CSC309H1S

Programming on the Web

Winter 2023

Lecture 10: Introduction to React

Instructor: Kuei (Jack) Sun

Department of Computer Science
University of Toronto

v1.02

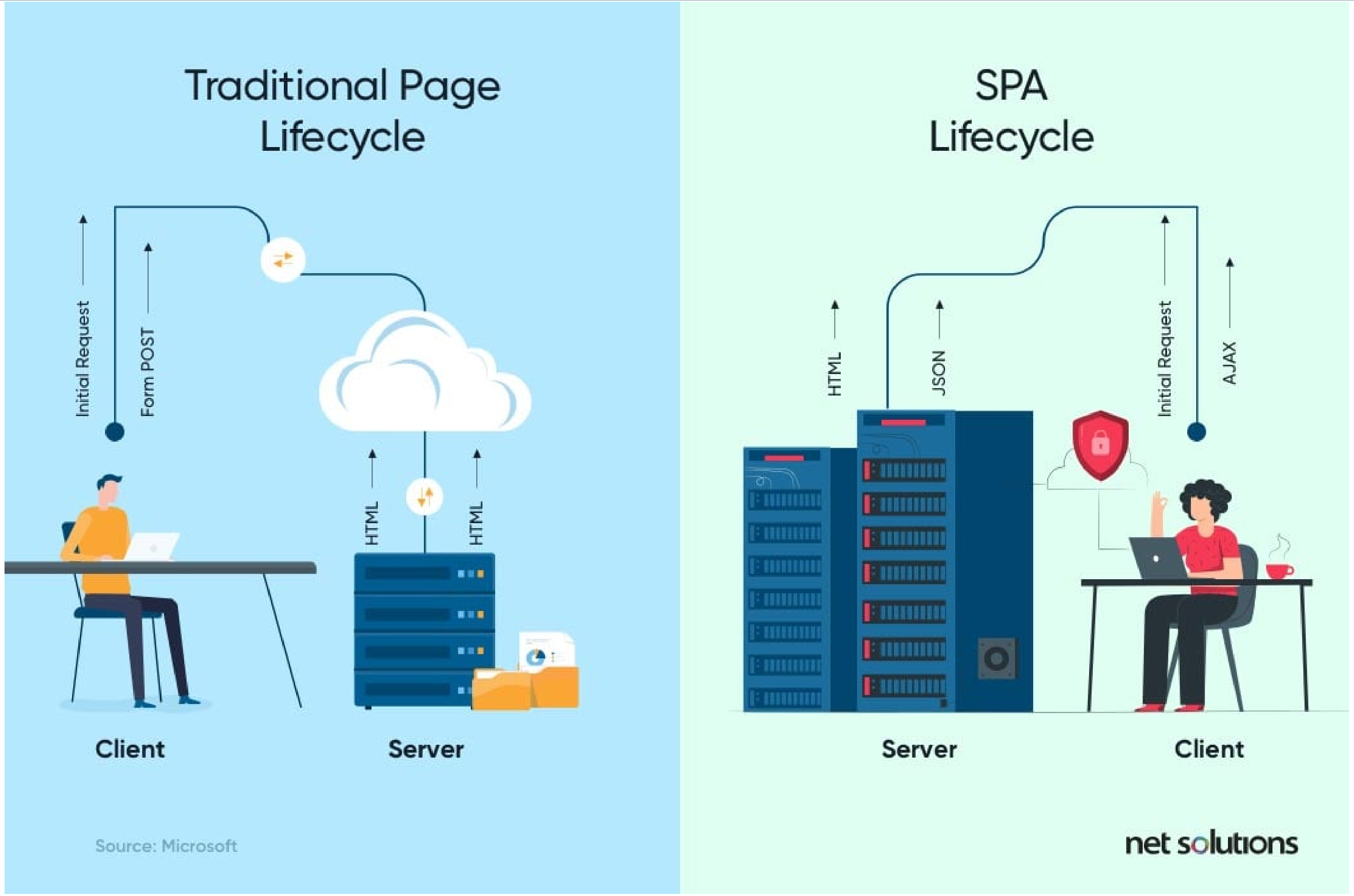
Review

- Server-side rendering
  1. Backend server listens for request
  2. Upon entering a URL, browser sends request to server
  3. Server returns an HTML page in response
     - Many contain links to other static files
       - E.g., js, css, image, etc
  4. Separate requests are sent for static files
  5. Browser renders HTML and CSS, runs scripts
- Each link or form submission yield a new web page
  - Requires a full reload. May degrade user experience (UX)
- Solution?

Single Page Application

- Single page application (SPA)
  - Executed in the browser's built-in JavaScript engine
  - Only requires one hard URL reload
  - Subsequent request/rendering can be done through Ajax in background
- Benefits
  - Seamless user experience
    - Performing an action does not reset the page
  - Efficiency
    - Only relevant parts of page are updated, not entire page
  - Improves load time
    - Initial load (when nothing is there) takes less time

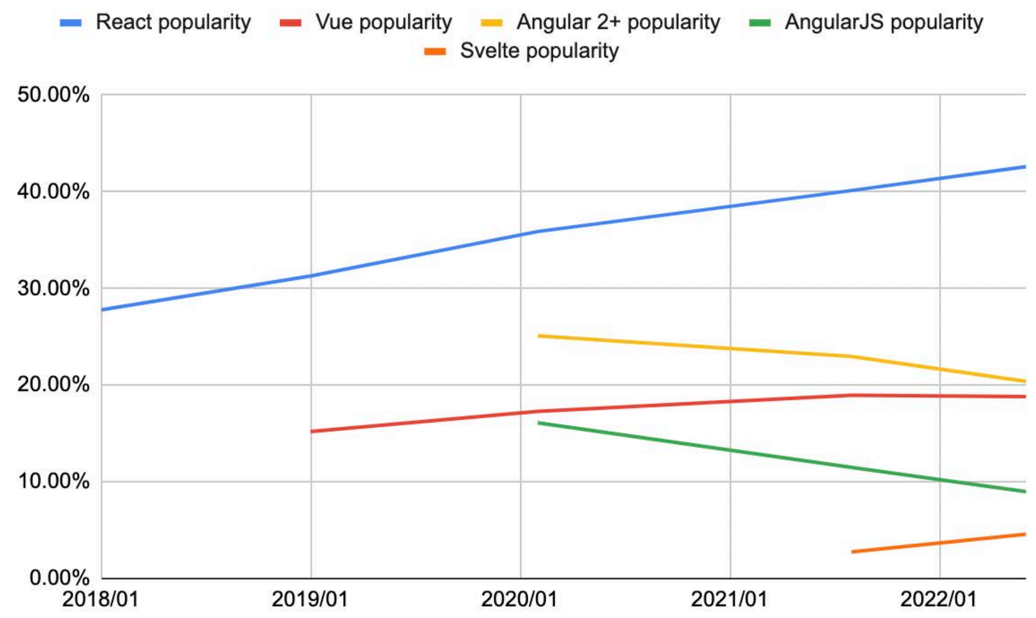
Page Lifecycle

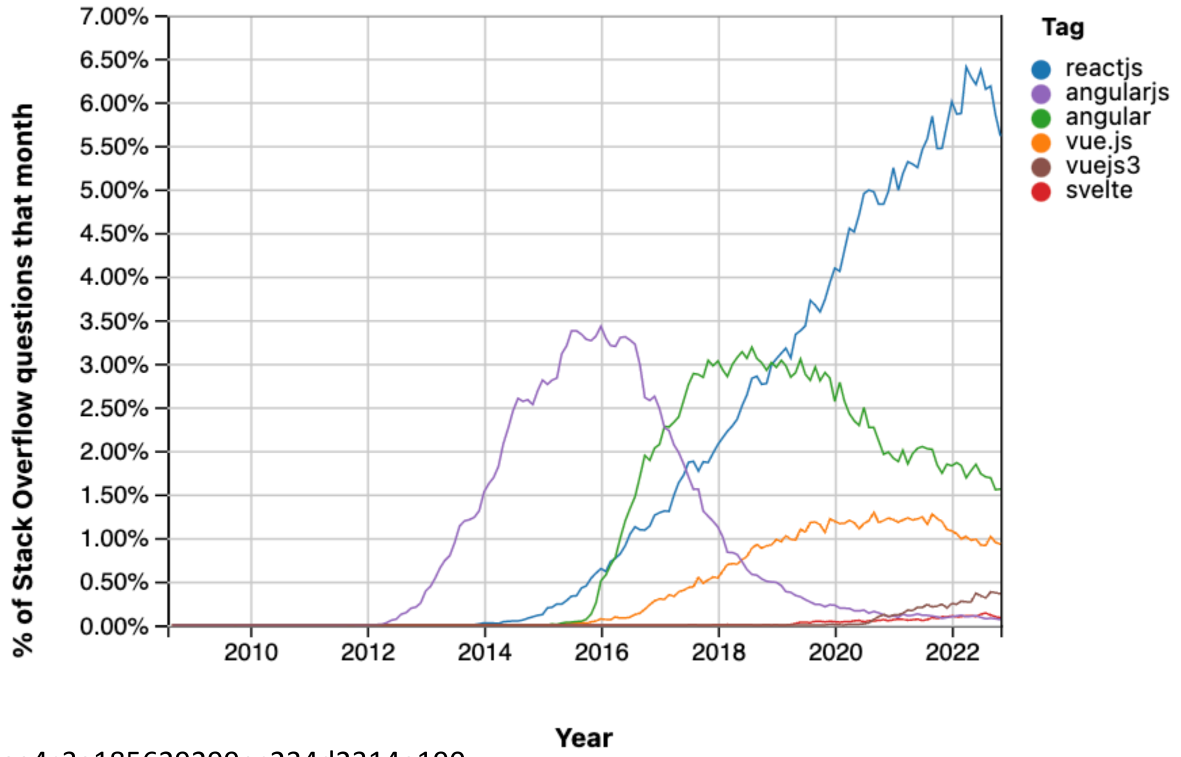


https://www.netsolutions.com/insights/single-page-application/

Building SPA

- Nobody build SPA with Ajax alone
- Frontend frameworks
  - React , Angular , Vue





https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190

React JS

- Released by Facebook in 2013
- A JS library for building interactive user interfaces
- React takes charge of re-rendering when *something* changes
  - You no longer need to manipulate elements manually
- Virtual DOM
  - A representation of UI kept in memory and synced with real DOM
    - Handled by a library named ReactDOM
  - When something changes, it compares new and old DOMs
    - Finds what has been updated
    - Updates only those elements in the browser’s DOM
      - Because updating re-rendering real DOM is expensive!

JSX

- React uses a special variation of JavaScript
- JSX
  - Short for JavaScript XML
  - Merges HTML and JavaScript into one language
  - Example:

```
const element = <h1>Hello world</h1>;
```

- Browser *does not* understand JSX natively
  - Requires translation before execution
- Babel JS
  - A JavaScript compiler. Can translate JSX code into pure JS code

Translation

- JSX

```
const element = <span className="red">
        Hello World!
    </span>;

const name = "Joe";
const id = "div-1";

const element2 = (
    <p>
        <div id={id}>
            Hi, {name}!
        </div>
    </p>
);
```

- JavaScript

```
const element = /*#__PURE__*/
       React.createElement("span", {
              className: "red"
       }, "Hello World!");

const name = "Joe";
const id = "div-1";

const element2 = /*#__PURE__*/
       React.createElement("p", null,
                /*#__PURE__*/
       React.createElement("div", {
              id: id
       }, "Hi, ", name, "!"));
```

These are React elements, not real JS elements

Make it Real

- Import React and Babel (JSX) libraries into your HTML

```
<script src="https://unpkg.com/react@18.2.0/umd/react.production.min.js"></script>
<script src="https://unpkg.com/react-dom@18.2.0/umd/react-dom.production.min.js"></script>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

- Render your element inside an actual JS element

```
<script type="text/babel">
    const element = <h1>Hello World!</h1>;
    const root = ReactDOM.createRoot(document.body);
    root.render(element);
</script>
```

- React roots are used to render React elements into the real DOM

React Components

- React components
  - Functions that return a JSX element, or
  - Classes that extend `React.Component` and implement the render method
- Key concept in React
- Allows you to make your elements reusable
  - Components can be reused like an HTML tag

Rendering React elements inside <main>

```
<main id="root"></main>
<script type="text/babel">
function Hello() { return <h1>Hello World</h1>; }
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<Hello />);
</script>
```

Reusing Hello component

Components Basics

- Void tags must always end with `/>`, e.g., `<Hello />`
- Component name must be capitalized
  - To distinguish from built-in HTML elements (always lowercase)
- A JSX element must be wrapped in one enclosing tag
- React fragment
  - A workaround for returning multiple elements

```
function Items() {
    return <>
        <li>Hello</li>
        <li>World</li>
    </>;
}
```

Components and Props

- You can put any JS expression inside curly braces in JSX

```
const id = "content";
<div id={id}>...</div>
```

- Props
  - Read-only arguments passed into React components via a dictionary

```
function Text(props) {
    return <p>{props.value}</p>;
}
```

  - You can pass arguments like specifying HTML attributes in JSX

```
root.render(<Text value="Hello world" />);
```

Styles and Classes

- Styles and classes uses JavaScript names, not CSS/HTML names
  - Important: styles must be placed inside a dictionary

```
function Text(props) {
    const {value, size} = props;
    return <p className="text" style={{fontSize: size}}>{value}</p>;
}
```

- Tips
  - Use destructuring to simplify components with many props
  - you *do not* need to add quotations marks around attribute values
    - Compiler does that for you automatically

```
root.render(<Text size={30} value="Hello world" />);
```

Loop-Generated Elements

- Elements created in a loop must have a unique key prop.
- key prop
  - Identifies which item has changed, is added, or is removed.
- Otherwise, React will have to re-render the whole list whenever something changes
- Only affects the virtual DOM
  - No visible difference in real DOM

Destructuring in function parameter

```
function List({title, values}) {
    return <>
        <Text value={title} size="2em" />
        <ul>
            {values.map((item, index) => (
                <li key={index}>
                    {item}
                </li>
            ))}
        </ul>
    </>;
}
```

key can just be the loop index

Paired Tag

- Components can be written as paired tags too
- Elements inside the tags are passed as the children props

```
function Box({children}) {
    return <div className="box">{ children }</div>;
}
```

- Example

```
const mybox = (
    <Box>
        <List title="Cats" values={["Felix", "Oscar", "Fluffy", "Whiskers"]} />
    </Box>
);

root.render(mybox);
```

Class Components

- Another way to define a component
  - Extends `React.Component` base class
  - Implements the `render` method
  - Can have *states*
    - In contrast, functional components are “stateless” components
- Props are passed to constructor. Can access through `this.props`
  - The super class constructor handles the above already
- Example

```
class Welcome extends React.Component {
    render() {
        return <h1>Hello, {this.props.name}</h1>;
    }
}
```

Component State

- Class components have a built-in state
  - Default value is `null`
  - Can override constructor to change the initial state
  - State values can be accessed via `this.state` in the render method
- Whenever the state changes, the component re-renders

```
class Counter extends React.Component {
    constructor(props){
        super(props);
        this.state = { counter: 0, };
    }

    render(){
        return <h3>{this.state.counter}</h3>;
    }
}
```

Updating State

- React states should never be mutated directly
  - Except inside the constructor
  - The two approaches below will *not* trigger re-rendering

```
// wrong way 1
this.state.counter += 1;

// wrong way 2
this.state = { counter: this.state.counter + 1 };
```

- `setState` method
  - Updates the state AND triggers re-rendering

```
// correct way to update state
this.setState({ counter: this.state.counter + 1 });
```

Events

- React has the same set of events as vanilla JavaScript
- Syntax differences
  1. React events are written in camelCase
     - E.g., `onClick` instead of `onclick`
  2. The actions must be a function, not just an expression
     - E.g. `onClick={() => alert()}` instead of `onclick="alert()"`
- Can define event handler with component method

```
increment() { this.setState({counter: this.state.counter + 1}); }

// in render method
<button onClick={this.increment}> Click me </button>
```

- But…, this doesn’t work. Why?

Instance Binding

- A regular function binds to instance when called
- The object that calls the event handler *is not* the component
- Solutions
  1. Use the special `bind` method. Enables early binding.
     - Very ugly and unrelated to application logic. Do not use.

```
constructor() {
    this.onClick = this.onClick.bind(this);
}
```

  2. Use arrow function in class definition!
     - Arrow function capture this from outer scope, which is the *class body*

```
increment = () => { this.setState({counter: this.state.counter + 1}); }
```

Event Handling

- `event.target`
  - The element that triggered the event

```
<input type="text"
       onChange={event => this.setState({message: event.target.value})} />
```

- Tip for building complex components
  - Lifting the state up: https://reactjs.org/docs/lifting-state-up.html
    1. Pass shared states between subcomponents through their common ancestor
    2. Initial value can be passed as props to subcomponents
    3. Pass a setter function to subcomponents as change handler
- Quercus Exercise Q1
  - Build a two-way Celsius to Fahrenheit converter

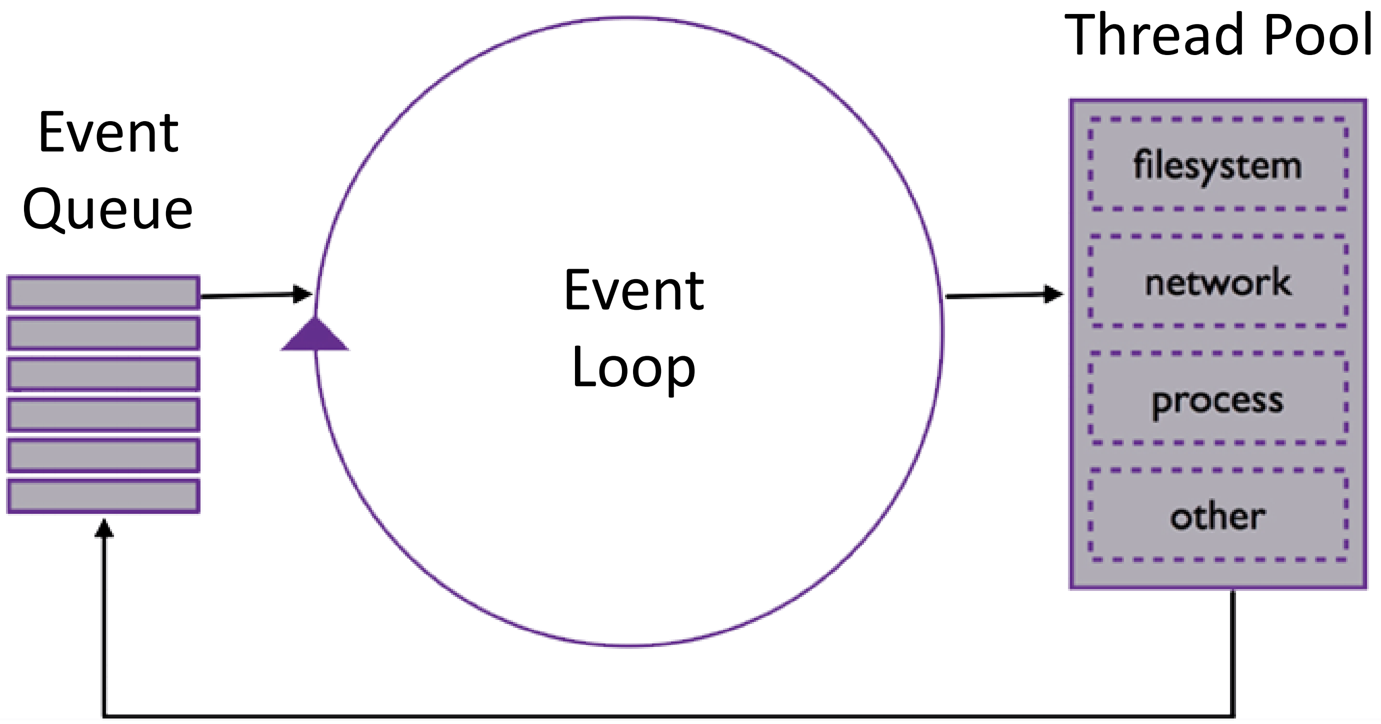
React Project

React

- Enabled by importing scripts to our HTML file
- JSX code are translated to JavaScript *every time* page is loaded
  - Very slow!
- Alternative: React project
  - Neither a backend nor HTML project
  - Frontend server that returns appropriate files per request
  - A precompiled and bundled build for production
- Node.js: a runtime environment for running JavaScript on *server-side*
  - Installation: https://nodejs.org/en/download/
  - Includes a package manager, interactive console, build tools, etc.

Node.js Processing Model

- JavaScript code still run in a single thread, but hidden threads exists
  - I/O requests can be handled asynchronously without blocking main thread



https://www.youtube.com/watch?v=zphcsoSJMvM&ab_channel=node.js

Node Console

- Can be opened with the node command
- Allows you to execute inline JavaScript code
- There is no `window` or `document` global object
  - We are no longer inside a browser
- Can execute scripts as well
  - `node <filename>`
- Console start up message:

```
~$ node
Welcome to Node.js v18.14.2.
Type ".help" for more information.
>
```

Installing Modules

- Node Package Manager (npm)
  - Extremely similar to Python pip
- Install packages via `npm install <package_name>`
  - Packages are stored in the `node_modules` directory
    - Similar to venv directory in a virtual environment
- Automatically generates and maintains a file named package.json
  - Similar to the requirement.txt file for tracking dependencies
- Node Package eXecute (npx)
  - Allows executing JS packages without having installed them
  - Will download all necessary packages to execute the command

Creating React Project

- Create React project

```
npx create-react-app <name>
```

- Run development server
  - Default port is 3000

  ```
  npm start
  ```

- Make a production build

  ```
  npm build
  ```

- Project contains same code but more organized

- Important files:
- public/index.html
  - Contains base HTML code
  - Note a div with id="root"
    - DOM rendered inside it
- src/index.js
  - Invokes ReactDOM.createRoot
  - By default, renders <App />
- src/App.js
  - Placeholder App component

Exports

- In JavaScript, each file is a module
- By default, all definitions in a module are *not* exported
  - i.e., they cannot be imported into another module
- export keyword
  - Allows variable/class/function to be exported
  - Syntax 1:

```
const var1 = 3, var2 = (x) => x + 1;
export { var1, var2 };
```

  - Syntax 2:

```
export const var1 = 3, var2 = (x) => x + 1;
```

- import statement:

```
import { var1 } from './App';
```

Default Export

- Each module can have one default export
  - Usually, it is the component defined within the module

```
export default App;
```

- Importing the default export

```
import App from './App';
```

- Importing default export *does not* require matching name
  - Can be imported under any arbitrary name

```
import OldApp from './App';
```

File Structure

- Put almost everything in the src folder
  - If not used by any React component, then place in public folder
- Images, fonts, and other static files
  - Create a src/assets folder and place them there
  - Import them directly into JS module to use them

```
import logo from './assets/logo.svg';

// in render method
<img src={logo} />
```

- Do NOT import anything into the HTML
  - All static file imports, including js and css, are handled automatically by server

Organizing Components

- All components should be placed in src/components folder
- Each component should be placed in its owner folder too
  - Name of folder should be same as component
  - JavaScript file should be named index.jsx
  - CSS file should be in same subfolder, usually named style.css
- Import local CSS file:

```
import './style.css';
```

- Import other components like this:

```
import Counter from './components/Counter';
```

Rule of thumb:

Components should be *small*, e.g., < 100 loc.

Large components should be split into small, nested child components

Final notes

- Quercus Exercise Q2
  - Redo Q1
  - Refactor the temperature converter into a React project
- Read React tutorials
  - https://reactjs.org/docs/hello-world.html
  - https://reactjs.org/tutorial/tutorial.html
- Important announcement
  - Class cancellation notice
  - Classes on March 22, March 24, and March 27 are *cancelled*
  - Please spend the extra time on timely completion of A3 and P3