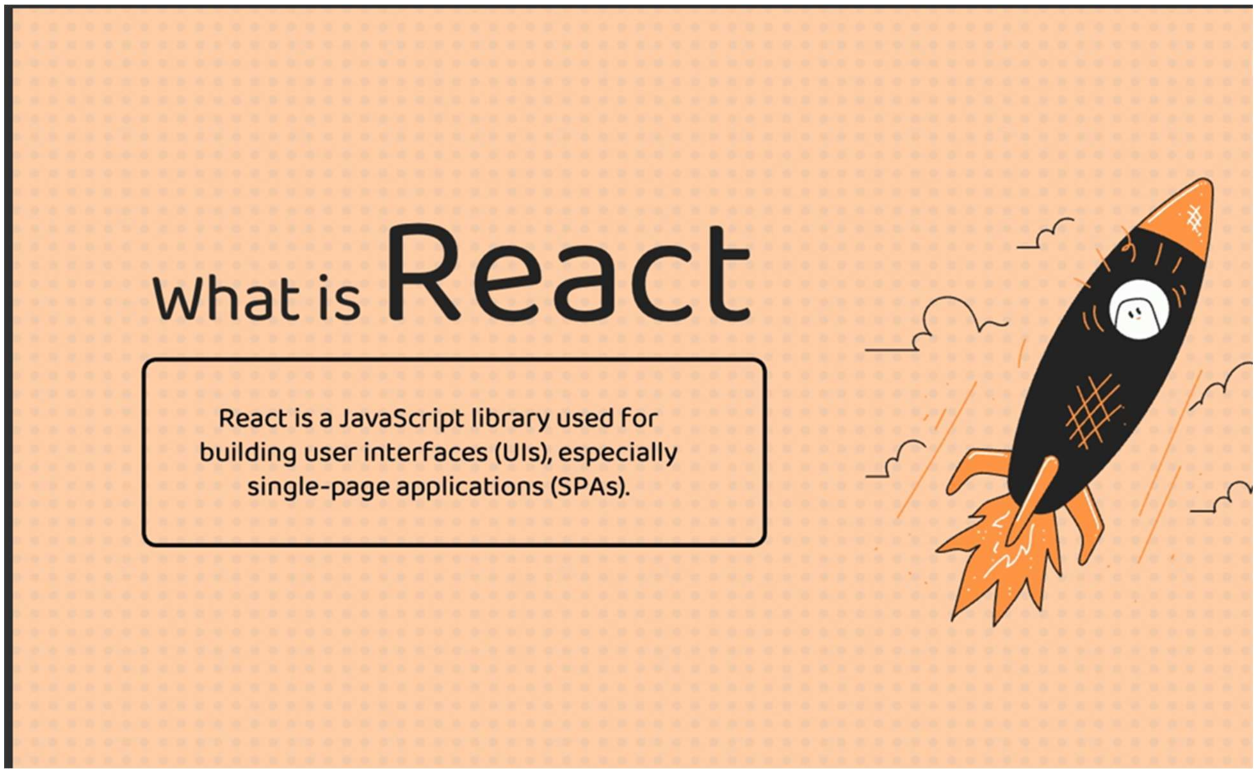
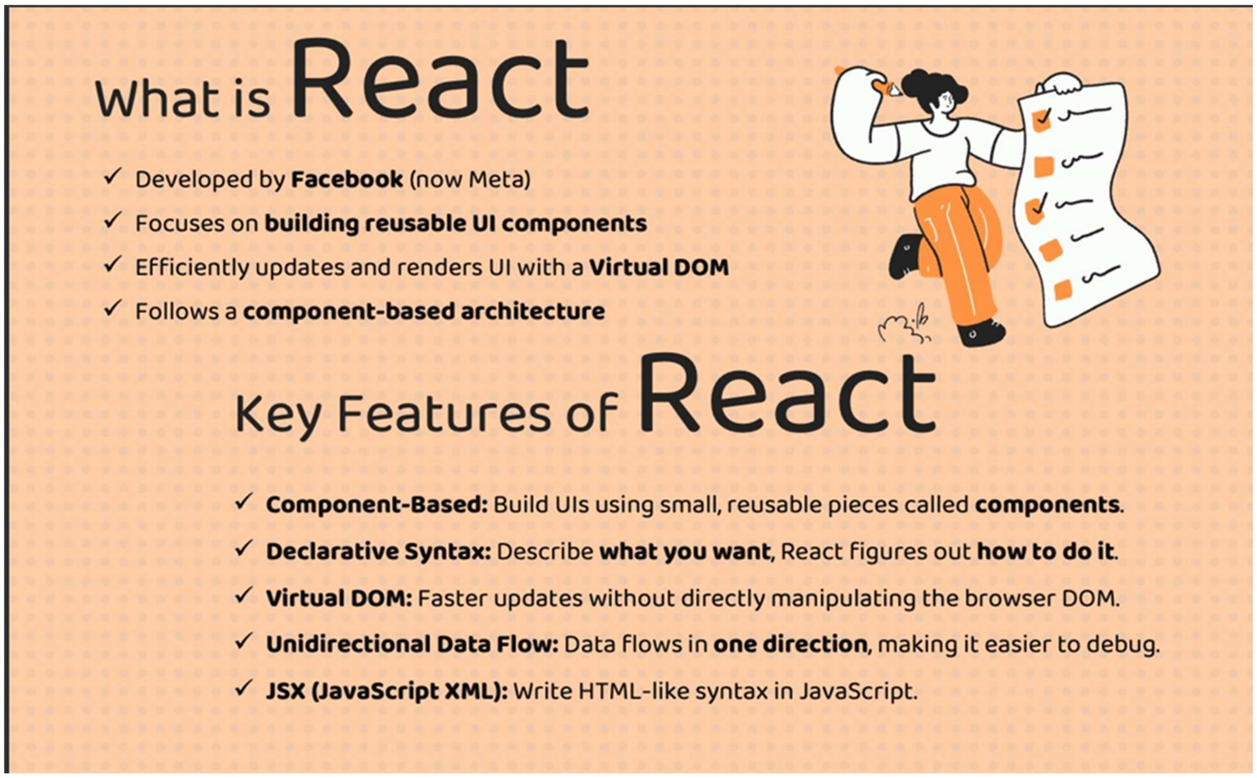


What is React

React is a JavaScript library used for building user interfaces (UIs), especially single-page applications (SPAs).

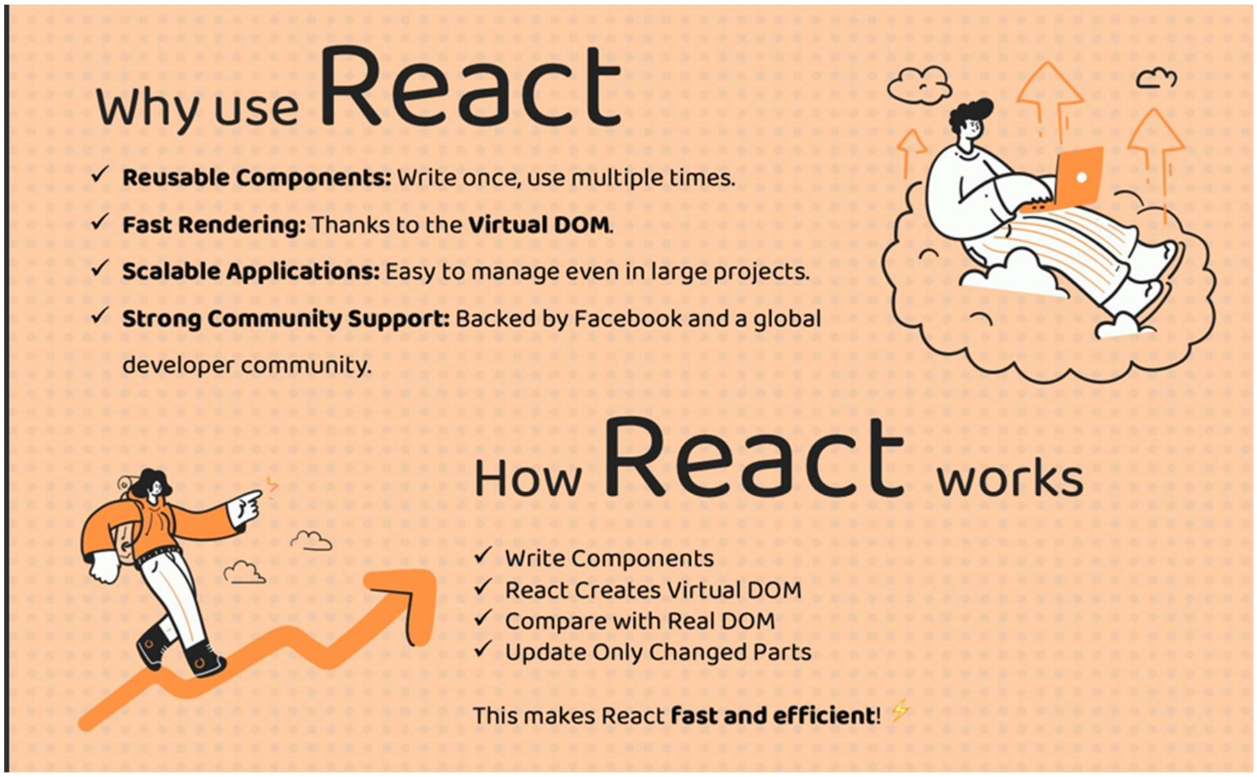


What is React

- ✓ Developed by **Facebook** (now Meta)
- ✓ Focuses on **building reusable UI components**
- ✓ Efficiently updates and renders UI with a **Virtual DOM**
- ✓ Follows a **component-based architecture**

Key Features of React

- ✓ **Component-Based:** Build UIs using small, reusable pieces called **components**.
- ✓ **Declarative Syntax:** Describe **what you want**, React figures out **how to do it**.
- ✓ **Virtual DOM:** Faster updates without directly manipulating the browser DOM.
- ✓ **Unidirectional Data Flow:** Data flows in **one direction**, making it easier to debug.
- ✓ **JSX (JavaScript XML):** Write HTML-like syntax in JavaScript.



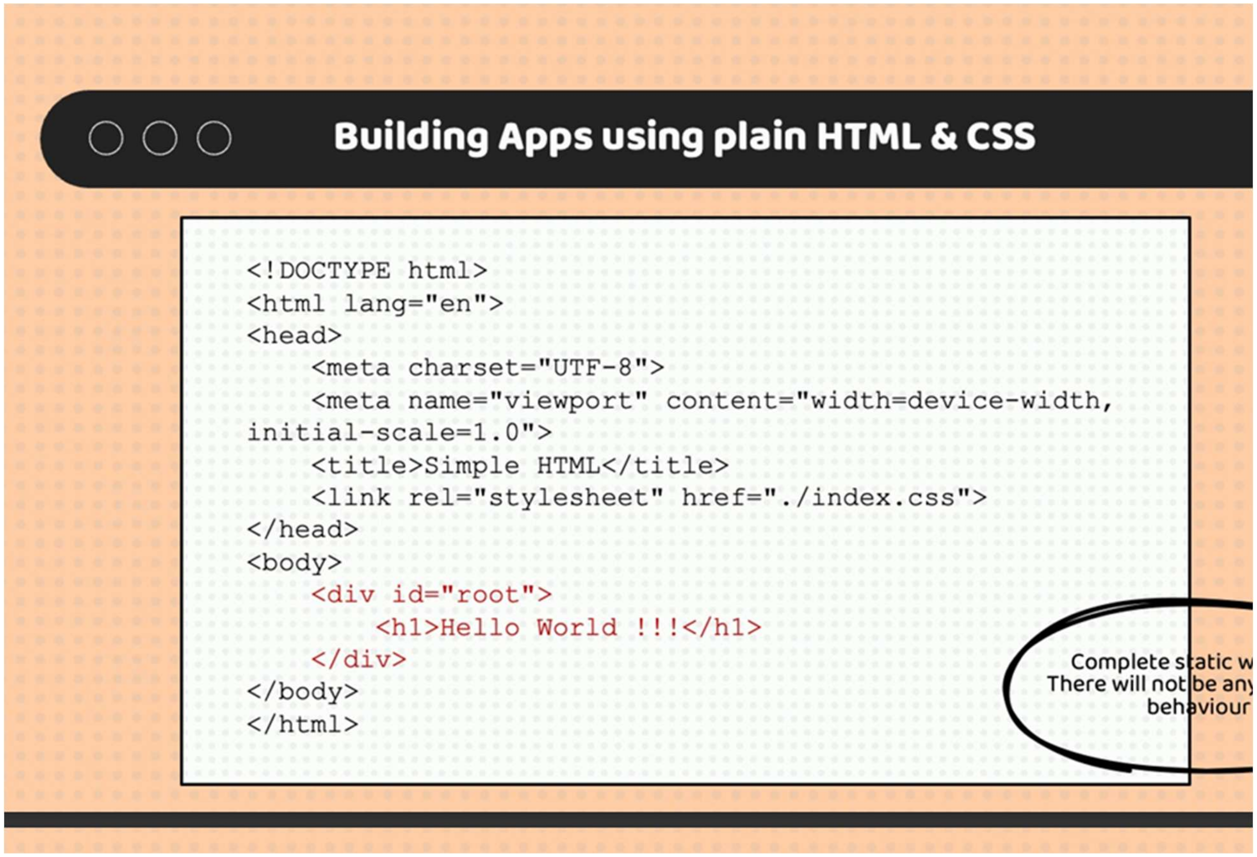
Why use React

- ✓ **Reusable Components:** Write once, use multiple times.
- ✓ **Fast Rendering:** Thanks to the **Virtual DOM**.
- ✓ **Scalable Applications:** Easy to manage even in large projects.
- ✓ **Strong Community Support:** Backed by Facebook and a global developer community.

How React works

- ✓ Write Components
- ✓ React Creates Virtual DOM
- ✓ Compare with Real DOM
- ✓ Update Only Changed Parts

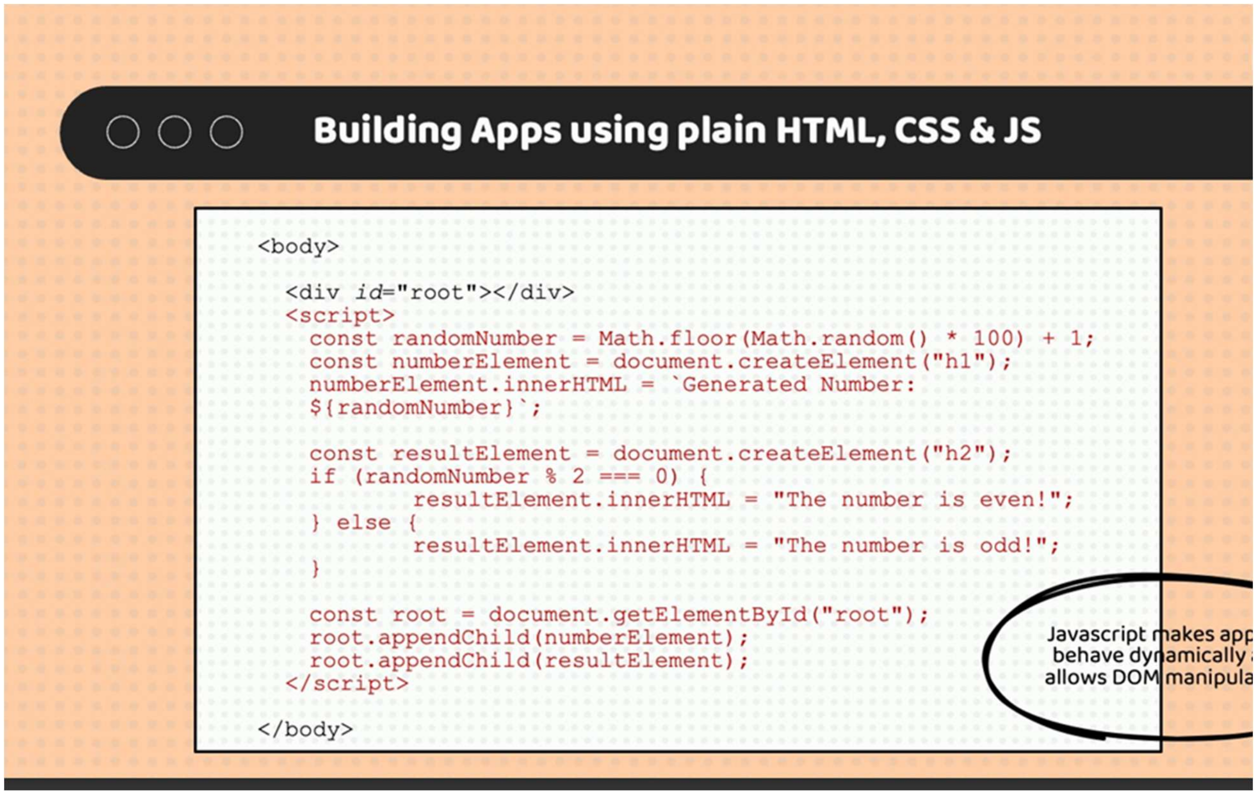
This makes React **fast and efficient**! ⚡



Building Apps using plain HTML & CSS

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Simple HTML</title>
    <link rel="stylesheet" href="./index.css">
</head>
<body>
    <div id="root">
        <h1>Hello World !!!</h1>
    </div>
</body>
</html>
```

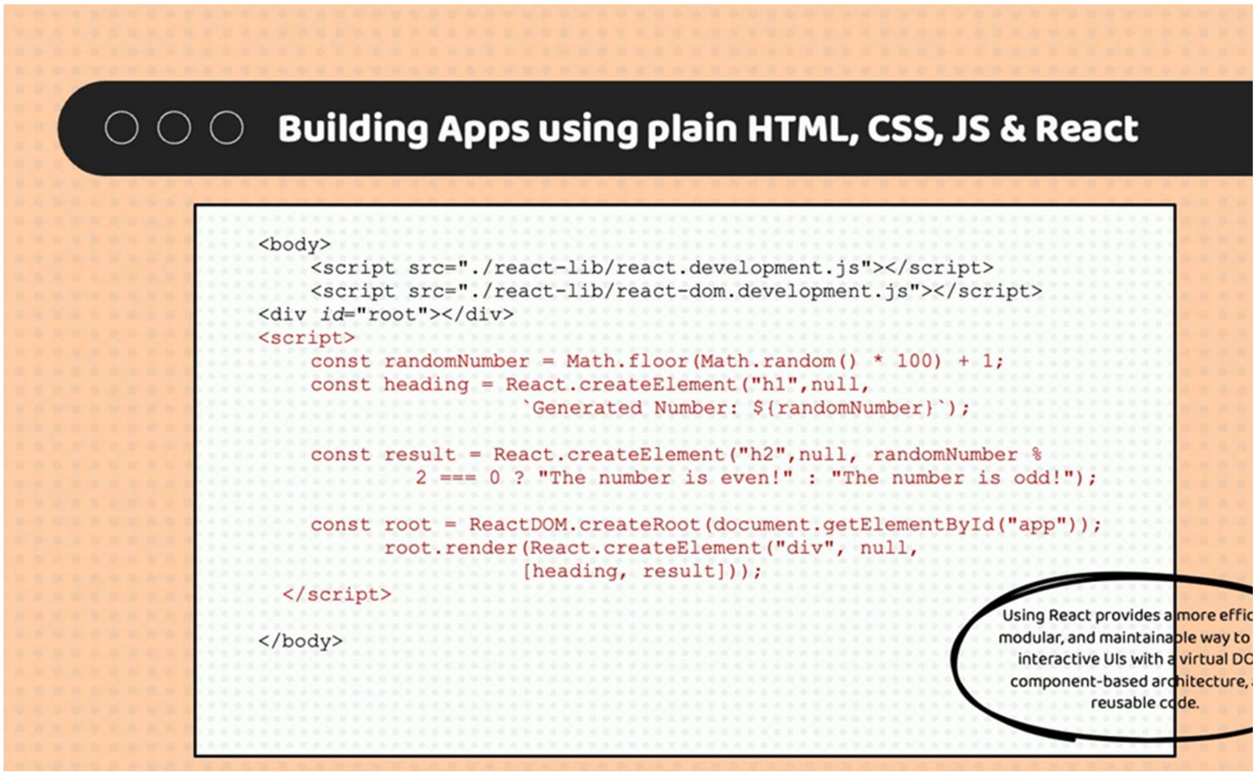
Complete static w
There will not be any
behaviour



Building Apps using plain HTML, CSS & JS

```
<body>

  <div id="root"></div>
  <script>
    const randomNumber = Math.floor(Math.random() * 100) + 1;
    const numberElement = document.createElement("h1");
    numberElement.innerHTML = `Generated Number:
    ${randomNumber}`;

    const resultElement = document.createElement("h2");
    if (randomNumber % 2 === 0) {
            resultElement.innerHTML = "The number is even!";
    } else {
            resultElement.innerHTML = "The number is odd!";
    }

    const root = document.getElementById("root");
    root.appendChild(numberElement);
    root.appendChild(resultElement);
  </script>

</body>
```

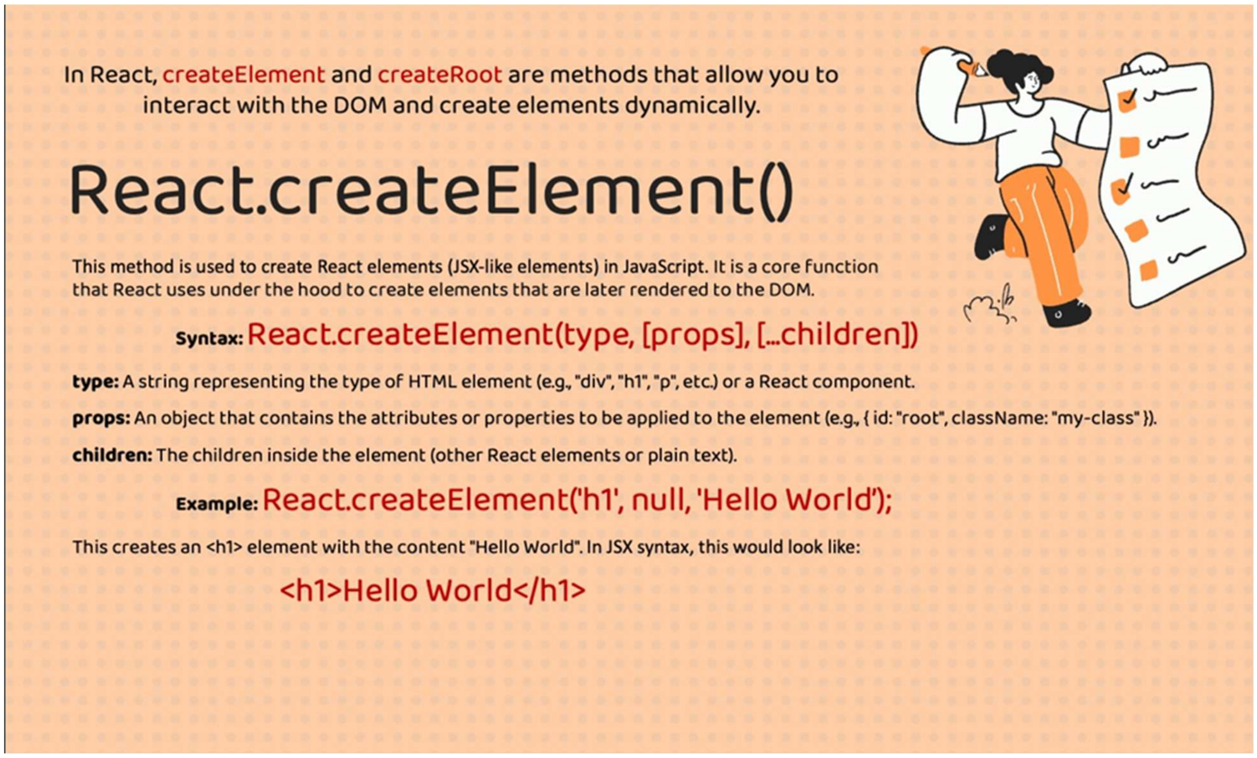
Javascript makes app
behave dynamically
allows DOM manipula



Building Apps using plain HTML, CSS, JS & React

```
<body>
    <script src="./react-lib/react.development.js"></script>
    <script src="./react-lib/react-dom.development.js"></script>
<div id="root"></div>
<script>
    const randomNumber = Math.floor(Math.random() * 100) + 1;
    const heading = React.createElement("h1",null,
                    `Generated Number: ${randomNumber}`);

    const result = React.createElement("h2",null, randomNumber %
            2 === 0 ? "The number is even!" : "The number is odd!");

    const root = ReactDOM.createRoot(document.getElementById("app"));
          root.render(React.createElement("div", null,
                    [heading, result]));
  </script>

</body>
```

Using React provides a more effic
modular, and maintainable way to
interactive UIs with a virtual DO
component-based architecture,
reusable code.



In React, createElement and createRoot are methods that allow you to interact with the DOM and create elements dynamically.

React.createElement()

This method is used to create React elements (JSX-like elements) in JavaScript. It is a core function that React uses under the hood to create elements that are later rendered to the DOM.

Syntax: React.createElement(type, [props], [...children])

type: A string representing the type of HTML element (e.g., "div", "h1", "p", etc.) or a React component.

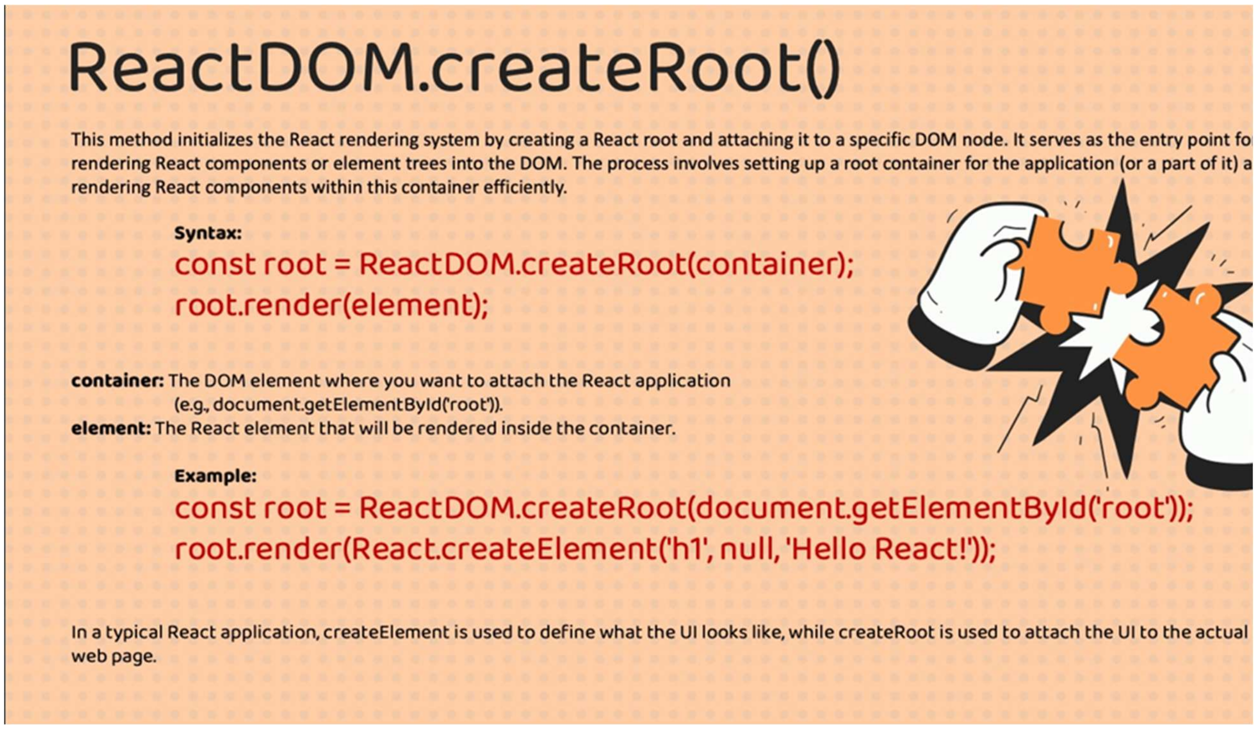
props: An object that contains the attributes or properties to be applied to the element (e.g., { id: "root", className: "my-class" }).

children: The children inside the element (other React elements or plain text).

Example: React.createElement('h1', null, 'Hello World');

This creates an <h1> element with the content "Hello World". In JSX syntax, this would look like:

```
<h1>Hello World</h1>
```



ReactDOM.createRoot()

This method initializes the React rendering system by creating a React root and attaching it to a specific DOM node. It serves as the entry point fo rendering React components or element trees into the DOM. The process involves setting up a root container for the application (or a part of it) a rendering React components within this container efficiently.

Syntax:

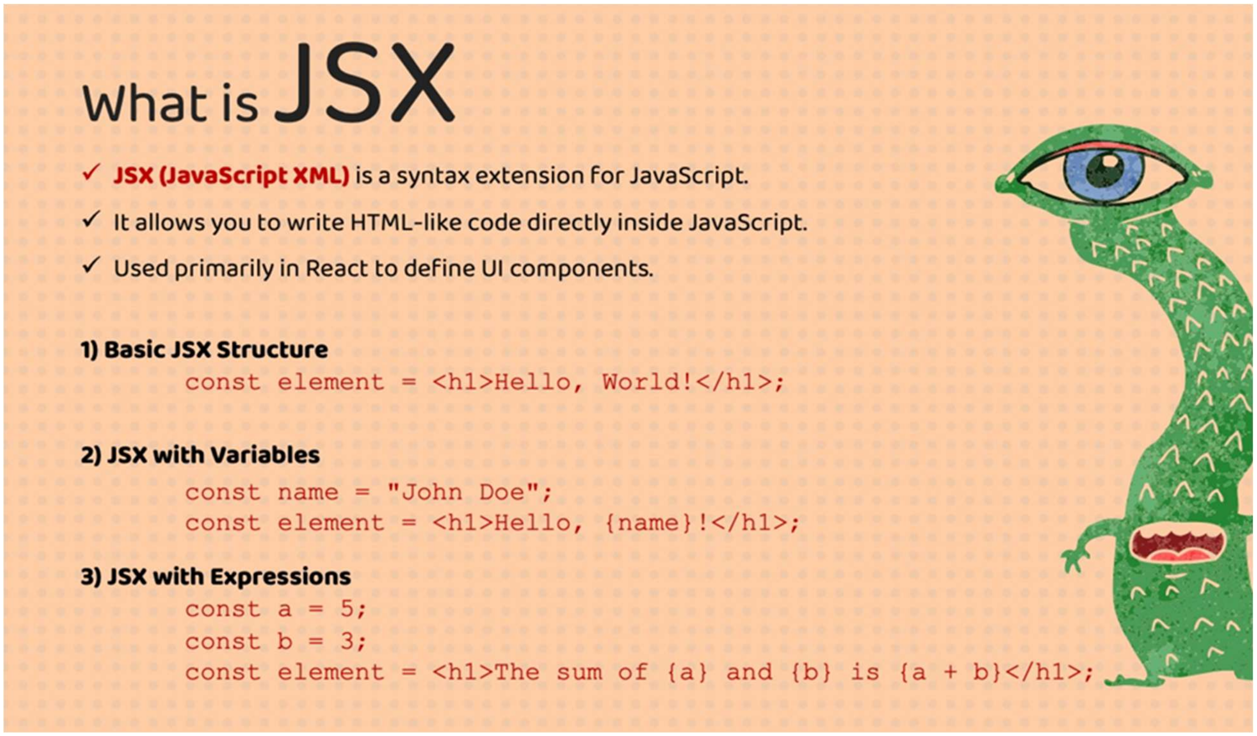
```
const root = ReactDOM.createRoot(container);
root.render(element);
```

container: The DOM element where you want to attach the React application (e.g., document.getElementById('root')).
element: The React element that will be rendered inside the container.

Example:

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(React.createElement('h1', null, 'Hello React!'));
```

In a typical React application, createElement is used to define what the UI looks like, while createRoot is used to attach the UI to the actual web page.



What is JSX

- ✓ **JSX (JavaScript XML)** is a syntax extension for JavaScript.
- ✓ It allows you to write HTML-like code directly inside JavaScript.
- ✓ Used primarily in React to define UI components.

1) Basic JSX Structure

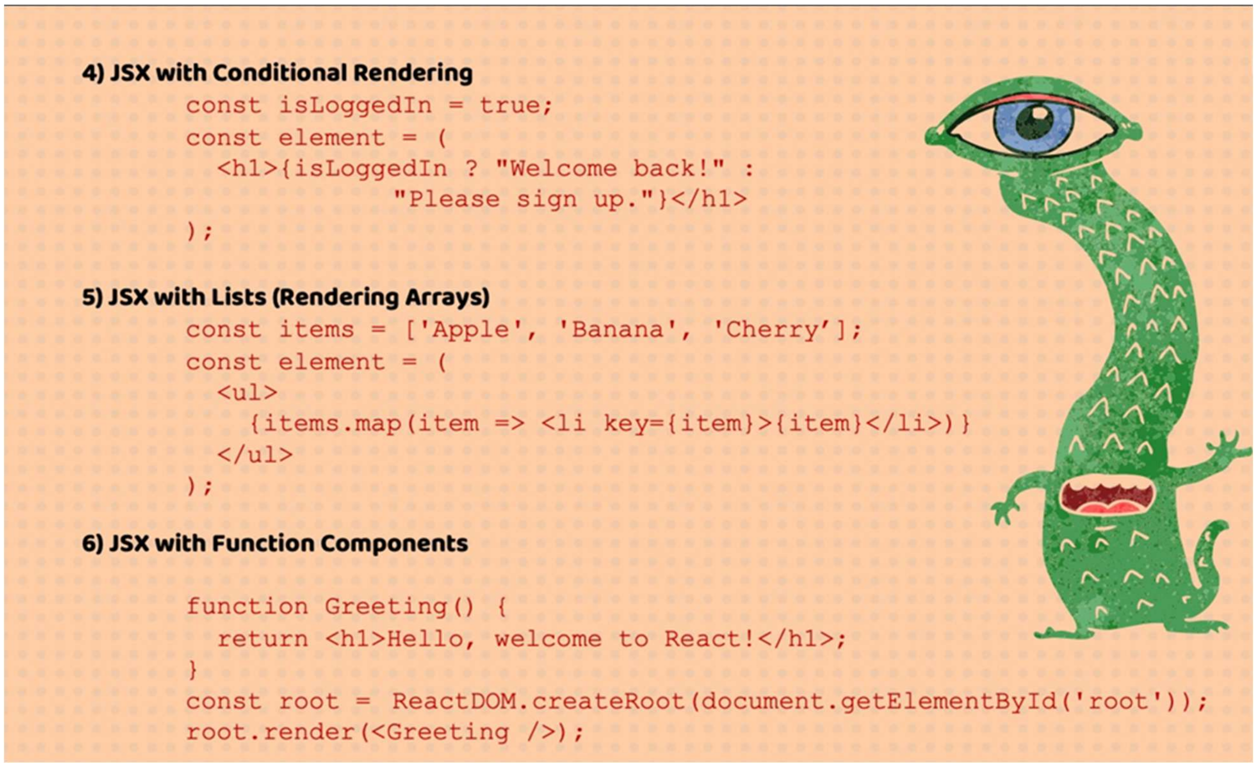
```
const element = <h1>Hello, World!</h1>;
```

2) JSX with Variables

```
const name = "John Doe";
const element = <h1>Hello, {name}!</h1>;
```

3) JSX with Expressions

```
const a = 5;
const b = 3;
const element = <h1>The sum of {a} and {b} is {a + b}</h1>;
```



4) JSX with Conditional Rendering

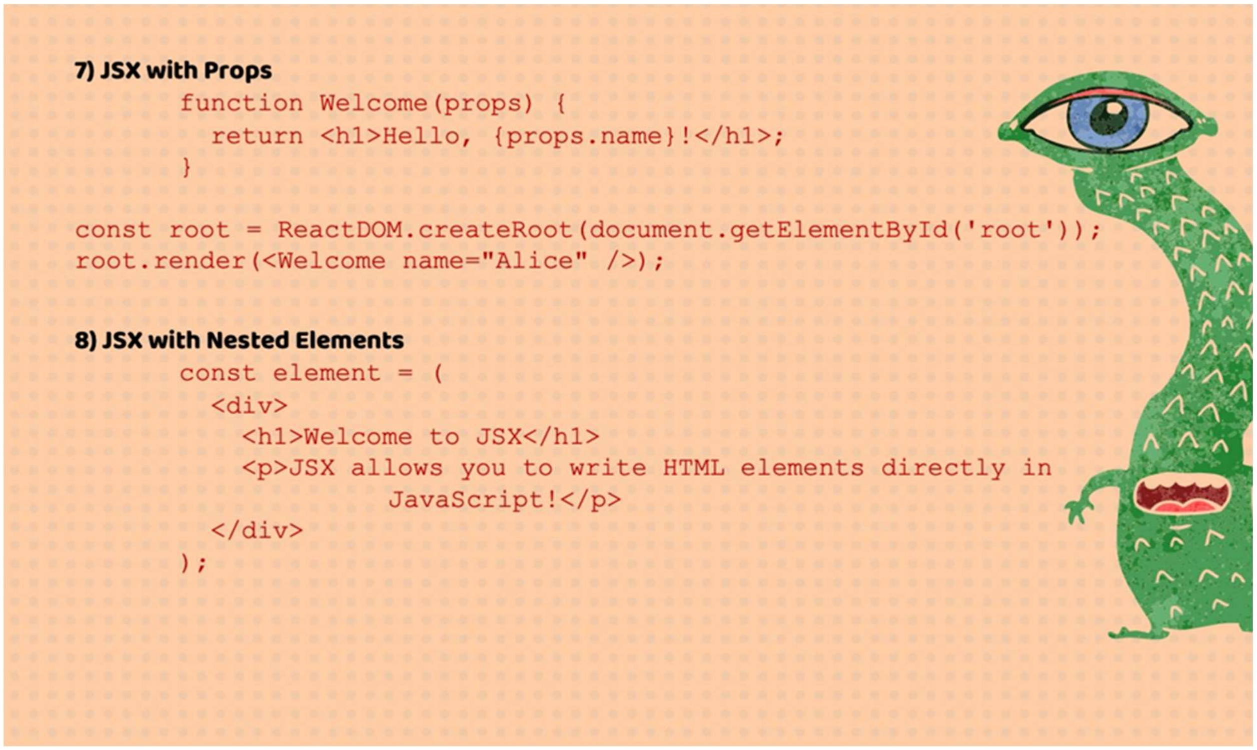
```
const isLoggedIn = true;
const element = (
  <h1>{isLoggedIn ? "Welcome back!" :
             "Please sign up."}</h1>
);
```

5) JSX with Lists (Rendering Arrays)

```
const items = ['Apple', 'Banana', 'Cherry'];
const element = (
  <ul>
    {items.map(item => <li key={item}>{item}</li>)}
  </ul>
);
```

6) JSX with Function Components

```
function Greeting() {
  return <h1>Hello, welcome to React!</h1>;
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Greeting />);
```

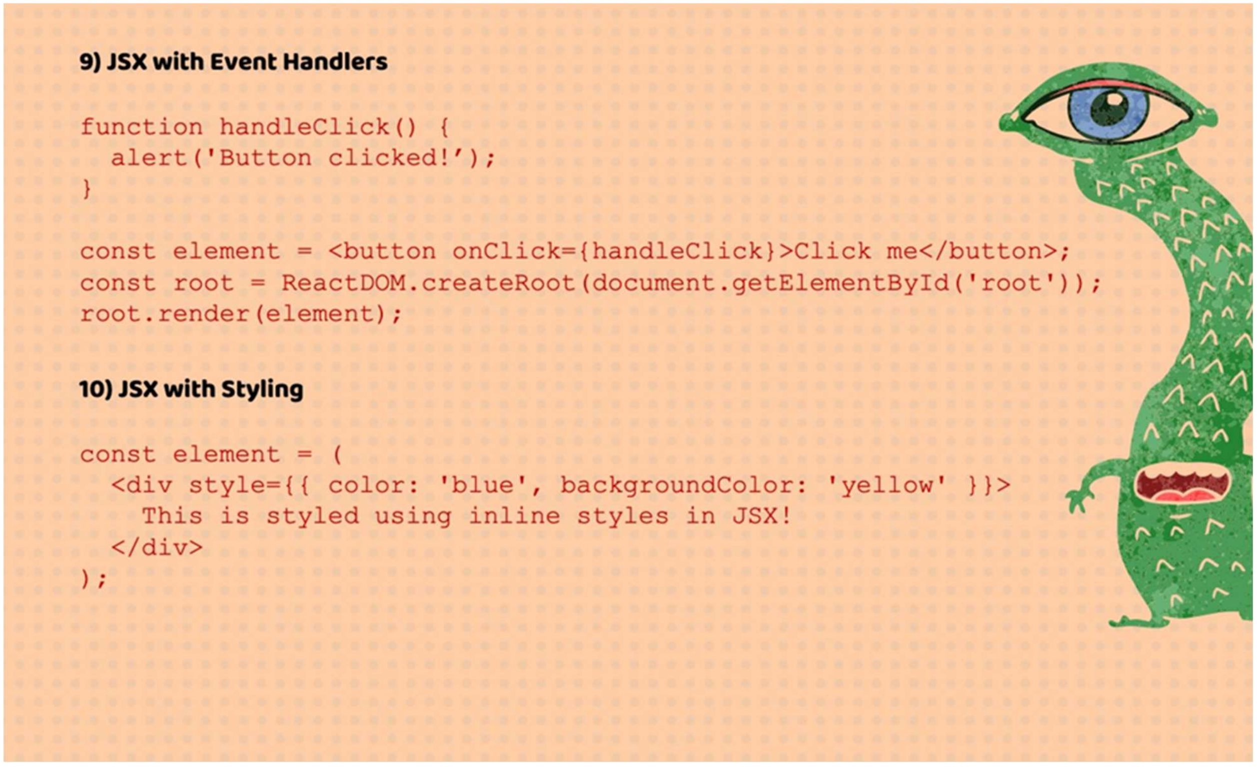


7) JSX with Props

```
       function Welcome(props) {
         return <h1>Hello, {props.name}!</h1>;
       }

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Welcome name="Alice" />);
```

8) JSX with Nested Elements

```
       const element = (
         <div>
           <h1>Welcome to JSX</h1>
           <p>JSX allows you to write HTML elements directly in
                     JavaScript!</p>
         </div>
       );
```

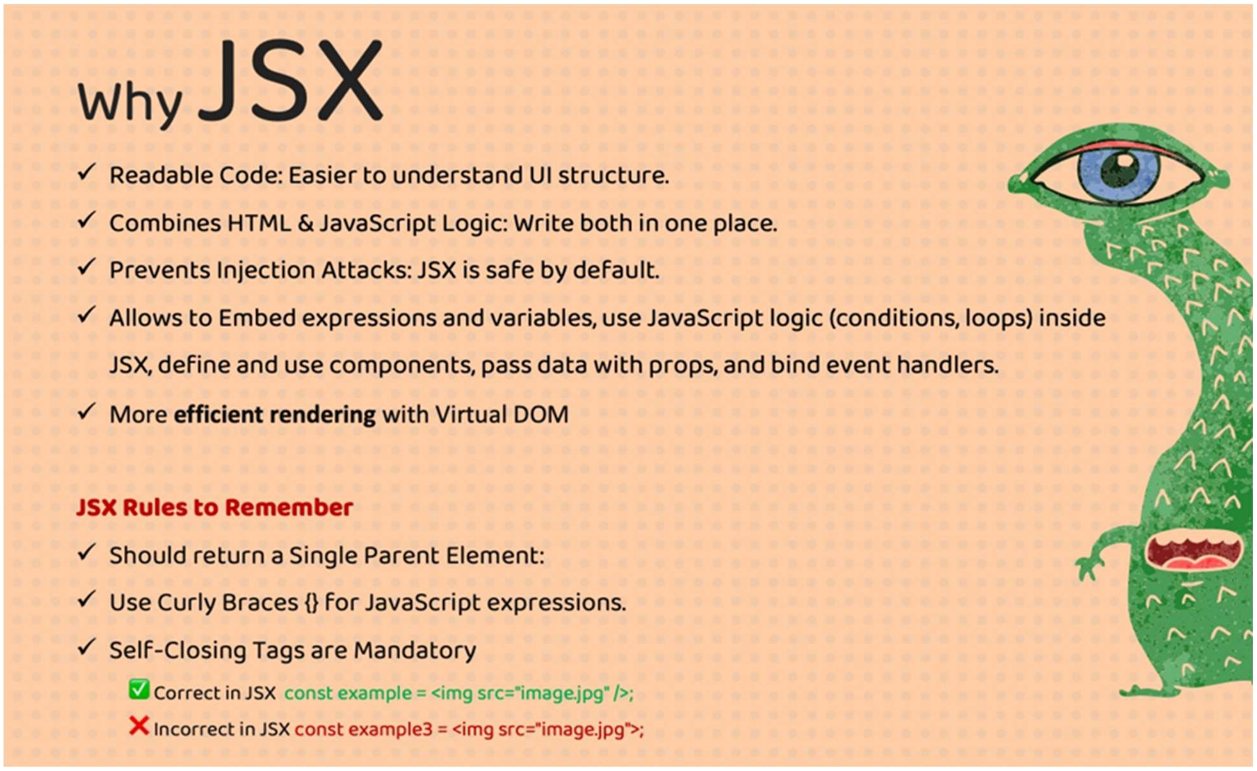


9) JSX with Event Handlers

```
function handleClick() {
  alert('Button clicked!');
}

const element = <button onClick={handleClick}>Click me</button>;
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(element);
```

10) JSX with Styling

```
const element = (
  <div style={{ color: 'blue', backgroundColor: 'yellow' }}>
    This is styled using inline styles in JSX!
  </div>
);
```

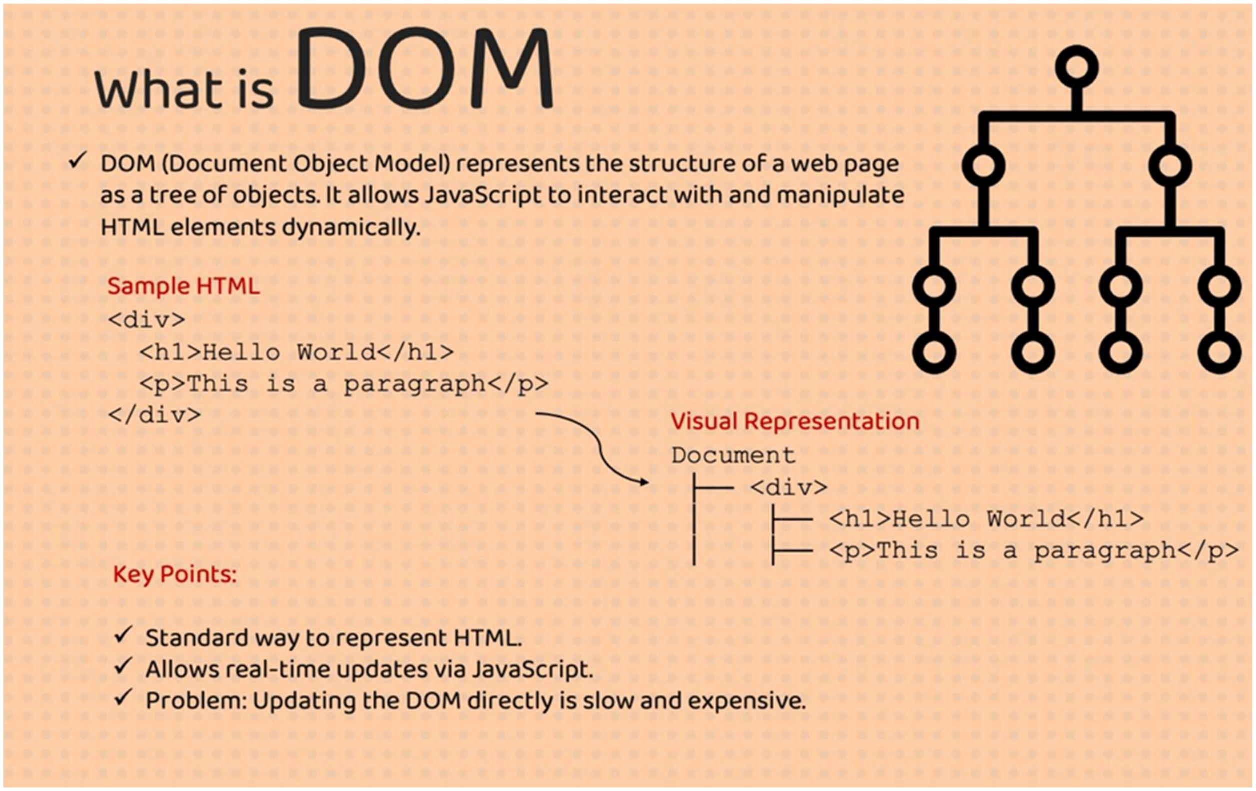


Why JSX

- ✓ Readable Code: Easier to understand UI structure.
- ✓ Combines HTML & JavaScript Logic: Write both in one place.
- ✓ Prevents Injection Attacks: JSX is safe by default.
- ✓ Allows to Embed expressions and variables, use JavaScript logic (conditions, loops) inside JSX, define and use components, pass data with props, and bind event handlers.
- ✓ More **efficient rendering** with Virtual DOM

JSX Rules to Remember

- ✓ Should return a Single Parent Element:
- ✓ Use Curly Braces {} for JavaScript expressions.
- ✓ Self-Closing Tags are Mandatory
  - ✅ Correct in JSX `const example = <img src="image.jpg" />;`
  - ❌ Incorrect in JSX `const example3 = <img src="image.jpg">;`



What is DOM

- ✓ DOM (Document Object Model) represents the structure of a web page as a tree of objects. It allows JavaScript to interact with and manipulate HTML elements dynamically.

Sample HTML

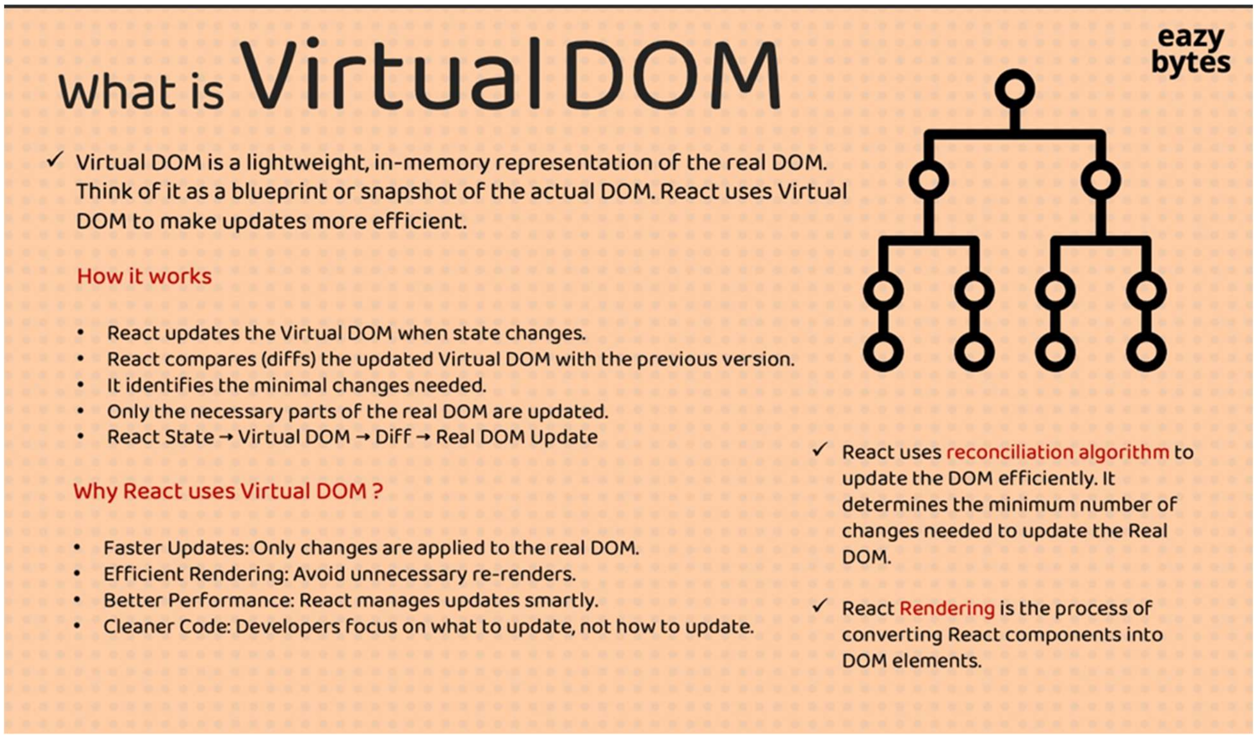
```
<div>
  <h1>Hello World</h1>
  <p>This is a paragraph</p>
</div>
```

Visual Representation

```
Document
 ├── <div>
 │    ├── <h1>Hello World</h1>
 │    ├── <p>This is a paragraph</p>
```

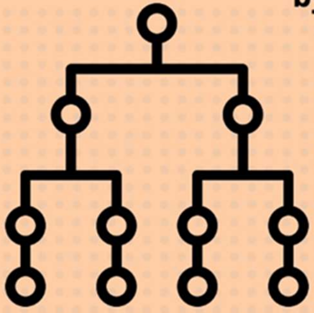
Key Points:

- ✓ Standard way to represent HTML.
- ✓ Allows real-time updates via JavaScript.
- ✓ Problem: Updating the DOM directly is slow and expensive.



What is Virtual DOM

eazy bytes



✓ Virtual DOM is a lightweight, in-memory representation of the real DOM. Think of it as a blueprint or snapshot of the actual DOM. React uses Virtual DOM to make updates more efficient.

How it works

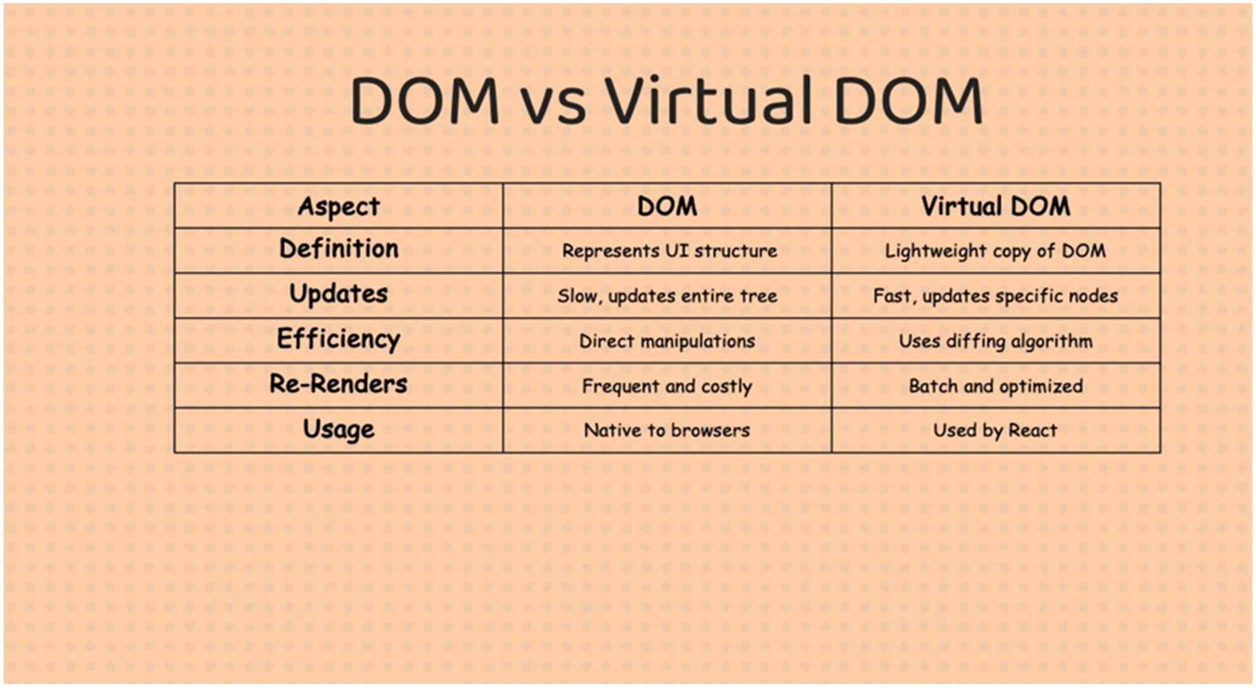
- React updates the Virtual DOM when state changes.
- React compares (diffs) the updated Virtual DOM with the previous version.
- It identifies the minimal changes needed.
- Only the necessary parts of the real DOM are updated.
- React State → Virtual DOM → Diff → Real DOM Update

Why React uses Virtual DOM ?

- Faster Updates: Only changes are applied to the real DOM.
- Efficient Rendering: Avoid unnecessary re-renders.
- Better Performance: React manages updates smartly.
- Cleaner Code: Developers focus on what to update, not how to update.

✓ React uses reconciliation algorithm to update the DOM efficiently. It determines the minimum number of changes needed to update the Real DOM.

✓ React Rendering is the process of converting React components into DOM elements.



DOM vs Virtual DOM

Aspect	DOM	Virtual DOM
Definition	Represents UI structure	Lightweight copy of DOM
Updates	Slow, updates entire tree	Fast, updates specific nodes
Efficiency	Direct manipulations	Uses diffing algorithm
Re-Renders	Frequent and costly	Batch and optimized
Usage	Native to browsers	Used by React

Semantic Versioning (SemVer) Basics

The symbols like ^, ~, and others in package.json dependencies are called SemVer (Semantic Versioning) Range Specifiers. They define the allowed range of versions for each dependency.

Semantic Versioning follows the format:

```
MAJOR.MINOR.PATCH
```

MAJOR: Introduces breaking changes.
MINOR: Adds new features without breaking existing functionality.
PATCH: Fixes bugs without changing functionality.

```
1.2.9 → MAJOR: 1, MINOR: 2, PATCH: 9
```

Version Specifiers in a Nutshell

^ (caret) → Update MINOR and PATCH safely. (Recommended for most libraries)
~ (tilde) → Update only PATCH.
No Symbol → Lock to an exact version.

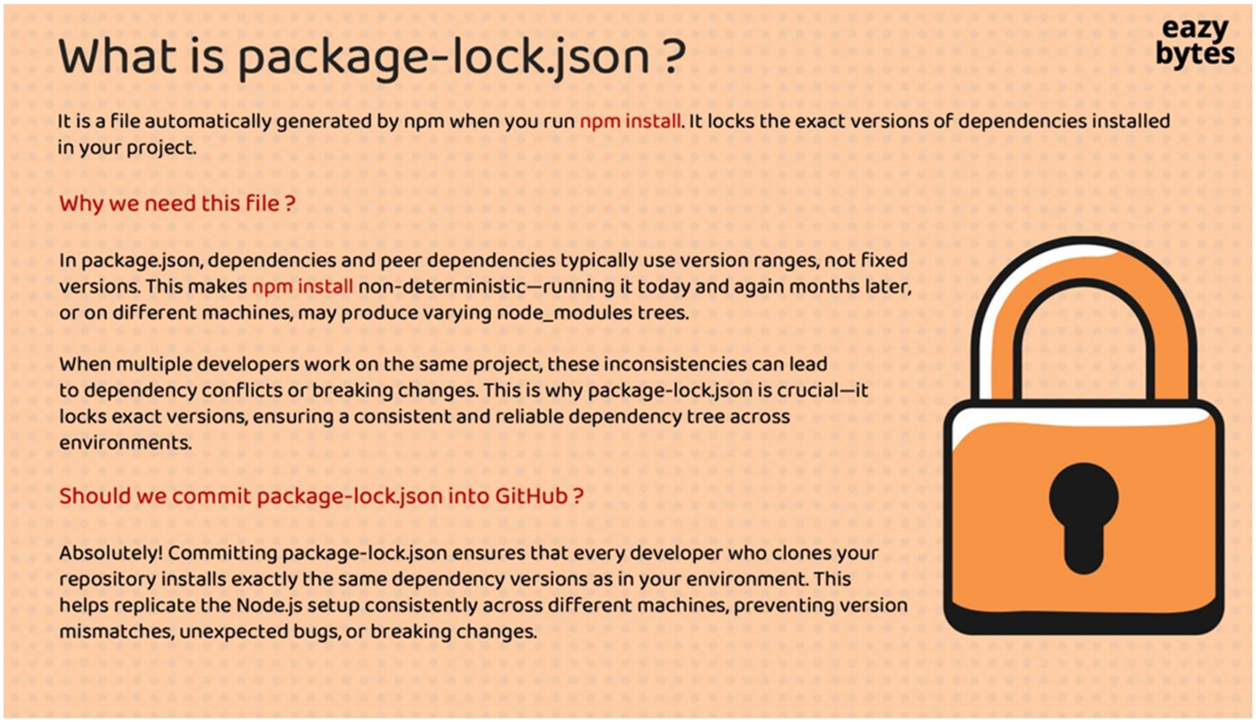
eazy bytes

Meaning of Symbols

Symbol	Example	Meaning
^	^1.2.9	Allows updates that do **not** change the **MAJOR** version. Example: 1.x.x but **not** 2.x.x.
~	~1.2.9	Allows updates that do **not** change the **MINOR** version. Example: 1.2.x but **not** 1.3.0.
>	>1.2.9	Allows versions **greater than** 1.2.9
>=	>=1.2.9	Allows versions **greater than or equal to** 1.2.9
<	<1.2.9	Allows versions **less than** 1.2.9
<=	<=1.2.9	Allows versions **less than or equal to** 1.2.9
*	*	Matches **any version**. (Rarely used in production)
x	1.x.x	Matches any version for the x position (wildcard).
latest	latest	Always installs the **latest published version**.

Best Practices

1.Use ^ for **libraries** and frameworks (**default behavior**) to stay updated with compatible MINOR and PATCH releases.
2.Use ~ for **tools or plugins** where patch-level updates are safe.
3.Use **exact versions** (1.2.9) for production-critical dependencies when stability is a must.



What is package-lock.json ?

eazy bytes

It is a file automatically generated by npm when you run npm install. It locks the exact versions of dependencies installed in your project.

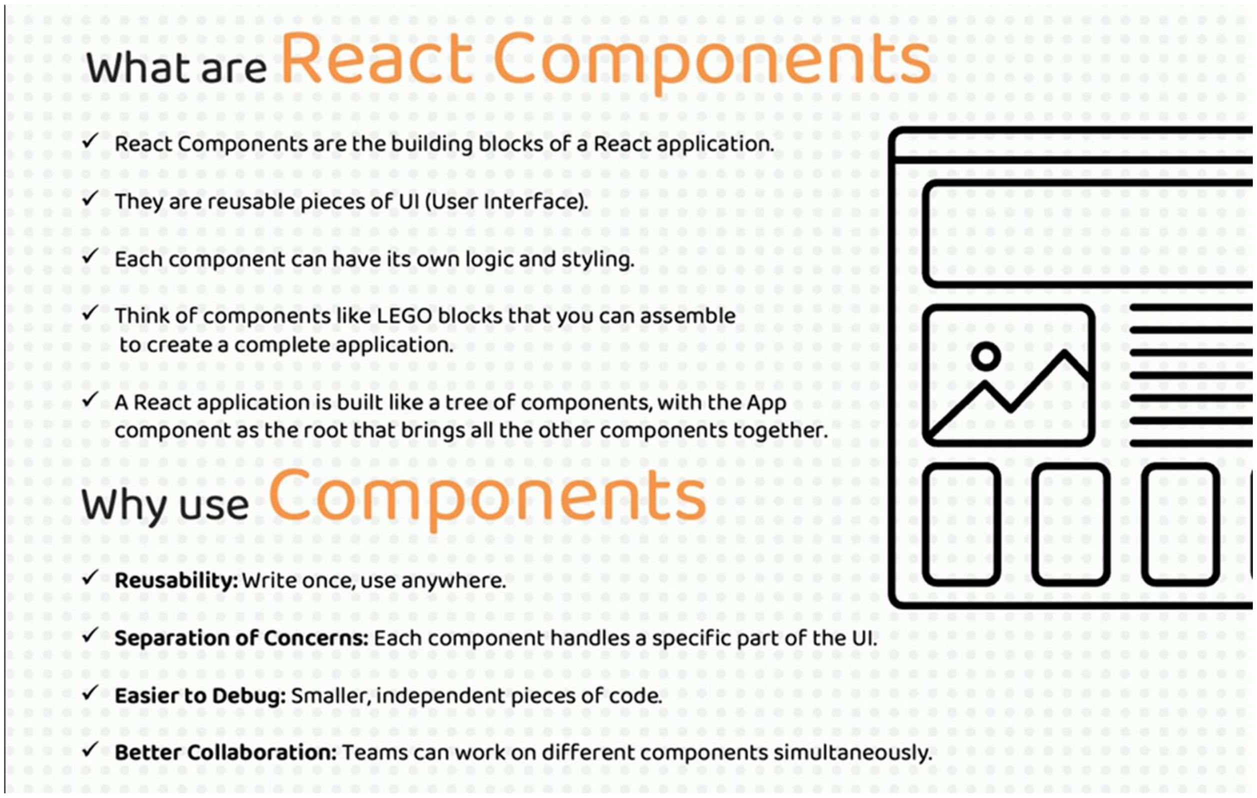
Why we need this file ?

In package.json, dependencies and peer dependencies typically use version ranges, not fixed versions. This makes npm install non-deterministic—running it today and again months later, or on different machines, may produce varying node_modules trees.

When multiple developers work on the same project, these inconsistencies can lead to dependency conflicts or breaking changes. This is why package-lock.json is crucial—it locks exact versions, ensuring a consistent and reliable dependency tree across environments.

Should we commit package-lock.json into GitHub ?

Absolutely! Committing package-lock.json ensures that every developer who clones your repository installs exactly the same dependency versions as in your environment. This helps replicate the Node.js setup consistently across different machines, preventing version mismatches, unexpected bugs, or breaking changes.



What are React Components

- ✓ React Components are the building blocks of a React application.
- ✓ They are reusable pieces of UI (User Interface).
- ✓ Each component can have its own logic and styling.
- ✓ Think of components like LEGO blocks that you can assemble to create a complete application.
- ✓ A React application is built like a tree of components, with the App component as the root that brings all the other components together.

Why use Components

- ✓ **Reusability:** Write once, use anywhere.
- ✓ **Separation of Concerns:** Each component handles a specific part of the UI.
- ✓ **Easier to Debug:** Smaller, independent pieces of code.
- ✓ **Better Collaboration:** Teams can work on different components simultaneously.

eazy bytes

React Component Rules

- ✓ React Component names should start with an uppercase letter.
- ✓ Choose a name that clearly describes the UI component (e.g., "Header" or "MyHeader").
- ✓ For multi-word names, use PascalCase formatting (e.g., "MyHeader").
- ✓ The component function must return a value that React can render (i.e., display on the screen).
- ✓ Typically, it returns JSX.
- ✓ It can also return a string, number, boolean, null, or an array of these values.