Software Engineering

Target group: 4th year Computer engineering students

ECE, GIT,DTU

By Misganaw Aguate

MSc. In Computer Engineering

ethiomisgie@gmail.com

You Tube: ethioptech

https://project.ethioptec.com

https://academics.ethioptec.com

Web development

Contents

- Introduction to web development
- Front end development
- Back end development
- Example of front end and back end using bootstrap, PHP and MYSQL
- JavaScript
- Exercises

Web development

Web development is the process of

- creating
- building, and
- maintaining websites or web applications.

It encompasses a range of tasks from developing a simple **static page** of plain text to complex **web-based applications**, e-commerce platforms, and social networking services.

The key components of web development are:

- Front end development
- Back end development and
- Full stack development

Web development

Types of Websites

Static Websites:

- Content is fixed and displayed the same way to every user.
- Created using HTML and CSS only.
- Examples: Personal portfolios, simple blogs.

Dynamic Websites:

- Content changes based on user interaction or data.
- Require both frontend and backend development.
- Examples: Social media platforms, e-commerce sites.

Web Applications:

- Feature-rich, interactive websites that function like software applications.
- Examples: Gmail, Google Docs, Slack.

Web development

Web development process

Planning:

•Define the purpose, goals, and target audience of the website.

•Create wireframes and mockups.

Design:

•Build the layout, visual style, and interactive elements.

•Ensure responsive and user-friendly design.

Development:

•Frontend: Build the UI using HTML, CSS, and JavaScript.

•Backend: Create server-side logic, database management, and APIs

Testing:

•Ensure the website works on various devices and browsers.

•Fix bugs and optimize performance.

Deployment:

•Launch the website on a hosting platform (e.g., AWS, Google Cloud, Netlify).

Maintenance:

•Update content, add new features, and fix any issues that arise

Web development

Career Paths in Web Development

Frontend Developer

Specializes in creating the visual and interactive parts of a website.

Backend Developer

Focuses on server-side logic, databases, and API development.

Full-Stack Developer

Handles both frontend and backend development.

Web Designer

Designs the layout and aesthetics of websites.

Web Developer

A general term for anyone involved in building and maintaining websites.

Web development

Frontend (Client-Side) development

- The frontend is the part of the application that the user **interacts** with directly.
- It focuses on the **visual** and **interactive elements** of a web or mobile application.

Tasks in back end development

- Designing and building user interfaces (UI).
- Ensuring a responsive and interactive user experience (UX).
- Communicating with the backend via APIs.
- Handling client-side validations

Web development

Key Features of Frontend Development

User Interface (UI):

- Building the layout and design of the website or app.
- Ensuring that the design is responsive, meaning it adjusts to various screen sizes like desktop, tablet, or mobile.

User Experience (UX):

- Adding interactivity and animations to make the application user-friendly.
- Implementing intuitive navigation and functionality.

Communication with Backend:

- Sending and receiving data using APIs (e.g., through REST or GraphQL).
- Handling asynchronous operations like loading data without refreshing the page

Web development

Frontend (Client-Side)

Programming Languages

- **HTML:** For structuring content.
- **CSS:** For styling content.
- **JavaScript:** For interactivity and dynamic content

Frameworks and Libraries:

JavaScript Frameworks:

- Angular
- React
- Vue.js
- Svelte

CSS Frameworks:

- Bootstrap
- Tailwind CSS
- Bulma

Frameworks and Libraries:

Others:

- jQuery (Legacy)
- Chart.js (Data visualization)
- Three.js (3D rendering)

Web development

Backend (Server-Side) development

The backend is the part of the application that operates **behind** the **scenes**.

It is responsible for

- **managing** and **processing** data.
- handling requests and
- ensuring proper communication between the database and the frontend.

Tasks in back end development

- Database management (storing and retrieving data).
- User authentication and authorization.
- Business logic (e.g., calculations, data processing).
- API creation and management (REST or GraphQL APIs).
- Server and application performance optimization.
- Security measures like encryption and secure data handling.

Web development

Key Features of Backend Development:

Server-Side Logic

Handles business logic, such as user authentication, authorization, and processing data.

Database Management

- Manages data storage, retrieval, updates, and security.
- Ensures data integrity and proper schema design.

APIs (Application Programming Interfaces)

- Provides endpoints for the frontend to interact with the server.
- Enables data exchange between the frontend and backend.

Security

- Implements measures to protect user data (e.g., encryption, secure tokens).
- Prevents unauthorized access and vulnerabilities

Web development

Back end

Those are the **databases**, **programming** languages and **frameworks** that used to develop back end components of the web application.

Databases

- **Relational Databases:** MySQL, PostgreSQL, Microsoft SQL Server, Oracle.
- **NoSQL Databases:** MongoDB, DynamoDB, CouchDB.

Programming Languages

- JavaScript
- Python
- Java
- Ruby
- PHP
- C#
- Go
- Kotlin
- Perl
- Rust

Frameworks

- Node.js (Express, NestJS)
- Django (Python)
- Laravel or codeigniter (PHP)
- Flask (Python)
- Spring Boot (Java)
- Ruby on Rails (Ruby)
- Laravel (PHP)
- ASP.NET Core (C#)
- Gin (Go)
- FastAPI (Python)

Web development

Example of front end end design

Step 1: Download Bootstrap Files

1. Go to the Bootstrap website. Or https://getbootstrap.com/docs/4.0/getting-started/download/
2. Download the **compiled CSS and JS** files (not the source files).
3. Extract the downloaded .zip file. You will see the following folders:
   - css/ (contains Bootstrap CSS files)
   - js/ (contains Bootstrap JavaScript files)

Step 2: Place Bootstrap Files in XAMPP/htdocs

1. Navigate to your XAMPP/htdocs directory (e.g., C:\xampp\htdocs).
2. Create a folder named bootstrap_assets in the htdocs directory.
3. Copy the css/ and js/ folders from the extracted Bootstrap files into the bootstrap_assets folder

Step 3: Link Bootstrap CSS and JavaScript Locally

When you create an HTML file, you can now reference the Bootstrap files located in the XAMPP/htdocs/bootstrap_assets folder.

1.Create a New Folder for Your Project:

- In htdocs, create a new folder, e.g., user_registration.

2.Add a New HTML File:

- Inside the user_registration folder, create a file named index.html.

3.Link Bootstrap Files Locally: Use the **href** and src attributes to link the local Bootstrap files in your **index.html** file.

Web development

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User Registration</title>
    <!-- Link Bootstrap CSS -->
    <link rel="stylesheet" href="../bootstrap_assets/css/bootstrap.min.css">
</head>
<body>
    <div class="container mt-5">
        <h1 class="text-center">User Registration</h1>
        <form action="register.php" method="POST" class="mt-4">
            <div class="mb-3">
                <label for="username" class="form-label">Username</label>
                <input type="text" name="username" id="username" class="form-control" required>
            </div>
            <div class="mb-3">
                <label for="email" class="form-label">Email</label>
                <input type="email" name="email" id="email" class="form-control" required>
            </div>
            <div class="mb-3">
                <label for="password" class="form-label">Password</label>
                <input type="password" name="password" id="password" class="form-control" required>
            </div>
            <button type="submit" class="btn btn-primary">Register</button>
        </form>
    </div>

    <!-- Link Bootstrap JavaScript -->
    <script src="../bootstrap_assets/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Web development

Example of back end design for the designed front end

Step 1: Install XAMPP

1. Download the XAMPP installer from ApacheFriends. Or https://www.apachefriends.org/
2. Install XAMPP and choose the components you need (ensure MySQL and phpMyAdmin are selected).
3. Complete the installation process and launch XAMPP Control Panel.

Step 2: Start the MySQL Server

1. Open the XAMPP Control Panel.
2. Start the **Apache** and **MySQL** modules by clicking the **Start** button next to them.
   - Green indicators show the services are running successfully

Step 3: Access phpMyAdmin

1. Open a web browser.
2. Navigate to http://localhost/phpmyadmin.
3. This is the phpMyAdmin interface where you can manage MySQL databases.

Web development

Example of back end design for the designed front end

Step 4: Create a Database

1. Open the XAMPP Control Panel and start the **Apache** and **MySQL** services.
2. Go to http://localhost/phpmyadmin in your browser.
3. Click on the **New** button in the left sidebar.
4. Name your database, e.g., user_registration, and click **Create**.
5. In the newly created database, click on the **SQL** tab or click **Create Table**.
6. Run the following SQL query or use the GUI to create the table:

```sql
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL
);
```

This table has:

- An id column as the primary key.
- **username** to store the user name.
- **email** to store unique user emails.
- **password** to store the password (hashed later for security).

Web development

Example of back end design for the designed front end

Step 5: Create a PHP Script for Database Connection

1. **Navigate to Your Project Folder:**

   In the htdocs directory, open your project folder (e.g., user_registration).

2. **Create a File for Database Connection:**

   Create a new file named db_connect.php and add the following code

```php
<?php
// Database credentials
$host = "localhost";
$username = "root"; // Default XAMPP username
$password = "";     // Default XAMPP password (leave empty)
$database = "user_registration";

// Create connection
$conn = new mysqli($host, $username, $password, $database);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>
```

Web development

Step 6: Create the User Registration Script

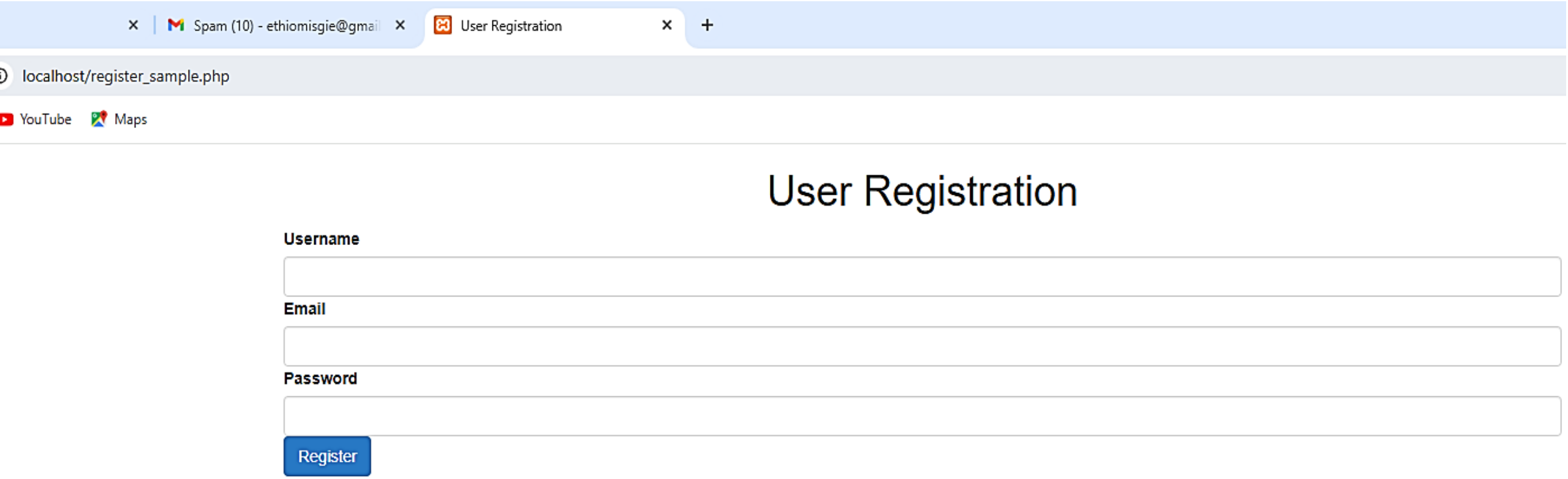
1.Create a register.php File:

- Add the following PHP code to handle form data and insert it into the database

```php
<?php
include 'db_connect.php';

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = mysqli_real_escape_string($conn, $_POST['username']);
    $email = mysqli_real_escape_string($conn, $_POST['email']);
    $password = password_hash($_POST['password'], PASSWORD_BCRYPT);

    $sql = "INSERT INTO users (username, email, password) VALUES ('$username', '$email', '$password')";

    if ($conn->query($sql) === TRUE) {
        echo "<div class='container mt-5'><div class='alert alert-success'>Registration successful!</div></div>";
    } else {
        echo "<div class='container mt-5'><div class='alert alert-danger'>Error: " . $conn->error . "</div></div>";
    }
}

$conn->close();
?>
```

Web development

The registration form will look like this



JavaScript

- **JavaScript** is a core technology for **building dynamic web applications**.
- Its main functions in dynamic web development include:
    - DOM manipulation
    - Event handling
    - Ajax and Fetch API
    - Form and input handling
    - Timer
    - Local and session storage
    - Modifying attributes and classes
    - Creaking and appending element and more

- JavaScript also defined as a **versatile programming language** primarily used to make **web pages interactive**.
- It allows developers to implement dynamic features such as responding to user actions, manipulating the DOM (Document Object Model), and communicating with servers.

DOM (data object model) functions

DOM Manipulation:

- document.getElementById(id) – Selects an element by its ID.
- document.getElementsByClassName(class) – Selects elements by their class name.
- document.getElementsByTagName(tag) – Selects elements by their tag name.
- document.querySelector(selector) – Selects the first element matching a CSS selector.
- document.querySelectorAll(selector) – Selects all elements matching a CSS selector.
- element.innerHTML – Gets or sets the HTML content inside an element.
- element.textContent – Gets or sets the text content inside an element.
- element.style – Modifies the inline styles of an element (e.g., element.style.color = "red";).

Event handling functions

Event Handling:

- element.addEventListener(event, function) – Adds an event listener to an element (e.g., click, keyup, submit).
- element.removeEventListener(event, function) – Removes an event listener from an element.
- event.preventDefault() – Prevents the default action of an event (e.g., preventing a form from submitting).
- event.stopPropagation() – Stops the event from bubbling up to parent elements

JavaScript

Ajax and fetch API functions

- XMLHttpRequest() – Used to send HTTP requests and handle responses asynchronously.
- fetch(url) – Fetches resources asynchronously from a network (commonly used for REST APIs).
- .then() – Handles the promise returned by fetch() (or other asynchronous operations).
- .catch() – Handles errors in the promise chain.

Form and input handling functions

- document.forms – Collection of all forms in the document.
- formElement.submit() – Submits a form programmatically.
- inputElement.value – Gets or sets the value of a form input element.
- formElement.reset() – Resets all input fields in a form to their default values.

JavaScript

Timers functions

- setTimeout(function, delay) – Executes a function after a specified delay (in milliseconds).
- setInterval(function, interval) – Repeatedly executes a function after every specified interval.
- clearTimeout(timeoutId) – Stops a setTimeout() before it completes.
- clearInterval(intervalId) – Stops a setInterval() from running further.

Local and Session Storage functions

- localStorage.setItem(key, value) – Stores data in the browser's local storage.
- localStorage.getItem(key) – Retrieves data from local storage.
- sessionStorage.setItem(key, value) – Stores data for the duration of the session.
- sessionStorage.getItem(key) – Retrieves data for the current session.

JavaScript

Array functions

- array.forEach() – Executes a function for each array element.
- array.map() – Creates a new array by applying a function to each element.
- array.filter() – Creates a new array with elements that pass a test function.
- array.find() – Returns the first element that satisfies the provided test function.
- array.reduce() – Reduces the array to a single value by applying a function to each element.

Modifying Attributes and Classes functions

- element.setAttribute(name, value) – Sets an attribute for an element.
- element.getAttribute(name) – Gets the value of an attribute.
- element.removeAttribute(name) – Removes an attribute from an element.
- element.classList.add(className) – Adds a class to an element.
- element.classList.remove(className) – Removes a class from an element.
- element.classList.toggle(className) – Toggles a class on or off.

JavaScript

Creating and appending element functions

- document.createElement(tag) – Creates a new HTML element.
- parentElement.appendChild(childElement) – Adds a child element to a parent.
- parentElement.removeChild(childElement) – Removes a child element from the parent.
- element.insertAdjacentHTML(position, html) – Inserts HTML at a specified position relative to the element (e.g., "beforebegin", "afterend").

Working with Events:

- event.target – Refers to the element that triggered the event.
- event.keyCode or event.code – Detects the key pressed in a keydown or keyup event.
- element.focus() – Sets focus on an element (useful for form inputs).
- element.blur() – Removes focus from an element.

JavaScript

Brief explanation of each **DOM manipulation** function with examples:

1. document.getElementById(id)

Explanation: This function retrieves an element by its unique ID.

Example

```
<p id="myParagraph">Hello World</p>
<script>
    const element = document.getElementById('myParagraph');
    element.style.color = 'blue'; // Changes the text color to blue
</script>
```

2. document.getElementsByClassName(class)

Explanation: Returns a collection of elements that have a specified class name. It returns a live HTMLCollection.

Example:

```
<p class="myClass">First Paragraph</p>
<p class="myClass">Second Paragraph</p>
<script>
    const elements = document.getElementsByClassName('myClass');
    elements[0].style.color = 'green'; // Changes the text color
</script>
```

JavaScript

Brief explanation of each **DOM manipulation** function with examples:

3.document.getElementsByTagName(tag)

Explanation: Retrieves all elements of a specified tag name and returns them as an HTML Collection.

Example:

```
<p>First Paragraph</p>
<p>Second Paragraph</p>
<script>
    const paragraphs = document.getElementsByTagName('p');
    paragraphs[1].textContent = 'Updated Paragraph'; // Changes the second paragraph
</script>
```

4. document.querySelector(selector)

Explanation: Returns the first element that matches a given CSS selector. It can be an ID, class, or any other valid CSS selector.

Example:

```
<p id="first">First Paragraph</p>
<p class="second">Second Paragraph</p>
<script>
    const element = document.querySelector('#first'); // Selects element by ID
    element.style.fontWeight = 'bold'; // Makes the text bold
</script>
```

JavaScript

Brief explanation of each **DOM manipulation** function with examples:

5. document.querySelectorAll(selector)

Explanation: Returns all elements that match a given CSS selector, as a static NodeList.

Example

```
<p class="myClass">First Paragraph</p>
<p class="myClass">Second Paragraph</p>
<script>
    const elements = document.querySelectorAll('.myClass');
    elements.forEach(el => el.style.color = 'red'); // Chang
</script>
```

6. element.innerHTML

Explanation: Gets or sets the HTML content inside an element.

Example:

```
<div id="myDiv">Old Content</div>
<script>
    document.getElementById('myDiv').innerHTML = '<p>New Content</p>';
</script>
```

JavaScript

Brief explanation of each **DOM manipulation** function with examples:

7. element.textContent

Explanation: Gets or sets the text content of an element, ignoring any HTML tags.

Example:

```
<div id="myDiv"><b>Bold Text</b></div>
<script>
    document.getElementById('myDiv').textContent = 'Plain Text';
</script>
```

8. element.style

Explanation: Directly modifies the inline styles of an element.

Example:

```
<p id="myParagraph">Styled Text</p>
<script>
    const element = document.getElementById('myParagraph');
    element.style.color = 'blue'; // Changes text color
    element.style.fontSize = '20px'; // Changes font size
</script>
```

JavaScript

Brief explanation of each **DOM manipulation** function with examples:

9. element.setAttribute(name, value)

Explanation: Adds a new attribute or changes the value of an existing attribute for the element.

Example:

```
<img id="myImage" src="oldImage.png">
<script>
    document.getElementById('myImage').setAttribute('src', 'newImage.png');
</script>
```

10. element.getAttribute(name)

Explanation: Retrieves the value of an attribute from an element.

Example:

```
<img id="myImage" src="image.png">
<script>
    const srcValue = document.getElementById('myImage').getAttribute('src');
    console.log(srcValue); // Outputs: image.png
</script>
```

JavaScript

Brief explanation of each **DOM manipulation** function with examples:

11. element.classList.add(className)

Explanation: Adds a new class to the element's class list.

Example:

```
<p id="myParagraph">Paragraph with new class</p>
<script>
    document.getElementById('myParagraph').classList.add('highlight');
</script>
```

12. element.classList.remove(className)

Explanation: Removes a specified class from an element.

Example:

```
<p id="myParagraph" class="highlight">Paragraph with class removed</p>
<script>
    document.getElementById('myParagraph').classList.remove('highlight');
</script>
```

JavaScript

Brief explanation of each **DOM manipulation** function with examples:

13. element.classList.toggle(className)

Explanation: Toggles a class on or off. If the class is present, it removes it; if not, it adds it.

Example:

```
<p id="myParagraph">Click to toggle class</p>
<script>
    const element = document.getElementById('myParagraph');
    element.addEventListener('click', () => {
        element.classList.toggle('highlight'); // Toggles '
    });
</script>
```

14. document.createElement(tagName)

Explanation: Creates a new HTML element of the specified tag name.

Example:

```
<div id="myDiv"></div>
<script>
    const newParagraph = document.createElement('p');
    newParagraph.textContent = 'This is a new paragraph';
    document.getElementById('myDiv').appendChild(newParagraph);
</script>
```

JavaScript

Brief explanation of each **DOM manipulation** function with examples:

15. element.appendChild(node)

Explanation: Adds a new child node to the specified element as the last child.

Example:

```
<div id="parentDiv"></div>
<script>
    const newElement = document.createElement('p');
    newElement.textContent = 'Appended paragraph';
    document.getElementById('parentDiv').appendChild(newElement);
</script>
```

16. element.removeChild(node)

Explanation: Removes a child node from the specified parent element.

Example:

```
<div id="parentDiv">
  <p id="childParagraph">Child Paragraph</p>
</div>
<script>
    const child = document.getElementById('childParagraph');
    document.getElementById('parentDiv').removeChild(child);
</script>
```

Java Script Event Handling

JavaScript event handling refers to the way JavaScript responds to events that happen in the web browser, such as clicks, key presses, form submissions, or page loading.

Brief explanation of each Event handling function.

1. addEventListener(event, handler)

This is the most common and flexible way to handle events. You can attach multiple event listeners to the same element and handle various types of events.

Example:

```
<button id="myButton">Click Me</button>
<script>
    const button = document.getElementById('myButton');
    button.addEventListener('click', () => {
        alert('Button Clicked!');
    });
</script>
```

JavaScript

Brief explanation of each Event handling function.

2. Inline Event Handlers (onclick, onmouseover, etc.)

You can directly specify event handlers within HTML attributes, but this approach is generally discouraged because it mixes HTML and JavaScript.

Example:

```
<button onclick="alert('Button Clicked!')">Click Me</button>
```

3. Event Handling using on[event] Properties

This method attaches an event handler directly to the on[event] property of an element (like onclick, onkeydown, etc.). It replaces any existing event handler for that event type.

Example

```
<button id="myButton">Click Me</button>
<script>
    const button = document.getElementById('myButton');
    button.onclick = () => {
        alert('Button Clicked!');
    };
</script>
```

Brief explanation of each Event handling function.

4. removeEventListener(event, handler)

This removes an event listener that was previously attached using addEventListener. To use this, the event handler function must be a named function, not an anonymous one.

Example:

```
<button id="myButton">Click Me</button>
<script>
    const button = document.getElementById('myButton');

    function handleClick() {
        alert('Button Clicked!');
    }

    button.addEventListener('click', handleClick);

    // Removing the event listener after 5 seconds
    setTimeout(() => {
        button.removeEventListener('click', handleClick);
    }, 5000);
</script>
```

Brief explanation of each Event handling function.

5. Event Object (event)

When an event is triggered, an event object is automatically passed to the event handler. This object contains information about the event, such as the type of event, the target element, and any additional data.

Example:

```
<button id="myButton">Click Me</button>
<script>
    const button = document.getElementById('myButton');
    button.addEventListener('click', (event) => {
        console.log('Event Type:', event.type); // Outputs 'click'
        console.log('Target Element:', event.target); // Outputs the clicked button
    });
</script>
```

Brief explanation of each Event handling function.

6. event.preventDefault()

This method is used to prevent the default behavior of an element, such as preventing a form from submitting, preventing a link from navigating, etc.

Example:

```
<a href="https://example.com" id="myLink">Click Me</a>
<script>
    const link = document.getElementById('myLink');
    link.addEventListener('click', (event) => {
        event.preventDefault(); // Prevents navigation
        alert('Navigation prevented');
    });
</script>
```

Brief explanation of each Event handling function.

7. event.stopPropagation()

This method prevents the event from bubbling up to parent elements. It's useful when you want to stop the event from triggering other event handlers on parent elements.

Example:

```
<div id="parentDiv">
    <button id="myButton">Click Me</button>
</div>
<script>
    const parentDiv = document.getElementById('parentDiv');
    const button = document.getElementById('myButton');

    parentDiv.addEventListener('click', () => {
        alert('Parent Div Clicked!');
    });

    button.addEventListener('click', (event) => {
        event.stopPropagation(); // Prevents the click event from reaching the parent
        alert('Button Clicked!');
    });
</script>
```

Brief explanation of each Event handling function.

9. Keyboard Events (keydown, keyup, keypress)

Keyboard events are triggered when a key is pressed. You can use these events to capture key presses and trigger specific actions based on the key code.

Example:

```
<input type="text" id="myInput" placeholder="Press a key">
<script>
    const input = document.getElementById('myInput');
    input.addEventListener('keydown', (event) => {
        alert(`Key pressed: ${event.key}`);
    });
</script>
```

10. Mouse Events (click, mouseover, mouseout, etc.)

Mouse events are triggered by mouse actions like clicking, hovering, or moving the cursor.

Example:

```
<button id="myButton">Hover Over Me</button>
<script>
    const button = document.getElementById('myButton');

    button.addEventListener('mouseover', () => {
        button.textContent = 'Mouse Over!';
    });

    button.addEventListener('mouseout', () => {
        button.textContent = 'Hover Over Me';
    });
</script>
```

JavaScript

Brief explanation of each Event handling function.

Form Events (submit, change, focus, etc.)

Form events are triggered when interacting with form elements like submitting a form, changing input values, or focusing on a field.

Example:

```
<form id="myForm">
    <input type="text" name="name" placeholder="Your Name">
    <button type="submit">Submit</button>
</form>
<script>
    const form = document.getElementById('myForm');

    form.addEventListener('submit', (event) => {
        event.preventDefault(); // Prevents form submission
        alert('Form submitted!');
    });
</script>
```

12. focus and blur Events

These events are triggered when an element gains or loses focus, respectively. They are often used with form elements.

Example:

```
<input type="text" id="myInput" placeholder="Focus on me">
<script>
    const input = document.getElementById('myInput');

    input.addEventListener('focus', () => {
        input.style.backgroundColor = 'lightyellow'; // Changes background on focus
    });

    input.addEventListener('blur', () => {
        input.style.backgroundColor = ''; // Resets background when focus is lost
    });
</script>
```

Brief explanation of each Event handling function.

13. Window Events (load, resize, scroll)

These events are triggered on the window object, such as when the page is loaded, resized, or scrolled.

Example:

```
<script>
    window.addEventListener('resize', () => {
        console.log(`Window resized to: ${window.innerWidth} x ${window.innerHeight}`);
    });
</script>
```

JavaScript

Form and Input Handling using JavaScript

1. Handling Form Submission

When a user submits a form, JavaScript can handle the submission event and prevent the page from refreshing. This is done using the submit event and the **event.preventDefault()** method to control form behavior.

Example:

```
<form id="myForm">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required>
    <button type="submit">Submit</button>
</form>
<div id="output"></div>

<script>
    document.getElementById('myForm').addEventListener('submit', function(event) {
        event.preventDefault(); // Prevents the page from refreshing

        // Get the form value
        const name = document.getElementById('name').value;

        // Output the value to the page
        document.getElementById('output').textContent = `Hello, ${name}!`;
    });
</script>
```

Explanation:

- The form has an id="myForm", and JavaScript listens to the submit event.
- event.preventDefault() prevents the form from refreshing the page.
- JavaScript retrieves the value of the input field using .value and displays it in a div.

JavaScript

Form and Input Handling using JavaScript

2. Input Validation

JavaScript can validate input fields before submission. This can include checking if fields are filled out, ensuring data is in a correct format (e.g., email, phone number), or checking specific criteria (like passwords).

Example

```
<form id="loginForm">
    <label for="username">Username:</label>
    <input type="text" id="username" required>
    <label for="password">Password:</label>
    <input type="password" id="password" required>
    <button type="submit">Login</button>
</form>
<div id="loginOutput"></div>

<script>
    Complexity is 4 Everything is cool!
    document.getElementById('loginForm').addEventListener('submit', function(event) {
        event.preventDefault();

        const username = document.getElementById('username').value;
        const password = document.getElementById('password').value;

        if (username === '' || password === '') {
            alert('All fields are required.');
        } else if (password.length < 6) {
            alert('Password must be at least 6 characters long.');
        } else {
            document.getElementById('loginOutput').textContent = `Welcome, ${username}!`;
        }
    });
</script>
```

Explanation:

•The form validation checks that both the username and password are filled out.

•If the password length is less than 6 characters, the user is alerted.

•If all validations pass, the username is displayed on the page.

JavaScript

Form and Input Handling using JavaScript

3. Handling Checkbox Input

Checkboxes allow users to select multiple options. JavaScript can check which checkboxes are selected and handle the form data accordingly.

Example:

```
<form id="hobbyForm">
    <label>Select your hobbies:</label><br>
    <input type="checkbox" id="reading" name="hobby" value="Reading"> Reading<br>
    <input type="checkbox" id="traveling" name="hobby" value="Traveling"> Traveling<br>
    <input type="checkbox" id="coding" name="hobby" value="Coding"> Coding<br>
    <button type="submit">Submit</button>
</form>
<div id="hobbyOutput"></div>

<script>
    document.getElementById('hobbyForm').addEventListener('submit', function(event) {
        event.preventDefault();

        const hobbies = [];
        const checkboxes = document.querySelectorAll('input[name="hobby"]:checked');

        checkboxes.forEach(checkbox => {
            hobbies.push(checkbox.value);
        });

        document.getElementById('hobbyOutput').textContent = `Your hobbies are: ${hobbies.join
        (', ')}`;
    });
</script>
```

Explanation:

- document.querySelectorAll('input[name="hobby"]:checked') gets all checked checkboxes.
- The selected values are stored in an array and displayed in the div.

Form and Input Handling using JavaScript

4. Handling Radio Button Input

Radio buttons allow users to select only one option from a group. JavaScript can capture the selected value for further processing.

Example:

```
<form id="genderForm">
    <label>Select your gender:</label><br>
    <input type="radio" id="male" name="gender" value="Male" required> Male<br>
    <input type="radio" id="female" name="gender" value="Female" required> Female<br>
    <button type="submit">Submit</button>
</form>
<div id="genderOutput"></div>

<script>
    document.getElementById('genderForm').addEventListener('submit', function(event) {
        event.preventDefault();

        const gender = document.querySelector('input[name="gender"]:checked').value;
        document.getElementById('genderOutput').textContent = `You selected: ${gender}`;
    });
</script>
```

Explanation:

- JavaScript uses document.querySelector('input[name="gender"]:checked') to get the selected radio button value.
- The selected gender is displayed in a div.

Form and Input Handling using JavaScript

5. Handling Select Dropdown

A dropdown (select) input lets users choose one or more options from a list. You can handle single and multi-select dropdowns using JavaScript.

Example (Single Select):

```
<form id="colorForm">
    <label for="color">Choose a color:</label>
    <select id="color" name="color">
        <option value="Red">Red</option>
        <option value="Green">Green</option>
        <option value="Blue">Blue</option>
    </select>
    <button type="submit">Submit</button>
</form>
<div id="colorOutput"></div>

<script>
    document.getElementById('colorForm').addEventListener('submit', function(event) {
        event.preventDefault();

        const color = document.getElementById('color').value;
        document.getElementById('colorOutput').textContent = `You chose: ${color}`;
    });
</script>
```

Explanation:

- document.getElementById('color').value gets the selected value from the dropdown.
- The selected color is displayed when the form is submitted.

Form and Input Handling using JavaScript

6. Handling Textarea Input

A textarea is used for multi-line text input. Handling it in JavaScript is similar to handling other text inputs.

Example:

```
<form id="messageForm">
    <label for="message">Your Message:</label><br>
    <textarea id="message" rows="4" cols="50" required></textarea><br>
    <button type="submit">Submit</button>
</form>
<div id="messageOutput"></div>

<script>
    document.getElementById('messageForm').addEventListener('submit', function(event) {
        event.preventDefault();

        const message = document.getElementById('message').value;
        document.getElementById('messageOutput').textContent = `Your message: ${message}`;
    });
</script>
```

Explanation:

- The value of the textarea is accessed using document.getElementById ('message').value.
- The user's message is displayed in a div.

Form and Input Handling using JavaScript

7. Handling File Input

File input allows users to upload files. JavaScript can handle the file selection and display information like the file name or size.

Example:

```
<form id="fileForm">
    <label for="file">Upload a file:</label>
    <input type="file" id="file" name="file" required><br>
    <button type="submit">Upload</button>
</form>
<div id="fileOutput"></div>

<script>
    document.getElementById('fileForm').addEventListener('submit', function(event) {
        event.preventDefault();

        const file = document.getElementById('file').files[0];
        if (file) {
            document.getElementById('fileOutput').textContent = `File selected: ${file.name}`;
        }
    });
</script>
```

Explanation:

- document.getElementById('file').files[0] accesses the selected file (if any).
- The file's name is displayed after selection.

Timers in JavaScript

Timers in JavaScript are used to execute a piece of code after a specified interval of time or to execute code repeatedly at certain intervals. JavaScript provides two main functions for handling timers:

1.setTimeout(): Executes code after a specified delay (in milliseconds).

2.setInterval(): Repeatedly executes code at a specified interval (in milliseconds).

Both functions have a corresponding method to cancel them:

- **clearTimeout()**: Cancels a setTimeout().
- **clearInterval()**: Cancels a setInterval().

Timers in JavaScript

1. setTimeout()

setTimeout() allows you to execute a function or a block of code **after a specified delay**. The delay is measured in milliseconds (1 second = 1000 milliseconds).

Explanation:

•The message "Hello! This message appears after 3 seconds!" will be displayed after a 3-second delay.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>setTimeout Example</title>
</head>
<body>
    <h1>Welcome to the Page!</h1>
    <p id="message"></p>

    <script>
        // Display a message after 3 seconds
        setTimeout(function() {
            document.getElementById('message').textContent = 'Hello! This message appears
        }, 3000); // 3000 milliseconds = 3 seconds
    </script>
</body>
</html>
```

Timers in JavaScript

2. clearTimeout()

You can cancel the execution of a setTimeout() before the delay expires using clearTimeout().

Explanation:

•The message is scheduled to display after 5 seconds.

•If the "Cancel Timer" button is clicked before 5 seconds, the timer is canceled, and the message "Timer canceled!" is displayed instead.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>clearTimeout Example</title>
</head>
<body>
    <h1>setTimeout and clearTimeout Example</h1>
    <p id="message"></p>
    <button id="cancelBtn">Cancel Timer</button>

    <script>
        // Schedule a message to be shown after 5 seconds
        let timeoutID = setTimeout(function() {
            document.getElementById('message').textContent = 'This message appears after 5
            seconds!';
        }, 5000);

        // Cancel the timer when the button is clicked
        document.getElementById('cancelBtn').addEventListener('click', function() {
            clearTimeout(timeoutID);
            document.getElementById('message').textContent = 'Timer canceled!';
        });
    </script>
</body>
</html>
```

JavaScript

Timers in JavaScript

3. setInterval()

setInterval() allows you to execute a function repeatedly at a specified interval. The interval is also measured in milliseconds.

Explanation:

- Every second, the counter displayed in the paragraph will increment by 1.
- This process will continue indefinitely, or until clearInterval() is called.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>setInterval Example</title>
</head>
<body>
    <h1>setInterval Example</h1>
    <p id="counter">0</p>

    <script>
        let count = 0;

        // Increment the counter every second (1000 milliseconds)
        let intervalID = setInterval(function() {
            count++;
            document.getElementById('counter').textContent = count;
        }, 1000);
    </script>
</body>
</html>
```

Timers in JavaScript

4. clearInterval()

You can stop the execution of setInterval() using clearInterval().

Explanation:

•The counter increments every second, just like before.

•Clicking the "Stop Counter" button will stop the interval, and the counter will stop at its current value.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>clearInterval Example</title>
</head>
<body>
    <h1>Counter with setInterval and clearInterval</h1>
    <p id="counter">0</p>
    <button id="stopBtn">Stop Counter</button>

    <script>
        let count = 0;

        // Increment the counter every second
        let intervalID = setInterval(function() {
            count++;
            document.getElementById('counter').textContent = count;
        }, 1000);

        // Stop the counter when the button is clicked
        document.getElementById('stopBtn').addEventListener('click', function() {
            clearInterval(intervalID);
            document.getElementById('counter').textContent = 'Counter stopped at ' + count;
        });
    </script>
</body>
</html>
```

Timers in JavaScript

5. Nested setTimeout() vs setInterval()

When you need precise timing for repeated actions (like game loops or animations), using **nested setTimeout()** instead of setInterval() is often preferred. Nested setTimeout() schedules the next execution only after the current execution finishes, preventing potential overlaps or inaccuracies.

Example: Nested setTimeout() (Simulating setInterval())

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Nested setTimeout Example</title>
</head>
<body>
    <h1>Nested setTimeout Example</h1>
    <p id="counter">0</p>

    <script>
        let count = 0;

        function incrementCounter() {
            count++;
            document.getElementById('counter').textContent = count;

            // Schedule the next increment after 1 second
            setTimeout(incrementCounter, 1000);
        }

        // Start the counter
        setTimeout(incrementCounter, 1000);
    </script>
</body>
</html>
```

Explanation:

- In this example, setTimeout() schedules the next call to incrementCounter() after 1 second, creating an effect similar to setInterval().
- This approach ensures that the next execution will not begin until the previous one is completely finished.

Review questions

1. What functionalities are performed using event handler?
2. Why JavaScript use DOM?
3. Why we use JavaScript in our web application development?
4. Write a JavaScript program to pop up the message “Welcome computer engineering students” while pressing “alert button”. You have to write HTML to design the button named “alert”
5. Write the JavaScript code that can change the paragraph “Welcome computer engineering students” to “welcome to software engineering class” while pressing “change button”. You have to use HTML code to display the first message and “change button”.
6. Write a JavaScript code to change the red “Subscribe” button to green ‘’Subscribed” button when you click on the button. Use HTML and CSS to create initial red colored “subscribe” button.

Review questions

7. Let you have list of fruits such as banana, orange, apple and Avocado as checkbox form. Let you want select two from the list. Write JavaScript code to perform this activity. The code should display **“you selected …… and ……. From the list”** message on the alert box. ……. Is the name of selected fruit.

8. Let you have list of fashion such as shoes, trouser, T-shirt and Jacket as radio button form. Let you want to select one from the list. Write JavaScript code to perform this activity. The code should display **“you selected …… and ……. From the list”** message on the alert box. ……. Is the name of selected fashion.

9. Let you have list of computer such as Lenevo, Thoshiba, Dell and Access as dropdown list form. Let you want to select three from the dropdown list. Write JavaScript code to perform this activity. The code should display **“you selected ……, …… and ……. From the list**” message on the alert box. ……. Is the name of selected fashion.

Review questions

10. Write a JavaScript code to display the message “key pressed” when you pressed “keyEvent button”

11. Write a JavaScript code to display your name on new window while you enter your name in text input feiled and click on “submit” button.

12. Write a JavaScript program that will do the following tasks.
    - The code enables you to enter password in the first input field and reenter the password in the second input field
    - While pressing “compare” button, compare the password input field and confirm password input field
    - If the password in the two input field do not matched, it will display error message with red color.

14. Write a JavaScript program that checks whether the password is grater than 8 character or not while pressing “login” button

Review questions

15. Repeat question 14 to check whether the input password is Mix of letters, numbers and characters.
16. Create two button named as “count UP” and “count Down”. Write a JavaScript code that increase the number from 0 to 10 while pressing “count UP” button and decrease the count from 10 to 0 when pressing “count Down button”.
17. Repeat question 16 to increase and decrees the counter in 3 second interval. Hint. Use a special JavaScript function setInterval().
18. Create a button named as “AddToCart” and write a JavaScript code that increase cart value when pressing the button. You can use paragraph to show the amount of cart to be increased while clicking the button.
19. Write a JavaScript program that can validate user registration. There will be input fields to enter first name, username, password and confirm password. There will be also “submit button”. There will be “successfully registered message on alert box if all input are not empty, password field are combination of Upper case and lower case letters, numbers and characters, password and confirm password are matched. Otherwise there will be failed message on the alert box.

End of chapter 5