BTI425/WEB422 - Web Programming for Apps and Services

Lecture Recap:

Week 4 – Handling Events & Rendering Data

Agenda

- Handling User Events
- Adding API Data
- Conditionally Displaying Data

Handling User Events

- Handling events on JS DOM objects, i.e. in HTML:

```
<button onclick="processClick()">Click Me!</button>
```

- Handling events in React Components, i.e. in JSX:

```
<button onClick={processClick}>Click Me!</button>
```

- Exapme: ClickCounter component:

```
import {useState} from 'react';
export default function ClickCounter(props){
    const [numClicks, setNumClicks] = useState(0);
    function increaseNumClicks(e){ // 'e' is the current event object
        setNumClicks(numClicks + 1);
    }
    return <button onClick={increaseNumClicks}>Clicks: {numClicks}</button>
}
```

- Adding Parameters, e.g. *e, message*
  - In JSX: `<button onClick ={(e)=>{increaseNumClicks(e, "Hello")}}>... ...`
  - In component: `function increaseNumClicks(e, msg) { ... }`

Updating State Based on Previous value

- For the state with primitive type:

  setNumClicks(numClicks + 1);

  The following is not necessary:

  setNumClicks(prevClicks => prevClicks + 1);

- For the state which is an array or object:

  setMyArray([...myArray, 'new element']);

  setPerson({...person, name: 'a new value'});

  The following will not work (**will not** cause the component to re-render):
  setMyArray(myArray.push('new element'));
  setPerson(person.name = 'a new value'});

Adding API Data

- SPA and Web indexing / SEO
- Pre-rendering
  - By default, Next.js pre-renders every page:
    - HTML is generated on server using client-side JS -> better performance and SEO
  - Each generated HTML is associated with minimal JavaScript code necessary for that page. When a (generated HTML) page is loaded by the browser, its JavaScript code runs and makes the page fully interactive. (This process is called hydration.)
- About Hydration
  - Application state (initial value): added to server-rendered HTML
    - Avoiding "hydration error" generated at "build"/compile time caused by initializing a state with a dynamic/run-time value which will be used in a pre-rendered page, e.g.:

```
const [date, setDate] = useState(null);  //  useState(new Date());  // run on server
useEffect(() => { setDate(new Date()); }, [ ]);  // executed in browser
```

Adding API Data

- Fetching API Data after Hydration

```
const [post, setPost] = useState();
useEffect(() => { fetch(`https://...  …/posts/1`); }, [ ]);  // executed in browser
```

  - Client-side data fetching with useEffect
- Fetching API Data after Hydration
  - SWR (stale-while-revalidate)

```
const { data, error } = useSWR('https://jsonplaceholder.typicode.com/posts/1', fetcher);
```

    - React Hooks (to replace useState & useEffect) for Data Fetching
    - To get a stream of data updates constantly and automatically
  - Client-side rendering in Next.js with SWR
- Fetching API Data for Pre-Rendered HTML
  - The getStaticProps is put inside index.js because
    - This will not work with custom components defined within the "components" folder
  - The Post page with the "props" (post data) is pre-rendered on server
  - SSG: Static-site generation in Next.js with getStaticProps

React Component: Conditionally Displaying Data

- The "type" of data in React components:
  - props – a js object passed-in a component
  - state – properties defined in a component
- In JSX, placing JS expression within braces { ... }
  - render the data in place, e.g. {date.toLocaleTimeString()}
  - provide a value to a property, e.g.
- Rendering data with the array of User objects – users
  - Logical && Operator (If)
  - Ternary Operator (If-Else)
  - Array.map() (Iteration)
    - can be used to render array of objects, e.g., the users array
    - attribute key="unique value" must be added to each item/row element
  - Returning Null

    if (isLoading) return null;

    return (<>……</>)

The End