A REPORT OF FOUR WEEK TRAINING

At

VproTech Digital

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD

OF THE DEGREE OF

**BACHELOR OF TECHNOLOGY**

(Information Technology)

July 2024


**SUBMITTED BY:**

JAY KUMAR

URN:2203844

DEPARTMENT OF INFORMATION TECHNOLOGY

GURU NANAK DEV ENGINEERING COLLEGE LUDHIANA

(An Autonomous College Under UGC ACT)

# CERTIFICATE

JAY KUMAR

URN: 2203844

CRN 2221063

## CANDIDATE'S DECLARATION

I "JAY KUMAR" hereby declare that I have undertaken four week training "**VproTech Digital"** during a period from **5 June 2024** to **16 July 2024** in partial fulfillment of requirements for the award of degree of B.Tech (Information Technology) at GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA. The work which is being presented in the training report submitted to Department of Information Technology at GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA is an authentic record of training work.

Signature of the Student

The four week industrial training Viva–Voce Examination of_____ has been held on _____ and accepted.

Signature of Internal Examiner

Signature of External Examiner

## ABSTRACT

This report details the training experience I undertook at **VproTech Digital** over a four-week period, focusing on web technology. The primary goal of this training was to deepen my understanding of web development and to acquire practical skills in creating responsive and user-friendly web applications.

During the training, I worked on several projects that involved both front-end and back-end development. I gained hands-on experience with key technologies such as HTML, CSS, and JavaScript, as well as frameworks like React .This experience allowed me to design intuitive user interfaces and implement features that enhance user interaction.

In addition to front-end development, I explored server-side programming with languages such as Node.js. I also gained knowledge in database management using systems like MySQL, MongoDB, which is essential for data storage and retrieval in web applications.

A significant component of my training involved understanding the software development lifecycle. I learned about Agile methodologies, which promote collaborative and iterative development processes. This approach not only improved my project management skills but also highlighted the importance of adaptability and continuous feedback in delivering successful software projects.

Moreover, I delved into API integration, which is crucial for enhancing web application functionality. By understanding how to connect with third-party services and utilize their data, I was able to create more dynamic and robust applications. Additionally, I learned about deployment strategies, ensuring that the applications developed could be effectively launched and maintained.

In summary, this training has significantly enriched my technical skills and provided a comprehensive understanding of web technology. The insights gained will be invaluable as I continue my studies in Information Technology and pursue a career in web development and digital solutions.

**ACKNOWLEDGEMENT**

I would like to express my heartfelt gratitude to everyone who contributed to the successful completion of my training at **VproTech Digital** This experience has greatly enriched my understanding of web technology and provided invaluable practical skills.

First and foremost, I would like to thank my supervisor **Suraj Thakur** for their guidance, support, and encouragement throughout my training. Their insights and expertise were instrumental in shaping my learning experience.

I extend my appreciation to the entire team at **VproTech Digital** for their cooperation and willingness to share their knowledge. The collaborative environment fostered my growth and allowed me to gain hands-on experience in various aspects of web development.

I also wish to acknowledge my professors and mentors at **Guru Nanak Dev Engineering College, Ludhiana**, whose teachings laid the foundation for my knowledge in Information Technology. Their constant support and motivation have inspired me to pursue my goals with passion and dedication.

I am highly grateful to **Dr. Sehajpal Singh**, Director, GNDEC, Ludhiana, for providing us opportunity to carry out one month training .We express gratitude to **DR.KULVINDER SINGH MANN (HOD, IT)** for their intellectual support.

Lastly, I am grateful to my family and friends for their unwavering support and encouragement during this training period. Their belief in my abilities has motivated me to strive for excellence.

Thank you all for making this training experience memorable and impactful.

## ABOUT THE COMPANY

Founded in March 2017 by Rajat Kumar, **Vprotech Digital** has established itself as a prominent technology solutions provider in Mohali. Driven by a passion for transforming students into technically proficient professionals, Rajat Kumar launched the company to offer comprehensive industrial training for B.Tech, Diploma, BCA, and MCA students in the Chandigarh region.

Viprotech Digital specializes in providing six-month and six-week industrial training programs that cover a wide array of subjects, including web development, web design, Android application development, SEO, social media marketing (SMM), and digital marketing. Our team of experienced trainers is dedicated to equipping students with the skills and knowledge needed to thrive in today's competitive job market.

In addition to training, we excel in creating effective virtual branding strategies and developing mobile applications for both Android and iPhone devices. Our web development services ensure that all websites are responsive and compatible across various devices, helping businesses establish a strong online presence.

We also offer logo design services to create unique brand identities and provide expertise in interior design. At Vprotech Digital, we not only focus on skill development but also assist our trainees in securing suitable job placements upon completion of their training programs.

With a commitment to excellence and innovation, Vprotech Digital continues to empower students and businesses, helping them navigate the ever-evolving digital landscape.

## I.INTRODUCTION

### 1.1 HTML (Hypertext Markup Language)

HTML is the standard language used to create the structure of web pages. It defines the content of web pages using elements enclosed in angle brackets, also known as tags. HTML provides a way to structure text, images, links, tables, and more. The most common tags are:

<h1> to <h6> for headings,

<p> for paragraphs,

<img> for images,

<a> for links.

HTML follows a hierarchical structure, meaning elements can be nested inside one another. HTML5 introduced new semantic tags such as <header>, <footer>, <article>, and <section>, which help define different parts of a webpage more meaningfully.

**Key Features:**

Tags can have attributes to provide more information, like id, class, src (for images), and href (for links).

HTML forms allow user interaction through elements like <input>, <button>, <select>, etc.

HTML is platform-independent and supported by all browsers.

Example:

```
<!DOCTYPE html>
<html>
<head>
   <title>My First Web Page</title>
</head>
<body>
   <h1>Welcome to HTML</h1>
   <p>This is a paragraph about HTML basics.</p>
</body>
</html>
```

**2.2 CSS (Cascading Style Sheets)**

CSS is a stylesheet language that describes the presentation of HTML content. CSS allows developers to control the layout, color, fonts, and spacing of elements, making web pages more visually appealing. CSS separates the content (HTML) from the presentation, ensuring easier maintenance and scalability of websites.

CSS works by selecting HTML elements and applying styles to them. Selectors target specific elements (e.g., body, p, #id, .class), while properties like color, background-color, font-size, and margin define the styles.

CSS Types:

Inline CSS: Applied directly to an HTML element using the style attribute.

Internal CSS: Placed within the <style> tag in the <head> section of an HTML document.

External CSS: Linked through an external file with a .css extension.

Key Features:

CSS allows responsiveness using media queries to adjust layouts for different screen sizes.

CSS Grid and Flexbox offer advanced layout capabilities.

Pseudo-classes (:hover, :active, etc.) allow for dynamic styling.

Example:

```
body {
   background-color: lightgray;
   font-family: Arial, sans-serif;
}
h1 {
   color: blue;
   text-align: center;
}
```

### 3.3 JavaScript

JavaScript is a dynamic programming language that enables interaction and interactivity on web pages. It is a client-side language (though it can also be used server-side with Node.js) that runs in the browser. JavaScript manipulates the Document Object Model (DOM), allowing you to change content, structure, and style dynamically.

JavaScript supports event handling, meaning that actions (like clicks or keystrokes) can trigger functions. It also supports asynchronous programming with promises, async/await, and callbacks, which are important for tasks like fetching data from servers without reloading the page (AJAX).

**Key Features:**

1. DOM Manipulation: Allows dynamic updating of HTML elements.
2. Event Handling: Responds to user actions like clicks, hovers, and form submissions.
3. Built-in Functions and Objects: Includes functions like alert(), console.log(), and objects like Date(), Math.
4. Asynchronous Programming: Supports handling of real-time data with Promises and async/await.
5. Compatibility with other frameworks and libraries like React, Angular, and jQuery.

**Example:**

```
document.getElementById("demo").innerHTML = "Hello, JavaScript!";
```

## 4.4 React

React is a JavaScript library for building user interfaces, developed by Facebook. It allows developers to create reusable UI components that update dynamically based on data changes. React is designed for building single-page applications (SPAs) where only certain parts of the page refresh when data changes, improving performance and user experience.

React uses a virtual DOM (Document Object Model) to efficiently update the actual DOM. Instead of updating the entire page, React only updates the parts that have changed, making it faster than traditional DOM manipulation.

**Key Features:**

- Component-based architecture: Build encapsulated components that manage their own state and reuse them.
- JSX: A syntax that allows you to write HTML-like elements directly in JavaScript.
- State and Props: Manage dynamic data in components and pass data between them.
- Hooks: New React features (introduced in React 16.8) that let you use state and other React features in functional components.

**Example**:

```
function App() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div> );}
```

## 2. Introduction to HTML

HTML (Hypertext Markup Language) is the foundation of web development. It defines the structure of web pages using elements, which browsers interpret and display. HTML consists of tags enclosed in angle brackets (< >), and most elements have opening (<tag>) and closing (</tag>) tags. HTML is not a programming language but a markup language that describes the structure of documents.

### History of HTML

HTML was created by Tim Berners-Lee in 1991 and has gone through various versions, with HTML5 being the latest major version introduced in 2014. HTML5 introduced new elements, attributes, and behaviors, making it more robust and capable for modern web applications.

### Basic Structure of an HTML Document

The basic structure of an HTML document consists of the following parts:

1. **DOCTYPE Declaration**: Declares the document type. In HTML5, it is simply <!DOCTYPE html>.
2. **HTML Element**: The root element that contains all the content of the page.
3. **Head Section**: Contains meta-information about the document, such as title, character encoding, links to CSS files, etc.
4. **Body Section**: Contains the actual content of the web page, including text, images, links, and other elements.

Example of a basic HTML structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My First Web Page</title>
</head>
<body>
    <h1>Welcome to HTML</h1>
```

This is a paragraph of text.

</body></html>

### 2.2 HTML Elements and Tags

HTML elements are the building blocks of web pages. Every element has specific roles and defines different parts of the document. There are two types of elements:

- **Block-level elements**: These elements take up the full width of their container, such as , <h1>, , and <section>.
- **Inline elements**: These elements take up only as much width as necessary, such as , <a>, and <img>.

**Common HTML Elements**

1. **Headings**: HTML provides six levels of headings, from <h1> to <h6>, with <h1> being the most important and <h6> the least.

   <h1>Main Heading</h1> <h2>Subheading</h2>

2. **Paragraphs**: The  tag is used for defining blocks of text as paragraphs.

   This is a paragraph in HTML.

3. **Links**: Links are created using the <a> (anchor) tag, with the href attribute specifying the destination URL.

   <a href="https://www.example.com">Click here</a>

4. **Images**: The <img> tag is used to embed images, with the src attribute specifying the image URL and alt providing alternative text for accessibility.

   <img src="image.jpg" alt="Description of image">

### 3. HTML Attributes

HTML elements can have attributes that provide additional information. Attributes are always defined in the opening tag and consist of a name and value pair. Common attributes include:

- id: Provides a unique identifier for an element.
- class: Groups elements for styling or scripting.

- src: Specifies the source of embedded content, such as images, scripts, or media.
- href: Specifies the URL for links.

Example:

<a href="https://example.com" target="_blank">Visit Example</a>

**2.3 Lists in HTML**

HTML supports ordered and unordered lists to organize content.

- **Ordered Lists (**<ol>**)**: Use numbers or letters to display list items.

  <ol> <li>Item 1</li> <li>Item 2</li> </ol>

- **Unordered Lists (**<ul>**)**: Use bullets to display list items.

  <ul> <li>Item A</li> <li>Item B</li> </ul>

  Additionally, **definition lists** (<dl>, <dt>, <dd>) are used for terms and descriptions.

**5. Forms in HTML**

Forms are crucial for gathering user input in web applications. The <form> element contains various input controls like text fields, radio buttons, checkboxes, dropdowns, and submit buttons. The action attribute specifies where the form data should be sent, and method defines the HTTP method (GET or POST).

**Common Form Elements**:

- **Text Input** (<input type="text">)
- **Password Input** (<input type="password">)
- **Radio Buttons** (<input type="radio">)
- **Checkboxes** (<input type="checkbox">)
- **Submit Button** (<input type="submit">)

Example:

<form action="/submit" method="post"> <label for="username">Username:</label> <input type="text" id="username" name="username"> <label for="password">Password:</label> <input type="password" id="password" name="password"> <input type="submit" value="Submit"> </form>

### 2.4 Semantic HTML

Semantic HTML uses meaningful tags to define the structure of web pages, improving accessibility and SEO. It helps browsers and search engines better understand the content and its structure. Examples of semantic elements include:

- <header>: Defines the header section of a page.
- <nav>: Defines a navigation menu.
- <main>: Defines the main content of the page.
- <article>: Defines a self-contained piece of content.
- <footer>: Defines the footer section of a page.

Example:

<header> <h1>Website Title</h1> <nav> <ul> <li><a href="#home">Home</a></li> <li><a href="#about">About</a></li> </ul> </nav> </header> <main> <article> <h2>Article Title</h2> This is an article. </article> </main> <footer> © 2024 My Website </footer>

### 7. Tables in HTML

Tables are used to display tabular data in rows and columns. The basic structure of a table includes:

- <table>: The container element for the table.
- <tr>: Defines a row.
- <th>: Defines a header cell.
- <td>: Defines a data cell.

Example:

<table> <tr> <th>Name</th> <th>Age</th> </tr> <tr> <td>John Doe</td> <td>30</td> </tr> </table>

Tables should be used sparingly for layout purposes and primarily for presenting structured data.

### 2.5. HTML5 Features

HTML5 introduced several new elements, APIs, and attributes that improve web development, including:

- **Audio and Video Elements**: <audio> and <video> allow multimedia embedding without external plugins.

  html

  Copy code

  <video controls> <source src="movie.mp4" type="video/mp4"> </video>

- **Canvas Element**: <canvas> allows for rendering 2D shapes, images, and animations via JavaScript.

  <canvas id="myCanvas"></canvas> <script> const canvas = document.getElementById('myCanvas'); const ctx = canvas.getContext('2d'); ctx.fillStyle = "blue"; ctx.fillRect(10, 10, 150, 100); </script>

- **Geolocation API**: Provides location-based services by accessing a user's geographical position.

  <button onclick="getLocation()">Get Location</button>  <script> function getLocation() { if (navigator.geolocation) { navigator.geolocation.getCurrentPosition(function(position) { document.getElementById("location").innerHTML = "Latitude: " + position.coords.latitude + "<br>Longitude: " + position.coords.longitude; }); } } </script>

**9. Accessibility in HTML**

Accessibility refers to making web content usable for all people, including those with disabilities. Developers should follow best practices

### 3. INTRODUCTION TO CSS

CSS (Cascading Style Sheets) is a style sheet language used to control the appearance of HTML documents. While HTML defines the structure of a webpage, CSS determines how it looks. With CSS, you can apply styles such as colors, fonts, and layout to a website. CSS enables separation of concerns by keeping design separate from the content, making websites more flexible and easier to maintain.

CSS was introduced in 1996 by the W3C (World Wide Web Consortium), and its evolution has seen different levels:

- **CSS1 (1996**): The first standard**.**
- **CSS2 (1998**): Introduced more features like positioning and z-index.
- **CSS3 (2011):** Introduced modules for better web styling capabilities, like transitions, animations, and responsive design techniques.

**Why CSS is Important**

1. **Separation of Content and Design**: With CSS, you can change the look of an entire website by modifying a single CSS file. It ensures that the visual presentation is separate from the structure.
2. **Reusability**: CSS rules can be applied to multiple HTML documents, ensuring consistency across all web pages.
3. **Responsive Design**: CSS plays a critical role in making websites responsive, i.e., ensuring they work well on different screen sizes and devices (like mobile phones and tablets).

### 2. Basic Structure of CSS

CSS follows a simple rule-based syntax. Each CSS rule consists of a selector and a declaration block:

**Selector: Specifies which HTML element(s) the style applies to.**

- **Declaration Block:** Contains one or more declarations, each made up of a property and a value.

**Example:**

```
h1 { color: blue; font-size: 24px; }
```

**In this example:**

- h1 is the selector.
- color and font-size are properties, and blue and 24px are the corresponding values.

CSS rules can be written in three ways:

1. **Inline CSS:** Directly in HTML elements using the style attribute**.**

   ** This is red text. **

2. **Internal CSS**: Within a <style> tag in the <head> section of an HTML document.

   <style> p { color: green; } </style>

3. **External CSS**: Linked through an external stylesheet using the <link> tag.

   <link rel="stylesheet" href="styles.css">

### 3.2 CSS SELECTOR

Selectors define the HTML elements to which styles should be applied. CSS provides a wide range of selectors:

- **Element Selectors**: Select elements by tag name, such as p, div, or h1.
- **Class Selectors**: Select elements by class name. Classes are defined with a dot (.) in CSS.

  .myClass { color: red; }

- **ID Selectors**: Select elements by ID, defined with a hash (#). IDs are unique to the page.

  #myId { color: blue; }

- **Group Selectors:** Apply styles to multiple selectors at once by separating them with commas.

  h1, h2, p { color: purple; }

- **Descendant Selectors**: Select elements that are descendants of another element.

  div p { font-size: 16px; }

## 4. Colors in CSS

CSS allows you to set colors using various formats:

- **Named Colors**: Simple color names, such as red, blue, green.

  p { color: red; }

- **Hexadecimal Colors:** Colors represented by hexadecimal values, such as #FF5733.

  h1 { color: #FF5733; }

- **RGB and RGBA Colors:** Defines colors using the Red, Green, Blue model, optionally with an alpha (transparency) value.

body { background-color: rgb(255, 0, 0); /* pure red */ } div { background-color: rgba(0, 0, 255, 0.5); /* semi-transparent blue */ }

- **HSL Colors**: HSL stands for Hue, Saturation, and Lightness.

  h2 { color: hsl(120, 100%, 50%); /* a bright green */ }

### 3.3. CSS Box Model

The box model is one of the most important concepts in CSS. Every HTML element is treated as a rectangular box, and the box model defines how these boxes are sized and spaced.

**The box model includes:**

1. **Content**: The actual content of the element (like text or images).
2. **Padding:** Space between the content and the border.
3. **Border:** The border around the padding and content.
4. **Margin**: Space outside the border, separating the element from others.

**Example:**

div { width: 200px; padding: 10px; border: 5px solid black; margin: 20px; }

In this example, the total width of the element is not just 200px but includes the padding, border, and margin.

### 6. CSS Typography

Typography involves controlling the appearance of text on a webpage. CSS provides properties to style fonts, such as:

- **Font Family:** Defines the typeface for text**.**

  body { font-family: Arial, sans-serif; }

- **Font Size:** Sets the size of the font, which can be in various units like px, em, rem, or %.

  p { font-size: 16px; }

- **Font Weight**: Controls the thickness of the text, such as bold, normal, or numerical values.

  h1 { font-weight: bold; }

- **Text Align**: Aligns text horizontally.

  h2 { text-align: center; }

- **Text Decoration**: Adds decorations like underlines, line-through, etc.

  a { text-decoration: none; }

### 3.4. CSS Layout Techniques

CSS offers several ways to create layouts, allowing developers to structure web pages:

- **Flexbox:** Flexbox simplifies the process of creating complex layouts by aligning elements along a row or column.

  .container { display: flex; justify-content: space-between; }

- **Grid Layout**: CSS Grid is a two-dimensional layout system for creating complex grid-based designs.

  .grid-container { display: grid; grid-template-columns: 1fr 1fr 1fr; grid-gap: 20px; }

- **Float: Older method for layout, where elements float left or right.**

  .sidebar { float: left; width: 25%; }

- **Positioning**: As mentioned earlier, CSS positioning is also used to create layouts by placing elements at specific points on the page.

### 3.5. Responsive Web Design

With the rise of mobile and tablet devices, it's essential to ensure websites look good on all screen sizes. CSS offers tools to make websites responsive:

- **Media Queries**: Allow different styles to be applied based on the screen size or other conditions.

  @media (max-width: 600px) { body { font-size: 14px; } }

- **Flexbox and Grid Layouts:** These tools are naturally responsive and adjust to different screen sizes.

- **Viewport Meta Tag**: Helps control how the site appears on mobile devices.

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

### 11. CSS Animations and Transitions

**CSS allows for creating smooth transitions and animations without using JavaScript:**

- **Transitions:** Allow property changes to occur smoothly over time.

  button { background-color: blue; transition: background-color 0.5s ease; } button:hover { background-color: red; }

- **Keyframe Animations:** Define animations that change the properties of an element over time.

  @keyframes slideIn { from { transform: translateX(-100%); } to { transform: translateX(0); } } .element { animation: slideIn 1s ease-in-out; }

  .

## 4.Introduction to JavaScript

### What is JavaScript?

- A versatile programming language primarily used for web development.

- Enables client-side scripting, allowing for dynamic content updates without reloading the page.

### History

- Created in 1995 by Brendan Eich for Netscape.

- Standardized as ECMAScript (first edition published in 1997).

- Continues to evolve with annual updates, the latest being ES2023.

### Use Cases

- Client-side web development (browser scripting).

- Server-side development (Node.js).

- Mobile app development (React Native).

- Game development (Phaser, Three.js).

### Basic Syntax

### Variables

- Declaration Keywords

  - `var`: Function-scoped, can be re-declared.

  - `let`: Block-scoped, cannot be re-declared in the same scope.

  - `const`: Block-scoped, must be initialized at declaration, cannot be reassigned.

### 4.2Data Types

### Primitive Types

- `String`: Represents text.

- `Number`: Represents integers and floating-point numbers.

- `Boolean`: Represents `true` or `false`.

- `Null`: Represents intentional absence of any value.
- `Undefined`: Represents a variable that has been declared but not assigned.
- `Symbol`: Represents a unique identifier.
- `BigInt`: Represents integers with arbitrary precision.

**Example**

```
let age = 25; // Number
const name = "Alice"; // String
let isStudent = true; // Boolean
let score = null; // Null
let subject; // Undefined
```

**Operators**

**String Concatenation**

```
let fullName = firstName + " " + lastName;
```

**Ternary Operator**

```
let canVote = age >= 18 ? "Yes" : "No";
```

**3. Control Structures**

**Conditional Statements**

**Switch Statement**

```
switch (day) {
    case 1:
        console.log("Monday");
        break;
    case 2:
```

```
 console.log("Tuesday");


        break;
    default:
        console.log("Another day");
 }
```

**Loops**

**ForEach Loop for Arrays**

```
fruits.forEach(fruit => console.log(fruit));
```

**For...of Loop**

```
for (const fruit of fruits) {
    console.log(fruit);
 }
```

**4. Functions**

**Function Types**

**Anonymous Functions**

```
 const multiply = function(x, y) {
    return x * y;
 };
```

**Immediately Invoked Function Expression (IIFE  ```javascript**

```
 (function() {
    console.log("Executed!");
 })();
```

**Higher-Order Functions**

- Functions that take other functions as arguments or return them.

- Example: Array methods like `map`, `filter`, and `reduce`.

const numbers = [1, 2, 3, 4];

const doubled = numbers.map(num => num * 2);

**5. Objects and Arrays**

**Creating Objects**

```
const car = {
    make: "Toyota",
    model: "Camry",
    year: 2020,
    drive: function() {
        console.log("Driving");
    }
};
```

**Arrays**

**Array** Methods - `push()`, `pop()`, `shift()`, `unshift()`, `slice()`, `splice()`

fruits.push("orange"); // Adds at the end

const removedFruit = fruits.pop(); // Removes last

**Multi-dimensional Arrays**

Arrays of arrays.

const matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

];

**4.3. ES6 Features**

**Template Literals**

- Allows multi-line strings and string interpolation.

const message = `Hello, ${name}. You are ${age} years old.`;

**Destructuring Assignment**

- Simplifies extracting values from arrays and objects.

**javascript**

const user = { id: 1, username: "Jane" };

const { id, username } = user

**Spread and Rest Operators**

**Spread: Unpacking elements of an array.**

const newArray = [...fruits, "kiwi"]

**- Rest:** Packing parameters into an array.

const sumAll = (...numbers) => numbers.reduce((total, num) => total + num, 0);

**7. Asynchronous JavaScript**

**Callbacks**

- Functions passed as arguments to other functions, executed after a task completes.

```javascript
function fetchData(callback) {
    setTimeout(() => {
        callback("Data received");
    }, 1000);
}
```

**Async/Await**

- Syntactic sugar over promises, making asynchronous code easier to read.

```
const getData = async () => {
    try {
        const data = await fetchData();
        console.log(data);
    } catch (error) {
        console.error(error);
    }
};
```

**Modifying Elements**

- Changing content, styles, and attributes.

```
header.textContent = "New Header";
header.style.color = "blue";
```

**4.4 Event Handling**

- Listening for events such as clicks, form submissions.

```javascript
button.addEventListener("click", () => {
    alert("Button clicked!");
});
```

**9. Error Handling**

Try...Catch

- Handling errors gracefully.

```javascript
try {
```

```
// Code that may throw an error
    throw new Error("An error occurred");
} catch (error) {
    console.error(error.message);
} finally {
    console.log("Execution completed.");
}
```

### 5.1 Introduction to React

React is an open-source JavaScript library for building user interfaces, particularly well-suited for single-page applications (SPAs). Developed and maintained by Facebook, React promotes a declarative style of programming, enabling developers to create interactive UIs that efficiently update in response to data changes.

**Core Concepts of React**

### 5.2 Components

**Components are the backbone of React applications. They encapsulate rendering logic, styles, and behavior, making it easier to build and maintain large UIs.**

**Types of Components**

- **Functional Components:** These are simple JavaScript functions that return JSX. They are stateless unless hooks are used.

  Example: Functional Component

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

- **Class Components:** These are ES6 classes that extend React.Component. They can manage their own state and lifecycle methods.

  **Example**: Class Component

```
class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
```

### 1.2 JSX

JSX is a syntax extension for JavaScript, allowing you to write HTML-like code in your JavaScript files. It improves readability and expressiveness.

**Key Features of JSX:**

- **HTML-like Syntax:** JSX looks similar to HTML but is actually syntactic sugar for React.createElement() calls.

  Example of JSX

  const element = <h1 className="greeting">Hello, world!</h1>;

- **Expressions:** You can embed JavaScript expressions within curly braces.

  Example

  const user = { name: 'Alice' };

  const element = <h1>Hello, {user.name}!</h1>;

### 1.3 Props

Props are short for properties and are used to pass data from parent to child components. They are read-only, meaning a child component cannot modify them.

Example of Passing Props

function App() {

  return <Greeting name="Alice" />;

}

In this example, App passes the name prop to the Greeting component.

### 1.4 State

State is a component's internal data store, which can change over time, typically in response to user actions or network responses.

**Managing State**

- **Class Components**: State is initialized in the constructor and updated using this.setState().

  Example

  jsx

  Copy code

  class Counter extends React.Component {

```
constructor(props) {
  super(props);
  this.state = { count: 0 };
 }

 increment = () => {
  this.setState({ count: this.state.count + 1 });
 };

 render() {
  return (
   <div>
    <p>{this.state.count}</p>
    <button onClick={this.increment}>Increment</button>
   </div>
  );
 }
}
```

- **Functional Components**: Use the useState hook to manage state.

```
Example
import React, { useState } from 'react';
function Counter() {
  const [count, setCount] = useState(0);
return (
  <div>
    <p>{count}</p>
    <button onClick={() => setCount(count + 1)}>Increment</button>
  </div>
```

```
  );
}
```

**5.3 Lifecycle Methods**

Lifecycle methods are hooks into the component's lifecycle in class components. Common lifecycle methods include:

- **componentDidMount():** Called after the component is mounted.
- **componentDidUpdate():** Called after the component updates.
- **componentWillUnmount():** Called before the component is removed from the DOM.

Example of Lifecycle Methods

jsx

Copy code

```
class MyComponent extends React.Component {
  componentDidMount() {
    console.log('Component mounted');
  }
  componentDidUpdate(prevProps, prevState) {
    if (this.state.value !== prevState.value) {
      console.log('Value changed');
    }
  }
  componentWillUnmount() {
    console.log('Component unmounted');
  }
  render() {
    return <div>My Component</div>;
  }
}
```

**Functional Components and Hooks**

With hooks, functional components can manage lifecycle events. The useEffect hook

serves a similar purpose to lifecycle methods.

Example with useEffect

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


function MyComponent() {
  useEffect(() => {
    console.log('Component mounted or updated');
    return () => {
      console.log('Component unmounted');
    };
  }, []); // Empty dependency array runs once on mount


  return <div>My Functional Component</div>;
}
```

**2. Advanced Concepts**

**2.1 Hooks**

Hooks provide a way to use state and other React features in functional components. They were introduced in React 16.8.

Commonly Used Hooks

- **useState**: For adding state to functional components.

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

- **useEffect**: For managing side effects, similar to lifecycle methods.

  ```
  useEffect(() => {
    // Side effect logic here
    return () => {
      // Cleanup logic here
    };
  }, [dependencies]); // Runs when dependencies change
  ```

- **useContext**: For accessing context in functional components.

```
const value = useContext(MyContext);
```

2.2 Context API

The Context API allows you to share data between components without passing props explicitly through every level of the component tree.

**Example of Context API**

1. **Creating Context**

```
const ThemeContext = React.createContext('light');
```

2. **Providing Context**

```
function App() {
  return (
    <ThemeContext.Provider value="dark">
      <Toolbar />
    </ThemeContext.Provider>
  );
}
```

3. **Consuming Context**

```
function Toolbar() {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}


function ThemedButton() {
  return (
    <ThemeContext.Consumer>
      {theme => <button className={theme}>Theme Button</button>}
    </ThemeContext.Consumer>
```

```
  );
}
```

**5.4 Redux**

Redux is a predictable state container for JavaScript apps. It's particularly useful for managing global state in larger applications.

Key Concepts in Redux

- **Store: Holds the entire state of the application.**

```
import { createStore } from 'redux';
const store = createStore(reducer);
```

- **Actions**: Objects that represent a change in the application state.

```
const ADD_TODO = 'ADD_TODO';
function addTodo(text) {
  return { type: ADD_TODO, text };
}
```

- **Reducers**: Functions that specify how the state changes in response to actions.

```
function todos(state = [], action) {
  switch (action.type) {
    case ADD_TODO:
      return [...state, action.text];
    default:
      return state;
  }
}
```

**Connecting Redux with React**

Use the react-redux library to connect Redux with your React components.

```
import { Provider, connect } from 'react-redux';

function App() {
  return (
```

```
 <Provider store={store}>
     <ConnectedTodoList />
   </Provider>
  );
}

const mapStateToProps = state => ({
  todos: state.todos,
});

const ConnectedTodoList = connect(mapStateToProps)(TodoList);
```

**3. Routing in React**

React Router is a powerful library that enables dynamic routing in React applications.

Basic Routing

1. **Install React Router**

```
npm install react-router-dom
```

2. **Set Up Routing**

```
import { BrowserRouter as Router, Route, Link } from 'react-router-dom';

function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>
      <Route path="/" exact component={Home} />
      <Route path="/about" component={About} />
```

```
</Router>
  );
}
```

**Nested Routes**

React Router also allows for nested routes, making it easy to structure your application's navigation.

**Example of Nested Routes**

```
function User() {
  return (
    <div>
      <h2>User</h2>
      <Route path="/user/:id" component={UserDetail} />
    </div>
  );
}
```

**Route Parameters**

Route parameters allow you to pass dynamic values in the URL.

**Example**

```
function UserDetail({ match }) {
  return <h3>User ID: {match.params.id}</h3>;
}
```

**4. Best Practices**

**4.1 Component Structure**

Organizing components logically helps in maintaining a clear codebase. A recommended structure is:

```
/src
  /components       # Reusable components
  /containers       # Components that manage state
  /pages            # Page-level components
```

```
/hooks          # Custom hooks
 /redux         # Redux-related files
```

### 4.2 State Management

- Use local state for localized needs and consider lifting state up when multiple components depend on the same state.
- For larger applications, consider using Redux or the Context API for better state management.

### 4.3 Performance Optimization

1. **React.memo**: Prevents unnecessary re-renders for functional components.

```
const MyComponent = React.memo(function MyComponent(props) {
  // only re-renders if props change
});
```

2. **Code Splitting**: Use React's built-in React.lazy() and Suspense for lazy loading components.

```
const LazyComponent = React.lazy(() => import('./LazyComponent'));
<Suspense fallback={<div>Loading...</div>}>
  <LazyComponent />
</Suspense>
```

3. **Use useMemo and useCallback:** Optimize performance by memoizing values and functions.

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
const memoizedCallback = useCallback(() => { /* function logic */ }, [dependencies]);
```

### 4.4 Testing

Testing is crucial for ensuring code reliability. Use tools like Jest and React Testing Library to write unit tests and integration tests.

Example Test with React Testing Library

```
import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
```

```
  render(<App />);

  const linkElement = screen.getByText(/learn react/i);

  expect(linkElement).toBeInTheDocument();

});
```

**5. Conclusion**

React has fundamentally transformed how developers build user interfaces, offering a component-based architecture that promotes reusability and maintainability. By mastering its core principles, advanced features, and best practices, developers can create robust, efficient, and scalable applications. As the React ecosystem continues to evolve, staying updated with the latest tools and techniques will further enhance your development capabilities.

**Reference**

Chatgpt: https://chatgpt.com/?oai-dm=1

GFG : https://www.geeksforgeeks.org/

W3School: https://www.w3schools.com/

# Table of Contents