Web Front-End Development
Week 12: Thinking in React

Thinking in React

https://reactjs.org/docs/thinking-in-react.html

React Process:

Start with a mockup of your React app.

1. Break the UI into a component hierarchy
    a. A component should ideally follow the single responsibility principle
        i. each module should have responsibility over a single piece of functionality
    b. UI components often represent one piece of your data model
    c. If it ends up growing, it should be decomposed into smaller subcomponents.
2. Build a static version in React
    a. Build a version that takes your data model and renders the UI but has no interactivity.
    b. Build components that reuse other components and pass data from parent to child components using `props`.
    c. Don't use state at all to build this static version. State is reserved only for interactivity (data that changes over time), that will come later.
    d. At the end of this step, you'll have a set of reusable components that render your data model. The components will only have `render()` methods since this is a static version of your app.
    e. The component at the top of the hierarchy will take your data model as a prop. If you make a change to your underlying data model and call `ReactDOM.render()` again, the UI will be updated.
3. Identify the minimal, but complete, representation of UI state
    a. To make your UI interactive, you need to be able to trigger changes to your underlying data model using state.
    b. Think of the minimal set of mutable state that your app needs.
    c. Figure out the absolute minimal representation of the state your application needs and compute everything else you need on-demand.
    d. The key here is DRY: Don't Repeat Yourself.
        i. Avoid redundancy
        ii. Any changes are synchronized across related elements
    e. To figure out what should be state
        i. Is it passed in from a parent via props? If so, it probably isn't state.
        ii. Does it remain unchanged over time? If so, it probably isn't state.
        iii. Can you compute it based on any other state or props in your component? If so, it isn't state.
    f. Data that changes over time and can't be computed should be represented as state.
4. Identify where state should live
    a. To determine where state should live for each piece of state in your application:
        i. Identify every component that renders something based on that state.
        ii. Find a common owner component (a single component above all the components that need the state in the hierarchy).

iii. Either the common owner or another component higher up in the hierarchy should own the state.
iv. If you can't find a component where it makes sense to own the state, create a new component simply for holding the state and add it somewhere in the hierarchy above the common owner component.

b. Lift State Up
i. Often, several components need to reflect the same changing data. In React, 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".
ii. When you want to aggregate data from multiple children or to have two child components communicate with each other, move the state upwards so that it lives in the parent component. The parent can then pass the state back down to the children via props, so that the child components are always in sync with each other and with the parent.
iii. State lives in one component and only that component can change it. Don't try to sync state between different components, let the data flow top-down.
iv. If something can be derived from either props or state, it probably shouldn't be in the state, just calculate it in render().

c. Pass state down to other components as props.

5. Add Inverse Data Flow
a. To update state in a parent component use events to pass callback functions that call setState() in the component that owns the state.