Certainly! Let's dive into two more detailed examples, one focusing on state and the other on props.

### Example 1: State

In this example, we'll create a simple todo list application. Users can add new todos, mark them as completed, and delete them. We'll manage the list of todos using component state.

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

const TodoList = () => {
  const [todos, setTodos] = useState([]);
  const [newTodo, setNewTodo] = useState('');

  const addTodo = () => {
    if (newTodo.trim() !== '') {
      setTodos([...todos, { text: newTodo, completed: false }]);
      setNewTodo('');
    }
  };

  const toggleTodo = (index) => {
    const newTodos = [...todos];
    newTodos[index].completed = !newTodos[index].completed;
    setTodos(newTodos);
  };

  const deleteTodo = (index) => {
    const newTodos = todos.filter((_, i) => i !== index);
    setTodos(newTodos);
  };

  return (
    <div>
      <input
        type="text"
        value={newTodo}
        onChange={(e) => setNewTodo(e.target.value)}
        placeholder="Enter new todo"
      />
      <button onClick={addTodo}>Add Todo</button>
      <ul>
        {todos.map((todo, index) => (
```

```
          <li key={index}>
            <span
              style={{ textDecoration: todo.completed ? 'line-through' : 'none' }}
              onClick={() => toggleTodo(index)}
            >
              {todo.text}
            </span>
            <button onClick={() => deleteTodo(index)}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
};

export default TodoList;
```

In this example:

- We use the `useState` hook to manage two pieces of state: `todos` for storing the list of todos and `newTodo` for storing the text of the new todo being entered.
- The `addTodo` function adds a new todo to the list when the "Add Todo" button is clicked.
- The `toggleTodo` function toggles the completion status of a todo when it is clicked.
- The `deleteTodo` function removes a todo from the list when the "Delete" button is clicked.

### Example 2: Props

In this example, we'll create a reusable `Button` component that accepts props to customize its appearance and behavior.

```jsx
import React from 'react';

const Button = ({ onClick, disabled, color, children }) => {
  return (
    <button onClick={onClick} disabled={disabled} style={{ backgroundColor: color }}>
      {children}
    </button>
  );
};

export default Button;
```

In this example:

- The `Button` component accepts props such as `onClick` for handling click events, `disabled` to disable the button, `color` to set the background color, and `children` to render the button label.
- These props allow the `Button` component to be customized and reused in various parts of the application.

These detailed examples demonstrate how to effectively use state and props in React components to manage dynamic data and build reusable UI components.