

1. Build a responsive web application for shopping cart with registration, login, catalog and cart pages using CSS3 features, flex and grid.

AIM:

Build a responsive web application for shopping cart with registration, login, catalog and cart pages using CSS3 features, flex and grid.

DESCRIPTION:

HTML, CSS, and JavaScript are essential components for creating a functional responsive web application. A condensed example of a shopping cart with registration, login, catalogue, and cart pages is provided.

Project Structure:

1. **index.html** - Main HTML file containing the structure of the web application.
2. **styles.css** - CSS file for styling the web pages.
3. **script.js** - JavaScript file for handling interactions and logic.
4. **images/** - Folder for storing images.

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="styles.css">
  <title>Shopping Cart</title>
</head>
<body>
  <header>
    <h1>Shopping Cart</h1>
    <nav>
      <ul>
        <li><a href="#catalog">Catalog</a></li>
        <li><a href="#cart">Cart</a></li>
        <li><a href="#login">Login</a></li>
        <li><a href="#register">Register</a></li>
      </ul>
    </nav>
  </header>
  <main id="content">
    <!-- Content will be loaded dynamically using JavaScript -->
  </main>
  <script src="script.js"></script>
</body>
</html>
```

styles.css:

```
body {
  font-family: 'Arial', sans-serif;
  margin: 0;
  padding: 0;
}
header {
```

```
background-color: #333;
color: #fff;
padding: 10px;
text-align: center;
}
nav ul {
list-style: none;
padding: 0;
display: flex;
justify-content: center;
}
nav li {
margin: 0 10px;
}
main {
padding: 20px;
}
/* Add more styles based on your design */
script.js:
// Dummy data for the catalog
const catalog = [
  { id: 1, name: 'Product 1', price: 20 },
  { id: 2, name: 'Product 2', price: 30 },
  { id: 3, name: 'Product 3', price: 25 },
];
// Function to load the catalog
function loadCatalog() {
  const catalogContainer = document.getElementById('content');
  catalogContainer.innerHTML = '<h2>Catalog</h2>';
  catalog.forEach(product => {
    const productCard = document.createElement('div');
    productCard.classList.add('product-card');
    productCard.innerHTML = `
      <h3>${product.name}</h3>
      <p>${product.price}</p>
      <button onclick="addToCart(${product.id})">Add to Cart</button>
    `;
    catalogContainer.appendChild(productCard);
  });
}
// Function to add a product to the cart
function addToCart(productId) {
  // Implement cart functionality here
  console.log(`Product ${productId} added to cart`);
}
// Initial load
loadCatalog();
```

Explanation

1. **HTML Structure:** The HTML file consists of a header, navigation, and a main content area, initially empty but dynamically populated using JavaScript.
2. **CSS Styles:** The CSS file offers basic styling for header, navigation, and main content area, which can be customized according to your design needs.
3. **JavaScript Logic:** The JavaScript file contains dummy catalog data and functions for loading and adding products to the cart, allowing real-world interaction with a server for catalog data retrieval and user cart management.

Output:

- The shopping cart application's basic structure is displayed in index.html, with the catalog section containing dummy product data.

Note: The example is intentionally simplified, and you would need to add more functionality, such as user authentication, cart management, and server communication, for a fully functional shopping cart application.

2. Make the above web application responsive web application using Bootstrap framework**AIM: Make the above web application responsive web application using Bootstrap framework****DESCRIPTION:**

Bootstrap is a popular CSS framework that makes it easy to create responsive web applications. The previous example can be modified using Bootstrap by following these steps:

Project Structure:

1. index.html - Main HTML file containing the structure of the web application with Bootstrap.
2. script.js - JavaScript file for handling interactions and logic (no changes from the previous example).
3. styles.css - You can include additional custom styles if needed.
4. images/ - Folder for storing images.

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
  <!-- Custom CSS -->
  <link rel="stylesheet" href="styles.css">
  <title>Shopping Cart</title>
</head>
<body>
  <header class="bg-dark text-white text-center py-3">
    <h1>Shopping Cart</h1>
    <nav>
      <ul class="nav justify-content-center">
        <li class="nav-item"><a class="nav-link" href="#catalog">Catalog</a></li>
        <li class="nav-item"><a class="nav-link" href="#cart">Cart</a></li>
        <li class="nav-item"><a class="nav-link" href="#login">Login</a></li>
        <li class="nav-item"><a class="nav-link" href="#register">Register</a></li>
      </ul>
    </nav>
  </header>
  <main class="container mt-3" id="content">
    <!-- Content will be loaded dynamically using JavaScript -->
  </main>
  <!-- Bootstrap JS (optional, for certain features) -->
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
  <script src="script.js"></script>
</body>
</html>
```

styles.css:

/ You can include additional custom styles here if needed */*

Explanation:

1. Bootstrap Integration: In the <head> section, we added links to the Bootstrap CSS and JS files from a CDN (Content Delivery Network). This allows us to use Bootstrap's styling and functionality.
2. Bootstrap Classes: We applied Bootstrap classes to the HTML elements. For example, we used container to create a responsive fixed-width container and various utility classes for styling the header and navigation.
3. Responsive Navigation: Bootstrap's grid system and utility classes help in creating a responsive navigation bar. The justify-content-center class is used to center the navigation links.
4. Responsive Main Content: The container class ensures that the main content area is responsive. Bootstrap will automatically adjust the width based on the screen size.

Output:

When you open index.html in a web browser, you'll see that the web application is now responsive.

The Bootstrap framework takes care of making the layout adapt to different screen sizes, providing a more user-friendly experience on various devices.

Remember to test the responsiveness by resizing your browser or using different devices to see how the layout adjusts.

3. Use JavaScript for doing client – side validation of the pages implemented in experiment 1 and experiment 2

AIM:

Use JavaScript for doing client – side validation of the pages implemented in experiment 1: Build a responsive web application for shopping cart with registration, login, catalog and cart pages using CSS3 features, flex and grid and experiment 2: Make the above web application responsive web application using Bootstrap framework

DESCRIPTION:

To perform client-side validation using JavaScript, you can add scripts to validate user inputs on the registration and login pages.

The modifications for both experiments are listed below.

Experiment 1: Responsive Web Application without Bootstrap

Add the following JavaScript code to **script.js**:

```
// Function to validate registration form
function validateRegistration() {
  const username = document.getElementById('username').value;
  const password = document.getElementById('password').value;
  if (username.trim() === "" || password.trim() === "") {
    alert('Please enter both username and password.');
```

return false;

```
  }
  // Additional validation logic can be added as needed
  return true;
}

// Function to validate login form
function validateLogin() {
  const username = document.getElementById('loginUsername').value;
  const password = document.getElementById('loginPassword').value;
  if (username.trim() === "" || password.trim() === "") {
    alert('Please enter both username and password.');
```

return false;

```
  }
  // Additional validation logic can be added as needed
  return true;
}
```

Modify the HTML login and registration forms:

```
<!-- Registration Form -->
<form onsubmit="return validateRegistration()">
  <!-- ... existing form fields ... -->
  <button type="submit">Register</button>
</form>

<!-- Login Form -->
<form onsubmit="return validateLogin()">
  <!-- ... existing form fields ... -->
  <button type="submit">Login</button>
</form>
```

Experiment 2: Responsive Web Application with Bootstrap

Add the following JavaScript code to **script.js**:

```
// Function to validate registration form
function validateRegistration() {
  const username = document.getElementById('username').value;
  const password = document.getElementById('password').value;
  if (username.trim() === "" || password.trim() === "") {
    alert('Please enter both username and password.');
```

return false;

```
  }
  // Additional validation logic can be added as needed
  return true;
}

// Function to validate login form
function validateLogin() {
  const username = document.getElementById('loginUsername').value;
  const password = document.getElementById('loginPassword').value;
  if (username.trim() === "" || password.trim() === "") {
    alert('Please enter both username and password.');
```

return false;

```
  }
  // Additional validation logic can be added as needed
  return true;
}
```

Modify the Bootstrap login and registration forms:

```
<!-- Registration Form -->
<form onsubmit="return validateRegistration()" class="needs-validation" novalidate>
  <!-- ... existing form fields ... -->
  <button type="submit" class="btn btn-primary">Register</button>
</form>

<!-- Login Form -->
<form onsubmit="return validateLogin()" class="needs-validation" novalidate>
  <!-- ... existing form fields ... -->
  <button type="submit" class="btn btn-primary">Login</button>
</form>
```

Explanation:

1. **Validation Functions:** We added two validation functions, **validateRegistration** and **validateLogin**, which retrieve form inputs and perform basic validation. Feel free to add more complex validation logic as needed.
2. **Form Modification:** We added the **onsubmit** attribute to the registration and login forms to call the respective validation functions when the forms are submitted. Additionally, for Bootstrap forms, we added the **novalidate** attribute to prevent the default browser validation, allowing us to handle validation using JavaScript.

Remember that client-side validation is essential for a better user experience, but server-side validation is crucial for security and should always be implemented. The provided examples cover basic client-side validation; you may enhance them based on your specific requirements.

4. Explore the features of ES6 like arrow functions, callbacks, promises, async/await. Implement an application for reading the weather information from openweathermap.org and display the information in the form of a graph on the web page.

AIM:

Explore the features of ES6 like arrow functions, callbacks, promises, async/await. Implement an application for reading the weather information from openweathermap.org and display the information in the form of a graph on the web page.

DESCRIPTION:

To implement an application for reading weather information from OpenWeatherMap.org and displaying the information in the form of a graph, we can use JavaScript with ES6 features like arrow functions, callbacks, promises, and async/await. For simplicity, we'll use the **axios** library to make HTTP requests and **Chart.js** for creating the graph.

The inclusion of these libraries is crucial for your project.

Project Structure:

1. **index.html** - Main HTML file.
2. **script.js** - JavaScript file for handling weather data and graph creation.
3. **styles.css** - CSS file for styling.
4. **node_modules/** - Folder for library dependencies.

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="styles.css">
  <title>Weather Graph</title>
</head>
<body>
  <div class="container">
    <h1>Weather Graph</h1>
    <canvas id="weatherGraph" width="400" height="200"></canvas>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <script src="script.js"></script>
</body>
</html>
```

styles.css:

```
body {
  font-family: 'Arial', sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f4f4f4;
}
.container {
  max-width: 600px;
  margin: 50px auto;
  background-color: #fff;
```



```
padding: 20px;
border-radius: 8px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
h1 {
text-align: center;
}
canvas {
display: block;
margin: 20px auto;
}
```

script.js:

```
document.addEventListener('DOMContentLoaded', () => {
const apiKey = 'YOUR_OPENWEATHERMAP_API_KEY';
const city = 'YOUR_CITY_NAME';
const apiUrl = `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${
apiKey}&units=metric`;
const fetchData = async () => {
try {
const response = await axios.get(apiUrl);
const weatherData = response.data;
updateGraph(weatherData.main.temp);
} catch (error) {
console.error('Error fetching weather data:', error.message);
}
};
const updateGraph = (temperature) => {
const ctx = document.getElementById('weatherGraph').getContext('2d');
new Chart(ctx, {
type: 'bar',
data: {
labels: ['Temperature'],
datasets: [{
label: 'Temperature (°C)',
data: [temperature],
backgroundColor: ['#36A2EB'],
}],
},
options: {
scales: {
y: {
beginAtZero: true,
},
},
},
});
};
fetchData();
```

});

Explanation:

1. **HTML Structure:** We have a simple HTML structure with a container for the graph canvas.
2. **CSS Styling:** Basic styling to make the application look presentable.
3. **JavaScript (script.js):**
 - The `fetchData` function uses `axios` to make an asynchronous HTTP request to OpenWeatherMap API.
 - The retrieved weather data is used to update the graph using the `updateGraph` function.
 - The graph is created using the `Chart.js` library.
4. **API Key:** Replace `'YOUR_OPENWEATHERMAP_API_KEY'` with your OpenWeatherMap API key and `'YOUR_CITY_NAME'` with the desired city.

Output:

When you open `index.html` in a web browser, the application fetches the current weather information for the specified city from OpenWeatherMap API and displays the temperature on a bar graph. Please note that you need to replace `'YOUR_OPENWEATHERMAP_API_KEY'` with your actual API key. Additionally, the responsiveness and appearance can be further enhanced based on your design preferences.

5. Develop a java stand alone application that connects with the database (Oracle / mySql) and perform the CRUD operation on the database tables.

AIM: Develop a java stand alone application that connects with the database (Oracle / mySql) and perform the CRUD operation on the database tables.

DESCRIPTION:

let's create a simple Java standalone application that connects to a MySQL database and performs CRUD (Create, Read, Update, Delete) operations on a table. For this example, we'll use JDBC (Java Database Connectivity) to interact with the MySQL database.

Prerequisites:

1. Make sure you have MySQL installed, and you know the database name, username, and password.
2. Download the MySQL JDBC driver (JAR file) from [MySQL Connector/J](#) and include it in your project.

Example Java Application:

Let's assume we have a table named **employees** with columns **id**, **name**, and **salary**.

import java.sql.*;

public class CRUExample {

// JDBC URL, username, and password of MySQL server

private static final String JDBC_URL = "jdbc:mysql://localhost:3306/your_database";

private static final String USERNAME = "your_username";

private static final String PASSWORD = "your_password";

public static void main(String[] args) {

try {

// Step 1: Establishing a connection

Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);

// Step 2: Creating a statement

Statement statement = connection.createStatement();

// Step 3: Performing CRUD operations

createRecord(statement, "John Doe", 50000);

readRecords(statement);

updateRecord(statement, 1, "John Updated", 55000);

readRecords(statement);

deleteRecord(statement, 1);

readRecords(statement);

// Step 4: Closing resources

statement.close();

connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

// Create a new record in the database

private static void createRecord(Statement statement, String name, int salary) throws

SQLException {

String insertQuery = "INSERT INTO employees (name, salary) VALUES ('" + name + "', " + salary + ")";

statement.executeUpdate(insertQuery);

```

    System.out.println("Record created successfully.");
}
// Read all records from the database
private static void readRecords(Statement statement) throws SQLException {
    String selectQuery = "SELECT * FROM employees";
    ResultSet resultSet = statement.executeQuery(selectQuery);
    System.out.println("ID\tName\tSalary");
    while (resultSet.next()) {
        int id = resultSet.getInt("id");
        String name = resultSet.getString("name");
        int salary = resultSet.getInt("salary");
        System.out.println(id + "\t" + name + "\t" + salary);
    }
    System.out.println();
}
// Update a record in the database
private static void updateRecord(Statement statement, int id, String newName, int newSalary)
throws SQLException {
    String updateQuery = "UPDATE employees SET name = " + newName + ", salary = " +
newSalary + " WHERE id = " + id;
    statement.executeUpdate(updateQuery);
    System.out.println("Record updated successfully.");
}
// Delete a record from the database
private static void deleteRecord(Statement statement, int id) throws SQLException {
    String deleteQuery = "DELETE FROM employees WHERE id = " + id;
    statement.executeUpdate(deleteQuery);
    System.out.println("Record deleted successfully.");
}
}

```

Explanation:

1. **Connection Parameters:** Replace your_database, your_username, and your_password with your actual database name, username, and password.
2. **JDBC Connection:** The getConnection method is used to establish a connection to the MySQL database.
3. **CRUD Operations:**
 - createRecord: Inserts a new record into the database.
 - readRecords: Reads and prints all records from the database.
 - updateRecord: Updates a record in the database.
 - deleteRecord: Deletes a record from the database.
4. **Output:** The application prints the records before and after each operation, showing the CRUD operations in action.

Output:

When you run this Java application, you should see output similar to the following in the console:

Record created successfully.

ID	Name	Salary
1	John Doe	50000

Record updated successfully.

ID	Name	Salary
1	John Updated	55000

Record deleted successfully.

ID	Name	Salary
----	------	--------

This demonstrates a simple Java standalone application for CRUD operations on a MySQL database using JDBC. Keep in mind that for a production environment, you would want to use prepared statements to prevent SQL injection and handle exceptions more gracefully.

6. Create an xml for the bookstore. Validate the same using both DTD and XSD**AIM: Create an xml for the bookstore. Validate the same using both DTD and XSD****DESCRIPTION:**

Let's create an XML file for a simple bookstore and validate it using both Document Type Definition (DTD) and XML Schema Definition (XSD).

Bookstore XML File (bookstore.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE bookstore SYSTEM "bookstore.dtd">
<bookstore>
  <book>
    <title>Introduction to XML</title>
    <author>John Doe</author>
    <price>29.99</price>
  </book>
  <book>
    <title>Web Development Basics</title>
    <author>Jane Smith</author>
    <price>39.95</price>
  </book>
  <!-- Add more book entries as needed -->
</bookstore>
```

DTD File (bookstore.dtd):

```
<!ELEMENT bookstore (book+)>
<!ELEMENT book (title, author, price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

XSD File (bookstore.xsd):

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="bookstore" type="bookstoreType"/>
  <xs:complexType name="bookstoreType">
    <xs:sequence>
      <xs:element name="book" type="bookType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="bookType">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string"/>
      <xs:element name="price" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Explanation:

1. **XML File (bookstore.xml):** Represents a simple XML structure with a list of books in a bookstore.

2. **DTD File (bookstore.dtd):**

Describes the structure of the XML document using Document Type Definition.

Specifies that a bookstore must contain one or more book elements.

Each book must contain title, author, and price elements.

3. **XSD File (bookstore.xsd):**

Describes the structure of the XML document using XML Schema Definition.

Defines complex types for bookstore and book.

Specifies that a bookstore must contain an unbounded sequence of book elements.

Each book must contain title (string), author (string), and price (decimal) elements.

Validation:

You can validate the XML file using a tool or programming language that supports DTD and XSD validation. Here, I'll show you how to do it using a Java program.

Java Program for Validation:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import javax.xml.XMLConstants;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
import org.xml.sax.SAXException;

public class XMLValidator {
    public static void main(String[] args) {
        validateWithDTD("bookstore.xml", "bookstore.dtd");
        validateWithXSD("bookstore.xml", "bookstore.xsd");
    }

    private static void validateWithDTD(String xmlFile, String dtdFile) {
        try {
            InputStream xmlStream = new FileInputStream(new File(xmlFile));
            InputStream dtdStream = new FileInputStream(new File(dtdFile));
            SchemaFactory schemaFactory =
                SchemaFactory.newInstance(XMLConstants.XML_DTD_NS_URI);
            Schema schema = schemaFactory.newSchema(new StreamSource(dtdStream));
            Validator validator = schema.newValidator();
            validator.validate(new StreamSource(xmlStream));
            System.out.println("Validation with DTD successful.");
        } catch (SAXException | IOException e) {
            System.out.println("Validation with DTD failed. Reason: " + e.getMessage());
        }
    }

    private static void validateWithXSD(String xmlFile, String xsdFile) {
        try {
            InputStream xmlStream = new FileInputStream(new File(xmlFile));
```

```
InputStream xsdStream = new FileInputStream(new File(xsdFile));
SchemaFactory schemaFactory =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
Schema schema = schemaFactory.newSchema(new StreamSource(xsdStream));
Validator validator = schema.newValidator();
validator.validate(new StreamSource(xmlStream));
System.out.println("Validation with XSD successful.");
} catch (SAXException | IOException e) {
    System.out.println("Validation with XSD failed. Reason: " + e.getMessage());
}
}
```

Output:

If the XML file adheres to the specified DTD and XSD, you will see output messages like:

Validation with DTD successful.

Validation with XSD successful.

If there are issues with the XML file, the program will print error messages explaining the validation failure.

Make sure to replace "bookstore.xml", "bookstore.dtd", and "bookstore.xsd" with the actual file paths in your project. Also, you can use various online XML validators to validate your XML files against DTD and XSD if you prefer a web-based approach.