**Guru Nanak Institutions Technical Campus (Autonomous)**



School of Engineering & Technology

**DEPARTMENT OF COMPUTER SCIENCE &ENGINEERING**

**Full Stack Development Lab**

Lab Manual [Subject Code: **22PE0CS16]**

For the Academic year 2024-25 III B.Tech Semester-II [CSE]

Guru Nanak Institutions Technical Campus (Autonomous)

Ibrahimpatnam, RR District – 501 506 (T.S.)

 **Department of Computer Science &Engineering**

**LAB MANUAL FOR THE ACADEMIC YEAR 2024-25**

**SUB : Full Stack Development Lab**

**SUB CODE : 22PE0CS16**

**SEMESTER : II**

**STREAM : CSE**

**DOCUMENT NO. : GNITC/CSE/ FSD/R22 DATE OF ISSUE : 01-03-2025**

**PREPARED : Ms.K.Sailaja**

**PROGRAMMER : Ms. Lavanya**

**VENUE : CSE-408 & 409**

**BLOCK : CSE**

**Authorized by Dr. Ravindra Ch**

**HOD-CSE**

**COURSE SYLLABUS**

|  |  |  |
| --- | --- | --- |
| **S.**  **No** | **Name of the experiment** | **Page**  **No** |
| 1 | Create an application to setup node JS environment and display “Hello World”. | 14-15 |
| 2 | Create a Node JS application for user login system. | 16-18 |
| 3 | Write a Node JS program to perform read, write and other operations on a file. | 19-21 |
| 4 | Write a Node JS program to read form data from query string and generate response using NodeJS | 22-23 |
| 5 | Create a food delivery website where users can order food from a particular restaurant listed in the website for handling http requests and responses using NodeJS. | 24-25 |
| 6 | Implement a program with basic commands on databases and collections using MongoDB. | 26-28 |
| 7 | Implement CRUD operations on the given dataset using MongoDB. | 29-31 |
| 8 | Perform Count, Limit, Sort, and Skip operations on the given collections using MongoDB. | 32-33 |
| 9 | Develop an angular JS form to apply CSS and Events | 34-36 |
| 10 | Develop a Job Registration form and validate it using angular JS. | 37-39 |
| 11 | Write an angular JS application to access JSON file data of an employee from a server using $http service. | 40-42 |
| 12 | Develop a web application to manage student information using Express and Angular JS | 43-46 |
| 13 | Write a program to create a simple calculator Application using React JS. | 47-50 |
| 14 | Write a program to create a voting application using React JS | 51-52 |
| 15 | Develop a leave management system for an organization where users can apply different types of leaves such as casual leave and medical leave. They also can view the available number of days using react application | 53-56 |
| 16 | Build a music store application using react components and provide routing among the web pages. | 57-58 |
| 17 | Create a react application for an online store which consist of registration, login, product information pages and implement routing to navigate through these pages. | 59-63 |

**VISION OF THE INSTITUTION: GNITC**

To be an internationally renowned institution in Engineering, Management, Pharmacy and related fields to produce scientists, engineers, entrepreneurs, leaders, academicians and thinkers of tomorrow with exemplary professional conduct and adherence to ethical values to serve for changing needs of industry and society.

**MISSION OF THE INSTITUTION: GNITC**

**M1:** Imbibe soft skills, technical skills, creatively and passion in students.

**M2:** Develop the faculty to reach the International standards.

**M3:** Maintain outcome based student centric teaching learning with high academic standards and quality that promotes the analytical thinking and independent judgement.

**M4:** Promote research, innovation, product development by collaborating with reputed industries & reputed universities in India and abroad. Offer collaborative industry programs in emerging areas

and instill the spirit of enterprising.

**M5:** To instill the ethical values in the faculty and students to serve the society.

**VISION OF THE DEPARTMENT: CSE**

To become a premier Computer Science & Engineering department by imparting high quality education, ethical values, provide creative environment for innovation and global career opportunities.

**MISSION OF THE DEPARTMENT: CSE**

**M1:** Nurture young individuals into knowledgeable, skillful and ethical professionals in their pursuit of Computer Science & Engineering.

**M2:** Foster the students through excellent teaching learning process and sustain high performance through innovation.

**M3:** Provide high quality soft skills and advanced industry specific technical trainings to meet global career opportunities.

**Course Information Sheet**

|  |  |
| --- | --- |
| Programme: **COMPUTER SCIENCE & ENGINEERING** | Degree: **B. Tech** |
| Course: Full Stack Development Lab | Semester: **II** CREDITS: **1** |
| Course Code**: 22PE0CS16**  Regulation: **R22** | Course Type: **CORE** |
| Course Area/Domain: **C** | Contact Hours/ Week: **3** |

**PROGRAMME EDUCATIONAL OBJECTIVES (PEOs):**

**PEO 1:** Graduates shall have the ability to apply knowledge across the disciplines and in emerging areas of Computer Science & Engineering for higher studies, research, employability, product development and handle the realistic problems.

**PEO 2:** Graduates shall have good communication skills, possess ethical values, sense of responsibility to serve the society, and protect the environment.

**PEO 3:** Graduates shall possess academic excellence with innovative insight, managerial skills,

leadership qualities, knowledge of contemporary issues and understand the need for lifelong

learning for a successful professional career.

**PROGRAMME OUTCOMES (POs): [Department of Computer Science & Engineering]**

1. **Engineering knowledge**: Apply the knowledge of mathematics, science,engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**PROGRAMME SPECIFIC OUTCOMES (PSOs):**

PSO 1: Ability to apply software engineering skills to design, develop and debug optimized solutions for all real world problems.

PSO 2: Ascertain knowledge in diverse areas of Computer Science and experience an environment conducive in novel skills for successful career, entrepreneurship and higher studies.

1. **Laboratory Program Schedule**

|  |  |  |
| --- | --- | --- |
| **S.**  **No** | **Name of the experiment** | **Number of Classes** |
| 1 | Create an application to setup node JS environment and display “Hello World”. | 1 |
| 2 | Create a Node JS application for user login system. | 1 |
| 3 | Write a Node JS program to perform read, write and other operations on a file. | 1 |
| 4 | Write a Node JS program to read form data from query string and generate response using NodeJS | 1 |
| 5 | Create a food delivery website where users can order food from a particular restaurant listed in the website for handling http requests and responses using NodeJS. | 1 |
| 6 | Implement a program with basic commands on databases and collections using MongoDB. | 1 |
| 7 | Implement CRUD operations on the given dataset using MongoDB. | 1 |
| 8 | Perform Count, Limit, Sort, and Skip operations on the given collections using MongoDB. | 1 |
| 9 | Develop an angular JS form to apply CSS and Events | 1 |
| 10 | Develop a Job Registration form and validate it using angular JS. | 1 |
| 11 | Write an angular JS application to access JSON file data of an employee from a server using $http service. | 1 |
| 12 | Develop a web application to manage student information using Express and Angular JS | 1 |
| 13 | Write a program to create a simple calculator Application using React JS. | 1 |
| 14 | Write a program to create a voting application using React JS | 1 |
| 15 | Develop a leave management system for an organization where users can apply different types of leaves such as casual leave and medical leave. They also can view the available number of days using react application | 1 |
| 16 | Build a music store application using react components and provide routing among the web pages. | 1 |
| 17 | Create a react application for an online store which consist of registration, login, product information pages and implement routing to navigate through these pages. | 1 |

1. **Text Books / Reference Books / Web References**
2. **Web Sources:**

|  |  |
| --- | --- |
| **T** | Brad Dayley. Brendan Dayley, Caleb *Doy\eY .* Node.js, MongoDB and Angular Web Development, 2ndEdition, Addison-Wesley, 2019. |
| **T** | Mark Tielens Thomas, React in Action, !" Edition, lv\anning Publications. |
| **R1** | Vasan Subromanian, Pro MERN Stack, Full Stack Web App Development with Mongo, Express, React, and Node, 2"^ Edition, Apress, 2019. |
| **R2** | Chris Northwood. The Full Stack Developer: Your Essential Guide to the Everyday SkillsExpected of a Modern Full Stack Web Developer', 1" edition, Apress, 2018. |
| **R3** | Kirupa Chinnathambi, Learning React: A Hands-On Guide to Building Web Applications UsingReact and Redux, 2n^edition, Addison-Wesley Professional. 20\ 8. |

1. [**https://developer.mozilla.org/**](https://developer.mozilla.org/)
2. [**https://www.jhipster.tech/**](https://www.jhipster.tech/)
3. [**https://www.codecademy.com/**](https://www.codecademy.com/)
4. **Course Pre-Requisites:**

|  |  |  |  |
| --- | --- | --- | --- |
| C.CODE | COURSE NAME | DESCRIPTION | SEM |
| 21PC0CS21 | Web Technologies | Web Technology refers to computers communicates with each other using markup languages and multimedia packages. | III/I |
| 22PC0CS06 | Object Oriented Programming through Java | Object Oriented Programming through Java refers to applets, java scripts, AWTs | II/I |

1. **Course Objective:**

Introduce fast, efficient, interactive and scalable web applications using run time environment

provided by the full stack components.

**Course Outcomes (COs) :**

CO 1: Design flexible and responsive Web applications using Node JS, React, Express and Angular.

CO 2: Perform CRUD operations with MongoDB on huge amount of data.

CO 3: Develop real time applications using react components.

CO 4: Use various full stack modules to handle http requests and responses.

CO5: Implement authentication and authorization mechanisms to enhance the security of web

applications.

**Mapping of COs with POs and PSOs**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Course Outcomes (COs)** | **PO 1** | **PO 2** | **PO 3** | **PO 4** | **PO 5** | **PO 6** | **PO 7** | **PO 8** | **PO 9** | **PO 10** | **PO 1 1** | **PO 1 2** | **PSO 1** | **PSO 2** | **PSO 3** |
| **CO1** | √ | √ | √ | - | **√** | - | - | - | - | √ | √ | √ | √ | √ | - |
| **CO2** | √ | √ | √ | - | √ | - | - | - | - | - | - | √ | √ | √ | - |
| **CO3** | √ | √ | √ | √ | √ | - | - | - | - | - | - | √ | √ | √ | - |
| **CO4** | √ | √ | √ | - | √ | - |  | - | - | - | √ | √ | √ | √ | - |
| **CO5** | √ | √ | √ | - | √ | √ | √ | √ | - | - | √ | √ | √ | √ | √ |

1. **Topics Beyond Syllabus/Advanced Topics/Design:**

|  |  |
| --- | --- |
| 1 | **Microservices Architecture**: Design and implementation of scalable, independent services using Node.js, including service discovery, API gateways, and event-driven architecture. |
| 2 | **Cloud Computing and Serverless Architecture**: Understanding and deploying applications on cloud platforms like AWS or Azure, and leveraging serverless computing with AWS Lambda or Google Cloud  Functions. |
| 3. | **CI/CD and DevOps Practices**: Implementing Continuous Integration and Continuous Deployment pipelines using tools like Jenkins, GitLab CI, and Docker for efficient software delivery and automation. |

1. **Delivery / Instructional Methodologies:**

|  |  |  |  |
| --- | --- | --- | --- |
| **√** PPT & TALK | **√** STUD. RECORD | **√** WEB RESOURCES | * VIDEO LECTURES |
| * LAB INTERNAL | **√** STUD. PRACTICE | * LAB EXTERNAL | * OBSERVATION |

**Lab Manual(Annexure)**

**List of Lab Exercises:**

|  |  |  |
| --- | --- | --- |
| **S.**  **No** | **Name of the experiment** | **Page**  **No** |
| 1 | Create an application to setup node JS environment and display “Hello World”. | 14-15 |
| 2 | Create a Node JS application for user login system. | 16-18 |
| 3 | Write a Node JS program to perform read, write and other operations on a file. | 19-21 |
| 4 | Write a Node JS program to read form data from query string and generate response using NodeJS | 22-23 |
| 5 | Create a food delivery website where users can order food from a particular restaurant listed in the website for handling http requests and responses using NodeJS. | 24-25 |
| 6 | Implement a program with basic commands on databases and collections using MongoDB. | 26-28 |
| 7 | Implement CRUD operations on the given dataset using MongoDB. | 29-31 |
| 8 | Perform Count, Limit, Sort, and Skip operations on the given collections using MongoDB. | 32-33 |
| 9 | Develop an angular JS form to apply CSS and Events | 34-36 |
| 10 | Develop a Job Registration form and validate it using angular JS. | 37-39 |
| 11 | Write an angular JS application to access JSON file data of an employee from a server using $http service. | 40-42 |
| 12 | Develop a web application to manage student information using Express and Angular JS | 43-46 |
| 13 | Write a program to create a simple calculator Application using React JS. | 47-50 |
| 14 | Write a program to create a voting application using React JS | 51-52 |
| 15 | Develop a leave management system for an organization where users can apply different types of leaves such as casual leave and medical leave. They also can view the available number of days using react application | 53-56 |
| 16 | Build a music store application using react components and provide routing among the web pages. | 57-58 |
| 17 | Create a react application for an online store which consist of registration, login, product information pages and implement routing to navigate through these pages. | 59-63 |

1. **INTRODUCTION TO FULL STACK DEVELOPMENT LAB**

The Full Stack Development Lab is equipped with 65 systems (Acer) with the following configurations:

1. **Hardware:** Intel® CORE™ i3-3240 CPU @ 3.40GHz, 4GB RAM.
2. **Software:** Node.js, MongoDB, React, Angular, Express, and other necessary tools.
3. **Operating Systems:** Dual Boot (Windows XP and Linux) for flexibility in programming environments.
4. **Database Access:** Each student has a separate login for MongoDB access, ensuring data security and privacy.
5. Explanation on today’s experiment by the concerned faculty using OHP/PPT/white Board

coveringthe following aspects: 60 mins.

* 1. Name of the experiment/Aim
  2. Software/Hardware required
  3. Description about the program
  4. Node js Program code

1. Writing of Node js programs by the student 30mins.
2. Compiling and execution of the program 90mins.

**2. Writing of the experiment in the Observation Book**:

The students will write the today’s experiment in the Observation book as per the followingformat:

1. Name of the experiment/Aim
2. Software/Hardware required
3. Source Progam
4. Results for the written code
5. Viva-Voce Questions and Answers
6. Errors observed (if any) during compilation/execution
7. Signature of the Faculty

**1. Guide Lines to Students in Lab**

Students are advised to maintain discipline and follow the guidelines given below:

* Keep all your bags in the racks and carry the observation book and record book.
* Mobile phones/pen drives/ CDs are not allowed in the labs.
* Maintain proper dress code along with ID Card
* Occupy the computers allotted to you and maintain the discipline.
* Student must submit the record with the last week experiment details and observation book with the brief of the present experiment.
* Read the write up of the experiment given in the manual.
* Students must use the equipment with care. Any damage is caused student is punishable
* After completion of every experiment, the observation notes to be shown to the lab in - charge and after correction the record must be updated and submit to the lab in charge for correction.
* Lab marks are given on Continuous Evaluation Basis as per GNITC(A)guidelines
* If any student is absent for any lab, they need to be complete the same experiment in the free time before attending next lab session.

Steps to perform experiments in the lab by the student

Step1: Students have to write the Date, aim, Software and Hardware requirements for the scheduled experiment in the observation book.

Step2: Students have to listen and understand the experiment explained by the faculty and note down the important points in the observation book.

Step3: Students need to write procedure/algorithm in the observation book.

Step4: Analyze and Develop/implement the logic of the program by the student in respective platform

Step5: After approval of logic of the experiment by the faculty then the experiment has to be executed on the system.

Step6: After successful execution, the results have to be recorded in the observation book and shown to the lab in charge faculty..

Step7: Students need to attend the Viva-Voce on that experiment and write the same in the observation book.

Step8: Update the completed experiment in the record and submit to the concerned faculty in- charge.

Instructions to maintain the record

* Before staring of the first lab session students must buy the record book and bring the

same to the lab.

* Regularly (Weekly) update the record after completion of the experiment and get it corrected with concerned lab in-charge for continuous evaluation.
* In case the record is lost, inform on the same day to the faculty in charge and submit the

new record within 2 days for correction.

* If record is not submitted in time or record is not written properly, the record evaluation marks (5M) will be reduced accordingly.
* **Awarding the marks for day to day evaluation:**

Total marks for day to day evaluation is 15 Marks as per JNTUH.These 15 Marks are distributed as:

|  |  |
| --- | --- |
| Record | 5 Marks |
| Exp setup/program written  and execution | 5 Marks |
| Result and Viva-Voce | 5 Marks |

**Allocation of Marks for Lab Internal**

Total marks for lab internal are 40 Marks as per GNITC (AUTONOMOUS).

These 40 Marks are distributed as:

**Average of day to day evaluation marks: 10 Marks Lab Mid exam: 10 Marks**

**Viva Marks: 10 Marks Additional lab project: 10 Marks**

**Allocation of Marks for Lab External**

Total marks for lab External are 60 Marks as per GNITC AUTONOMOUS).

These 60 Marks are distributed as:

**Program Written: 20 Marks**

**Program Execution and Result: 20 Marks Viva-Voce: 10 Marks**

**Record: 10 Marks**

**List of Lab Exercises:**

**7. Content of Lab Experimentt**

**Week-1**

**Create an application to setup node JS environment and display “Hello Worl**d”.

**Aim:** To Write a program Create an application to setup node JS environment and display “Hello World”.

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code

**Source Program :**

1. Install Node.js from [nodejs.org](https://nodejs.org/).
2. Create a new folder for your project.
3. Initialize a Node.js project using npm init -y.
4. Create a file named app.js.
5. Write the code to display "Hello World".

// app.js

const http = require('http');

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

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

res.end('Hello World\n');

});

server.listen(3000, () => {

console.log('Server is running on http://localhost:3000');

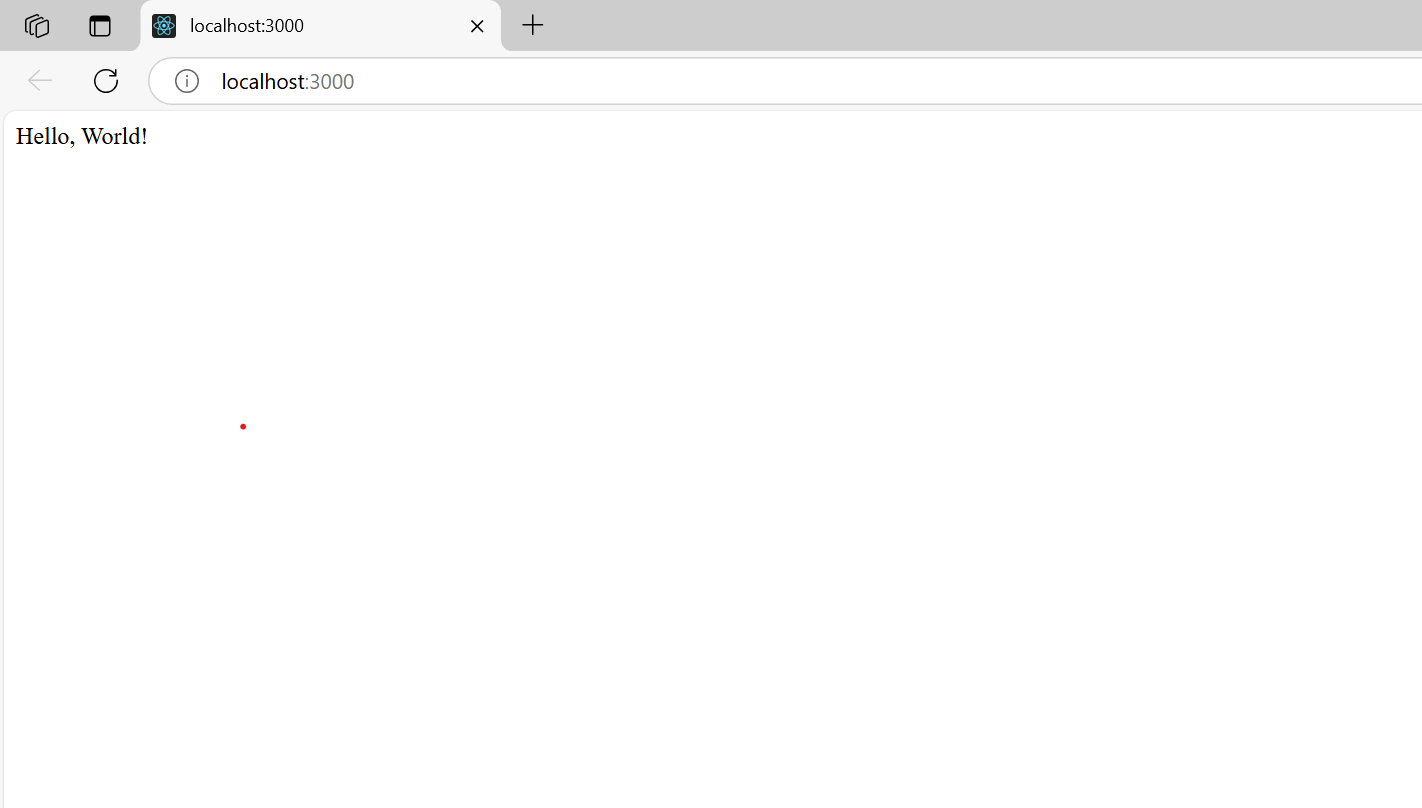
});

**Run the application:**

node app.js

Visit http://localhost:3000 in your browser to see "Hello World".

**Sample Output:**

****

**Viva Questions:**

1.Define Nodejs?

2.Define Express?

3.Define NPM?

4.How do you check the installed version of Node.js?

5.What is the purpose of package.json?

**Week-2**

**Create a Node JS application for user login system**

**Aim:** To Write a program Create a Node JS application for user login system

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code

**Steps:**

**1. Use Express.js for routing.**

**2. Create a simple login form.**

**3. Validate user credentials.**

const express=require('express');

        const path=require('path');

       const app=express();

       const port=process.env.PORT||3000;

       app.set('view engine','ejs');

       //load static assets

      app.use('/static',express.static(path.join(\_\_dirname,'public')))

      app.use('/assets',express.static(path.join(\_\_dirname,'public/assets')))

      //home route

      app.get('/',(req,res)=>{

    res.render('base',{title:"Login System"});

    })

    app.listen(port,()=>{console.log("Listening to the server on http://localhost:3000")});

<%-include('header')-%>

<div class="text-center center-div" id="login">

<div class="container w-25 border py-5">

<div class="title pb-5">

<h2 class="font-weight-bold">Login System</h2>

<span>Log in for the existing user</span>

</div>

<form action="#" class"pt-3">

<div class="forn-group">

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

<small class="form-text text-muted-text-left">Register email address</small>

</div>

<div class="form-group">

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

</div>

<button type="submit" class="btn btn-success rounded-pill">Submit</button>

</form>

</div>

</div>

<% -include('footer')-%>

import

url('https://fonts.googleapis.com/css2?family=Open+Sans:ital,wght@0,300..800;1,300..800&display=swap');

body{

    font-family: "Open Sans", serif;

    background-image:url('/assets/background\_png.png');

    background-size: cover;

    background-repeat: no-repeat;

}

.center-div{

    padding-top:10%;

}

.btn.btn-success{

    padding:.4em 2em;

}

form{

    padding:0 3em;

}

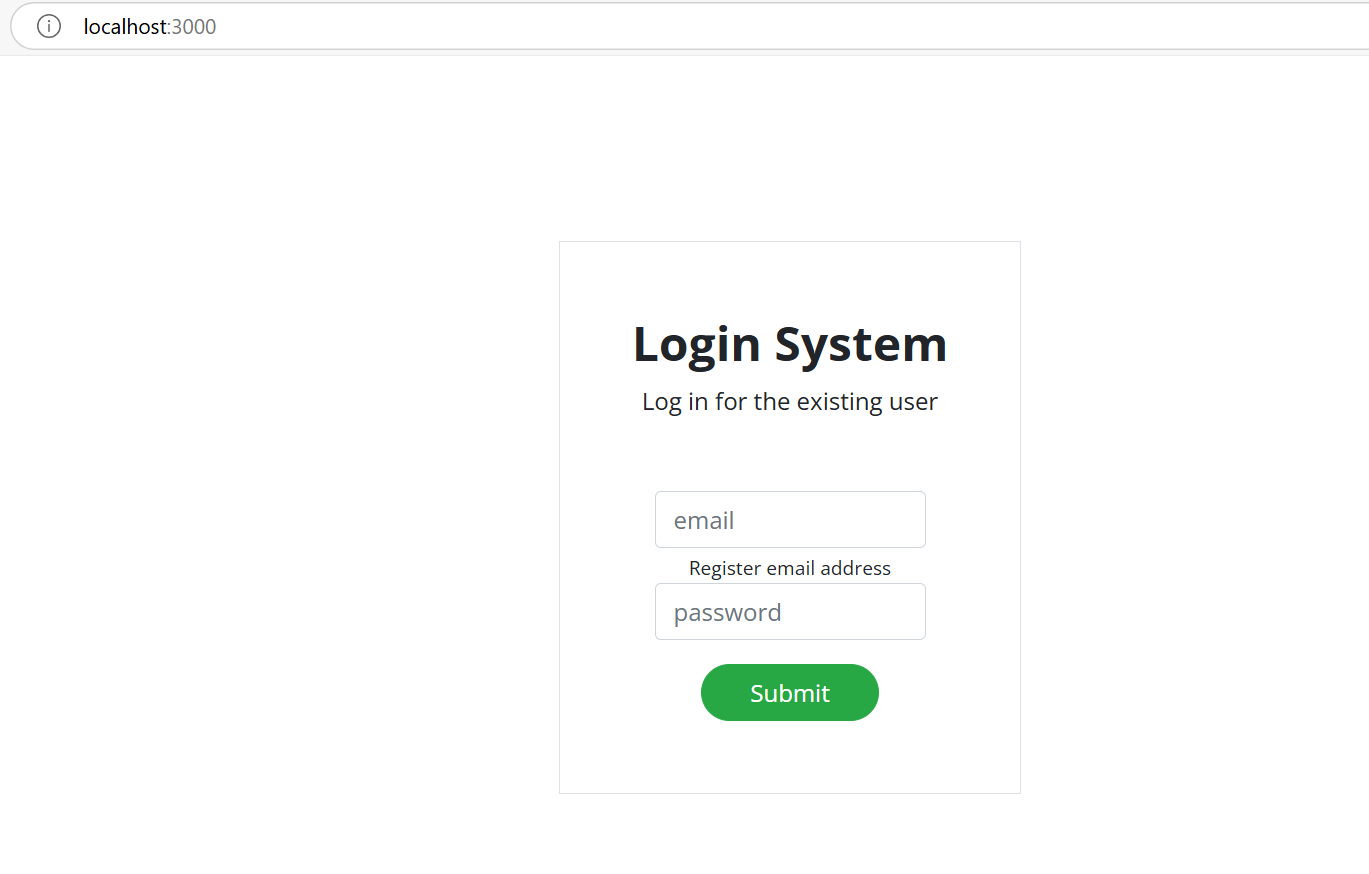
#login.title>span{

    font-size:.8em;

}

}

**Sample Output:**

****

**Viva Questions:**

1.What is Node.js, and why did you use it for this application?

2. What is the role of Express.js in your application?

3. How does your login system handle user authentication?

4.What is the difference between authentication and authorization?

5.How do you manage user sessions in your application?

**Week-3**

**Write a Node JS program to perform read, write and other operations on a file.**

**Aim:** The aim of this experiment is to understand how to perform file operations such as reading, writing, appending, renaming, and deleting files using Node.js. This helps in handling file system operations asynchronously and synchronously, which is a fundamental concept in backend development.

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code

**Source Program :**

[**fs.readFile():**](https://www.geeksforgeeks.org/node-js-fs-readfile-method) Reads the contents of a file asynchronously.

[**fs.writeFile():**](https://www.geeksforgeeks.org/node-js-fs-writefile-method) Writes data to a file asynchronously, replacing the file if it already exists.

[**fs.appendFile()**](https://www.geeksforgeeks.org/node-js-fs-appendfile-function)**:** Appends data to a file asynchronously, creating the file if it does not exist.

[**fs.unlink()**](https://www.geeksforgeeks.org/node-js-fs-unlink-method)**:** Deletes a file asynchronously.

[**fs.mkdir()**](https://www.geeksforgeeks.org/node-js-fs-mkdir-method)**:** Creates a directory asynchronously.

[**fs.rmdir()**](https://www.geeksforgeeks.org/node-js-fs-rmdir-method)**:** Removes a directory asynchronously.

**importing the fs Module**

To use the fs module in your Node.js application, you need to import it first

const fs = require('fs');

**Reading a File**

You can read a file asynchronously using fs.readFile:

let fs=require('fs')

fs.readFile('./content.txt',(err,data)=>{

if(err){

console.log("file not found")

}

else{

console.log(data)

}

}

)

**Writing to a File**

To write data to a file, you can use fs.writeFile

Writing File;

let fs=require('fs')

fs.writeFile('./content1.txt',"updating program",(err)=>{

if(err){

console.log("Not updated")

}

    else{

       console.log("Updated Successfully")

    }})

**3. Appending to a File**

If you want to append data to an existing file, use fs.appendFile:

console.log("before writing")

let fs=require('fs')

fs.appendFile('./topics.txt',"adding extra",(err)=>{

if(err){

console.log("Not Written")

}

else{

console.log("File has been written successfully")

}

})

**4. Deleting a File**

To delete a file, you can use fs.unlink:

let fs=require('fs')

fs.unlike('./content.txt'),(err)=>{

if(err){

console.log("File not deleted");

}

else{

console.log("File has been deleted successfully")

}

}

**5. Creating a Directory**

To create a new directory, use fs.mkdir:

let fs=require('fs')

fs.mkdir('./test2',(err)=>{

if(err){

console.log("Directory not created")

}

else{

console.log("Directory created successfully")

}

})

console.log("before writing")

**Sample Output:**

File written successfully!

File content: Hello, this is a Node.js file operation example.

Content appended successfully!

Updated file content: Hello, this is a Node.js file operation example.

Appending new content.

File deleted successfully!

**Viva Voce Questions**

**1.** What is the fs module in Node.js?

2. How do you read a file in Node.js?

3.What is the difference between fs.readFile() and fs.readFileSync()?

4. What is the purpose of fs.unlink()?

5. What is the purpose of fs.appendFile()?

**Week-4**

**Write a Node JS program to read form data from query string and generate response using**

**NodeJS**

**Aim:** The aim of this experiment is to understand how to handle user input via query strings

in a Node.js application and generate an appropriate response.

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code

**Source code**:

const http = require('http');

const url = require('url');

// Create an HTTP server

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

// Parse the URL and extract query parameters

const query = url.parse(req.url, true).query;

const name = query.name || 'Guest';

const age = query.age || 'Unknown';

const email = query.email || 'Not Provided';

// Set response header

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

// Generate response with submitted data

res.write('<h1>Form Data Received</h1>');

res.write('<p>Here is the data you submitted:</p>');

res.write(`<ul>

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

<li><strong>Age:</strong> ${age}</li>

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

</ul>`);

res.end();

});

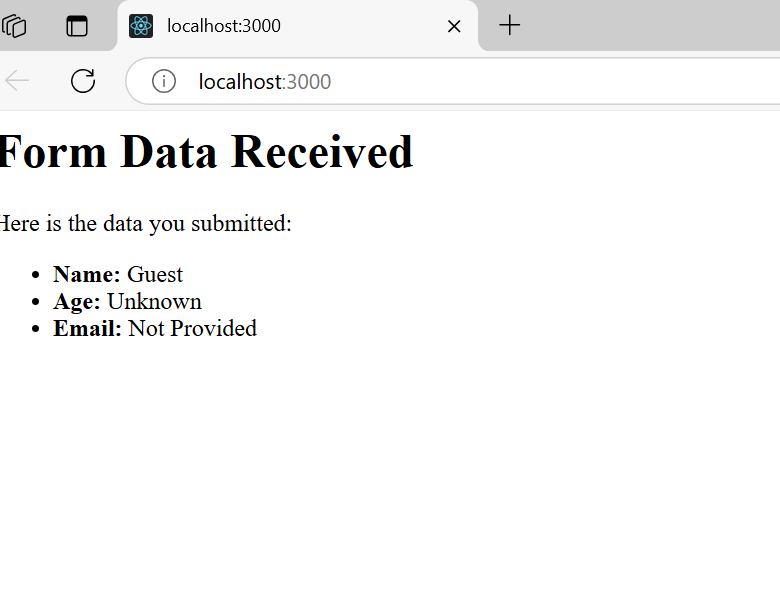
// Start the server on port 3000

server.listen(3000, () => {

console.log('Server is running on http://localhost:3000');

});

**OUTPUT:**

****

**Viva Questions:**

1.What is Middleware in Express.js

2. Define Node.js and Databases

3. Define Callbacks

4. Role of package.json in Node.js

**Week-5**

**Create a food delivery website where users can order food from a particular restaurant**

**listed in the website for handling http requests and responses using NodeJS**

**Aim:** The aim of this experiment is to build a simple food delivery website using **Node.js**,

where users can browse available restaurants, select a restaurant, and order food. This

experiment helps in understanding how to handle **HTTP requests and responses** in

Node.js and serve dynamic content using the Express framework.

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code

**Source Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Food Order</title>

</head>

<body>

<h2>Place Your Order</h2>

<form action="/order" method="POST">

<label for="item">Item:</label>

<input type="text" id="item" name="item" required><br><br>

<label for="quantity">Quantity:</label>

<input type="number" id="quantity" name="quantity" min="1" required><br><br>

<button type="submit">Place Order</button>

</form>

</body>

</html>

**Order.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Food Order</title>

</head>

<body>

<h2>Place Your Order</h2>

<form action="/order" method="POST">

<label for="item">Item:</label>

<input type="text" id="item" name="item" required><br><br>

<label for="quantity">Quantity:</label>

<input type="number" id="quantity" name="quantity" min="1" required><br><br>

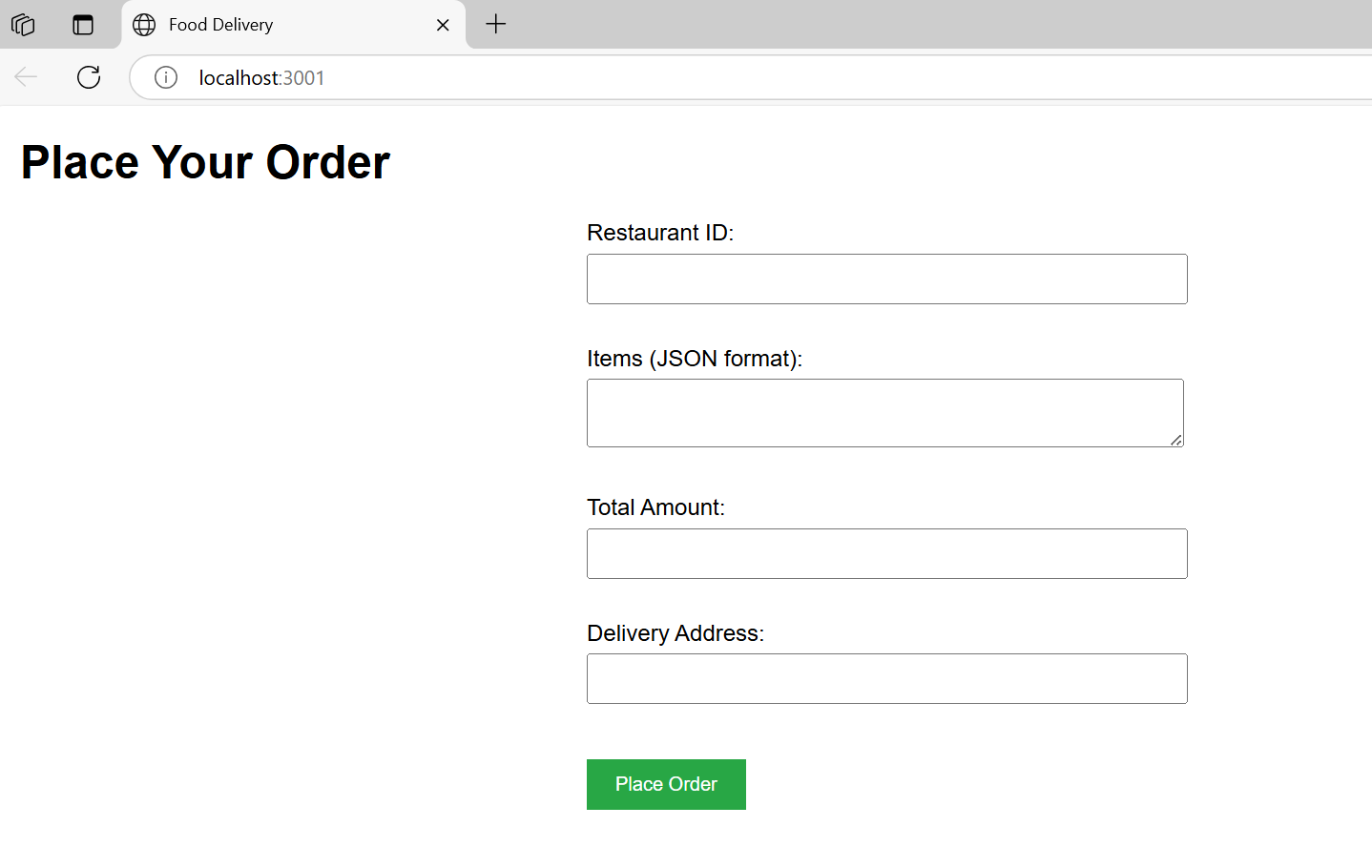
<button type="submit">Place Order</button>

</form>

</body>

</html>

**Sample Output**:



**Viva Questions**:

1.What is the purpose of Express.js in this project?

2. What is the difference between GET and POST requests?

3. How does Express serve HTML content dynamically?

4. How would you implement a user authentication system in this project

**Week-6**

**Implement a program with basic commands on databases and collections using MongoDB.**

**Aim:** The aim of this experiment is to understand how to perform basic database operations

using **MongoDB**. This includes creating a database, creating collections, inserting

documents, querying data, updating records, and deleting data using **Node.js and MongoDB**.

1. Connects to MongoDB using MongoClient.

2. Creates/Selects a database (myDatabase) and a collection (users).

3. Performs CRUD operations:

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code

**Source Code:**

const { MongoClient } = require('mongodb');

// MongoDB connection URL

const url = 'mongodb://localhost:27017';

// Create a new MongoDB client

const client = new MongoClient(url);

async function run() {

try {

// Connect to MongoDB

await client.connect();

console.log("Connected to MongoDB");

// Select the database and collection

const database = client.db('myDatabase');

const collection = database.collection('users');

// Insert a document

const insertResult = await collection.insertOne({ name: 'Alice', age: 25, city: 'New York' });

console.log("Inserted document ID:", insertResult.insertedId);

// Insert multiple documents

await collection.insertMany([

{ name: 'Bob', age: 28, city: 'Los Angeles' },

{ name: 'Charlie', age: 32, city: 'Chicago' }

]);

console.log("Multiple documents inserted");

// Find a single document

const user = await collection.findOne({ name: 'Alice' });

console.log('User found:', user);

// Find multiple documents

const users = await collection.find({}).toArray();

console.log('All Users:', users);

// Update a document

await collection.updateOne({ name: 'Alice' }, { $set: { age: 26 } });

console.log("Document updated");

// Delete a document

await collection.deleteOne({ name: 'Charlie' });

console.log("Document deleted");

// List all collections in the database

const collections = await database.listCollections().toArray();

console.log("Collections:", collections.map(col => col.name));

} catch (error) {

console.error("Error:", error);

} finally {

// Close the MongoDB connection

await client.close();

console.log("Connection closed");

}

}

// Execute the function

run().catch(console.dir);

**Sample Output:**

Connected to MongoDB

Inserted document ID: 660c1e7...

Multiple documents inserted

User found: { \_id: ObjectId("660c1e7..."), name: 'Alice', age: 25, city: 'New York' }

All Users: [ { \_id: ObjectId(...), name: 'Alice', age: 25, city: 'New York' }, { \_id: ObjectId(...), name: 'Bob',

age: 28, city: 'Los Angeles' }, { \_id: ObjectId(...), name: 'Charlie', age: 32, city: 'Chicago' } ]

Document updated

Document deleted

Collections: [ 'users' ]

Connection closed

**Viva Questions:**

* 1. What is MongoDB
  2. What is the difference between SQL and NoSQL databases?
  3. What is a collection in MongoDB?
  4. What is a document in MongoDB?

5. What is the difference between findOne() and find() in MongoDB?

**Week-7**

**Implement CRUD operations on the given dataset using MongoDB.**

**Aim:** The aim of this experiment is to implement **CRUD (Create, Read, Update, Delete)**

**operations** on a given dataset using **MongoDB**. This will help in understanding how to

manipulate data in a NoSQL database using **Node.js** and the mongodb package.

1. Connects to MongoDB using MongoClient.

2. Creates/Selects a database (myDatabase) and a collection (users).

3. Performs CRUD operations:

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code

Source Code:

const { MongoClient } = require('mongodb');

// MongoDB Connection URL

const url = 'mongodb://localhost:27017';

const client = new MongoClient(url);

async function run() {

try {

// Connect to MongoDB

await client.connect();

console.log("Connected to MongoDB");

// Select or create a database

const database = client.db('testdb');

// Select or create a collection

const collection = database.collection('users');

// Insert a document

const insertResult = await collection.insertOne({ name: 'John Doe', age: 30, city: 'New York' });

console.log("Inserted Document ID:", insertResult.insertedId);

// Find a single document

const user = await collection.findOne({ name: 'John Doe' });

console.log('User found:', user);

// Insert multiple documents

await collection.insertMany([

{ name: 'Alice', age: 25, city: 'Los Angeles' },

{ name: 'Bob', age: 28, city: 'Chicago' }

]);

console.log("Multiple documents inserted");

// Retrieve all documents

const allUsers = await collection.find({}).toArray();

console.log('All Users:', allUsers);

// Update a document

await collection.updateOne({ name: 'John Doe' }, { $set: { age: 31 } });

console.log("Document updated");

// Delete a document

await collection.deleteOne({ name: 'John Doe' });

console.log("Document deleted");

// List all collections in the database

const collections = await database.listCollections().toArray();

console.log("Collections in database:", collections.map(col => col.name));

} catch (error) {

console.error("Error:", error);

} finally {

// Close the database connection

await client.close();

console.log("Connection closed");

}

}

// Run the function

run().catch(console.dir);

**Sample Output:**

Connected to MongoDB

Inserted Document ID: 65a7b2c4f9a3e4c1b2a4f9c1

User found: { \_id: ObjectId("65a7b2c4f9a3e4c1b2a4f9c1"), name: 'John Doe', age: 30, city: 'New York' }

2 Documents Inserted

All Users: [

{ \_id: ObjectId("..."), name: 'John Doe', age: 30, city: 'New York' },

{ \_id: ObjectId("..."), name: 'Alice', age: 25, city: 'Los Angeles' },

{ \_id: ObjectId("..."), name: 'Bob', age: 28, city: 'Chicago' }

]

Document Updated

Document Deleted

Collections in Database: [ 'users' ]

Connection Closed

**Viva Questions:**

1. What is the difference between insertOne() and insertMany()?
2. What is the purpose of the \_id field in MongoDB?
3. What is a replica set in MongoDB?
4. What is sharding in MongoDB?
5. How can MongoDB be secured?

**Week-8**

**Perform Count, Limit, Sort, and Skip operations on the given collections using**

**MongoDB.**

**Aim:** The aim of this experiment is to understand how to perform count, limit, sort, and skip

operations on a given MongoDB collection. These operations are useful for data retrieval

optimization, pagination, and ordering of results when working with large datasets.

1. Connects to MongoDB using MongoClient.

2. Creates/Selects a database (myDatabase) and a collection (users).

3. Performs CRUD operations:

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code, MongoDB

Let's assume we have a **MongoDB collection named students** with the following documents:

JSON

[

{ "\_id": 1, "name": "Alice", "age": 22, "marks": 85 },

{ "\_id": 2, "name": "Bob", "age": 21, "marks": 90 },

{ "\_id": 3, "name": "Charlie", "age": 23, "marks": 78 },

{ "\_id": 4, "name": "David", "age": 20, "marks": 88 },

{ "\_id": 5, "name": "Eva", "age": 22, "marks": 95 },

{ "\_id": 6, "name": "Frank", "age": 24, "marks": 70 }

]

* 1. **Count Documents**

To count the number of documents in a collection:

db.collection\_name.countDocuments({})

Output: 6

### **2. Limit Results**

Query:db.students.find().limit(3)

**Output:**

[

{ "\_id": 1, "name": "Alice", "age": 22, "marks": 85 },

{ "\_id": 2, "name": "Bob", "age": 21, "marks": 90 },

{ "\_id": 3, "name": "Charlie", "age": 23, "marks": 78 }

]

**3. Sort Documents (Descending by Marks)**

Query:

db.students.find().sort({ marks: -1 })

Output:

[

{ "\_id": 5, "name": "Eva", "age": 22, "marks": 95 },

{ "\_id": 2, "name": "Bob", "age": 21, "marks": 90 },

{ "\_id": 4, "name": "David", "age": 20, "marks": 88 },

{ "\_id": 1, "name": "Alice", "age": 22, "marks": 85 },

{ "\_id": 3, "name": "Charlie", "age": 23, "marks": 78 },

{ "\_id": 6, "name": "Frank", "age": 24, "marks": 70 }

]

**4. Skip First 2 and Limit 3 (Sorted by Marks)**

Query:

db.students.find().sort({ marks: -1 }).skip(2).limit(3)

[

{ "\_id": 4, "name": "David", "age": 20, "marks": 88 },

{ "\_id": 1, "name": "Alice", "age": 22, "marks": 85 },

{ "\_id": 3, "name": "Charlie", "age": 23, "marks": 78 }

]

**Viva Questions:**

1. What is the use of the limit() function in MongoDB?
2. What is the purpose of the sort() function in MongoDB?
3. What is the skip() function used for in MongoDB?
4. What is the difference between countDocuments() and estimatedDocumentCount**()?**

**Week-9**

**Develop an angular JS form to apply CSS and Events**

**Aim:** The aim of this experiment is to develop a **form using AngularJS** that applies **CSS styles**

**dynamically** and handles **events** such as button clicks, input changes, and form

submission. This experiment helps in understanding how to bind data, manipulate CSS

classes, and handle user interactions using AngularJS.

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code, MongoDB

Source Code:

<!DOCTYPE html>

<html lang="en" ng-app="cssEventApp">

<head>

<meta charset="UTF-8">

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

<title>AngularJS Form with CSS and Events</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

<style>

body { font-family: Arial, sans-serif; text-align: center; }

.container { width: 50%; margin: auto; padding: 20px; border: 1px solid #ddd; border-radius: 10px; box-shadow: 2px 2px 10px #aaa; }

.highlight { border: 2px solid green; background-color: #eaffea; }

.error { color: red; font-size: 12px; }

input, select { width: 100%; padding: 8px; margin: 5px 0; border: 1px solid #ddd; }

button { padding: 10px; background: green; color: white; border: none; cursor: pointer; }

.hover-effect { background-color: yellow; cursor: pointer; }

</style>

</head>

<body ng-controller="FormController">

<h2>AngularJS Form with CSS & Events</h2>

<div class="container">

<form name="userForm" ng-submit="submitForm()">

<label>Full Name:</label>

<input type="text" name="fullName" ng-model="user.fullName" required

ng-class="{'highlight': isFocused}"

ng-focus="isFocused=true" ng-blur="isFocused=false">

<div class="error" ng-show="userForm.fullName.$touched && userForm.fullName.$invalid">

Name is required.

</div>

<label>Email:</label>

<input type="email" name="email" ng-model="user.email" required>

<div class="error" ng-show="userForm.email.$touched && userForm.email.$invalid">

Enter a valid email.

</div>

<label>Favorite Color:</label>

<select ng-model="user.color" ng-change="colorChanged()" required>

<option value="">Select a color</option>

<option value="red">Red</option>

<option value="blue">Blue</option>

<option value="green">Green</option>

</select>

<p>Your selected color: <span ng-style="{'color': user.color}">{{ user.color }}</span></p>

<br>

<button type="submit" ng-disabled="userForm.$invalid" ng-mouseover="hover=true" ng-mouseleave="hover=false"

ng-class="{'hover-effect': hover}">

Submit

</button>

</form>

<h3 style="color: green;" ng-show="submitted">Form Submitted Successfully!</h3>

</div>

<script>

var app = angular.module("cssEventApp", []);

app.controller("FormController", function($scope) {

$scope.user = {};

$scope.isFocused = false;

$scope.hover = false;

$scope.submitted = false;

$scope.colorChanged = function() {

console.log("Color changed to:", $scope.user.color);

};

$scope.submitForm = function() {

if ($scope.userForm.$valid) {

$scope.submitted = true;

console.log("Form Data:", $scope.user);

}

};

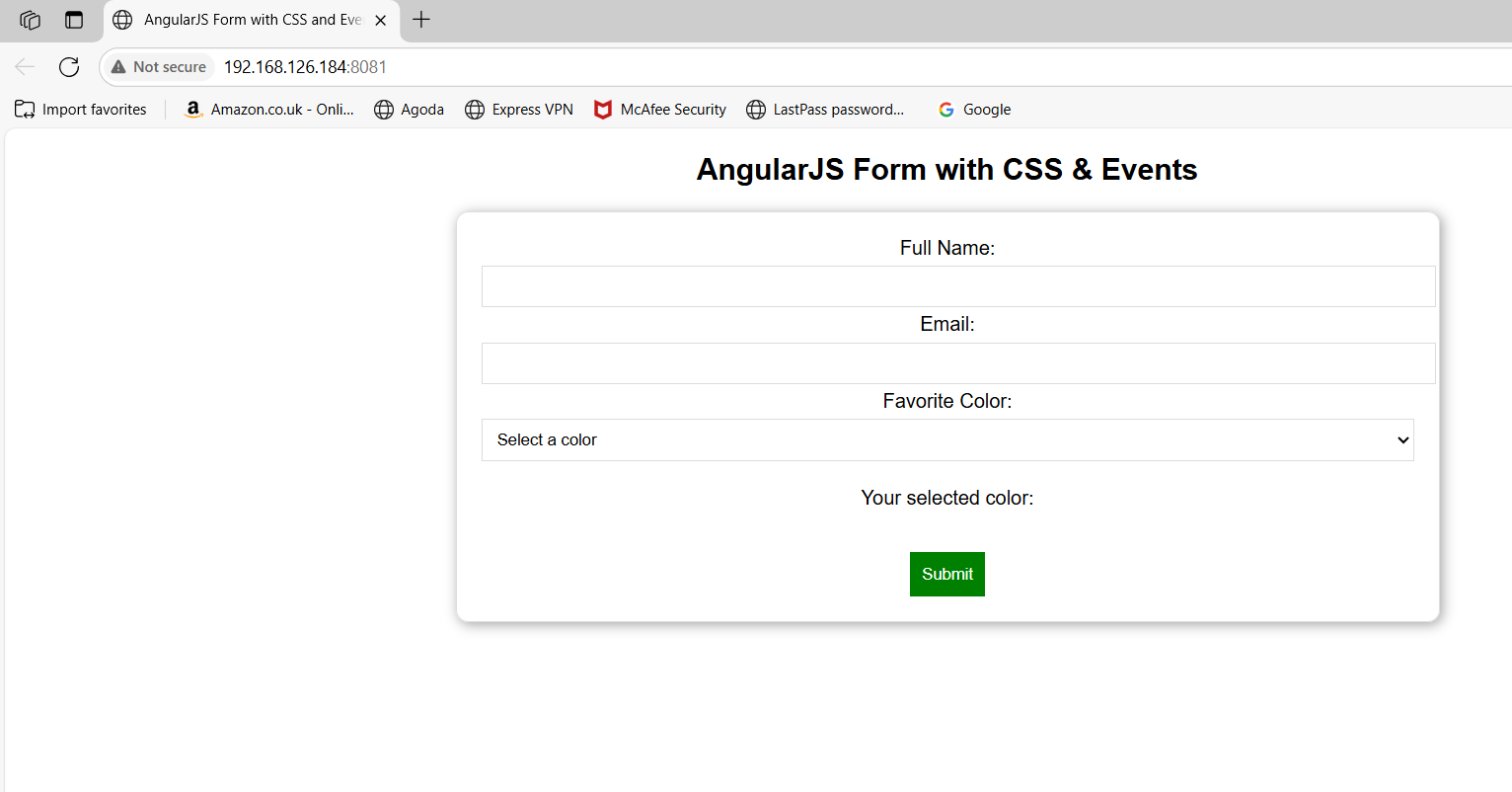
});

</script>

</body>

</html>

**Output:**

****

**Viva Questions:**

1.What is AngularJS?

2.What are the advantages of using AngularJS?

3.What is the difference between AngularJS and Angular?

4. What is ng-model in AngularJS?

5. What is $scope in AngularJS?

**Week-10**

**Develop a Job Registration form and validate it using angular JS.**

**Aim**: The aim of this experiment is to develop a **Job Registration Form** using **AngularJS** and implement **form validation**. This experiment helps in understanding how to:

* Create a **dynamic form** using **AngularJS directives**.
* Implement **real-time validation** for required fields, email format, and password strength.
* Display error messages when validation fails.
* Enable or disable the submit button based on validation.

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code, MongoDB

**Source Code:**

<!DOCTYPE html>

<html lang="en" ng-app="jobApp">

<head>

<meta charset="UTF-8">

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

<title>Job Registration Form</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

<style>

body { font-family: Arial, sans-serif; text-align: center; }

.container { width: 40%; margin: auto; }

.error { color: red; font-size: 12px; }

input, select { width: 100%; padding: 8px; margin: 5px 0; }

button { padding: 10px; background: green; color: white; border: none; cursor: pointer; }

</style>

</head>

<body ng-controller="JobController">

<h2>Job Registration Form</h2>

<div class="container">

<form name="jobForm" ng-submit="submitForm()">

<label>Full Name:</label>

<input type="text" name="fullName" ng-model="user.fullName" required>

<div class="error" ng-show="jobForm.fullName.$touched && jobForm.fullName.$invalid">

Name is required.

</div>

<label>Email:</label>

<input type="email" name="email" ng-model="user.email" required>

<div class="error" ng-show="jobForm.email.$touched && jobForm.email.$invalid">

Enter a valid email.

</div>

<label>Phone Number:</label>

<input type="text" name="phone" ng-model="user.phone" ng-pattern="/^\d{10}$/" required>

<div class="error" ng-show="jobForm.phone.$touched && jobForm.phone.$invalid">

Enter a valid 10-digit phone number.

</div>

<label>Password:</label>

<input type="password" name="password" ng-model="user.password" required>

<div class="error" ng-show="jobForm.password.$touched && jobForm.password.$invalid">

Password is required.

</div>

<label>Confirm Password:</label>

<input type="password" name="confirmPassword" ng-model="user.confirmPassword" required ng-change="checkPassword()">

<div class="error" ng-show="user.password !== user.confirmPassword">

Passwords do not match.

</div>

<label>Job Role:</label>

<select name="jobRole" ng-model="user.jobRole" required>

<option value="">Select a role</option>

<option value="Developer">Developer</option>

<option value="Designer">Designer</option>

<option value="Manager">Manager</option>

</select>

<div class="error" ng-show="jobForm.jobRole.$touched && jobForm.jobRole.$invalid">

Please select a job role.

</div>

<br>

<button type="submit" ng-disabled="jobForm.$invalid">Register</button>

</form>

<h3 style="color: green;" ng-show="submitted">Registration Successful!</h3>

</div>

<script>

var app = angular.module("jobApp", []);

app.controller("JobController", function($scope) {

$scope.user = {};

$scope.submitted = false;

$scope.checkPassword = function() {

if ($scope.user.password !== $scope.user.confirmPassword) {

$scope.jobForm.confirmPassword.$setValidity("passwordMatch", false);

} else {

$scope.jobForm.confirmPassword.$setValidity("passwordMatch", true);

}

};

$scope.submitForm = function() {

if ($scope.jobForm.$valid) {

$scope.submitted = true;

console.log("Form Submitted!", $scope.user);

}

};

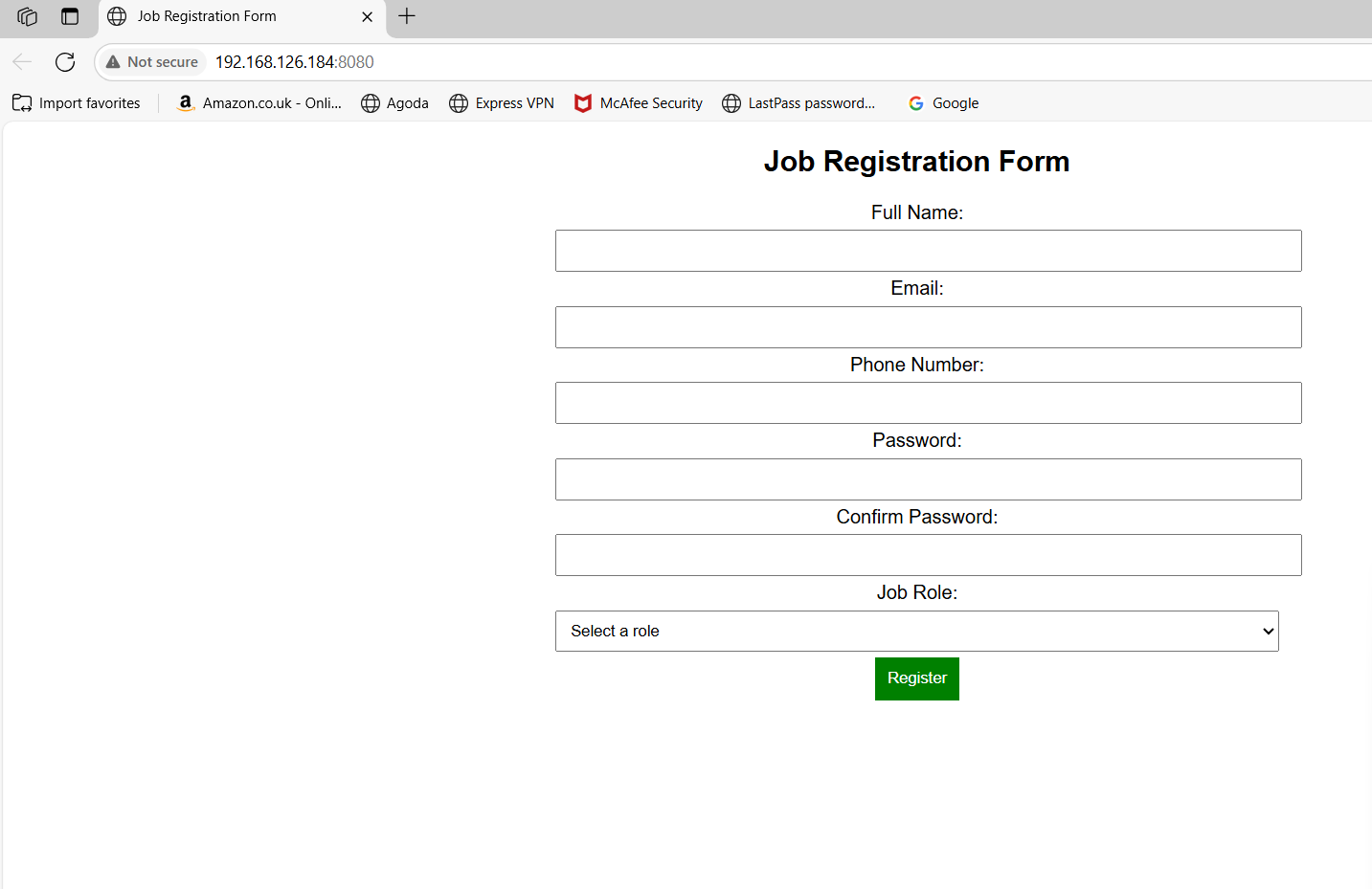
});

</script>

</body>

</html>

**Output:**



**Viva Questions:**

* 1. What are the key features of AngularJS
  2. What is the difference between ng-show and ng-if?
  3. What is novalidate in an HTML form, and why is it used?
  4. What is $dirty, $pristine, $valid, and $invalid in AngularJS forms?

**Week-11**

**Write an angular JS application to access JSON file data of an employee from a**

**server using $http service.**

**Aim:** The aim of this experiment is to develop an **AngularJS application** that accesses **employee data** from a **JSON file** stored on a server using the $http service. This experiment helps in understanding:

* How to use AngularJS’s $http service to fetch data from an external JSON file.
* How to display the retrieved data dynamically on an HTML page.
* How to handle HTTP responses and errors in AngularJS.

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code, MongoDB

Source Code:

**// app.js**

var app = angular.module('EmployeeApp', []);

app.controller('EmployeeController', function($scope, $http) {

// Define the URL of the JSON file on the server

var url = 'employees.json';

// Fetch data using $http service

$http.get(url)

.then(function(response) {

// Assign the response data to the scope variable

$scope.employees = response.data;

})

.catch(function(error) {

console.error('Error fetching employee data:', error);

});

});

**// index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Employee Data</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

<script src="app.js"></script>

</head>

<body ng-app="EmployeeApp" ng-controller="EmployeeController">

<h1>Employee List</h1>

<table border="1">

<tr>

<th>ID</th>

<th>Name</th>

<th>Department</th>

</tr>

<tr ng-repeat="employee in employees">

<td>{{ employee.id }}</td>

<td>{{ employee.name }}</td>

<td>{{ employee.department }}</td>

</tr>

</table>

</body>

</html>

**// employees.json**

[

{ "id": 1, "name": "John Doe", "department": "Engineering" },

{ "id": 2, "name": "Jane Smith", "department": "HR" },

{ "id": 3, "name": "Mike Johnson", "department": "Finance" }

]

**How to run:**

**1. Set Up the Project**

* Create a folder for your project (e.g., EmployeeApp).
* Inside the folder, create the following files:
  + index.html
  + app.js
  + employees.json

**2. Add the Code**

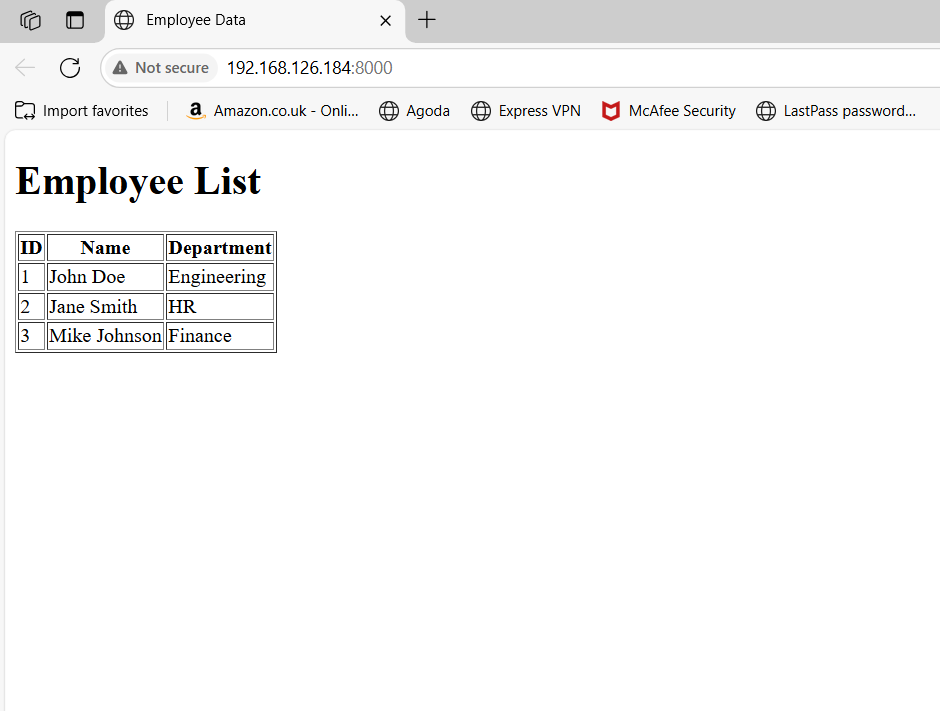
* Write code into the respective files**.**

**3. Run a Local Server**

Since JSON files cannot be loaded directly using file:// protocol due to CORS restrictions, you need to run a local server.

Open your browser and go to:  
http://localhost:8000/index.html

**Output:**

****

**Viva Question:**

1. What is the $http service in AngularJS?

2. What are the HTTP methods supported by $http service

3. What is JSON, and why is it used?

4. How does AngularJSparse JSON data?

**Week-12**

**Develop a web application to manage student information using Express and**

**Angular JS.**

**Aim**: The aim of this experiment is to **develop a web application** using **Express.js** (backend) and **AngularJS** (frontend) to **manage student information**. This application will allow users to:

* **Add new students** to the database.
* **Retrieve and display student records** dynamically.
* **Update student details** when needed.
* **Delete student records** if necessary.

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code, MongoDB

**Server.js :**

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

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

const app = express();

app.use(cors());

app.use(bodyParser.json());

mongoose.connect('mongodb://127.0.0.1:27017/studentsDB', { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('MongoDB Connected'))

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

// Student Schema

const Student = mongoose.model('Student', new mongoose.Schema({

name: String,

age: Number,

course: String

}));

// Routes

app.get('/students', async (req, res) => {

const students = await Student.find();

res.json(students);

});

app.post('/students', async (req, res) => {

const student = new Student(req.body);

await student.save();

res.json(student);

});

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

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

res.json({ message: 'Student deleted' });

});

app.listen(3000, () => console.log('Server running on port 3000'));

**index.html :**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Student Management</title>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

<script src="app.js"></script>

</head>

<body ng-app="studentApp" ng-controller="StudentController">

<div class="container mt-5">

<h2>Student Management System</h2>

<form ng-submit="addStudent()">

<input type="text" ng-model="newStudent.name" placeholder="Name" required class="form-control mb-2">

<input type="number" ng-model="newStudent.age" placeholder="Age" required class="form-control mb-2">

<input type="text" ng-model="newStudent.course" placeholder="Course" required class="form-control mb-2">

<button type="submit" class="btn btn-success">Add Student</button>

</form>

<table class="table table-bordered mt-3">

<tr>

<th>Name</th>

<th>Age</th>

<th>Course</th>

<th>Actions</th>

</tr>

<tr ng-repeat="student in students">

<td>{{student.name}}</td>

<td>{{student.age}}</td>

<td>{{student.course}}</td>

<td>

<button class="btn btn-danger" ng-click="deleteStudent(student.\_id)">Delete</button>

</td>

</tr>

</table>

</div>

</body>

</html>

**App.js:**

var app = angular.module('studentApp', []);

app.controller('StudentController', function($scope, $http) {

$scope.students = [];

$scope.newStudent = {};

// Fetch students

$scope.getStudents = function() {

$http.get('http://localhost:3000/students').then(function(response) {

$scope.students = response.data;

});

};

// Add student

$scope.addStudent = function() {

$http.post('http://localhost:3000/students', $scope.newStudent).then(function(response) {

$scope.students.push(response.data);

$scope.newStudent = {};

});

};

// Delete student

$scope.deleteStudent = function(id) {

$http.delete('http://localhost:3000/students/' + id).then(function() {

$scope.getStudents();

});

};

// Initial Fetch

$scope.getStudents();

});

**How to run:**

Step 1: Initialize Node.js and Install Dependencies

* cd Folder Name
* npm init -y
* npm install express mongoose cors body-parser

Step 2: Create the Server (server.js):

Create a server.js file and set up Express.js

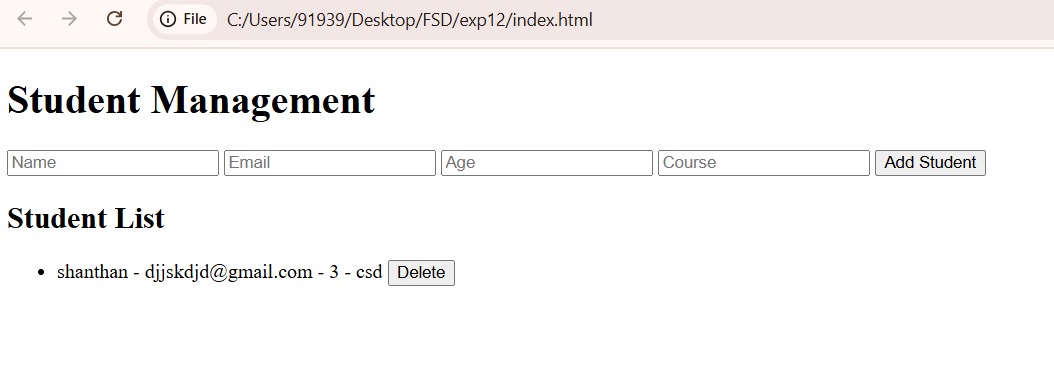
Step 3: Create an AngularJS Project:

Create an index.html file

Step 3: Create AngularJS App (app.js):

Create an app.js file to handle API requests

**Output:**



**Viva Questions:**

1.What is Express.js, and why is it used?

2. What is middleware in Express.js

3.What is CORS, and why is it needed

4. How can you secure an Express.js API

5.What is the purpose of body-parser?

**Week-13**

**Write a program to create a simple calculator application using ReactJS**

**Aim:** The aim of this experiment is to **develop a simple calculator application using ReactJS**. This application will allow users to perform basic arithmetic operations like **addition, subtraction, multiplication, and division**

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code, MongoDB

**Source Program:**

***Calculator.js***

import React, { useState } from "react";

import "./Calculator.css"; // Import the updated CSS file

const Calculator = () => {

const [input, setInput] = useState("");

const [result, setResult] = useState("");

// Handle button clicks

const handleClick = (value) => {

setInput((prev) => prev + value);

};

// Evaluate the expression

const calculate = () => {

try {

setResult(eval(input).toString());

} catch (error) {

setResult("Error");

}

};

// Clear the input

const clear = () => {

setInput("");

setResult("");

};

return (

<div className="calculator">

<h2>Simple Calculator</h2>

<div className="display">

<input type="text" value={input} readOnly placeholder="0" />

<div className="result">{result ? `= ${result}` : ""}</div>

</div>

<div className="buttons">

{["7", "8", "9", "/"].map((btn) => (

<button key={btn} onClick={() => handleClick(btn)}>{btn}</button>

))}

{["4", "5", "6", "\*"].map((btn) => (

<button key={btn} onClick={() => handleClick(btn)}>{btn}</button>

))}

{["1", "2", "3", "-"].map((btn) => (

<button key={btn} onClick={() => handleClick(btn)}>{btn}</button>

))}

{["0", ".", "=", "+"].map((btn) =>

btn === "=" ? (

<button key={btn} className="equals" onClick={calculate}>{btn}</button>

) : (

<button key={btn} onClick={() => handleClick(btn)}>{btn}</button>

)

)}

<button className="clear" onClick={clear}>C</button>

</div>

</div>

);

};

export default Calculator;

***Calculator.css***

.calculator {

width: 320px;

margin: 50px auto;

padding: 20px;

border-radius: 10px;

background: #222;

color: white;

text-align: center;

box-shadow: 0px 5px 15px rgba(0, 0, 0, 0.2);

}

h2 {

margin-bottom: 10px;

font-size: 1.5rem;

}

.display {

padding: 15px;

border-radius: 8px;

margin-bottom: 15px;

}

.display input {

width: 100%;

height: 50px;

text-align: right;

font-size: 1.4rem;

background: #444;

color: white;

border: none;

outline: none;

padding: 1%;

border-radius: 5px;

}

.result {

background: none;

text-align: right;

font-size: 1.3rem;

color: #0f0;

margin-top: 5px;

}

.buttons {

display: grid;

grid-template-columns: repeat(4, 1fr);

gap: 10px;

}

button {

height: 60px;

font-size: 1.5rem;

border: none;

background: #555;

color: white;

border-radius: 8px;

cursor: pointer;

transition: 0.2s;

}

button: hover {

background: #777;

}

.equals {

background: #ff9800;

}

.equals: hover {

background: #ff5722;

}

.clear {

grid-column: span 4;

background: #f44336;

}

.clear: hover {

background: #d32f2f;

}

***App.js***

import React from 'react';

import Calculator from './Calculator';

import './App.css';

function App() {

return (

<div className="App">

<h1>Simple Calculator</h1>

<Calculator />

</div>

);

}

export default App;

***index.js***

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

import './index.css';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

**How to run:**

**1. Install Node.js and npm**

**2. Create a New React App**

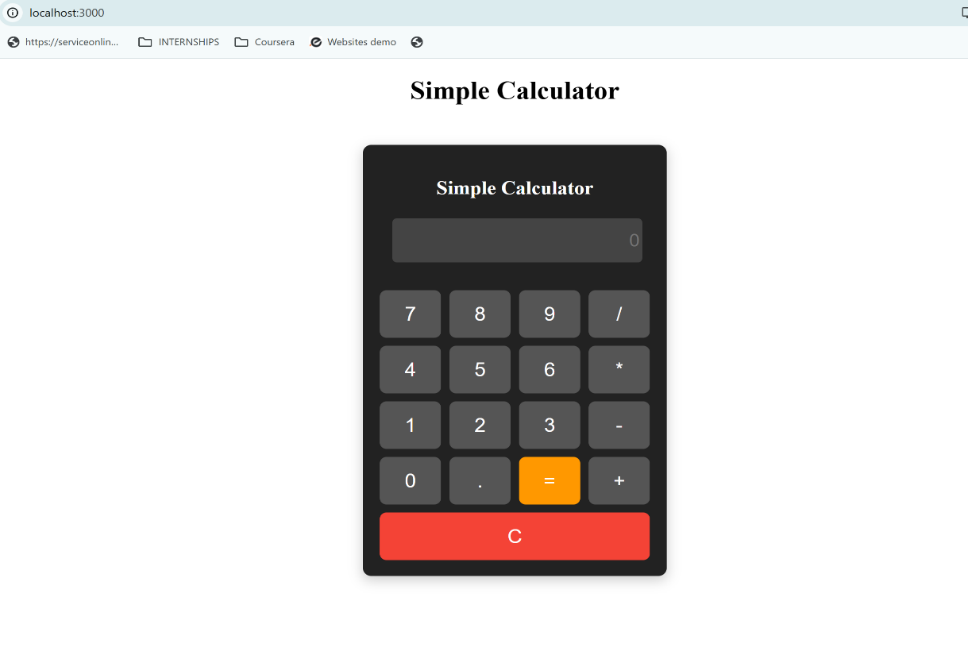
**3. Navigate to the Project Folder**

**4. Replace Files with Calculator Code**

**5. Start the Development Server**

**Note:** If the app does not open, manually visit:  
 http://localhost:3000/

**OUTPUT:**



**Viva Questions:**

1.What are the advantages of ReactJS?

2. What is JSX?

3.What is the difference between functional and class components in React?

4.What is the purpose of useState in React

**Week-14**

**Write a program to create a voting application using React JS**

**Aim:** The aim of this experiment is to develop a **voting application using ReactJS**. This application will allow users to **vote for their favorite candidate or option** and display the total votes dynamically.

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code, MongoDB

import { useState } from "react";

import { Card, CardContent } from "@/components/ui/card";

import { Button } from "@/components/ui/button";

export default function VotingApp() {

const [candidates, setCandidates] = useState([

{ name: "Alice", votes: 0 },

{ name: "Bob", votes: 0 },

{ name: "Charlie", votes: 0 },

]);

const handleVote = (index) => {

const updatedCandidates = [...candidates];

updatedCandidates[index].votes += 1;

setCandidates(updatedCandidates);

};

return (

<div className="flex flex-col items-center gap-4 p-4">

<h1 className="text-2xl font-bold">Voting Application</h1>

<div className="grid grid-cols-1 md:grid-cols-3 gap-4">

{candidates.map((candidate, index) => (

<Card key={index} className="p-4 shadow-lg text-center">

<CardContent>

<h2 className="text-xl font-semibold">{candidate.name}</h2>

<p className="text-lg">Votes: {candidate.votes}</p>

<Button className="mt-2" onClick={() => handleVote(index)}>

Vote

</Button>

</CardContent>

</Card>

))}

</div>

</div>

);

}

1.Set Up Your React Project

npm create vite@latest voting-app --template react

cd voting-app

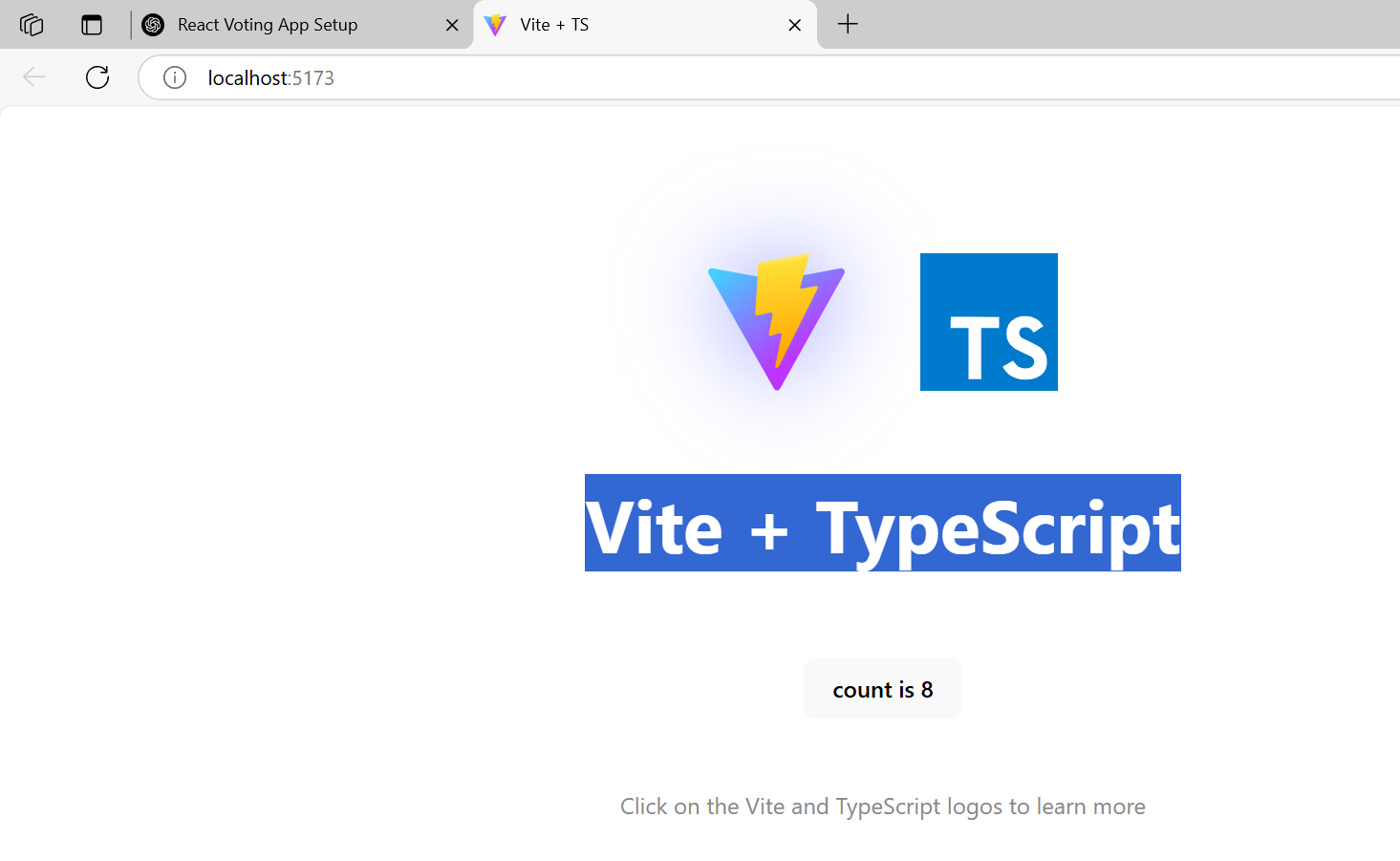
2. npm install @shadcn/ui @radix-ui/react lucide-react

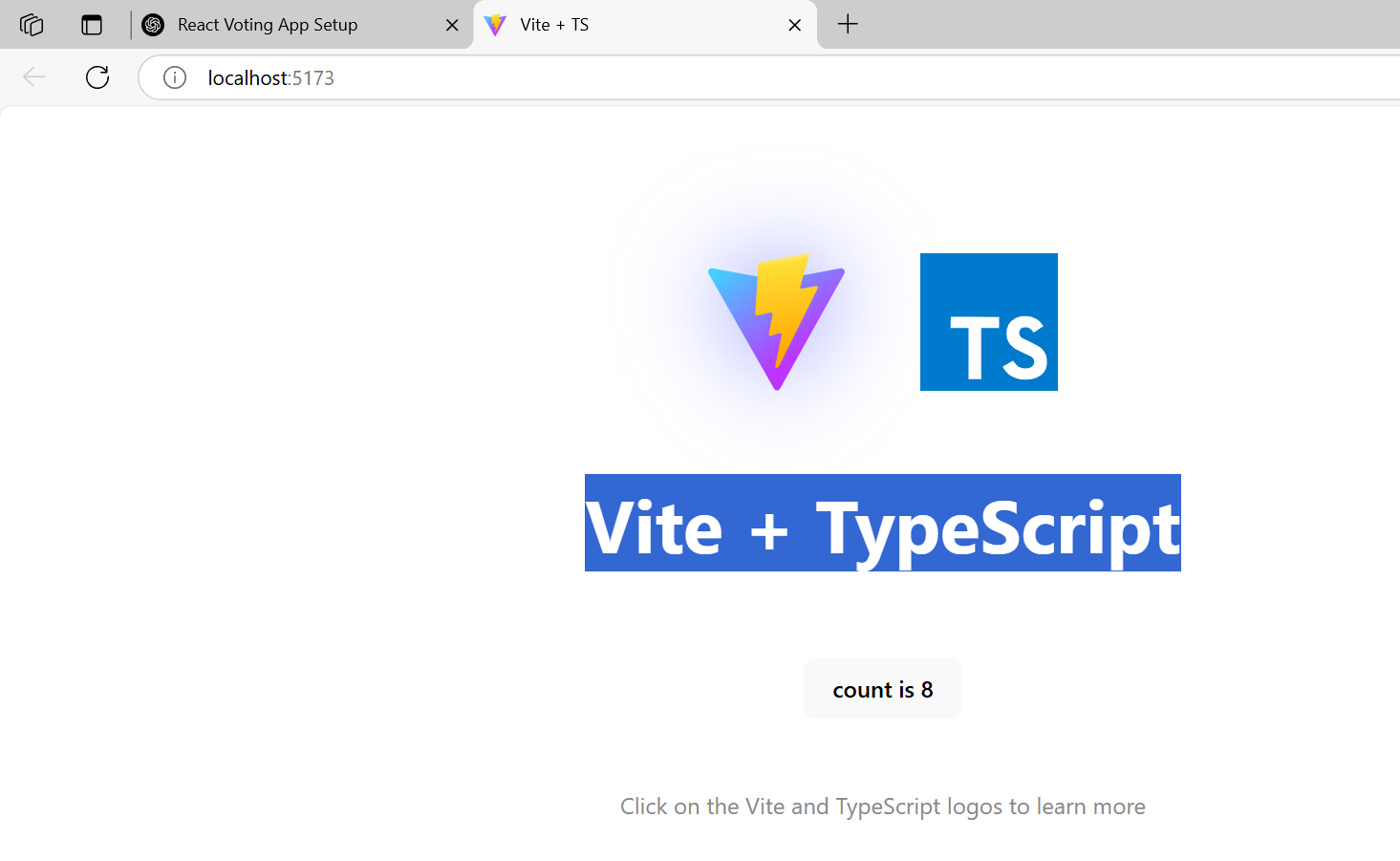
3. Run the Application

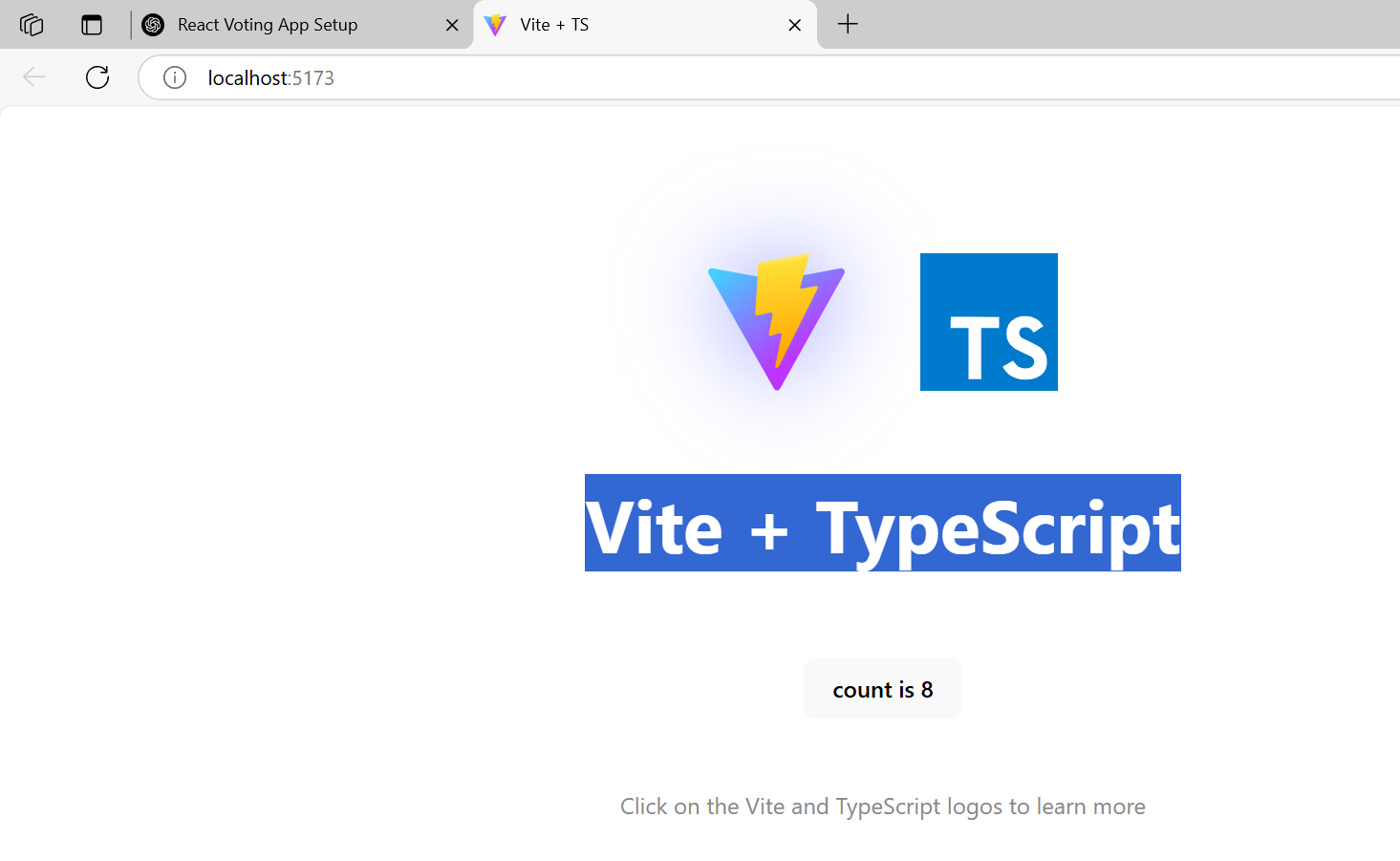
npm install # Install dependencies

npm run dev # For Vite projects

**Output:**



****



**Viva Questions:**

* 1. How does the voting application update votes dynamically
  2. Why do we use key inside map() in React
  3. Why is className used instead of class in React
  4. What is React’s Virtual DOM
  5. How does React efficiently update the UI

**Week-15**

**Develop a leave management system for an organization where users can apply**

**different types of leaves such as casual leave and medical leave. They also can view**

**the available number of days using react application.**

**Aim:** The aim of this experiment is to **develop a Leave Management System using ReactJS** that allows employees to **apply for leave, track different types of leave (casual, medical, etc.), and check their available leave balance**.

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code, MongoDB

// src/LeaveManagementSystem.js

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

import './LeaveManagementSystem.css';

const LeaveManagementSystem = () => {

  // State for available leaves

  const [leaves, setLeaves] = useState({ casual: 0, medical: 0 });

  // Form state and feedback message

  const [formData, setFormData] = useState({

    leaveType: 'casual',

    days: 1,

    reason: '',

  });

  const [message, setMessage] = useState('');

  // Fetch available leaves from backend when component mounts

  useEffect(() => {

    fetch('http://localhost:5000/api/leaves')

      .then((res) => res.json())

      .then((data) => setLeaves(data))

      .catch((error) => console.error('Error fetching leaves:', error));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

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

  };

  const handleSubmit = (e) => {

    e.preventDefault();

    const { leaveType, days } = formData;

    fetch('http://localhost:5000/api/apply', {

      method: 'POST',

      headers: { 'Content-Type': 'application/json' },

      body: JSON.stringify({ leaveType, days }),

    })

      .then((res) => res.json().then((data) => ({ status: res.status, data })))

      .then(({ status, data }) => {

        if (status !== 200) {

          setMessage(data.error);

        } else {

          setMessage(data.message);

          setLeaves(data.availableLeaves);

          // Reset the form

          setFormData({ leaveType: 'casual', days: 1, reason: '' });

        }

      })

      .catch((error) => {

        console.error('Error applying leave:', error);

        setMessage('An error occurred. Please try again later.');

      });

  };

  return (

    <div className="container">

      <h1 className="header">Leave Management System</h1>

      <section className="section">

        <h2>Available Leaves</h2>

        <ul className="leave-list">

          <li><strong>Casual Leave:</strong> {leaves.casual} days</li>

          <li><strong>Medical Leave:</strong> {leaves.medical} days</li>

        </ul>

      </section>

      <section className="section">

        <h2>Apply for Leave</h2>

        <form onSubmit={handleSubmit}>

          <div className="form-group">

            <label htmlFor="leaveType">Leave Type</label>

            <select

              id="leaveType"

              name="leaveType"

              value={formData.leaveType}

              onChange={handleChange}

            >

              <option value="casual">Casual Leave</option>

              <option value="medical">Medical Leave</option>

            </select>

          </div>

          <div className="form-group">

            <label htmlFor="days">Number of Days</label>

            <input

              id="days"

              type="number"

              name="days"

              min="1"

              value={formData.days}

              onChange={handleChange}

            />

          </div>

          <div className="form-group">

            <label htmlFor="reason">Reason</label>

            <textarea

              id="reason"

              name="reason"

              value={formData.reason}

              onChange={handleChange}

              placeholder="Enter your reason here..."

              rows="4"

            />

          </div>

          <button type="submit">Apply Leave</button>

        </form>

      </section>

      {message && <p className="message">{message}</p>}

    </div>

  );

};

***LeaveManagementSystem.css***

/\* src/LeaveManagementSystem.css \*/

.container {

    max-width: 800px;

    margin: 40px auto;

    background: #fff;

    padding: 30px;

    border-radius: 8px;

    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);

    font-family: 'Helvetica Neue', Helvetica, Arial, sans-serif;

  }

    .header {

    text-align: center;

    color: #333;

    margin-bottom: 30px;

  }

    .section {

    margin-top: 30px;

  }

    .leave-list {

    list-style: none;

    padding: 0;

  }

    .leave-list li {

    background: #f9f9f9;

    border: 1px solid #e0e0e0;

    padding: 15px;

    margin-bottom: 10px;

    border-radius: 4px;

    font-size: 1.1em;

  }

  .form-group {

    margin-bottom: 20px;

  }

    .form-group label {

    display: block;

    margin-bottom: 8px;

    font-weight: 600;

    color: #555;

  }

    input,

  select,

  textarea {

    width: 100%;

    padding: 12px;

    border: 1px solid #ccc;

    border-radius: 4px;

    box-sizing: border-box;

    font-size: 1em;

  }

    button {

    background: #007BFF;

    color: #fff;

    border: none;

    padding: 12px 20px;

    border-radius: 4px;

    cursor: pointer;

    font-size: 1em;

  }

    button:hover {

    background: #0056b3;

  }

    .message {

    margin-top: 20px;

    text-align: center;

    font-weight: bold;

    color: #333;

  }

***App.js***

import React from 'react';

import LeaveManagementSystem from './LeaveManagementSystem';

function App() {

  return (

    <div className="App">

      <LeaveManagementSystem />

    </div>

  );

}

export default App;

**How to run:**

1. Install Node.js and npm

2. Create a New React App

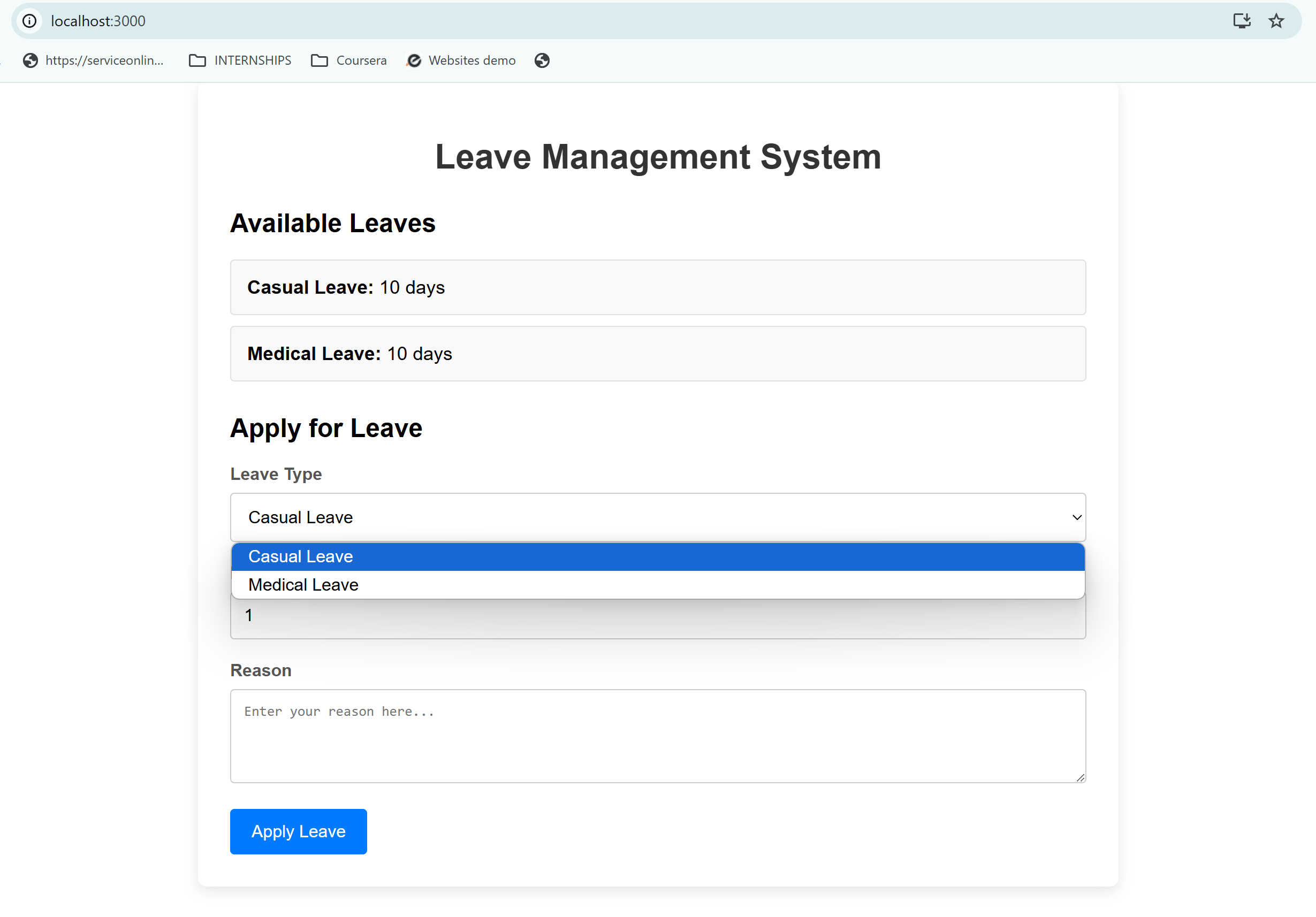
3. Navigate to the Project Folder

4. Replace Files with Code

5. Start the Development Server

**Note:** If the app does not open, manually visit:  
 http://localhost:3000/

Output:



**Viva Questions:**

1.How does React update the UI when state changes

2. How does the Leave Management System track available leave balance?

3. How does the application handle form inputs for leave requests?

4.Why is className used instead of class in JSX?

**Week-16**

**Build a music store application using react components and provide routing among the web pages**

**Aim:** Build a music store application using react components and provide routing among the web

pages

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code, MongoDB

**Source Code:**

import { BrowserRouter as Router, Route, Routes, Link } from "react-router-dom";

function Home() {

return <h1 className="text-center text-2xl p-4">Welcome to the Music Store</h1>;

}

function Store() {

const albums = [

{ id: 1, title: "Album One", artist: "Artist A" },

{ id: 2, title: "Album Two", artist: "Artist B" },

{ id: 3, title: "Album Three", artist: "Artist C" },

];

return (

<div className="p-4">

<h1 className="text-2xl font-bold">Music Store</h1>

<ul className="mt-4">

{albums.map((album) => (

<li key={album.id} className="border p-2 m-2 rounded">

<strong>{album.title}</strong> - {album.artist}

</li>

))}

</ul>

</div>

);

}

function About() {

return <h1 className="text-center text-2xl p-4">About Our Music Store</h1>;

}

function Contact() {

return <h1 className="text-center text-2xl p-4">Contact Us at music@store.com</h1>;

}

function Navbar() {

return (

<nav className="flex gap-4 bg-gray-800 p-4 text-white">

<Link to="/">Home</Link>

<Link to="/store">Store</Link>

<Link to="/about">About</Link>

<Link to="/contact">Contact</Link>

</nav>

);

}

export default function MusicStoreApp() {

return (

<Router>

<Navbar />

<Routes>

<Route path="/" element={<Home />} />

<Route path="/store" element={<Store />} />

<Route path="/about" element={<About />} />

<Route path="/contact" element={<Contact />} />

</Routes>

</Router>

);

}

How to Run:

Install dependencies

Start the development server

Open http://localhost:3000 in your browser.

Output:

**Viva Questions:**

1.What is JSX in ReactJS?

2. What is React Router, and why is it used

3. What is the purpose of useParams() in React Router?

**Week-17**

**Create a react application for an online store which consist of registration, login, product**

**information pages and implement routing to navigate through these pages.**

**Aim:** The aim of this experiment is to develop a **ReactJS Online Store Application** that includes:

* **User Registration Page**
* **Login Page**
* **Product Information Page**

.

**Recommended Hardware / Software Requirements:**

* Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
* Nodejs, VS Code, MongoDB

**program:**

const Navbar = () => {

  return (

    <nav>

      <ul>

        <li><Link to="/">Home</Link></li>

        <li><Link to="/login">Login</Link></li>

        <li><Link to="/register">Register</Link></li>

        <li><Link to="/product">Product</Link></li>

      </ul>

    </nav>

  );

};

export default Navbar;

**Home Page (Home.js)**

This is the home page, which will display a welcome message and a brief description of the store

import React from 'react';

**program:**

const Home = () => {

  return (

    <div>

      <h1>Welcome to the Online Store</h1>

      <p>Browse through our amazing products and make your purchase today!</p>

    </div>

  );

};

export default Home;

.

**Login Page (Login.js)**

The Login page will contain a form for users to enter their email and password. Upon form submission, the input data can be validated or sent to a server for authentication.

**Program:**

import React, { useState } from 'react';

const Login = () => {

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const handleLogin = (e) => {

    e.preventDefault()

    console.log('Logging in with', { email, password });

  };

  return (

    <div>

      <h1>Login</h1>

      <form onSubmit={handleLogin}>

        <div>

          <label>Email: </label>

          <input

            type="email"

            value={email}

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

            required

          />

        </div>

        <div>

          <label>Password: </label>

          <input

            type="password"

            value={password}

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

            required

          />

        </div>

        <button type="submit">Login</button>

      </form>

    </div>

  );

};

export default Login;

**Register Page (Register.js)**

The Register page will include a form for creating a new account. It will have input fields for email, password, and confirm password.

Program:

import { useState } from "react";

import { useNavigate } from "react-router-dom";

const Register = () => {

  const [user, setUser] = useState({ username: "", password: "" });

  const navigate = useNavigate();

  const handleSubmit = (e) => {

    e.preventDefault();

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

    alert("Registration successful!");

    navigate("/login");

  };

  return (

    <div>

      <h2>Register</h2>

      <form onSubmit={handleSubmit}>

        <input type="text" placeholder="Username" required

          onChange={(e) => setUser({ ...user, username: e.target.value })} />

        <input type="password" placeholder="Password" required

          onChange={(e) => setUser({ ...user, password: e.target.value })} />

        <button type="submit">Register</button>

      </form>

    </div>

  );

};

export default Register;

**Product Page (Product.js)**

The Product page will display information about a product, such as its name, description, and price.

**Program:**

import { useEffect, useState } from "react";

import axios from "axios";

const Products = () => {

  const [products, setProducts] = useState([]);

  useEffect(() => {

    axios.get("https://fakestoreapi.com/products")

      .then(response => setProducts(response.data))

      .catch(error => console.error("Error fetching products", error));

  }, []);

  return (

    <div>

      <h2>Product List</h2>

      <ul>

        {products.map((product) => (

          <li key={product.id}>

            <h3>{product.title}</h3>

            <img src={product.image} alt={product.title} width="100" />

            <p>{product.description.substring(0, 100)}...</p>

            <strong>${product.price}</strong>

          </li>

        ))}

      </ul>

    </div>

  );

};

export default Products;

const Navbar = () => {

  return (

    <nav>

      <ul>

        <li><Link to="/">Home</Link></li>

        <li><Link to="/login">Login</Link></li>

        <li><Link to="/register">Register</Link></li>

        <li><Link to="/product">Product</Link></li>

      </ul>

    </nav>

  );

};

export default Navbar;

**Step1**: Set up the project

Open your Terminal and run the following Command

* Cd Folder\_Name

**Creating the Components and Pages:**

We will break down the creation of individual components and pages, then wire them together using React Router.

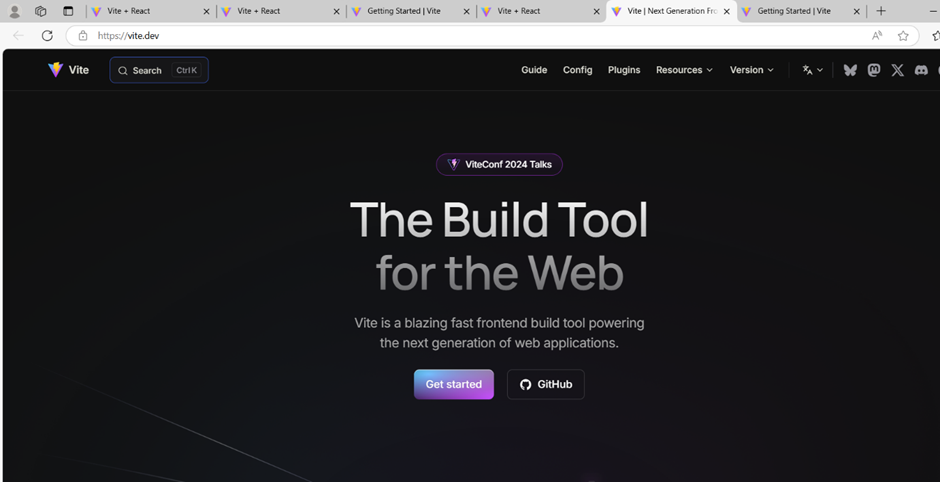
**Navbar Component (Navbar.js)**

The Navbar component will display navigation links to the Home, Login, Register, and Product pages. This component will be placed at the top of all pages

import React from 'react';

import { Link } from 'react-router-dom';

**Output:**



**Viva Questions:**

1.What happens when state is updated in React?

2.How is user data stored in the Online Store application?

3.How does the login page validate user credentials

4.Why do we use preventDefault() in form submissions