BTI425/WEB422 - Web Programming for Apps and Services

Lecture Recap:

Week 5 – Pages / Routing

Agenda

- Layouts & Pages
- Dynamic Routes
- API Routes Introduction

Pages

- Pages - routing in Next.js
  - Default (existing) route:
    - Route: "/" - Renders (Home) component from "pages/index.js"
  - Defining new route & creating corresponding component, e.g.:
    - Route "/about" - Renders (About) component from "pages/about.js"
  - Defining nested routes:
    - Route: "/dashboard/" - Renders component from "pages/dashboard/index.js"
    - Route "/dashboard/preferences" - Renders component from "pages/dashboard/preferences.js"

- Component File Naming Convention - e.g., the name of NotFound component:
  - Next.js pages: not-found.js or not-found.jsx
  - React: NotFound.js
  - Angular: not-found.component.ts, not-found.component.html, not-found.component.css

Layouts

- Add file: "components/layout.js"
  - {props.children} - a placeholder to render the component (at a specific point in the layout) as children
  - <layout>[the place for children]</layout> - the "[the place for children]" is passed to the component via "props" as children
  - NOTE: Since layouts are not "pages", they must not be placed within the "pages" folder - instead, place them in a "components" folder.

- Apply layout to project
  - In File "pages/_app.js", wrap <Component {…pageProps} /> with <layout> tags:

```
<Layout>
    <Component {...pageProps} />
</Layout>
```

  - Drawback: each route takes a moment to load when using HTML Hyperlink - the HTML anchor tags (<a>…)

Client-Side Page Transitions
— Using JSX Markup Code

- Link Component
  - Used for client-side routing or links within the front-end app
  - The mechanism of Next.js to transition between pages without reloading every .js file for the application - known as "client-side routing" / "client-side route transitions".
  - only the relevant .js is loaded when each route is first accessed - known as "Lazy Loading"
  - Note: this Link component is different the Link component in React project.

- The Link component accepts the following props:
  - href, as, logacyBehavior, passHref, profetch, replace, scroll, shallow, locale
  - e.g.

```
<Link href={{  pathname: '/about', query: { name: 'test' }  }} > About … </Link>
<Link href="/#hashid" scroll={false} as="/dashboard"> … … </Link>
```

Client-Side Page Transitions
— Using JavaScript Code

- useRouter Hook
  1. Import the "useRouter" hook:
     ```
     import { useRouter } from 'next/router';
     ```
  2. Obtain a "router" object:
     ```
     const router = useRouter();
     ```
  3. Navigate/Transition to a new route:
     ```
     router.push('/about');
     ```
- The router object itself has many useful properties, e.g.: pathname, query
  ```
  router.push({  pathname: '/about', query: { name: 'test' }  });  // with query parameter
  ```
- This method accepts three arguments, i.e.:
  - url
  - as
  - option:
    - scroll, shallow, locale

Dynamic Routes

1. Paths that support "route" parameters:
   - Defining paths - add file under the "/pages" folder, e.g.:
     - "pages/products/[id].js"
   - Using "route" parameters
     - "/products/*id*" - where *id* can be any value, e.g., "/products/5"

2. Paths that support ""query" parameters:
   - Defining paths
     - "pages/products/index.js"
     - no change to our filenames to support optional query strings
   - Using "query" parameters
     - "/products?*query*" - where *query* can be an optional query string, e.g.: "/products?page=1&category=stationary"

Reading Route Parameters

- To read the route parameter "id" from path "/products/id", e.g., "/products/5", the "router" object must be used through the "useRouter" hook
- Add File "pages/products/[id].js":

```
import { useRouter } from 'next/router';

export default function Product() {
const router = useRouter();
const { id } = router.query;

return <p>Product: {id}</p>
}
```

Reading Query Parameters

- Reading the query parameters - "page" and "category" from path:
  /products?page=1&category=stationery

- Add File "pages/products.js":

```
import { useRouter } from 'next/router'

export default function Products() {
    const router = useRouter()
    const { page } = router.query;
    const { category } = router.query;

    if (page && category) {
        return <p>Products for page: {page} &amp;amp; category: {category}</p>
    } else {
        return <p>Page and/or Category Parameters Missing</p>
    }
}
```

Dynamic Routes with 'getStaticProps'

- File: "pages/post/[id].js"

```
export async function getStaticPaths() { // pre-render and support post/1 through post/5 only
    return { paths: [{ params: { id: "1" } }, { params: { id: "2" } }, { params: { id: "3" } }, { params: { id: "4" } },
        { params: { id: "5" } } ], fallback: false // any pages not identified above, will result in a 404 error, ie post/6
    }
}

export async function getStaticProps(context) {
    const res = await fetch(`https://jsonplaceholder.typicode.com/posts/${context.params.id}`)
    const data = await res.json()

    return { props: { post: data } }
}

export default function Post(props) {
    return <>
        <p><strong>User ID:</strong> {props.post.userId}</p>
        <p><strong>ID:</strong> {props.post.id}</p>
        <p><strong>Title:</strong> {props.post.title}</p>
        <p><strong>Body:</strong> {props.post.body}</p>
    </>
}
```

Custom Error Pages

- File: "pages/404.js"

```
export default function Custom404() {
    return <h1>404 - Page Not Found</h1>
}
```

- File: "pages/500.js"

```
export default function Custom500() {
    return <h1>500 - Server-side error occurred</h1>
}
```

"Api" Routes Introduction

- Next.js feature - specify routes for a Web API to be executed on the same server serving your site.
  - It not only pre-renders and serves your files but also can act as a back-end API for your application
- Route Definitions
  - Access a "hello" route from the "api" folder, ie: "http://localhost:3000/api/hello"
  - File: "pages/api/hello.js"

```
export default function handler(req, res) {
    res.status(200).json({ name: 'John Doe' });
}
```

- 'req' object properties:
  - req.cookies, req.query, req.body
- 'res' object functions:
  - res.status(code), res.json(body), res.send(body), res.redirect([status,] path)

"Api" Routes Introduction

- HTTP Methods

```
export default function handler(req, res) {
    const { method } = req;

    switch (method) {
        case 'GET':
            res.status(200).json({ name: 'John Doe' });
            break;
        case 'POST':
            // return the 'name' value provided in the body of the rquest
            res.status(200).json({ name: req.body.name });
            break;
        default:
            // send an error message back, indicating that the method is not supported by this route
            res.setHeader('Allow', ['GET', 'POST']);
            res.status(405).end(`Method ${method} Not Allowed`);
    }
}
```

- Dynamic Routes
  - File: "/pages/api/users/[id].js"

```
export default function handler(req, res) {
    const { id } = req.query; // "id" route parameter
    res.status(200).json({ name: `user ${id}` });
}
```

Web API Structure - CRUD Operations

- File: "/pages/api/users/index.js" - for C, R (Get All)

```
export default function handler(req, res) {
    const { method } = req;

    switch (method) {
        case 'GET':
            res.status(200).json({ name: 'John Doe' });
            break;
        case 'POST':
            // return the 'name' value provided in the body of the rquest
            res.status(200).json({ name: req.body.name });
            break;
        default:
            // send an error message back, indicating that the method is not supported by this route
            res.setHeader('Allow', ['GET', 'POST']);
            res.status(405).end(`Method ${method} Not Allowed`);
}   }
```

- File: "/pages/api/users/[id].js" - for R (Get All), U, D

```
export default function handler(req, res) {
    const { id } = req.query;
    const { name } = req.body;
    const { method } = req;
    switch (method) {
        case 'GET':
            // Read data from your database
            res.status(200).json({ message: `TODO: Get User with id: ${id} ` });
            break;
        case 'PUT':
            // Update data in your database
            res.status(200).json({ message: `TODO: Update User with id: ${id} - Set Name: ${name}` });
            break;
        case 'DELETE':
            // Delete data in your database
            res.status (200).json({ message: `TODO: Delete User with id: ${id}` });
            break;
        default:
            res.setHeader('Allow', ['GET', 'PUT', 'DELETE']);
            res.status(405).end(`Method ${method} Not Allowed`);
}   }
```

The End