

INTERNSHIP REPORT

WEEK 6 DAY 4

Submitted to:	Ali Hyder	Submission Date:	31 st July 29, 2025
Internship Domain:	Front Development	Internship Name:	ProSensia
Student Name:	Yasal Qamar	Roll No.	S25031

React.js Basics

Topics: Conditional Rendering, Lists with .map()

Objective

To learn how to conditionally render content in React components and how to dynamically generate list items using the .map() function.

Topics Covered:

- Conditional Rendering in React
- Rendering Lists using .map()

What is React?

React is a **JavaScript library** used to build **user interfaces (UI)**, especially for **web applications**. It helps you create **dynamic, reusable components** that can change based on data without needing to reload the page.

Why Use React?

- **Component-Based:** Break your UI into small, reusable pieces (like buttons, forms, headers).
- **Fast and Efficient:** Uses a virtual DOM to update only the changed parts of the UI.
- **One-Way Data Flow:** Data flows in one direction, making the app easier to debug and manage.
- **Reusable JSX Code:** Write HTML-like code inside JavaScript using JSX.

1. Conditional Rendering

- Conditional rendering in React lets us show or hide UI elements based on application state or logic. We explored different approaches such as:
- Using simple if statements inside functions
- Using ternary operators in JSX
- Using logical && operators for short conditional rendering

```
function Greeting(props) {  
  
  return (  
  
    <div>  
  
      {props.isLoggedIn ? <h1>Welcome Back!</h1> : <h1>Please log in.</h1>}  
  
    </div>  
  
  );  
  
}
```

2. Rendering Lists with .map()

We used the `.map()` method to render a list of elements from an array. This is a common pattern in React apps for displaying dynamic data.

Example:

```
const fruits = ["Apple", "Banana", "Mango"];  
  
function FruitList() {  
  
  return (  
  
    <ul>  
  
      {fruits.map((fruit, index) => (  
  
        <li key={index}>{fruit}</li>  
  
      ))}  
  
    </ul>  
  
  );  
  
}
```

- We also learned the importance of using the key prop to help React identify which items have changed, are added, or are removed.

CODING:

.app.js

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

function CardGame() {
  const [cards, setCards] = useState([]);
  const [total, setTotal] = useState(0);

  const cardDeck = [
    { name: 'King', value: 13 },
    { name: 'Queen', value: 12 },
    { name: 'Jack', value: 11 },
    { name: 'Ace', value: 1 },
  ];

  function drawCard() {
    const randomCard = cardDeck[Math.floor(Math.random() * cardDeck.length)];
    setCards(prev => [...prev, randomCard]);
    setTotal(prev => prev + randomCard.value);
  }

  return (
    <div style={{ textAlign: 'center', padding: '20px', fontFamily: 'Arial' }}>
      <h2>🎴 Card Game: King, Queen, Jack, Ace</h2>
      <button onClick={drawCard} style={{ padding: '10px 20px', fontSize: '16px' }}>
        Draw Card
      </button>
      <h3>Drawn Cards:</h3>
      <ul>
        {cards.map((card, index) => (
          <li key={index}>{card.name} (Value: {card.value})</li>
        ))}
      </ul>
      <h3>Total Score: {total}</h3>
    </div>
  );
}
```

```
export default CardGame;
```

.index.css

```
body {
  margin: 0;
  padding: 0;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background: linear-gradient(135deg, #1e3c72, #2a5298);
  color: white;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}

h1 {
  font-size: 3rem;
  margin-bottom: 10px;
  text-shadow: 2px 2px 5px rgba(0,0,0,0.5);
}

button {
  padding: 12px 25px;
  font-size: 18px;
  border: none;
  border-radius: 10px;
  background: #ff4e50;
  color: white;
  cursor: pointer;
  box-shadow: 0 4px 15px rgba(255, 78, 80, 0.6);
  transition: 0.3s ease;
}

button:hover {
  background: #fc913a;
  transform: scale(1.05);
}

.card-box {
  margin-top: 30px;
  background: rgba(255, 255, 255, 0.1);
  padding: 20px 30px;
```

```

border-radius: 15px;
backdrop-filter: blur(10px);
box-shadow: 0 0 25px rgba(255, 255, 255, 0.2);
width: 90%;
max-width: 500px;
}

ul {
  list-style-type: none;
  padding: 0;
  margin-top: 15px;
}

li {
  font-size: 20px;
  background: rgba(255, 255, 255, 0.15);
  padding: 10px 15px;
  margin-bottom: 10px;
  border-radius: 8px;
  text-shadow: 1px 1px 3px rgba(0,0,0,0.4);
  box-shadow: inset 0 0 8px rgba(255, 255, 255, 0.2);
}

h2 {
  margin: 10px 0;
}

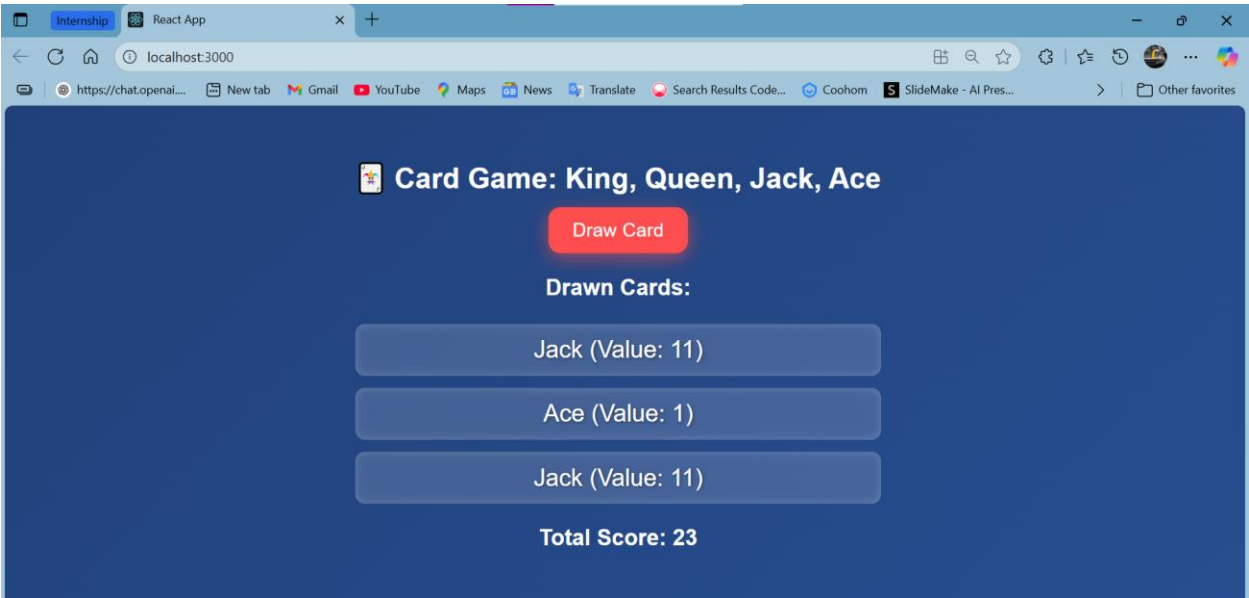
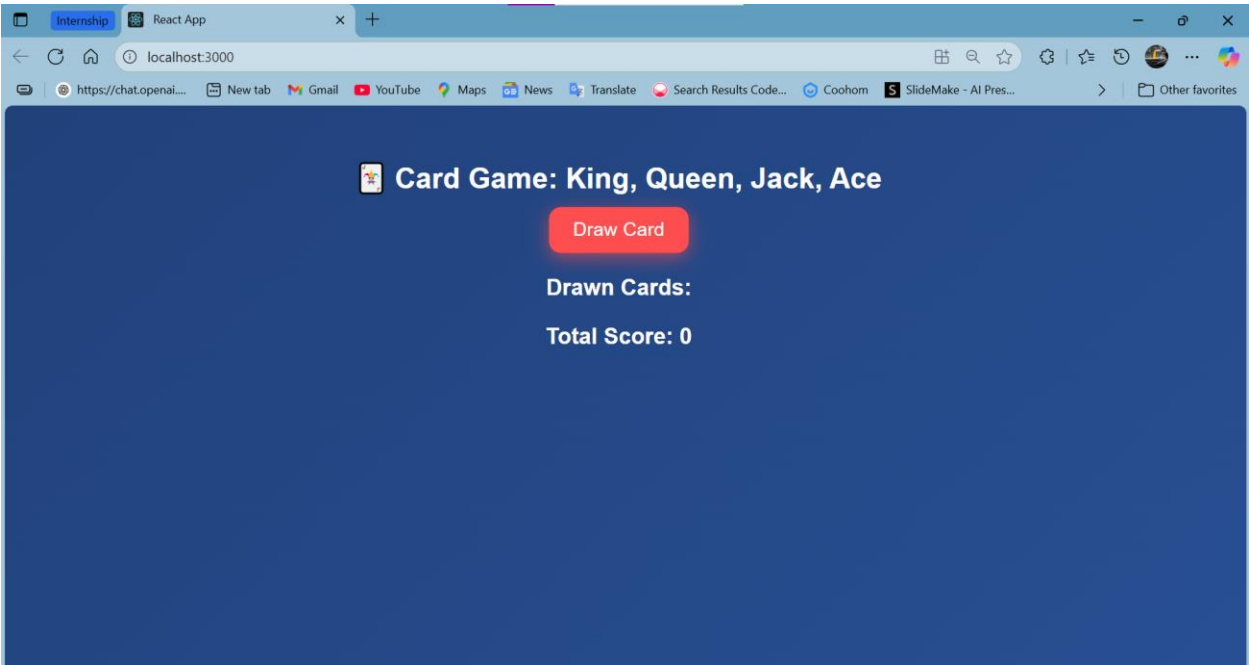
#totalScore {
  font-size: 24px;
  font-weight: bold;
  color: #ffd700;
  text-shadow: 1px 1px 4px black;
}


```




Features:

- Randomly draw a card on button click
- Cards: King, Queen, Jack, Ace
- Each card has a score:
 - King = 13
 - Queen = 12
 - Jack = 11
 - Ace = 1

- Display all drawn cards
- Show total score



Step	Action
 1	A random card is selected (King, Queen, Jack, Ace)

 2	That card is displayed in the list of drawn cards
 3	Its value is added to the running total score
 4	The updated score is shown on the screen

CONCUSLION:

Today's task significantly enhanced my understanding of React's core features, particularly **state management using useState** and **event-driven UI updates**. Through hands-on practice with conditional rendering and dynamic list rendering using **.map()**, I learned how to build responsive and interactive components.

Implementing the card game project allowed me to apply these concepts in a practical scenario updating UI elements based on user actions and maintaining cumulative state across renders. I also gained deeper insight into **React's rendering logic**, component composition, and the importance of using unique key props when rendering lists.

Overall, this session reinforced essential frontend development skills and increased my confidence in using React to build modern, dynamic web applications.
