**LAB EXPERIMENT**

**Question:**

Write Bootstrap 5 code to design a responsive layout using .container and .container-fluid classes. Add a heading and paragraph inside each container and show the difference between them.

**Title:**

Responsive Layout Design using .container and .container-fluid in Bootstrap 5

**Aim:**

To develop a responsive web page using Bootstrap 5 that demonstrates the difference between the .container (fixed-width layout) and .container-fluid (full-width layout) classes.

**Procedure:**

1. **Create an HTML File:**  
   Open VS Code and create a new file named container\_example.html.
2. **Add Bootstrap 5 CDN:**  
   Use the Bootstrap 5 CSS link inside the <head> tag to import Bootstrap styling.
3. **Design a Fixed Container Section:**  
   Use <div class="container"> to create a box that adjusts based on device screen width.
4. **Design a Fluid Container Section:**  
   Use <div class="container-fluid"> to create a full-width responsive section.
5. **Add Heading and Paragraph:**  
   Add <h2> and <p> inside each container to show the layout clearly.
6. **Add Some Styles:**  
   Use Bootstrap utility classes like bg-light, bg-info, p-4, and mt-4 for padding and margin spacing.
7. **Run the Code:**  
   Open the file in a browser and resize the window to see how .container and .container-fluid behave differently.

**Program Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Bootstrap Container Example</title>

<!-- Bootstrap 5 CDN -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<!-- Fixed width container -->

<div class="container mt-4 p-4 bg-light border rounded">

<h2>Container Example (Fixed Width)</h2>

<p>

This content is inside a <code>.container</code> class.

The container width adjusts automatically based on screen size

(e.g., 540px, 720px, 960px, or 1140px depending on device size).

</p>

</div>

<!-- Full width container -->

<div class="container-fluid mt-4 p-4 bg-info text-white rounded">

<h2>Container Fluid Example (Full Width)</h2>

<p>

This section uses the <code>.container-fluid</code> class.

It always stretches to occupy 100% width of the viewport

regardless of screen size.

</p>

</div>

</body>

</html>

**Output (Expected):**

✅ The webpage displays two sections:

* The **first** section (light grey background) has a fixed width and centered layout.
* The **second** section (blue background) spans the **entire screen width**.

When you resize the browser window, you’ll notice that the first box shrinks or expands only up to a limit, while the second always covers the full width.

**Result:**

Thus, a responsive layout was designed using Bootstrap 5 demonstrating the difference between .container and .container-fluid classes successfully.

**Question:**

Prepare a responsive To-Do List application for a project management tool.  
The application should allow users to add, edit, delete, and mark tasks as complete.  
Each task should have a title, description, and status (complete or incomplete).  
The application should maintain the state using React Hooks and store tasks in browser local storage.

**Title:**

Responsive To-Do List Application using React Hooks and Local Storage

**Aim:**

To design and develop a responsive To-Do List web application using ReactJS, Bootstrap 5, and React Hooks for state management, with data persistence using the browser’s local storage.

**Procedure:**

1. **Step 1: Create a React App**  
   Open the terminal and type:

 npx create-react-app todo-app

cd todo-app

 **Step 2: Install Bootstrap**  
Use CDN or install via npm:

1. npm install bootstrap
2. **Step 3: Open App.js**  
   Replace the default code with the To-Do List code given below.
3. **Step 4: Define States**  
   Use useState to define task title, description, and task list.
4. **Step 5: Handle Functions**  
   Create functions to add, edit, delete, and toggle task completion.
5. **Step 6: Use Local Storage**  
   Use useEffect to automatically save and load tasks from browser local storage.
6. **Step 7: Design UI**  
   Use Bootstrap 5 components for buttons, input fields, and list items.
7. **Step 8: Run the App**  
   Run using npm start and view the app in your browser.

**Program Code (App.js):**

import React, { useState, useEffect } from "react";

import "bootstrap/dist/css/bootstrap.min.css";

function App() {

const [tasks, setTasks] = useState(() => {

const saved = localStorage.getItem("tasks");

return saved ? JSON.parse(saved) : [];

});

const [newTask, setNewTask] = useState("");

const [desc, setDesc] = useState("");

const [editIndex, setEditIndex] = useState(null);

useEffect(() => {

localStorage.setItem("tasks", JSON.stringify(tasks));

}, [tasks]);

const addTask = () => {

if (!newTask.trim()) return;

const task = { title: newTask, description: desc, complete: false };

if (editIndex !== null) {

const updated = [...tasks];

updated[editIndex] = task;

setTasks(updated);

setEditIndex(null);

} else {

setTasks([...tasks, task]);

}

setNewTask("");

setDesc("");

};

const deleteTask = (index) => {

const filtered = tasks.filter((\_, i) => i !== index);

setTasks(filtered);

};

const toggleComplete = (index) => {

const updated = [...tasks];

updated[index].complete = !updated[index].complete;

setTasks(updated);

};

const editTask = (index) => {

const t = tasks[index];

setNewTask(t.title);

setDesc(t.description);

setEditIndex(index);

};

return (

<div className="container mt-5">

<h2 className="text-center mb-4 text-primary">📝 To-Do List (Project Management)</h2>

<div className="card p-3 shadow-sm mb-3">

<input

className="form-control mb-2"

type="text"

placeholder="Task Title"

value={newTask}

onChange={(e) => setNewTask(e.target.value)}

/>

<textarea

className="form-control mb-2"

placeholder="Task Description"

value={desc}

onChange={(e) => setDesc(e.target.value)}

></textarea>

<button className="btn btn-primary" onClick={addTask}>

{editIndex !== null ? "Update Task" : "Add Task"}

</button>

</div>

<ul className="list-group">

{tasks.map((task, index) => (

<li

key={index}

className={`list-group-item d-flex justify-content-between align-items-center ${

task.complete ? "list-group-item-success" : ""

}`}

>

<div>

<h5 className={task.complete ? "text-decoration-line-through" : ""}>

{task.title}

</h5>

<p className="mb-1">{task.description}</p>

</div>

<div>

<button

className="btn btn-success btn-sm me-2"

onClick={() => toggleComplete(index)}

>

{task.complete ? "Undo" : "Complete"}

</button>

<button

className="btn btn-warning btn-sm me-2"

onClick={() => editTask(index)}

>

Edit

</button>

<button

className="btn btn-danger btn-sm"

onClick={() => deleteTask(index)}

>

Delete

</button>

</div>

</li>

))}

</ul>

</div>

);

}

export default App;

**Output (Expected):**

✅ The application displays a list where:

* Users can **add** new tasks with title and description.
* Tasks can be **edited** or **deleted** easily.
* Clicking **Complete** changes task color and marks it done.
* Data remains saved even after page refresh (due to local storage).

**Result:**

* Thus, a responsive To-Do List web application was successfully developed using ReactJS, Bootstrap 5, and React Hooks.  
  The application allows adding, editing, deleting, and marking tasks as complete while maintaining persistent state using browser local storage.

**Question:**

Design a Student Details table using Bootstrap 5 .table, .table-striped, and .table-bordered classes.  
Include columns: **Name**, **Roll No**, **Department**, and **Email**.

**Title:**

Creating a Responsive Student Details Table using Bootstrap Table Classes

**Aim:**

To design a well-formatted, responsive student details table using Bootstrap 5 classes .table, .table-striped, and .table-bordered.

**Procedure:**

1. **Create a new HTML file** named student\_table.html.
2. **Add Bootstrap 5 CDN** link in the <head> section.
3. **Use the <table> tag** with class names table table-striped table-bordered.
4. **Create a table header (<thead>)** for column headings — Name, Roll No, Department, and Email.
5. **Add table rows (<tr>)** inside <tbody> with sample student data.
6. **Use Bootstrap utilities** like container and mt-5 to provide padding and margin.
7. **Save and open** the file in the browser to view the styled table.

**Program Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Student Details Table</title>

<!-- Bootstrap 5 CDN -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<div class="container mt-5">

<h2 class="text-center mb-4">Student Details</h2>

<table class="table table-striped table-bordered">

<thead class="table-primary text-center">

<tr>

<th>Name</th>

<th>Roll No</th>

<th>Department</th>

<th>Email</th>

</tr>

</thead>

<tbody>

<tr>

<td>Priya K</td>

<td>22IT101</td>

<td>Information Technology</td>

<td>priya101@gmail.com</td>

</tr>

<tr>

<td>Arun S</td>

<td>22CS034</td>

<td>Computer Science</td>

<td>aruns@gmail.com</td>

</tr>

<tr>

<td>Meena R</td>

<td>22EC045</td>

<td>Electronics & Comm.</td>

<td>meenar45@gmail.com</td>

</tr>

</tbody>

</table>

</div>

</body>

</html>

**Output (Expected):**

✅ A neat, responsive table appears with striped rows and borders.  
Each column displays student details (Name, Roll No, Department, Email) clearly with a blue header row.

**Result:**

Thus, a responsive student details table was successfully designed using Bootstrap 5 .table, .table-striped, and .table-bordered classes.

**Question:**

Implement a controlled form component in React JS to create a Signup Page.

**Title:**

Implementation of a Controlled Signup Form Component using React JS

**Aim:**

To develop a controlled form component in React JS that accepts user inputs for a signup page and manages data using React state variables.

**Procedure:**

1. **Create a new React app:**

 npx create-react-app signup-form

cd signup-form

 **Install Bootstrap (optional):**

 npm install bootstrap

 **Open App.js** and import Bootstrap if needed.

 **Define state variables** for all form inputs such as name, email, password, and department using useState().

 **Create an onChange handler** for each field to update the corresponding state.

 **Add an onSubmit handler** to display the collected data or validate input.

 **Apply Bootstrap classes** to make the form responsive and attractive.

 **Run the application** using npm start to view the signup form in the browser.

**Program Code (App.js):**

import React, { useState } from "react";

import "bootstrap/dist/css/bootstrap.min.css";

function App() {

const [formData, setFormData] = useState({

name: "",

email: "",

password: "",

department: ""

});

const handleChange = (e) => {

const { name, value } = e.target;

setFormData({ ...formData, [name]: value });

};

const handleSubmit = (e) => {

e.preventDefault();

alert(`Signup Successful!\nName: ${formData.name}\nEmail: ${formData.email}\nDepartment: ${formData.department}`);

};

return (

<div className="container mt-5">

<h2 className="text-center text-primary mb-4">Signup Page</h2>

<form className="card p-4 shadow-sm" onSubmit={handleSubmit}>

<div className="mb-3">

<label className="form-label">Full Name</label>

<input

type="text"

name="name"

className="form-control"

placeholder="Enter your name"

value={formData.name}

onChange={handleChange}

required

/>

</div>

<div className="mb-3">

<label className="form-label">Email</label>

<input

type="email"

name="email"

className="form-control"

placeholder="Enter your email"

value={formData.email}

onChange={handleChange}

required

/>

</div>

<div className="mb-3">

<label className="form-label">Password</label>

<input

type="password"

name="password"

className="form-control"

placeholder="Enter password"

value={formData.password}

onChange={handleChange}

required

/>

</div>

<div className="mb-3">

<label className="form-label">Department</label>

<select

name="department"

className="form-select"

value={formData.department}

onChange={handleChange}

required

>

<option value="">Select Department</option>

<option value="CSE">Computer Science Engineering</option>

<option value="IT">Information Technology</option>

<option value="ECE">Electronics & Communication</option>

</select>

</div>

<button type="submit" className="btn btn-success w-100">

Signup

</button>

</form>

</div>

);

}

export default App;

**Output (Expected):**

✅ A responsive signup form is displayed with the following fields:

* Full Name
* Email
* Password
* Department (dropdown list)

When the user enters details and clicks **Signup**, an alert message confirms successful signup and shows the entered data.

**Result:**

Thus, a controlled form component for a signup page was successfully implemented in React JS using state management through useState() hook, ensuring real-time synchronization of user inputs with component state.

**Question:**

Write a program to display different types of buttons using Bootstrap classes such as .btn-primary, .btn-success, .btn-warning, and .btn-outline-danger.

**Title:**

Creating Various Types of Buttons using Bootstrap 5 Button Classes

**Aim:**

To design and display different styles of buttons using Bootstrap 5 button classes and understand their visual differences.

**Procedure:**

1. **Step 1:** Create a new HTML file and name it bootstrap\_buttons.html.
2. **Step 2:** Include the Bootstrap 5 CSS file via CDN in the <head> section.
3. **Step 3:** Inside the <body>, create a <div> with class container to hold the buttons.
4. **Step 4:** Add buttons using <button> tags and apply different Bootstrap classes such as .btn-primary, .btn-success, .btn-warning, and .btn-outline-danger.
5. **Step 5:** Use margin classes like m-2 to add spacing between buttons.
6. **Step 6:** Save and open the file in a browser to see the styled buttons.

**Program Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Bootstrap Buttons Example</title>

<!-- Bootstrap 5 CDN -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<div class="container mt-5 text-center">

<h2 class="mb-4">Different Types of Bootstrap Buttons</h2>

<button class="btn btn-primary m-2">Primary Button</button>

<button class="btn btn-success m-2">Success Button</button>

<button class="btn btn-warning m-2">Warning Button</button>

<button class="btn btn-outline-danger m-2">Outline Danger Button</button>

<button class="btn btn-dark m-2">Dark Button</button>

</div>

</body>

</html>

**Output (Expected):**

✅ The web page displays multiple buttons in a single row with different colors and outlines:

* Blue for **Primary**
* Green for **Success**
* Yellow for **Warning**
* Red border for **Outline Danger**
* Black for **Dark**

**Result:**

* Thus, a webpage was successfully created using Bootstrap 5 button classes to display various types of buttons with different colors and styles.

**Question:**

Develop a user authentication and registration form for any application using React.js function components.

**Title:**

User Authentication and Registration Form using React.js Functional Components

**Aim:**

To create a responsive registration and login (authentication) form using ReactJS functional components and state management with React Hooks.

**Procedure:**

1. **Step 1:** Create a new React app using:

 npx create-react-app user-auth

cd user-auth

 **Step 2:** Install Bootstrap for UI styling:

 npm install bootstrap

 **Step 3:** Open App.js file and import Bootstrap styles.

 **Step 4:** Create two forms — one for Registration and one for Login.

 **Step 5:** Use useState() to store form field values (like name, email, password).

 **Step 6:** On clicking "Register", store user data in local storage.

 **Step 7:** On clicking "Login", compare input credentials with saved data.

 **Step 8:** Display success or error messages using alert boxes.

 **Step 9:** Test the application in the browser using npm start.

**Program Code (App.js):**

import React, { useState } from "react";

import "bootstrap/dist/css/bootstrap.min.css";

function App() {

const [isLogin, setIsLogin] = useState(true);

const [userData, setUserData] = useState({ name: "", email: "", password: "" });

const [loginData, setLoginData] = useState({ email: "", password: "" });

const handleRegister = (e) => {

e.preventDefault();

localStorage.setItem("user", JSON.stringify(userData));

alert("Registration Successful!");

setIsLogin(true);

};

const handleLogin = (e) => {

e.preventDefault();

const savedUser = JSON.parse(localStorage.getItem("user"));

if (savedUser && savedUser.email === loginData.email && savedUser.password === loginData.password) {

alert(`Welcome ${savedUser.name}! Login Successful`);

} else {

alert("Invalid credentials! Please register first.");

}

};

return (

<div className="container mt-5">

<h2 className="text-center text-primary mb-4">

{isLogin ? "User Login" : "User Registration"}

</h2>

{isLogin ? (

<form className="card p-4 shadow-sm" onSubmit={handleLogin}>

<div className="mb-3">

<label className="form-label">Email</label>

<input

type="email"

className="form-control"

value={loginData.email}

onChange={(e) => setLoginData({ ...loginData, email: e.target.value })}

required

/>

</div>

<div className="mb-3">

<label className="form-label">Password</label>

<input

type="password"

className="form-control"

value={loginData.password}

onChange={(e) => setLoginData({ ...loginData, password: e.target.value })}

required

/>

</div>

<button className="btn btn-success w-100" type="submit">Login</button>

<p className="text-center mt-3">

Don’t have an account?{" "}

<button type="button" className="btn btn-link p-0" onClick={() => setIsLogin(false)}>

Register here

</button>

</p>

</form>

) : (

<form className="card p-4 shadow-sm" onSubmit={handleRegister}>

<div className="mb-3">

<label className="form-label">Full Name</label>

<input

type="text"

className="form-control"

value={userData.name}

onChange={(e) => setUserData({ ...userData, name: e.target.value })}

required

/>

</div>

<div className="mb-3">

<label className="form-label">Email</label>

<input

type="email"

className="form-control"

value={userData.email}

onChange={(e) => setUserData({ ...userData, email: e.target.value })}

required

/>

</div>

<div className="mb-3">

<label className="form-label">Password</label>

<input

type="password"

className="form-control"

value={userData.password}

onChange={(e) => setUserData({ ...userData, password: e.target.value })}

required

/>

</div>

<button className="btn btn-primary w-100" type="submit">Register</button>

<p className="text-center mt-3">

Already have an account?{" "}

<button type="button" className="btn btn-link p-0" onClick={() => setIsLogin(true)}>

Login here

</button>

</p>

</form>

)}

</div>

);

}

export default App;

**Output (Expected):**

✅ The app displays:

* A **registration form** with fields for name, email, and password.
* After successful registration, data is stored in local storage.
* A **login form** where the user enters email and password to authenticate.
* On valid credentials, an alert displays a successful login message.

**Result:**

* Thus, a user authentication and registration form was successfully developed using ReactJS functional components with proper state management using useState() and local storage for data persistence.

**Question:**

Write Bootstrap code to create a dismissible alert with the message “Form submitted successfully!” using .alert-success and .alert-dismissible.

**Title:**

Creating a Dismissible Success Alert using Bootstrap 5

**Aim:**

To design and display a dismissible Bootstrap alert message that notifies the user about successful form submission.

**Procedure:**

1. **Step 1:** Create a new HTML file named alert\_demo.html.
2. **Step 2:** Include Bootstrap 5 CSS and JS CDN links in the <head> and before the closing </body> tag.
3. **Step 3:** Create a container using .container class.
4. **Step 4:** Inside the container, add a <div> with classes .alert, .alert-success, .alert-dismissible, .fade, and .show.
5. **Step 5:** Add a close button using <button> with class .btn-close and attribute data-bs-dismiss="alert".
6. **Step 6:** Save the file and open it in a browser.
7. **Step 7:** Click the close (×) button to dismiss the alert message.

**Program Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Dismissible Alert Example</title>

<!-- Bootstrap CSS -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<div class="container mt-5">

<h3 class="mb-4 text-center">Bootstrap Dismissible Alert Example</h3>

<div class="alert alert-success alert-dismissible fade show" role="alert">

<strong>Success!</strong> Form submitted successfully!

<button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>

</div>

</div>

<!-- Bootstrap JS (for dismiss feature) -->

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>

</body>

</html>

**Output (Expected):**

✅ A green-colored success alert box appears with the message:  
**“Success! Form submitted successfully!”**  
When the user clicks the close (×) button, the alert smoothly disappears.

**Result:**

Thus, a dismissible success alert message was successfully created using Bootstrap 5 classes .alert-success and .alert-dismissible.

**Question:**

Design a webpage to maintain the employees’ payroll for any organization using CRUD operations in MongoDB.

**Title:**

Employee Payroll Management System using CRUD Operations in MongoDB

**Aim:**

To develop a web-based application that allows an organization to perform CRUD (Create, Read, Update, Delete) operations on employee payroll records using MongoDB as the backend database.

**Procedure:**

1. **Step 1:** Create a new project folder named employee-payroll.
2. **Step 2:** Open the terminal and initialize Node.js project:

 npm init -y

 **Step 3:** Install necessary packages:

 npm install express mongoose body-parser cors

 **Step 4:** Create a file named server.js.

 **Step 5:** Connect MongoDB using Mongoose.

 **Step 6:** Define a Mongoose schema for employee payroll details.

 **Step 7:** Create API routes for performing CRUD operations.

 **Step 8:** Run the server using node server.js.

 **Step 9:** Test the API using Postman or a frontend (like React or HTML).

**Program Code (server.js):**

const express = require("express");

const mongoose = require("mongoose");

const bodyParser = require("body-parser");

const cors = require("cors");

const app = express();

app.use(cors());

app.use(bodyParser.json());

// Step 1: Connect MongoDB

mongoose.connect("mongodb://localhost:27017/payrollDB")

.then(() => console.log("MongoDB Connected Successfully"))

.catch(err => console.log(err));

// Step 2: Define Schema

const employeeSchema = new mongoose.Schema({

empId: String,

name: String,

department: String,

salary: Number

});

const Employee = mongoose.model("Employee", employeeSchema);

// Step 3: CREATE - Add new employee

app.post("/add", async (req, res) => {

const emp = new Employee(req.body);

await emp.save();

res.send("Employee Added Successfully!");

});

// Step 4: READ - Get all employees

app.get("/employees", async (req, res) => {

const employees = await Employee.find();

res.json(employees);

});

// Step 5: UPDATE - Modify employee salary

app.put("/update/:id", async (req, res) => {

await Employee.findByIdAndUpdate(req.params.id, req.body);

res.send("Employee Updated Successfully!");

});

// Step 6: DELETE - Remove employee

app.delete("/delete/:id", async (req, res) => {

await Employee.findByIdAndDelete(req.params.id);

res.send("Employee Deleted Successfully!");

});

// Step 7: Start Server

app.listen(5000, () => {

console.log("Server running on http://localhost:5000");

});

**Output (Expected):**

✅ Using Postman or a web frontend, the following results occur:

* **POST /add** → Adds new employee details to payroll database.
* **GET /employees** → Displays all employee records.
* **PUT /update/:id** → Updates salary or department.
* **DELETE /delete/:id** → Deletes employee from payroll.

Example Record:

{

"empId": "E101",

"name": "John Mathew",

"department": "Finance",

"salary": 45000

}

**Result:**

Thus, an employee payroll management webpage was successfully designed and implemented using MongoDB and Node.js, demonstrating all CRUD operations effectively.

**Question:**

Design a responsive table to show the below information using Bootstrap 5.

| **Name** | **Roll No** | **Department** | **Email** |
| --- | --- | --- | --- |
| Priya | 101 | CSE | priya@gmail.com |

|  |  |  |  |
| --- | --- | --- | --- |
| Ramesh | 102 | ECE | ramesh@gmail.com |
| Kavya | 103 | IT | kavya@gmail.com |
| Karthik | 104 | MECH | karthik@gmail.com |

**Title:**

Creating a Responsive Table using Bootstrap 5

**Aim:**

To design a responsive and well-structured table using Bootstrap 5 table classes for displaying student information.

**Procedure:**

1. **Step 1:** Create a new file named responsive\_table.html.
2. **Step 2:** Include Bootstrap 5 CDN in the <head> section.
3. **Step 3:** Create a <div> container with class .table-responsive.
4. **Step 4:** Create a table using <table> tag and apply .table, .table-striped, and .table-bordered classes.
5. **Step 5:** Add header row (<thead>) and data rows (<tbody>).
6. **Step 6:** Save and open the file in a web browser to verify responsiveness.

**Program Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Responsive Table Example</title>

<!-- Bootstrap 5 CDN -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<div class="container mt-5">

<h3 class="text-center mb-4 text-primary">Responsive Student Details Table</h3>

<div class="table-responsive">

<table class="table table-striped table-bordered table-hover">

<thead class="table-dark text-center">

<tr>

<th>Name</th>

<th>Roll No</th>

<th>Department</th>

<th>Email</th>

</tr>

</thead>

<tbody>

<tr>

<td>Priya</td>

<td>101</td>

<td>CSE</td>

<td>priya@gmail.com</td>

</tr>

<tr>

<td>Ramesh</td>

<td>102</td>

<td>ECE</td>

<td>ramesh@gmail.com</td>

</tr>

<tr>

<td>Kavya</td>

<td>103</td>

<td>IT</td>

<td>kavya@gmail.com</td>

</tr>

<tr>

<td>Karthik</td>

<td>104</td>

<td>MECH</td>

<td>karthik@gmail.com</td>

</tr>

</tbody>

</table>

</div>

</div>

</body>

</html>

**Output (Expected):**

✅ A clean, bordered, and striped responsive table is displayed with all student details.  
When viewed on smaller screens, the table becomes horizontally scrollable.

**Result:**

Thus, a responsive table was successfully designed using Bootstrap 5 classes .table, .table-striped, .table-bordered, and .table-responsive.

**Question:**

Design a calculator using the **module concept** in Node.js.

**Title:**

Creating a Calculator Application using Node.js Module Concept

**Aim:**

To design and implement a simple calculator application that performs basic arithmetic operations using user-defined modules in Node.js.

**Procedure:**

1. **Step 1:** Create a new folder named node\_calculator.
2. **Step 2:** Inside the folder, create two files:
   * calculator.js → module file (contains arithmetic functions)
   * app.js → main file (imports and uses module functions)
3. **Step 3:** Open terminal and initialize the project using:

 npm init -y

 **Step 4:** Write function definitions in calculator.js and export them.

 **Step 5:** In app.js, import the module using require('./calculator').

 **Step 6:** Call each arithmetic function with sample values.

 **Step 7:** Run the program using node app.js in terminal.

 **Step 8:** Observe the output for all operations.

**Program Code**

**File 1: calculator.js**

// Calculator Module

function add(a, b) {

return a + b;

}

function subtract(a, b) {

return a - b;

}

function multiply(a, b) {

return a \* b;

}

function divide(a, b) {

if (b === 0) return "Division by zero not allowed";

return a / b;

}

// Export all functions

module.exports = { add, subtract, multiply, divide };

**File 2: app.js**

// Importing the calculator module

const calc = require('./calculator');

console.log("Simple Calculator using Node.js Modules\n");

let num1 = 10, num2 = 5;

console.log(`Addition of ${num1} and ${num2} = ${calc.add(num1, num2)}`);

console.log(`Subtraction of ${num1} and ${num2} = ${calc.subtract(num1, num2)}`);

console.log(`Multiplication of ${num1} and ${num2} = ${calc.multiply(num1, num2)}`);

console.log(`Division of ${num1} and ${num2} = ${calc.divide(num1, num2)}`);

**Execution Command:**

node app.js

**Output (Expected):**

Simple Calculator using Node.js Modules

Addition of 10 and 5 = 15

Subtraction of 10 and 5 = 5

Multiplication of 10 and 5 = 50

Division of 10 and 5 = 2

**Result:**

Thus, a simple calculator was successfully implemented using the **module concept** in Node.js by exporting arithmetic functions from one module and importing them into another program for execution.

**Question:**

Design a responsive web page for a school.  
The page should include a navigation bar at the top for easy access to different sections such as **Home**, **About**, **Academics**, **Admissions**, and **Contact**.  
Use **Bootstrap’s navigation bar** to implement this functionality.

**Title:**

Designing a Responsive School Website using Bootstrap 5 Navigation Bar

**Aim:**

To design a responsive school web page with a navigation bar using Bootstrap 5 for easy navigation across different sections like Home, About, Academics, Admissions, and Contact.

**Procedure:**

1. **Step 1:** Create a new HTML file named school\_navbar.html.
2. **Step 2:** Include Bootstrap 5 CSS and JS CDN in <head> and before </body>.
3. **Step 3:** Add a navigation bar using <nav> tag with class .navbar and .navbar-expand-lg.
4. **Step 4:** Add menu items — Home, About, Academics, Admissions, and Contact.
5. **Step 5:** Add a brand name like “Sunshine Public School”.
6. **Step 6:** Add dummy sections for each menu link.
7. **Step 7:** Save and open the file in a browser to verify responsiveness.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>School Website - Bootstrap Navbar</title>

<!-- Bootstrap 5 CDN -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<!-- Navigation Bar -->

<nav class="navbar navbar-expand-lg navbar-dark bg-primary">

<div class="container-fluid">

<a class="navbar-brand" href="#">Sunshine Public School</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav ms-auto">

<li class="nav-item"><a class="nav-link active" href="#">Home</a></li>

<li class="nav-item"><a class="nav-link" href="#">About</a></li>

<li class="nav-item"><a class="nav-link" href="#">Academics</a></li>

<li class="nav-item"><a class="nav-link" href="#">Admissions</a></li>

<li class="nav-item"><a class="nav-link" href="#">Contact</a></li>

</ul>

</div>

</div>

</nav>

<!-- Page Content -->

<div class="container mt-5">

<h2 class="text-center text-primary">Welcome to Sunshine Public School</h2>

<p class="text-center">Empowering students through quality education and holistic development.</p>

<div class="row mt-5">

<div class="col-md-6">

<h4>About Our School</h4>

<p>Sunshine Public School provides world-class education to nurture creativity, leadership, and integrity among students.</p>

</div>

<div class="col-md-6">

<img src="https://via.placeholder.com/500x300" class="img-fluid rounded" alt="School Image">

</div>

</div>

</div>

<!-- Bootstrap JS -->

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>

</body>

</html>

**Output (Expected):**

✅ A responsive school website is displayed with a **blue navigation bar** containing menu links:  
Home | About | Academics | Admissions | Contact  
When viewed on smaller screens, the menu collapses into a **hamburger icon**.

**Result:**

Thus, a responsive school web page with a navigation bar was successfully designed using Bootstrap 5.

**Question:**

Design a responsive registration page to get user details using **GET** and **POST** methods in **Node.js**.

**Title:**

Creating a Registration Page using Node.js with GET and POST Methods

**Aim:**

To design a responsive registration form using HTML and Bootstrap, and handle form data in Node.js using GET and POST HTTP method

**Procedure:**

1. **Step 1:** Create a folder named user\_registration.
2. **Step 2:** Initialize Node.js project:

 npm init -y

 **Step 3:** Install Express and Body-parser:

 npm install express body-parser

 **Step 4:** Create a file named server.js.

 **Step 5:** Create a registration form using Bootstrap.

 **Step 6:** Use **GET** method to load the form and **POST** to handle submitted data.

 **Step 7:** Run the program using node server.js.

 **Step 8:** Test in browser at http://localhost:3000/register.

**Program Code:**

**File: server.js**

const express = require("express");

const bodyParser = require("body-parser");

const app = express();

// Middleware

app.use(bodyParser.urlencoded({ extended: true }));

// GET - Display registration form

app.get("/register", (req, res) => {

res.send(`

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>User Registration</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<div class="container mt-5">

<h3 class="text-center text-primary mb-4">User Registration Form</h3>

<form action="/register" method="POST" class="card p-4 shadow-sm">

<div class="mb-3">

<label class="form-label">Full Name</label>

<input type="text" name="name" class="form-control" required>

</div>

<div class="mb-3">

<label class="form-label">Email</label>

<input type="email" name="email" class="form-control" required>

</div>

<div class="mb-3">

<label class="form-label">Password</label>

<input type="password" name="password" class="form-control" required>

</div>

<div class="mb-3">

<label class="form-label">Department</label>

<select name="department" class="form-select">

<option value="CSE">CSE</option>

<option value="ECE">ECE</option>

<option value="IT">IT</option>

<option value="MECH">MECH</option>

</select>

</div>

<button type="submit" class="btn btn-success w-100">Register</button>

</form>

</div>

</body>

</html>

`);

});

// POST - Handle form data

app.post("/register", (req, res) => {

const { name, email, password, department } = req.body;

res.send(`

<h2>Registration Successful!</h2>

<p><strong>Name:</strong> ${name}</p>

<p><strong>Email:</strong> ${email}</p>

<p><strong>Department:</strong> ${department}</p>

`);

});

// Start Server

app.listen(3000, () => console.log("Server running at [http://localhost:3000/register")](http://localhost:3000/register%22)));

**Execution Command:**

node server.js

**Output (Expected):**

✅ A responsive **registration form** appears with fields for **Name, Email, Password, Department**.  
On clicking **Register**, the entered details are displayed on a new page showing successful registration.

**Result:**

Thus, a responsive registration page was successfully created and executed using **Node.js** to handle form data via **GET and POST methods**.

**Question:**

Create a web server to perform HTTP request and response with the following URLs:

* [**http://localhost:3000/courses**](http://localhost:3000/courses)

[**http://localhost:3000/departments**](http://localhost:3000/departments)

for designing any college website.

**Title:**

Designing a Simple College Website Web Server using Node.js HTTP Module

**Aim:**

To create a simple web server in Node.js that handles different HTTP requests and responses for the given URLs /courses and /departments for a college website.

**Procedure:**

1. **Step 1:** Create a new folder named college\_server.
2. **Step 2:** Open the folder in VS Code and create a file named server.js.
3. **Step 3:** Import the http module.
4. **Step 4:** Create an HTTP server using http.createServer().
5. **Step 5:** Check the request URL and send appropriate responses.
6. **Step 6:** Listen on port 3000.
7. **Step 7:** Run the server using the command node server.js.
8. **Step 8:** Open the browser and visit the given URLs to see the response.

**Program Code:**

// server.js

const http = require("http");

const server = http.createServer((req, res) => {

res.writeHead(200, { "Content-Type": "text/html" });

if (req.url === "/courses") {

res.write("<h2>College Courses</h2>");

res.write("<ul><li>B.E. Computer Science</li><li>B.E. ECE</li><li>B.E. Mechanical</li><li>B.Tech IT</li></ul>");

}

else if (req.url === "/departments") {

res.write("<h2>Departments</h2>");

res.write("<ul><li>Computer Science Department</li><li>Electronics Department</li><li>Mechanical Department</li><li>Information Technology Department</li></ul>");

}

else {

res.write("<h2>Welcome to ABC College Website</h2>");

res.write("<p>Use /courses or /departments in the URL to view details.</p>");

}

res.end();

});

server.listen(3000, () => {

console.log("Server running at http://localhost:3000/");

});

**Execution Steps:**

1. Open terminal → type

 node server.js

 Open your browser and visit:

* http://localhost:3000/
* http://localhost:3000/courses
* <http://localhost:3000/departments>

**Output (Expected):**

* ✅ On opening:
* http://localhost:3000/ → Displays **Welcome to ABC College Website** message.
* http://localhost:3000/courses → Lists college courses.
* http://localhost:3000/departments → Lists college departments.

**Result:**

* Thus, a simple web server was successfully created in Node.js to handle multiple HTTP requests and responses for a college website.

**Question:**

* Write Bootstrap 5 code that allows you to group a series of buttons together (on a single line) in a button group as shown below.

**Title:**

Creating a Button Group using Bootstrap 5

**Aim:**

To design a responsive button group using Bootstrap 5 that arranges multiple buttons together on a single line.

**Procedure:**

1. **Step 1:** Create a new HTML file named button\_group.html.
2. **Step 2:** Include Bootstrap 5 CSS CDN in the <head>.
3. **Step 3:** Create a <div> with the class .btn-group.
4. **Step 4:** Add multiple buttons inside using .btn classes.
5. **Step 5:** Save and open the file in your browser.

**Program Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Bootstrap 5 Button Group</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body class="text-center mt-5">

<h3 class="text-primary mb-4">Bootstrap Button Group Example</h3>

<div class="btn-group" role="group" aria-label="Basic example">

<button type="button" class="btn btn-primary">Home</button>

<button type="button" class="btn btn-success">About</button>

<button type="button" class="btn btn-warning">Services</button>

<button type="button" class="btn btn-danger">Contact</button>

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>

</body>

</html>

**Output (Expected):**

✅ Four buttons — **Home**, **About**, **Services**, and **Contact** — are displayed side by side in a single line as a grouped button bar.

**Result:**

Thus, a responsive button group was successfully designed using Bootstrap 5 .btn-group class.

**Question:**

Build a navbar that features links to **Home**, **About**, **Projects**, and **Contact**. Include a **dropdown** for Projects that expands to show individual project categories. The navbar should collapse on smaller screens using Bootstrap 5.

**Title:**

Designing a Responsive Navbar with Dropdown using Bootstrap 5

**Aim:**

To create a responsive navigation bar with dropdown functionality using Bootstrap 5, which automatically collapses on smaller screens.

**Procedure:**

1. Create a new HTML file named navbar\_dropdown.html.
2. Include Bootstrap 5 CSS and JS CDN links.
3. Use <nav> tag with .navbar, .navbar-expand-lg, and .navbar-dark or .navbar-light.
4. Add links for Home, About, Projects (with dropdown), and Contact.
5. Use .navbar-toggler for responsive behavior.
6. Save and open in a web browser to test responsiveness.

**Program Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Responsive Navbar with Dropdown</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<div class="container-fluid">

<a class="navbar-brand" href="#">MyPortfolio</a>

<!-- Toggler button -->

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNavDropdown">

<span class="navbar-toggler-icon"></span>

</button>

<!-- Navbar links -->

<div class="collapse navbar-collapse" id="navbarNavDropdown">

<ul class="navbar-nav ms-auto">

<li class="nav-item"><a class="nav-link active" href="#">Home</a></li>

<li class="nav-item"><a class="nav-link" href="#">About</a></li>

<!-- Dropdown Menu -->

<li class="nav-item dropdown">

<a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Projects</a>

<ul class="dropdown-menu">

<li><a class="dropdown-item" href="#">Web Development</a></li>

<li><a class="dropdown-item" href="#">Mobile Apps</a></li>

<li><a class="dropdown-item" href="#">Machine Learning</a></li>

</ul>

</li>

<li class="nav-item"><a class="nav-link" href="#">Contact</a></li>

</ul>

</div>

</div>

</nav>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>

</body>

</html>

**Output (Expected):**

✅ A responsive navigation bar appears with menu links:  
**Home | About | Projects (dropdown) | Contact**  
On small screens, the navbar collapses into a hamburger menu.

**Result:**

Thus, a responsive navigation bar with dropdown functionality was successfully designed using Bootstrap 5.

**Question:**

Create and manage a **MongoDB database** for student details.  
Perform the following operations:  
(i) Retrieve documents with **GPA greater than 8**  
(ii) Display the **maximum GPA** of students in the collection.

**Title:**

Managing Student Database and Performing Queries in MongoDB

**Aim:**

To create and manage a MongoDB database for student details and to perform queries using comparison and aggregation operations.

**Procedure:**

1. Open **MongoDB shell** using the command:
2. mongosh
3. Create or switch to a database named collegeDB.
4. Create a collection named students.
5. Insert multiple student documents with fields: name, regno, and gpa.
6. Retrieve all students with **GPA greater than 8** using $gt.
7. Find the **maximum GPA** using aggregation or sorting.

**MongoDB Commands and Code:**

// Step 1: Create / use database

use collegeDB

// Step 2: Create and insert documents

db.students.insertMany([

{ name: "Arun", regno: "CSE101", gpa: 7.8 },

{ name: "Priya", regno: "CSE102", gpa: 8.5 },

{ name: "Ramesh", regno: "CSE103", gpa: 9.2 },

{ name: "Kavya", regno: "CSE104", gpa: 8.9 },

{ name: "Karthik", regno: "CSE105", gpa: 7.4 }

]);

// Step 3: Retrieve students with GPA greater than 8

db.students.find({ gpa: { $gt: 8 } });

// Step 4: Display maximum GPA of students

db.students.aggregate([

{ $group: { \_id: null, maxGPA: { $max: "$gpa" } } }

]);

**Output (Expected):**

**For GPA > 8:**

{ "\_id": ObjectId("..."), "name": "Priya", "regno": "CSE102", "gpa": 8.5 }

{ "\_id": ObjectId("..."), "name": "Ramesh", "regno": "CSE103", "gpa": 9.2 }

{ "\_id": ObjectId("..."), "name": "Kavya", "regno": "CSE104", "gpa": 8.9 }

**For Maximum GPA:**

{ "\_id": null, "maxGPA": 9.2 }

**Result:**

Thus, the MongoDB student database was successfully created and the required queries were executed to:  
1️⃣ Retrieve all students with GPA > 8  
2️⃣ Display the maximum GPA among all students.

**Question:**

Develop a MongoDB database for a library's book inventory system. The database should handle book details, such as **title, author, genre, publication year, availability status**, and **number of copies**. Perform the following operations:

* 1. Create a "books" collection and insert at least five documents.  
     (ii) Retrieve all books where the **genre** is "Mystery" or "Thriller".
  2. Delete books published before the year 2000.

**Title:**

Library Book Inventory Management System using MongoDB

**Aim:**

To design and manage a MongoDB database for a library to store and manipulate book inventory data and perform various CRUD operations.

**Procedure:**

1. Open **MongoDB shell** using mongosh.
2. Create a new database named libraryDB.
3. Create a collection named books.
4. Insert at least five book documents with details such as title, author, genre, year, availability, and copies.
5. Retrieve books with genre “Mystery” or “Thriller” using $or.
6. Delete books published before 2000 using $lt (less than) operator.
7. Verify the result after each operation.

**Program / Commands:**

// Step 1: Create or switch to the database

use libraryDB

// Step 2: Create 'books' collection and insert multiple documents

db.books.insertMany([

{ title: "The Silent Patient", author: "Alex Michaelides", genre: "Thriller", year: 2019, available: true, copies: 5 },

{ title: "The Da Vinci Code", author: "Dan Brown", genre: "Mystery", year: 2003, available: true, copies: 8 },

{ title: "Pride and Prejudice", author: "Jane Austen", genre: "Romance", year: 1813, available: false, copies: 2 },

{ title: "Gone Girl", author: "Gillian Flynn", genre: "Thriller", year: 2012, available: true, copies: 4 },

{ title: "Angels & Demons", author: "Dan Brown", genre: "Mystery", year: 2000, available: true, copies: 6 }

]);

// Step 3: Retrieve all books where genre is 'Mystery' or 'Thriller'

db.books.find({ $or: [ { genre: "Mystery" }, { genre: "Thriller" } ] });

// Step 4: Delete books published before the year 2000

db.books.deleteMany({ year: { $lt: 2000 } });

// Step 5: Display the remaining books

db.books.find();

**Expected Output:**

**📘 After Insertion:**

{ title: "The Silent Patient", author: "Alex Michaelides", genre: "Thriller", year: 2019, available: true, copies: 5 }

{ title: "The Da Vinci Code", author: "Dan Brown", genre: "Mystery", year: 2003, available: true, copies: 8 }

{ title: "Pride and Prejudice", author: "Jane Austen", genre: "Romance", year: 1813, available: false, copies: 2 }

{ title: "Gone Girl", author: "Gillian Flynn", genre: "Thriller", year: 2012, available: true, copies: 4 }

{ title: "Angels & Demons", author: "Dan Brown", genre: "Mystery", year: 2000, availabl

**📗 After Query (Mystery or Thriller):**

{ title: "The Silent Patient", genre: "Thriller" }

{ title: "The Da Vinci Code", genre: "Mystery" }

{ title: "Gone Girl", genre: "Thriller" }

{ title: "Angels & Demons", genre: "Mystery" }

**📙 After Deleting Books Published Before 2000:**

{ title: "The Silent Patient", year: 2019 }

{ title: "The Da Vinci Code", year: 2003 }

{ title: "Gone Girl", year: 2012 }

{ title: "Angels & Demons", year: 2000 }

**Result:**

Thus, the MongoDB database for the library system was successfully created, queried, and modified using CRUD operations.

**Question:**

Design a web server to perform HTTP request and response with the following URLs:

* [**http://localhost:5000/products**](http://localhost:5000/products)

[**http://localhost:3000/contact**](http://localhost:3000/contact)

for designing any online shopping website.

**Title:**

Designing an HTTP Web Server for Online Shopping Website using Node.js

**Aim:**

To design and implement a Node.js web server that handles multiple HTTP requests and sends different responses for product and contact pages.

**Procedure:**

1. Open Visual Studio Code and create a file named shopping\_server.js.
2. Import the HTTP module using require("http").
3. Create the server using http.createServer().
4. Use conditional statements to check req.url.
5. Send appropriate HTML responses for /products and /contact.
6. Make the server listen on ports **5000** and **3000** for each URL.
7. Run the files using the Node.js command.
8. Check responses in the browser.

**Program Code:**

**File 1 – products\_server.js**

const http = require("http");

const productServer = http.createServer((req, res) => {

res.writeHead(200, { "Content-Type": "text/html" });

if (req.url === "/products") {

res.write("<h2>Online Shopping - Product List</h2>");

res.write("<ul><li>Laptop</li><li>Smartphone</li><li>Headphones</li><li>Smart Watch</li></ul>");

} else {

res.write("<h3>Welcome to Our Online Store</h3>");

}

res.end();

});

productServer.listen(5000, () => {

console.log("Product server running at http://localhost:5000/");

});

**File 2 – contact\_server.js**

const http = require("http");

const contactServer = http.createServer((req, res) => {

res.writeHead(200, { "Content-Type": "text/html" });

if (req.url === "/contact") {

res.write("<h2>Contact Us</h2>");

res.write("<p>Email: support@onlineshop.com</p>");

res.write("<p>Phone: +91 98765 43210</p>");

} else {

res.write("<h3>Welcome to Online Shopping Contact Portal</h3>");

}

res.end();

});

contactServer.listen(3000, () => {

console.log("Contact server running at http://localhost:3000/");

});

**Execution Steps:**

1. Open terminal → navigate to project folder.
2. Run both servers using:
3. node products\_server.js
4. node contact\_server.js
5. Open the following in your browser:
   * http://localhost:5000/products → Product list
   * http://localhost:3000/contact → Contact info

**Expected Output:**

**For http://localhost:5000/products**

Online Shopping - Product List

• Laptop

• Smartphone

• Headphones

• Smart Watch

**For http://localhost:3000/contact**

Contact Us

Email: support@onlineshop.com

Phone: +91 98765 43210

**Result:**

Thus, two Node.js servers were successfully created to handle multiple HTTP requests and responses for an online shopping website.

Develop a User Registration Page with Client-Side Validation using Bootstrap 5

**Title:**

Responsive User Registration Form with Real-Time Client-Side Validation using Bootstrap 5

**Aim:**

To design a responsive web registration form using Bootstrap 5 that collects user information (username, email, password, and phone number) and provides instant validation feedback for correct or incorrect input entries

**Procedure:**

1. Create an HTML file named register.html.
2. Link the **Bootstrap 5 CDN** in the <head> section.
3. Create a <form> that includes input fields for username, email, password, and phone number.
4. Add validation attributes like required, type="email", and pattern.
5. Use Bootstrap classes .invalid-feedback and .valid-feedback for feedback messages.
6. Add JavaScript to prevent submission if any field is invalid.
7. Test in a browser by entering different data to see real-time validation.

**Program Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>User Registration Form</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body class="bg-light">

<div class="container mt-5">

<div class="card shadow-lg p-4">

<h3 class="text-center text-primary mb-4">User Registration</h3>

<form class="needs-validation" novalidate>

<div class="mb-3">

<label class="form-label">Username</label>

<input type="text" class="form-control" required>

<div class="valid-feedback">Looks good!</div>

<div class="invalid-feedback">Please enter your username.</div>

</div>

<div class="mb-3">

<label class="form-label">Email</label>

<input type="email" class="form-control" required>

<div class="valid-feedback">Valid email!</div>

<div class="invalid-feedback">Enter a valid email address.</div>

</div>

<div class="mb-3">

<label class="form-label">Password</label>

<input type="password" class="form-control" minlength="6" required>

<div class="valid-feedback">Strong password!</div>

<div class="invalid-feedback">Password must be at least 6 characters.</div>

</div>

<div class="mb-3">

<label class="form-label">Phone Number</label>

<input type="tel" class="form-control" pattern="[0-9]{10}" required>

<div class="valid-feedback">Valid phone number!</div>

<div class="invalid-feedback">Enter a 10-digit phone number.</div>

</div>

<button class="btn btn-success w-100" type="submit">Register</button>

</form>

</div>

</div>

<script>

// Bootstrap validation script

(() => {

'use strict';

const forms = document.querySelectorAll('.needs-validation');

Array.from(forms).forEach(form => {

form.addEventListener('submit', event => {

if (!form.checkValidity()) {

event.preventDefault();

event.stopPropagation();

}

form.classList.add('was-validated');

}, false);

});

})();

</script>

</body>

</html>

**Output (Browser Preview):**

✅ A neat, mobile-friendly registration form with fields:

* Username
* Email
* Password
* Phone Number

When invalid data is entered, the input box turns red with an error message.  
When valid data is entered, it turns green with success feedback.

**Result:**

Thus, a responsive registration form was successfully designed using Bootstrap 5 with real-time client-side validation.

Create a MongoDB Database for Employee Details

**Title:**

Employee Details Management using MongoDB

**Aim:**

To create and manage an employee collection in MongoDB and perform CRUD operations such as insertion, retrieval, and updating employee data.

**Procedure:**

1. Start MongoDB shell using mongosh.
2. Create a new database named employeeDB.
3. Create a collection named employee.
4. Insert 5 documents with fields: name, designation, address, and salary.
5. Retrieve employees having salary greater than 30000 using the $gt operator.
6. Update the address of any two employees using the updateOne() command.
7. Display all employee details after modification.

**Program / Commands:**

// Step 1: Switch to or create a database

use employeeDB

// Step 2: Create collection and insert multiple employee records

db.employee.insertMany([

{ name: "Arun Kumar", designation: "Manager", address: "Chennai", salary: 45000 },

{ name: "Priya Sharma", designation: "Developer", address: "Coimbatore", salary: 32000 },

{ name: "Rahul Das", designation: "Designer", address: "Bangalore", salary: 28000 },

{ name: "Sneha Reddy", designation: "HR", address: "Hyderabad", salary: 35000 },

{ name: "Vignesh", designation: "Tester", address: "Erode", salary: 27000 }

]);

// Step 3: Retrieve documents where salary > 30000

db.employee.find({ salary: { $gt: 30000 } });

// Step 4: Update address of any two employees

db.employee.updateOne({ name: "Priya Sharma" }, { $set: { address: "Madurai" } });

db.employee.updateOne({ name: "Sneha Reddy" }, { $set: { address: "Salem" } });

// Step 5: Display all employee details after update

db.employee.find();

**Expected Output:**

**📘 After Insertion:**

{ name: "Arun Kumar", designation: "Manager", address: "Chennai", salary: 45000 }

{ name: "Priya Sharma", designation: "Developer", address: "Coimbatore", salary: 32000 }

{ name: "Rahul Das", designation: "Designer", address: "Bangalore", salary: 28000 }

{ name: "Sneha Reddy", designation: "HR", address: "Hyderabad", salary: 35000 }

{ name: "Vignesh", designation: "Tester", address: "Erode", salary: 27000 }

**📗 Query (Salary > 30000):**

{ name: "Arun Kumar", salary: 45000 }

{ name: "Priya Sharma", salary: 32000 }

{ name: "Sneha Reddy", salary: 35000 }

**📙 After Address Update:**

{ name: "Priya Sharma", address: "Madurai" }

{ name: "Sneha Reddy", address: "Salem" }

**Result:**

Thus, the employee collection was successfully created, queried, and updated in MongoDB to manage employee details effectively.