CBIT Open Source Community

In Association With **CDC**

Presents

REACTJS & FASTAPI

BOOTCAMP

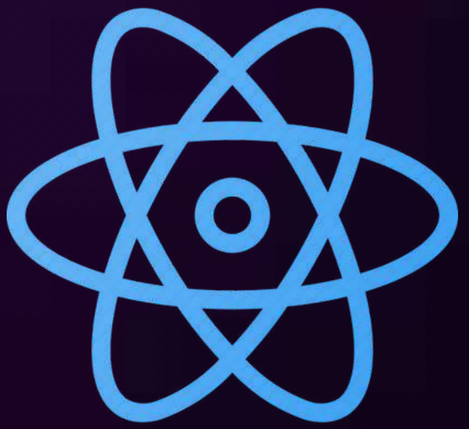
ABOUT COSC

COSC, an **esteemed tech community** based in Chaitanya Bharathi Institute of Technology - Hyderabad, is dedicated to promoting an open-source ethos. We organize a diverse range of events such as **hackathons, bootcamps, and workshops**, aimed at educating students about various technologies while actively fostering a culture of open source.

In these dynamic gatherings, students immerse themselves in collaborative **coding sessions, innovative problem-solving challenges, and hands-on experiences**, all centered around open source principles. By embracing this approach, we empower students to not only learn about cutting-edge technologies but also actively contribute to their development and dissemination.

BOOTCAMP OVERVIEW

- This boot camp will provide hands-on experience with ReactJS, focusing on component-based architecture, state management, and hooks.
- This boot camp will also cover FastAPI, emphasizing speed and simplicity in creating robust backend services.
- You will learn to integrate ReactJS with FastAPI, enabling them to build full-stack web applications.
- Also, a structured schedule includes practical assignments and a final assessment, with certificates awarded upon completion



TIMELINE
13 AUG 2024 (OFFLINE)

9:10 - 10:10

Bootcamp Overview
Configuring IDEs

10:15 - 11:15

Basics of HTML & CSS

11:20 - 12:10

Essential JS concepts

12:10 - 13:00

Lunch

13:10 - 14:00

React Components

14:05 - 16:00

Props, Hooks
API Introduction

TIMELINE
14 AUG 2024 (OFFLINE)

9:10 - 10:10

API Overview
Middleware

10:15 - 12:10 **9:10 - 10:10**

CRUD operations

12:10 - 13:00

Lunch

13:10 - 14:00

Async &
Background Tasks

14:05 - 16:00

Incorporating an ML
model in a Website

TIMELINE (ONLINE)

16 AUG
Assignment 1

18 AUG
Assignment 2

20 AUG
Assignment 3

22 AUG
Doubts Clearance Session
Assignment Submission Deadline

23 AUG
Allotment of Final Assessment

25 AUG
Final Assessment Deadline

Intro To <HTML>

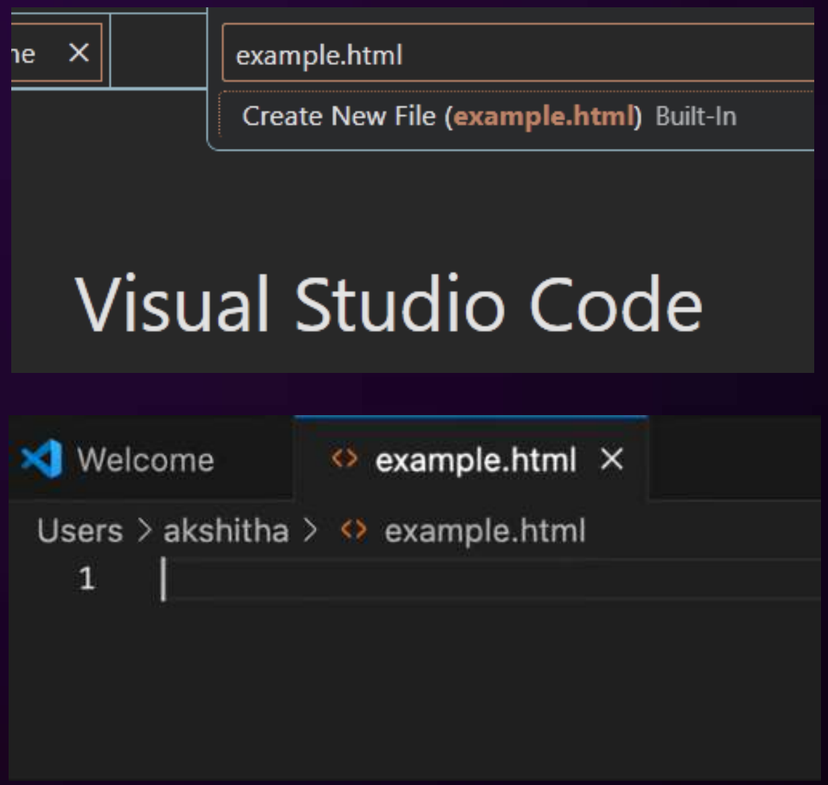
INTRODUCTION TO HTML

- HTML (Hypertext Markup Language) is a tag-based language used to format static content on web pages.
- HTML files are plain text with extensions .html or .htm, the latter from older systems with a three-character limit.
- While static content remains unchanged, dynamic content uses scripts or applets for user interaction.

INTRODUCTION TO HTML

Steps to create a HTML page:

- Write your HTML code in Visual Studio Code.
- Save the file with a `.html` extension on your computer.
- Open the file in a web browser and refresh to view updates.
- Continuously edit, save, and test the code to ensure accuracy.



INTRODUCTION TO HTML

Tag Overview:

- HTML wraps content in tags enclosed by < >. Most tags require both a start tag and an end tag, with the content placed between them.
- A start tag and its end tag are viewed as a container.
- For example:

```
<p> CBIT Open Source Community </p>
```

- Some tags have only a start tag, with all the necessary information embedded within the tag.
- For example:

```
<img src="COSC-logo.jpg">
```

INTRODUCTION TO HTML

Tags that start with “<!” are never displayed within the browser.

There are two types :

1. Comments

```
<!-- This is a comment -->
```

2. DOCTYPE tag: It is used to identify that the file is a html code.

```
<!DOCTYPE html>
```

A html code starts with the tag <html> and ends with the tag </html>.

The <html> tag contains two direct child elements:

1. <head> </head>
2. <body> </body>

HEAD TAG

<head> : The <head> tag in HTML contains essential information about the document, which is not directly displayed on the web page

- Metadata: Provides information such as character encoding and viewport settings.
- <title>: Specifies the title of the web page, shown in the browser's title bar or tab.
- <link>: Links external resources like CSS files to the HTML document.
- <meta>: Defines metadata like description, keywords, and author.
- <style>: Allows you to include internal CSS for styling.

HEAD TAG

tutorial.html

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <meta name="description" content="A brief description of the webpage
content.">
    <title>My Web Page</title>
    <link rel="stylesheet" href="styles.css">
</head>
```

BODY TAG

<body> : The <body> tag contains all the visible elements that appear on a webpage. These elements include headings, paragraphs, images, links, and more

Some of them are:

- <h1>: Defines the main heading, the largest and most important text on the page.
- <p>: Creates a paragraph of text, grouping content into blocks.
-
: Inserts a line break, moving the next content to a new line without starting a new paragraph.
- <img/>: Embeds an image into the webpage.
- <a>: Creates a hyperlink, allowing users to navigate to other pages or resources.

BODY TAG

```
<body>
    <h1>Main Heading of the Page</h1> <!-- Main heading -->
    <p>This is a paragraph of text.</p> <!-- Paragraph -->
    <p>This is another paragraph with an <a href="#">embedded link</a>.
</p> <!-- Paragraph with link -->
    <br/> <!-- Line break -->
    <p>Text after a line break.</p> <!-- Paragraph after line break -->
    <img src="image.jpg" alt="Sample Image"/> <!-- Embedded image -->
</body>
```

IMG TAG

SOME OTHER IMPORTANT TAGS:

<img>: The <img> tag is used to embed images in a webpage. It is self-closing, meaning it doesn't require an end tag (</img>). The <img> tag includes the following attributes:

- src: Specifies the path to the image file, which can be a relative path or a URL.
- alt: Provides alternative text that displays if the image fails to load, improving accessibility.

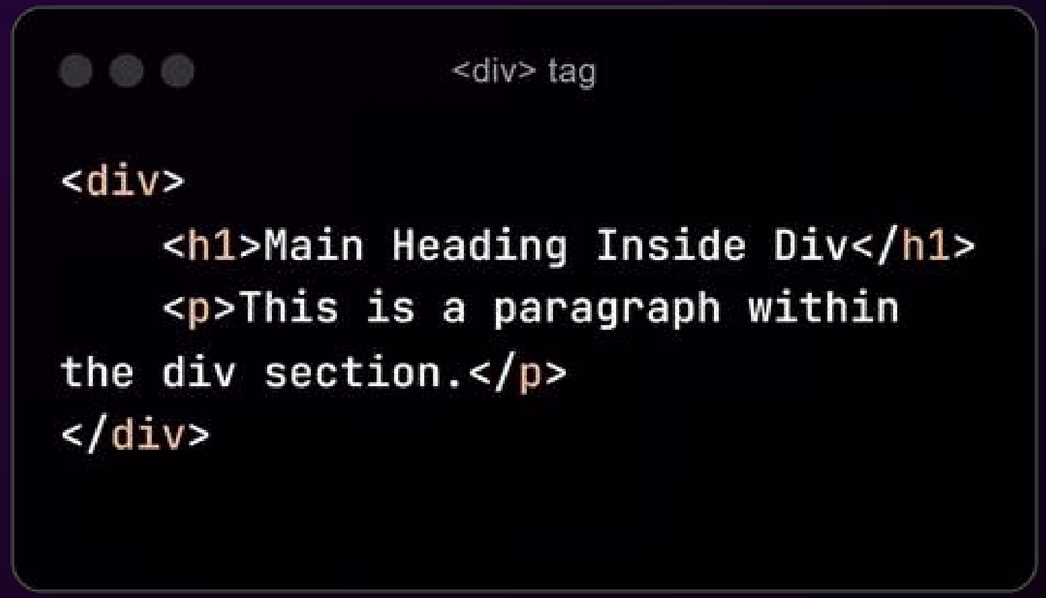
tutorial.html

```
<img src="images/photo.jpg" alt="A beautiful sunset"/> <!-- Embedded image with
a sample path -->
```

DIV TAG

** :**

- The tag is used to group multiple elements together, creating a distinct section within the webpage.
- It closes with the tag, allowing for easy management and organization of content.
- Groups content like headings and paragraphs into a section, making it easier to structure and manage the layout of the webpage.



<a> TAG

<a>:

- The <a> tag is used to create hyperlinks, enabling navigation to other pages, sections, or resources.
- href Attribute: Specifies the destination URL for the link.
- target Attribute: Defines where to open the linked document (e.g., _blank for a new tab).
- title Attribute: Provides additional information about the link when hovered over.

TABLES

Table:

- Tables in HTML are used to organize and present data in a structured format, consisting of rows and columns.
- Tables are represented with the start tag <table> and end tag </table>.
- The rows are represented with <tr> tags and columns are represented with <td> tags inside the <tr> </tr> tags.

<table> tag

```
<table>
        <tr>
            <th>Name</th>
            <th>Age</th>
        </tr>
        <tr>
            <td>John Doe</td>
            <td>25</td>
        </tr>
        <tr>
            <td>Jane Smith</td>
            <td>30</td>
        </tr>
</table>
```

FORM IN HTML

Form: Forms in HTML are essential for collecting user input and facilitating interaction on web pages. They allow users to submit data to a server for processing, such as logging in, signing up, or providing feedback.

Basic Structure of a Form:

- <form>: Defines the form and specifies where and how the data will be submitted.
- <input>: Collects user input, with various types like text, password, and checkbox.
- <button>: Creates a clickable button to submit the form or trigger an action.
- <label>: Provides a label for an input element, enhancing accessibility.
- <select>: Creates a dropdown list for users to select an option.
- <option>: Defines individual options within a <select> dropdown list.

FORM IN HTML

Registration Form

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Registration Form</title>

</head>
<body>

<div class="container">
    <h2>Registration Form</h2>

    <p>Please fill in this form to create an account.</p>
```

FORM IN HTML

```
    <img src="https://via.placeholder.com/150" alt="Registration Image"
style="display: block; margin: 0 auto;">

    <form action="/submit_registration" method="post">
        <table>
            <tr>
                <td><label for="fname">First Name:</label></td>
                <td><input type="text" id="fname" name="firstname" required></td>
            </tr>
            <tr>
                <td><label for="lname">Last Name:</label></td>
                <td><input type="text" id="lname" name="lastname" required></td>
            </tr>
            <tr>
                <td><label for="email">Email:</label></td>
                <td><input type="email" id="email" name="email" required></td>
            </tr>
            <tr>
```

FORM IN HTML

```
            <tr>
                <td><label for="password">Password:</label></td>
                <td><input type="password" id="password" name="password" required>
</td>
            </tr>
            <tr>
                <td><label for="country">Country:</label></td>
                <td>
                    <select id="country" name="country" required>
                        <option value="">Select your country</option>
                        <option value="usa">United States</option>
                        <option value="canada">Canada</option>
                        <option value="uk">United Kingdom</option>
                        <option value="australia">Australia</option>
                    </select>
                </td>
            </tr>
        </table>
```

FORM IN HTML

```
        <div style="text-align: center;">
            <input type="submit" value="Register">
            <button type="reset">Reset</button>
        </div>
    </form>

    <p>Already have an account? <a href="/login">Login here</a>.</p>
</div>

</body>
</html>
```

LISTS IN HTML

Lists : In HTML, lists are used to group related items together, making content more organized and easier to read. There are different types of lists such as ordered lists, unordered lists, and description lists.

The tag for lists can be simply represented as <li> and

Ordered list:

- The <ol> tag is used to create an ordered list, where the items are presented in a specific sequence, typically numbered.
- : Each item in the list is defined by the <li> tag, which stands for "list item."

Unordered list:

- : The <ul> tag is used to create an unordered list, where the items are presented with bullet points and the order is not significant.
- : Like in an ordered list, each item in the unordered list is defined by the <li> tag

LISTS IN HTML

Ordered List

```
<ol>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
</ol>
```

Unordered List

```
<ul>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
</ul>
```


CO
SC
CSS
Cascading Style Sheets

What is CSS ?

CSS (Cascading Style Sheets) is a stylesheet language used for describing the presentation of a document written in HTML or XML.

Purpose:

- Separates content from design
- Enhanced User Experience
- Reusability

Advantages:

- Consistency
- Improved Performance
- Accessibility

Basic Structure

1. Selector:

Identifies the HTML element(s) to style.

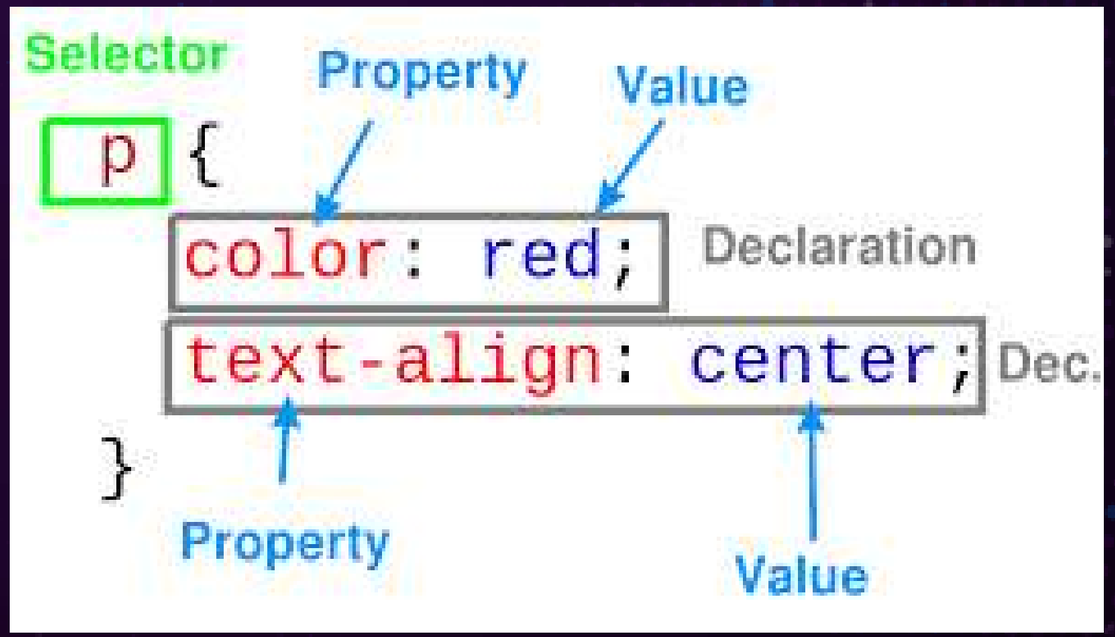
2. Property:

The aspect of the element you want to change (e.g., color, font-size).

3. Value:

The specific setting for the property.

4. Commenting: Use /* comment */ to add comments in your CSS.



CSS Types

1.Inline CSS:

- Applied directly to the HTML element using the style attribute.
- Useful for quick, one-off styling but not ideal for large projects.

Inline CSS

```
<p style="color: blue; font-size:
14px;">
  This is an inline styled paragraph.
</p>
```

2.Internal CSS:

- Placed within the <style> tag inside the <head> section of an HTML document.
- Good for styling a single page without affecting other pages.

Internal CSS

```
<head>
    <style>
        p { color: green; font-size:
16px; }
    </style>
</head>
<body>
    <p>This is a paragraph with
internal CSS.</p>
</body>
```

CSS Types

3.External CSS:

- Stored in a separate .css file and linked to multiple HTML files.
- Ideal for maintaining consistency across multiple pages and for large projects.

External CSS

```
<!-- HTML File -->
<head>
    <link rel="stylesheet"
href="styles.css">
</head>
<body>
    <p>This is a paragraph with
external CSS.</p>
</body>

/* styles.css */
p { color: red; font-size: 18px; }
```

CSS Selectors

CSS Selectors target specific HTML elements for applying styles.

1. Universal Selector (*):

- Selects all elements on the page.

2. Element Selector:

- Targets all elements of a specific type .

```
/* Universal Selector */

* {
    color: blue;
}
/* Element Selectors */
p {
    color: red;
}
h1 {
    color: green;
}
```

CSS Selectors

CSS Selectors target specific HTML elements for applying styles.

3. Class Selector:

- Targets elements with a specific class attribute.

4. ID Selector:

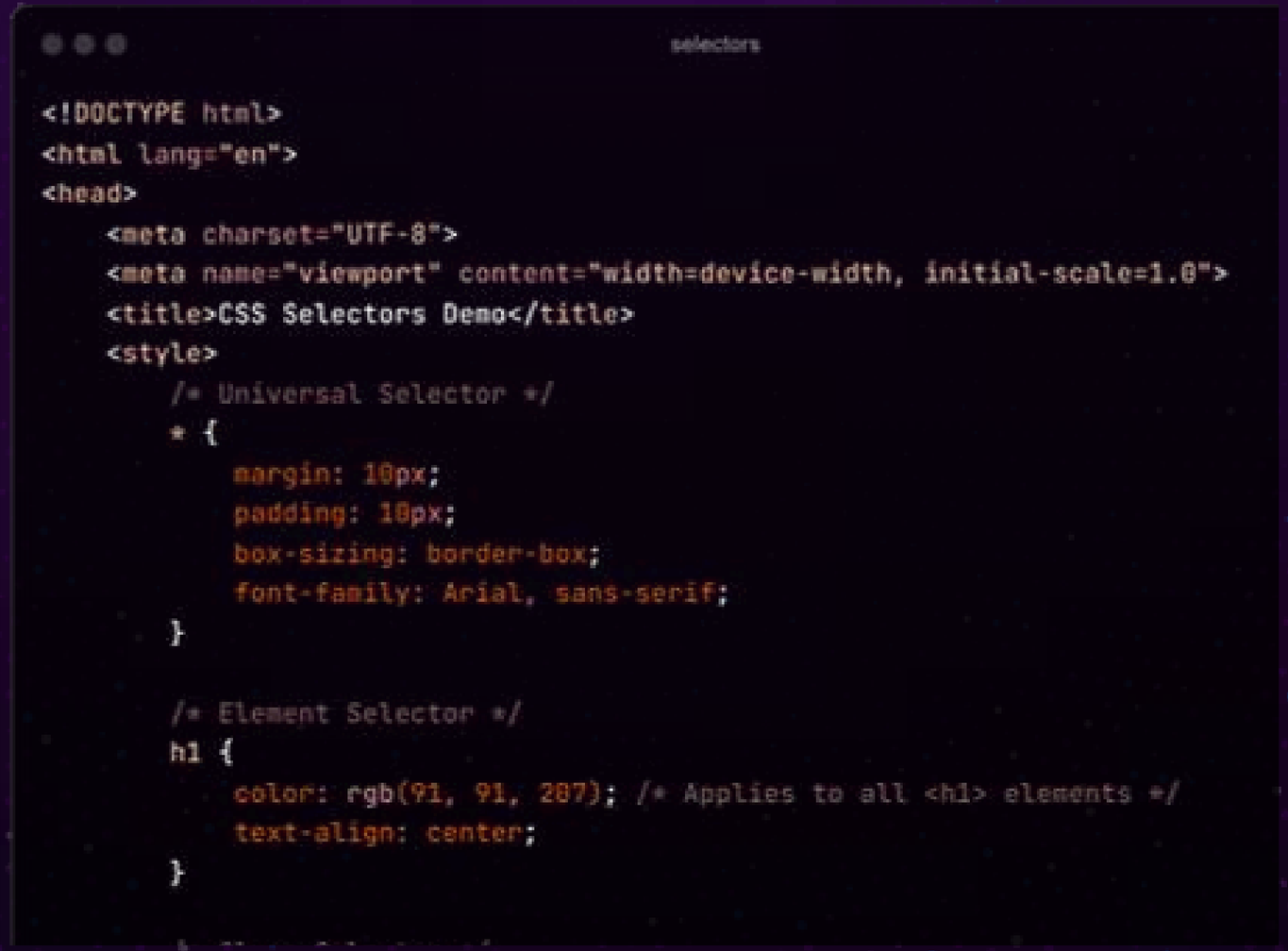
- Targets a single, unique element by its ID.

selectors

```css
/* ID Selector */
#header {
    background-color: lightgray;
    text-align: center;
}

/* Class Selector */
.highlight {
    color: orange;
    font-weight: bold;
}
```

CSS Selectors



CSS Selectors

```
        /* Class Selector */
        .highlight {
            background-color: rgb(144, 102, 127); /* Applies to all elements with
class="highlight" */
            padding: 10px;
        }

        /* ID Selector */
        #unique {
            color: rgb(214, 42, 42); /* Applies to the element with id="unique" */
            font-size: 1.5em;
        }
    </style>
</head>
<body>
    <h1>CSS Selectors Demonstration</h1>
    <p class="highlight">This paragraph has a pink background and padding.</p>
    <p id="unique">This paragraph has a unique style with red text and larger font
size.</p>
    <p>This is a regular paragraph.</p>
    <div>
        <p class="highlight">Another highlighted paragraph inside a div.</p>
    </div>
</body>
</html>
```

CSS Box Model

1. Content:
The actual content of the element, like text or images.

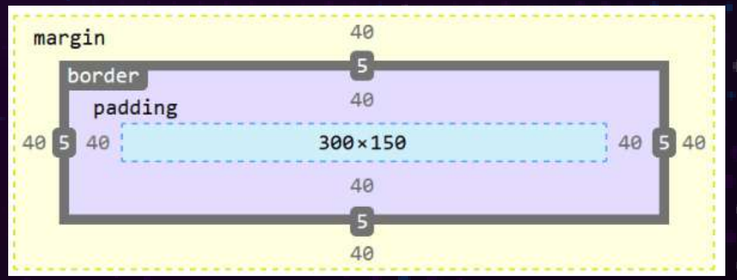
2. Padding:
Space between the content and the border.

3. Border:
Surrounds the padding and content.

4. Margin:
Space outside the border, separating the element from other elements.

Box Model

```
div {
  width: 300px;
  padding: 20px;
  border: 5px solid black;
  margin: 15px;
}
```

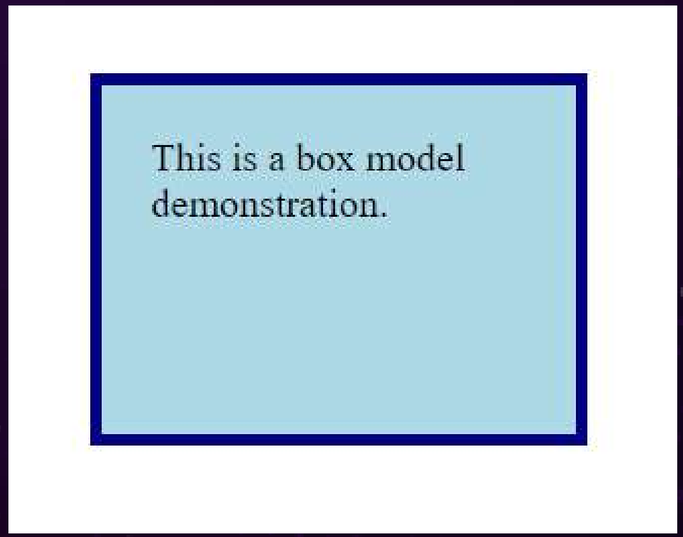


CSS Box Model

selectors

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CSS Box Model Demo</title>
    <style>

        .box {
            width: 200px;
            height: 150px;
            background-color: lightblue;
            padding: 20px;  /* Space between content and border */
            border: 5px solid navy; /* Border around the box */
            margin: 30px; /* Space outside the border */
            box-sizing: border-box;
        }
    </style>
</head>
<body>
    <div class="box">
        This is a box model demonstration.
    </div>
</body>
</html>
```



CSS Positioning

CSS Positioning controls the placement of elements on a webpage, using properties like static, relative, absolute, and fixed to define their location.

1. Static:

- Default positioning for elements.
- No special positioning; elements follow the normal document flow.

CSS Positioning

```css
.static-box {
  position: static;
  /* Default behavior, no
effect from top, right,
bottom, left, or z-index */
}
```

2. Fixed:

- Position relative to the viewport.
- Stays in place when scrolling.

CSS Positioning

```css
div {
    position: fixed;
    top: 0;
    left: 0;
}
```

CSS Positioning

3. Relative:

- Position relative to its normal position.

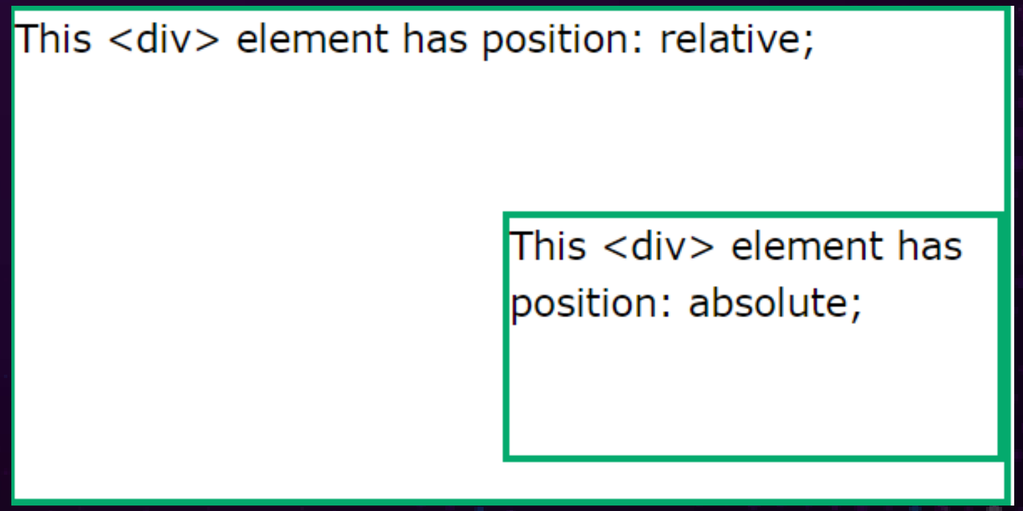
CSS Positioning

```css
div.relative {
  position: relative;
  width: 400px;
  height: 200px;
}

div.absolute {
  position: absolute;
  top: 80px;
  right: 0;
}
```

4. Absolute:

- Position relative to the nearest positioned ancestor.



Fonts and Text

Font Family:

Specifies the font of an element.

```
font-family: Arial, sans-serif;
```

Font Size:

Specifies the size of font.

```
font-size: 16px;
```

Font Weight:

Specifies the thickness of the font.

```
font-weight: bold;
```

Text Alignment:

Aligns the text horizontally (left, center, right, justify).

```
text-align: center;
```

Text Decoration:

Adds decoration to text (underline, overline, line-through).

```
text-decoration: underline;
```

Line Height:

Sets the space between lines of text.

```
line-height: 1.5;
```

CSS Flexbox

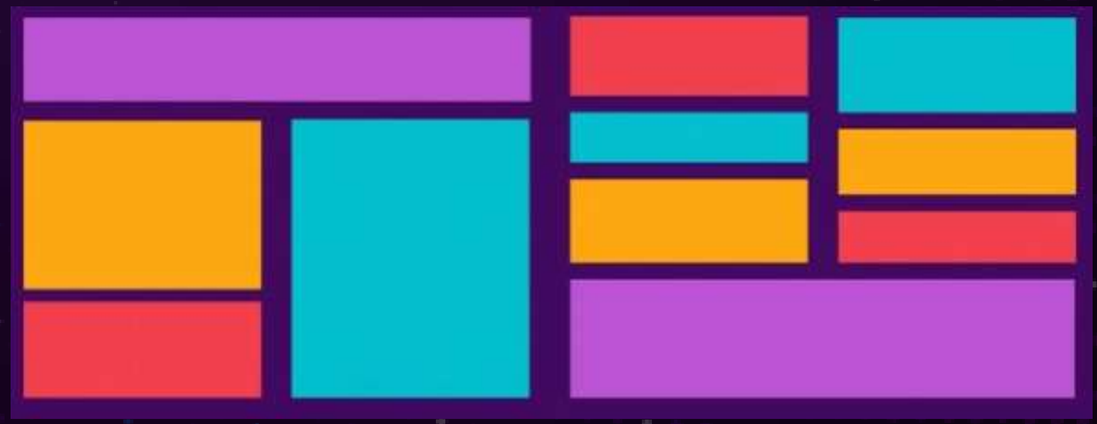
- A layout module for designing flexible and responsive layouts.
- Provides control over alignment, spacing, and distribution of items.

Properties:

- display: flex; - Enables flexbox on an element.
- flex-direction: - Defines the direction of the flex items (row, column).
- justify-content: - Aligns items horizontally (start, center, space-between).
- align-items: - Aligns items vertically (stretch, center, flex-start).

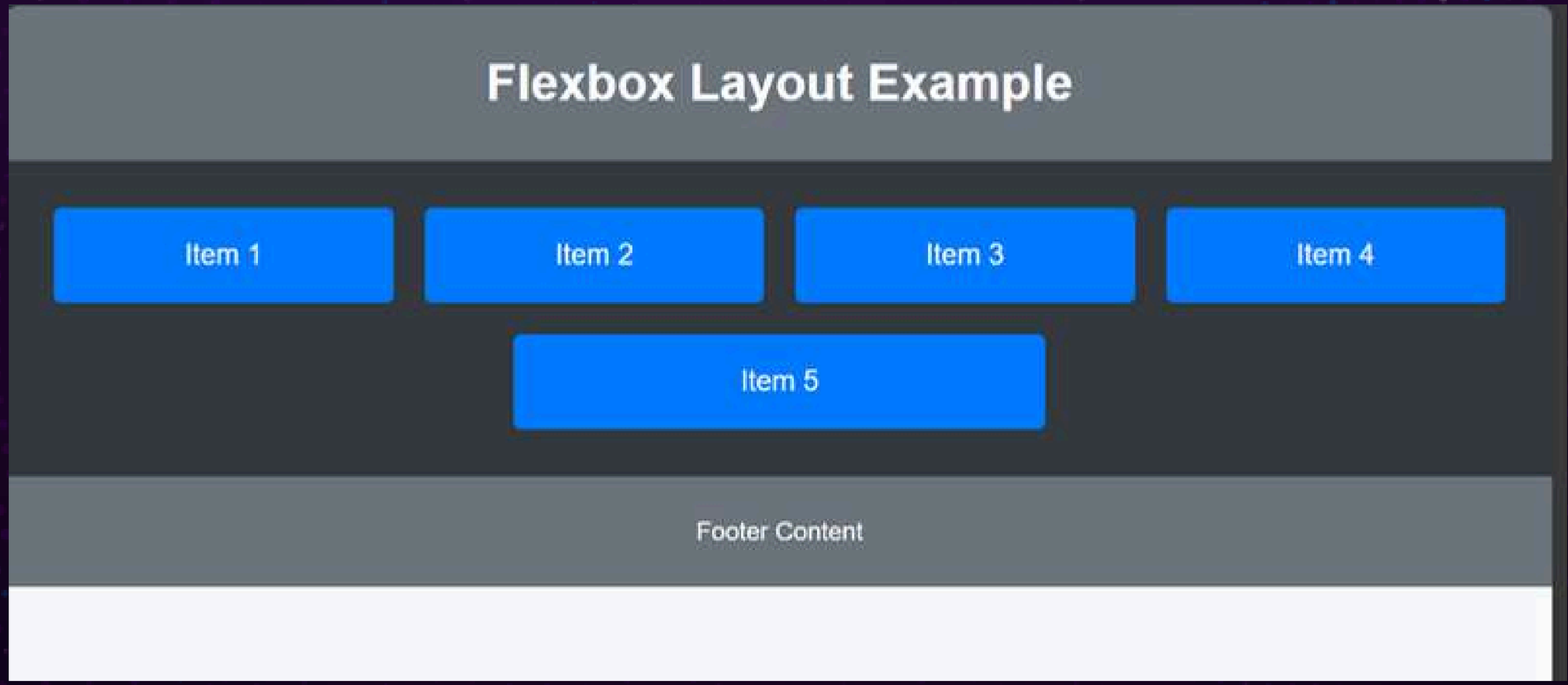
CSS Flexbox

```
.container {
    display: flex;
    flex-direction: row;
    justify-content:
center;
    align-items: center;
}
```



CSS Flexbox

CSS Flexbox


Flexbox Layout Example
Item 1
Item 2
Item 3
Item 4
Item 5
Footer Content

Transitions

CSS Transitions allow you to change property values smoothly over a given duration, instead of happening instantly.

1. transition-duration

Defines how long the transition should take.

```
transition-duration: 2s;
```

2. transition-timing-function:

Describes the speed curve of the transition.

```
transition-timing-function: ease-in-out;
```

3. transition-delay:

Specifies a delay before the transition starts.

```
transition-delay: 0.5s;
```

Transitions

```
button {
    width: 100px;
    transition: width 2s
ease-in-out;;
}
button: hover{
  width: 200px;
}
```

Animations

CSS Animations create smooth transitions between element states by defining keyframes and timing.

Keyframes

Define the start and end points of an animation, as well as possible intermediate points.

Animation Properties:

- animation-name
- animation-duration
- animation-timing-function
- animation-delay
- animation-iteration-count

Keyframes

```
@keyframes example {
    0% { background-color:
red; }
  100% { background-color:
yellow; }
}
```

Animations

```
div {
  animation-name: example;
  animation-duration: 4s;
  animation-delay: 0.5s;
  animation-iteration-
count: infinite;
}
```

Responsive Design

CSS Responsive Design adapts the layout and style of a website to different screen sizes and devices for optimal user experience across desktops, mobile devices..

1. Media Queries:

Allow you to apply different styles based on device characteristics like screen width, height, and orientation.

```
@media (max-width: 720px) {
  body {
    background-color: lightblue;
  }
}
```

The above code changes the background color of the webpage to light blue when the screen width is 720 pixels or less.

2. Viewport Meta Tag:

Sets the viewport width and scaling for mobile devices, ensuring the page scales correctly on different screens.

```
<meta name="viewport"
content="width=device-width,
initial-scale=1.0">
```

This tag ensures that the webpage is displayed at the correct width and scale on mobile devices.

JavaScript

Essential Concepts of JS

What is JavaScript

JavaScript is a powerful programming language used to create dynamic content on webpages. It's **interactive**, **responsive**, **popular** and used for **client** and **server** side scripting.

Internal JavaScript

Write JavaScript code directly in your HTML file using the <script> tag for small, closely linked scripts.

hello.html

```
<head>
    <script>
        function showMessage() {
            alert("Hello from Internal JS!");
        }
    </script>
</head>
```

External JavaScript

Create a separate .js file, such as script.js, and link it in your HTML with the <script> tag to keep your code modular.

hello.html

```
<head>
    <script src="script.js"></script>
</head>
```

Logging to Console

tutorial.js

```
console.log("Hello, World!"); // Output: Hello, World!
let name = "React";
console.log("Name:", name); // Output: Name: React
```

console.log() a widely used tool for debugging and tracking the flow of a program by printing out values

Understanding Scope

- **Global Scope** applies to the entire program
- **Local Scope** is on a functional level
- **Block Scope** is on a control structure level

tutorial.js

```
let globalVar = 'Global scope';

function sampleFunction() {
  let localVar = 'Local scope';

  if (condition){
    let blockVar = 'Block scope';
  }
}
```

JS Paradigms

1. Variables
2. Data Types
3. Operators
4. Control Structures
5. Functions
6. Objects and Arrays
7. Event Handling
8. Async JS

Primitive Datatypes

String: Represents textual data (a sequence of characters). Strings are created using single quotes ('...'), double quotes ("..."), or backticks for template literals (`...`).

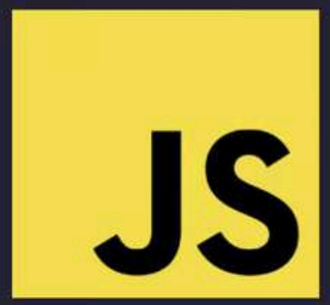
Number: Represents numeric values, including integers and floating-point numbers.

Boolean: Represents a logical entity and can have two values: **true** or **false**.

Undefined: Represents a variable that has been declared but has not yet been assigned a value.

Null: Represents the intentional absence of any object value. It's often used to indicate that a variable should be empty.

```
typeof("Hola") // string
typeof(12) // integer
typeof(true) // boolean
typeof(undefined) // undefined
typeof(null) // object
typeof({}) // object
typeof(Symbol()) // symbol
typeof(1n) // bigint
typeof(function(){}) // function
```



Non Primitive Datatypes

Arrays: An array is an ordered collection of elements, where each element is stored at a specific index. Arrays can store any type of data, including numbers, strings, objects, etc.

Common Array Methods

- push(element): Adds an element to the end of the array.
- pop(): Removes the last element from the array.
- shift(): Removes the first element from the array.
- unshift(element): Adds an element to the beginning of the array.
- length: Returns the number of elements in the array.
- forEach(): Iterates through an array using .forEach(function(element, index, array) { });

tutorial.js

```
let emptyArray = [];
let fruits = ["apple", "banana", "cherry"];

console.log(fruits[0]); // Output: "apple"

fruits[1] = "blueberry";
console.log(fruits); // Output: ["apple", "blueberry", "cherry"]
```

Non Primitive Datatypes

Objects: An object is a collection of **key-value pairs**, where each key (also called a property) is associated with a value.

Object Dereferencing

tutorial.js

```
let emptyArray = [];
let fruits = ["apple", "banana", "cherry"];

console.log(fruits[0]); // Output: "apple"

fruits[1] = "blueberry";
console.log(fruits); // Output: ["apple", "blueberry", "cherry"]
```

Object Dereferencing

tutorial.js

```
const person = { name: "Alice", age: 25 };

const { name, age } = person;

console.log(name); // Output: "Alice"
```

Variables and Declarations

var: Creates a variable at a global ad local scope and allows reassignment and redeclaration.

let: Creates a variable at a global, local and block scope and allows reassignment and redeclaration as a part of ES6.

const: .Creates a variable at a global, local and block scope and **does not** allow reassignment and redeclaration.

tutorial.js

```
var color = "White";
let num1 = 10;
const PI = 3.14159265;
```

Operators

Arithmetic Operators:
Perform arithmetic operations (+, -, *, /, %).

Assignment Operators:
Assign values to variables (=, +=, -=, *=, /=).

Comparison Operators:
Compare values (==, ===, !=, !==, >, <, >=, <=).

Logical Operators:
Combine or modify conditions (&&, ||, !).

tutorial.js

```
let a = 5;
let b = 10;
let sum = a + b;
let isEqual = (a === b);
```

Ternary Operators

The ternary operator in JavaScript is a shorthand way to write an if-else statement as:

condition ? expressionIfTrue : expressionIfFalse;

component.jsx

```jsx
let age = 20;
const Component = () => {

  return age > 10 ? <div> Name1 </div> : <div> Name2 </div>
}
```

Spread Operator

The spread operator is used to "spread" or expand the elements of an array, object, or other iterable into individual elements.

JS Paradigms

tutorial.js

```jsx
let numbers = [1, 2, 3];
let moreNumbers = [...numbers, 4, 5, 6];

console.log(moreNumbers); // Output: [1, 2, 3, 4, 5, 6]
```

If-Else Statement

The **if-else** statement is used to execute a block of code based on a condition. If the condition evaluates to true, the if block is executed. If the condition is false, the else block is executed (if provided).

tutorial.js

```
if (condition) {
    // Code to run if the condition is true
} else if {
    // Code to run other condition is true
} else {
    // Code to run if the condition is false
}
```

Looping Statements

- **for Loop:** Best for running a loop a specific number of times.
- **while Loop:** Runs as long as a condition is true. Ideal when the number of iterations isn't known beforehand.
- **do...while Loop:** Similar to while, but ensures the loop runs at least once.
- **for...in Loop:** Used for iterating over object properties or array indices, but be cautious when using it with arrays.
- **for...of Loop:** Used for iterating over iterable objects (like arrays) and directly accesses the values.

tutorial.js

```
for (initialization; condition; increment) {
    // Code to be executed
}

while (condition) {
    // Code to be executed
}

for (key in object) {
    // Code to be executed
}

for (value of iterable) {
    // Code to be executed
}
```

Functions and Arrow Functions in JS

- **Arrow** functions are more concise.
- **Traditional** functions use the function keyword and are more verbose.

tutorial.js

```
function traditionalFunction() {
    console.log(arguments);
}

traditionalFunction(1, 2, 3); // Output: [1, 2, 3]

const arrowFunction = (...args) => {
    console.log(args);
};

arrowFunction(4, 5, 6); // Output: [4, 5, 6]
```

Event Handling in JS

Key Concepts:

- **Events:** Actions like clicks, key presses, or page loads.
- **Event Listener:** A function that waits for an event to occur.

Common Event Types:

- **Mouse Events:** click, mouseover
- **Keyboard Events:** keydown, keyup
- **Form Events:** submit, change

tutorial.html

```html
<html>
  <body>
    <button id="myButton">Click Me</button>
    <p id="message"></p>

    <script>
        // Select elements
        const button = document.getElementById('myButton');
        const message = document.getElementById('message');

        // Add event listener
        button.addEventListener('click', function(event) {
            message.textContent = `Button clicked! Event type:
${event.type}`;
        });
    </script>
  </body>
</html>
```

Async, Await, Fetch and Promises

Promises:

- A **Promise** represents a value that may be available now, or in the future, or never.
- It has three states:
  - **Pending:** Initial state, neither fulfilled nor rejected.
  - **Fulfilled:** Operation completed successfully.
  - **Rejected**: Operation failed.

fetch: A modern API for making network requests, returns a Promise. Useful for fetching data from APIs.

async Functions: Declared with async keyword. Always return a Promise and makes asynchronous code easier to write and read.

await Expression: Used inside async functions and waits for the Promise to resolve and returns the result and makes asynchronous code appear synchronous.

Installing Node.js

Node.js Official Website

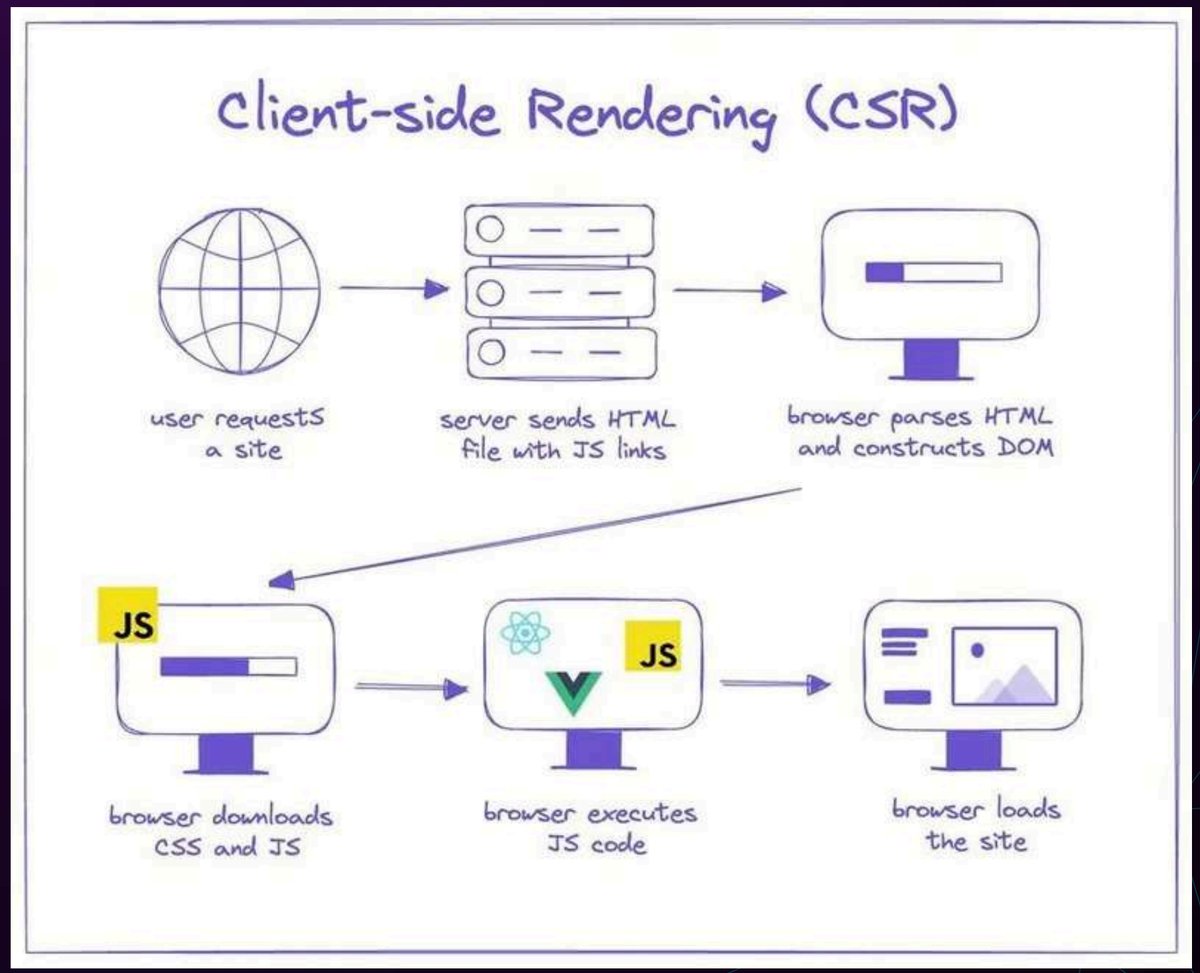
- Run *node -v* to check the Node.js version.
- Run *npm -v* to check the npm (Node Package Manager) version.

React

Introduction to React

What is React?

- Arose from **necessity and evolution**, through the rise of native apps
- React is an open source JavaScript library used to build user interfaces.
- When a webpage is created using react, we call it a “React App”.
- It allows developers to create large web applications that can update and render efficiently in response to data changes.
- It has a **component based architecture**, this makes code highly reusable, modular and easy to maintain.
- It uses a **Virtual DOM** to improve the performance by efficiently updating the real DOM.



Why’s React so popular?

Why is React so popular?

- Flexibility: Can be used for web, mobile (React Native), and desktop applications.
- Declarative Style: Developers focus on describing the desired UI, and React handles the updates.
- Community Support: The large and active community surrounding React contributes to a wealth of libraries, tools, and resources, facilitating easier problem-solving and innovation.

Component-based nature

React is component-based: A component in React is a reusable piece of code that represents a part of your user interface. They act as building blocks that you can use for building complex UIs. With React components the code becomes much more flexible and structured.

For example, in the code, each component (MyAwesomeNavbar, MainContent, MyAwesomeFooter) encapsulates its own structure and behavior where the entire webpage is composed of three custom components.

```
<body>
    <MyAwesomeNavbar />
    <MainContent />
    <MyAwesomeFooter />
</body>
```

Installation

To start React installation, it is required to install Node first because React.js is a JavaScript library, and Node.js is a JavaScript runtime environment that allows you to run JavaScript on the server side.

To install Node, navigate to the Node.js website , download the current version and install it on device. Now, type in the below code to check the version and confirm the installation

```
node -v
```

Version details


```sh
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Alex>node -v
v14.17.0

C:\Users\Alex>
```

Installing React using Vite:

In the command prompt window, navigate to the directory that you want to use in creating your React project. To do this, type the below then click enter.

cd Documents

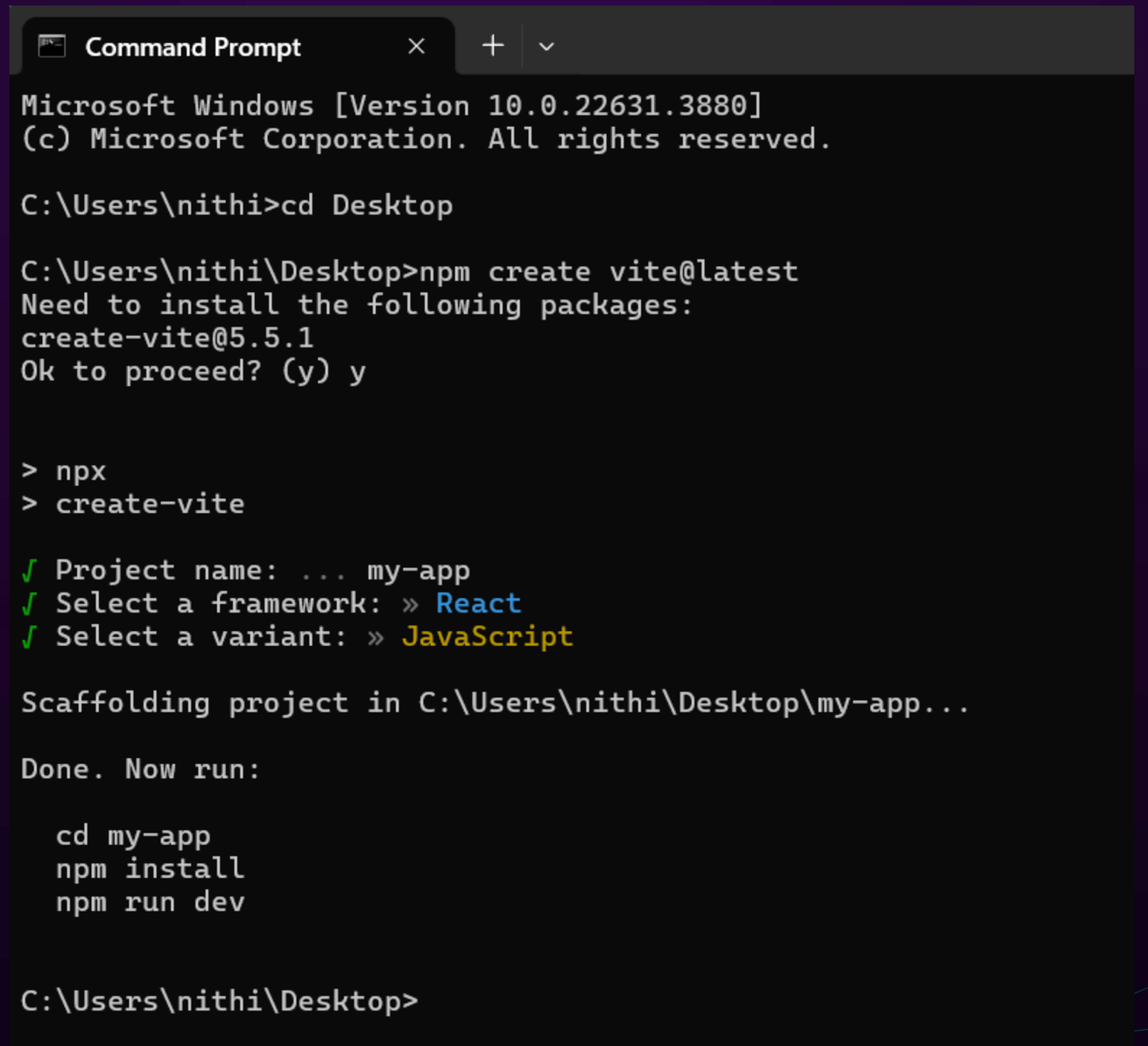
Type **mkdir [folder name]** then navigate to the newly created directory using

cd [folder name]

Now type in the below code to create the react app followed by desired App Name.

npm create vite@latest

Code Sample


Command Prompt
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.
C:\Users\nithi>cd Desktop
C:\Users\nithi\Desktop>npm create vite@latest
Need to install the following packages:
create-vite@5.5.1
Ok to proceed? (y) y
> npx
> create-vite
√ Project name: ... my-app
√ Select a framework: » React
√ Select a variant: » JavaScript
Scaffolding project in C:\Users\nithi\Desktop\my-app...
Done. Now run:
cd my-app
npm install
npm run dev
C:\Users\nithi\Desktop>

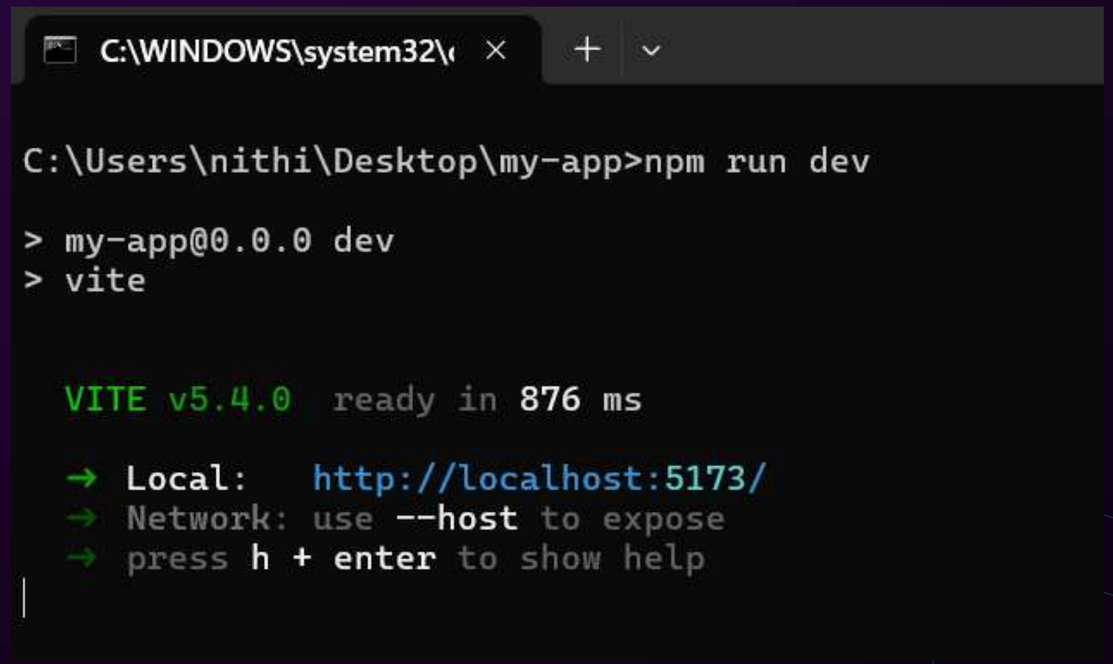
Running a React App

Once the installation is completed, type in the below commands to navigate to and start the react app:

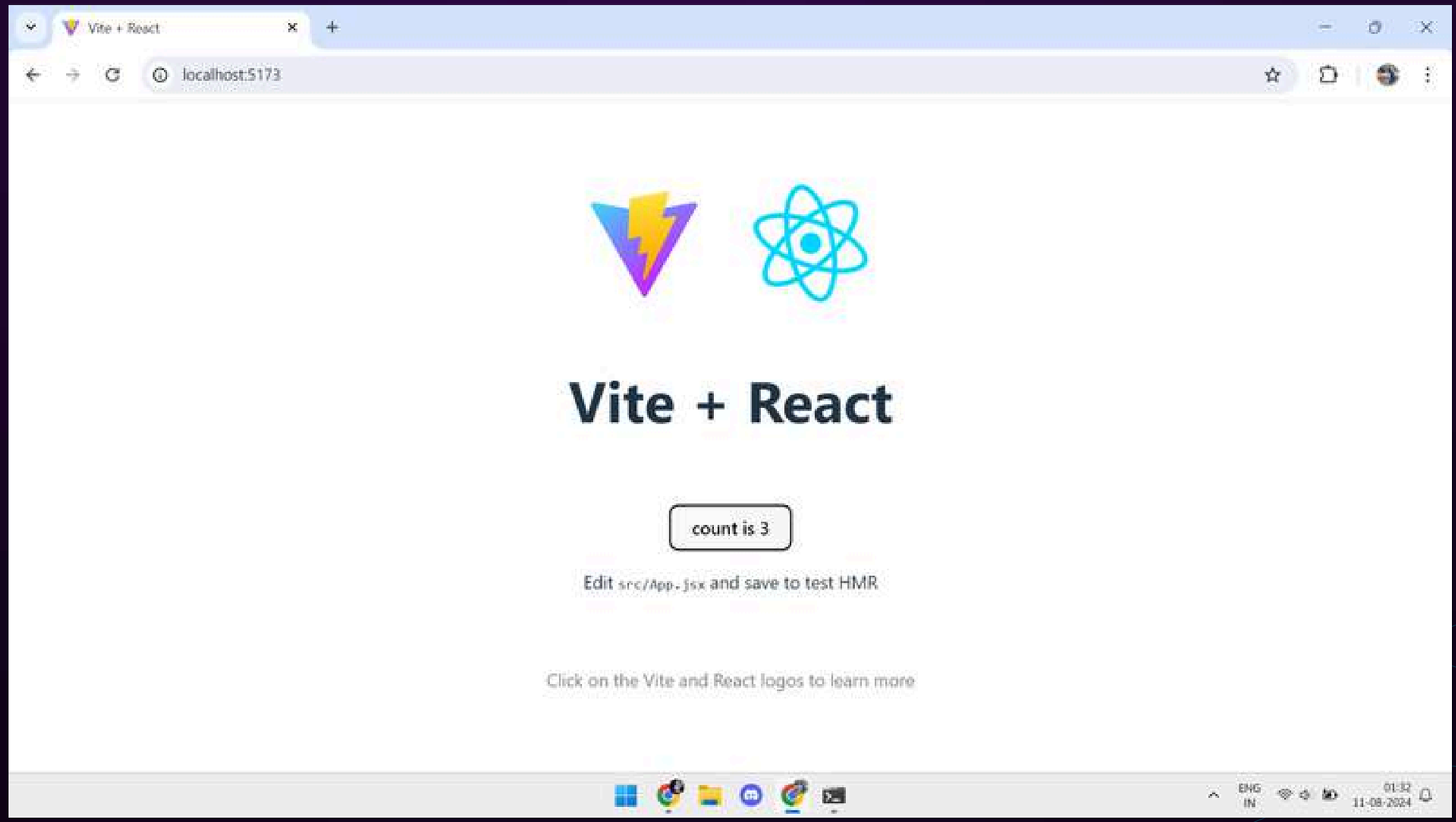
cd my-app

npm run dev

Click on the localhost link to open the react app.



OUTPUT


Vite + React
localhost:5173
Vite + React
count is 3
Edit src/App.jsx and save to test HMR
Click on the Vite and React logos to learn more

JSX

JSX (JavaScript XML) is a syntax extension for JavaScript that looks similar to HTML, which makes writing React components more intuitive and easier. JSX enables developers to embed HTML-like code directly within JavaScript, allowing for a more declarative way to create UI elements. JSX gets transpiled into React.createElement calls, which create the elements that React uses to build the DOM

Syntax:

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

Implementation:

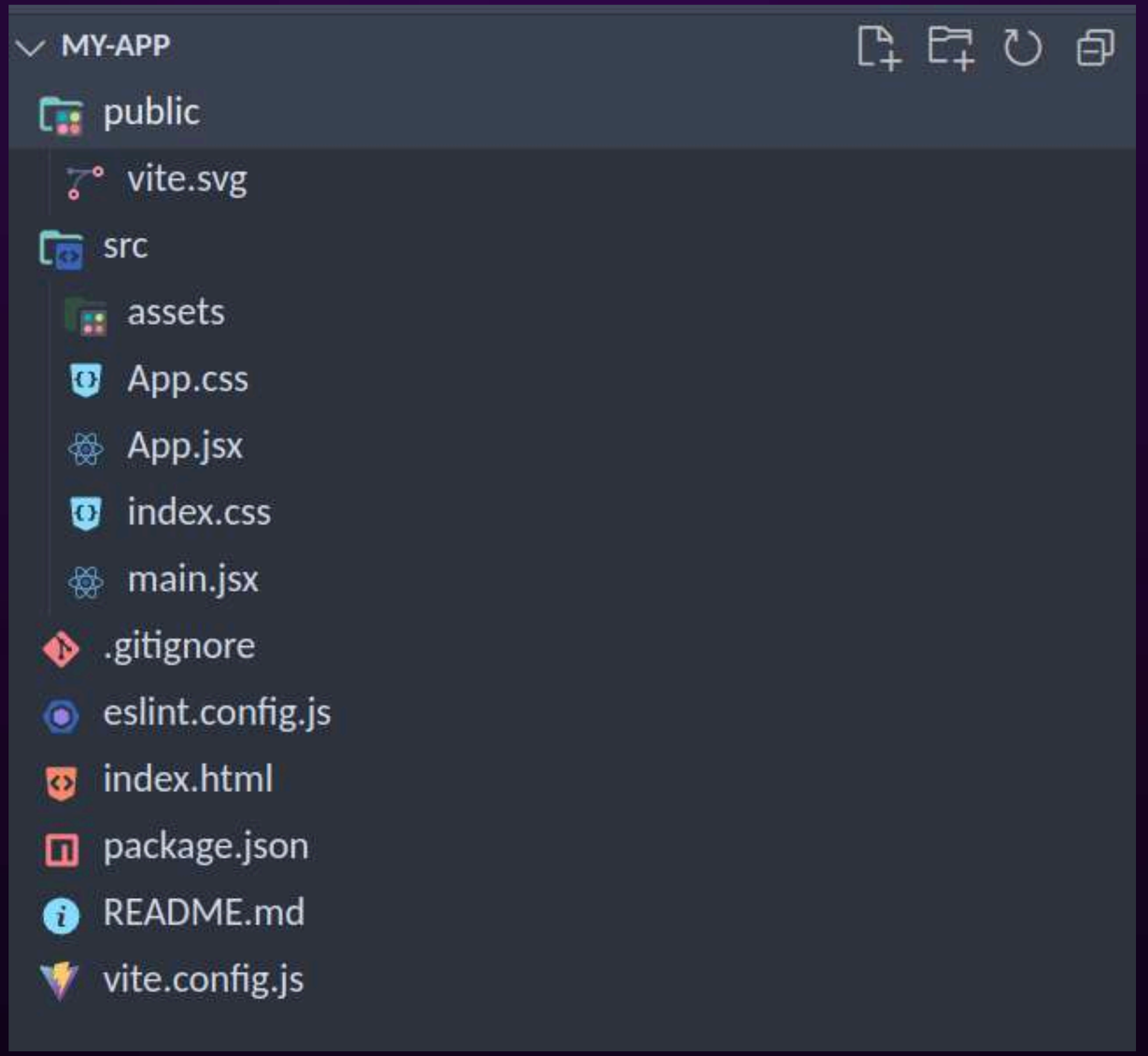
```
import React from 'react';
const App = () => {
 return <h1>Hello, world!</h1>;
};
export default App;
```

React Arrow Function

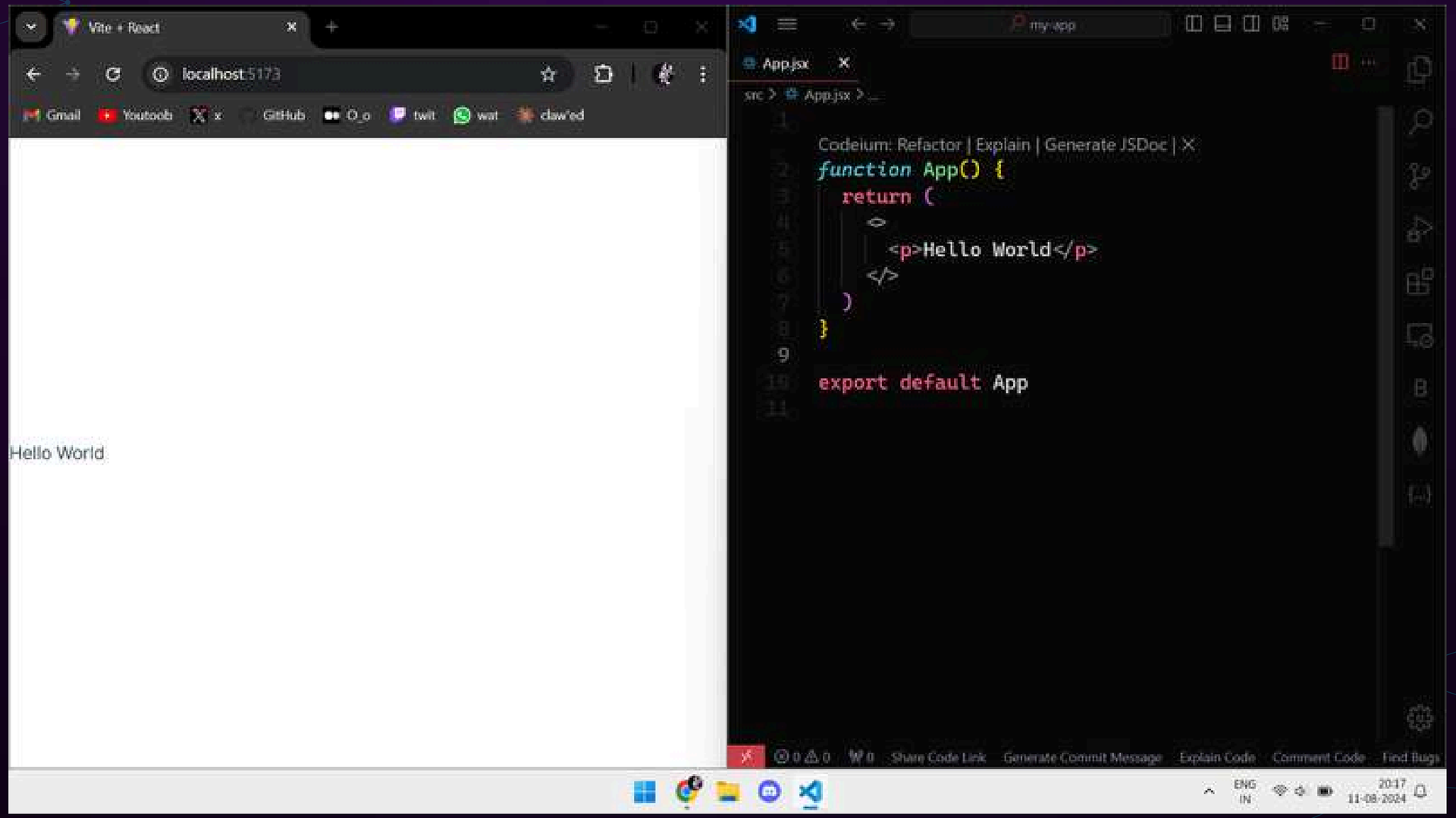
```
import React from 'react';

 const ComponentName = () => {
    return (
      <div>
            {/* JSX goes here */}
      </div>
    );
 };
export default ComponentName;
```

Project Structure



Code – App.jsx


Vite + React
localhost:5173
Gmail
Youtoob
x
GitHub
O_o
twit
wat
claw'ed
Hello World
my-app
App.jsx
src > App.jsx > ...
Codeium: Refactor | Explain | Generate JSDoc |
function App() {
return (
<>
Hello World
</>
)
}
export default App
Share Code Link
Generate Commit Message
Explain Code
Comment Code
Find Bugs
ENG
IN
20:17
11-08-2024

DOM

React uses a Virtual DOM to optimize rendering

But what is DOM?

The DOM is a tree-like structure representing the HTML elements of a webpage. Whenever a change occurs, the browser must update this structure, leading to performance issues.

This process involves recalculating styles, rearranging elements, and repainting the screen, which can be slow, especially for complex UIs.

To address this, React introduced the Virtual DOM.

Virtual DOM

React employs a V**irtual DOM** to optimize performance. Instead of directly manipulating the actual DOM, React creates a lightweight, in-memory copy. When changes occur, React updates this virtual representation and then efficiently determines the minimal changes needed to synchronize with the real DOM.

This approach avoids unnecessary DOM manipulations, resulting in faster and smoother updates.

Changes to the Virtual DOM are fast because they are performed in memory and do not involve any direct browser operations.

Virtual DOM

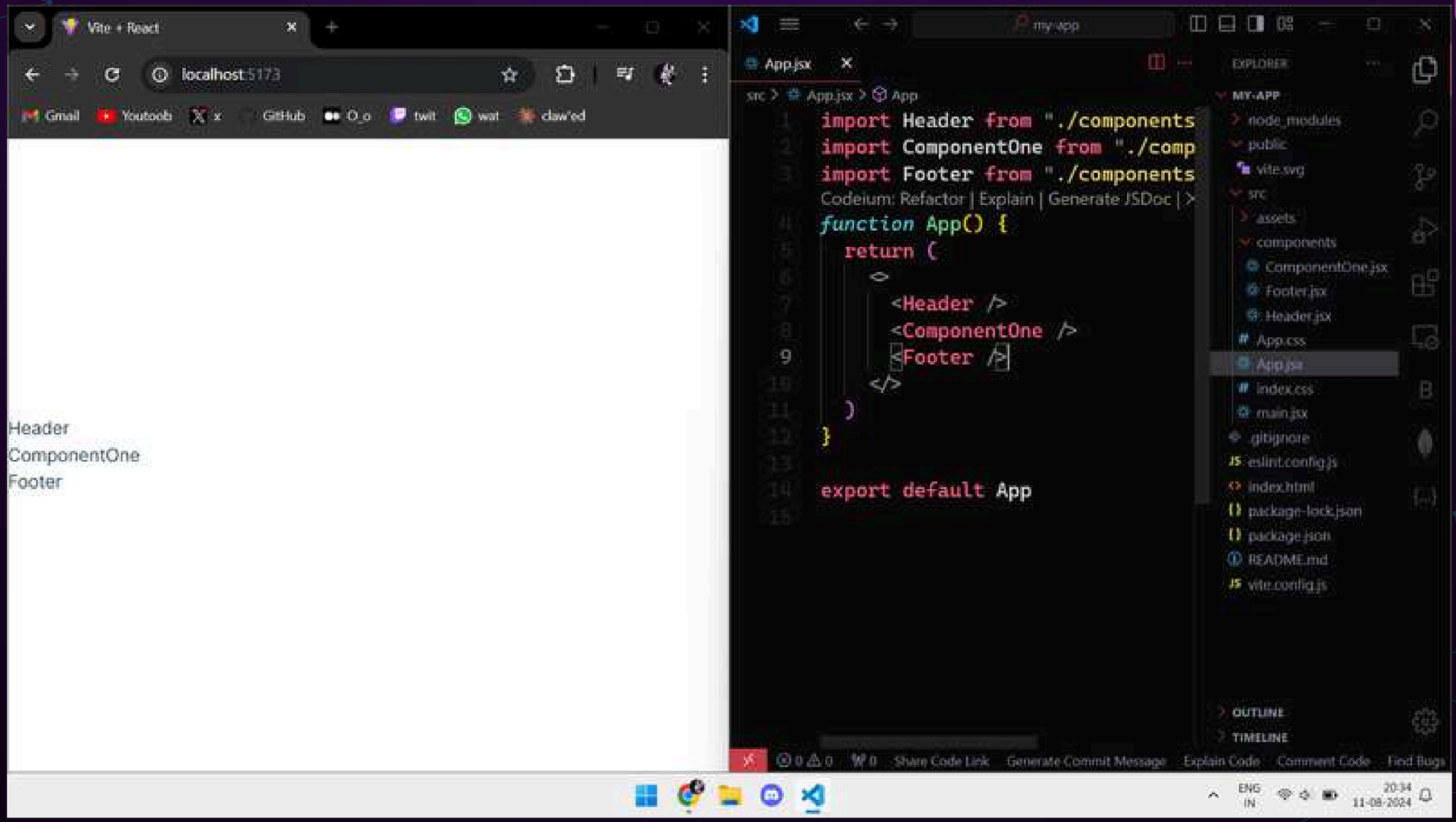
How the Virtual DOM Works

1. **Initial Render:** When a React component is first rendered, a Virtual DOM tree is created, representing the structure of the UI.
2. **State/Props Update:** When the state or props of a component change, a new Virtual DOM tree is created, representing the updated UI.
3. **Diffing:** React compares the new Virtual DOM tree with the previous one to identify what has changed. This process is called "diffing."
4. **Reconciliation:** Based on the differences found, React updates the actual DOM with the minimal set of changes required, ensuring efficient updates and rendering.

Components and Props

- Create a components folder in the app directory
- Now create the files which contains the components
- Insert the components wherever you want to use them in your project.
- Props can be used to send data and event handlers from child component to the parent components.
- Components are called with self closing tags with component name.

Using a Component



Sending Props to the Component

App.jsx

src > App.jsx > App

```jsx
import SendingProps from "./components/SendingPr

Codeium: Refactor | Explain | Generate JSDoc | ×
function App() {
  return (
    <>
      <SendingProps name={"blue"}/>
    </>
  )
}

export default App
```

SendingProps.jsx

src > components > SendingProps.jsx > SendingProps

```jsx


Codeium: Refactor | Explain | Generate JSDoc | ×
const SendingProps = ({name}) => {
  return (
    <div>Color is {name}</div>
  )
}

export default SendingProps
```

Hooks (useState and useEffect)

1. **Hooks,** introduced in React 16.8, allow functional components to manage state and side effects.

2. The **useState** hook lets you add and manage state in functional components without class components.

3. State can be initialized with a default value, and you get the current state along with a function to update it.

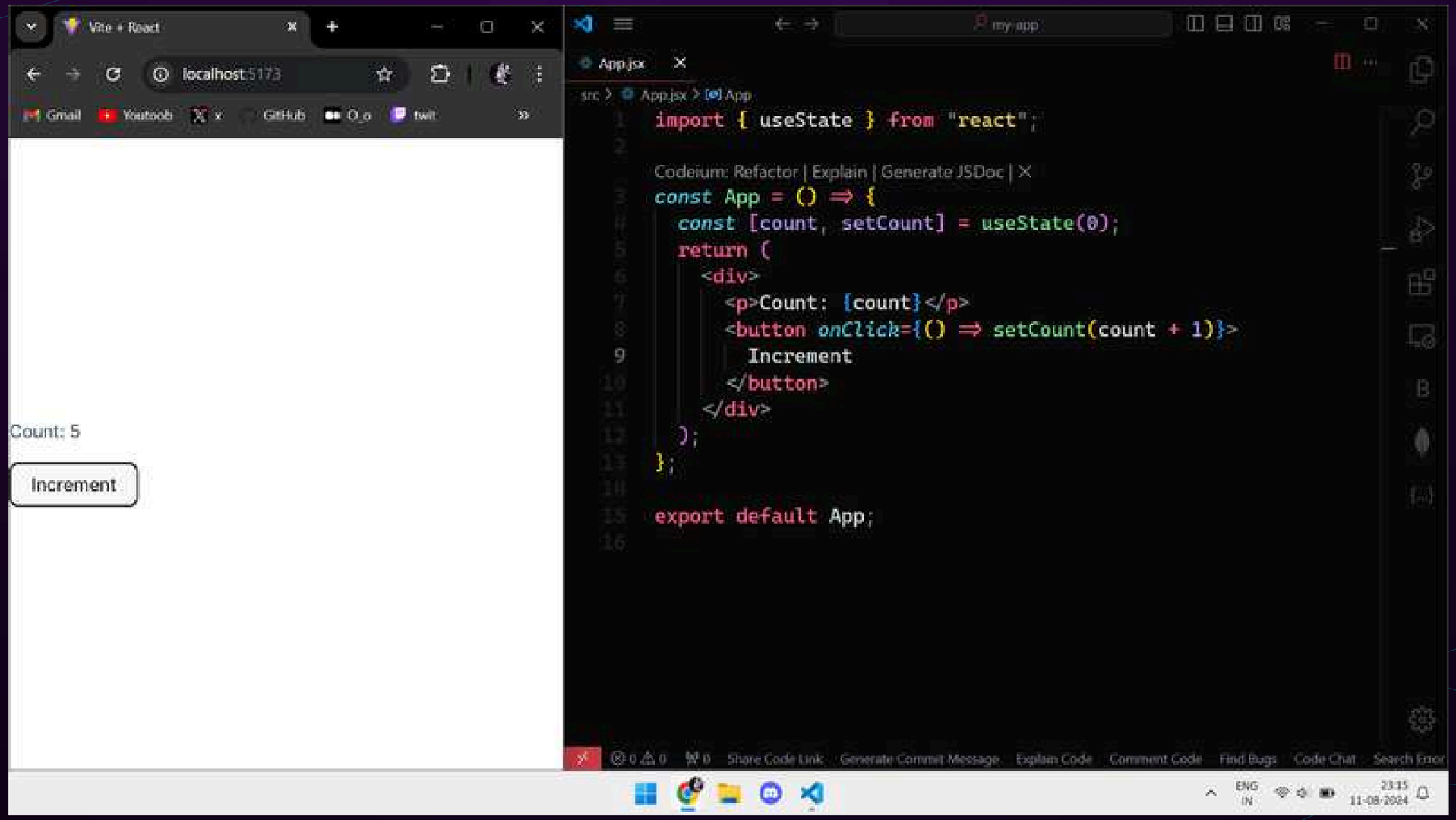
4. The **useEffec**t hook manages side effects such as data fetching, event subscriptions, and DOM manipulations in functional components.

5. You can specify dependencies in the useEffect hook to control when the effect should re-run.

```
Syntax for useState:
const [state, setState] =
useState(initialState);


Syntax for useEffect:
useEffect(() => {
  // Side effect code here
}, [dependencies]);
```

Code(useState)



Code(useEffect)

App.jsx

src > App.jsx > App > useEffect() callback

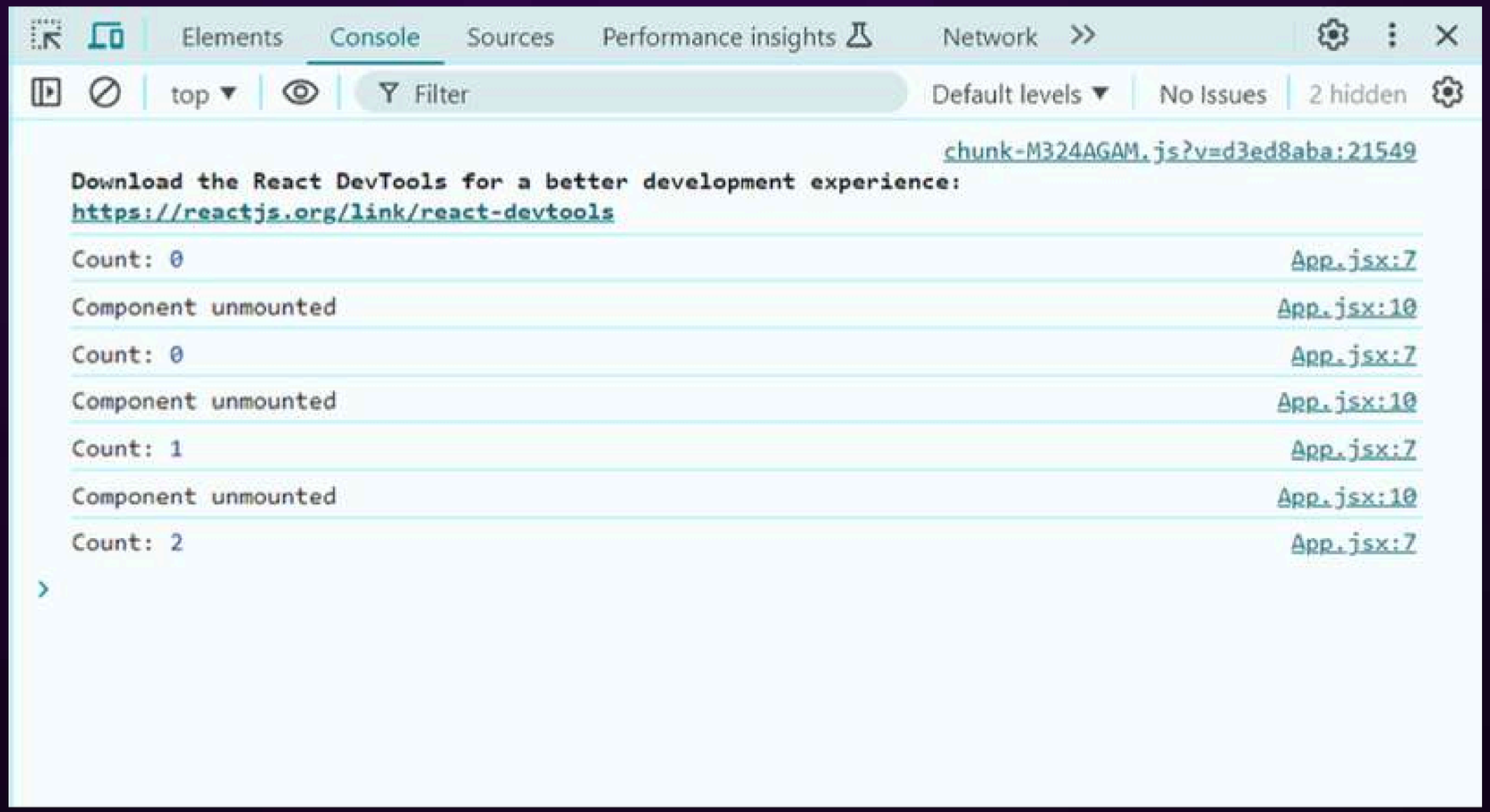
```jsx
import { useState,useEffect } from "react";

Codeium: Refactor | Explain | Generate JSDoc | ×
const App = () => {
  const [count, setCount] = useState(0);
  useEffect(() => {
    console.log('Count:', count);

    return () => {
      console.log('Component unmounted');
    };
  }, [count]);
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};

export default App;
```

Code(useEffect)



Conditional Rendering

1. **Conditional rendering** in React lets you display different UI elements based on specific conditions.

2. It is useful for dynamically displaying elements depending on the application's state or user interactions.

3. JavaScript expressions, such as the ternary operator, logical AND (&&), or helper functions, can be used to determine what to render.

4. This approach improves user experience by showing relevant content based on the current state.

5. Examples include displaying a login button when the user is logged out or a welcome message when logged in.

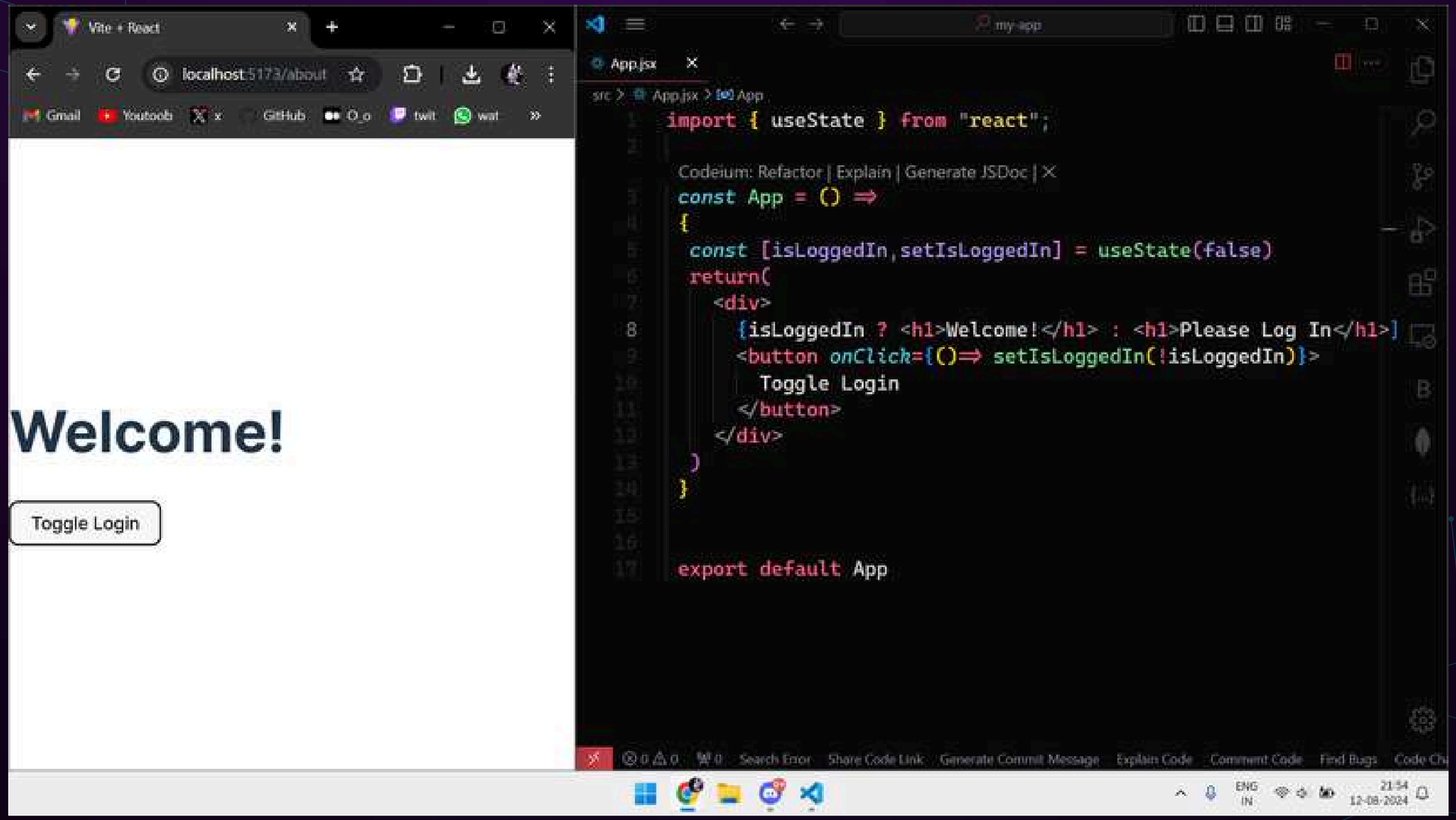
Syntax:

```
{condition ? <ComponentA /> : <ComponentB />}
```

Implementation:

```
import React, { useState } from 'react';
const App = () => {
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  return (
    <div>
      {isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please log in.
</h1>}
      <button onClick={() => setIsLoggedIn(!isLoggedIn)}>
        Toggle Login
      </button>
    </div>
  );
};
```

Code


Vite + React
localhost:5173/about
Gmail
Youtoob
x
GitHub
O_o
twit
wat
Welcome!
Toggle Login
my-app
App.jsx
src > App.jsx > App
import { useState } from "react";
Codeium: Refactor | Explain | Generate JSDoc | ×
const App = () =>
{
const [isLoggedIn,setIsLoggedIn] = useState(false)
return(

{isLoggedIn ? <h1>Welcome!</h1> : <h1>Please Log In</h1>}
<button onClick={()=> setIsLoggedIn(!isLoggedIn)}>
Toggle Login
</button>

)
}
export default App
Search Error
Share Code Link
Generate Commit Message
Explain Code
Comment Code
Find Bugs
ENG
IN
21:54
12-08-2024

User input forms

1. React forms allow you to collect and manage user input efficiently by controlling the form elements through state.

2. Controlled components in React use state to manage the value of form elements like inputs, text areas, and selects.

3. Handling form submission in React involves capturing the form data, often through an event handler like `onSubmit`.

4. Validation logic can be implemented to check the user input before form submission, ensuring data integrity.

5. Forms in React can be styled and structured dynamically, allowing for a customizable user input experience.

Code

App.jsx × UserForm.jsx ×

src > UserForm.jsx > default

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

Codeium: Refactor | Explain | Generate JSDoc | ×
function UserForm() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  Codeium: Refactor | Explain | Generate JSDoc | ×
  const handleSubmit = (event) => {
    event.preventDefault();
    console.log('Form submitted:', { name, email, password });
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={(event) => setName(event.target.value)} />
      </label>
      <br />
      <label>
        Email:
        <input type="email" value={email} onChange={(event) => setEmail(event.target.value)} />
```

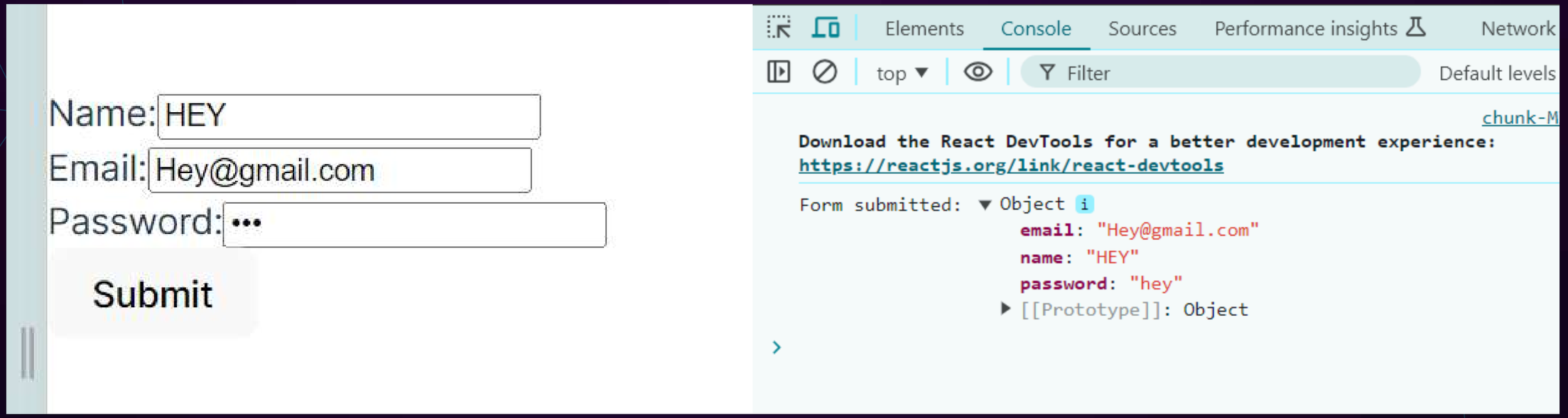
Code

```
        <label>
          Name:
          <input type="text" value={name} onChange={(event) => setName(event.target.value)} />
        </label>
        <br />
        <label>
          Email:
          <input type="email" value={email} onChange={(event) => setEmail(event.target.value)} />
        </label>
        <br />
        <label>
          Password:
          <input type="password" value={password} onChange={(event) => setPassword(event.target.value)} />
        </label>
        <br />
        <button type="submit">Submit</button>
      </form>
    );
}

export default UserForm;
```

Form Data

Console



React Router

1. React Router DOM is a library for handling routing in React applications, enabling navigation between components or pages.

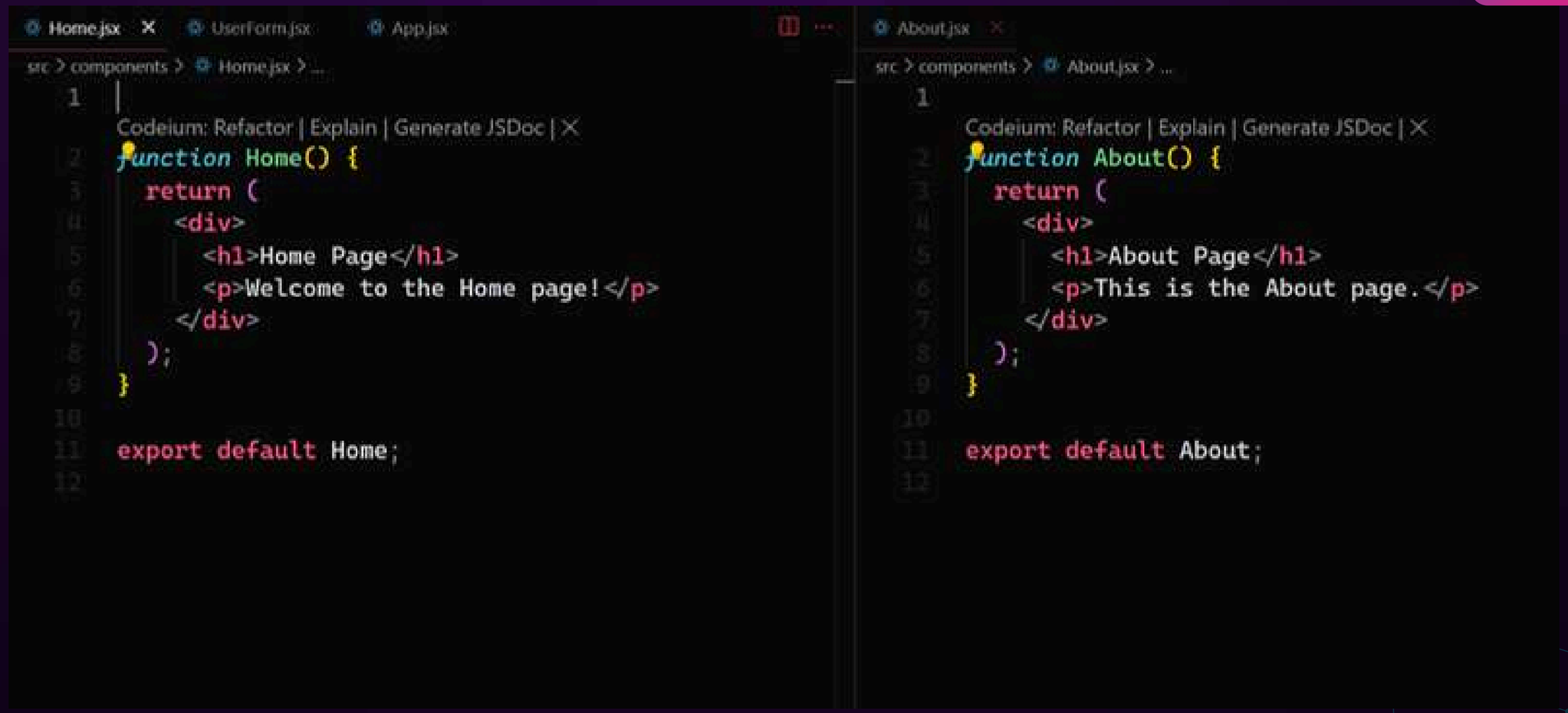
2. It allows developers to define routes within their application and link to these routes from components.

3. The library uses a declarative approach, making route management straightforward.

4. Routes are defined using specific components provided by React Router DOM.

5. React Router DOM automatically renders the appropriate component when the URL matches a defined route's path.

Code


Home.jsx
UserForm.jsx
App.jsx
src > components > Home.jsx > ...
Codeium: Refactor | Explain | Generate JSDoc |
function Home() {
return (

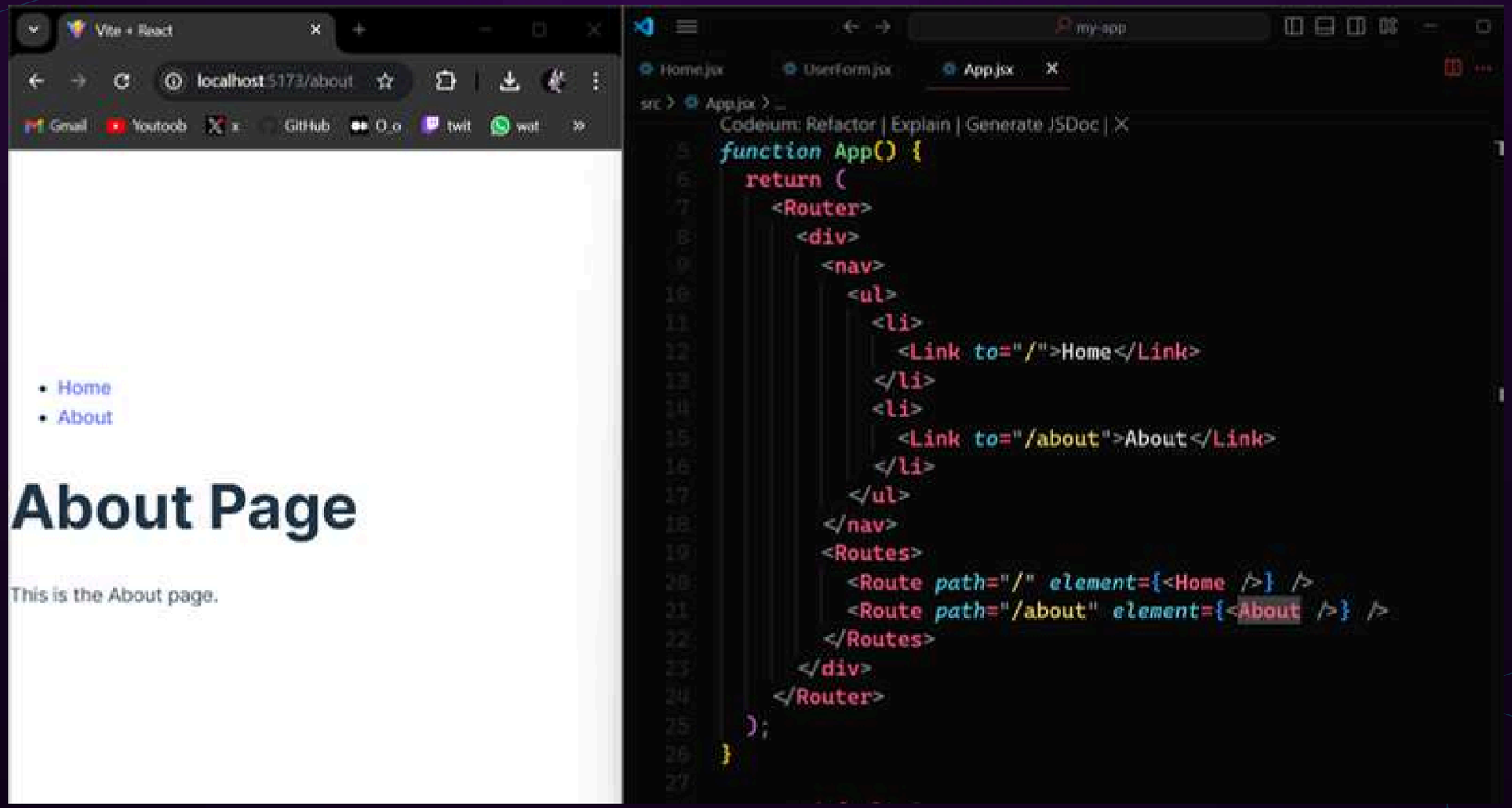
<h1>Home Page</h1>
Welcome to the Home page!

);
}
export default Home;
About.jsx
src > components > About.jsx > ...
Codeium: Refactor | Explain | Generate JSDoc |
function About() {
return (

<h1>About Page</h1>
This is the About page.

);
}
export default About;

Code


Vite + React
localhost:5173/about
Gmail
Youtoob
x
GitHub
O_o
twit
wat
Home
About
About Page
This is the About page.
my-app
Home.jsx
UserForm.jsx
App.jsx
src > App.jsx > ...
Codeium: Refactor | Explain | Generate JSDoc |
function App() {
return (
<Router>

<nav>

<Link to="/">Home</Link>

<Link to="/about">About</Link>

</nav>
<Routes>
<Route path="/" element={<Home />} />
<Route path="/about" element={<About />} />
</Routes>

</Router>
);
}

API

Application Programming Interface

What is an API?

APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols.

API stands for Application Programming Interface. In the context of APIs, the word Application refers to any software with a distinct function. Interface can be thought of as a contract of service between two applications. This contract defines how the two communicate with each other using requests and responses. Their API documentation contains information on how developers are to structure those requests and responses.

Fetching data from random APIs

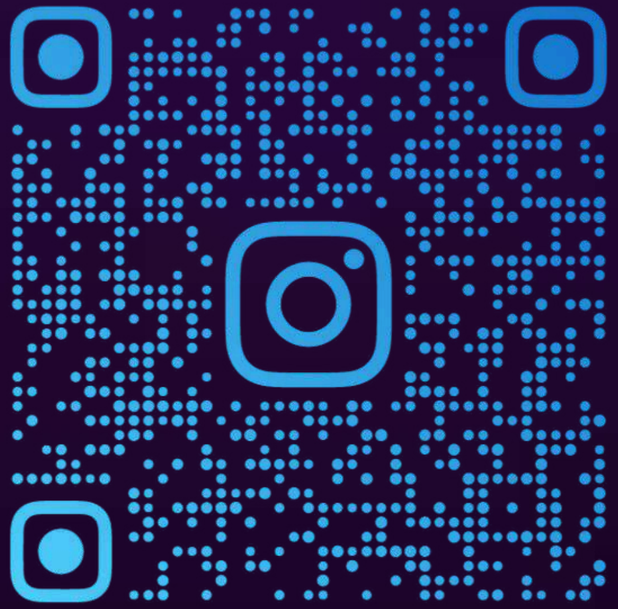
Here, we will be using https://randomuser.me/ to implement this. This is a free and open source API for generating random user data.

How to use:
You can use AJAX to call the Random User Generator API and will receive a randomly generated user in return. We will do this using the fetch() method:

Code

```
const apiUrl = 'https://randomuser.me/';
// Make a GET request
fetch(apiUrl)
 .then(response => {
 if (!response.ok) {
 throw new Error('Network response was not ok');
 }
 return response.json();
 })
 .then(data => {
 console.log(data);
 })
 .catch(error => {
 console.error('Error:', error);
 });
```

Follow our Socials



@CBITOSC