State Management and Events

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. React Hooks
2. State variables
3. Attaching event handlers directly within JSX
4. Global State Management

Understanding Hooks

- Hooks are functions that allow you to "hook into" React state and lifecycle features from functional components.
- Eliminate the need for class components in many cases.
- Simplify the code by reducing nesting and improving reusability.
- Enable better separation of concerns within components.

Basic Hooks

- The useState hook allows functional components to manage state variables and re-render based on changes.

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);
  // ...
}
```

Effects Hook

- The **useEffect** hook is a powerful tool that enables functional components to manage side effects. Side effects include operations such as data fetching, DOM manipulation, and subscriptions.

```
import React, { useState, useEffect } from 'react';
function DataFetcher() {
  const [data, setData] = useState(null);

  useEffect(() => {
    // Fetch data and update state

  }, [  /* Dependency array */           ]);
}
```

Effects Hook

- Different faces of **useEffect**:
  - No Dependency Array: The effect runs after every render.
  - Empty Dependency Array: The effect runs only after the initial render (like **componentDidMount**).
  - Dependency Array: The effect runs when values in the array change (like **componentDidUpdate**).

Custom Hooks

- Custom hooks are functions that encapsulate logic and can be reused across different components.

```
import { useState, useEffect } from 'react';
function useCustomHook(initialValue) {
  const [value, setValue] = useState(initialValue);
  useEffect(() => {
    // Custom logic
  }, []);

  return value;
}
```

Understanding State Variables

- State variables in React are dynamic pieces of data that can change over time and affect the rendering of a component. They allow you to store and manage information that may be modified based on user interactions, external data, or any other triggers.
- State variables enable components to re-render and reflect changes in the UI when their values are updated.
- They are essential for building interactive interfaces that respond to user actions like clicks, inputs, and more.
- State variables are specific to individual components, providing localized data management.

Defining State Variables

- State is defined using the useState hook.

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);
}
```

Updating State Variables

- To update state in function components, use the setter function provided by the useState hook.

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  const handleClick = () => {
    setCount(count + 1);
  };
}
```

Rendering State Variables

- When state changes, React re-renders the functional component, reflecting the updated data.
- State values can be used within JSX with {}.

```
return <p>Count: {count}</p>;
```

Event Handling

- Event handling remains the same in function components as in class components.
- Attach event handlers directly within JSX.

```
return <button onClick={handleClick}>Click me</button>;
```

Event Handling

- Event handling remains the same in function components as in class components.
- Attach event handlers directly within JSX.

```
return <button onClick={handleClick}>Click me</button>;
```

Event Handling

- Example:

```
import React, { useState } from 'react';

function ClickCounter() {
  const [count, setCount] = useState(0);

  const handleClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <button onClick={handleClick}>Click me</button>
      <p>Click count: {count}</p>
    </div>
  );
}
```

Preventing Default Behavior in Function Components

- Preventing default behavior works the same way in function components.

```
const handleLinkClick = (event) => {
  event.preventDefault();
  // Your code here
};
```

What is Global State Management?

- Global State Management involves managing the state of your application that is shared across multiple components.
- In larger applications, passing state between components can become complex and inefficient.
- Global state management libraries like Redux offer a solution to this problem.

Introduction to Redux

- Redux is a popular JavaScript library for managing application state.
- Provides a predictable state container for your JavaScript apps.
- Works well with various UI libraries and frameworks.
- Core Concepts:
  - Store: Holds the entire state tree of your application.
  - Actions: Describes what happened in your app.
  - Reducers: Specify how the state changes in response to actions.
  - Selectors: Retrieve specific data from the state.

Benefits of Redux

- Centralized State:
  - Simplifies state management by centralizing it.
- Predictable State Changes:
  - Debugging becomes easier with predictable state changes.
- Time Traveling Debugger:
  - Redux DevTools enable you to trace past states and actions.
- Scalability:
  - Suitable for large applications with complex state structures.

When to Use Redux

- Use Redux when:
  - Your application's state becomes complex and interconnected.
  - Multiple components need access to the same state.
  - You need a reliable way to manage and debug state changes.

References

- https://react.dev/learn/describing-the-ui
- https://react.dev/learn/adding-interactivity
- https://react.dev/learn/managing-state
- https://redux.js.org/usage/
- https://redux.js.org/tutorials/fundamentals/part-1-overview
- https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow
- https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers
- https://redux.js.org/tutorials/fundamentals/part-4-store

Questions and Answers

Thank You!