Day 5 :

Class project:1

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

// ErrorBoundary component to catch errors
class ErrorBoundary extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  // This lifecycle method is invoked after an error has been thrown by a descendant component
  componentDidCatch(error, info) {
    this.setState({ hasError: true });
    console.error('Error caught by error boundary:', error, info);
  }

  render() {
    return this.state.hasError ? (
      // Render a custom fallback UI when an error occurs
      <div>Something went wrong.</div>
    ) : (
      // Render the children if there is no error
      this.props.children
    );
  }
}

// Example component that renders an array of list elements
const ListComponent = ({ items }) => {
  return (
    <ul>
      {items.map(item => (
        <li key={item.id}>{item.text}</li>
      ))}
    </ul>
  );
};

const App = () => {
  const data = [
    { id: 1, text: 'Pasta' },
    { id: 2, text: 'Dhaal' },
    { id: 3, text: 'Wheat' },
    { id: 4, text: 'Rice' },
  ];

  return (
    <div>
      <h1>React Error Handling Example with Error Boundary</h1>

      {/* Wrap the component tree with the ErrorBoundary component */}
      <ErrorBoundary>
        <h2>List Component Inside Error Boundary</h2>
        {/* Pass the correct items prop to ListComponent */}
        <ListComponent items={data} />
      </ErrorBoundary>
    </div>
  );
};

export default App;
```

output:

React Error Handling Example with Error Boundary

List Component Inside Error Boundary

- Pasta
- Dhaal
- Wheat
- Rice

Something error code:

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

// ErrorBoundary component to catch errors
class ErrorBoundary extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  // This lifecycle method is invoked after an error has been thrown by a descendant component
  componentDidCatch(error, info) {
    this.setState({ hasError: true });
    console.error('Error caught by error boundary:', error, info);
  }

  render() {
    return this.state.hasError ? (
      // Render a custom fallback UI when an error occurs
      <div>Something went wrong.</div>
    ) : (
      // Render the children if there is no error
      this.props.children
    );
  }
}

// Example component that may throw an error
const ErrorProneComponent = () => {
  // Intentional error: Simulating an error within the component
  throw new Error('Simulated error!');
};

const App = () => {
  return (
    <div>
      <h1>React Error Handling Example with Error Boundary</h1>

      {/* Wrap the component tree with the ErrorBoundary component */}
      <ErrorBoundary>
        <h2>Example Component Inside Error Boundary</h2>
        {/* Intentionally trigger an error */}
        <ErrorProneComponent />
      </ErrorBoundary>
```

```
    </div>
  );
};

export default App;
```

output:

React Error Handling Example with Error Boundary

Something went wrong.

Pro 3:

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


// ErrorBoundary component to catch errors
class ErrorBoundary extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false, error: null, errorInfo: null };
  }

  // This lifecycle method is invoked after an error has been thrown by a descendant component
  componentDidCatch(error, info) {
    this.setState({ hasError: true, error, errorInfo: info });
    console.error('Error caught by error boundary:', error, info);
  }

  render() {
    if (this.state.hasError) {
      // Render a custom fallback UI with the error message
```

```
      return (
        <div>
          <h3>Something went wrong:</h3>
          <p>{this.state.error && this.state.error.toString()}</p>
          <p>Component stack trace:</p>
          <pre>{this.state.errorInfo && this.state.errorInfo.componentStack}</pre>
        </div>
      );
    }

    // Render the children if there is no error
    return this.props.children;
  }
}

// Example component that may throw an error based on a condition
const ErrorProneComponent = ({ throwError }) => {
  if (throwError) {
    throw new Error('Simulated error!');
  }

  return <div>No error occurred.</div>;
};

const App = () => {
```

```
  return (
    <div>
      <h1>React Error Handling Example with Conditional Error Boundary</h1>

      {/* Wrap the component tree with the ErrorBoundary component */}
      <ErrorBoundary>
        <h2>ErrorProneComponent Inside Error Boundary (Condition: true)</h2>
        {/* Conditionally pass true to simulate an error */}
        <ErrorProneComponent throwError={true} />
      </ErrorBoundary>

      <ErrorBoundary>
        <h2>ErrorProneComponent Inside Error Boundary (Condition: false)</h2>
        {/* Conditionally pass false to avoid an error */}
        <ErrorProneComponent throwError={false} />
      </ErrorBoundary>
    </div>
  );
};

export default App;
```

output:

React Error Handling Example with Conditional Error Boundary

Something went wrong:

Error: Simulated error!

Component stack trace:

```
    at ErrorProneComponent (http://localhost:3001/static/js/bundle.js:88:3)
    at ErrorBoundary (http://localhost:3001/static/js/bundle.js:28:5)
    at div
    at App
```

ErrorProneComponent Inside Error Boundary (Condition: false)

No error occurred.