**Lifecycle Methods**

In React, lifecycle methods are hooks that allow you to run code at specific points in a component's lifetime.

These methods are particularly useful for managing side effects, performing cleanup, and optimizing performance.

**Class Component Lifecycle Methods**

For class components, lifecycle methods are categorized into **three main phases**: **Mounting**, **Updating**, and **Unmounting**.

**1. Mounting (Component Creation)**

* **constructor(props)**:
  + Called when a component is initialized.
  + Used for setting initial state and binding event handlers.

constructor(props) {

super(props);

this.state = { count: 0 };

}

* **static getDerivedStateFromProps(props, state)**:
  + Called right before rendering when new props are received.
  + Used to update state based on props.
  + Returns an object to update the state or null to do nothing.

static getDerivedStateFromProps(nextProps, prevState) {

if (nextProps.value !== prevState.value) {

return { value: nextProps.value };

}

return null;

}

* **render()**:
  + Required method that returns the JSX to render.
  + Called during the mounting phase and every time the component updates.

render() {

return <div>{this.state.count}</div>;

}

* **componentDidMount()**:
  + Called immediately after the component is mounted.
  + Ideal for initiating network requests, setting up subscriptions, or performing DOM manipulations.

componentDidMount() {

fetch('/api/data')

.then(response => response.json())

.then(data => this.setState({ data }));

}

**2. Updating (Component Updates)**

* **static getDerivedStateFromProps(props, state)**:
  + Also called during the updating phase.
* **shouldComponentUpdate(nextProps, nextState)**:
  + Determines whether the component should re-render.
  + Can optimize performance by returning false if the update is unnecessary.

shouldComponentUpdate(nextProps, nextState) {

return nextProps.value !== this.props.value;

}

* **render()**:
  + Called again if shouldComponentUpdate returns true.
* **getSnapshotBeforeUpdate(prevProps, prevState)**:
  + Called right before the DOM is updated.
  + Useful for capturing information from the DOM (e.g., scroll position) before changes are applied.
  + Returns a value that is passed to componentDidUpdate.

getSnapshotBeforeUpdate(prevProps, prevState) {

if (prevState.items.length < this.state.items.length) {

return this.listRef.scrollHeight;

}

return null;

}

* **componentDidUpdate(prevProps, prevState, snapshot)**:
  + Called immediately after updating occurs.
  + Useful for performing actions after the DOM is updated.
  + snapshot is the value returned from getSnapshotBeforeUpdate.
  + Example:

componentDidUpdate(prevProps, prevState, snapshot) {

if (snapshot !== null) {

this.listRef.scrollTop = this.listRef.scrollHeight - snapshot;

}

}

**3. Unmounting (Component Removal)**

* **componentWillUnmount()**:
  + Called immediately before the component is removed from the DOM.
  + Ideal for cleanup tasks like invalidating timers, canceling network requests, or cleaning up subscriptions.

componentWillUnmount() {

clearInterval(this.timerID);

}

**Functional Components with Hooks**

For functional components, **lifecycle methods are replaced by hooks.**

Here's how you can achieve similar behavior with hooks:

* **useEffect()**:
  + Handles side effects in functional components.
  + You can specify when to run effects (on mount, update, or unmount) by passing dependencies.

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

function Example() {

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

useEffect(() => {

// Effect runs after every render

document.title = `You clicked ${count} times`;

// Cleanup function (runs on unmount)

return () => {

console.log('Cleaning up...');

};

}, [count]); // Dependency array, effect runs when 'count' changes

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>Click me</button>

</div>

);

}

export default Example;

**Summary**

* **Mounting**: Initialize and set up resources.
* **Updating**: Respond to changes in props or state and manage re-rendering.
* **Unmounting**: Clean up resources before removal.

Class components use specific lifecycle methods, while functional components use the useEffect hook to manage side effects and lifecycle events.

**Example 01**

Let's create a more comprehensive React application to demonstrate lifecycle methods. We'll build a **User Management Dashboard** with class components, featuring various lifecycle method scenarios.

**User Management Dashboard**

**Features:**

1. **UserList**: Displays a list of users.
2. **UserDetails**: Shows detailed information about a selected user.
3. **AddUserForm**: Form to add a new user.
4. **UpdateUserForm**: Form to update user information.
5. **Header**: Displays the application header.
6. **Footer**: Displays footer information.

**Project Structure**

/user-management

/components

Header.js

UserList.js

UserCard.js

UserDetails.js

AddUserForm.js

UpdateUserForm.js

Footer.js

/styles

Header.css

UserList.css

UserCard.css

UserDetails.css

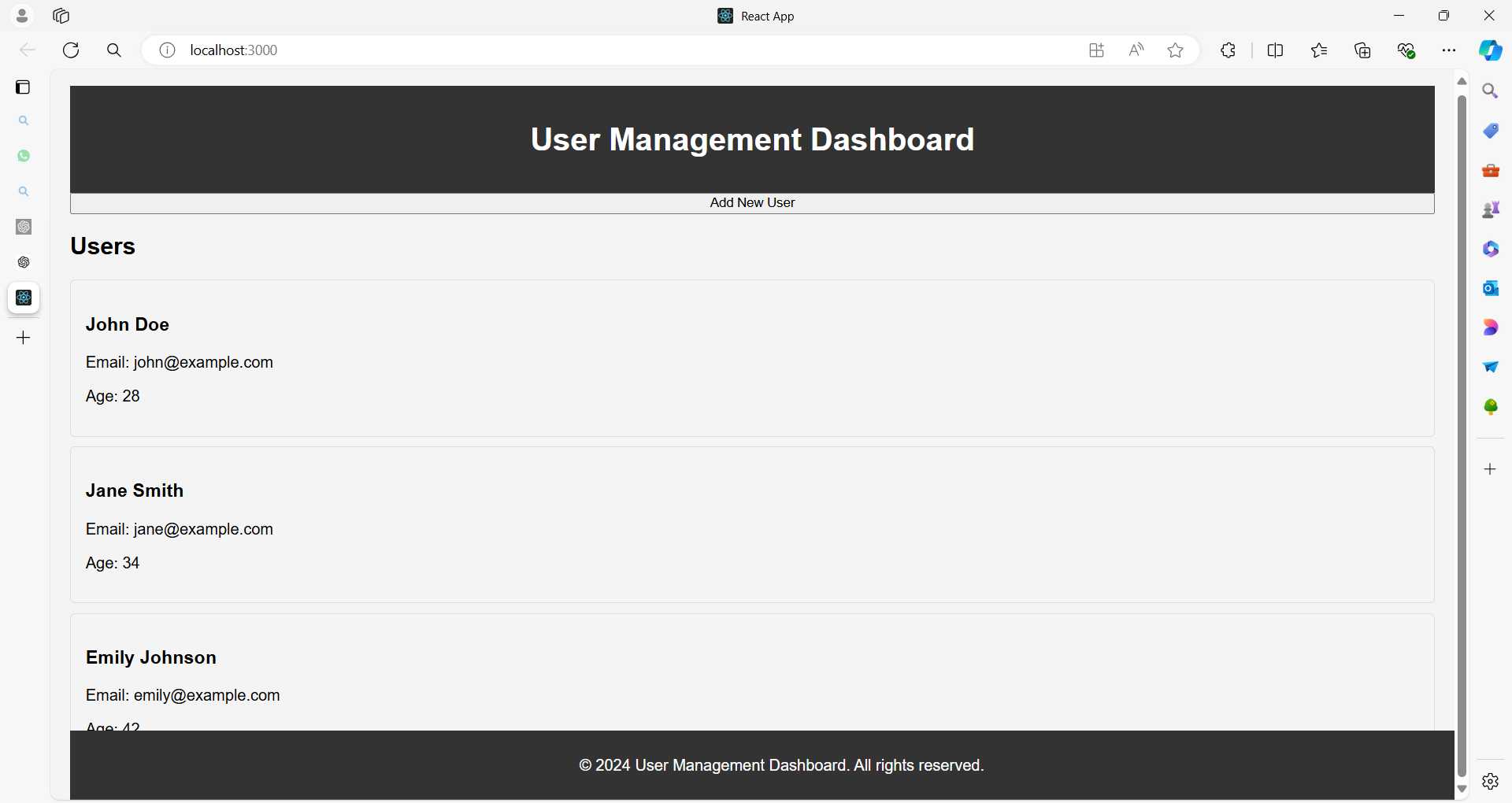
AddUserForm.css

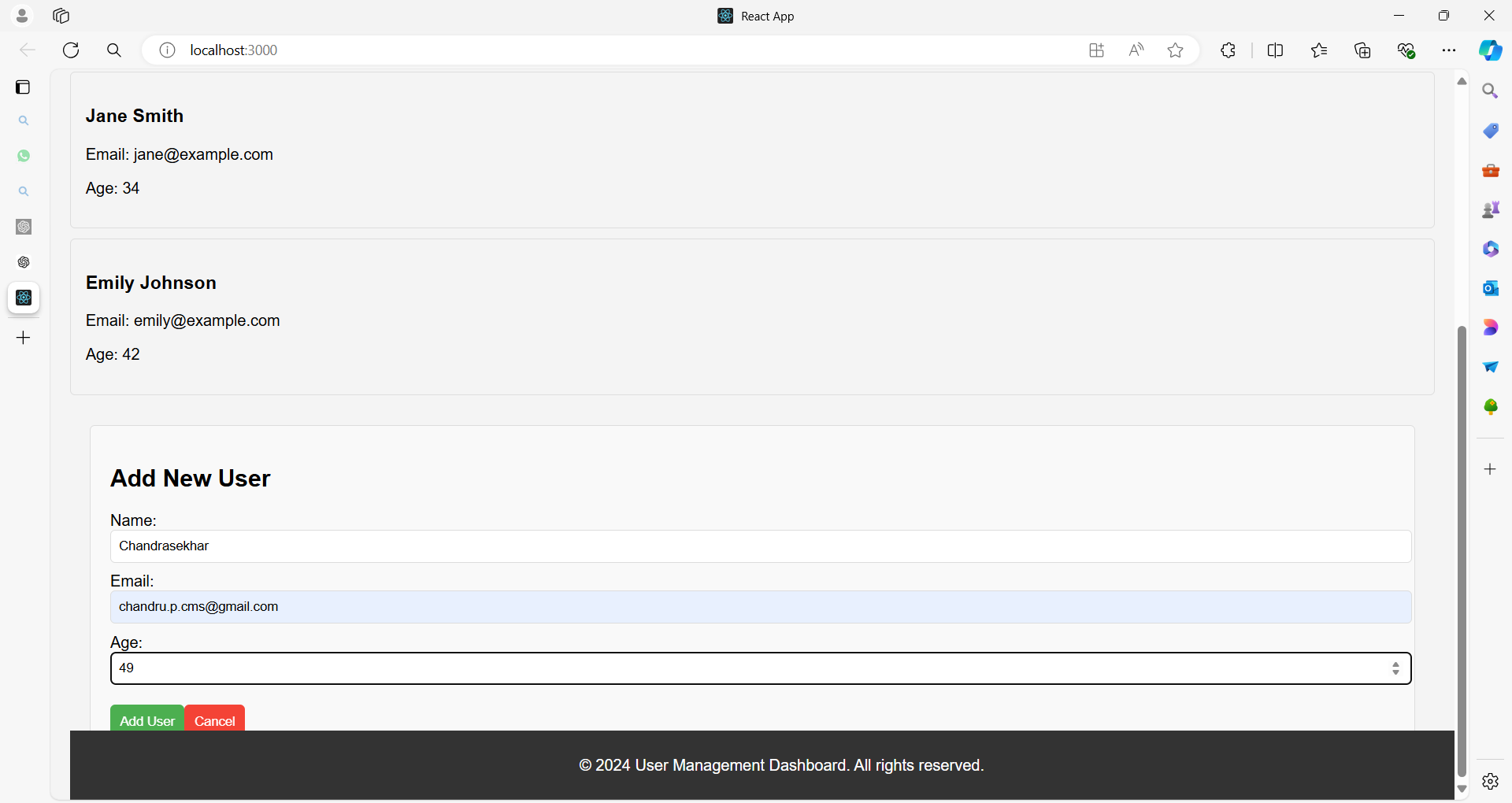
UpdateUserForm.css

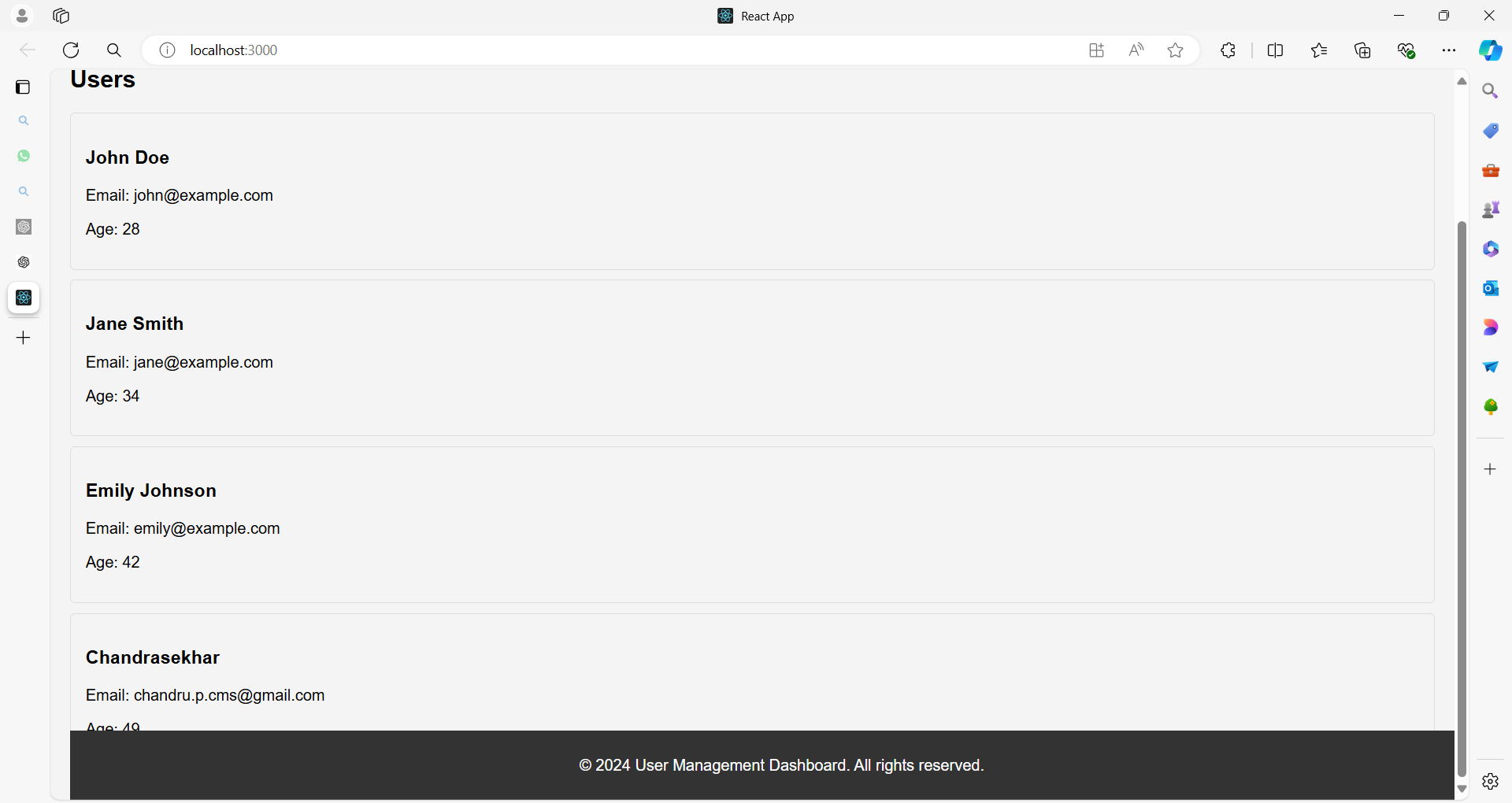
Footer.css

App.js

index.js







**1. App.js**

import React, { Component } from 'react';

import Header from './components/Header';

import UserList from './components/UserList';

import UserDetails from './components/UserDetails';

import AddUserForm from './components/AddUserForm';

import UpdateUserForm from './components/UpdateUserForm';

import Footer from './components/Footer';

import './styles/App.css'; // Global styles

// Sample user data

const initialUsers = [

{ id: 1, name: 'John Doe', email: 'john@example.com', age: 28 },

{ id: 2, name: 'Jane Smith', email: 'jane@example.com', age: 34 },

{ id: 3, name: 'Emily Johnson', email: 'emily@example.com', age: 42 },

// More users...

];

class App extends Component {

constructor(props) {

super(props);

this.state = {

users: initialUsers,

selectedUser: null,

editingUser: null,

showAddUserForm: false,

showUpdateUserForm: false,

};

}

componentDidMount() {

// Simulate a network request to fetch users

console.log('Component mounted and users fetched');

}

componentDidUpdate(prevProps, prevState) {

// Perform side effects if needed when state changes

if (prevState.users !== this.state.users) {

console.log('User list updated');

}

}

componentWillUnmount() {

// Cleanup before the component is removed

console.log('Component will unmount, perform cleanup');

}

handleUserSelect = (user) => {

this.setState({ selectedUser: user, showUpdateUserForm: false });

};

handleAddUser = (newUser) => {

this.setState((prevState) => ({

users: [...prevState.users, newUser],

showAddUserForm: false,

}));

};

handleUpdateUser = (updatedUser) => {

this.setState((prevState) => ({

users: prevState.users.map(user =>

user.id === updatedUser.id ? updatedUser : user

),

showUpdateUserForm: false,

}));

};

handleShowAddUserForm = () => {

this.setState({ showAddUserForm: true, selectedUser: null });

};

handleShowUpdateUserForm = () => {

this.setState({ showUpdateUserForm: true });

};

handleCloseForms = () => {

this.setState({ showAddUserForm: false, showUpdateUserForm: false });

};

render() {

const { users, selectedUser, showAddUserForm, showUpdateUserForm } = this.state;

return (

<div className="app">

<Header />

<button onClick={this.handleShowAddUserForm}>Add New User</button>

<UserList users={users} onUserSelect={this.handleUserSelect} />

{selectedUser && (

<UserDetails

user={selectedUser}

onEdit={this.handleShowUpdateUserForm}

/>

)}

{showAddUserForm && (

<AddUserForm

onAddUser={this.handleAddUser}

onClose={this.handleCloseForms}

/>

)}

{showUpdateUserForm && selectedUser && (

<UpdateUserForm

user={selectedUser}

onUpdateUser={this.handleUpdateUser}

onClose={this.handleCloseForms}

/>

)}

<Footer />

</div>

);

}

}

export default App;

**2. Header.js**

import React from 'react';

import '../styles/Header.css';

function Header() {

return (

<header className="header">

<h1>User Management Dashboard</h1>

</header>

);

}

export default Header;

**Header.css**

.header {

background-color: #333;

color: white;

padding: 15px;

text-align: center;

}

**3. UserList.js**

import React from 'react';

import UserCard from './UserCard';

import '../styles/UserList.css';

function UserList({ users, onUserSelect }) {

return (

<div className="user-list">

<h2>Users</h2>

{users.map(user => (

<UserCard

key={user.id}

user={user}

onClick={() => onUserSelect(user)}

/>

))}

</div>

);

}

export default UserList;

**UserList.css**

.user-list {

display: flex;

flex-direction: column;

padding: 20px;

}

**4. UserCard.js**

import React from 'react';

import '../styles/UserCard.css';

function UserCard({ user, onClick }) {

return (

<div className="user-card" onClick={onClick}>

<h3>{user.name}</h3>

<p>Email: {user.email}</p>

<p>Age: {user.age}</p>

</div>

);

}

export default UserCard;

**UserCard.css**

.user-card {

border: 1px solid #ddd;

border-radius: 5px;

padding: 15px;

margin: 10px 0;

cursor: pointer;

transition: background-color 0.3s;

}

.user-card:hover {

background-color: #f4f4f4;

}

**5. UserDetails.js**

import React from 'react';

import '../styles/UserDetails.css';

function UserDetails({ user, onEdit }) {

return (

<div className="user-details">

<button className="edit-btn" onClick={onEdit}>Edit</button>

<h2>{user.name}</h2>

<p>Email: {user.email}</p>

<p>Age: {user.age}</p>

</div>

);

}

export default UserDetails;

**UserDetails.css**

.user-details {

padding: 20px;

border: 1px solid #ddd;

border-radius: 5px;

margin: 20px;

}

.edit-btn {

background-color: #4CAF50;

color: white;

border: none;

padding: 10px;

cursor: pointer;

margin-bottom: 15px;

}

**6. AddUserForm.js**

import React, { Component } from 'react';

import '../styles/AddUserForm.css';

class AddUserForm extends Component {

constructor(props) {

super(props);

this.state = {

name: '',

email: '',

age: ''

};

}

handleChange = (e) => {

this.setState({ [e.target.name]: e.target.value });

};

handleSubmit = (e) => {

e.preventDefault();

const newUser = {

id: Date.now(), // Use current timestamp as unique ID

...this.state

};

this.props.onAddUser(newUser);

};

render() {

return (

<div className="add-user-form">

<h2>Add New User</h2>

<form onSubmit={this.handleSubmit}>

<label>

Name:

<input

type="text"

name="name"

value={this.state.name}

onChange={this.handleChange}

required

/>

</label>

<label>

Email:

<input

type="email"

name="email"

value={this.state.email}

onChange={this.handleChange}

required

/>

</label>

<label>

Age:

<input

type="number"

name="age"

value={this.state.age}

onChange={this.handleChange}

required

/>

</label>

<button type="submit">Add User</button>

<button type="button" onClick={this.props.onClose}>Cancel</button>

</form>

</div>

);

}

}

export default AddUserForm;

**AddUserForm.css**

.add-user-form {

padding: 20px;

border: 1px solid #ddd;

border-radius: 5px;

margin: 20px;

background-color: #f9f9f9;

}

.add-user-form label {

display: block;

margin-bottom: 10px;

}

.add-user-form input {

width: 100%;

padding: 8px;

border: 1px solid #ddd;

border-radius: 5px;

}

.add-user-form button {

margin-top: 10px;

padding: 10px;

border: none;

border-radius: 5px;

cursor: pointer;

}

.add-user-form button[type="submit"] {

background-color: #4CAF50;

color: white;

}

.add-user-form button[type="button"] {

background-color: #f44336;

color: white;

}

**7. UpdateUserForm.js**

import React, { Component } from 'react';

import '../styles/UpdateUserForm.css';

class UpdateUserForm extends Component {

constructor(props) {

super(props);

this.state = {

...props.user

};

}

handleChange = (e) => {

this.setState({ [e.target.name]: e.target.value });

};

handleSubmit = (e) => {

e.preventDefault();

this.props.onUpdateUser(this.state);

};

render() {

return (

<div className="update-user-form">

<h2>Update User</h2>

<form onSubmit={this.handleSubmit}>

<label>

Name:

<input

type="text"

name="name"

value={this.state.name}

onChange={this.handleChange}

required

/>

</label>

<label>

Email:

<input

type="email"

name="email"

value={this.state.email}

onChange={this.handleChange}

required

/>

</label>

<label>

Age:

<input

type="number"

name="age"

value={this.state.age}

onChange={this.handleChange}

required

/>

</label>

<button type="submit">Update User</button>

<button type="button" onClick={this.props.onClose}>Cancel</button>

</form>

</div>

);

}

}

export default UpdateUserForm;

**UpdateUserForm.css**

.update-user-form {

padding: 20px;

border: 1px solid #ddd;

border-radius: 5px;

margin: 20px;

background-color: #f9f9f9;

}

.update-user-form label {

display: block;

margin-bottom: 10px;

}

.update-user-form input {

width: 100%;

padding: 8px;

border: 1px solid #ddd;

border-radius: 5px;

}

.update-user-form button {

margin-top: 10px;

padding: 10px;

border: none;

border-radius: 5px;

cursor: pointer;

}

.update-user-form button[type="submit"] {

background-color: #4CAF50;

color: white;

}

.update-user-form button[type="button"] {

background-color: #f44336;

color: white;

}

**8. Footer.js**

import React from 'react';

import '../styles/Footer.css';

function Footer() {

return (

<footer className="footer">

<p>&copy; 2024 User Management Dashboard. All rights reserved.</p>

</footer>

);

}

export default Footer;

**Footer.css**

.footer {

  background-color: #333;

  color: #fff;

  text-align: center;

  padding: 10px 20px;

  position: fixed;

  bottom: 0;

  width: 100%;

  box-shadow: 0 -2px 5px rgba(0, 0, 0, 0.2);

  font-size: 14px;

}

.footer p {

  margin: 0;

}

**9. index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));

**Summary**

This example demonstrates:

* **Mounting**: componentDidMount simulates fetching data.
* **Updating**: componentDidUpdate logs updates to the user list.
* **Unmounting**: componentWillUnmount handles cleanup.

Each component illustrates different aspects of lifecycle methods in class components, from initializing and updating state to managing side effects and cleanup.

This setup also demonstrates how to structure a React application with multiple components and CSS styling.