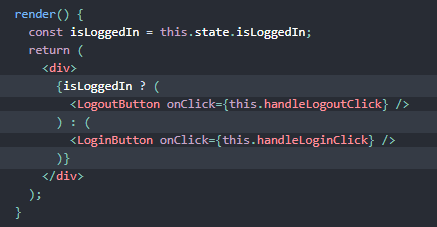
* Inline If with Logical && Operator:



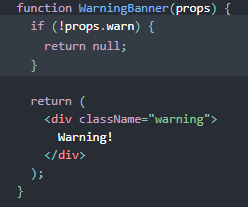
* + This works because in JavaScript, true && expression always evaluates to expression, and false && expression always evaluates to false. Therefore, if the condition is true, the element right after && will appear in the output. If it is false, React will ignore and skip it.
* Inline If-Else with Conditional Operator:
  + Uses JavaScript’s conditional operator – condition ? true : false



or…

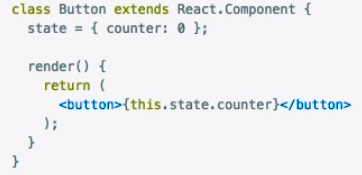
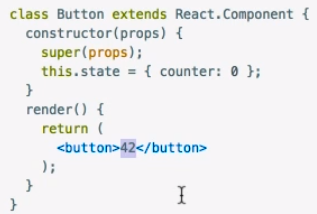


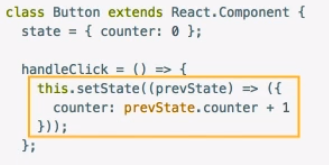
* Preventing Components from Rendering:
  + In rare cases you might want a component to hide itself even though it was rendered by another component. To do this, return null instead of its render output.



Misc./Unorganized:

* Syntax shortcut with the constructor/state object:



* The first decision you need to make in a React application is the component structure
  + How many components to use and what each component should describe
  + If you know that the component will not have any interactivity to the user and if the component is not a top-level component where a state needs to be managed, start that component as a function component
* If your state update depends on the current state, use the pictured syntax instead of this.setState({ counter: this.state.counter + 1 });
* Options for styling a React component
  + Include a CSS file as normal and use the element class to select and style any element
    - Ex: <div className=”info”>
      * In React, you need to use the className property instead of class to match the JavaScript DOM API for elements
      * CSS file for this example: .info { color: red; }
  + Put your CSS in JavaScript
    - Include a style property (a special React property that is passed a JavaScript object instead of a string)
      * Uses the JavaScript API for styles
      * Ex: <div style={{display: ‘inline-block’, marginLeft:
* React escapes all content by default
* React applications consist of React components arranged in a hierarchy. Data is passed down the component hierarchy by passing values into components’ props. Data is passed back up the component hierarchy by passing values as function arguments to functions passed in as props
* Popular React libraries:

