Here's a simple example of a React.js login and logout system using functional components, state management via `useState`, and basic conditional rendering.

## 1. Create a new React project:

```
cmd(terminal):-
npx create-react-app login-logout-example
cd login-logout-example
npm start
```

## 2. Update `App.js` with the following code:

Jsx code:-

```
import React, { useState } from 'react';
import './App.css';

function App() {
  // State to track if the user is logged in
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  // State for handling user credentials (just for demo purposes)
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');

  // Handle login
  const handleLogin = (e) => {
    e.preventDefault();

    // In a real-world app, you'd validate the username and password
    if (username === 'user' && password === 'password') {
      setIsLoggedIn(true);
    } else {
      alert('Invalid credentials');
    }
  };

  // Handle logout
  const handleLogout = () => {
    setIsLoggedIn(false);
    setUsername('');
    setPassword('');
  };

  return (
    <div className="App">
      <h1>Login and Logout Example</h1>

      {/* Show login form if user is not logged in */}
      {!isLoggedIn ? (
        <div>
          <h2>Please Log In</h2>
          <form onSubmit={handleLogin}>
            <div>
              <label htmlFor="username">Username: </label>
              <input
                type="text"
```

```
                id="username"
                value={username}
                onChange={(e) => setUsername(e.target.value)}
                required
              />
            </div>
            <div>
              <label htmlFor="password">Password: </label>
              <input
                type="password"
                id="password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                required
              />
            </div>
            <button type="submit">Log In</button>
          </form>
        </div>
      ) : (
        // Show logged-in UI
        <div>
          <h2>Welcome, {username}!</h2>
          <button onClick={handleLogout}>Log Out</button>
        </div>
      )}
    </div>
  );
}

export default App;
```

## Explanation:

1. **State Management**:
   - `isLoggedIn`: Tracks whether the user is logged in or not.
   - `username` and `password`: Store the user input for the login form.
2. **Login Logic**:
   - When the login form is submitted, the `handleLogin` function checks if the username and password are correct (for simplicity, it's hardcoded as "user" and "password").
   - If the credentials are valid, `setIsLoggedIn(true)` logs the user in.
3. **Logout Logic**:
   - The `handleLogout` function resets the login state, effectively logging the user out.
4. **Conditional Rendering**:
   - If the user is logged in (`isLoggedIn` is `true`), show a welcome message with a "Log Out" button.
   - If the user is not logged in, display the login form.

### 3. Styles (optional):

You can add some basic styling in `App.css` for better UI:

`css code`:-

```
.App {
  text-align: center;
  margin-top: 50px;
}

form {
  display: inline-block;
  text-align: left;
  padding: 20px;
  border: 1px solid #ccc;
  border-radius: 5px;
}

input {
  margin-bottom: 10px;
  padding: 8px;
  width: 200px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

button {
  padding: 10px 20px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

button:hover {
  background-color: #0056b3;
}
```

### 4. Run the application:

After making these changes, run your app using:

`cmd(terminal):-`

```
npm start
```

This will start the development server, and you should see the login page. Upon successful login, the page will display a welcome message and a logout button.