WEB422 - Web Programming for Apps and Services

Week 3 – Lecture Recap:

React / Next.js Introduction

Agenda

- Introduction to React
- Next.js
- Components

Introduction to React

- Front-end apps
  - Plain/native JS with BS5, Knockout, Ember, Vue.js, Angular, etc.
- MVVM design pattern
  - M – Model: data stored for your application
  - V – View: UI
  - VM – ViewModel: the data and operations on a UI
- React
  - A JS library for building interactive UI
  - the Facebook platform, originally created by Jordan Walke 2011, open sourced in May 2013.
  - the front-end engine on both browser and mobile device platforms.
  - component-based/oriented architecture

React Introduction

- Getting Started
  - A 'Hello World!' app
  - Babel - a JavaScript compiler (transcompiler)

- Toolchains, e.g.
  - Create React App: command line tool for creating React apps.
  - **Next.js**: framework for static and server-rendered app built with React.
  - **Gatsby**: for creating static website with React.
  - Others: Neutrino, Nx, Parcel Razzle etc.

Introduction to Next.js

- Next.js
  - A flexible **React framework** for building blocks to create **fast web applications**, provided by Vercel.
    - Building blocks of a web app: UI, routing, data fetching, rendering, …
  - "the best developer experience with all the features you need for production: hybrid static & server-side rendering, TypeScript support, smart bundling, route pre-fetching, and more. No config needed"
- Create a Next.js app:

```
npx create-next-app my-app --use-npm
        ✓ … TypeScript ? … No  (using arrow key '←' to select 'No')
        ✓ … ESLint ? … Yes
cd my-app
npm run dev
```

- File Structure:
  - "/pages" – holds components that act as routes, e.g., http://localhost:3000/api/hello
  - "/public" – keeps static resources such as image files
  - "/styles" – stores (global) css file and locally scope CSS Module files
  - …

Components

- **Component:** a rectangle area on UI that contains 'V' and 'VM'
- **Function Component:** a way to define a (React) component
  - example: the "Home" component (defined in pages/index.js)

```
import Head from 'next/head';
… …
export default function Home() {
     return (
          … … // JSX syntax (html-like markup in JavaScript)
     );
}
```

  - "export" – function modifier
    - public?
  - "<Head>...</Head>" or "<Image />":
    - rendering Next.js's built-in (React) components in the "Home" component.
  - className={styles.**someClass**}:
    - using the CSS rules defined in the imported CSS Module: ../styles/Home.module.css
- **Create our own Component**
  - /components/Hello.js
  - Rendered in the 'return' statement in Home component: <Hello />

Introducing JSX

- JSX: JavaScript eXtension (in React)
  - a template/markup language , but it comes with the full power of JS:
    `const element = <p className="greeting">Hello, world!</p>`
- Returning a Single Element
  - The 'return' statement must return ONE element – may need a wrapper component '<div>...</div>' or '<span>...</span>' or a "JSX Fragment" (ie: <>...</>)
- Empty Elements (void elements in html5)
  - must use closing or self-closing tag, e.g. `<br></br>` or `<br />` rather than `<br>`
- Embedding Expressions in JSX
  - Use variable(s) inside curly braces, e.g., <p>Hello {name}</p>
- JSX is an Expression Too, e.g.:

```
const elem = <p>Hello, world!</p>;
```

- Specifying Attributes with JSX
  - JSX expression as value of an attribute, e.g. `<img src={user.avatarUrl} />`
  - JSX attribute/property naming convention: camelCase, e.g. className, tableIndex (corresponding HTML element attributes: class, tableindex)

Components – Accepting "Props"

- Component: Accepting "Props"

```
export default function Hello(props) {
    return (
        <p>Hello {props.fName} {props.lName}! </p>
    );
}
```

- Data in “props” is passed from the (parent) component:

```
<Hello fName="Jason" lName="Perez" />
```

- Setting up *default* values for props

```
Hello.defaultProps = {
    fName: 'First Name’,
    lName: 'Last Name',
};
```

Introducing "Hooks" in (function) Components

- Hooks are built-in functions that are hooked to and executed due to the state or lifecycle events of a component. Hooks don't work inside class components.
- Using "Hooks" in the Clock component,
  - useState(), useEffect()
- The "state" to a component
  - used to store data within a component
  - is synchronized with the component's UI
- Adding "state" to a component <u>using the "useState()" hook</u>

  const [date, setDate] = useState(null);
  - the 'date' is the state (initialized with a null value),
    - which is a constant - cannot be changed using, e.g. state = …
  - The 'setDate' is the constant function used to modify the value of the state 'date'
- Resetting a "state": const [num, setNum] = useState(0);
  - Must use the setter function, e.g. setNum(100);
  - If the newcomputed using the previous state, e.g. to increase it's value by 1
    - setNum(prev = state is > prev + 1);

Using "Hooks" in (the Clock) component

- Using the "useEffect()" hooks in the Clock component

  The hook is used to ensure that:

  - The function is executed just after the component is "mounted" (ie: after the first render).
    - Note: the function will be also executed when a variable's value is changed if you put this variable in the second parameter (ie []), as an element, e.g. useEffect(…, [num]);
  - The statement in callback function in the "return" statement will be called when (right before) the component is "unmounted" or removed from the DOM

- "state" vs. props"
  - props get passed to the component, whereas
  - state is managed within the component.

React Components cont'd

Communication between (parent and child) components – using "props":

- Example: passing data from parent component to child component:
  - Component that accepts "props"

```
function Hello(props) {
  return (<p>Hello {props.fName} {props.lName}!</p>);
}
```

  - Rendering the (child) component and passing "props" value(s) into it:

```
<Hello fName="Jason" lName="Perez" />
```

- Example: passing data from child component to parent component:
  - Parent Component

```
function handleMessage(msg){
    console.log(`Child Says: ${msg}`)
}
return <Child sendMessage={handleMessage} />;
```

  - Child Component

```
props.sendMessage("Hello");
```

The End