COMP 431/531: Web Development

Lecture 10: Intro to React

Mack Joyner (mjoyner@rice.edu)

https://www.clear.rice.edu/comp431

Announcements & Reminders

- Quiz #2 (Events, Storage, Arrays) due Thursday, Oct. 10th at 11:59pm

- HW #4 (Draft Front-end) available today, due Tuesday, Oct. 21st at 11:59pm
  GitHub classroom hw4 repo: https://classroom.github.com/a/hfLD_40C

In-Class Exercise 9: Angular Service

React

React is a JavaScript library for creating user interfaces by Facebook and Instagram. Many people choose to think of React as the V in MVC. Facebook built React to solve one problem: building large applications with data that changes over time.

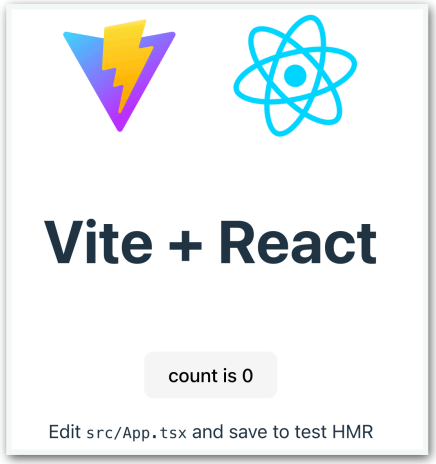
https://reactjs.org/tutorial/tutorial.html

Creating a React App (npx)

Tool intended to make it easy to use CLI out-of-the box (comes with Node.js)

Getting Started with React

```
% sudo npm create vite@latest tictactoe

> npx
> create-vite tictactoe

│
◇  Select a framework:
│  React
│
◇  Select a variant:
│  TypeScript
│
◇  Use rolldown-vite (Experimental)?:
│  No
│
◇  Install with npm and start now?
│  Yes
│
◇  Scaffolding project in /Users/mjoyner/comp431/F25/exercises/react/tictactoe...
│
◇  Installing dependencies with npm...

added 188 packages, and audited 189 packages in 10s

47 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
│
◇  Starting dev server...

> tictactoe@0.0.0 dev
> vite
```



package.json

- Like Angular, creates scripts that run using npm (e.g. npm run dev)
- Displays project dependencies (e.g. npm install)

```
{
  "name": "tictactoe",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "tsc -b && vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "^19.1.1",
    "react-dom": "^19.1.1"
  },
```


React Concepts

- React uses VDOM for fast and efficient rendering updates
  - *render* function first updates VDOM
  - React decides when to update DOM
- Divide page into Components
  - divide and conquer
  - reduce complexity
- Components can be simple or complex
  - simple Component has no explicit functionality
  - complex Component may contain state
- Components have props (attributes), state (data)

React Virtual DOM

- Virtual DOM is an abstraction of HTML DOM
- Updating VDOM is faster and more efficient
- ReactDOM allows you to dynamically build VDOM
- The *render* updates the VDOM
  - specify VDOM node
  - HTML parent tag to inject

index.html

```
1   <!doctype html>
2   <html lang="en">
3     <head>
4       <meta charset="UTF-8" />
5       <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6       <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7       <title>tictactoe</title>
8     </head>
9     <body>
10      <div id="root"></div>
11      <script type="module" src="/src/main.tsx"></script>
12    </body>
13  </html>
```


React Virtual DOM

- Virtual DOM is an abstraction of HTML DOM
- Updating VDOM is faster and more efficient
- ReactDOM allows you to dynamically build VDOM
- The *render* updates the VDOM
  - specify VDOM node
  - HTML parent tag to inject

main.tsx

```
1   import { StrictMode } from 'react'
2   import { createRoot } from 'react-dom/client'
3   import './index.css'
4   import App from './App.tsx'
5
6   createRoot(document.getElementById('root')!).render(
7     <StrictMode>
8       <App />
9     </StrictMode>,
10  )
```


React Virtual DOM with JSX

JSX is a preprocessor step that adds HTML syntax to JavaScript

With JSX

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

Without JSX

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```


JSX

- Has single top-level element
- Parentheses required for multi-level statements
- Embed JavaScript with { }

```
const bad = (
        <p>a</p>
        <p>b</p>
    )

const good = (
        <div>
            <p>a</p>
            <p>b</p>
        </div>
    )
```


React Component ES2015+

Component Lifecycle Methods

- componentWillMount
  - invoked once, on client and server, before rendering occurs
- componentDidMount
  - invoked once, on client, after rendering occurs
- componentWillUpdate
  - preprocessing, before render called
- componentDidUpdate
  - postprocessing, after render called
- componentWillUnmount
  - invoked once, before unmounting component

Composing Components and Props

```
class ParentNode extends React.Component {

    constructor(props) {
        super(props)
        this.state = { message: '?' }
    }

    render() {
        return (
            <div>
                <input placeholder="write something"
                    value={ this.state.message }
                    onChange={(e) => this.setState({ message: e.target.value }) }
                /><br/>
                <ChildNode message={ this.state.message } />
            </div>
    )}
}

ReactDOM.render(<ParentNode/>, document.getElementById('app'));
```


Composing Components and Props

```
class ChildNode extends React.Component {
    render() {
        return <span>Your message: { this.props.message }</span>
    }
}

class P
    con
    }

    render() {
        return (
            <div>
                <input placeholder="write something"
                    value={ this.state.message }
                    onChange={(e) => this.setState({ message: e.target.value }) }
                /><br/>
                <ChildNode message={ this.state.message } />
            </div>
    )}
}

ReactDOM.render(<ParentNode/>, document.getElementById('app'));
```


Composing Components and Props

Not necessary to use component when functionality can be implemented in a function

```
class ChildNode extends React.Component {
    render() {
        return <span>Your message: { this.props.message }</span>
    }
}
```

```
const ChildNode = ( message ) => <span>Your message: { message }</span>
```


React Function with Props

- Enable users to pass info from parent to child node
- In Comment example, *user* will be in props in Avatar (i.e. props.user)
- Props are read-only, function or class cannot modify its props

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <Avatar user={props.author} />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

https://reactjs.org/docs/components-and-props.html

React Props

- Props can be event listeners
- Event listener can reference prop event listener

```
onClick={()=>this.props.onClick()}
```

https://reactjs.org/docs/components-and-props.html

React Component State (old way)

- Set initial value(s) of state in Component constructor

```
// Wrong
this.state.comment = 'Hello';
```

```
// Correct
this.setState({comment: 'Hello'});
```

- Don't modify state directly
  - modify local variables
  - use *this.setState()* to modify state fields

React Hooks

- Builtin functionality at the component level
- Reduce wrapper code (e.g. no needs for render(), constructors)
- Used with functions instead of components (component methods become nested functions
- React Hooks
  - useState
  - useSelector
  - useDispatch
  - useEffect
  - useReducer
- Affects state next time “rendered”

React Node with Functions (react hook)

If there's no state, can use function to return React virtual DOM node

```
1   import { useState } from 'react'
2   import reactLogo from './assets/react.svg'
3   import viteLogo from '/vite.svg'
4   import './App.css'
5
6   function App() {
7     const [count, setCount] = useState(0)
8
9     return (
10      <>
11        <div>
12          <a href="https://vite.dev" target="_blank">
13            <img src={viteLogo} className="logo" alt="Vite logo" />
14          </a>
15          <a href="https://react.dev" target="_blank">
16            <img src={reactLogo} className="logo react" alt="React logo" />
17          </a>
18        </div>
19        <h1>Vite + React</h1>
20        <div className="card">
21          <button onClick={() => setCount((count) => count + 1)}>
22            count is {count}
23          </button>
```

function modifies local state

React Hook: useState

```
1  import { useState } from 'react';
2
3  export default function Counter() {
4    const [count, setCount] = useState(0);
5
6    function handleClick() {
7      setCount(count + 1);
8    }
9
10   return (
11     <button onClick={handleClick}>
12       You pressed me {count} times
13     </button>
14   );
15 }
16
```

function modifies local state with new value

https://react.dev/reference/react/useState

Repeating Elements in Virtual DOM

```
<li><span className="fancy">thing1</span></li>
<li>thing2</li>
{ Array(5).fill(1).map((x, i) => <li key={i}>thing{3 + i}</li>) }
```

React uses the key to identify separate repeated elements.

- thing1
- thing2
- thing3
- thing4
- thing5
- thing6
- thing7

Resources

- Thinking in React:

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

- React Docs:

  https://react.dev/

  https://www.digitalocean.com/community/tech_talks/getting-started-with-react