



**AMRITA**  
**VISHWA VIDYAPEETHAM**



---

**22AIE457**

---

**FULL STACK DEVELOPMENT  
LAB REPORT**



**SHREE PRASAD M  
CH.EN.U4AIE22050**



# AMRITA

## VISHWA VIDYAPEETHAM

### List of Lab Experiments

S. No	List of Experiments	Page
01	Git and GitHub Tutorial	3
02	Installation of Node.js	6
03	HTML and CSS	11
04	HTML Responsive Web Design	22
05	Using React in Visual Studio Code	25
06	JavaScript	28
07	Login Form using Node.js and MongoDB	40
08	Basic Login System with Node.js, Express, and MySQL	47
09	Live Editable Table with jQuery, PHP and MySQL	51
10	AJAX Forms with jQuery	54
11	AJAX Database Operations with PHP and MySQL	59

# Ex 1: Git and GitHub Tutorial - Version Control for Beginners

## Prerequisites

To complete this tutorial, you'll need the following:

- A command line interface.
- A text editor of your choice (e.g., VS Code).
- A GitHub account.

## What is Git?

Git is a version control system which lets you track changes you make to your files over time. With Git, you can revert to various states of your files (like a time traveling machine). You can also make a copy of your file, make changes to that copy, and then merge these changes to the original copy. For example, if you were working on a website's landing page and wanted to experiment with the navigation bar, you could create a copy (a branch), make your changes, and then merge it back into the original file once you're satisfied. Git is not just for developers; it can track changes in text files, images, and more.

## How to push a repository to GitHub

Before pushing, you need to understand the stages of a file being tracked by Git:

- **Modified state:** A file in the modified state has some changes made to it but it's not yet saved to the local repository.
- **Staged state:** A file in the staged state is marked as ready to be committed. All necessary changes have been made.
- **Committed state:** A file is in the committed state when all the changes have been saved in the local repo. Files in this stage are ready to be pushed to the remote repo (on GitHub).

## Commands

# Navigate into your project folder.

# This command works on both Windows and Linux/macOS. Or clone an existing repo

git clone <link for repository>

```
PS C:\Users\CH.EN.U4AIE22050\Downloads\new> git clone https://github.com/S-Prasad-M/git-and-github-tutorial.git
Cloning into 'git-and-github-tutorial'...
info: please complete authentication in your browser...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
fatal: not a git repository (or any of the parent directories): .git
```

# Initialize a Git repository

git init

# Add all files to the staging area

git add .

```
PS C:\Users\CH.EN.U4AIE22050\Downloads\new> git init
Initialized empty Git repository in C:/Users/CH.EN.U4AIE22050/Downloads/new/.git/
PS C:\Users\CH.EN.U4AIE22050\Downloads\new> git add .
warning: adding embedded git repository: git-and-github-tutorial
```

# Check the status of your files

git status

```
PS C:\Users\CH.EN.U4AIE22050\Downloads\new> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   git-and-github-tutorial

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
  (commit or discard the untracked or modified content in submodules)
        modified:   git-and-github-tutorial (untracked content)
```

# Commit your changes

git commit -m "first commit"

```
PS C:\Users\CH.EN.U4AIE22050\Downloads\new> git commit -m "committed"
[master (root-commit) 9188f00] committed
 1 file changed, 1 insertion(+)
 create mode 160000 git-and-github-tutorial
PS C:\Users\CH.EN.U4AIE22050\Downloads\new> █
```

# Rename the default branch to 'main'

git branch -M main

# Push your committed changes to the remote repository

git push -u origin main

## How to Use Branches in Git

With branches, you can create a copy of a file you would like to work on without messing up the original copy. You can either merge these changes to the original copy or just let the branch remain independent.

To create a new branch, run the `git checkout -b <branch_name>` command.

After making changes in your new branch, you stage and commit them. Then, to merge them back into the main branch, you switch back to main and run `git merge <branch_name>`.

## Branching and Merging Commands

# Create a new branch named 'test' and switch to it  
`git checkout -b test`

# Switch back to the main branch  
`git checkout main`

# See all branches in your repository  
`git branch`

# Merge the 'test' branch into the current branch (main)  
`git merge test`

# Push the 'test' branch to the remote repository  
`git push -u origin test`

## How to Pull a Repository in Git

To "pull" in Git means to clone or download a remote repository's current state to your computer. This is useful when working from a different computer or contributing to an open-source project.

## Cloning Command

# Navigate up one directory. This command works on both Windows and Linux/macOS.  
`cd ..`

# Clone a remote repository to your local machine  
`git clone <url>`

```
PS C:\Users\CH.EN.U4AIE22050\Downloads\new> git clone https://github.com/S-Prasad-M/git-and-github-tutorial.git
Cloning into 'git-and-github-tutorial'...
info: please complete authentication in your browser...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
fatal: not a git repository (or any of the parent directories): .git
```

## Ex 2: Installation of Node.js on Windows

Node.js can be installed in multiple ways. The most common approach is using the installer from the official website.

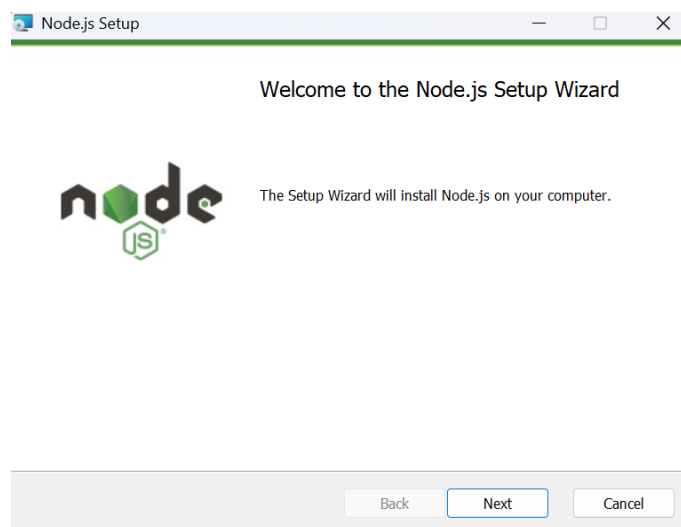
Step 1: Download the NodeJS Installer

Visit the official Node.js website at [Download and install Node.js](#) and download the .msi installer for Windows.

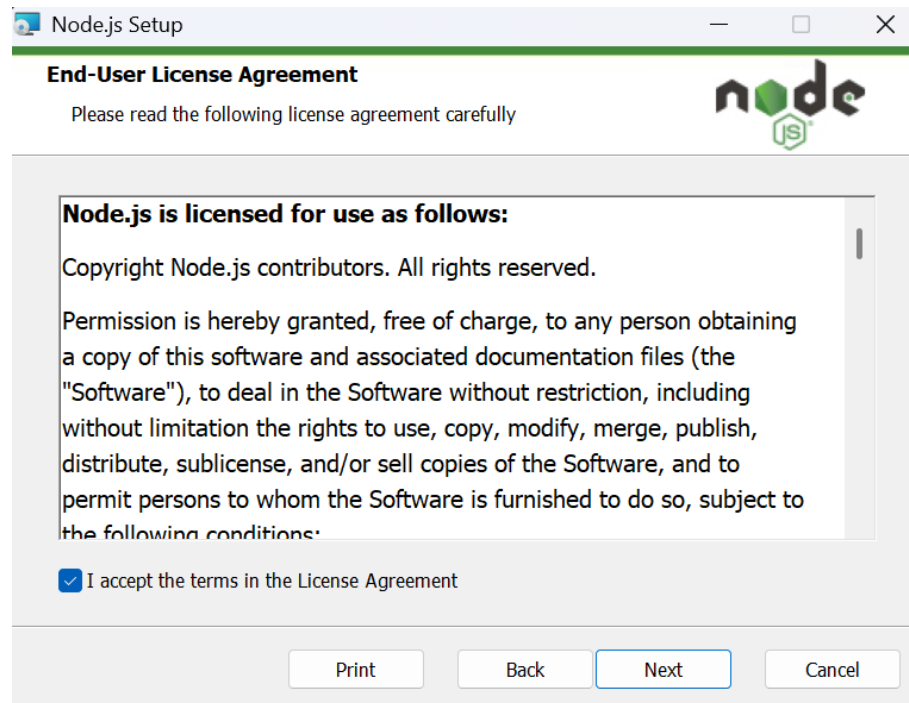
Step 2: Running the Node.js installer

Double-click the downloaded .msi file to launch the setup wizard. Follow the prompts:

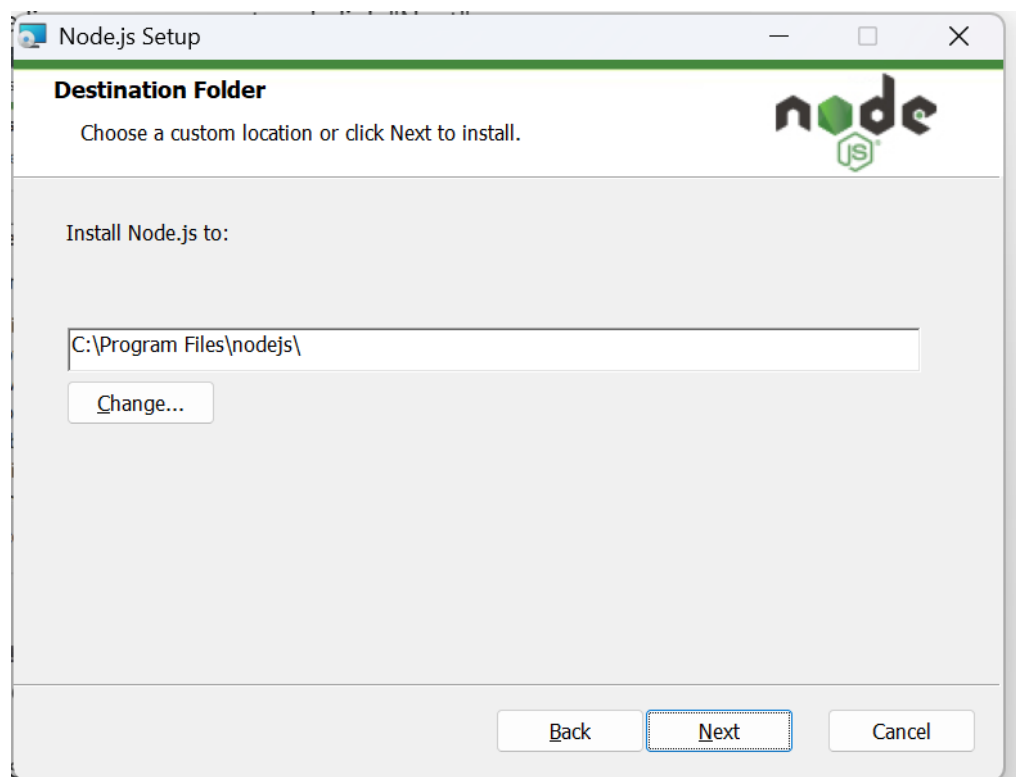
1. Click "Next" on the welcome screen.



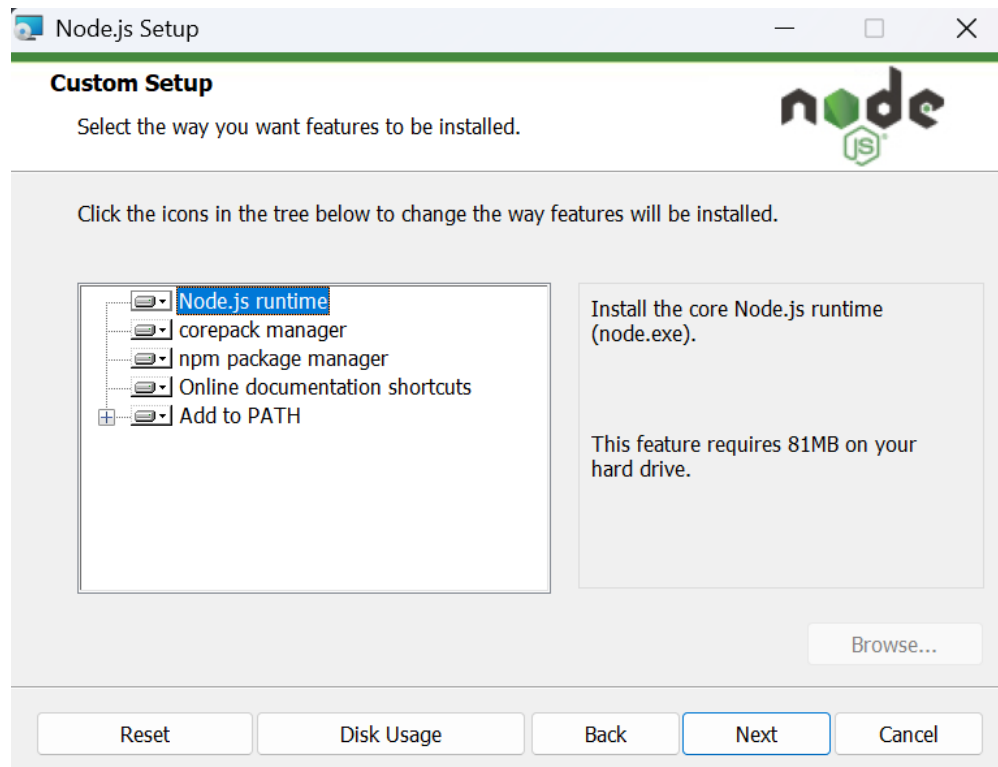
- Accept the license agreement and click "Next".



2. Choose the destination folder and click "Next".



3. On the Custom Setup screen, click "Next".



4. Click "Install" to begin the installation.
5. Click "Finish" to complete the installation.

### Step 3 & 4: Verify Installation and Update NPM

Open Command Prompt or PowerShell and run the following commands to verify that Node.js and npm are installed correctly and to update npm to the latest version.

#### Verification Commands

```
# Verify Node.js installation
node -v
```

```
# The output should be the version number, e.g., v10.15.3
# Update npm (Node Package Manager)
npm install npm --global
```



## Step 5 & 6: Running a Sample Application

Create a file named `server.js`, add the following code, and run it from your terminal using `node server.js`. Then, open your browser and navigate to `http://127.0.0.1:3000` to see the "Hello World" message.

### `server.js`

```
JS server.js > [?] http
1 // Import the http module
2 const http = require("http");
3
4 // Create a server
5 const server = http.createServer((req, res) => {
6   res.statusCode = 200;
7   res.setHeader("Content-Type", "text/plain");
8   res.end("Welcome to the Node.js Tutorial");
9 });
10
11 // Listen on port 3000
12 server.listen(3000, () => {
13   console.log(
14     "Server is running on http://localhost:3000");
15 });
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! <https://aka.ms/PSWindows>

PS C:\Users\CH.EN.U4AIE22050\Downloads\FSD-26> node server.js  
Server is running on http://localhost:3000

```
JS server.js > [?] http
1 // Import the http module
2 const http = require("http");
3
4 // Create a server
5 const server = http.createServer((req, res) => {
6   res.statusCode = 200;
7   res.setHeader("Content-Type", "text/plain");
8   res.end("Welcome to the Node.js Tutorial");
9 });
10
11 // Listen on port 3000
12 server.listen(3000, () => {
13   console.log(
14     "Server is running on http://localhost:3000");
15 });
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! <https://aka.ms/PSWindows>

PS C:\Users\CH.EN.U4AIE22050\Downloads\FSD-26> node server.js  
Server is running on http://localhost:3000

## Step 7: Using VS Code with Node.js

Create a folder, open it in VS Code, and create a file `First.js` with the following content. You can run it from the integrated terminal.

## First.js

```
const x = "My first web page";  
console.log(x);
```

// To run from terminal: node ./First.js

```
JS test1.js > ...  
1  const http = require("http");  
2  
3  const server = http.createServer((req, res) => {  
4    res.statusCode = 200;  
5    res.setHeader("Content-Type", "text/plain");  
6    res.end(`${greet("Developer")}\nThe sum of 5 and 4 is: ${add(5, 4)}`);  
7  });  
8  let name = 'Node.js';  
9  const version = 20;  
10  
11  function greet(user) {  
12    return `Hello, ${user}!`;  
13  }  
14  const add = (a, b) => a + b;  
15  
16  server.listen(3000, () => {  
17    console.log("Server is running on http://localhost:3000");  
18  });  
19  
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
PS C:\Users\CH.EN.U4AIE22050\Downloads\FSD-26> node test1.js  
Server is running on http://localhost:3000  
█
```

```
localhost:3000  x  +  
←  ↻  ⓘ  localhost:3000  
Hello, Developer!  
The sum of 5 and 4 is: 9
```

## Ex 3: HTML and CSS

### Part A: Basic Structure of an HTML Document

An HTML document starts with a `<!DOCTYPE html>` declaration, followed by `<html>`, `<head>`, and `<body>` tags. The `<head>` contains meta-information and the `<title>`, while the `<body>` contains the visible page content.

### Part B: Online Book Store Static Pages

This exercise involves creating a static website for an online bookstore using HTML frames.

#### home.html

```
<html>
<head>
  <frameset rows="20%,80%">
    <frame name="title" src="titlepage.html">
    <frameset cols="25%,75%">
      <frame name="list" src="list.html">
      <frame name="display" src="display.html">
    </frameset>
  </frameset>
</head>
</html>
```

#### titlepage.html

```
<html>
<head>
  <title>title page</title>
</head>
<body bgcolor="yellow" text="red">
  <h1 align="center"> ONLINE BOOK STORE </h1>
</body>
</html>
```

#### list.html

```
<html>
```

```
<head>
  <title> List</title>
</head>
<body bgcolor="pink">
  <center>
    <a href="login.html" target="display"><b> LOGIN </b></a><br>
    <a href="catalogue.html" target="display"><b> CATALOGUE </b></a>
  </center>
</body>
</html>
```

### **login.html**

```
<html>
<body bgcolor="cyan">
  <h2 align="center">LOGIN PAGE</h2>
  <center>
    <form>
      <table border="1">
        <tr>
          <td><label>USER NAME</label></td>
          <td><input type="text" name="uname"></td>
        </tr>
        <tr>
          <td><label>PASSWORD</label></td>
          <td><input type="password" name="pwd"></td>
        </tr>
        <tr>
          <td colspan="2" align="center"><input type="submit" value="LOGIN"></td>
        </tr>
      </table>
    </form>
  </center>
</body>
</html>
```

### **catalogue.html**

```
<html>
<body bgcolor="orange">
```

```

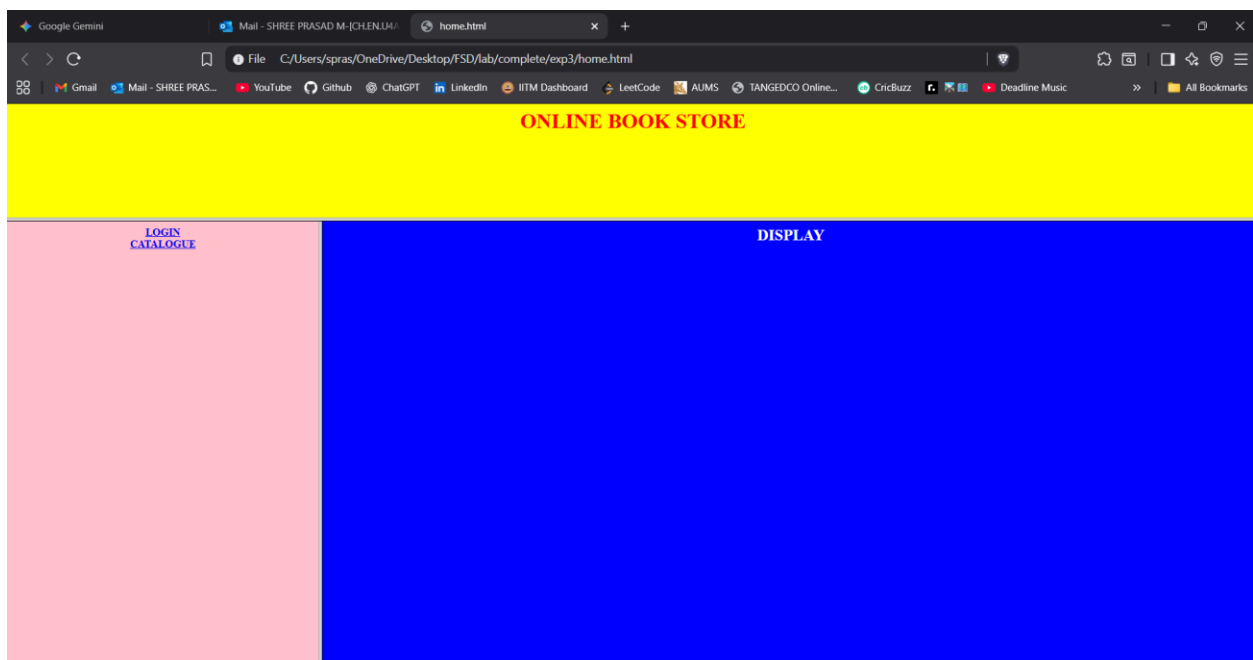
<h1 align="center">CATALOGUE PAGE</h1>
<center>
  <table border="1">
    <tr>
      <th>COVER IMAGE</th>
      <th>AUTHOR NAME</th>
      <th>TEXT BOOK-NAME</th>
      <th>PRICE</th>
      <th>ADD TO CART</th>
    </tr>
    <tr>
      <td></td>
      <td>Harsh Bhasin</td>
      <td>Programming in C#</td>
      <td>200/-</td>
      <td><input type="button" value="ADD TO CART"></td>
    </tr>
    <tr>
      <td></td>
      <td>Andrew S Tanenbaum</td>
      <td>Computer Networks</td>
      <td>400/-</td>
      <td><input type="button" value="ADD TO CART"></td>
    </tr>
    <tr>
      <td></td>
      <td>Chris Bates</td>
      <td>Web Programming, Building Internet Applications</td>
      <td>599/-</td>
      <td><input type="button" value="ADD TO CART"></td>
    </tr>
    <tr>
      <td></td>
      <td>Kogent</td>
      <td>Web Technologies, Black book</td>
      <td>499/-</td>
      <td><input type="button" value="ADD TO CART"></td>
    </tr>
  </table>

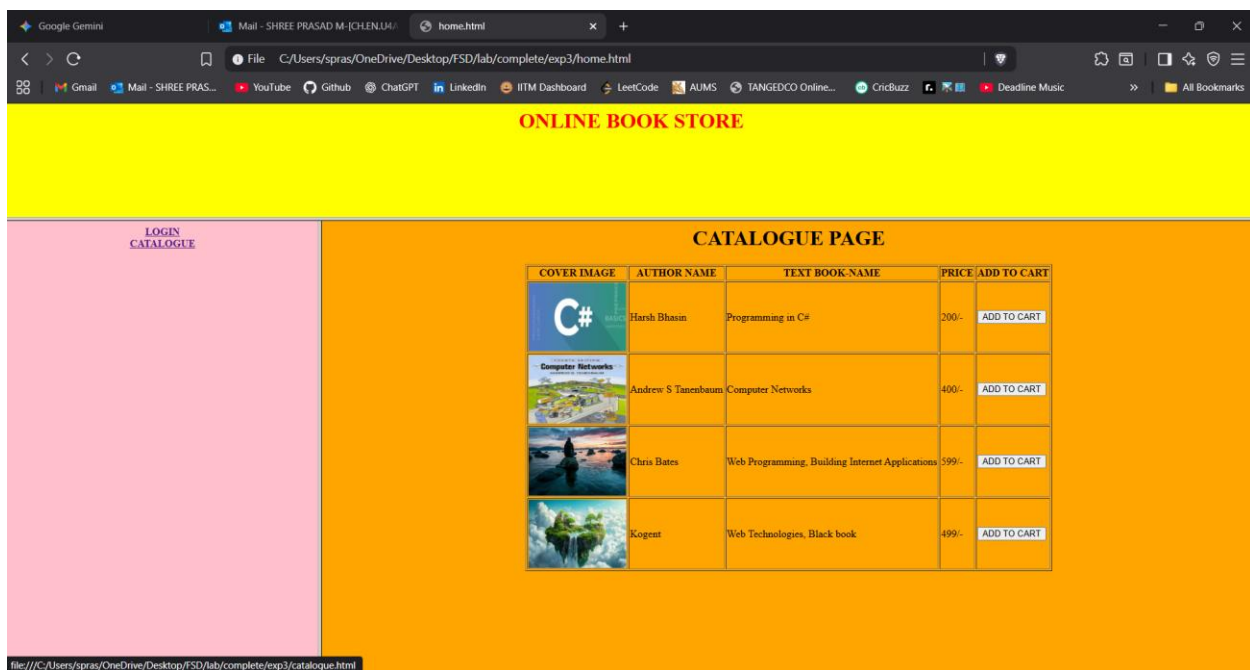
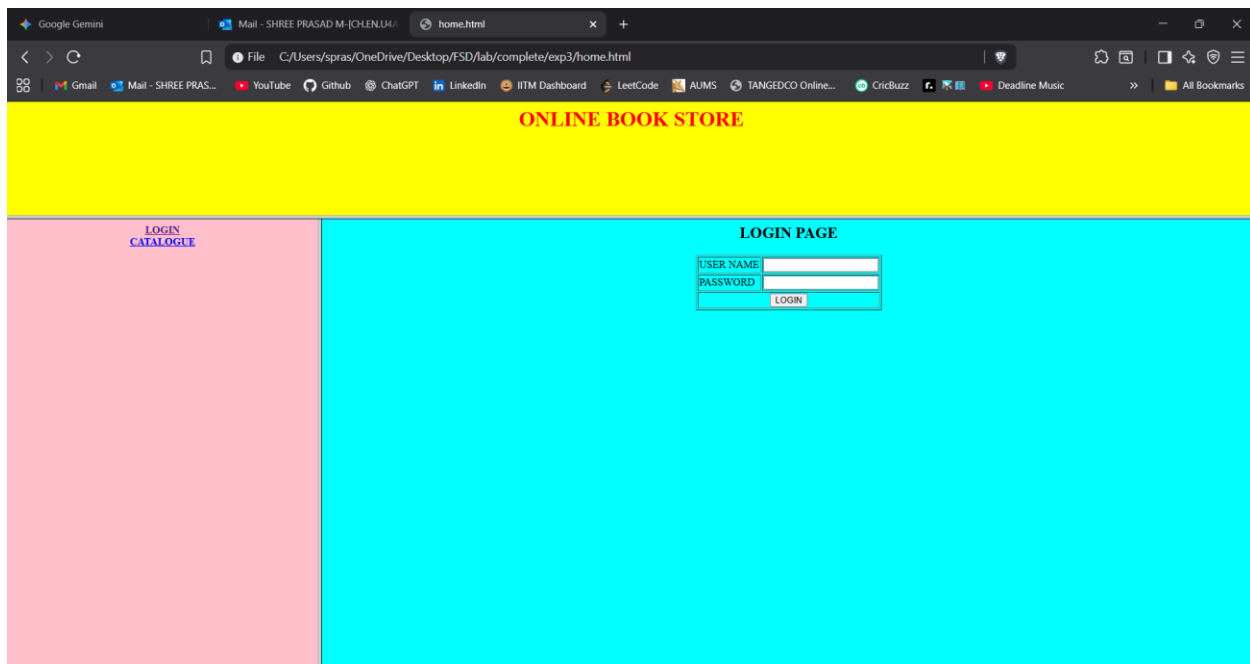
```

```
</table>
</center>
</body>
</html>
```

## display.html

```
<html>
<head>
  <title>display</title>
</head>
<body bgcolor="blue" text="white">
  <h2 align="center">DISPLAY</h2>
</body>
</html>
```





## Part C: Web Page Design with CSS

This exercise demonstrates various CSS properties to style a web page.

### 1. Font Styles

#### fontstyle.html

```
<html>
<head>
  <link rel="stylesheet" href="font.css">
</head>
<body>
  <p class="center">My text with font-arial</p>
  <p class="right">My text with font-calibre</p>
  <p class="left">My text with font-cooper</p>
</body>
</html>
```

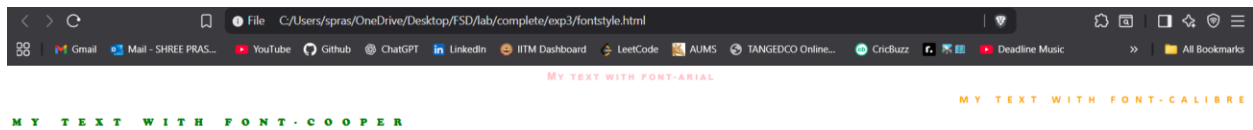
#### font.css

```
p.center {
  color: pink;
  text-align: center;
  letter-spacing: 3px;
  font-family: arial;
  font-variant: small-caps;
  font-weight: bold;
}
p.right {
  color: orange;
  text-align: right;
  letter-spacing: 10px;
  font-family: 'Calibri', sans-serif;
  text-transform: uppercase;
  font-weight: bold;
}
p.left {
  color: green;
  text-align: left;
  letter-spacing: 15px;
```



```
font-family: 'Cooper Black', sans-serif;
text-transform: uppercase;
font-weight: bold;
}
```

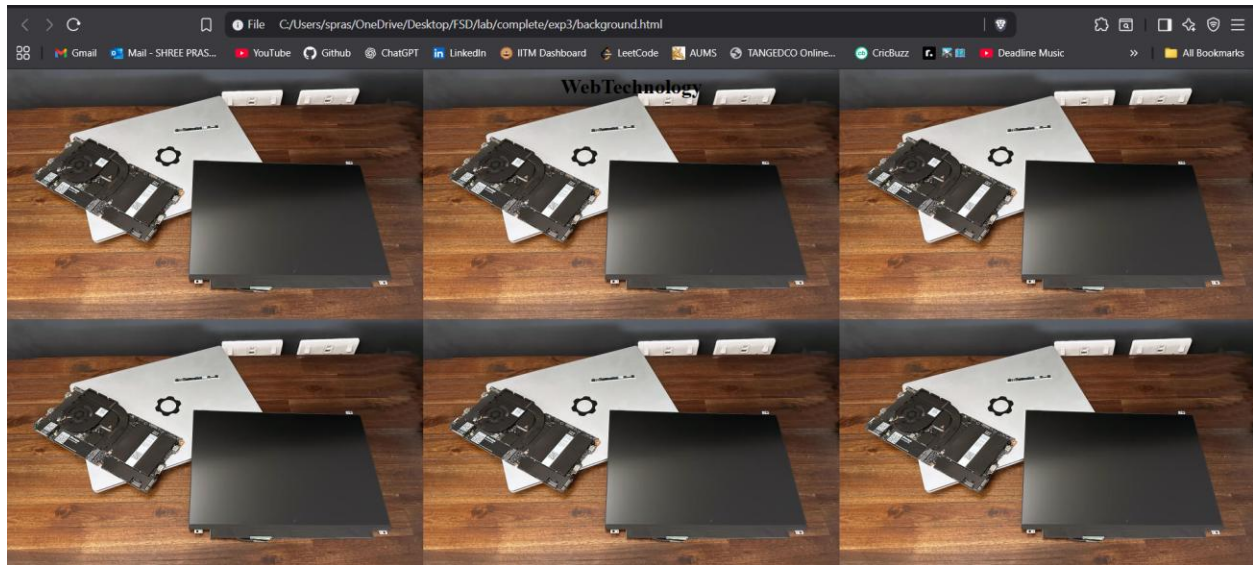
```
body{background-color: black}
```



## 2. Background Image

### background.html

```
<html>
<head>
  <style type="text/css">
    body {
      background-image: url(wt.jpg);
      background-repeat: repeat;
    }
  </style>
</head>
<body>
  <h1 align="center">WebTechnology</h1>
</body>
</html>
```



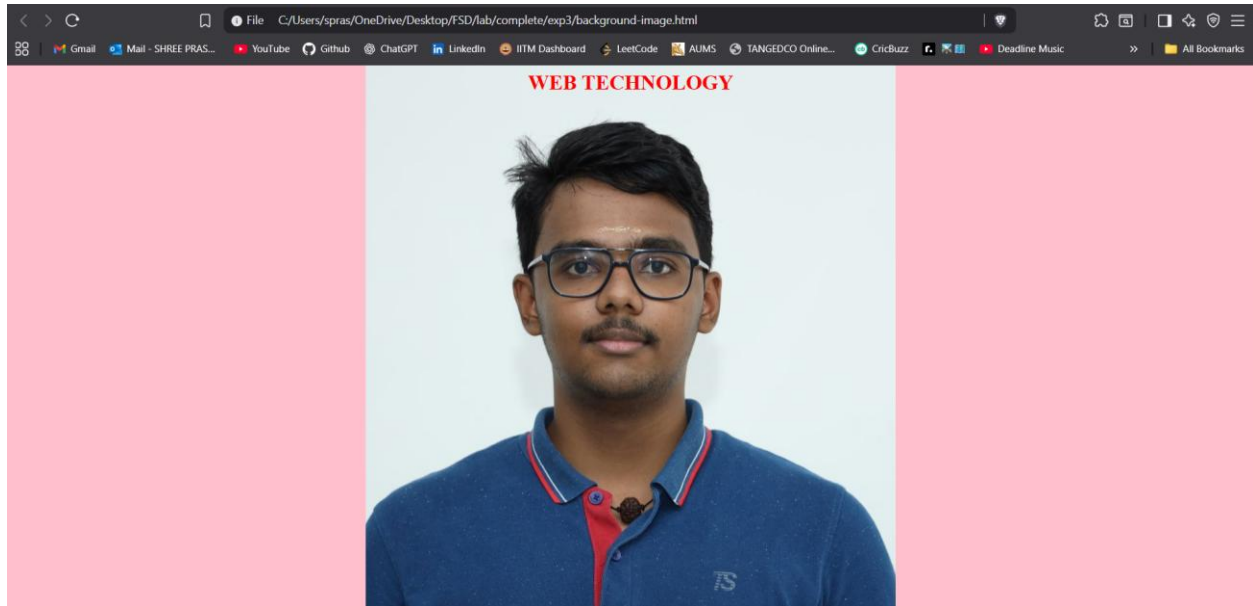
### 3. Background Image Properties

#### background-image.html

```
<html>
<head>
  <style type="text/css">
    body {
      background-image: url(wt1.jpg);
      background-color: rgb(111, 95, 97);
      background-repeat: no-repeat;
      background-position: center;
      background-attachment: fixed;
    }
    h1 {
      background-image: url(wt.jpg);
      background-position: right;
      color: rgb(116, 255, 232);
      background-repeat: no-repeat;
    }
  </style>
</head>
<body>
  <h1 align="center">WEB TECHNOLOGY<br></h1>
```

</body>

</html>



#### 4. Link Styles

##### Link.html

<html>

<head>

<style type="text/css">

a:link { color: red; text-decoration: underline; font-family: arial; font-size: 20pt; }

a:visited { color: yellow; text-decoration: line-through; font-family: arial; font-size: 20pt; }

a:hover { color: blue; text-decoration: underline; font-family: arial; font-size: 20pt; }

a:active { color: green; text-decoration: none; font-family: arial; font-size: 20pt; }

</style>

</head>

<body bgcolor="pink" align="center">

<a href="wt.jpg" target="\_blank">Link</a><br><br>

<a href="wt1.jpg" target="\_blank">Link1</a><br><br>

</body>

</html>

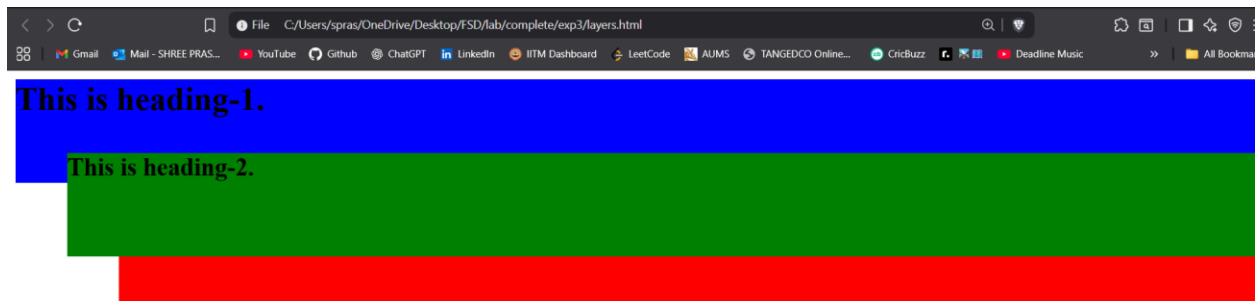
#### 5. Working with Layers

##### layers.html

```

<html>
<body bgcolor="white" align="left">
  <h1 style="background-color:blue; position:relative; top:0; left:0; z-index:2;
height:100px;">This is heading-1.</h1>
  <h2 style="background-color:green; position:relative; top:-50px; left:50px; z-index:3;
height:100px;">This is heading-2.</h2>
  <h3 style="background-color:red; position:relative; top:-125px; left:100px; z-index:1;
height:100px;">This is heading-3.</h3>
</body>
</html>

```



## 6. Customized Cursors

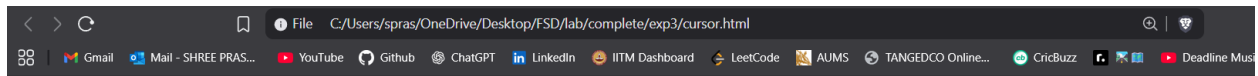
### cursor.html

```

<html>
<body>
  <p style="cursor:not-allowed;">NOT-ALLOWED CURSOR</p>
  <p style="cursor:progress;">PROGRESS CURSOR</p>
  <p style="cursor:wait;">WAIT CURSOR</p>
  <p style="cursor:zoom-in;">ZOOM-IN CURSOR</p>
  <p style="cursor:zoom-out;">ZOOM-OUT CURSOR</p>
  <p style="cursor:no-drop;">NO-DROP CURSOR</p>
  <p style="cursor: move;">MOVE CURSOR</p>
  <p style="cursor:default;">DEFAULT CURSOR</p>
  <p style="cursor:crosshair;">CROSSHAIR CURSOR</p>
  <p style="cursor:help;">HELP CURSOR</p>
</body>

```

</html>



NOT-ALLOWED CURSOR

PROGRESS CURSOR

WAIT CURSOR

ZOOM-IN CURSOR

ZOOM-OUT CURSOR

NO-DROP CURSOR

MOVE CURSOR

DEFAULT CURSOR

CROSSHAIR CURSOR

HELP CURSOR

## Ex 4: HTML Responsive Web Design

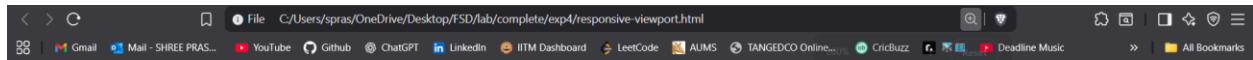
HTML Responsive Web Design is about creating web pages that look good on all devices (desktops, tablets, and phones). A responsive web design will automatically adjust for different screen sizes and viewports.

### 1. HTML Viewport Meta Tag

The viewport is the user's visible area of a web page. The `<meta name="viewport">` tag gives the browser instructions on how to control the page's dimensions and scaling.

#### Viewport Meta Tag Example

```
<!DOCTYPE html>
<html>
<head>
  <title>GeeksforGeeks Responsive Example</title>
  <meta charset="utf-8" name="viewport" content="width=device-width, initial-scale=1.0" />
  <style>
    .gfg { font-size: 40px; font-weight: bold; color: green; text-align: center; }
    .geeks { font-size: 17px; text-align: center; }
    p { text-align: justify; }
  </style>
</head>
<body>
  <div class="gfg">GeeksforGeeks</div>
  <div class="geeks">HTML Introduction</div>
  <p>HTML stands for HyperText Markup Language...</p>
</body>
</html>
```



# GeeksforGeeks

## HTML Introduction

HTML stands for HyperText Markup Language...

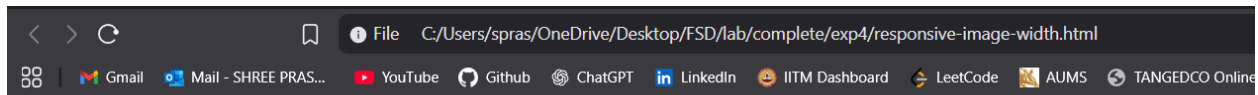
## 2. Responsive Images

Responsive images adjust their size based on the browser width.

### 2.1 Using width Property

Setting the CSS width property to 100% makes an image scale up and down.

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
</head>
<body>
  
  <h2>Responsive Images</h2>
  <p>Responsive images are just a part of Responsive websites...</p>
</body>
</html>
```



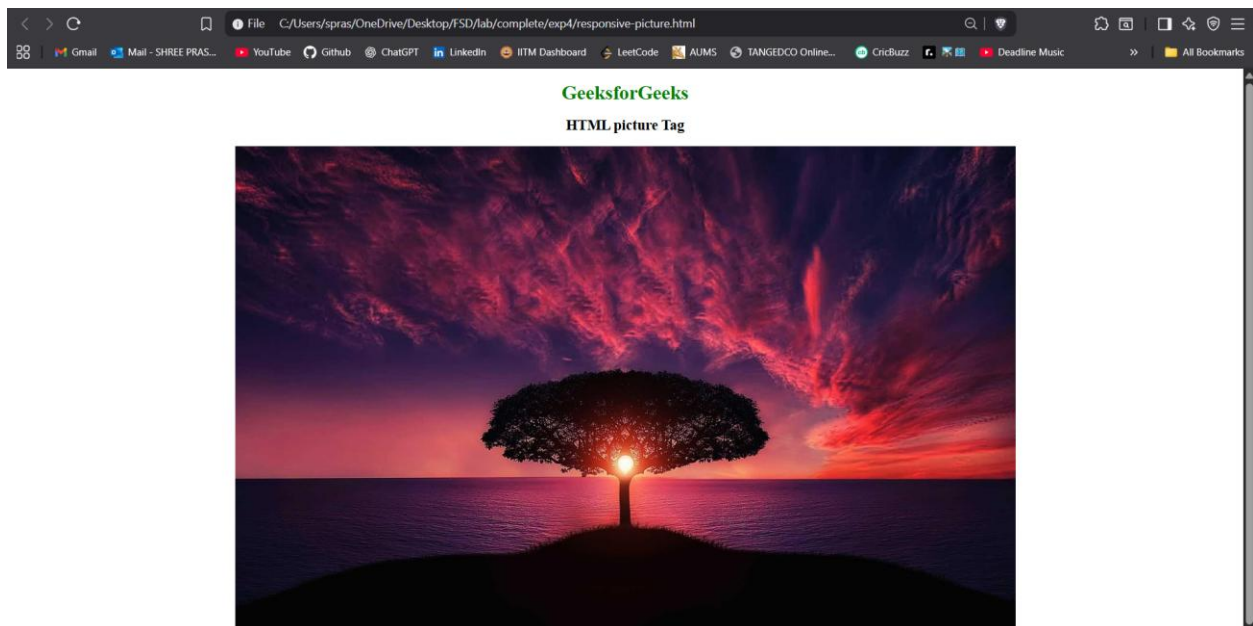
### Responsive Images

Responsive images are just a part of Responsive websites...

## 2.2 Using the <picture> Element

The <picture> element allows you to define different image sources for different browser window sizes.

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML Responsive Web Design</title>
</head>
<body style="text-align: center;">
  <h1 style="color: green;">GeeksforGeeks</h1>
  <h2>HTML picture Tag</h2>
  <picture>
    <source media="(max-width: 700px)"
Srcset="""E:\dOWNLOADS\amrita logo.png"">
    <source media="(max-width: 450px)"
srcset="https://images.pexels.com/photos/414612/pexels-photo-414612.jpeg">
    
  </picture>
</body>
</html>
```





## Ex 5: Using React in Visual Studio Code

React is a popular JavaScript library for building user interfaces. This exercise uses create-react-app to set up a new React project.

### Commands

```
npx create-react-app my-app
cd my-app
npm start
code .
npm install -g eslint
```

### Hello World Example

Let's update the sample application to "Hello World!". Modify src/index.js.

#### index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import reportWebVitals from './reportWebVitals';

function HelloWorld() {
  return <h1 className="greeting">Hello, world!</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <HelloWorld />
  </React.StrictMode>
);

reportWebVitals();
```

### Debugging React

Use the built-in JavaScript debugger in VS Code. You'll need to create a launch.json file.

## launch.json

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "msedge",
      "request": "launch",
      "name": "Launch Edge against localhost",
      "url": "http://localhost:3000",
      "webRoot": "${workspaceFolder}"
    }
  ]
}
```

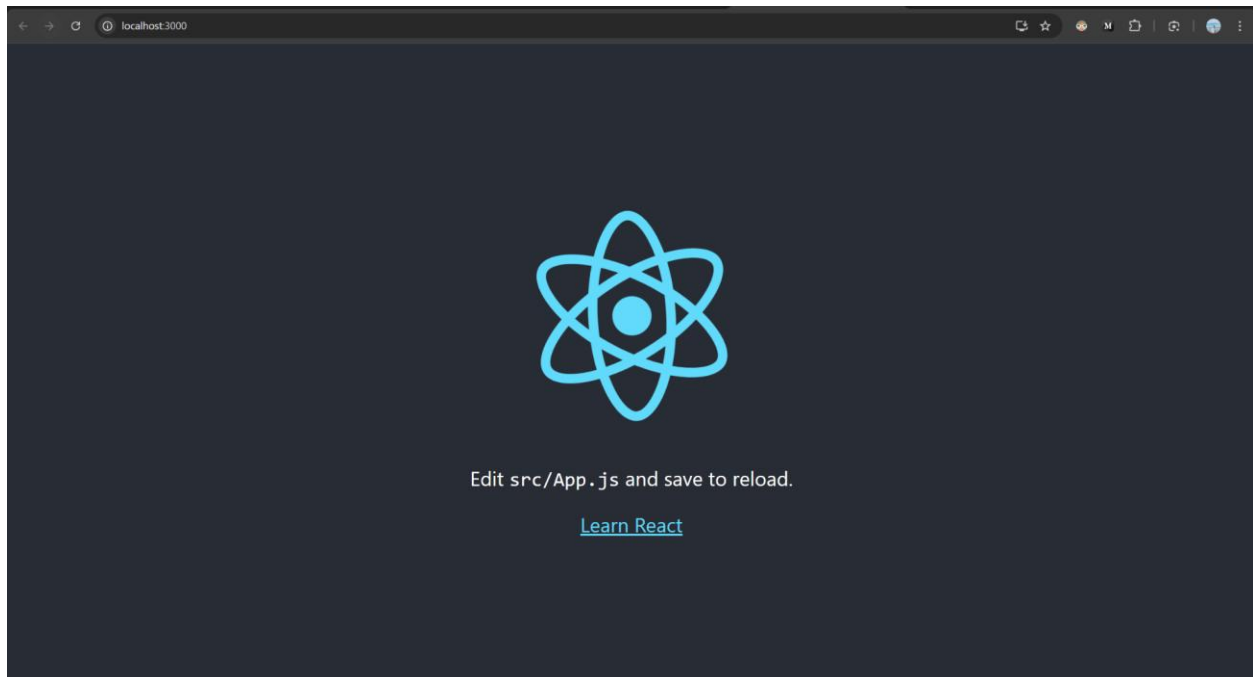
## Linting

ESLint is a popular linter for JavaScript. Install the VS Code extension and create a configuration file

### .eslintrc.js

```
module.exports = {
  env: {
    browser: true,
    es2020: true
  },
  extends: [
    'eslint:recommended',
    'plugin:react/recommended'
  ],
  parserOptions: {
    ecmaFeatures: {
      jsx: true
    },
    ecmaVersion: 11,
    sourceType: 'module'
  },
  plugins: [
```

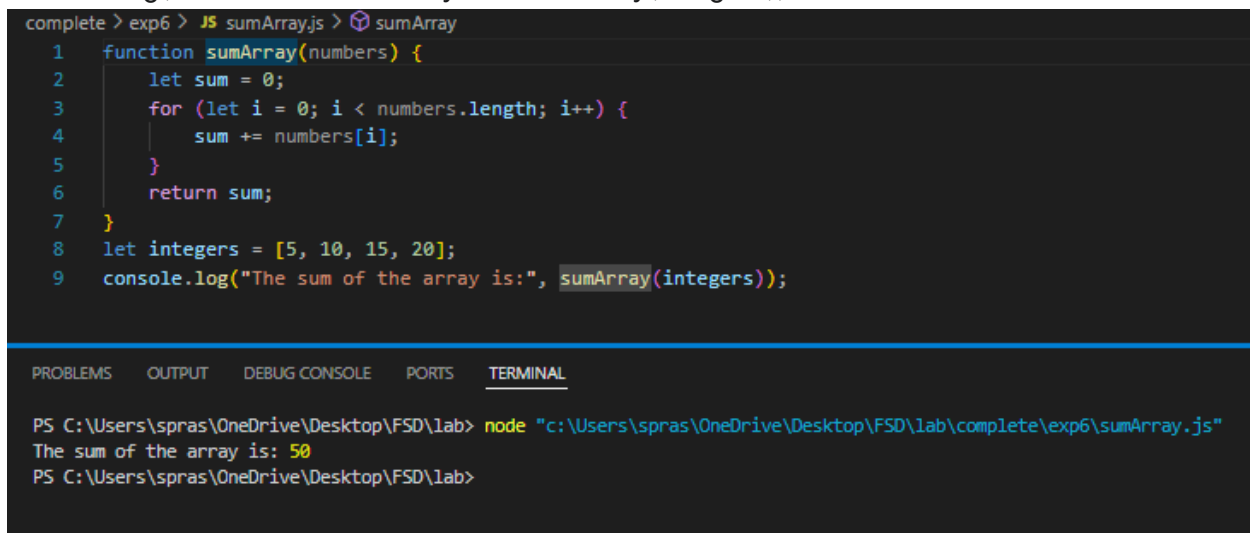
```
    'react'  
  ],  
  rules: {  
    "no-extra-semi": "error"  
  }  
};
```



## Ex 6: JavaScript

### 1. Write a JavaScript program to compute the sum of an array of integers

```
function sumArray(numbers) {  
    let sum = 0;  
    for (let i = 0; i < numbers.length; i++) {  
        sum += numbers[i];  
    }  
    return sum;  
}  
let integers = [5, 10, 15, 20];  
console.log("The sum of the array is:", sumArray(integers));
```



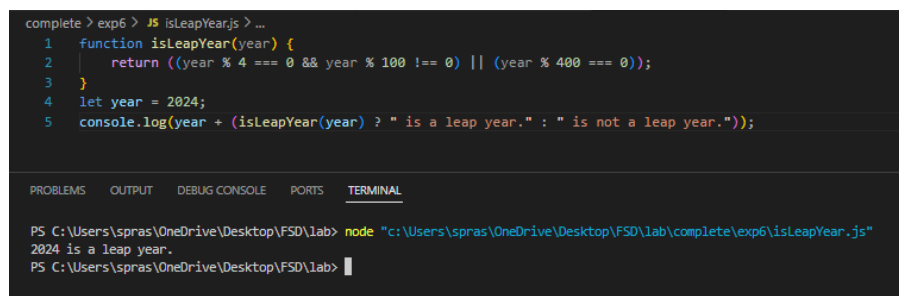
The screenshot shows a VS Code editor with a file named `sumArray.js` containing the following code:

```
1 function sumArray(numbers) {  
2     let sum = 0;  
3     for (let i = 0; i < numbers.length; i++) {  
4         sum += numbers[i];  
5     }  
6     return sum;  
7 }  
8 let integers = [5, 10, 15, 20];  
9 console.log("The sum of the array is:", sumArray(integers));
```

Below the editor, the **TERMINAL** tab is active, showing the command `node "c:\Users\spras\OneDrive\Desktop\FSD\lab\complete\exp6\sumArray.js"` and its output: `The sum of the array is: 50`.

### 2. Write a JavaScript program to determine whether a given year is a leap year

```
function isLeapYear(year) {  
    return ((year % 4 === 0 && year % 100 !== 0) || (year % 400 === 0));  
}  
let year = 2024;  
console.log(year + (isLeapYear(year) ? " is a leap year." : " is not a leap year."));
```



The screenshot shows a VS Code editor with a file named `isLeapYear.js` containing the following code:

```
1 function isLeapYear(year) {  
2     return ((year % 4 === 0 && year % 100 !== 0) || (year % 400 === 0));  
3 }  
4 let year = 2024;  
5 console.log(year + (isLeapYear(year) ? " is a leap year." : " is not a leap year."));
```

Below the editor, the **TERMINAL** tab is active, showing the command `node "c:\Users\spras\OneDrive\Desktop\FSD\lab\complete\exp6\isLeapYear.js"` and its output: `2024 is a leap year.`

### 3. Write a JavaScript function that checks whether a passed string is a palindrome

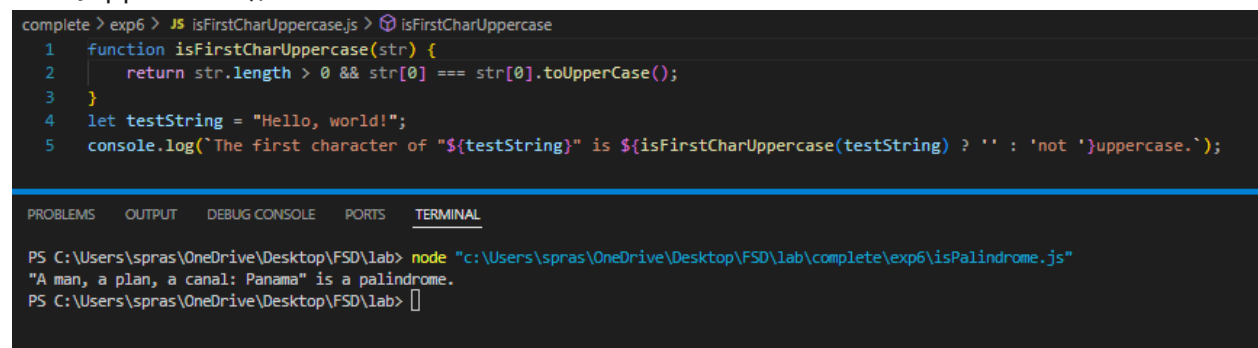
```
function isPalindrome(str) {  
    let cleanedStr = str.toLowerCase().replace(/[^a-z0-9]/g, '');  
    let reversedStr = cleanedStr.split('').reverse().join('');  
    return cleanedStr === reversedStr;  
}  
  
let testString = "A man, a plan, a canal: Panama";  
console.log(`"${testString}" is ${isPalindrome(testString) ? '' : 'not '}a palindrome.`);
```



The screenshot shows a VS Code editor with a file named `isPalindrome.js` open. The code in the editor matches the text above. Below the editor, the `TERMINAL` tab is active, showing the command `node "c:\Users\spras\OneDrive\Desktop\FSD\lab\complete\exp6\isPalindrome.js"` being executed. The output of the command is `"A man, a plan, a canal: Panama" is a palindrome.`

### 4. Write a JavaScript program to test if the first character of a string is uppercase

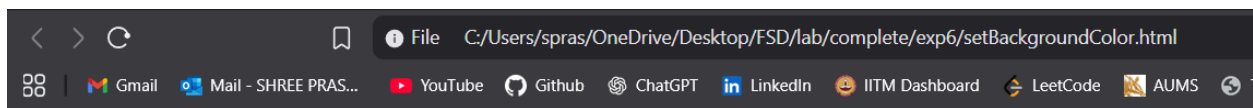
```
function isFirstCharUppercase(str) {  
    return str.length > 0 && str[0] === str[0].toUpperCase();  
}  
  
let testString = "Hello, world!";  
console.log(`The first character of "${testString}" is ${isFirstCharUppercase(testString) ? '' : 'not '}uppercase.`);
```



The screenshot shows a VS Code editor with a file named `isFirstCharUppercase.js` open. The code in the editor matches the text above. Below the editor, the `TERMINAL` tab is active, showing the command `node "c:\Users\spras\OneDrive\Desktop\FSD\lab\complete\exp6\isFirstCharUppercase.js"` being executed. The output of the command is `The first character of "Hello, world!" is uppercase.`

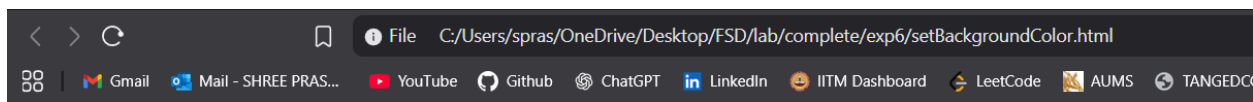
## 5. Write a JavaScript program to set the background colour of a paragraph

```
<!DOCTYPE html>
<html>
<head>
  <title>Set Background Color</title>
</head>
<body>
  <p id="myParagraph">This is a sample paragraph.</p>
  <button onclick="document.getElementById('myParagraph').style.backgroundColor =
'lightblue'">Change Color</button>
</body>
</html>
```



This is a sample paragraph.

Change Color

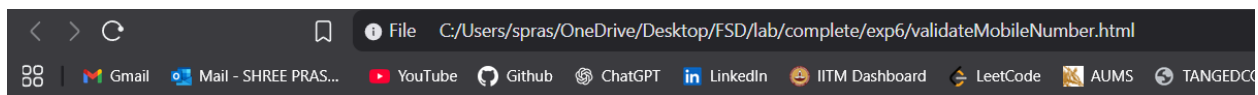


This is a sample paragraph.

Change Color

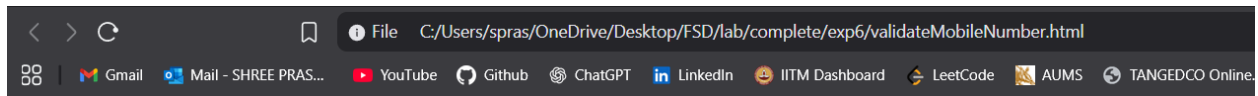
## 6. Write a JavaScript program to check if a given number is a mobile number using a form

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Mobile Number Validation</title>
</head>
<body>
  <h2>Mobile Number Validation Form</h2>
  <form id="mobileForm">
    <label for="mobileNumber">Enter Mobile Number:</label>
    <input type="text" id="mobileNumber" placeholder="Enter 10-digit mobile number"
required>
    <button type="button" onclick="validateMobileNumber()">Submit</button>
  </form>
  <p id="result"></p>
  <script>
    function validateMobileNumber() {
      const mobileNumber = document.getElementById("mobileNumber").value;
      const mobilePattern = /^[0-9]{10}$/;
      const resultP = document.getElementById("result");
      resultP.innerText = mobilePattern.test(mobileNumber) ? "Valid mobile number." :
"Invalid mobile number. Please enter a 10-digit number.";
    }
  </script>
</body>
</html>
```



### Mobile Number Validation Form

Enter Mobile Number:



## Mobile Number Validation Form

Enter Mobile Number:

Invalid mobile number. Please enter a 10-digit number.

### 7. Design a student registration form and apply validation using external JavaScript

#### index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Student Registration Form</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h2>Student Registration Form</h2>
  <form id="registrationForm" onsubmit="return validateForm()">
    <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="age">Age:</label>
    <input type="number" id="age" name="age" min="1" required>
    <label for="mobile">Mobile Number:</label>
    <input type="text" id="mobile" name="mobile" required placeholder="Enter 10-digit
number">
    <label for="address">Address:</label>
    <textarea id="address" name="address" required></textarea>
    <button type="submit">Register</button>
  </form>
  <div id="errorMessages"></div>
```



```
<script src="validation.js"></script>
</body>
</html>
```

### **style.css**

```
body { font-family: Arial, sans-serif; }
h2 { text-align: center; }
form { max-width: 400px; margin: auto; padding: 20px; border: 1px solid #ccc; border-radius: 5px; }
label { display: block; margin-top: 10px; }
input, textarea, button { width: 100%; padding: 8px; margin-top: 5px; box-sizing: border-box; }
button { background-color: #4CAF50; color: white; border: none; cursor: pointer; margin-top: 15px; }
button:hover { background-color: #45a049; }
#errorMessages { color: red; text-align: center; white-space: pre-line; margin-top: 10px;}
```

### **validation.js**

```
function validateForm() {
  const name = document.getElementById("name").value;
  const age = document.getElementById("age").value;
  const mobile = document.getElementById("mobile").value;
  const address = document.getElementById("address").value;
  let errors = [];

  if (!/^[A-Za-z\s]+$/.test(name)) errors.push("Name must contain only letters and spaces.");
  if (isNaN(age) || age < 1) errors.push("Please enter a valid age.");
  if (!/^[0-9]{10}$/.test(mobile)) errors.push("Mobile number must be exactly 10 digits.");
  if (address.trim() === "") errors.push("Address is required.");

  const errorContainer = document.getElementById("errorMessages");
  if (errors.length > 0) {
    errorContainer.innerText = errors.join("\n");
    return false;
  }
  errorContainer.innerText = "";
  alert("Registration Successful!");
  return true;
}
```

}

The screenshot shows a web browser window with the address bar displaying the file path: C:/Users/spras/OneDrive/Desktop/FSD/lab/complete/exp6/student-registration.html. The browser's taskbar at the top includes icons for YouTube, Github, ChatGPT, LinkedIn, IITM Dashboard, LeetCode, AUMS, TANGEDCO Online..., CricBuzz, and Deadline Music. The main content area features a form titled "Student Registration Form". The form contains the following fields: "Name:" with the value "Shree Prasad M", "Email:" with the value "sprasad2804@gmail.com", "Age:" with the value "21", "Mobile Number:" with the value "8610775211", and "Address:" with the value "G1, Gayathri Homes, Ashok Nagar". A green "Register" button is located at the bottom of the form.

This screenshot shows the same web browser window as the previous one, but with a dark overlay message box that reads "This page says Registration Successful!". The message box has an "OK" button. The registration form is visible in the background, showing the same data as before: Name: Shree Prasad M, Email: sprasad2804@gmail.com, Age: 21, Mobile Number: 8610775211, and Address: G1, Gayathri Homes, Ashok Nagar. The "Register" button is still present at the bottom of the form.

## 8. Design a simple Calculator using HTML, CSS, and JavaScript

index.html

```
<!DOCTYPE html>
```

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

```
<head>
```

```
  <meta charset="UTF-8">
```

```

<title>Simple Calculator</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="calculator">
    <input type="text" id="display" disabled>
    <div class="buttons">
      <div class="row">
        <button onclick="clearDisplay()">C</button>
        <button onclick="deleteLast()">DEL</button>
        <button class="operator" onclick="appendToDisplay('%')">%</button>
        <button class="operator" onclick="appendToDisplay('/')">/</button>
      </div>
      <div class="row">
        <button onclick="appendToDisplay('7')">7</button>
        <button onclick="appendToDisplay('8')">8</button>
        <button onclick="appendToDisplay('9')">9</button>
        <button class="operator" onclick="appendToDisplay('*')">*</button>
      </div>
      <div class="row">
        <button onclick="appendToDisplay('4')">4</button>
        <button onclick="appendToDisplay('5')">5</button>
        <button onclick="appendToDisplay('6')">6</button>
        <button class="operator" onclick="appendToDisplay('-')">-</button>
      </div>
      <div class="row">
        <button onclick="appendToDisplay('1')">1</button>
        <button onclick="appendToDisplay('2')">2</button>
        <button onclick="appendToDisplay('3')">3</button>
        <button class="operator" onclick="appendToDisplay('+')">+</button>
      </div>
    </div>
  </div>

```

```

    </div>
    <div class="row">
        <button onclick="appendToDisplay('0')">0</button>
        <button onclick="appendToDisplay('.')">.</button>
        <button class="operator" style="width: 98px;"
onclick="calculateResult()">=</button>
    </div>
</div>
</div>
</div>
<script src="script.js"></script>
</body>
</html>

```

### style.css

```

body { font-family: Arial, sans-serif; }
h2 { text-align: center; }
form { max-width: 400px; margin: auto; padding: 20px; border: 1px solid #ccc; border-radius:
5px; }
label { display: block; margin-top: 10px; }
input, textarea, button { width: 100%; padding: 8px; margin-top: 5px; box-sizing: border-box;
}
button { background-color: #4CAF50; color: white; border: none; cursor: pointer; margin-top:
15px; }
button:hover { background-color: #45a049; }
#errorMessages { color: red; text-align: center; white-space: pre-line; margin-top: 10px;}
/* Calculator Styles */
.calculator {
    width: 320px;
    margin: 40px auto;
    padding: 20px;
    background: #f7f7f7;
    border-radius: 12px;
    box-shadow: 0 2px 8px rgba(0,0,0,0.15);
}

```

```
}  
#display {  
    width: 100%;  
    height: 48px;  
    font-size: 1.5em;  
    margin-bottom: 16px;  
    text-align: right;  
    padding: 8px 12px;  
    border-radius: 6px;  
    border: 1px solid #ccc;  
    background: #fff;  
}  
.buttons {  
    display: flex;  
    flex-direction: column;  
    gap: 8px;  
}  
.row {  
    display: flex;  
    gap: 8px;  
}  
.row button {  
    flex: 1;  
    height: 48px;  
    font-size: 1.2em;  
    margin: 0;  
    border-radius: 6px;  
    background-color: #e0e0e0;  
    color: #333;  
    border: none;
```

```

        transition: background 0.2s;
    }
    .row button.operator {
        background-color: #4CAF50;
        color: #fff;
    }
    .row button:active {background-color: #bdbdbd;}
    .row button.operator:active {background-color: #388e3c;}
    .row button:last-child {flex: 1.2;}

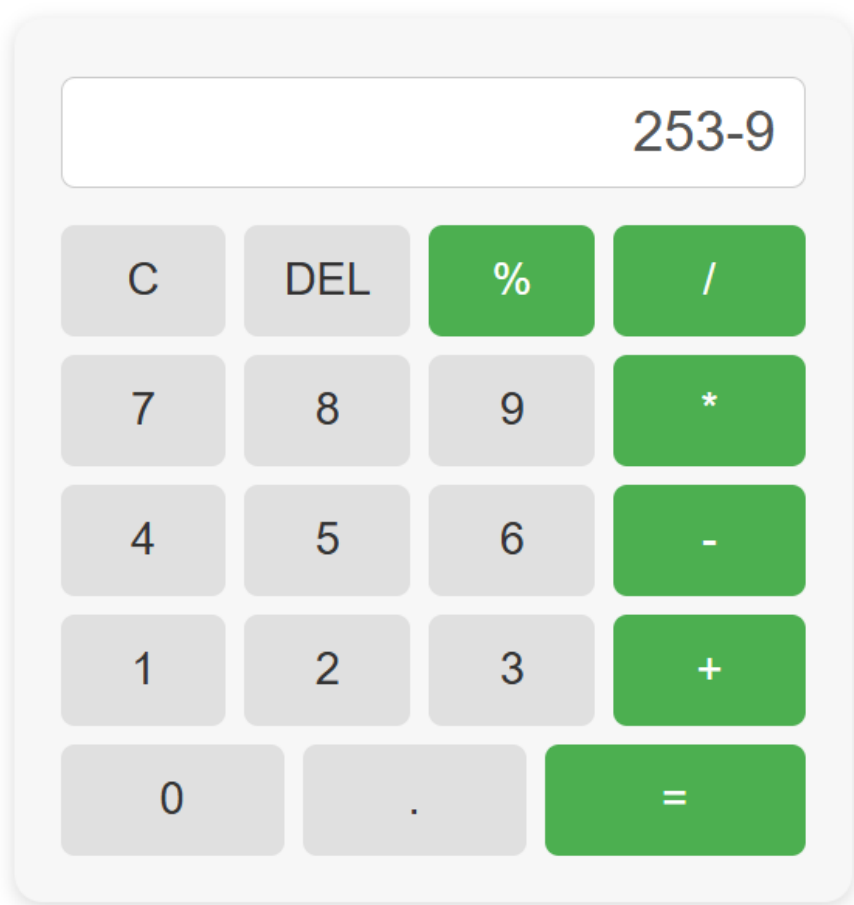
```

### **script.js**

```

        function appendToDisplay(value) {
            document.getElementById("display").value += value;
        }
        function clearDisplay() {
            document.getElementById("display").value = "";
        }
        function deleteLast() {
            let display = document.getElementById("display");
            display.value = display.value.slice(0, -1);
        }
        function calculateResult() {
            let display = document.getElementById("display");
            try {
                let sanitizedInput = display.value.replace(/[^-()\d/*+.]/g, "");
                display.value = eval(sanitizedInput);
            } catch (error) {
                display.value = "Error";
            }
        }

```



## Ex 7: Login Form using Node.js and MongoDB

### Introduction

This experiment demonstrates how to build a complete user authentication system from scratch using a popular technology stack: Node.js with the Express framework for the backend, MongoDB as the database, and Passport.js for handling the complexities of authentication. This is a very common pattern for building modern, secure web applications.

- **Node.js & Express:** We use Node.js as our server-side runtime environment, allowing us to write our backend in JavaScript. Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications, making it easier to handle routing, middleware, and requests.
- **MongoDB & Mongoose:** MongoDB is our database of choice. It's a NoSQL database, which means it stores data in flexible, JSON-like documents. This approach is often more intuitive for JavaScript developers. Mongoose is an Object Data Modeling (ODM) library that provides a straightforward, schema-based solution to model our application data, helping us define what a "User" looks like and how to interact with the database.
- **Passport.js:** Authentication is a critical but often complex part of any application. Passport.js is middleware for Node.js that makes it incredibly easy to implement authentication. It's flexible and modular, allowing us to use different "strategies" for authentication (e.g., username/password, Google, Facebook). In this experiment, we use passport-local for traditional username and password login.
- **EJS (Embedded JavaScript):** This is our templating engine. EJS allows us to write standard HTML and embed JavaScript directly into it. This is useful for dynamically generating content, such as displaying different links depending on whether a user is logged in or not.

### Commands

```
# Initialize a new Node.js project
npm init
```

```
# Install all required dependencies
npm i express ejs mongoose body-parser express-session passport passport-local passport-local-mongoose
```

```
# Create the necessary folder structure. This works on Linux/macOS and Windows.
mkdir model views
```

```
# Create the main application file
# For Linux/macOS:
touch app.js
```



# For Windows CMD:

```
type nul > app.js
```

# Create the schema file

# For Linux/macOS:

```
touch model/User.js
```

# For Windows CMD:

```
type nul > model\User.js
```

# Create the view files

# For Linux/macOS:

```
touch views/home.ejs views/login.ejs views/secret.ejs views/register.ejs
```

# For Windows CMD:

```
type nul > views\home.ejs & type nul > views\login.ejs & type nul > views\secret.ejs & type nul  
> views\register.ejs
```

# Ensure all packages are installed correctly based on package.json

```
npm i
```

### **package.json dependencies**

```
{  
  "dependencies": {  
    "body-parser": "^1.20.0",  
    "ejs": "^3.1.8",  
    "express": "^4.18.2",  
    "express-session": "^1.17.3",  
    "mongoose": "^6.9.1",  
    "passport": "^0.6.0",  
    "passport-local": "^1.0.0",  
    "passport-local-mongoose": "^7.1.2"  
  }  
}
```

### **app.js (The Core Server Logic)**

```
// Import all necessary modules  
const express = require("express"),  
      mongoose = require("mongoose"),  
      passport = require("passport"),
```

```

bodyParser = require("body-parser"),
LocalStrategy = require("passport-local"),
User = require("../model/User");

// Create the Express app
let app = express();
// Connect to the MongoDB database
mongoose.connect("mongodb://localhost/auth_demo_app");

// Set EJS as the view engine for rendering templates
app.set("view engine", "ejs");
// Use body-parser to parse form data
app.use(bodyParser.urlencoded({ extended: true }));
// Configure express-session for session management
app.use(require("express-session")({
  secret: "Rusty is the best and cutest dog in the world",
  resave: false,
  saveUninitialized: false
}));

// Initialize Passport and configure it for session support
app.use(passport.initialize());
app.use(passport.session());

// Use the local strategy for authentication with the User model
passport.use(new LocalStrategy(User.authenticate()));
// Serialize and deserialize user instances to and from the session.
passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());

// ===== ROUTES =====

// Home page route
app.get("/", (req, res) => res.render("home"));

// Secret page route, protected by the isLoggedIn middleware
app.get("/secret", isLoggedIn, (req, res) => res.render("secret"));

// Show the registration form
app.get("/register", (req, res) => res.render("register"));

```

```

// Handle user registration
app.post("/register", (req, res) => {
  User.register(new User({ username: req.body.username }), req.body.password, (err, user)
=> {
    if (err) {
      console.log(err);
      return res.render("register");
    }
    // Log the user in after successful registration
    passport.authenticate("local")(req, res, () => res.redirect("/secret"));
  });
});

// Show the login form
app.get("/login", (req, res) => res.render("login"));

// Handle user login
app.post("/login", passport.authenticate("local", {
  successRedirect: "/secret",
  failureRedirect: "/login"
}), (req, res) => {});

// Handle user logout
app.get("/logout", (req, res, next) => {
  req.logout((err) => {
    if (err) { return next(err); }
    res.redirect('/');
  });
});

// Middleware to check if the user is authenticated
function isLoggedIn(req, res, next) {
  if (req.isAuthenticated()) return next();
  res.redirect("/login");
}

// Start the server
let port = process.env.PORT || 3000;
app.listen(port, () => console.log("Server Has Started!"));

```

### **model/User.js (Database Schema)**

```
// Import mongoose
const mongoose = require('mongoose');
const passportLocalMongoose = require('passport-local-mongoose');

// Define the User schema
const UserSchema = new mongoose.Schema({});

// Plugin passport-local-mongoose to add username, hash and salt fields to store the
// username, the hashed password and the salt value.
UserSchema.plugin(passportLocalMongoose);

// Export the User model
module.exports = mongoose.model('User', UserSchema);
```

### **views/home.ejs (Home Page Template)**

```
<h1>This is home page</h1>
<ul>
  <li><a href="/register">Sign up!!</a></li>
  <li><a href="/login">Login</a></li>
  <li><a href="/logout">Logout</a></li>
</ul>
```

### **views/login.ejs (Login Form Template)**

```
<h1>Login</h1>
<form action="/login" method="POST">
  <input type="text" name="username" placeholder="username">
  <input type="password" name="password" placeholder="password">
  <button>login</button>
</form>
```

### **views/register.ejs (Registration Form Template)**

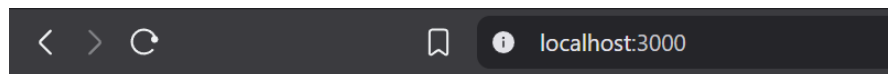
```
<h1> Sign up form </h1>
<form action="/register" method="POST">
  <input type="text" name="username" placeholder="username">
```

```
<input type="password" name="password" placeholder="password">
<button>Submit</button>
</form>
```

### views/secret.ejs (Protected Content Template)

```
<h1>This is the secret page</h1>
<p>You can only see this page if you are logged in!</p>

<p><a href="/logout">Logout</a></p>
```



## This is home page

- [Sign up!!](#)
- [Login](#)
- [Logout](#)



## Sign up form

<input type="text" value="admin"/>	<input type="password"/>	<input type="button" value="Submit"/>
------------------------------------	--------------------------	---------------------------------------

---

## This is the secret page

You can only see this page if you are logged in!



[Logout](#)

---

## Login

## Ex 8: Basic Login System with Node.js, Express, and MySQL

### Introduction

This experiment builds a fundamental user login system using a combination of Node.js for the server-side logic, Express as the web framework, and MySQL for the database. The system is composed of several key files:

- **SQL Setup:** This script is the first step. It creates the nodelogin database and the accounts table within it. This table is designed to store user credentials (username, password, email) and includes a sample user for testing purposes.
- **login.js (Backend Server):** This is the core of the application. It's a Node.js script that uses Express to create a web server. It establishes a connection to the MySQL database, sets up session management to track logged-in users, and defines the routes for handling web requests. It serves the HTML login page, processes the submitted form data for authentication, and displays a welcome message upon successful login.
- **login.html (Frontend Form):** This file provides the user interface. It's a standard HTML page containing a form with fields for username and password. The form is set up to send a POST request to the /auth route defined in login.js.
- **static/style.css (Styling):** This CSS file is used to style the login.html page, making the form more visually organized and user-friendly.

### Why Node.js over PHP?

Node.js is a powerful server environment using JavaScript. Its package manager (NPM) has a vast ecosystem, making it a popular choice for modern web applications.

### Setup & File Structure

#### SQL Setup

```
CREATE DATABASE IF NOT EXISTS `nodelogin` DEFAULT CHARACTER SET utf8 COLLATE
utf8_general_ci;
USE `nodelogin`;
CREATE TABLE IF NOT EXISTS `accounts` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(50) NOT NULL,
  `password` varchar(255) NOT NULL,
  `email` varchar(100) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
INSERT INTO `accounts` (`id`, `username`, `password`, `email`) VALUES (1, 'test', 'test',
'test@test.com');
```

## static/style.css

```
* { box-sizing: border-box; font-family: sans-serif; font-size: 16px; }
body { background-color: #435165; }
.login { width: 400px; background-color: #ffffff; box-shadow: 0 0 9px 0 rgba(0, 0, 0, 0.3);
margin: 100px auto; }
.login h1 { text-align: center; color: #5b6574; font-size: 24px; padding: 20px 0; border-
bottom: 1px solid #dee0e4; }
.login form { display: flex; flex-wrap: wrap; justify-content: center; padding-top: 20px; }
.login form label { display: flex; justify-content: center; align-items: center; width: 50px;
height: 50px; background-color: #3274d6; color: #ffffff; }
.login form input[type="password"], .login form input[type="text"] { width: 310px; height:
50px; border: 1px solid #dee0e4; margin-bottom: 20px; padding: 0 15px; }
.login form input[type="submit"] { width: 100%; padding: 15px; margin-top: 20px;
background-color: #3274d6; border: 0; cursor: pointer; font-weight: bold; color: #ffffff;
transition: background-color 0.2s; }
.login form input[type="submit"]:hover { background-color: #2868c7; }
```

## login.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Login</title>
  <link rel="stylesheet"
href="[https://use.fontawesome.com/releases/v5.7.1/css/all.css](https://use.fontawesome.co
m/releases/v5.7.1/css/all.css)">
  <link href="/style.css" rel="stylesheet" type="text/css">
</head>
<body>
  <div class="login">
    <h1>Login</h1>
    <form action="/auth" method="post">
      <label for="username"><i class="fas fa-user"></i></label>
      <input type="text" name="username" placeholder="Username" id="username"
required>
      <label for="password"><i class="fas fa-lock"></i></label>
      <input type="password" name="password" placeholder="Password" id="password"
required>
```



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

## login.js

```
const mysql = require('mysql');
const express = require('express');
const session = require('express-session');
const path = require('path');

const connection = mysql.createConnection({
  host: 'localhost', user: 'root', password: '', database: 'nodelogin'
});

const app = express();
app.use(session({ secret: 'secret', resave: true, saveUninitialized: true }));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(express.static(path.join(__dirname, 'static')));

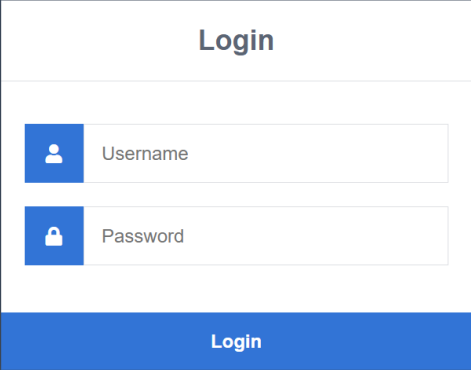
app.get('/', (request, response) => response.sendFile(path.join(__dirname + '/login.html')));

app.post('/auth', (request, response) => {
  let { username, password } = request.body;
  if (username && password) {
    connection.query('SELECT * FROM accounts WHERE username = ? AND password = ?',
[username, password], (error, results) => {
      if (error) throw error;
      if (results.length > 0) {
        request.session.loggedin = true;
        request.session.username = username;
        response.redirect('/home');
      } else {
        response.send('Incorrect Username and/or Password!');
      }
      response.end();
    });
  } else {
  }
```

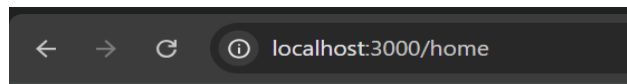
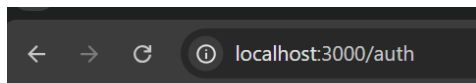
```
    response.send('Please enter Username and Password!');
    response.end();
  }
});

app.get('/home', (request, response) => {
  if (request.session.loggedin) {
    response.send(' Welcome back, ${request.session.username}!');
  } else {
    response.send('Please login to view this page!');
  }
  response.end();
});

app.listen(3000, () => console.log('Server listening on http://localhost:3000'));
```



The image shows a login form titled "Login" centered on a dark blue background. The form has a white background and contains two input fields: "Username" with a blue user icon and "Password" with a blue lock icon. Below the fields is a blue button labeled "Login".



## Ex 9: Live Editable Table with jQuery, PHP and MySQL

### Introduction

Live HTML table edit, or inline table edit, is a user-friendly feature that enables users to edit HTML table values directly by clicking on table cells. This experiment uses the jQuery Tabledit plugin to achieve this functionality. The setup involves three main files:

- **index.php (Frontend & Data Display):** This is the main page that users will see. It connects to the MySQL database, fetches the developer records, and displays them in an HTML table. It is also responsible for including the jQuery library and the Tabledit plugin, along with our custom JavaScript file.
- **custom\_table\_edit.js (Client-Side Logic):** This file contains the JavaScript code needed to activate the Tabledit plugin on our HTML table. It defines which columns are editable and tells the plugin where to send the updated data (the URL of live\_edit.php) when a change is made.
- **live\_edit.php (Backend Update Handler):** This PHP script acts as the backend endpoint. When a user edits a cell and saves it, the Tabledit plugin sends an AJAX request to this file. This script receives the updated data, validates it, and executes a SQL UPDATE query to save the changes permanently in the MySQL database.

### SQL Setup

```
CREATE TABLE `developers` (  
  `id` int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  `name` varchar(255) NOT NULL,  
  `skills` varchar(255) NOT NULL,  
  `address` varchar(255) NOT NULL,  
  `gender` varchar(255) NOT NULL,  
  `designation` varchar(255) NOT NULL,  
  `age` int(11) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

### index.php

```
<?php $conn = mysqli_connect("localhost", "root", "", "your_db_name"); ?>  
<!DOCTYPE html>  
<html>  
<head>  
  <title>Live Editable Table</title>  
  <script  
src="[https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js](https://ajax.googleapis.  
com/ajax/libs/jquery/2.1.3/jquery.min.js)"></script>
```

```

    <script type="text/javascript" src="dist/jquery.tableedit.js"></script>
</head>
<body>
    <table id="data_table" class="table table-striped">
        <thead>

<tr><th>Id</th><th>Name</th><th>Gender</th><th>Age</th><th>Designation</th><th>Address</th></tr>
        </thead>
        <tbody>
            <?php
                $sql = "SELECT id, name, gender, address, designation, age FROM developers LIMIT
10";
                $result = mysqli_query($conn, $sql);
                while( $dev = mysqli_fetch_assoc($result) ) { ?>
                    <tr id="<?php echo $dev['id']; ?>">
                        <td><?php echo $dev['id']; ?></td>
                        <td><?php echo $dev['name']; ?></td>
                        <td><?php echo $dev['gender']; ?></td>
                        <td><?php echo $dev['age']; ?></td>
                        <td><?php echo $dev['designation']; ?></td>
                        <td><?php echo $dev['address']; ?></td>
                    </tr>
                    <?php } ?>
                </tbody>
            </table>
            <script type="text/javascript" src="custom_table_edit.js"></script>
        </body>
    </html>

```

### custom\_table\_edit.js

```

$(document).ready(function(){
    $('#data_table').Tableedit({
        url: 'live_edit.php',
        deleteButton: false,
        editButton: false,
        columns: {
            identifier: [0, 'id'],
            editable: [[1, 'name'], [2, 'gender'], [3, 'age'], [4, 'designation'], [5, 'address']]
        }
    });

```

```

    },
    hideIdentifier: true
  });
});

```

## live\_edit.php

```

<?php
$conn = mysqli_connect("localhost", "root", "", "your_db_name");
$input = filter_input_array(INPUT_POST);
if ($input['action'] == 'edit') {
    $update_fields = [];
    $allowed_fields = ['name', 'gender', 'address', 'age', 'designation'];
    foreach ($allowed_fields as $field) {
        if (isset($input[$field])) {
            $update_fields[] = "$field=" . mysqli_real_escape_string($conn, $input[$field]) . " ";
        }
    }
    if (!empty($update_fields) && isset($input['id'])) {
        $id = mysqli_real_escape_string($conn, $input['id']);
        $sql = "UPDATE developers SET " . implode(' ', $update_fields) . " WHERE id=$id";
        mysqli_query($conn, $sql);
    }
}
?>

```



## Employee Information

Id	Name	Gender	Age	Designation	Address
1	Shree Prasad	Male	21	Goldman Sachs	New York, USA
2	Guru Prasath	Male	21	Oracle	BLR, IND
3	Neelraj	Male	21	Oracle	BLR, IND
4	Shyam Prasath	Male	21	GE Healthcare	BLR, IND
5	Pradeep Raj	Male	21	Fidelity	Chennai, India

## Ex 10: AJAX Forms with jQuery

### Introduction

jQuery can be paired with form submission to handle validation. This provides users with immediate feedback on their input without a page reload. This experiment consists of three files that work together.

- **index.html (Frontend):** This file creates the structure of our web form using standard HTML. It includes input fields for the user to enter their name, email, and a superhero alias. For styling, it links to the Bootstrap CSS framework and for functionality, it includes the jQuery library and our custom form.js script.
- **process.php (Backend):** This is the server-side PHP script that receives the data submitted from the form. Its job is to perform validation – in this case, simply checking if the fields are empty. It then packages the results (either success or a list of errors) into a JSON object and sends it back to the browser.
- **form.js (Client-Side Logic):** This JavaScript file is the core of the AJAX functionality. It uses jQuery to intercept the form's normal submission process. Instead of reloading the page, it sends the form data to process.php in the background. It then waits for the JSON response and dynamically updates the HTML to show either a success message or specific error messages next to the invalid fields, providing a smooth user experience.

### process.php

```
<?php
header('Content-Type: application/json');
$errors = [];
$data = [];

if (empty($_POST['name'])) $errors['name'] = 'Name is required.';
if (empty($_POST['email'])) $errors['email'] = 'Email is required.';
if (empty($_POST['superheroAlias'])) $errors['superheroAlias'] = 'Superhero alias is required.';

if (!empty($errors)) {
    $data['success'] = false;
    $data['errors'] = $errors;
} else {
    $data['success'] = true;
    $data['message'] = 'Success!';
}

echo json_encode($data);
```

?>

## index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>jQuery Form Example</title>
  <link rel="stylesheet"
href="[https://netdna.bootstrapcdn.com/bootstrap/3.0.3/css/bootstrap.min.css](https://netd
na.bootstrapcdn.com/bootstrap/3.0.3/css/bootstrap.min.css)"/>
  <script
src="[https://ajax.googleapis.com/ajax/libs/jquery/2.0.3/jquery.min.js](https://ajax.googleapis.
com/ajax/libs/jquery/2.0.3/jquery.min.js)"></script>
  <script src="form.js"></script>
</head>
<body>
  <div class="col-sm-6 col-sm-offset-3">
    <h1>Processing an AJAX Form</h1>
    <form action="process.php" method="POST">
      <div id="name-group" class="form-group">
        <label for="name">Name</label>
        <input type="text" class="form-control" id="name" name="name" placeholder="Full
Name"/>
      </div>
      <div id="email-group" class="form-group">
        <label for="email">Email</label>
        <input type="text" class="form-control" id="email" name="email"
placeholder="email@example.com"/>
      </div>
      <div id="superhero-group" class="form-group">
        <label for="superheroAlias">Superhero Alias</label>
        <input type="text" class="form-control" id="superheroAlias" name="superheroAlias"
placeholder="Ant Man, Wonder Woman, etc."/>
      </div>
      <button type="submit" class="btn btn-success">Submit</button>
    </form>
  </div>
</body>
</html>
```

## form.js

```
$(document).ready(function () {
    $("form").submit(function (event) {
        event.preventDefault();
        $(".form-group").removeClass("has-error");
        $(".help-block").remove();

        var formData = {
            name: $("#name").val(),
            email: $("#email").val(),
            superheroAlias: $("#superheroAlias").val(),
        };

        $.ajax({
            type: "POST",
            url: "process.php",
            data: formData,
            dataType: "json",
            encode: true,
        }).done(function (data) {
            if (!data.success) {
                if (data.errors.name) {
                    $("#name-group").addClass("has-error").append('<div class="help-block">' +
data.errors.name + "</div>");
                }
                if (data.errors.email) {
                    $("#email-group").addClass("has-error").append('<div class="help-block">' +
data.errors.email + "</div>");
                }
                if (data.errors.superheroAlias) {
                    $("#superhero-group").addClass("has-error").append('<div class="help-block">'
+ data.errors.superheroAlias + "</div>");
                }
            } else {
                $("form").html('<div class="alert alert-success">' + data.message + "</div>");
            }
        }).fail(function () {
            $("form").html('<div class="alert alert-danger">Could not reach server, please try
again later.</div>');
        });
    });
});
```



```
});  
});
```

## Processing an AJAX Form

Name

Full Name

Name is required.

Email

sprasad2804@gmail.com

Superhero Alias

Prazzo

Submit

## Processing an AJAX Form

Name

Shree Prasad M

Email

sprasad2804@gmail.com

Superhero Alias

Ant Man, Wonder Woman, etc.

Superhero alias is required.

Submit

# Processing an AJAX Form

Name

Shree Prasad M

Email

email@example.com

Email is required.

Superhero Alias

Prazzo

Submit

## Ex 11: AJAX Database Operations with PHP and MySQL

### Description

This experiment demonstrates a fundamental dynamic web application feature: retrieving data from a server database without reloading the page. We will create a simple frontend with an input field where a user can enter a CGPA. Using JavaScript's XMLHttpRequest object (the core of AJAX), we will send this CGPA to a PHP script on the server. The PHP script will then connect to a MySQL database, execute a query to find students matching that CGPA, and return the formatted results as an HTML table. This response is then dynamically inserted into our webpage, providing a seamless user experience

### index.html (Client-Side)

```
<!DOCTYPE html>
<html>
<head>
  <title>AJAX DB Operations</title>
</head>
<body>
  <h1>AJAX example</h1>
  <h3>Ajax Database Operations</h3><hr><br>
  <div>
    <label for="cgpa_val">Enter the CGPA: </label>
    <input type="text" id="cgpa_val" name="stu_cgpa">
    <input type="button" onclick="ajax_fun()" value="Submit">
  </div>
  <br><hr>
  <div id="result-box">Result will display here</div>

  <script>
    function ajax_fun() {
      var str = document.getElementById('cgpa_val').value;
      if (str === "") {
        document.getElementById("result-box").innerHTML = "Please enter a valid CGPA.";
        return;
      }
      const ajax_Request = new XMLHttpRequest();
      ajax_Request.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
          document.getElementById("result-box").innerHTML = this.responseText;
        }
      }
    }
  </script>
</body>
</html>
```

```

    };
    ajax_Request.open("GET", "index.php?q=" + str, true);
    ajax_Request.send();
}
</script>
</body>
</html>

```

### index.php (Server-Side)

```

<?php
$dbhost = "localhost";
$dbuser = "root";
$dbpass = ""; // Default XAMPP password
$dbname = "sampledb";

$con = mysqli_connect($dbhost, $dbuser, $dbpass, $dbname);
if (!$con) exit('Could not connect: ' . mysqli_connect_error());

if (isset($_GET['q']) && !empty($_GET['q'])) {
    $cgpa = $_GET['q'];
    $stmt = $con->prepare("SELECT * FROM collegedb WHERE cgpa = ?");
    $stmt->bind_param("s", $cgpa);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows > 0) {
        echo "<table border='1' style='border-collapse:collapse;'>
            <tr><th>FirstName</th><th>LastName</th><th>Rollno</th><th>CGPA</th></tr>";
        while ($row = $result->fetch_assoc()) {
            echo "<tr>
                <td>" . htmlspecialchars($row['firstname']) . "</td>
                <td>" . htmlspecialchars($row['lastname']) . "</td>
                <td>" . htmlspecialchars($row['rollno']) . "</td>
                <td>" . htmlspecialchars($row['cgpa']) . "</td>
            </tr>";
        }
        echo "</table>";
    } else {
        echo "No records found for CGPA: " . htmlspecialchars($cgpa);
    }
}

```

```
}  
$stmt->close();  
} else {  
    echo "Please provide a valid CGPA.";  
}  
mysqli_close($con);  
?>
```

---

## AJAX example

### Ajax Database Operations

---

Enter the CGPA:

---

FirstName	LastName	Rollno	CGPA
Arun	Kumar	CS001	8.50
Sneha	Patel	CS004	8.50

---

## AJAX example

### Ajax Database Operations

---

Enter the CGPA:

---

No records found for CGPA: 8.4