Lecture 5: Input / Output (I/O) Design

IST 3108 Application Development

Dr. Peter Khisa Wakholi

Dept of Information Systems, Makerere University

Outline

Introduction

- In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the input devices such as PC, MICR, OMR, etc
- Input and Output (I/O) design is a critical aspect of application development.
- It focuses on how users interact with the system through input mechanisms and how the system provides feedback and output to users.
- The quality of system input determines the quality of system output.

What is a requirement?

- A requirement is a statement about something the application must accomplish.
- Good requirements are
  - Verifiable — it can be written in a way that you can check that it’s met
  - Necessary — you can’t take it away
  - Irreducible — it can’t be split into smaller requirements
  - Don’t prescribe a design — it can be met using multiple different designs.

User Requirements to I/O Design

1. **Understanding User Requirements**
   - Start by thoroughly understanding and documenting user requirements. This includes gathering information through user interviews, surveys, and feedback.
2. **Identifying Key User Interactions**
   - Identify the key interactions that users will have with the application. These interactions can include user inputs, data displays, form submissions, and more.
3. **Creating User Stories**
   - Break down user requirements into user stories. Each user story should represent a specific user task or action.
   - Example User Story: "As a student, I want to register for courses online."

User Requirements to I/O Design ….

4. Wireframing and Sketching

- Create wireframes or sketches that visually represent the layout and components of the user interface. Sketching can be done on paper or using digital tools.

5. **Mapping User Stories to Components**

- Map each user story to specific React components. Identify which components will be responsible for handling user interactions and displaying data.
- Example Component Mapping:
  - User Registration Form Component
  - Course Selection Component
  - User Profile Component

User Requirements to I/O Design ....

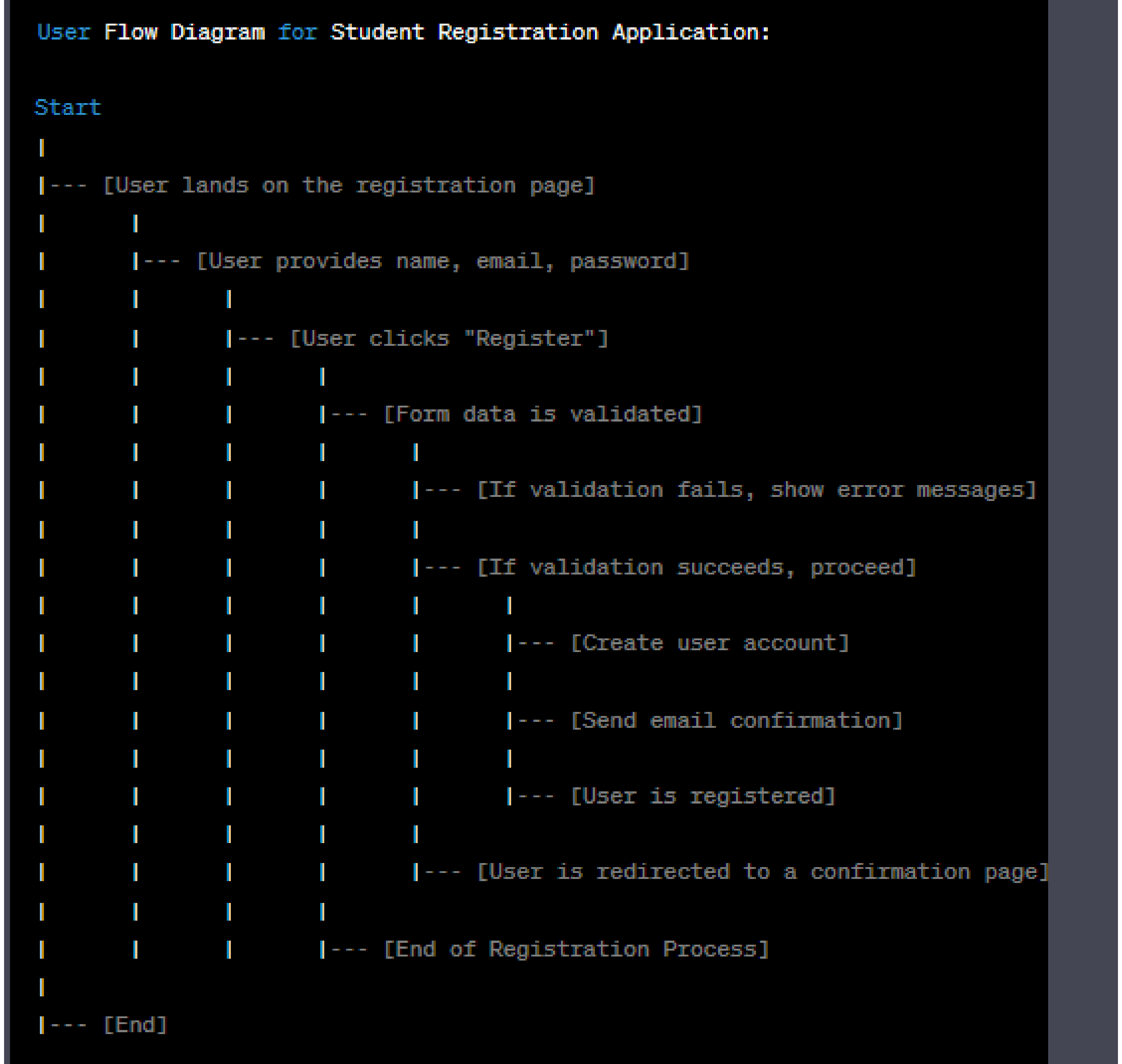
6. Defining Component Properties (Props) and States

- Determine what data (props) each component will need to function. Consider the information required for rendering, such as user details, course options, etc.
- Identify the component's state variables, which represent dynamic data that can change during user interactions.

7. User Flow Diagram

- Create a user flow diagram to visualize how users will navigate through the application. This diagram helps ensure a logical and intuitive flow.
- Example User Flow Diagram:

User Flow diagram example



```
User Flow Diagram for Student Registration Application:

Start
|
|--- [User lands on the registration page]
|      |
|      |--- [User provides name, email, password]
|      |      |
|      |      |--- [User clicks "Register"]
|      |      |      |
|      |      |      |--- [Form data is validated]
|      |      |      |      |
|      |      |      |      |--- [If validation fails, show error messages]
|      |      |      |      |
|      |      |      |      |--- [If validation succeeds, proceed]
|      |      |      |      |      |
|      |      |      |      |      |--- [Create user account]
|      |      |      |      |      |
|      |      |      |      |      |--- [Send email confirmation]
|      |      |      |      |      |
|      |      |      |      |      |--- [User is registered]
|      |      |      |      |
|      |      |      |      |--- [User is redirected to a confirmation page]
|      |      |      |
|      |      |      |--- [End of Registration Process]
|
|--- [End]
```

User Requirements to I/O Design .....

8. Prototyping and Mockups

- Develop interactive prototypes or mockups to test the proposed I/O design with real users. Tools like Figma, Sketch, or Adobe XD can be used for this purpose.

9. User Testing and Feedback

- Conduct usability testing sessions with actual users to gather feedback on the prototypes. Adjust the I/O design based on user insights and preferences.

User Requirements to I/O Design ….

10. Implementing I/O Design with React

- Begin implementing the I/O design by creating React components based on the wireframes and user stories.
- Utilize React's component structure, state management, and props to handle user interactions and data display.

11. Testing and Iteration

- Test the application as you implement components. Ensure that it behaves as expected, is user-friendly, and meets the requirements.
- Iterate on the I/O design and code based on testing results and user feedback.

Key I/O Design Principles

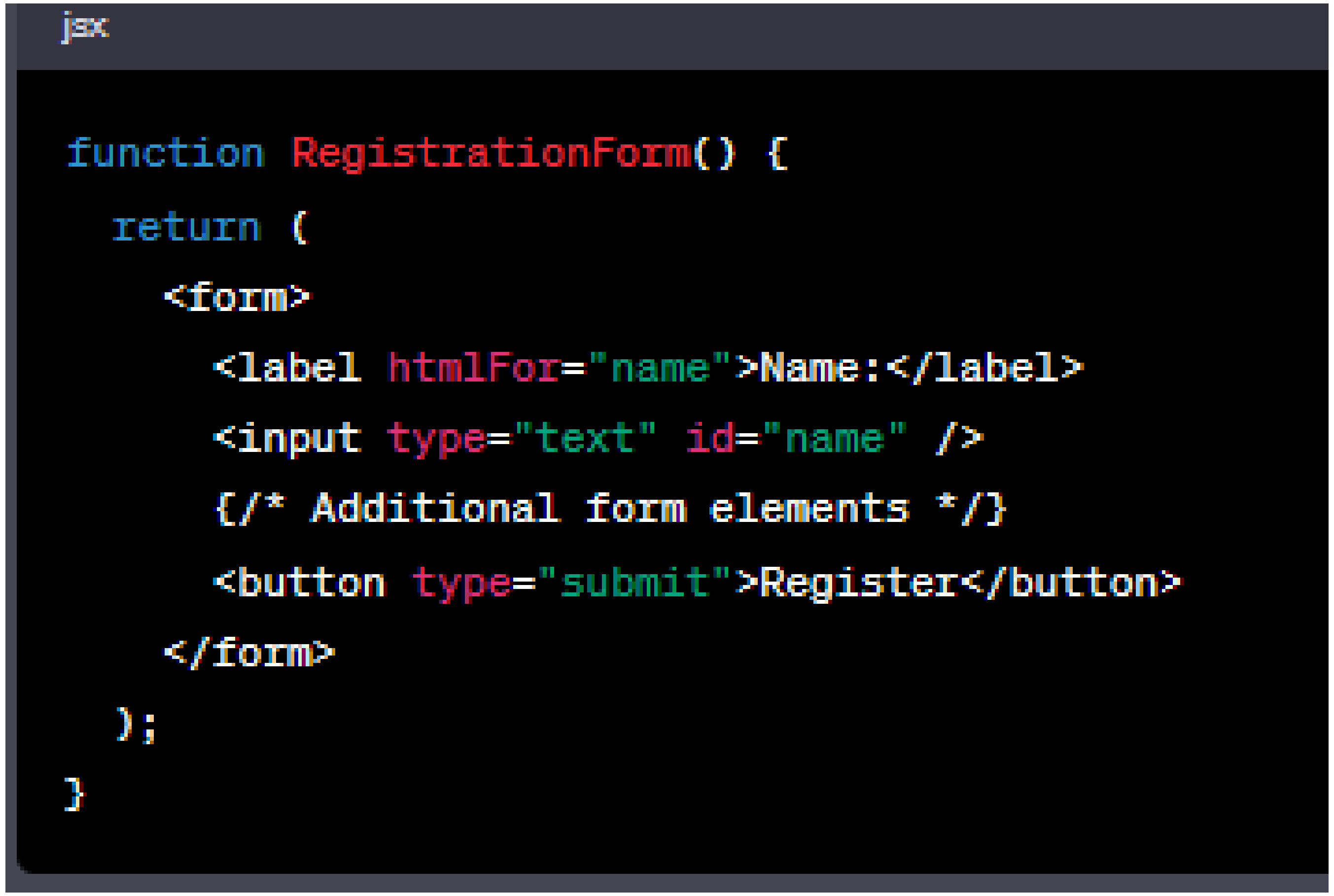
- **Simplicity:** Keep the I/O design simple and intuitive. Users should be able to understand and use the interface with minimal effort.
- **Consistency:** Maintain consistency in the design of input forms, buttons, labels, and navigation elements. A consistent layout helps users feel more comfortable.
- **Efficiency:** Design I/O processes to be efficient. Minimize the number of steps and clicks required to perform common tasks.
- **Feedback:** Provide clear and immediate feedback to users. Let them know that their actions were successful or if an error occurred.

Key I/O Design Principles …..

- **Error Handling:** Design input forms with error prevention and error handling in mind. Give users guidance on how to correct errors.
- **Flexibility:** Consider different types of users and their needs. Make the interface adaptable to various devices and screen sizes.
- **Accessibility:** Ensure that the interface is accessible to users with disabilities. Follow accessibility standards (e.g., WCAG) to accommodate all users.

Principles of I/O Design with React

- **Simplicity and Consistency:**
- React components are inherently modular, promoting simplicity and consistency in design. Each component can encapsulate a specific UI element or functionality, making it easier to manage and maintain.
- Example: A simple React registration form component:


jsx

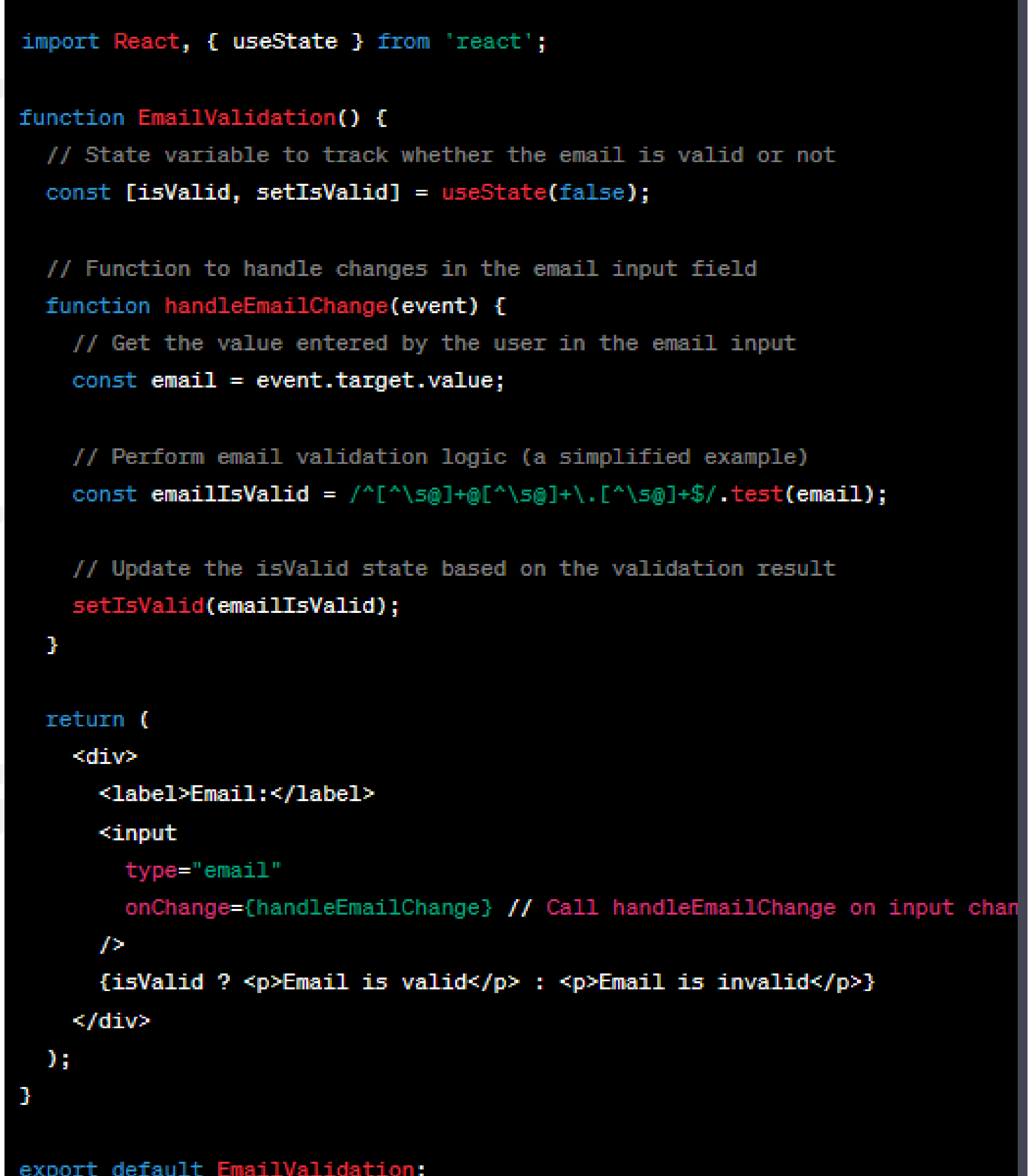
```
function RegistrationForm() {
  return (
    <form>
      <label htmlFor="name">Name:</label>
      <input type="text" id="name" />
      {/* Additional form elements */}
      <button type="submit">Register</button>
    </form>
  );
}
```

Efficiency:

- React's virtual DOM and re-rendering optimizations ensure that UI updates are efficient. Components update only when necessary, reducing unnecessary rendering.
- **Example:**
  - Suppose you have a web page with multiple components, including a user profile panel and a sidebar with notifications. If a user changes their profile picture, React's Virtual DOM will identify that only the user profile panel needs to be updated, not the entire page. This focused update reduces the computational cost and improves performance.
- **Benefits:**
  - The key benefit of this approach is improved performance. By minimizing unnecessary rendering, React ensures that your application remains responsive even when dealing with complex user interfaces and frequent data updates.

Feedback:

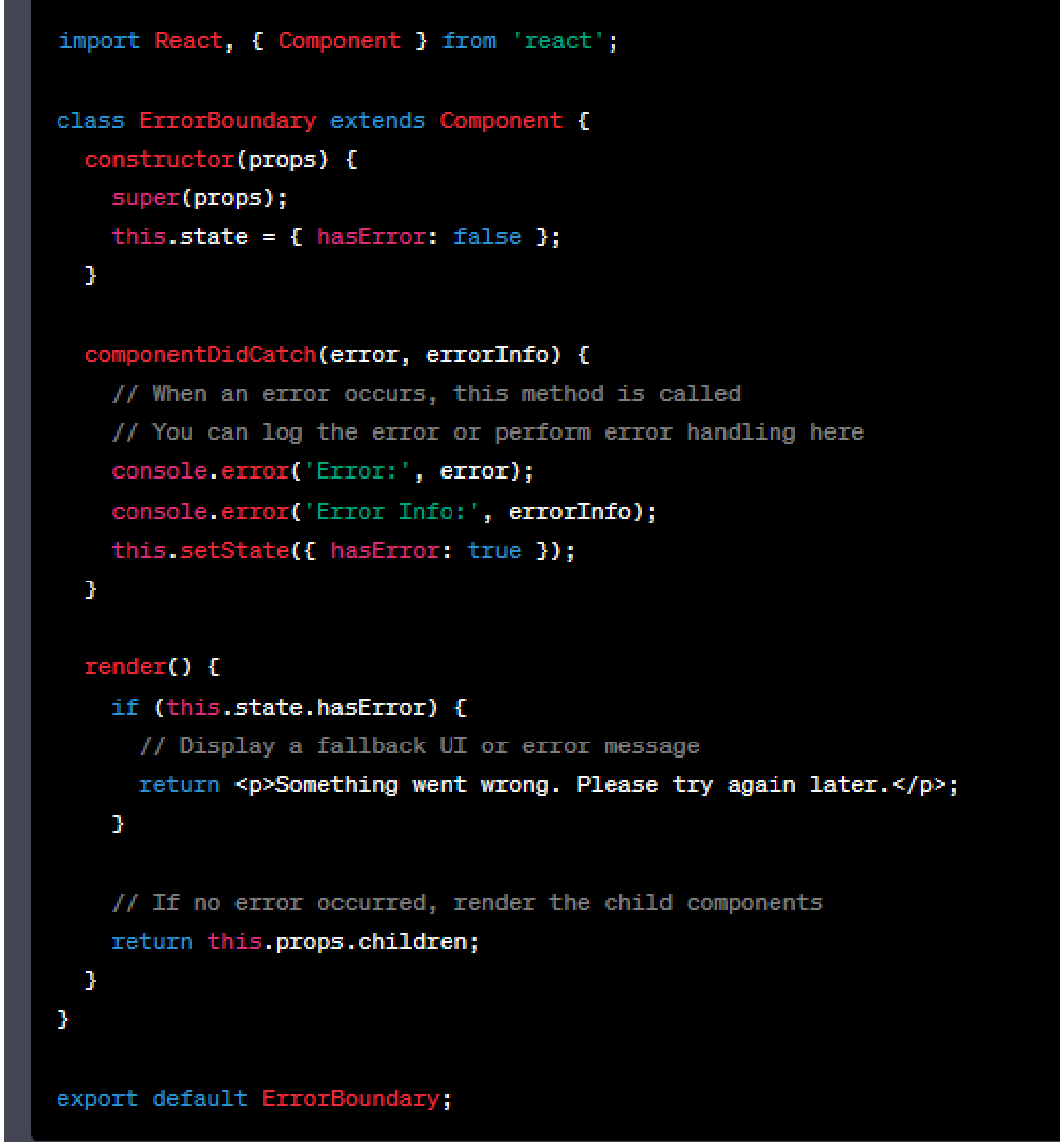
- React components can provide real-time feedback to users through state management. For instance, form validation errors or success messages can be displayed dynamically.
- Example: Real-time email validation in a React component:
- As the user types or modifies their email in the input field, the component provides real-time feedback. If the email format matches the regex pattern, "Email is valid" is displayed; otherwise, "Email is invalid" is shown.



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

function EmailValidation() {
  // State variable to track whether the email is valid or not
  const [isValid, setIsValid] = useState(false);

  // Function to handle changes in the email input field
  function handleEmailChange(event) {
    // Get the value entered by the user in the email input
    const email = event.target.value;

    // Perform email validation logic (a simplified example)
    const emailIsValid = /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);

    // Update the isValid state based on the validation result
    setIsValid(emailIsValid);
  }

  return (
    <div>
      <label>Email:</label>
      <input
        type="email"
        onChange={handleEmailChange} // Call handleEmailChange on input chan
      />
      {isValid ? <p>Email is valid</p> : <p>Email is invalid</p>}
    </div>
  );
}

export default EmailValidation;
```

Error Handling:

- React components can handle errors gracefully, displaying error messages or fallback UI when errors occur.
- Example - Error boundary component in React: In a React application, errors can sometimes occur during the rendering of components. These errors might be due to unexpected issues, such as JavaScript runtime errors or unexpected data.
- When an error occurs within the components wrapped by the ErrorBoundary, the Error Boundary component steps in to prevent the entire application from crashing. Instead, it displays a user-friendly error message or fallback UI, providing a better user experience.



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

class ErrorBoundary extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  componentDidCatch(error, errorInfo) {
    // When an error occurs, this method is called
    // You can log the error or perform error handling here
    console.error('Error:', error);
    console.error('Error Info:', errorInfo);
    this.setState({ hasError: true });
  }

  render() {
    if (this.state.hasError) {
      // Display a fallback UI or error message
      return <p>Something went wrong. Please try again later.</p>;
    }

    // If no error occurred, render the child components
    return this.props.children;
  }
}

export default ErrorBoundary;
```

Flexibility and Responsiveness:

- React components can be designed to be responsive by using CSS media queries or responsive libraries. They adapt to different screen sizes and orientations.
- CSS media queries are a fundamental tool for creating responsive web designs. You can use media queries to apply different styles to a component based on the screen size and other device characteristics.
- There are libraries in the React ecosystem, such as react-responsive and react-bootstrap, that provide responsive components and utilities to simplify responsive design.

References

- **React Official Documentation:**
  - React's official documentation is an excellent resource for learning how to handle input/output in React components. It covers topics like form handling, event handling, and more.
  - Website: React Official Documentation
- **React Forms:**
  - The official React documentation provides detailed information on working with forms and handling user input in React applications.
  - Documentation Link: Forms - React

References ....

- **React Event Handling:**
  - Understanding how to handle user interactions and events is crucial in React. The official documentation explains event handling in React components.
  - Documentation Link: Handling Events - React
- **React Bootstrap:**
  - If you're interested in building responsive and user-friendly UIs in React, React Bootstrap is a popular library that offers responsive components and design guidelines.
  - GitHub Repository: React Bootstrap

References ….

- **Material-UI:**
  - Material-UI is another widely used library for building UIs with React, following Google's Material Design principles. It provides various components and theming options.
  - Website: Material-UI
- **Online React Tutorials and Articles:**
  - Websites like Medium, Dev.to, and freeCodeCamp often have tutorials and articles related to React input/output design, including best practices and examples.
  - Examples: Medium React Tutorials, Dev.to React Articles
- **React Design Patterns:**
  - Books like "React Design Patterns and Best Practices" by Carlos Santana Roldán cover various design patterns and practices in React, which are relevant to input/output design.
  - Book Link: React Design Patterns and Best Practices

References ....

- **React Design System Libraries:**
  - Explore design system libraries like Storybook or Bit, which can help you create and manage reusable UI components efficiently.
  - Storybook Website: Storybook
- **Online Courses:**
  - Platforms like Udemy, Coursera, and edX offer React-focused courses that cover UI design principles and best practices.
  - Examples: Udemy React Courses, Coursera React Courses, edX React Courses