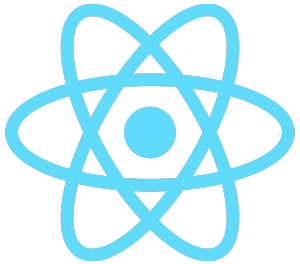


React.js

Lecture 2

Agenda

- Recap last lecture points.
- Integrate with UI libraries
- React Hooks
- Sharing data between components - Props
- Rendering List
- Conditional Rendering
- Questions!

UI Libraries

- Bootstrap and React Bootstrap
  - https://react-bootstrap.github.io/docs/getting-started/introduction
  - https://reactstrap.github.io/?path=/story/home-installation--page
- Material ui
  - https://mui.com/material-ui/getting-started/installation/
- Ant design
  - https://ant.design/
- Semantic UI
  - https://react.semantic-ui.com/usage

React Hooks

What is hooks ?

Hooks are a new addition to React in version 16.8 that allows you use state and other React features, like lifecycle methods, without writing a class.

React Hooks

React Hooks are simple JavaScript functions that we can use to isolate the reusable part from a functional component. Hooks can be stateful and can manage side-effects.

The Rules of React Hooks

- Only Call Hooks at the Top Level
- Only Call Hooks from React Functions

React Hooks

A hook is a special function that lets you "hook into" various React features. Imagine a function that returns an array with two values that's what **useState** has:

- **The first value:** a variable with the state.
- The second value: a variable with an handler (a function to change the current state).

React Hooks - useState

useState :

To manage states. Returns a stateful value and an updater function to update it,

State lets a component “remember” information like user input. For example, a form component can use state to store the input value.

We use the array destructure, as we know its first and second position values, and we know they correspond to the state value and to a handler to change it.

Usage :

const [state, setState] = useState(initialStateValue)

https://www.freecodecamp.org/news/learn-react-usestate-hook-in-10-minutes/

React Hooks - useEffect

- Declare an Effect. By default, your Effect will run after every render.

- Specify the Effect dependencies. Most Effects should only re-run when needed rather than after every render. For example, a fade-in animation should only trigger when a component appears. Connecting and disconnecting to a chat room should only happen when the component appears and disappears, or when the chat room changes. You will learn how to control this by specifying dependencies.

- Add cleanup if needed. Some Effects need to specify how to stop, undo, or clean up whatever they were doing. For example, “connect” needs “disconnect”, “subscribe” needs “unsubscribe”, and “fetch” needs either “cancel” or “ignore”. You will learn how to do this by returning a cleanup function.

https://www.freecodecamp.org/news/react-useeffect-absolute-beginners/

React Hooks - useEffect

Every React component goes through the same lifecycle:

- A component mounts when it's added to the screen.
- A component updates when it receives new props or state, usually in response to an interaction.
- A component unmounts when it's removed from the screen.

React Hooks - useEffect

useEffect :

accepts a function which can perform any side effects.

Usage :

```
useEffect(() => {
    return () => {
     //clean your code on unmount component
    }
},[arrayDependencies]);
```

[] : empty array means to fire on mount only

[dependencies] : means to fire on mount and with every change in this dependency value

React Hooks - useRef [self- study]

useRef :

It returns a ref object with a .current property. The ref object is mutable, Refs let a component hold some information that isn't used for rendering, like a DOM node or a timeout ID. Unlike with state, updating a ref does not re-render your component.

Usage :

Const inputRef = useRef();

<input ref={inputRef} ... />

Props

Passing Props to a Component

React components use props to communicate with each other. Every parent component can pass some information to its child components by giving them props. Props might remind you of HTML attributes, but you can pass any JavaScript value through them, including objects, arrays, and functions.

```
<Avatar
    person={{ name: 'Lin Lanying', imageId: '1bX5QH6' }}
    size={100}
/>
```

Read props inside the child component

You can read the props in the child component function

```
function Avatar(props) {
  //props.person and props.size
}
```

Or you can use ES6 destruct option to read the props sent keys

```
function Avatar({person,size = 50}) {}
```

Rendering Lists

Rendering Lists

- You will often want to display multiple similar components from a collection of data. You can use the JavaScript array methods to manipulate an array of data like filter() and map() with React to filter and transform your array of data into an array of components.

- You need to give each array item a key — a string or a number that uniquely identifies it among other items in that array.

- Keys tell React which array item each component corresponds to, so that it can match them up later. This becomes important if your array items can move (e.g. due to sorting), get inserted, or get deleted. A well-chosen key helps React infer what exactly has happened, and make the correct updates to the DOM tree.

https://react.dev/learn/rendering-lists#why-does-react-need-keys

Conditional Rendering

Conditional Rendering

Your components will often need to display different things depending on different conditions. In React, you can conditionally render JSX using JavaScript syntax like if statements, &&, and ? : operators.

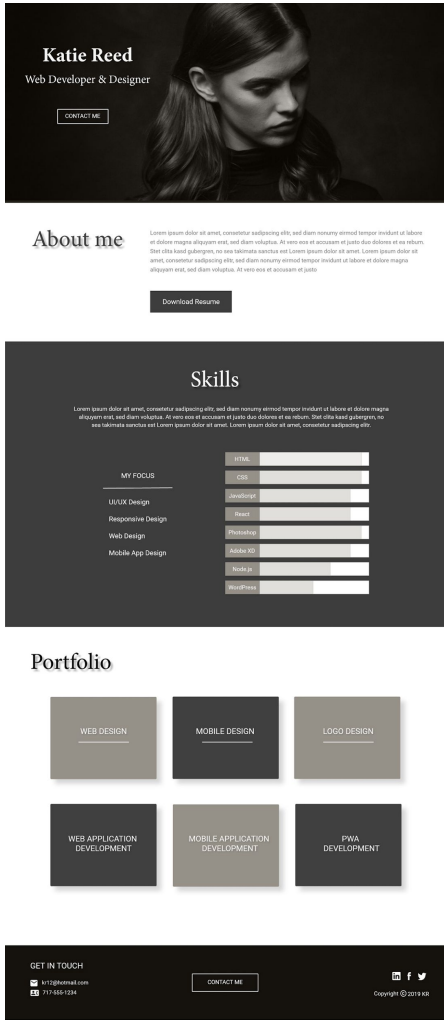
Thank you

Lap

Task: Portfolio

Refactor the previous task with the following:

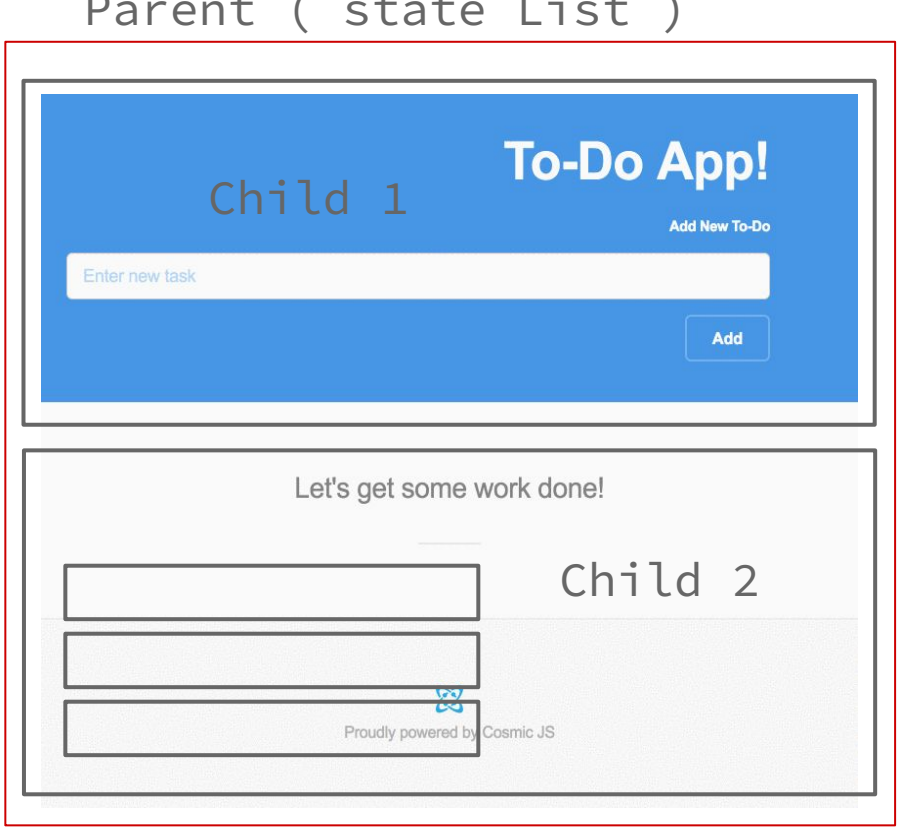
- To read the list of skills and list of projects from an array to iterate over.
- Create a Skill progress bar as reusable component and pass data to the component using props and same with the portfolio cards
- Use conditions to change the background color for the portfolio cards to be
  - If card number is odd to be with light grey color
  - If card number is even to be with dark grey color



Task : To Do App

Second task will be To do form with a required input field with add button for tasks as shown in the image.

After adding task should be added in list with mark as done and delete buttons :

- Mark as done will add strikethrough on text [Bonus]
- Delete will remove task from list

Using reusable components and passing data through components