| Ex. No: 1 | Personal CV Using HTML |
|---|---|
| 13.07.2023 | |

**Aim:**

To create a CV using only HTML.

**Algorithm:**

1. Plan your CV.
2. Create a HTML document.
3. Add the required content.
4. Preview and Save

**Program:**

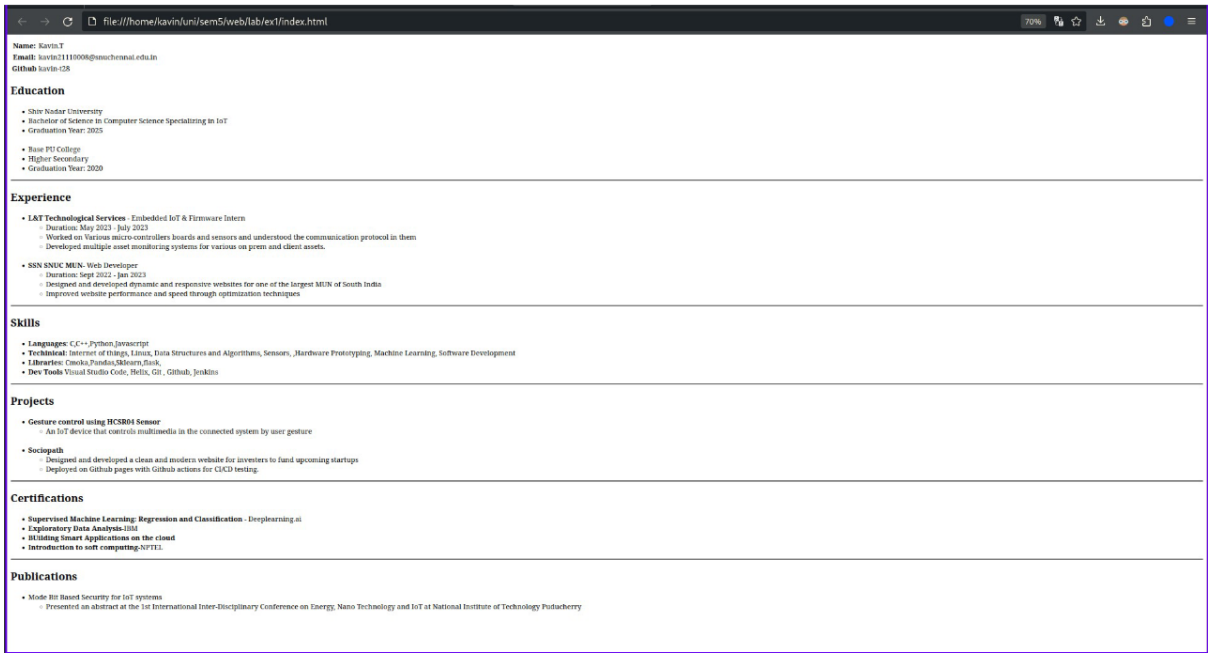
```
<!DOCTYPE html>
<html>

<head>
    <title>Resume</title>
</head>

<body>
    <table>
        <tr>
            <td colspan="2">
            </td>
        </tr>
        <tr>
            <th>Name:</th>
            <td>Kavin.T</td>
        </tr>
        <tr>
            <th>Email:</th>
            <td>kavin21110008@snuchennai.edu.in</td>
        </tr>
        <tr>
            <th>Github</th>
            <td>kavin-t28</td>
        </tr>
    </table>
    <h2>Education</h2>
    <ul>
        <li>Shiv Nadar University</li>
        <li>Bachelor of Science in Computer Science Specializing in
IoT</li>
        <li>Graduation Year: 2025</li>
        <br>
        <li>Base PU College </li>
        <li>Higher Secondary </li>
```

```
        <li>Graduation Year: 2020</li>
    </ul>
    <hr>
    <h2>Experience</h2>
    <ul>
        <li>
            <strong>L&T Technological Services</strong> - Embedded IoT &
Firmware
            Intern
            <ul>
                <li>Duration: May 2023 - July 2023</li>
                <li> Worked on Various micro-controllers boards and
sensors and
                    understood the communication protocol in them</li>
                <li>Developed multiple asset monitoring systems for
various on prem and
                    client assets.</li>
                <br>
            </ul>
        </li>
        <li>
            <strong>SSN SNUC MUN</strong>- Web Developer
            <ul>
                <li>Duration: Sept 2022 - Jan 2023 </li>
                <li>Designed and developed dynamic and responsive websites
for one
                    of the largest MUN of South India</li>
                <li>Improved website performance and speed through
optimization
                    techniques</li>
            </ul>
        </li>
    </ul>
    <hr>
    <h2>Skills</h2>
    <ul>
        <li><strong>Languages</strong>: C,C++,Python,Javascript</li>
        <li><strong>Techinical:</strong> Internet of things, Linux, Data
Structures
            and Algorithms, Sensors, ,Hardware Prototyping,
            Machine Learning, Software Development</li>
        <li><strong>Libraries:</strong> Cmoka,Pandas,Sklearn,flask,</li>
        <li><strong>Dev Tools</strong> Visual Studio Code, Helix, Git ,
Github,
            Jenkins</li>
    </ul>
    <hr>
    <h2>Projects</h2>
    <ul>
        <li>
            <strong>Gesture control using HCSR04 Sensor</strong>
            <ul>
                <li>An IoT device that controls multimedia in the
connected system by
```

```
                    user gesture</li>
            </ul>
        </li>
        <br>
        <li>
            <strong>Sociopath</strong>
            <ul>
                <li>Designed and developed a clean and modern website for
investers
                    to fund upcoming startups</li>
                <li>Deployed on Github pages with Github actions for CI/CD
                    testing.</li>
            </ul>
        </li>
    </ul>
    <hr>
    <h2>Certifications</h2>
    <ul>
        <li><strong>Supervised Machine Learning: Regression and
Classification
            </strong>- Deeplearning.ai</li>
        <li><strong>Exploratory Data Analysis</strong>-IBM</li>
        <li><strong>BUilding Smart Applications on the cloud</strong></li>
        <li><strong>Introduction to soft computing</strong>-NPTEL</li>
    </ul>
    <hr>
    <h2>Publications</h2>
    <ul>
        <li>Mode Bit Based Security for IoT systems</li>
        <ul>
            <li>Presented an abstract at the 1st International Inter-
Disciplinary
                Conference on Energy, Nano Technology and IoT at
                National Institute of Technology Puducherry</li>
        </ul>
    </ul>
</body>

</html>
```

**Output:**

Github Link: https://github.com/kavin-t28/CS3809-Web-Technologies-Lab



file:///home/kavin/uni/sem5/web/lab/ex1/index.html

**Name:** Kavin.T
**Email:** kavin21110008@snuchennai.edu.in
**Github** kavin-t28

## Education

- Shiv Nadar University
- Bachelor of Science in Computer Science Specializing in IoT
- Graduation Year: 2025

- Base PU College
- Higher Secondary
- Graduation Year: 2020

---

## Experience

- **L&T Technological Services** - Embedded IoT & Firmware Intern
  - Duration: May 2023 - July 2023
  - Worked on Various micro-controllers boards and sensors and understood the communication protocol in them
  - Developed multiple asset monitoring systems for various on prem and client assets.

- **SSN SNUC MUN**- Web Developer
  - Duration: Sept 2022 - Jan 2023
  - Designed and developed dynamic and responsive websites for one of the largest MUN of South India
  - Improved website performance and speed through optimization techniques

---

## Skills

- **Languages**: C,C++,Python,Javascript
- **Technical**: Internet of things, Linux, Data Structures and Algorithms, Sensors, ,Hardware Prototyping, Machine Learning, Software Development
- **Libraries**: Cmoka,Pandas,Sklearn,flask,
- **Dev Tools** Visual Studio Code, Helix, Git , Github, Jenkins

---

## Projects

- **Gesture control using HCSR04 Sensor**
  - An IoT device that controls multimedia in the connected system by user gesture

- **Sociopath**
  - Designed and developed a clean and modern website for investers to fund upcoming startups
  - Deployed on Github pages with Github actions for CI/CD testing.

---

## Certifications

- **Supervised Machine Learning: Regression and Classification** - Deeplearning.ai
- **Exploratory Data Analysis**-IBM
- **BUilding Smart Applications on the cloud**
- **Introduction to soft computing**.NPTEL.

---

## Publications

- Mode Bit Based Security for IoT systems
  - Presented an abstract at the 1st International Inter-Disciplinary Conference on Energy, Nano Technology and IoT at National Institute of Technology Puducherry

**Result:**
Therefore, we've successfully created a CV using HTML.

| Ex. No: 2 | CSS enabled CV |
|---|---|
| 20.07.2023 | |

**Aim:**
To apply CSS to the Assignment done for LAB 1

**Algorithm:**
1. Create a CSS file
2. Link the CSS file to the HTML file
3. Define Styles
4. Apply Classes and IDs
5. Preview and Refine.

**Program:**
```
<!DOCTYPE html>
<html>

<head>
    <title>Resume</title>
    <link rel="stylesheet" href="style1.css">
</head>

<body>
    <div class="container">
        <table>
            <tr>
                <th>Name:</th>
                <td>Kavin.T</td>
            </tr>
            <tr>
                <th>Email:</th>
                <td>kavin21110008@snuchennai.edu.in</td>
            </tr>
            <tr>
                <th>Github</th>
                <td><a href="https://github.com/kavin-t28" target="_blank">kavin-
t28</a></td>
            </tr>
        </table>
        <h2>Education</h2>
        <ul>
            <li>Shiv Nadar University</li>
            <li>Bachelor of Science in Computer Science Specializing in IoT</li>
            <li>Graduation Year: 2025</li>
            <br>
            <li>Base PU College</li>
            <li>Higher Secondary</li>
            <li>Graduation Year: 2020</li>
        </ul>
        <hr>
        <h2>Experience</h2>
        <ul>
            <li>
```

```
                <strong>L&T Technological Services</strong> - Embedded IoT &
Firmware
                Intern
                <ul>
                    <li>Duration: May 2023 - July 2023</li>
                    <li>Worked on Various micro-controllers boards and sensors and
                        understood the communication protocol in them</li>
                    <li>Developed multiple asset monitoring systems for various
on-prem
                        and client assets.</li>
                    <br>
                </ul>
            </li>
            <li>
                <strong>SSN SNUC MUN</strong> - Web Developer
                <ul>
                    <li>Duration: Sept 2022 - Jan 2023</li>
                    <li>Designed and developed dynamic and responsive websites for
one of
                        the largest MUN of South India</li>
                    <li>Improved website performance and speed through
optimization
                        techniques</li>
                </ul>
            </li>
        </ul>
        <hr>
        <h2>Skills</h2>
        <ul>
            <li><strong>Languages</strong>: C, C++, Python, Javascript</li>
            <li><strong>Technical</strong>: Internet of things, Linux, Data
Structures
                and Algorithms, Sensors, Hardware Prototyping,
                Machine Learning, Software Development</li>
            <li><strong>Libraries</strong>: Cmoka, Pandas, Sklearn, Flask,</li>
            <li><strong>Dev Tools</strong>: Visual Studio Code, Helix, Git,
Github,
                Jenkins</li>
        </ul>
        <hr>
        <h2>Projects</h2>
        <ul>
            <li>
                <strong>Gesture control using HCSR04 Sensor</strong>
                <ul>
                    <li>An IoT device that controls multimedia in the connected
system by
                        user gesture</li>
                </ul>
            </li>
            <br>
            <li>
                <strong>Sociopath</strong>
                <ul>
                    <li>Designed and developed a clean and modern website for
investors to
                        fund upcoming startups</li>
                    <li>Deployed on Github pages with Github actions for CI/CD
                        testing.</li>
```

```
                </ul>
            </li>
        </ul>
        <hr>
        <h2>Certifications</h2>
        <ul>
            <li><strong>Supervised Machine Learning: Regression and Classification
                </strong>- Deeplearning.ai</li>
            <li><strong>Exploratory Data Analysis</strong> - IBM</li>
            <li><strong>Building Smart Applications on the cloud</strong></li>
            <li><strong>Introduction to soft computing</strong> - NPTEL</li>
        </ul>
        <hr>
        <h2>Publications</h2>
        <ul>
            <li>Mode Bit Based Security for IoT systems</li>
            <ul>
                <li>Presented an abstract at the 1st International Inter-
Disciplinary
                    Conference on Energy, Nano Technology, and IoT at
                    National Institute of Technology Puducherry</li>
            </ul>
        </ul>
    </div>
</body>

</html>
```

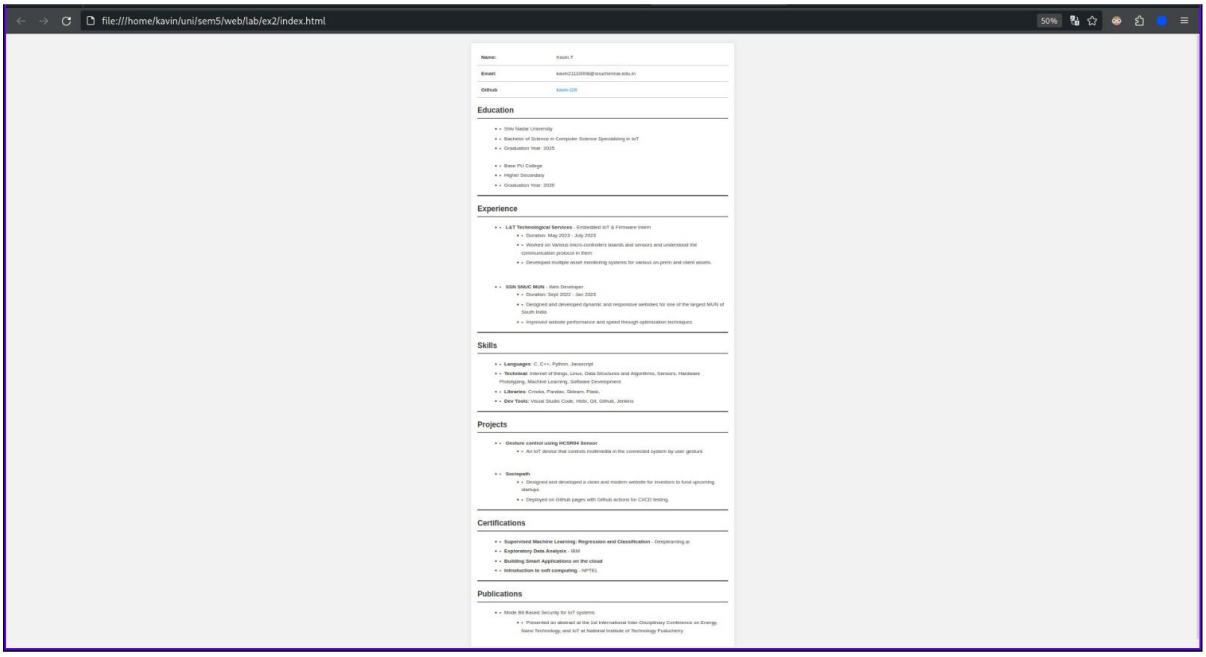
## CSS:

```
/* Global Styles */
body {
    font-family: 'Arial', sans-serif;
    line-height: 1.6;
    margin: 30px;
    background-color: #f2f2f2;
    color: #333;
}

.container {
    max-width: 800px;
    margin: 0 auto;
    padding: 20px;
    background-color: #fff;
    box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
}

header {
    text-align: center;
    background-color: #333;
    color: #fff;
    padding: 20px;
}

h1 {
    margin: 0;
    font-size: 32px;
}

table {
```

```
    width: 100%;
    border-collapse: collapse;
    margin-bottom: 20px;
}

th,
td {
    padding: 12px;
    text-align: left;
    border-bottom: 1px solid #ccc;
}

th {
    width: 30%;
    font-weight: bold;
}

h2 {
    margin-top: 20px;
    border-bottom: 2px solid #333;
    padding-bottom: 5px;
    font-size: 24px;
}

ul {
    list-style-type: disc;
    margin-left: 30px;
    margin-bottom: 20px;
}

ul li {
    margin-bottom: 5px;
}

ul li:before {
    content: "•";
    color: #333;
    margin-right: 10px;
}

strong {
    font-weight: bold;
}

/* Styling links */
a {
    color: #007BFF;
    text-decoration: none;
}

a:hover {
    text-decoration: underline;
}
```

**Output:**

Github Link: https://github.com/kavin-t28/CS3809-Web-Technologies-Lab




**Result:**
Therefore, we've successfully implemented the creation of Thread using C.

| Ex. No: 3 | Form Making and Validation using JavaScript |
|---|---|
| 27.07.2023 | |

**Aim:**

To create a Form with usual form elements in JavaScript including the Alert(), Confirm(), and Response() functions. Additionally, validate the form elements.

**Algorithm:**

1. Create HTML and CSS file
2. Setup JavaScript in the HTML file
3. Access the form elements
4. Setup a validation logic
5. Use the Alert(), confirm() and response() functions for validation
6. Display validation results

**Program:**

HTML

```html
<!DOCTYPE html>
<html>
<head>
  <title>Fancy Form with Validation</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container">
    <form id="myForm" onsubmit="return validateForm()">
      <div class="form-group">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
      </div>

      <div class="form-group">
        <label>Gender:</label>
        <input type="radio" name="gender" value="male"> Male
        <input type="radio" name="gender" value="female"> Female
      </div>

      <div class="form-group">
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>
      </div>

      <div class="form-group">
        <label>Languages:</label>
        <input type="checkbox" name="language" value="english"> English
        <input type="checkbox" name="language" value="spanish"> Spanish
        <input type="checkbox" name="language" value="french"> French
      </div>

      <div class="form-group">
        <label for="message">Message:</label>
        <textarea id="message" name="message" required></textarea>
      </div>

      <div class="form-group">
        <label for="birthdate">Birthdate:</label>
        <input type="date" id="birthdate" name="birthdate" required>
      </div>

      <div class="form-group">
        <label for="country">Country:</label>
        <select id="country" name="country" required>
          <option value="" disabled selected>Select your country</option>
          <option value="usa">USA</option>
          <option value="canada">Canada</option>
          <option value="uk">UK</option>
          <option value="australia">Australia</option>
        </select>
      </div>

      <div class="form-group">
        <label for="avatar">Avatar:</label>
        <input type="file" id="avatar" name="avatar" accept="image/*">
      </div>

      <button type="submit">Submit</button>
    </form>
  </div>
```

CSS

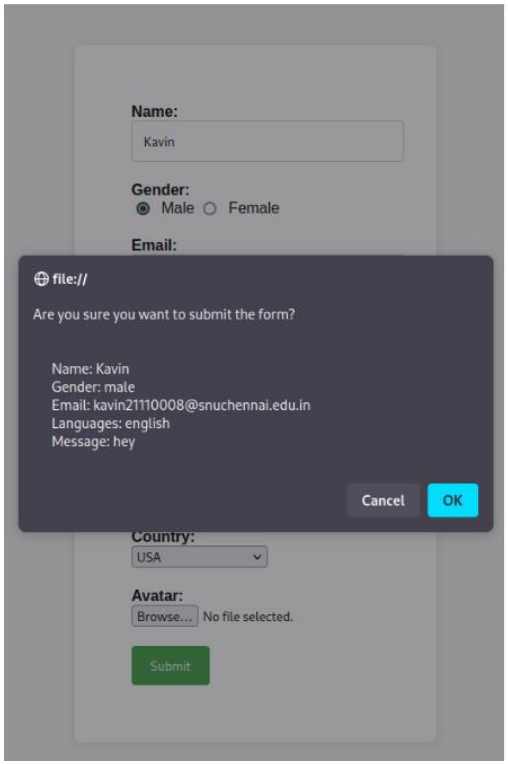
```
# style.css > button:hover
body {
    font-family: Arial, sans-serif;
    margin: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    background-color: #f5f5f5;
}

.container {
    background-color: #fff;
    border-radius: 8px;
    padding: 60px;
    box-shadow: 0px 2px 10px rgba(0, 0, 0, 0.1);
}

.form-group {
    margin-bottom: 20px;
}

label {
    display: block;
    font-weight: bold;
}

input[type="text"],
input[type="email"],
textarea {
    width: 100%;
    padding: 12px;
    border: 1px solid #ccc;
    border-radius: 4px;
    transition: border-color 0.3s;
}

input[type="text"]:focus,
input[type="email"]:focus,
textarea:focus {
    border-color: #4CAF50;
}

input[type="radio"],
input[type="checkbox"] {
    margin-right: 8px;
}

button {
    padding: 12px 20px;
    background-color: #4CAF50;
    color: #fff;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #45a049;
}
```

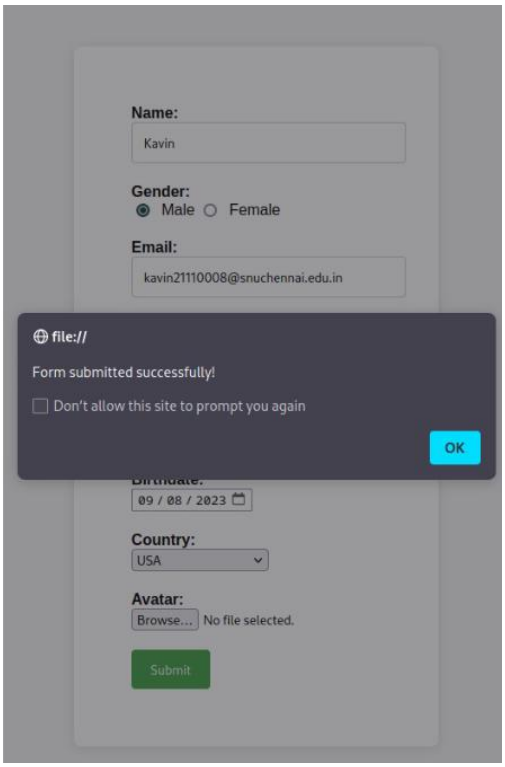
Javascript

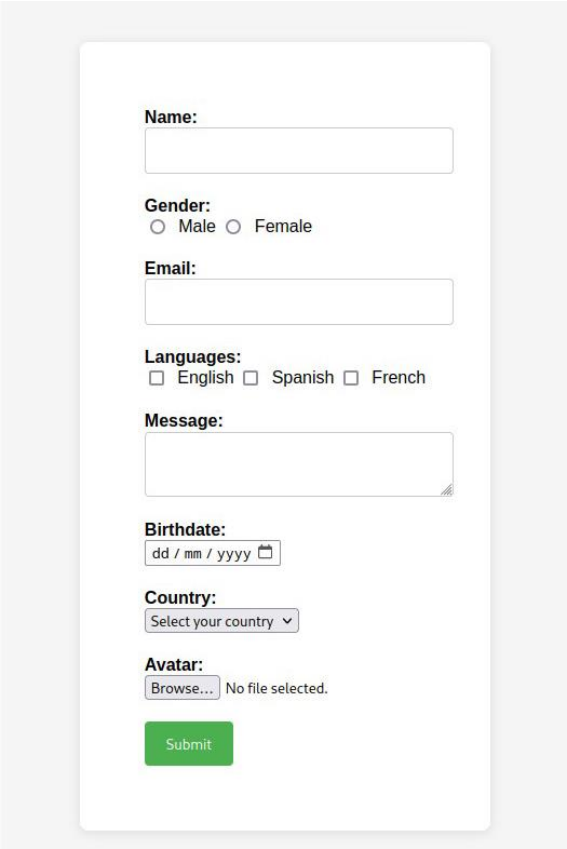
```
script.js > validateForm > confirmationMessage
function validateForm() {
    const name = document.getElementById("name").value;
    const gender = document.querySelector('input[name="gender"]:checked');
    const email = document.getElementById("email").value;
    const languages = document.querySelectorAll('input[name="language"]:checked');
    const message = document.getElementById("message").value;

    if (!name || !gender || !email || languages.length === 0 || !message) {
        alert("Please fill in all the required fields.");
        return false;
    }

    const confirmationMessage = `
        Name: ${name}
        Gender: ${gender.value}
        Email: ${email}
        Languages: ${[...languages].map(lang => lang.value).join(", ")}
        Message: ${message}
    `;

    const confirmed = confirm("Are you sure you want to submit the form?\n\n" + confirmationMessage);

    if (confirmed) {
        alert("Form submitted successfully!");
    } else {
        alert("Form submission canceled.");
    }

    return confirmed;
}
```

**Output:**

Github Link: https://github.com/kavin-t28/CS3809-Web-Technologies-Lab



**Result:**
Therefore, created a form and validated it using javascript.

| Ex. No: 4 | Angular based App creation |
|---|---|
| 09.08.2023 | |

**Aim:**

To Create an App using ANGULAR with Components, Binding, and Services usage.

**Algorithm:**

1. Setup angular using the ng serve command
2. Create all the required components.
3. Organize the app structure.
4. Implement the services that are needed.
5. Define component HTML templates with data binding to display dynamic content
6. Enable component communication using input/output properties and event binding.
7. Apply CSS styles to components, optimize for performance, and deploy the app.

**Program:**

**Component code:**

```
import { Component, OnInit } from '@angular/core';
import { CalculationService } from '../calculation.service';
@Component({
selector: 'app-calculator',
templateUrl: './calculator.component.html',
styleUrls: ['./calculator.component.css']
})
export class CalculatorComponent implements OnInit {
num1: number = 0;
num2: number = 0;
result: number = 0;
constructor(private calculationService: CalculationService) { }
ngOnInit(): void {
}
calculateSum(): void {
this.result = this.calculationService.calculateSum(this.num1, this.num2);
}
}
```

**Service code:**

```
import { Injectable } from '@angular/core';
@Injectable({
providedIn: 'root'
})
export class CalculationService {
constructor() { }
calculateSum(num1: number, num2: number): number {
return num1 + num2;
}
}
```

**app.component.ts:**

```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
title = 'sum-calculator';
}
```

**Component html code:**

```
<div>
<h2>Simple Sum Calculator</h2>
<label for="num1">Number 1:</label>
<input type="number" id="num1" [(ngModel)]="num1">
<br>
<label for="num2">Number 2:</label>
<input type="number" id="num2" [(ngModel)]="num2">
<br>
<button (click)="calculateSum()">Calculate Sum</button>
<br>
<p>Result: {{ result }}</p>
</div>
```

**Output:**

Github Link:

https://github.com/kavin-t28/CS3809-Web-Technologies-Lab

-


**Result:**

Therefore, we've successfully created a simple react app.

| Ex. No: 5 | React based App Development |
|---|---|
| 23.02.2023 | |

**Aim:**

To Create an App using React with Components, Rendering, and Data Sharing.

**Algorithm:**

1. Create a new React project using a tool like Create React App.
2. Build individual components to represent various parts of your app.
3. Arrange components hierarchically, defining parent-child relationships.
4. Define the UI using JSX within components for rendering.
5. Pass data between components using props or consider state management for more complex data sharing.
6. Implement local or global state management for dynamic data and user interactions.
7. Style components using CSS, CSS modules, or CSS-in-JS libraries while keeping styling separate from logic.

**Program:**

```
import React, { useState } from 'react';
import './App.css';
import TodoList from './components/TodoList';

function App() {
  const [todos, setTodos] = useState([]);
  const [text, setText] = useState('');

  const addTodo = () => {
    if (text.trim() !== '') {
      const newTodo = { id: Date.now(), text, completed: false };
      setTodos([...todos, newTodo]);
      setText('');
    }
  };

  const toggleComplete = (id) => {
    const updatedTodos = todos.map((todo) =>
      todo.id === id ? { ...todo, completed: !todo.completed } : todo
    );
    setTodos(updatedTodos);
  };

  const deleteTodo = (id) => {
    const updatedTodos = todos.filter((todo) => todo.id !== id);
    setTodos(updatedTodos);
  };

  return (
    <div className="App">
```

```
      <h1>Todo List App</h1>
      <input
        type="text"
        value={text}
        onChange={(e) => setText(e.target.value)}
        placeholder="Enter a new task"
      />
      <button onClick={addTodo}>Add</button>
      <TodoList
        todos={todos}
        toggleComplete={toggleComplete}
        deleteTodo={deleteTodo}
      />
    </div>
  );
}
export default App;
```

**Todoitem.js Component**

```
import React from 'react';

const TodoItem = ({ todo, toggleComplete, deleteTodo }) => {
  return (
    <div className={`todo-item ${todo.completed ? 'completed' : ''}`}>
      <input
        type="checkbox"
        checked={todo.completed}
        onChange={() => toggleComplete(todo.id)}
      />
      <p>{todo.text}</p>
      <button onClick={() => deleteTodo(todo.id)}>Delete</button>
    </div>
  );
};

export default TodoItem;
```

**TodoList.js Component**

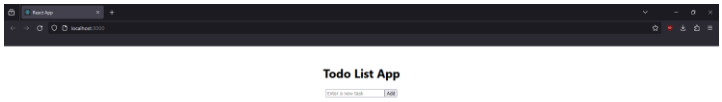
```
import React from 'react';
import TodoItem from './TodoItem';

const TodoList = ({ todos, toggleComplete, deleteTodo }) => {
  return (
    <div className="todo-list">
      {todos.map((todo) => (
        <TodoItem
          key={todo.id}
          todo={todo}
          toggleComplete={toggleComplete}
          deleteTodo={deleteTodo}
        />
      ))}
```

```
    </div>
  );
};

export default TodoList;
```

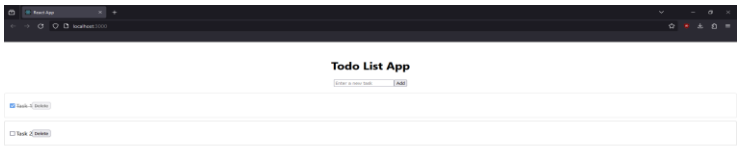
**Output:**

Github Link:

https://github.com/kavin-t28/CS3809-Web-Technologies-Lab




---




**Result:**
Therefore, we've successfully created a simple react app.

| Ex. No: 6 | Web Server Creation using NodeJS |
|---|---|
| 21.09.2023 | |

**Aim:** To Create a Web Server offering basic web service(s) to the front-end.
**Algorithm:**

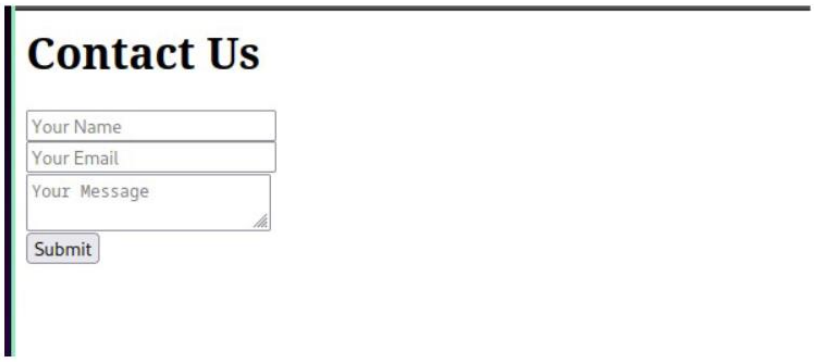
1. Ensure you have Node.js installed on your system.
2. Develop a JavaScript file (e.g., `server.js`) for your web server.
3. In `server.js`, require Node.js's built-in `http` module using `require('http')`.
4. Use the `http.createServer()` method to create an HTTP server, specifying a request handling function.
5. Inside the request handling function, use the `request` and `response` objects to define how your server should respond to different routes and HTTP methods.
6. Test your web server using tools like cURL or Postman. Debug and refine your route handling as needed.
7. Optionally, configure the web server to serve static HTML, CSS, and JavaScript files if your front-end includes them, using the `fs` (file system) module.

**Program:**
```
const http = require('http');
const url = require('url');
const fs = require('fs');
// Create an HTTP server
const server = http.createServer((req, res) => {
// Parse the request URL
const parsedUrl = url.parse(req.url, true);
const pathname = parsedUrl.pathname;
// Set the response header with a status code and content type
res.setHeader('Content-Type', 'text/html');
if (pathname === '/') {
// Serve the homepage
fs.readFile('index.html', (err, data) => {
if (err) {
res.writeHead(500);
res.end('Error reading the file');
} else {
res.writeHead(200);
res.end(data);
}
});
} else if (pathname === '/about') {
// Serve an about page
res.writeHead(200);
res.end('<h1>About Us</h1>');
} else if (pathname === '/contact') {
// Serve a contact form
if (req.method === 'GET') {
res.writeHead(200);
res.end(`
<h1>Contact Us</h1>
```

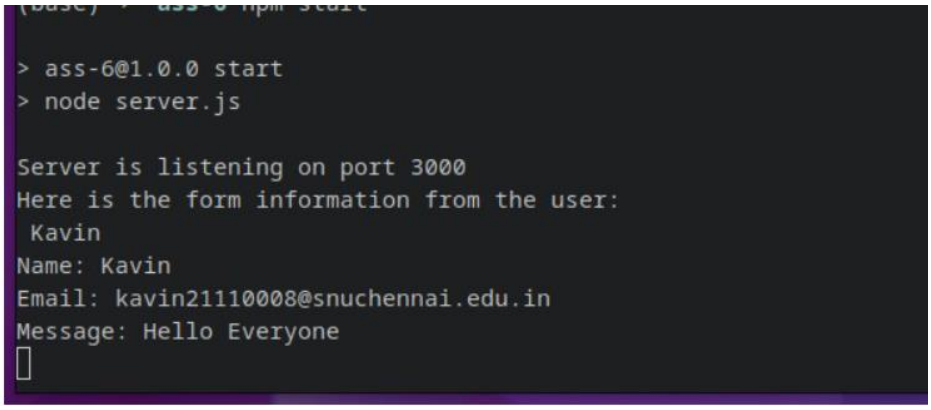
```
<form method="post" action="/contact">
<input type="text" name="name" placeholder="Your Name"><br>
<input type="email" name="email" placeholder="Your Email"><br>
<textarea name="message" placeholder="Your Message"></textarea><br>
<input type="submit" value="Submit">
</form>
`);
} else if (req.method === 'POST') {
// Handle form submission
let body = '';
req.on('data', (chunk) => {
body += chunk.toString();
});
req.on('end', () => {
const formData = new URLSearchParams(body);
const name = formData.get('name');
const email = formData.get('email');
const message = formData.get('message');
// The form data can be stored in a
console.log("Here is the form information from the user: \n", name);
console.log('Name:', name);
console.log('Email:', email);
console.log('Message:', message);
res.writeHead(200);
res.end('<h1>Thank you for your message!</h1>');
});
}
} else {
// Handle 404 Not Found
res.writeHead(404);
res.end('<h1>404 Not Found</h1>');
}
});
// Listen on port 3000
const port = 3000;
server.listen(port, () => {
console.log(`Server is listening on port ${port}`);
});
```

**Output:**

Github Link: https://github.com/kavin-t28/CS3809-Web-Technologies-Lab


Server Side output:




**Result:**
Therefore, we've successfully implemented a web server backend using NodeJS .

| Ex. No: 7 | Routing Implementation using ExpressJS |
|---|---|
| 28.09.2023 | |

**Aim:** To Implement the routing feature(s) using the ExpressJS.

**Algorithm:**

1. Include the required header files thread creation and sleep() function.
2. Write a function that executes as a thread when it is called. (sleep – print - return)
3. thread_id is declared to identify the thread in the system, we call pthread_create() function to create a thread.
4. The pthread_join() function for threads is the equivalent of wait() for processes. A call to pthread_join blocks the calling thread until the thread with identifier equal to the first argument terminates.

**Program:**

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const port = 3000; // You can change this to any port you prefer
app.set('view engine', 'ejs'); // Use EJS as the template engine
app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static(__dirname + '/public'));
const posts = []; // Simulated database for storing blog posts
// Home page route
app.get('/', (req, res) => {
res.render('home', { posts });
});
// New post form route
app.get('/newpost', (req, res) => {
res.render('newpost');
});
// Create a new post route
app.post('/create', (req, res) => {
const { title, content } = req.body;
const newPost = { title, content };
posts.push(newPost);
res.redirect('/');
});
// JSON API endpoint to fetch all posts
app.get('/api/posts', (req, res) => {
res.json(posts);
});
// Start the server
app.listen(port, () => {
console.log(`Server is running on port ${port}`);
});
```

**Output:**

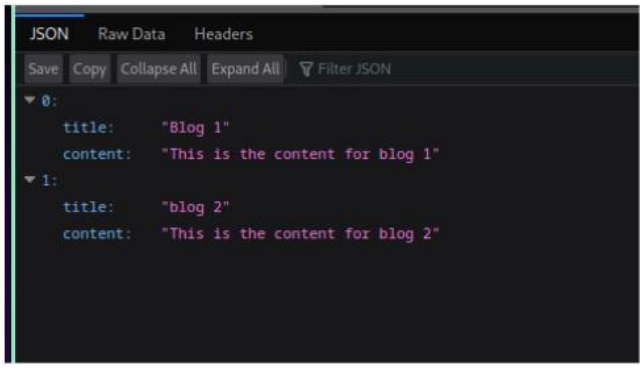
Github Link: https://github.com/kavin-t28/CS3809-Web-Technologies-Lab

1. Initial webpage when the server is started.


2. New blog creation page


3. When we create new blogs the title of the blog gets updated in the home page


4. All the blog content for that session can be assesed in the /api/posts route in json formate



**Result:**
Therefore, we've successfully implemented a basic routing implementation using Express JS

| Ex. No: 8 | Building a REST API with Express, Node, and MongoDB |
|---|---|
| 05.10.2023 | |

**Aim:**

To create a REST API with express node and mongoDB.

**Algorithm:**

1. Ensure Node.js, npm, and MongoDB are installed on your system.
2. Create a project directory and set up its structure.
3. Use npm to install necessary packages, including Express and a MongoDB driver like Mongoose.
4. Create API routes and handlers for various HTTP methods to manage different data operations.
5. Establish a connection to your MongoDB database using the installed MongoDB driver.
6. Define data models and schemas to structure the data you'll work with in the MongoDB database.
7. Implement Create, Read, Update, and Delete (CRUD) operations in your API routes for database interaction.
8. Test API endpoints using tools like Postman. Debug, refine, and handle errors as needed.

**Program:**

**1) Index.js (server):**

- **Connecting to mongo dB, mongoose and express**

```
const express=require("express");
const mongoose=require("mongoose");
const url='mongodb://127.0.0.1:27017/studentDB';
const app=express();
mongoose.connect(url,{
useNewUrlParser:true
})
const con =mongoose.connection
con.on('open',function(){
console.log("connected to mongodb database")
})
app.use(express.json())
const studentRouter=require('./routes/students')
app.use('/students',studentRouter)
app.listen(3000,function(){
console.log("Server started")
})
```

**2) students.js**

- **Creating routes (GET, PATCH, GET single object by ID, POST)**

```
const express=require("express");
const router=express.Router()
const Student=require('../models/student')
```

```
router.get('/',async(req,res)=>{
try{
const stud=await Student.find()
res.json(stud)
}catch(err){
res.send("Error")
}
res.send("Get request made")
})
router.get('/:id',async(req,res)=>{
try{
const stud1=await Student.findById(req.params.id)
res.json(stud1)
}catch(err){
res.send("Error")
}
})
router.patch('/:id', async (req, res) => {
try {
const studPatch = await Student.findById(req.params.id);
studPatch.name = req.body.name;
const s = await studPatch.save();
res.json(studPatch);
} catch (err) {
res.status(500).send("Error"); // Sending an error response
}
});
router.post('/',async(req,res)=>{
const student=new Student({
name: req.body.name,
course: req.body.course
})
try{
const s=await student.save()
res.json(s)
}catch(err){
res.send("Error")
}
})
module.exports=router
```

**3) student.js**

- **Creating mongoose schema for a single object (student here)**

```
const mongoose=require("mongoose")
const studSchema=new mongoose.Schema({
name:{
type: String,
required:true
},
course:{
type: String,
required:true
}
})
```

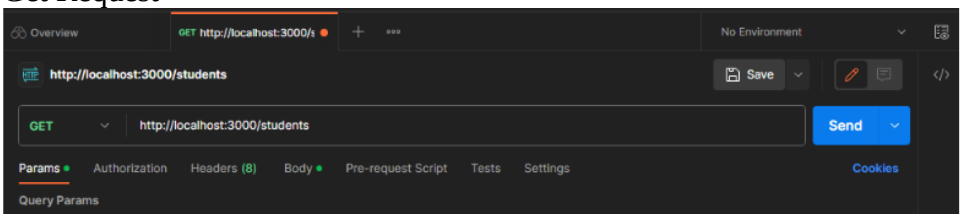
```
module.exports=mongoose.model('Student',studSchema)
```

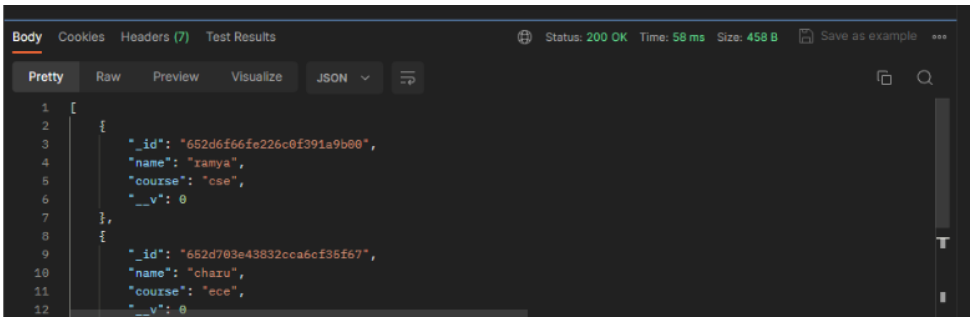
**Output:**

Github Link:

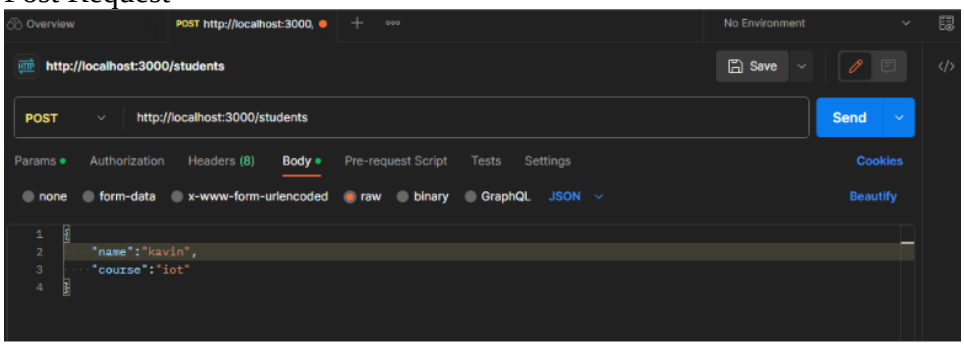
Request made by the postman API:

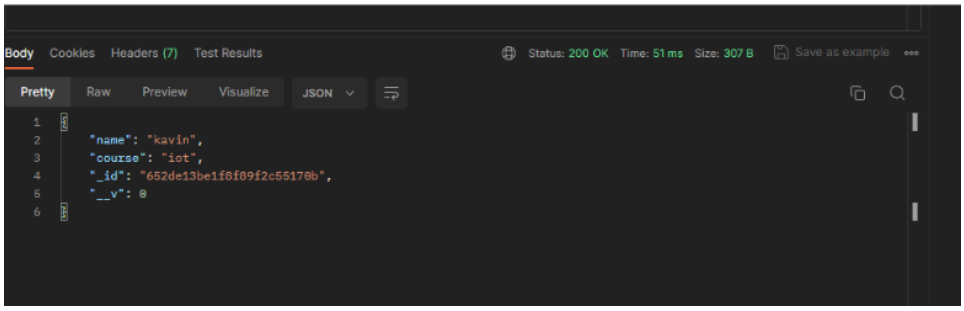
1. Get Request



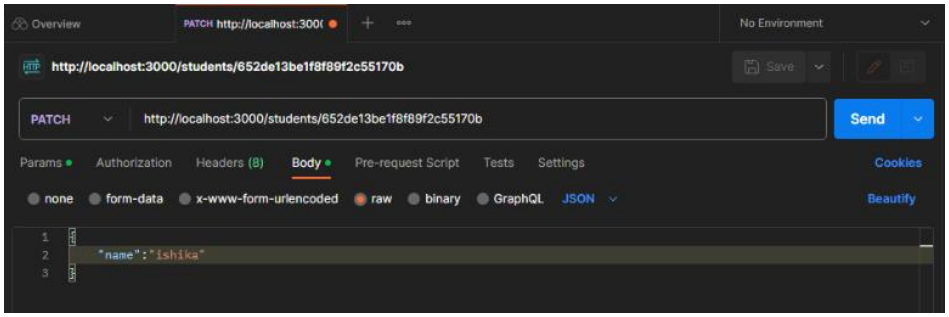



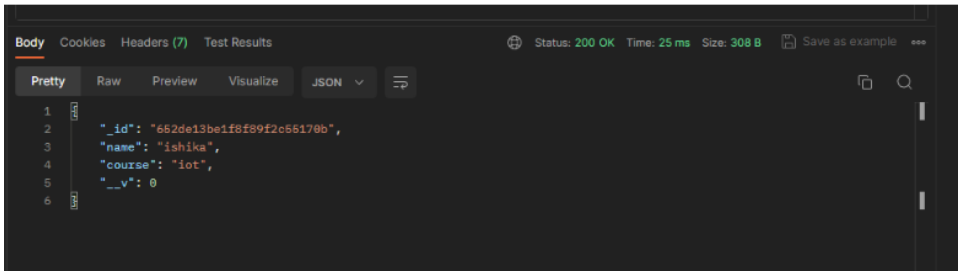

2. Post Request



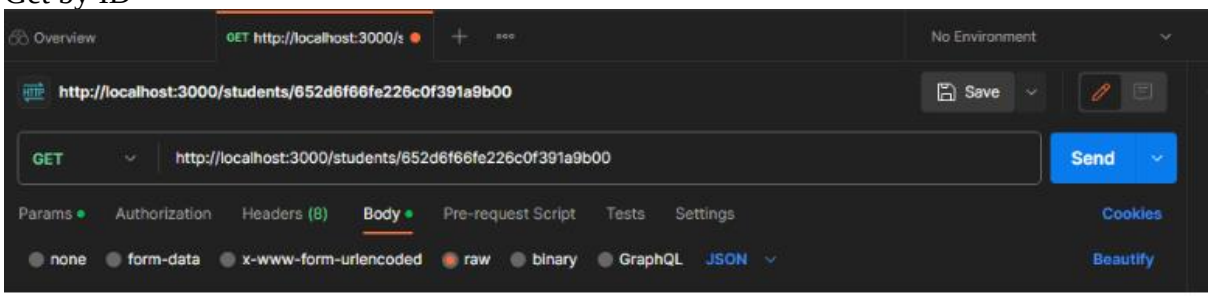



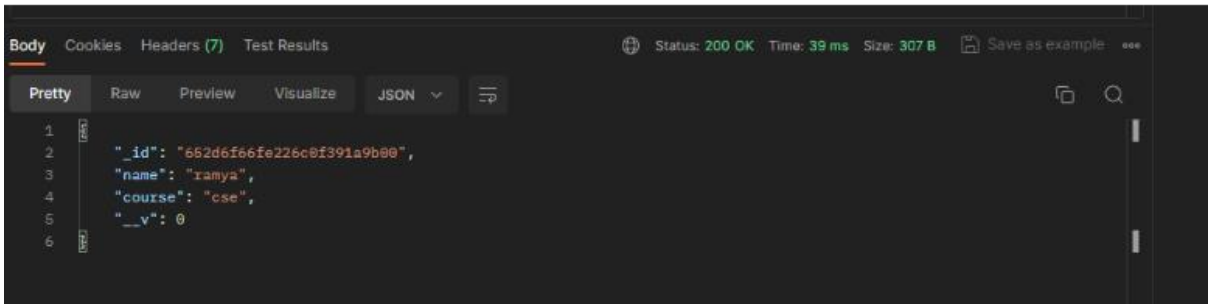

3. Patch Request




4. Get by ID




**Result:**
Therefore, we've successfully implemented the creation of a REST API with express node and mongoDB.