

# **Front End Design Techniques Practical File (IT-620)**

*Submitted in partial fulfillment of the requirements for the award of the degree*

*Of*

## **Masters of Computer Application (SE)**

**Submitted To**

**Ms. Avni Mishra**

USIC&T

**Submitted By**

**Khushal Sachdeva**

Enroll. No. 00516404524

Semester - II



**University School of Information , Communication & Technology  
Guru Gobind Singh Indraprastha University  
Sector-16C , Dwarka New Delhi - 110078  
(2024-2026)**

# Practical - 1

## Basics of Internet, HTML, and HTTP

**Objective:** Understand the history of the internet, World Wide Web, and HTTP. Learn basic HTML tags.

**Task 1:** Discuss HTTP, SMTP, and MIME with practical examples using a browser's developer tools.

### 1. HTTP (Hypertext Transfer Protocol)

HTTP (HyperText Transfer Protocol) is the foundation of data communication on the web. It is an application-layer protocol that enables communication between clients (such as web browsers) and servers (where websites are hosted).

#### Key Features of HTTP

1. Stateless – Each request is independent; the server does not retain session information.
2. Request-Response Model – The client sends a request, and the server responds.
3. TCP-Based – It runs over TCP (Transmission Control Protocol), typically on port 80 (or 443 for HTTPS).
4. Methods – HTTP defines different request methods such as:
  - GET – Retrieve data (e.g., webpage content).
  - POST – Submit data to the server (e.g., form submission).
  - PUT – Update existing resources.
  - DELETE – Remove a resource.
  - PATCH – Partially update a resource.
5. Headers – HTTP requests and responses contain metadata (e.g., content type, cache control).

## HTTP Request Structure

1. **Request Line:** The first line of an HTTP request specifies:

- Method (GET, POST, PUT, DELETE, etc.)
- Resource Path (the endpoint being accessed)
- HTTP Version (e.g., HTTP/1.1 or HTTP/2)

Example

```
GET /index.html HTTP/1.1
```

2. **Headers:** Headers provide metadata about the request, such as:

- Host: Specifies the server domain.
- User-Agent: Identifies the client (browser, script, etc.).
- Accept: Specifies the content types the client can handle.
- Content-Type: Defines the data format in a POST/PUT request.
- Authorization: Sends authentication credentials.

Example

```
Host: www.xyz.com
User-Agent: Mozilla/5.0
Accept: text/html
```

3. **Body (Optional):** The body contains data sent to the server (like form inputs or JSON payloads). Used in methods like **POST**, **PUT**, **PATCH**.

Example

```
POST /login HTTP/1.1
Host: www.xyz.com
Content-Type: application/json
Content-Length: 42

{"username": "khushal", "password": "secure123"}
```

## HTTP Response Structure

1. **Status Line:** The first line of an HTTP response specifies:
  - HTTP Version (e.g., HTTP/1.1 or HTTP/2)
  - Status Code (e.g., 200, 404, 500)
  - Status Message (a short description of the status code)

Example

```
HTTP/1.1 200 OK
```

### Common Status Codes:

**1xx (Informational):** Request received, processing continues (e.g., 100 Continue).

**2xx (Success):** Request processed successfully (e.g., 200 OK, 201 Created).

**3xx (Redirection):** Further action needed (e.g., 301 Moved Permanently, 302 Found).

**4xx (Client Errors):** Error caused by the client (e.g., 400 Bad Request, 404 Not Found).

**5xx (Server Errors):** Error caused by the server (e.g., 500 Internal Server Error).

2. **Response Headers:** Headers provide metadata about the response, such as:
  - Content-Type: Specifies the data format (HTML, JSON, etc.).
  - Content-Length: Size of the response body.
  - Server: Information about the web server.
  - Set-Cookie: Used for session management.

Example

```
Content-Type: text/html
Content-Length: 1234
Server: Apache/2.4.41
Set-Cookie: sessionId=abc123; Path=/; HttpOnly
```

3. **Response Body (Optional):** The body contains the actual content being returned (HTML, JSON, XML, etc.). If the request was for a webpage, this would be the HTML content.

Example

```
<html>
  <head><title>Welcome</title></head>
  <body><h1>Hello, World!</h1></body>
</html>
```

## Full Example

### HTTP Request

```
GET /home HTTP/1.1
Host: www.xyz.com
User-Agent: Mozilla/5.0
Accept: text/html
Connection: keep-alive
```

### HTTP Response

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Server: nginx/1.18.0

<html>
  <head><title>Welcome</title></head>
  <body><h1>Hello, World!</h1></body>
</html>
```

## 2. SMTP (Simple Mail Transfer Protocol)

SMTP (Simple Mail Transfer Protocol) is an application-layer protocol used for sending emails over the internet. It is a text-based protocol that follows a client-server model, where email clients send messages to an SMTP server, which then forwards them to the recipient's mail server.

### Ports used by SMTP

- Port 25: Default SMTP port (but often blocked by ISPs for security reasons).
- Port 465: SMTP with SSL (secure but deprecated).
- Port 587: SMTP with STARTTLS (recommended for sending emails securely).
- Port 2525: Alternative SMTP port (used by some email providers).

### SMTP Request Structure

#### 1. SMTP Handshake (Connecting to the Server)

Client initiates the connection:

```
telnet smtp.xyz.com 25
```

Server responds with a greeting:

```
220 smtp.xyz.com ESMTP Service Ready
```

Client introduces itself:

```
HELO mail.client.com
```

Server acknowledges:

```
250 Hello mail.client.com
```

## 2. Sending an Email

Client specifies the sender:

```
MAIL FROM:<sender@xyz.com>
```

Server confirms:

```
250 OK
```

Client specifies the recipient:

```
RCPT TO:<receiver@xyz.com>
```

Server confirms:

```
250 OK
```

Client starts message input:

```
DATA
```

Server prompts for message body:

```
354 Start mail input; end with <CRLF>.<CRLF>
```

Client sends email content:

```
Subject: Test Email  
From: sender@xyz.com  
To: receiver@xyz.com  
Hello, this is a test email.  
.
```

Server acknowledges email sent:

```
250 OK: Queued as 12345
```

## 3. Closing the Connection

Client terminates the session:

```
QUIT
```

Server confirms and closes connection:

```
221 smtp.xyz.com closing connection
```

## How SMTP Works in Real Life

1. Email Client (e.g., Gmail, Outlook) → SMTP Server  
The email is sent from the sender's email client to the SMTP server.
2. SMTP Server → Recipient's Mail Server (via MX Record Lookup)  
The SMTP server looks up the recipient's mail server using DNS (Mail Exchange records).
3. Recipient's Mail Server → Inbox (via POP3/IMAP)  
The recipient retrieves the email using POP3 or IMAP.

## 3. MIME (Multipurpose Internet Mail Extensions)

MIME (Multipurpose Internet Mail Extensions) is an extension of SMTP that allows emails to support multimedia content such as text, images, audio, video, and attachments. It defines how different types of content should be formatted and transmitted.

### Why is MIME Needed?

- SMTP was originally designed to handle only plain text. MIME enhances it adding more formats
- Sending attachments (PDFs, images, videos)
- Formatting text as HTML (rich text emails)
- Supporting multiple parts (multipart emails)

### MIME Request Structure

#### 1. Plain Text Email (Without MIME)

```
Subject: Hello
From: sender@xyz.com
To: receiver@xyz.com
This is a plain text email.
```

#### 2. Multipart MIME Email (HTML + Plain Text)

```
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary="boundary123"

--boundary123
Content-Type: text/plain; charset="UTF-8"

This is a plain text version of the email.
```

```
--boundary123
Content-Type: text/html; charset="UTF-8"

<html>
  <body>
    <h1>This is an HTML email</h1>
    <p>Welcome to our service!</p>
  </body>
</html>

--boundary123--
```

### 3. MIME Email with Attachment

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="boundary456"

--boundary456
Content-Type: text/plain; charset="UTF-8"

This is an email with an attachment.

--boundary456
Content-Type: application/pdf
Content-Disposition: attachment; filename="example.pdf"
Content-Transfer-Encoding: base64

JVBERi0xLjQKJeLjz9MKMyAwIG9iago8...
(base64 encoded file data)
--boundary456--
```

### How MIME Works in Email Transmission

1. User Composes Email
  - The email client (e.g., Gmail, Outlook) formats the email using MIME.
2. Email Sent via SMTP
  - The MIME-encoded email is transmitted over SMTP.
3. Recipient's Email Server Receives the MIME Email
  - The email is stored and processed.
4. Email Client (Gmail, Outlook) Decodes MIME
  - The email is displayed with formatted text and attachments.



**Task 2:** Create a basic HTML page with a heading, paragraph, list, table, and image.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Khushal Sachdeva's Resume</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      line-height: 1.6;
      margin: 0;
      padding: 20px;
      background-color: #f0f0f0;
    }
    .container {
      max-width: 800px;
      margin: 0 auto;
      background-color: white;
      padding: 30px;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0,0,0,0.1);
    }
    h1 {
      color: #2c3e50;
      text-align: center;
      border-bottom: 2px solid #3498db;
      padding-bottom: 10px;
    }
    p {
      color: #34495e;
      margin-bottom: 20px;
      text-align: center;
    }
    ul {
      background-color: #f8f9fa;
      padding: 20px;
      border-radius: 5px;
      list-style-type: square;
      list-style: inside;
    }
    table {
      width: 100%;
      border-collapse: collapse;
      margin: 20px 0;
    }
    th, td {
      border: 1px solid #ddd;
      padding: 12px;
      text-align: left;
```

```

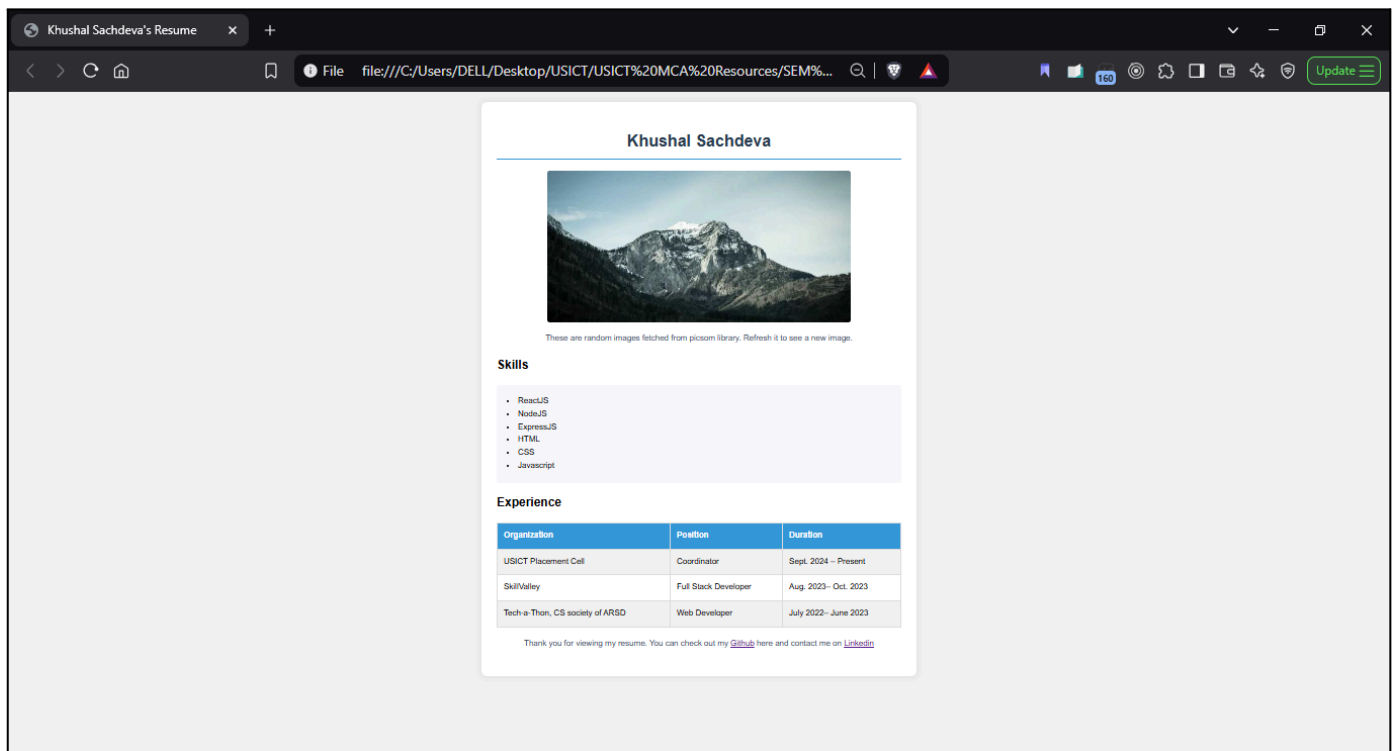
    }
    th {
        background-color: #3498db;
        color: white;
    }
    tr:nth-child(even) {
        background-color: #f2f2f2;
    }
    img {
        display: block;
        max-width: 100%;
        height: auto;
        margin: 20px auto;
        border-radius: 5px;
    }
</style>
</head>
<body>
    <div class="container">
        <h1>Khushal Sachdeva</h1>
        
        <p>These are random images fetched from picsom library. Refresh it to see a new
image.</p>
        <h2>Skills</h2>
        <ul>
            <li>ReactJS</li>
            <li>NodeJS</li>
            <li>ExpressJS</li>
            <li>HTML</li>
            <li>CSS</li>
            <li>Javascript</li>
        </ul>
        <h2>Experience</h2>
        <table>
            <tr>
                <th>Organization</th>
                <th>Position</th>
                <th>Duration</th>
            </tr>
            <tr>
                <td>USICT Placement Cell</td>
                <td>Coordinator</td>
                <td>Sept. 2024 - Present</td>
            </tr>
            <tr>
                <td>SkillValley</td>
                <td>Full Stack Developer</td>
                <td>Aug. 2023- Oct. 2023</td>
            </tr>
            <tr>
                <td>Tech-a-Thon, CS society of ARSD</td>
                <td>Web Developer</td>
                <td>July 2022- June 2023</td>
            </tr>
        </table>
    </div>
</body>
</html>

```

```

        </tr>
      </table>
      <p>Thank you for viewing my resume. You can check out my <a
href="https://github.com/CodeKhushal">Github</a> here and contact me on <a
href="https://www.linkedin.com/in/khushalsachdeva/">Linkedin</a></p>
    <p></p>
  </div>
</body>
</html>

```



# Practical - 2

## Forms, Frames, and Cascading Style Sheets (CSS)

**Objective:** Work with forms, frames, and style HTML pages using CSS.

**Tasks:**

1. Create an HTML form for user registration with fields like name, email, and password.
2. Use frames to split a webpage into multiple sections.
3. Style the form and frames using an external CSS file.

// registration.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Registration Form</title>
</head>
<frameset rows="20%,80%">
  <frame src="header.html" name="headerFrame" />
  <frameset cols="30%,70%">
    <frame src="menu.html" name="menuFrame" />
    <frame src="form.html" name="contentFrame" />
  </frameset>
</frameset>
</html>
```

//header.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Header</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="header">
    <h1>Welcome to Registration Form</h1>
    <p>You can register here!</p>
  </div>
</body>
</html>
```

//menu.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Menu</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="menu">
    <a href="home.html" target="contentFrame">Home</a>
    <a href="about.html" target="contentFrame">About</a>
    <a href="services.html" target="contentFrame">Services</a>
    <a href="contact.html" target="contentFrame">Contact</a>
  </div>
</body>
</html>
```

//style.css

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

.header {
  background-color: #4CAF50;
  color: white;
  text-align: center;
  padding: 10px 0;
}

.menu {
  padding: 10px;
  background-color: #ffffff;
  color: rgb(0, 0, 0);
  height: 100%;
  text-align: center;
}

.menu a {
  display: block;
  color: rgb(0, 0, 0);
  text-decoration: none;
  padding: 8px 0;
  border-bottom: 1px solid #444;
  margin: 5px 0;
}

.menu a:hover {
  background-color: #4CAF50;
}
```

//form.html

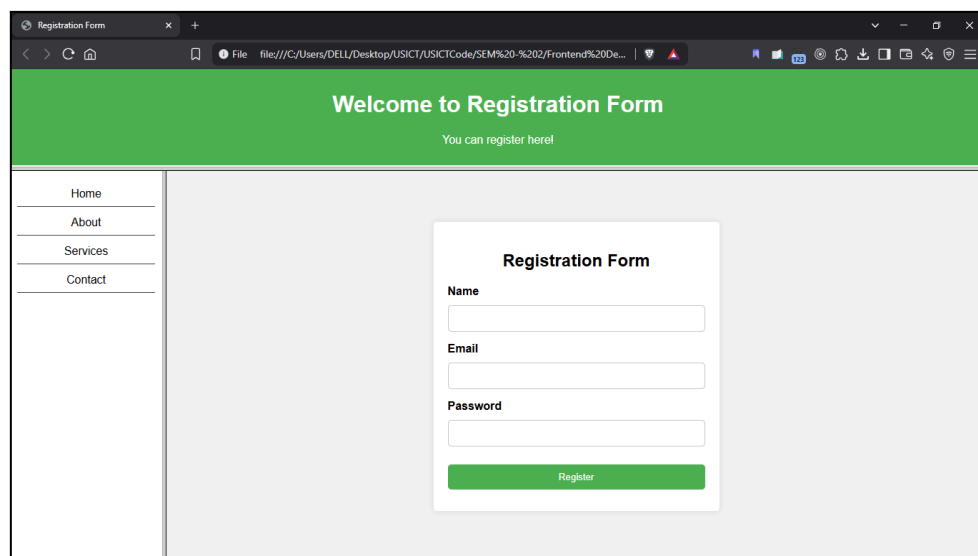
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Register</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f2f2f2;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
    }
    .container {
      background-color: #fff;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      width: 100%;
      max-width: 400px;
      box-sizing: border-box;
    }
    .container h2{
      text-align: center;
      margin-bottom: 20px;
    }
    .container label {
      display: block;
      margin-bottom: 5px;
      font-weight: bold;
    }
    .container input[type="text"],
    .container input[type="email"],
    .container input[type="password"] {
      width: 100%;
      padding: 10px;
      margin: 5px 0 15px 0;
      border: 1px solid #ccc;
      border-radius: 5px;
      box-sizing: border-box;
    }
    .container input[type="submit"] {
      width: 100%;
      margin: 10px 0;
      padding: 10px;
      background-color: #4CAF50;
      color: white;
      border: none;
      border-radius: 5px;
    }
  </style>
</head>
<body>
  <div class="container">
    <h2>Register</h2>
    <div>
      <label>Name</label>
      <input type="text">
    </div>
    <div>
      <label>Email</label>
      <input type="email">
    </div>
    <div>
      <label>Password</label>
      <input type="password">
    </div>
    <input type="submit" value="Register">
  </div>
</body>
</html>
```

```

        cursor: pointer;
    }
    .container input[type="submit"]:hover {
        background-color: #45a049;
    }
}
@media (max-width: 600px) {
    .container {
        padding: 15px;
    }
    .container input[type="text"],
    .container input[type="email"],
    .container input[type="password"] {
        padding: 8px;
    }
    .container input[type="submit"] {
        padding: 8px;
    }
}
}
</style>
</head>
<body>
    <div class="container">
        <h2>Registration Form</h2>
        <form action="/" method="post" onsubmit="validateForm(event)">
            <label for="name">Name</label>
            <input type="text" id="name" name="name" required>
            <label for="email">Email</label>
            <input type="email" id="email" name="email" required>
            <label for="password">Password</label>
            <input type="password" id="password" name="password" required>

            <input type="submit" value="Register">
        </form>
    </div>
</body>
</html>

```



# Practical - 3

## JavaScript Basics and Dynamic HTML

**Objective:** Learn JavaScript basics and implement dynamic behaviors in HTML.

**Tasks:**

1. Write a JavaScript program to validate the email field in the form created earlier.

// registration.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Register</title>
</head>
<body>
  <div class="container">
    <h2>Registration Form</h2>
    <form action="/" method="post" onsubmit="validateForm(event)">
      <label for="name">Name</label>
      <input type="text" id="name" name="name" required>

      <label for="email">Email</label>
      <input type="email" id="email" name="email" required>

      <label for="password">Password</label>
      <input type="password" id="password" name="password" required>

      <input type="submit" value="Register">
    </form>
  </div>

  <script>
    const emailTag = document.getElementById("email");

    function validateForm(event) {
      const emailValue = emailTag.value;
      const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
      if (!emailRegex.test(emailValue)) {
        alert("Please enter a valid email address.");
        event.preventDefault();
        return;
      }
    }
  </script>
</body>
</html>
```



Registration Form

File file:///C:/Users/DELL/Desktop/USICT/USICTCode/SEM%20-%20202/Frontend%20De... 123

This page says  
Please enter a valid email address.  
OK

Home

About

Services

Contact

Registration Form

Name

Khushal

Email

work.khushalsachdeva@gmailcom

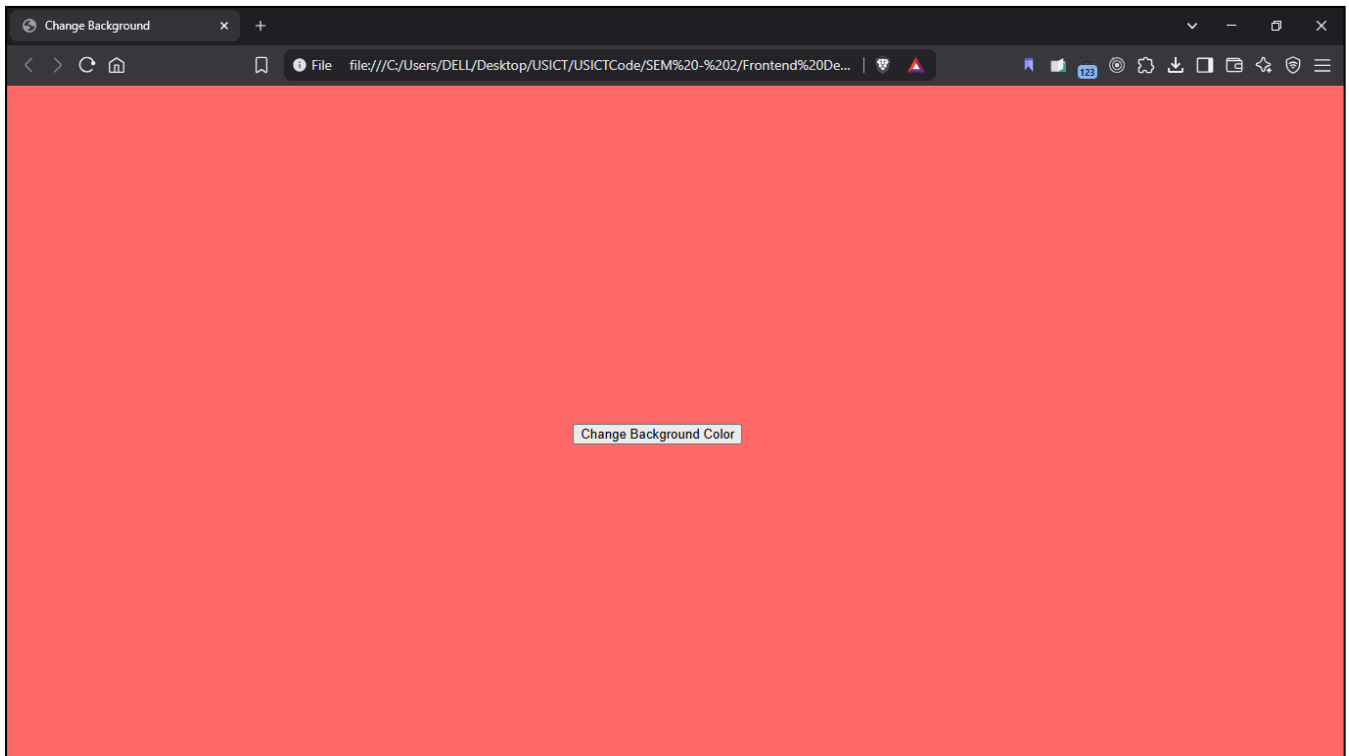
Password

.....

Register

**2. Use JavaScript to dynamically change the background color of a web page on button click.**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Change Background</title>
</head>
<body>
  <button
    id="change-bg-button"
    style="position: absolute; top: 50%; right: 45%;"
  >
    Change Background Color
  </button>
  <script>
    document.getElementById('change-bg-button').addEventListener('click', function() {
      let colors = ["#f8b400", "#ff6b6b", "#6a0dad", "#4caf50", "#3498db"];
      document.body.style.backgroundColor =
        colors[Math.floor(Math.random() * colors.length)];
    });
  </script>
</body>
</html>
```



# Practical - 4

## Advanced JavaScript and Objects

**Objective:** Explore JavaScript objects and manipulate HTML elements dynamically.

**Tasks:**

1. Create a JavaScript object for a "Product" with properties like name, price, and category.
2. Use the object to display product details dynamically in an HTML table.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Product Details</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      padding: 30px;
      background-color: #f4f4f9;
    }
    h2 {
      text-align: center;
      color: #333;
    }
    table {
      width: 80%;
      margin: 0 auto;
      border-collapse: collapse;
      box-shadow: 0 0 15px rgba(0, 0, 0, 0.2);
      background-color: #ffffff;
    }
    th,
    td {
      padding: 12px 20px;
      border: 1px solid #ddd;
      text-align: left;
    }
    th {
      background-color: #28a745;
      color: white;
    }
    tr:nth-child(even) {
      background-color: #f2f2f2;
    }
    tr:hover {
```

```
        background-color: #d4edda;

    }
</style>
</head>

<body>

    <h2>Products Available</h2>
    <table id="productTable">
        <thead>
            <tr>
                <th>Name</th>
                <th>Price</th>
                <th>Category</th>
            </tr>
        </thead>
        <tbody>
        </tbody>
    </table>

    <script>
        const products = [
            { name: "Smartphone", price: 25000, category: "Electronics" },
            { name: "Headphones", price: 3000, category: "Audio" },
            { name: "Sports Watch", price: 8000, category: "Wearables" },
            { name: "Backpack", price: 2000, category: "Accessories" },
            { name: "Pen", price: 100, category: "Stationery" }
        ];

        const tableBody = document.querySelector('#productTable tbody');

        products.forEach(product => {
            const row = document.createElement('tr');
            row.innerHTML = `
                <td>${product.name}</td>
                <td>${product.price.toFixed(2)}</td>
                <td>${product.category}</td>
            `;
            tableBody.appendChild(row);
        });
    </script>

</body>

</html>
```

## Products Available

Name	Price	Category
Smartphone	25000.00	Electronics
Headphones	3000.00	Audio
Sports Watch	8000.00	Wearables
Backpack	2000.00	Accessories
Pen	100.00	Stationery

# Practical - 5

## Introduction to Bootstrap

**Objective:** Build responsive web pages using Bootstrap.

### Tasks:

1. Create a responsive navigation bar using Bootstrap classes.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Responsive Navbar</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>

<body>
  <!-- Navigation Bar -->
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top">
    <div class="container-fluid">
      <a class="navbar-brand" href="#">MyWebsite</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarNav"
      aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle
navigation">
        <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" aria-current="page"
href="#home">Home</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#about">About</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#services">Services</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#contact">Contact</a>
          </li>
        </ul>
      </div>
    </div>
  </nav>
```

```
</nav>

<!-- Home Section -->
<section id="home" class="d-flex flex-column justify-content-center
align-items-center bg-light vh-100">
  <h1>Welcome to MyWebsite</h1>
  <p>Your one-stop solution for all your needs.</p>
</section>

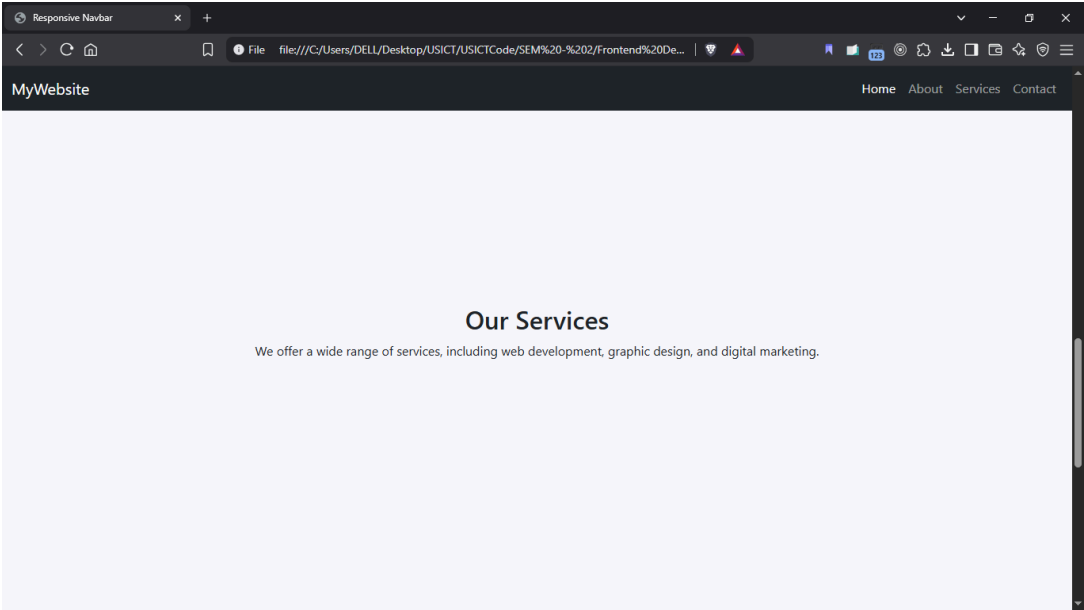
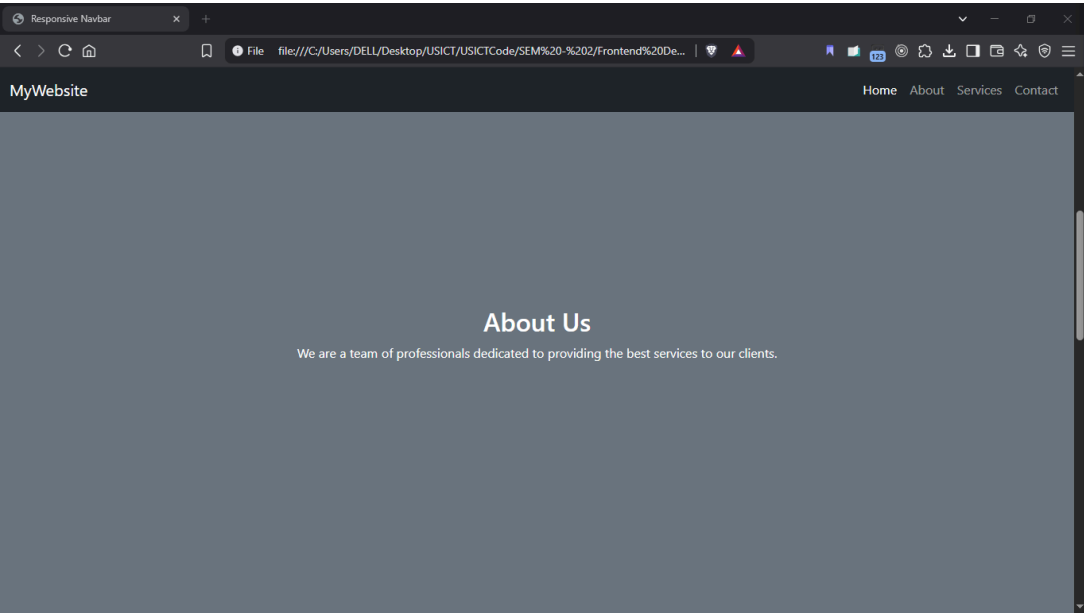
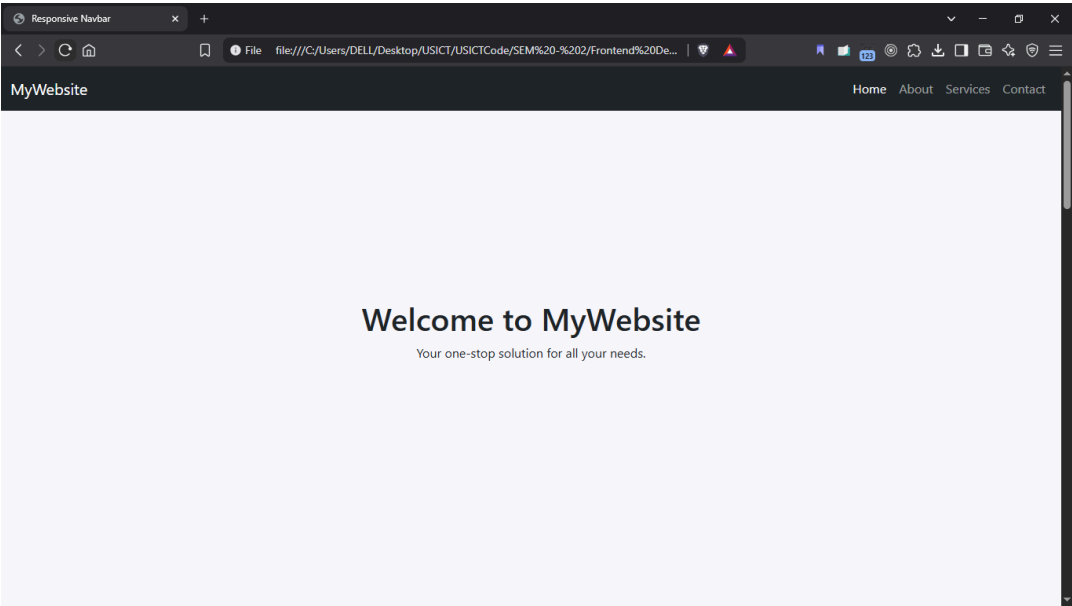
<!-- About Section -->
<section id="about"
  class="d-flex flex-column justify-content-center align-items-center
bg-secondary text-white vh-100">
  <h2>About Us</h2>
  <p>We are a team of professionals dedicated to providing the best services
to our clients.</p>
</section>

<!-- Services Section -->
<section id="services" class="d-flex flex-column justify-content-center
align-items-center bg-light vh-100">
  <h2>Our Services</h2>
  <p>We offer a wide range of services, including web development, graphic
design, and digital marketing.</p>
</section>

<!-- Contact Section -->
<section id="contact"
  class="d-flex flex-column justify-content-center align-items-center
bg-secondary text-white vh-100">
  <h2>Contact Us</h2>
  <p>Feel free to reach out to us at contact@mywebsite.com or call us at +123
456 7890.</p>
</section>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"><
/script>
</body>

</html>
```





## 2. Design a grid-based layout to display images with captions.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Image Grid</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>

<body>
  <div class="container my-5">
    <h2 class="text-center mb-4">Landscape Gallery</h2>
    <div class="row g-4">
      <div class="col-md-4">
        <div class="card p-2">
          
          <div class="card-body">
            <h5 class="card-title text-center">Argentina</h5>
          </div>
        </div>
      </div>

      <div class="col-md-4">
        <div class="card p-2">
          
          <div class="card-body">
            <h5 class="card-title text-center">Brazil</h5>
          </div>
        </div>
      </div>

      <div class="col-md-4">
        <div class="card p-2">
          
          <div class="card-body">
            <h5 class="card-title text-center">India</h5>
          </div>
        </div>
      </div>

      <div class="col-md-4">
        <div class="card p-2">
          
          <div class="card-body">
            <h5 class="card-title text-center">Barbados</h5>
          </div>
        </div>
      </div>
    </div>
  </div>
```

```

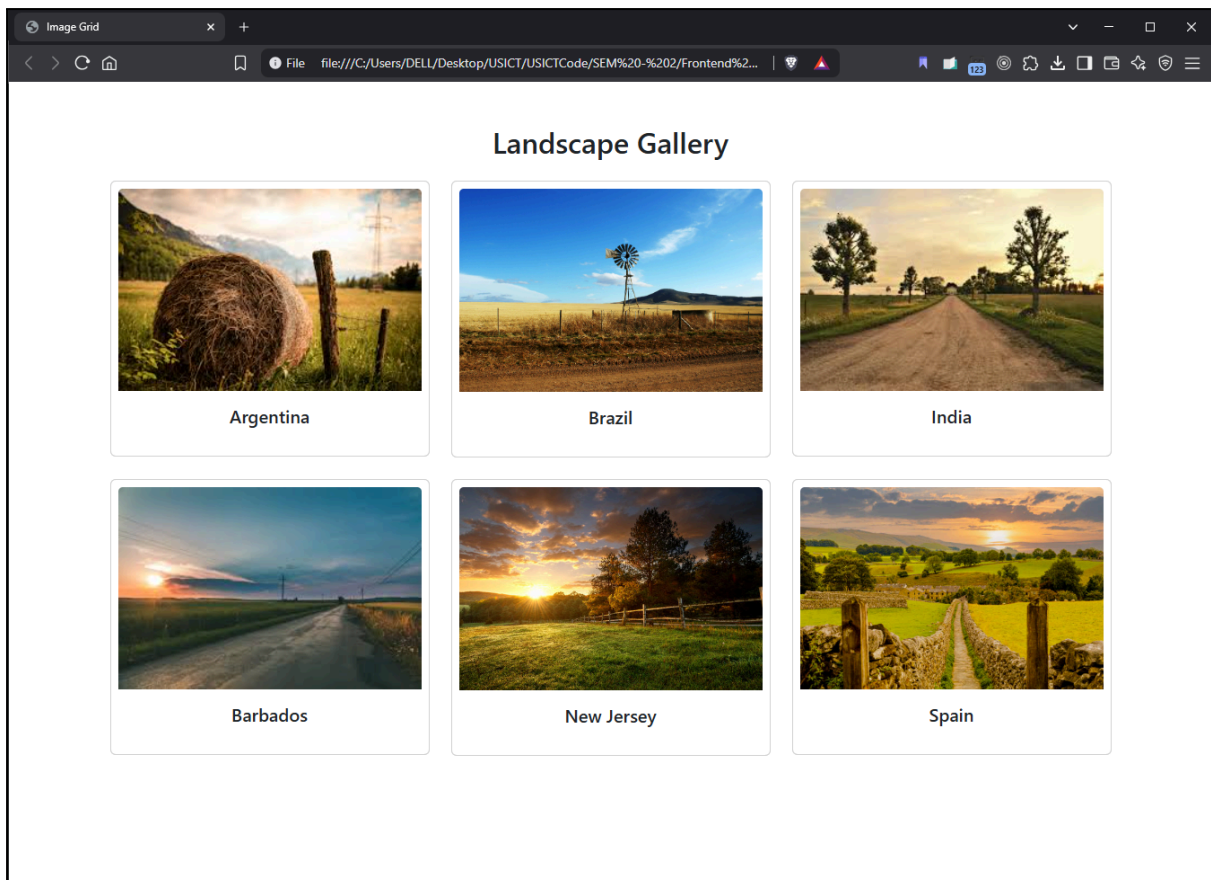
<div class="col-md-4">
  <div class="card p-2">
    
    <div class="card-body">
      <h5 class="card-title text-center">New Jersey</h5>
    </div>
  </div>
</div>

<div class="col-md-4">
  <div class="card p-2">
    
    <div class="card-body">
      <h5 class="card-title text-center">Spain</h5>
    </div>
  </div>
</div>
</div>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"><
/script>
</body>

</html>

```



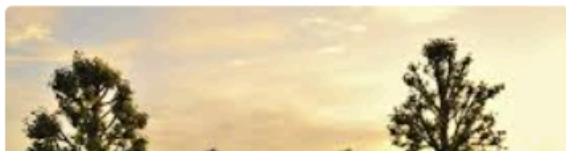
## Landscape Gallery



Argentina



Brazil



# Practical - 6

## Introduction to MEAN Stack

**Objective:** Understand the MEAN stack architecture and set up a basic project.

### Tasks:

#### 1. Install Node.js and MongoDB.

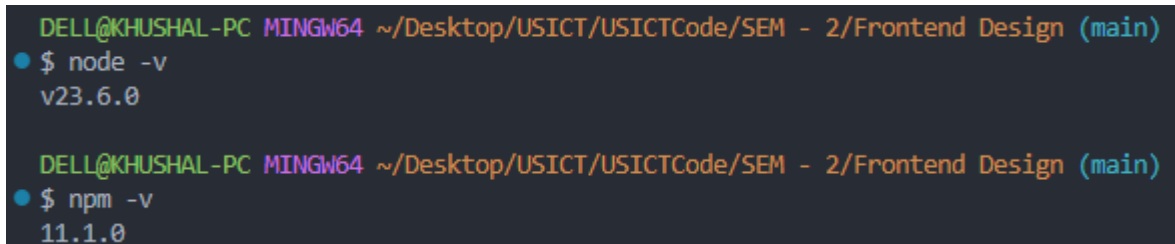
##### A. Install Node.js:

- (i) Go to the official Node.js website: <https://nodejs.org/>.
- (ii) Download and install the LTS (Long Term Support) version for your operating system.
- (iii) Accept the license agreement.
- (iv) Choose the installation path (default is recommended). Ensure the "Add to PATH" option is checked.
- (v) Verify Installation: Open the Command Prompt (cmd).

Run the following commands:

```
node -v
npm -v
```

These commands will display the installed versions of Node.js and npm (Node Package Manager).



```
DELL@KHUSHAL-PC MINGW64 ~/Desktop/USICT/USICTCode/SEM - 2/Frontend Design (main)
$ node -v
v23.6.0

DELL@KHUSHAL-PC MINGW64 ~/Desktop/USICT/USICTCode/SEM - 2/Frontend Design (main)
$ npm -v
11.1.0
```

##### B. Install MongoDB:

- (i) Go to the official MongoDB website <https://www.mongodb.com/try/download/community>.
- (ii) Select the Community Server edition. Download and install the LTS (Long Term Support) version for your operating system.
- (iii) Accept the license agreement. Ensure the "Install MongoDB as a Service" option is checked (this allows MongoDB to run automatically in the background).
- (iv) By default, MongoDB will be installed in C:\Program Files\MongoDB\Server\<version>\.
- (v) The data directory is usually located at C:\data\db. If it doesn't exist, create it manually.
- (v) Verify Installation: Open the Command Prompt (cmd).

Run the following command to start the MongoDB server:

```
mongod
```

This will open the MongoDB shell, indicating that MongoDB is running.

```
mongo
```

## 2. Create a simple Express server that responds with "Hello, World!"

```
DELL@KHUSHAL-PC MINGW64 ~/Desktop/USICT/USICTCode/SEM - 2/Frontend Design/LAB6 (main)
● $ npm init -y
Wrote to C:\Users\DELL\Desktop\USICT\USICTCode\SEM - 2\Frontend Design\LAB6\package.json:

{
  "name": "lab6",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs"
}
```

### Install Express

```
DELL@KHUSHAL-PC MINGW64 ~/Desktop/USICT/USICTCode/SEM - 2/Frontend Design/LAB6 (main)
● $ npm install express

added 66 packages, and audited 67 packages in 14s

14 packages are looking for funding
  run `npm fund` for details

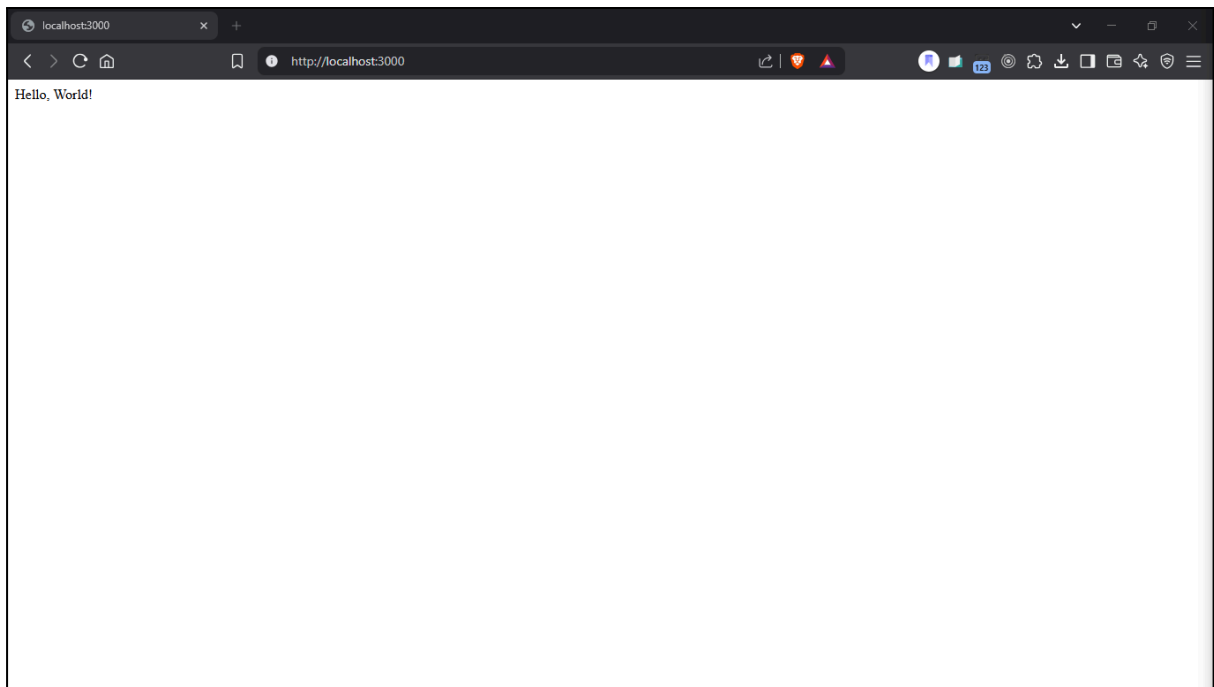
found 0 vulnerabilities
```

//server.js

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello, World!');
});
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

```
DELL@KHUSHAL-PC MINGW64 ~/Desktop/USICT/USICTCode/SEM - 2/Frontend Design/LAB6 (main)
○ $ node server.js
Server is running on port 3000
```



### 3. Connect the server to a MongoDB database and store/retrieve a simple record.

// db.js

```
const express = require("express");
const mongoose = require("mongoose");
const app = express();

mongoose
  .connect("mongodb://localhost:27017/userDB", {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => console.log("Connected to MongoDB"))
  .catch((err) => console.error("Error connecting to MongoDB:", err));

const userSchema = new mongoose.Schema({
  name: String,
  email: String,
});

const User = mongoose.model("User", userSchema);

app.get("/create", async (req, res) => {
  const user = new User({ name: "Khushal Sachdeva", email: "khushal@gmail.com" });
  await user.save();
  res.send("User created!");
});
```

```

app.get("/user", async (req, res) => {
  const user = await User.findOne({ email: "khushal@gmail.com" });
  if (user) {
    res.send(`User found: ${user.name} - ${user.email}`);
  } else {
    res.send("User not found");
  }
});

app.get("/", (req, res) => {
  res.send("Hello, World!");
});

const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

```

```

DELL@KHUSHAL-PC MINGW64 ~/Desktop/USICT/USICTCode/SEM - 2/Frontend Design/LAB6 (main)
● $ npm i mongoose

added 17 packages, and audited 84 packages in 15s

15 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

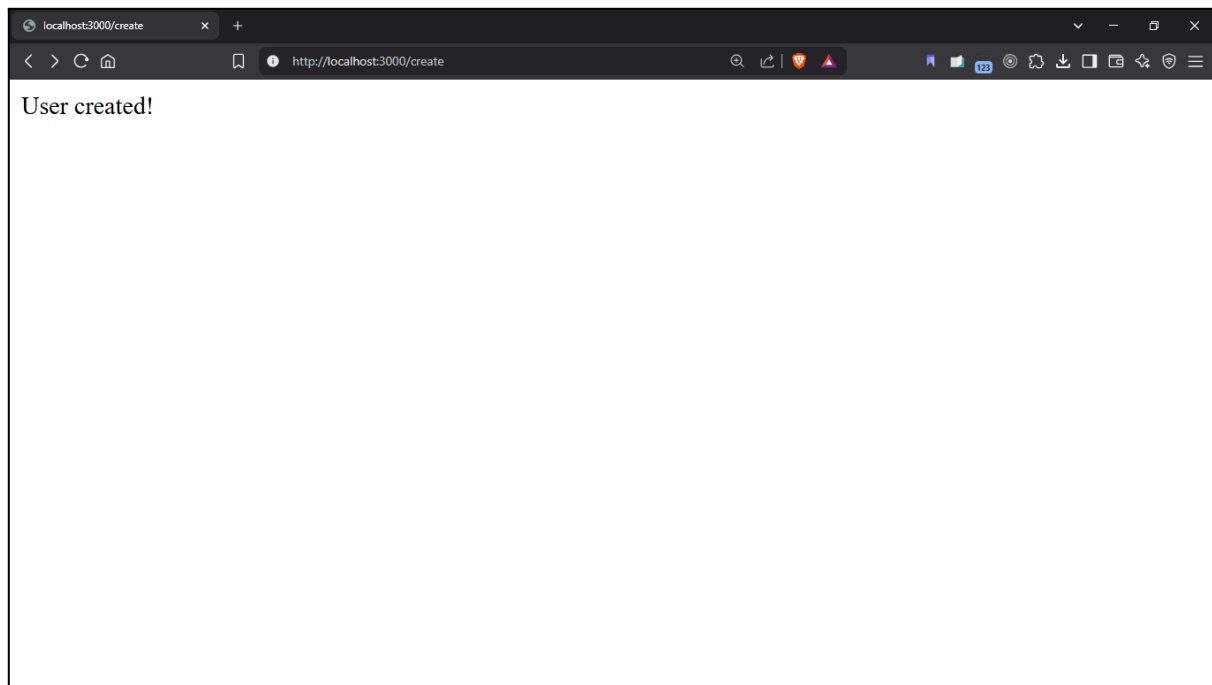
```

```

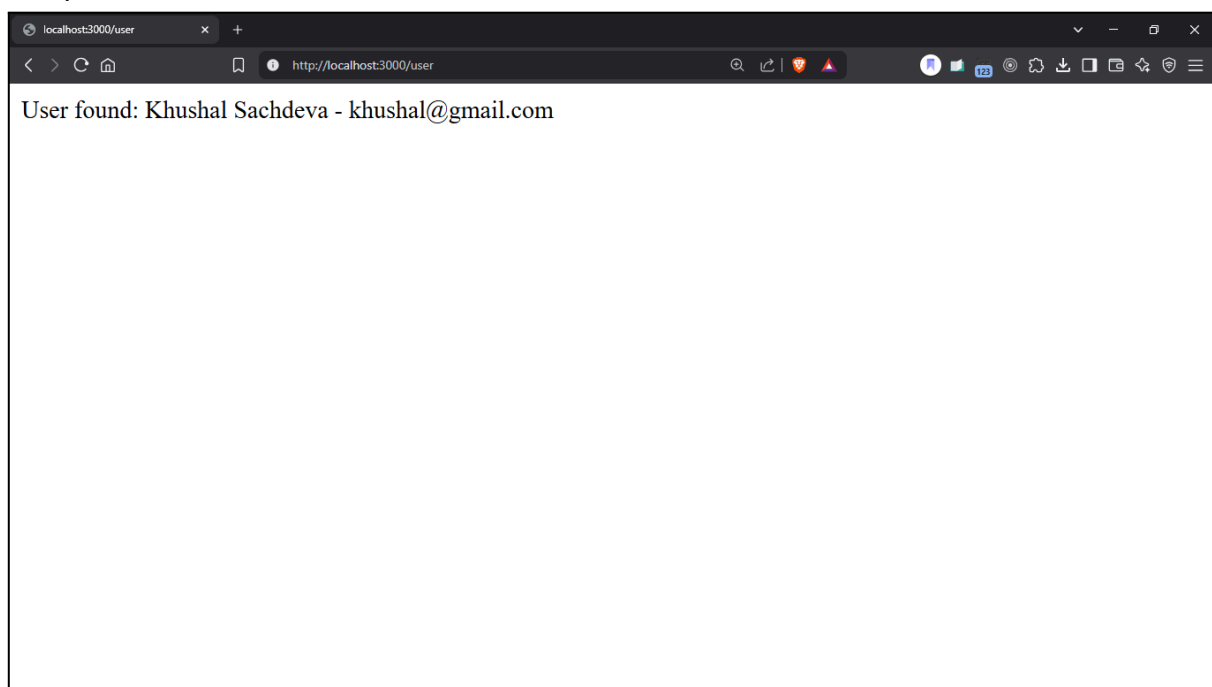
DELL@KHUSHAL-PC MINGW64 ~/Desktop/USICT/USICTCode/SEM - 2/Frontend Design/LAB6 (main)
$ node db.js
(node:9964) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will
be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:9964) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 an
d will be removed in the next major version
(node:9964) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 an
d will be removed in the next major version
Server is running on port 3000
Connected to MongoDB
+--[4A(node:9964) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.
0 an(node:9964) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.
0 an(node:9964) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.
0 an(node:9964) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 an
d will be removed in the next major version
Server is running on port 3000
Connected to MongoDB

```

// http://localhost:3000/create



// http://localhost:3000/create



```
_id: ObjectId('67fce0486e9bbd9f98ae3838')  
name : "Khushal Sachdeva"  
email : "khushal@gmail.com"  
__v : 0
```