skillPrompt

Presentation 2024

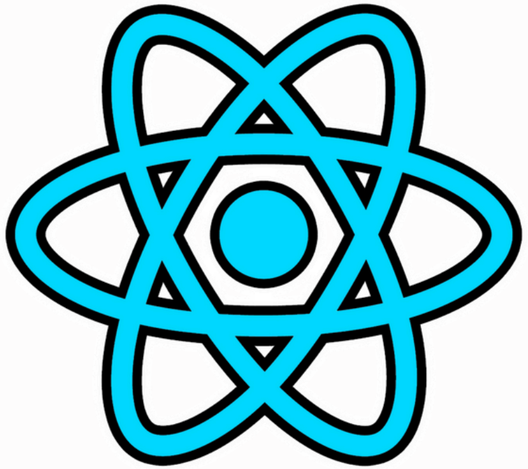
REACT JS

Revolutionizing Frontend Development

Presented by:
Sujal Gaha Magar
Frontend Developer

React JS

- A JavaScript library for building User Interfaces (UI).
- Created by META (formerly Facebook).
- Widely Adopted in the industry.
- Examples:
    - Facebook, Github, Whatsapp web, Discord, etc.



Why use React?

Lets see a comparison between Basic JS and React JS.

Challenges of basic JS

01 Performance Issue

Updates the UI inefficiently

Benefits of React JS

01 Virtual DOM

Improve efficiency of your app with the help of virtual dom.

01. Performance Issue

Lets understand how Traditional DOM works.

time.js

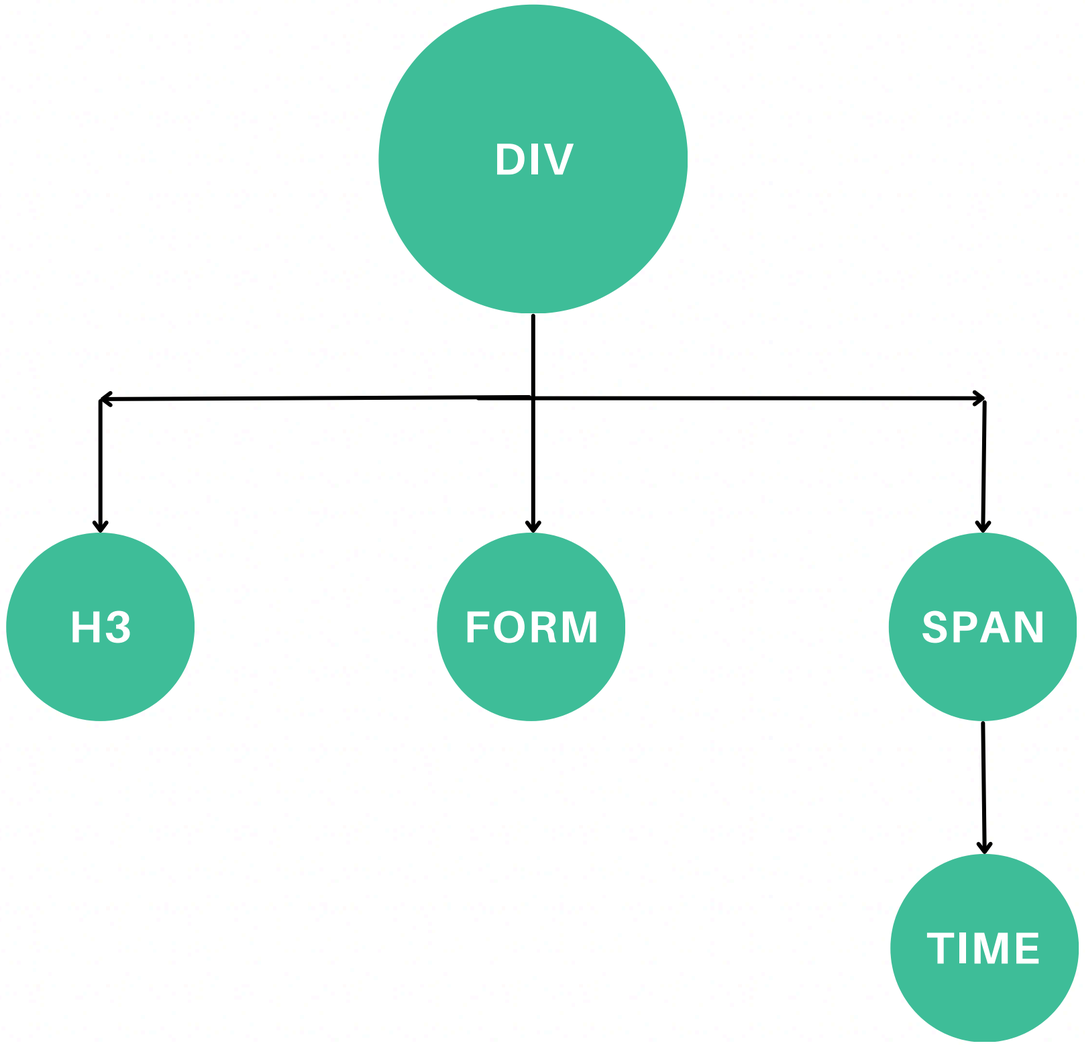
time.js > ...

```javascript
const update = () => {
  const element = `
      <h3>JavaScript:</h3>
      <form>
       <input type="text"/>
      </form>
      <span>Time: ${new Date().toLocaleTimeString()}</span>
    `;

  document.getElementById("root1").innerHTML = element;
};

setInterval(update, 1000);
```

```
DOCTYPE: html
HTML
  HEAD
  BODY
    DIV id="root1"
      #text:
      H3
        #text: JavaScript:
      #text:
      FORM
        #text:
        INPUT type="text"
        #text:
      #text:
      SPAN
        #text: Time: 19:34:57
      #text:
```



DOM Structure

JavaScript:

Time: 3:48:50 AM

Search HTML

```
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <div id="root1">
      <h3>JavaScript:</h3>
      <form> ... </form>
      <span>Time: 3:48:50 AM</span>
```

Here's what happened!

- The **setInterval()** function call the **update()** function every `1` second.
- The **update()** function then renders the element inside it with the help of DOM.
- The "**new Date().toLocaleTimeString()**" generates the current time.
- When something is entered in the input field, it resets after `1` second.
- This happens because:
  - The **update()** function is called every `1` second.
  - On each function call, a new element is being set on the DOM.
  - Which causes a new input field with no value to re-render in the UI.
  - Causing performance issue.

01. Virtual DOM

How React handles this better

React:

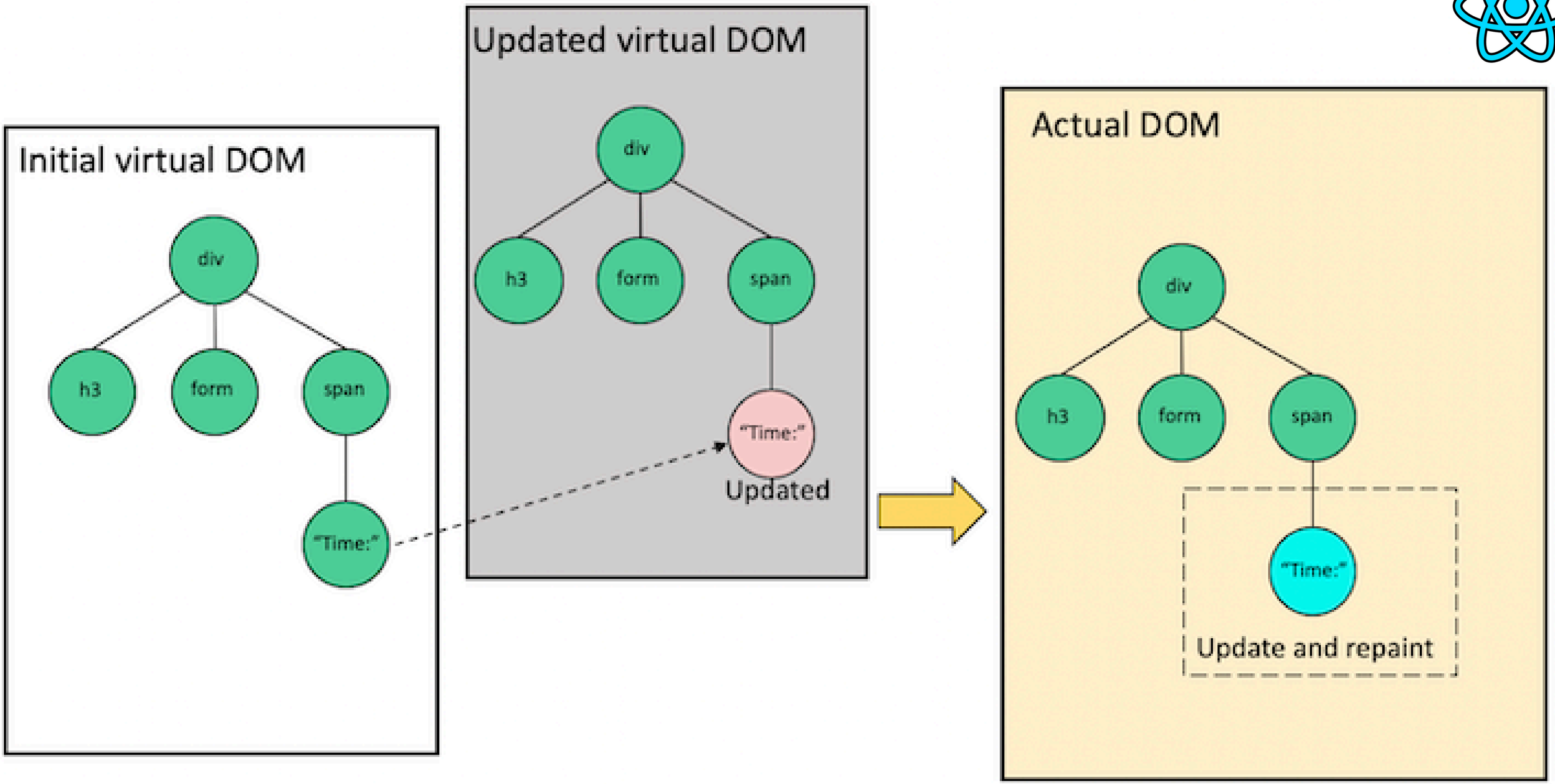
Hello

Time: 3:36:09 AM

Console Inspector

Search HTML

```
<div id="root"> event
  <h3>React:</h3>
  <form> ... </form>
  <span>
    Time:
    3:36:09 AM
  </span>
```


Initial virtual DOM
div
h3
form
span
"Time:"
Updated virtual DOM
div
h3
form
span
"Time:"
Updated
Actual DOM
div
h3
form
span
"Time:"
Update and repaint

Here's what happened

- React creates a **virtual DOM**.
- If any changes occurs then React **creates** a new snapshot of **virtual DOM**.
- Then, React **compares** the new snapshot with the previous snapshot.
- It compares by a diffing process called **Reconciliation**.
- Then it only **changes** the node (or element) which is **updated**.
- Without affecting any other nodes.

Challenges of basic HTML, CSS & JS

01 Performance Issue

Updates the UI inefficiently

02 Code Maintainability

Tedious and difficult to make reusable component

Benefits of React JS

01 Virtual DOM

Improve efficiency of your app with the help of virtual dom.

02 Re-usability of code

Make reusable code with component based approach

02. Code Maintainability

Why code maintainability in JS is tedious?

home.html - prompt - Visual Studio Code

File Edit Selection View Go Run Terminal Help

home.html | about.html

home.html > html > head

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, ini
    <title>Document</title>
    <link rel="stylesheet" href="./navbar.css" />
  </head>
  <body>
    <nav>
      <!-- The code for navbar in home page -->
    </nav>
    <main>
      <h1>Home Page</h1>
    </main>
    <script src="./navbar.js"></script>
  </body>
</html>
```

FOL...

- about.html
- contact.html
- home.html

0 0 0 | Ln 8, Col 10 | Spaces: 2 | UTF-8 | LF | HTML | Go Live | Prettier

Menu 2:31 PM

contact.html - prompt - Visual Studio Code

File Edit Selection View Go Run Terminal Help

home.html | contact.html

contact.html > html > head

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, ini
    <title>Document</title>
    <link rel="stylesheet" href="./navbar.css" />
  </head>
  <body>
    <nav>
      <!-- The same code for navbar in contact page -->
    </nav>
    <main>
      <h1>Contact us form</h1>
    </main>
    <script src="./navbar.js"></script>
  </body>
</html>
```

FOL...

- about.html
- contact.html
- home.html

Ln 8, Col 10 | Spaces: 2 | UTF-8 | LF | HTML | Go Live | Prettier

Menu 2:31 PM

about.html - prompt - Visual Studio Code

File Edit Selection View Go Run Terminal Help

home.html | about.html

about.html > html > body > script

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, ini
    <title>Document</title>
    <link rel="stylesheet" href="./navbar.css" />
  </head>
  <body>
    <nav>
      <!-- The same code for navbar in about page -->
    </nav>
    <main>
      <h1>About us</h1>
    </main>
    <script src="./navbar.js"></script>
  </body>
</html>
```

FOL...

- about.html
- contact.html
- home.html

0 0 0 | Ln 16, Col 40 | Spaces: 2 | UTF-8 | LF | HTML | Go Live | Prettier

Menu 2:31 PM

Here's what happened

- The HTML code for the **NavBar** is repeated in all .html files.
- It is done for the **NavBar** to be rendered on every page.
- The **NavBar** code also requires both the css and script code.
- Which also needs to be imported in every file containing the **NavBar** code.
- This violates the concept of **DRY** (Don't Repeat Yourself).
    - This might be a seem like a simple problem, but using this method becomes difficult to implement and manage in a large scale application.

02. Reusability of Code

How react helps make resuable code

Navbar.tsx - prompt-js - Visual Studio Code

File Edit Selection View Go Run Terminal Help

Navbar.tsx

src > components > Navbar.tsx > default

```
1  function Navbar() {
2    return (
3      <nav className="flex justify-around items-center py-6 bg-black tex
4        <h1 className="text-2xl">Skillprompt</h1>
5        <div className="flex gap-8">
6          <a href="">Home</a>
7          <a href="">About</a>
8          <a href="">Contact us</a>
9          <a href="">Blog</a>
10       </div>
11     </nav>
12   );
13 }
14
15 export default Navbar;
16
```

Ln 15, Col 23 Spaces: 2 UTF-8 LF TypeScript JSX Go Live Prettier

Menu 8:09 PM

App.tsx - prompt-js - Visual Studio Code

File Edit Selection View Go Run Terminal Help

Counter.tsx | App.tsx

src > App.tsx > ...

```
import "./App.css";
import Navbar from "./components/Navbar";

function HomePage() {
  return (
    <div>
      <Navbar />
      <main>
        <h1>Home Page Content</h1>
      </main>
    </div>
  );
}

export default HomePage;
```

0 0 0 | Ln 16, Col 1 | Spaces: 2 | UTF-8 | LF | TypeScript JSX | Go Live | Prettier

Menu 11:41 AM

Here’s what happened

- The code for **NavBar** is written in a different file.
- Then the **NavBar** component is exported.
- In the App.tsx file, The **NavBar** component is called.
- Now, that **NavBar** code is reusable and can be called at any page you want.

Challenges of basic HTML, CSS & JS

01 Performance Issue

Updates the UI inefficiently

02 Code Maintainability

Tedious and difficult to make reusable component

03 Event Handling and Data Binding

Complex for handling events and binding data.

Benefits of React JS

01 Virtual DOM

Improve efficiency of your app with the help of virtual dom.

02 Re-usability of code

Make reusable code with component based approach

03 Declarative Approach

React handles everything

03. Event Handling and Data Binding

count.html - prompt - Visual Studio Code

File Edit Selection View Go Run Terminal Help

count.js | count.html

eventHandling > count.html > ...

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Event Handling Example</title>
  </head>
  <body>
    <button id="incrementBtn">Click to Increment</button>
    <p>Count: <span id="count">0</span></p>

    <script src="./count.js"></script>
  </body>
</html>
```

0 0 0 | Ln 14, Col 1 | Spaces: 2 | UTF-8 | LF | HTML | Port : 5500 | Prettier

Menu 3:31 PM

count.js - prompt - Visual Studio Code

File Edit Selection View Go Run Terminal Help

JS count.js | count.html

eventHandling > JS count.js > ...

```javascript
let count = 0;

const countDisplay = document.getElementById("count");
const incrementBtn = document.getElementById("incrementBtn")

function incrementCount() {
  count++;
  countDisplay.innerText = count;
}

incrementBtn.addEventListener("click", incrementCount);
```

0 0 0 | Ln 12, Col 1 | Spaces: 4 | UTF-8 | LF | JavaScript | Port : 5500 | Prettier

Menu 3:32 PM

Here's what happened

- In the HTML code:
  - The **<button>** tag is given the **id="incrementBtn"**.
  - The **** tag is given the **id="count"**.

- In the JS code:
  - A **count** variable is declared and initialized with the value 0.
  - The **countDisplay** selects and stores the reference of the element with **id="count"**.
  - The **incrementBtn** selects and stores the reference of the element with **id="incrementBtn"**.
  - A function **incrementCount** is defined which is responsible for incrementing the count.
  - **incrementBtn.addEventListener("click", incrementCount**) adds a event listener to the **incrementBtn** element.
    - A **click** event listener is added.
    - On click event, the **incrementCount** function is invoked.

03. Declarative Approach

React handles everything for you.

Counter.tsx - prompt-js - Visual Studio Code

File Edit Selection View Go Run Terminal Help

Counter.tsx

src > components > Counter.tsx > default

```
import { useState } from "react";

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

  const incrementCount = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <button onClick={incrementCount}>Click to Increment</button>
      <p>Count: {count}</p>
    </div>
  );
};

export default EventHandlingExample;
```

0 0 0 Ln 18, Col 37 Spaces: 4 UTF-8 LF TypeScript JSX Go Live Prettier

Menu 8:19 PM

Here's what happened

- An **EventHandlingExample** component is created.
- A **useState** hook is used for declaring **count** variable.
  - Hooks are tools in React to manage states easily.
  - They also helps to make the code cleaner and easier to understand.
- A **incrementCount** function is defined which updates the **count** by `1`.
- Then it renders the UI part
- An **onClick** event handler is added in the button which invoke the **incrementCount**.

This is why React is better

skillPrompt

Any Queries ?

THANK YOU