Routing and Forms

Muhammad Zahir Junejo

Lecture - Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
  - Please review Code of Conduct (in Student Undertaking Agreement) if unsure
- No question is daft or silly - **ask them!**
- Q&A session at the end of the lesson, should you wish to ask any follow-up questions.
- Should you have any questions after the lecture, please schedule a mentor session.
- For all non-academic questions, please submit a query: www.hyperiondev.com/support

Lecture Objectives

1. The significance of routing in single-page applications.
2. Introduction to React Router as a powerful library for routing in React applications.
3. How to set up React Router, define routes, and handle navigation using **<Link>** components.
4. Managing dynamic routes with route parameters and handling redirects.

Lecture Objectives (Cont'd)

5. The significance of forms as a means of obtaining user input in web applications.
6. Handling form submissions, accessing data, and sending it to the server.
7. Managing different types of user inputs, including text, numbers, checkboxes, radios, and files.
8. Exploring form libraries and frameworks that simplify form management and validation.

Understanding Routing

- Routing is the process of determining how an application responds to a specific URL or path.
- In single-page applications (SPAs), routing allows different content to be displayed based on the URL without reloading the entire page.
- React.js provides tools and libraries to implement routing effectively.

Single Page Applications (SPAs)

- Single Page Applications (SPAs) are a modern approach to web application design and development.
- Key Characteristics:
  - Loads a single HTML page initially.
  - Updates content using AJAX, providing a smoother user experience.
  - Improves performance and reduces server load.
- Advantages:
  - Faster navigation and interaction.
  - Seamlessly transition between views.
  - Enhanced user experience, similar to native apps.
- Example SPAs: Gmail, Facebook, X, Trello.
- Built with Technologies like: React, Angular, Vue.js.

React Router

- ❏ React Router is a popular library for handling routing in React applications.
- ❏ It provides a set of components that enable dynamic navigation and rendering based on URLs.
- ❏ React Router enables building multi-page-like applications while maintaining the SPA experience.

Setting Up React Router

- Installation:
  - Install React Router using npm:
    - npm install react-router-dom

- Importing:
  - Import necessary components from 'react-router-dom':
    - import { BrowserRouter, Route, Link } from 'react-router-dom';

Basic Routing

- <BrowserRouter> Component:
  - Wrap your entire application with the <BrowserRouter> component from React Router.
  - Provides a foundation for routing by managing URL changes and rendering the appropriate components.

- <Route> Component:
  - Define routes using the <Route> component.
  - Specify the path and the component to render for that path.
  - Use the exact prop for exact path matches.

Navigation with Links

- **<Link>** Component:
  - Use the <Link> component to create navigation links.
  - Specify the to prop with the target path.

```
import { Link } from 'react-router-dom';
function Navigation() {
  return (
    <nav>
      <Link to="/">Home</Link>
      <Link to="/about">About</Link>
    </nav>
  );
}
```

Route Parameters

- Route Parameters:
  - Import useParams hook from react-router-dom package.
  - Use colon notation to define dynamic route parameters.
  - Access parameters using the match object in the component.

```
<Route path="/users/:id" component={UserDetail} />

// Accessing the param inside the UserDetail component
const { id } = useParams();
```

Nested Routes

- Nested Routes:
  - Use nested <Route> components for complex UI structures.
  - Nested routes are defined within parent components.

```
<Route path="/dashboard" component={Dashboard}>
  <Route path="/dashboard/profile" component={UserProfile} />
  <Route path="/dashboard/settings" component={Settings} />
</Route>
```

Forms

- Forms are a fundamental part of web applications that allow users to submit data to the server.
- They collect user inputs such as text, numbers, selections, and more.
- Forms are essential for interacting with users, gathering information, and facilitating data-driven actions.

Handling Form Submissions

- <form> Element:
  - Wrap form components with the <form> HTML element.
  - Use onSubmit event to handle form submissions.
- Prevent Default:
  - Use event.preventDefault() to prevent the default form submission behavior.
- Accessing Form Data:
  - Collect data from state or input references to create a payload.
- Sending Data to Server:
  - Utilize APIs, libraries, or frameworks to send form data to the server.

```
function handleSubmit(event) {
  event.preventDefault();
  const formData = { name: nameValue, email: emailValue };
  // Send formData to the server
}
```

Validation and Error Handling

- Client-Side Validation:
  - Validate input data on the client side before submitting.
  - Use built-in HTML attributes or custom functions.
- Error Messaging:
  - Provide clear error messages for validation failures.
  - Inform users about invalid inputs and required fields.
- State Handling:
  - Use React state to manage input errors and display messages.

Validation and Error Handling

```
const [emailError, setEmailError] = useState('');

function handleEmailChange(event) {
  const email = event.target.value;
  if (!isValidEmail(email)) {
    setEmailError('Invalid email format');
  } else {
    setEmailError('');
  }
}
```

Handling Different Input Types

- Text Inputs:
  - Use the <input type="text"> element.
- Number Inputs:
  - Use the <input type="number"> element.
  - Utilize the min, max, and step attributes.
- Checkbox and Radio Inputs:
  - Use the <input type="checkbox"> and <input type="radio"> elements.
- Select Inputs:
  - Use the <select> element for dropdown lists.
- Populate options using <option> elements.

Textareas and File Inputs

- Textarea:
  - Use the <textarea> element for multi-line text inputs.
  - Set the value prop to control its content.
- File Input:
  - Use the <input type="file"> element for file uploads.
  - Use the onChange event to handle file selection.

```
<textarea value={textareaValue} onChange={handleTextareaChange} />
<input type="file" onChange={handleFileChange} />
```

Form Libraries and Frameworks

- Form Libraries:
  - Libraries like Formik and react-hook-form provide enhanced form management capabilities.
  - Offer built-in validation, form state management, and error handling.

- Integration with UI Frameworks:
  - Integrate forms seamlessly with UI frameworks like Material-UI, Ant Design, etc.
  - UI frameworks often provide form components with enhanced features.

References

- https://reactrouter.com/en/main/start/concepts
- https://react.dev/reference/react-dom/components#form-components
- https://formik.org/docs/tutorial
- https://mui.com/material-ui/getting-started/

Questions and Answers

Thank You!