Lab Exercise 1: Resume Webpage using HTML and CSS

Question

Create an HTML and CSS webpage that captures a resume (preferably yours) for a fresher full-stack developer at an IT company. Decide on the subdivisions and content. Push your content to GitHub and maintain proper version control with clear README files.

Design / Plan

The webpage is designed with semantic HTML5 elements for clarity and SEO-friendly structure. It consists of the following sections:

- Header - Name and tagline
- About Me - Short profile summary
- Skills - Categorized technical stack
- Projects - Portfolio projects with GitHub/Live links
- Education - Academic background
- Experience - Internship and professional exposure
- Certifications - Relevant courses and achievements
- Footer - Contact and social links

CSS provides a minimalistic, modern aesthetic with consistent padding, section styling, and accent colors.

Code

HTML (resume.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-
     scale=1.0" />
  <title>Ashwin V - Full Stack Developer</title>
  <link rel="stylesheet" href="styleResume.css" />
</head>
<body>
  <header>
    <h1>Ashwin V</h1>
    <p>Full Stack Developer | AI & Data Science Enthusiast</p>
  </header>
```

```
<section id="about">
  <h2>About Me</h2>
  <p>A passionate developer with hands-on experience building
     scalable full-stack applications. Strong interest in
     backend systems, data pipelines, and AI integrations.</p>
</section>

<section id="skills">
  <h2>Skills</h2>
  <ul>
    <li><strong>Frontend:</strong> HTML, CSS, JavaScript, React
       , Flutter</li>
    <li><strong>Backend:</strong> FastAPI, Django, Flask</li>
    <li><strong>Databases:</strong> PostgreSQL, MongoDB, MySQL<
       /li>
    <li><strong>Tools:</strong> Git, Docker, Firebase, Postman<
       /li>
  </ul>
</section>

<section id="projects">
  <h2>Projects</h2>
  <div class="project">
    <h3>Facial Recognition Attendance System</h3><a href="https
       ://github.com/ashvp/face-attendance" target="_blank">
       GitHub Repo</a>
    <ul>
      <li>Engineered facial recognition backend supporting 500+
          users with 99.6% accuracy using FaceNet embeddings</
         li>
      <li>Built 15+ production-grade REST APIs with offline
         sync protocol for 100+ mobile devices</li>
      <li>Integrated PostgreSQL vector search with cosine
         similarity for efficient face matching</li>
    </ul>

  </div>
  <div class="project">
    <h3>AI-Powered Question Paper Generator</h3><a href="https
       ://question-paper-generator-nine.vercel.app/" target="
       _blank">Live Demo</a>
    <ul>
      <li>Built fully-deployed AI system generating
         customizable question papers and answer keys from
         uploaded PDFs</li>
      <li>Designed prompt-chained backend using LangChain for
         context extraction and question generation with
         marking schemes</li>
      <li>Developed responsive frontend integrated via REST API
          with automated deployment and health monitoring</li>
```

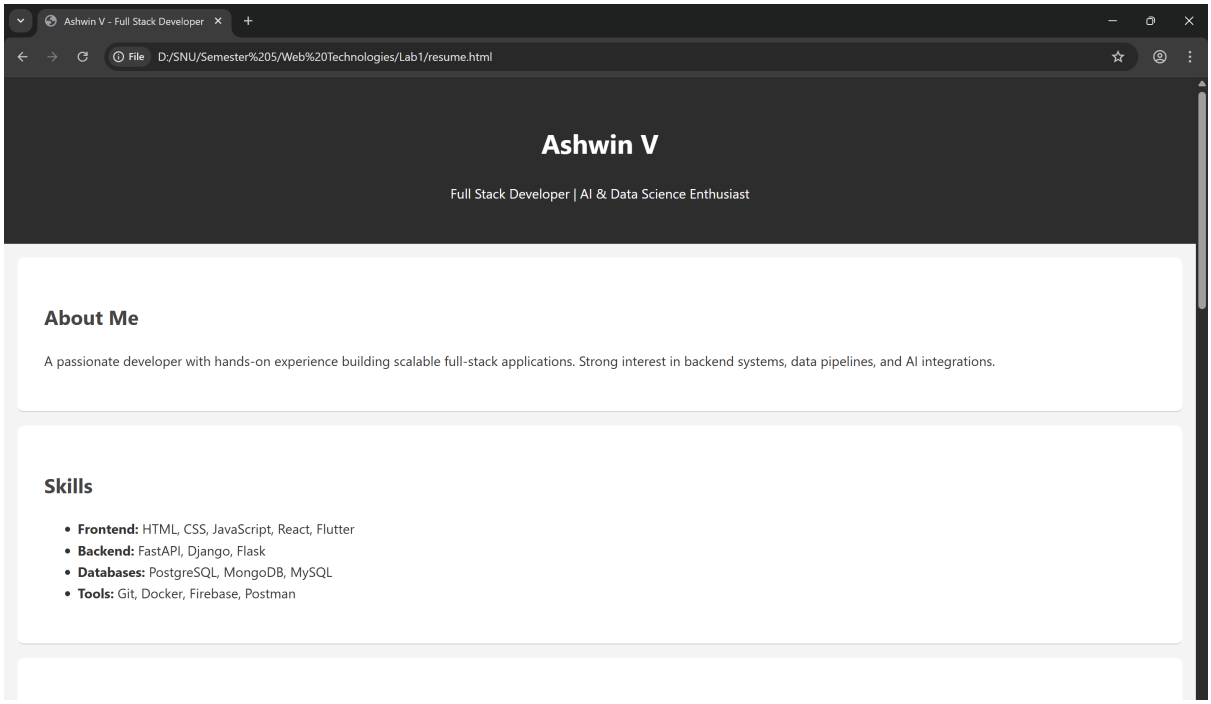
```
    </div>
  </section>

  <section id="education">
    <h2>Education</h2>
    <p><strong>B.Tech in AI & Data Science</strong>, Shiv Nadar
       University, 2027 (Expected)<br />CGPA: 8.8/10</p>
    <p><strong>B.S in Programming & Data Science</strong>, Indian
        Institute of Technology, Madras, 2027 (Expected)<br />
       CGPA: 8.2/10</p>
  </section>

  <section id="experience">
    <h2>Experience</h2>
    <p><strong>Backend Developer Intern</strong>, Sudha
       Gopalakrishnan Brain Centre, IIT Madras (July 2025 -
       Present)</p>
<ul>
  <li>Contributed to 6+ Django microservices powering internal
     neuroscience research tools used across multiple labs.</li>
  <li>Integrated reverse proxy logic to unify access to internal
     services under a central `/lab/` route using NGINX.</li>
  <li>Built and deployed Model Context Protocol (MCP) tools to
     Anthropic's Claude for testing model inference pipelines and
      agentic workflows.</li>
</ul>

  </section>

  <section id="certifications">
    <h2>Certifications</h2>
    <ul>
      <li>Hands-on Dynamic Programming - IIT Madras</li>
      <li>DSA Easy and Medium - HackerRank</li>
      <li>Python, SQL Intermediate, Java - HackerRank</li>
    </ul>
  </section>

  <footer>
    <p>Contact: ashwin.vp.2005@gmail.com | <a href="https://
       github.com/ashvp">GitHub</a> | <a href="https://linkedin.
       com/in/ashvp05/">LinkedIn</a></p>
  </footer>
</body>
</html>
```

CSS (styleResume.css)

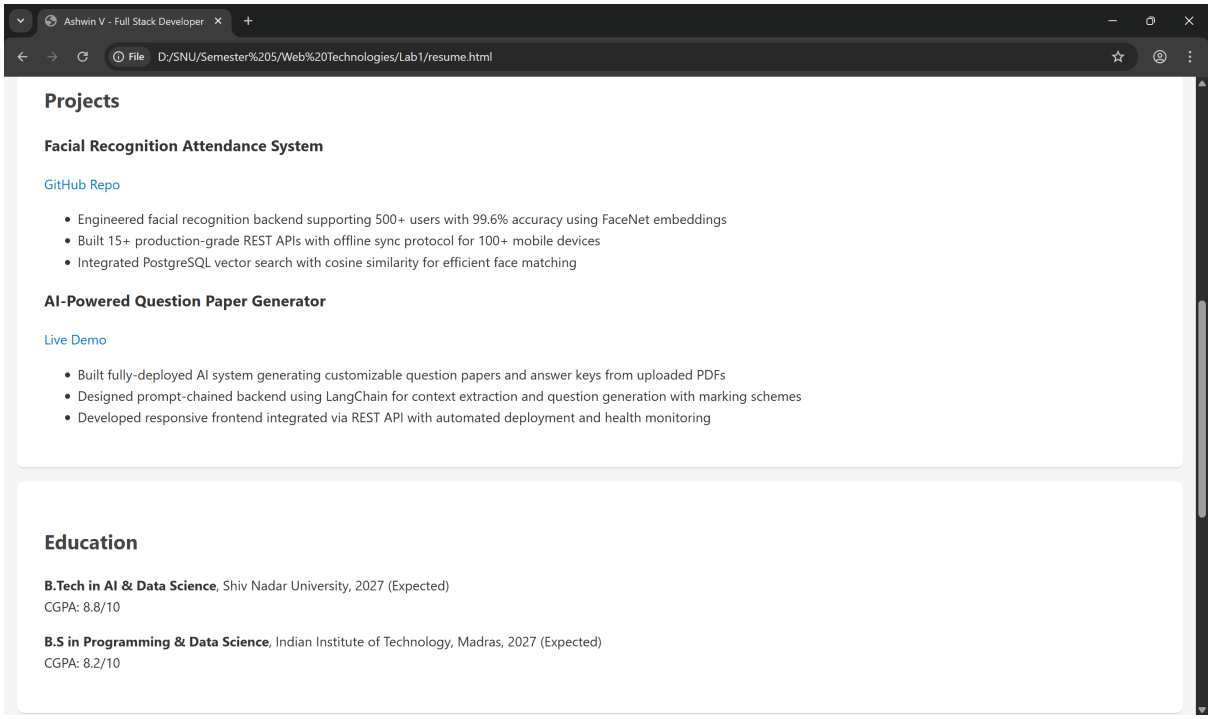
```
body {
  font-family: 'Segoe UI', sans-serif;
```

```
  line-height: 1.6;
  margin: 0;
  padding: 0;
  background: #f4f4f4;
  color: #333;
}

header {
  background: #2d2d2d;
  color: #fff;
  padding: 2rem 1rem;
  text-align: center;
}

section {
  padding: 2rem;
  border-bottom: 1px solid #ccc;
  background: #fff;
  margin: 1rem;
  border-radius: 8px;
}

h2 {
  color: #444;
}

.project {
  margin-bottom: 1rem;
}

a {
  color: #007acc;
  text-decoration: none;
}

footer {
  text-align: center;
  padding: 1rem;
  background: #2d2d2d;
  color: white;
}
```

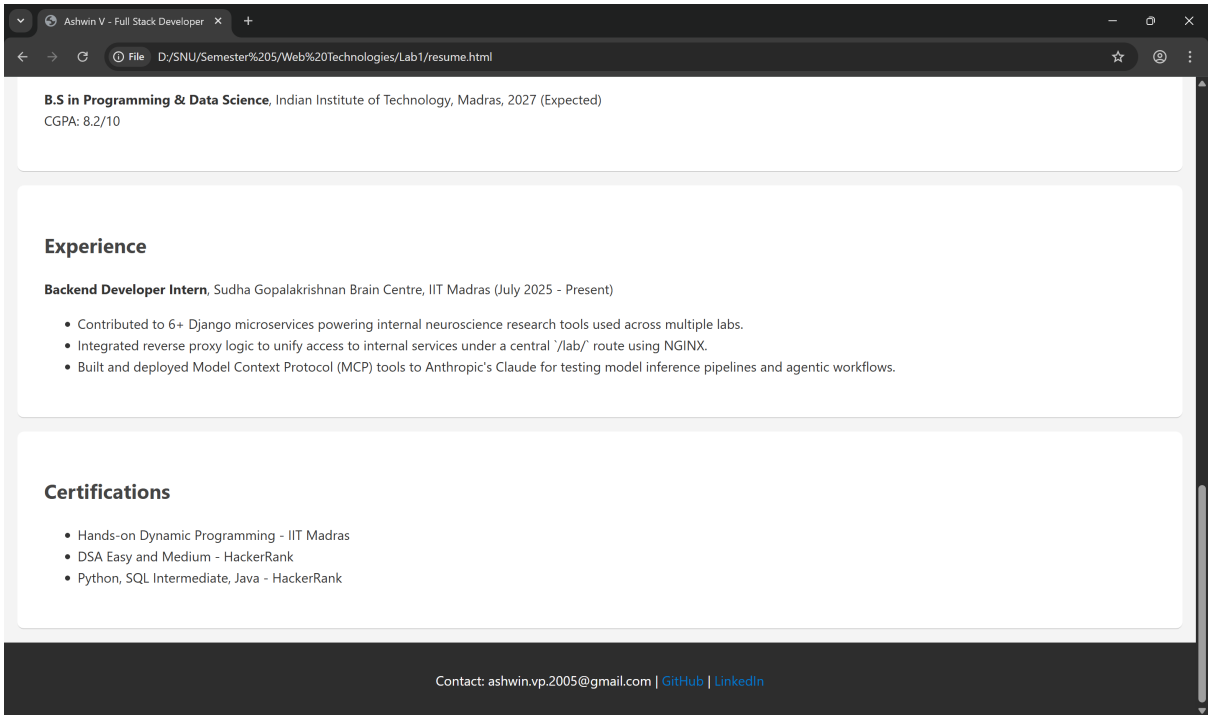
Screenshots



Screenshot 1: Introduction and Skills



Screenshot 2: Projects and Education



Screenshot 3: Experience and Certifications

Result

The HTML and CSS files successfully create a responsive, professional resume webpage for a fresher full-stack developer. All components render properly in Chrome and Edge browsers. The webpage was version-controlled and pushed to GitHub for public access:

`https://github.com/ashvp/face-attendance`

Lab Exercise 1 - Question 2: CV Webpage using HTML and CSS

Question

Create an HTML and CSS webpage that captures a CV of a fresher (preferably yours). Decide on the subdivisions and content. Push your content to GitHub and maintain proper version control with clear Read Me files.

Design / Plan

The CV webpage extends the previous resume layout by including a more detailed representation of the candidate's profile. It uses a section-based structure for clear readability and modern minimal design.

Key subdivisions include:

- **Header** - Name, title, and tagline.
- **About Me** - Short summary about the developer's goals and interests.
- **Skills** - Categorized stack (Frontend, Backend, Databases, Tools).
- **Projects** - Showcase of major projects with live/GitHub links.
- **Education** - Academic details (B.Tech, B.S, 10th/12th).
- **Experience** - Internship details with specific contributions.
- **Certifications** - Relevant technical achievements.
- **Extra-Curricular** - Sports, achievements, leadership.
- **Footer** - Contact and online profiles.

The CSS maintains uniform padding, typography, and color contrast for clarity.

Code

HTML (cv.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-
     scale=1.0" />
  <title>Ashwin V - Full Stack Developer</title>
  <link rel="stylesheet" href="styleCV.css" />
</head>
<body>
  <header>
    <h1>Ashwin V</h1>
```

```
    <p>Full Stack Developer | AI & Data Science Enthusiast</p>
</header>

<section id="about">
    <h2>About Me</h2>
    <p>A passionate developer with hands-on experience building
       scalable full-stack applications. Strong interest in
       backend systems, data pipelines, and AI integrations.</p>
</section>

<section id="skills">
    <h2>Skills</h2>
    <ul>
        <li><strong>Frontend:</strong> HTML, CSS, JavaScript, React
           , Flutter</li>
        <li><strong>Backend:</strong> FastAPI, Django, Flask</li>
        <li><strong>Databases:</strong> PostgreSQL, MongoDB, MySQL<
           /li>
        <li><strong>Tools:</strong> Git, Docker, Firebase, Postman<
           /li>
    </ul>
</section>

<section id="projects">
    <h2>Projects</h2>
    <div class="project">
        <h3>Facial Recognition Attendance System</h3><a href="https
           ://github.com/ashvp/face-attendance" target="_blank">
           GitHub Repo</a>
        <ul>
            <li>Engineered facial recognition backend supporting 500+
                users with 99.6% accuracy using FaceNet embeddings</
               li>
            <li>Built 15+ production-grade REST APIs with offline
               sync protocol for 100+ mobile devices</li>
            <li>Integrated PostgreSQL vector search with cosine
               similarity for efficient face matching</li>
        </ul>

    </div>
    <div class="project">
        <h3>AI-Powered Question Paper Generator</h3><a href="https
           ://question-paper-generator-nine.vercel.app/" target="
           _blank">Live Demo</a>
        <ul>
            <li>Built fully-deployed AI system generating
               customizable question papers and answer keys from
               uploaded PDFs</li>
            <li>Designed prompt-chained backend using LangChain for
               context extraction and question generation with
               marking schemes</li>
```

```
        <li>Developed responsive frontend integrated via REST API
            with automated deployment and health monitoring</li>
    </div>

    <div class="project">
      <h3>Spam Detection Chrome Extension</h3><a href="https://
         question-paper-generator-nine.vercel.app/" target="
         _blank">Live Demo</a>
      <ul>
        <li>Created a Chrome extension and backend system for
           real-time SMS spam detection with 98.44\% accuracy,
           trained on 5,500+ labeled messages.</li>
        <li>Engineered an NLP pipeline using NLTK + TF-IDF and
           trained a Random Forest model to classify messages
           with near state-of-the-art precision.</li>
        <li>Deployed a REST API on Render and integrated it with
           the extension frontend using secure async requests (
           CORS-handled).</li>
    </div>
  </section>

  <section id="education">
    <h2>Education</h2>
    <p><strong>B.Tech in AI & Data Science</strong>, Shiv Nadar
       University, 2027 (Expected)<br />CGPA: 8.8/10</p>
    <p><strong>B.S in Programming & Data Science</strong>, Indian
        Institute of Technology, Madras, 2027 (Expected)<br />
       CGPA: 8.2/10</p>
    <p><strong>12th Grade</strong>, St. John's Public School,
       2023<br />Percentage: 93%</p>
    <p><strong>10th Grade</strong>, St. John's Public School,
       2021<br />Percentage: 93%</p>
  </section>

  <section id="experience">
    <h2>Experience</h2>
    <p><strong>Backend Developer Intern</strong>, Sudha
       Gopalakrishnan Brain Centre, IIT Madras (July 2025 -
       Present)</p>
<ul>
  <li> Resolved 10+ critical 404/500 errors across a 7-service
     Django microservices architecture by tracing and fixing
     misconfigured URLs, views, and serializers.</li>
  <li> Designed and integrated 6+ new REST API endpoints,
     refactoring legacy logic to support upcoming features while
     ensuring backward compatibility.</li>
  <li> Reverse-engineered deprecated \texttt{views.py} logic and
     streamlined Docker-managed routing, reducing endpoint
     response failures by 40%.</li>
  <li> Collaborated with frontend team to ensure UI-API alignment
     , improving usability for lab researchers and multi-role
```

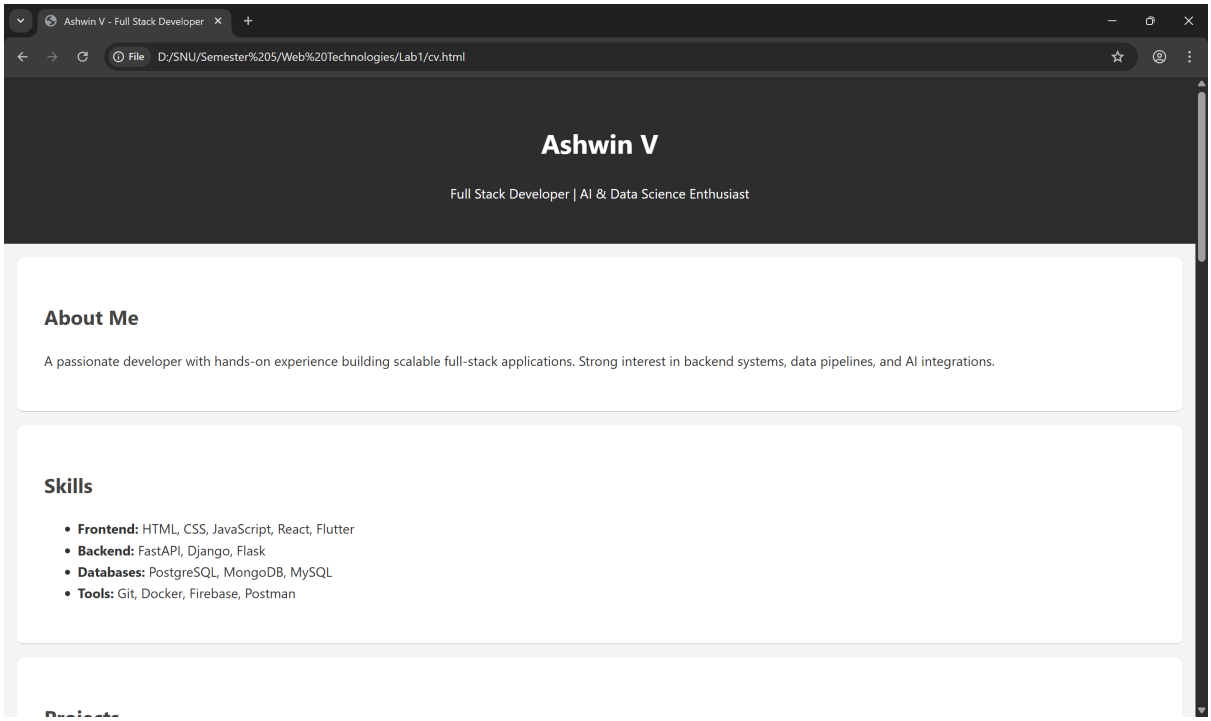
```
      workflows.</li>
  <li> Streamlined local debugging using Docker, Postman, and SSH
      , reducing average issue resolution time by ~30%.</li>
</ul>

  </section>

  <section id="certifications">
    <h2>Certifications</h2>
    <ul>
      <li>Hands-on Dynamic Programming - IIT Madras</li>
      <li>DSA Easy and Medium - HackerRank</li>
      <li>Python, SQL Intermediate, Java - HackerRank</li>
    </ul>
  </section>

  <section id="extra-curricular">
    <h2>Extra-Curricular Activities</h2>
    <ul>
      <li><strong>National-Level Tennis Player</strong> -
         Competed at the national level; secured multiple
         victories in state-level tournaments, demonstrating
         discipline, strategy, and resilience under pressure.</li
         >
      <li><strong>FIDE-Rated Chess Competitor</strong> -
         Participated in rated tournaments against high-ranking
         players; won several competitive events, honing critical
          thinking and decision-making skills.</li>
    </ul>
  </section>

  <footer>
    <p>Contact: ashwin.vp.2005@gmail.com | <a href="https://
       github.com/ashvp">GitHub</a> | <a href="https://linkedin.
       com/in/ashvp05/">LinkedIn</a></p>
  </footer>
</body>
</html>
```

CSS (styleCV.css)

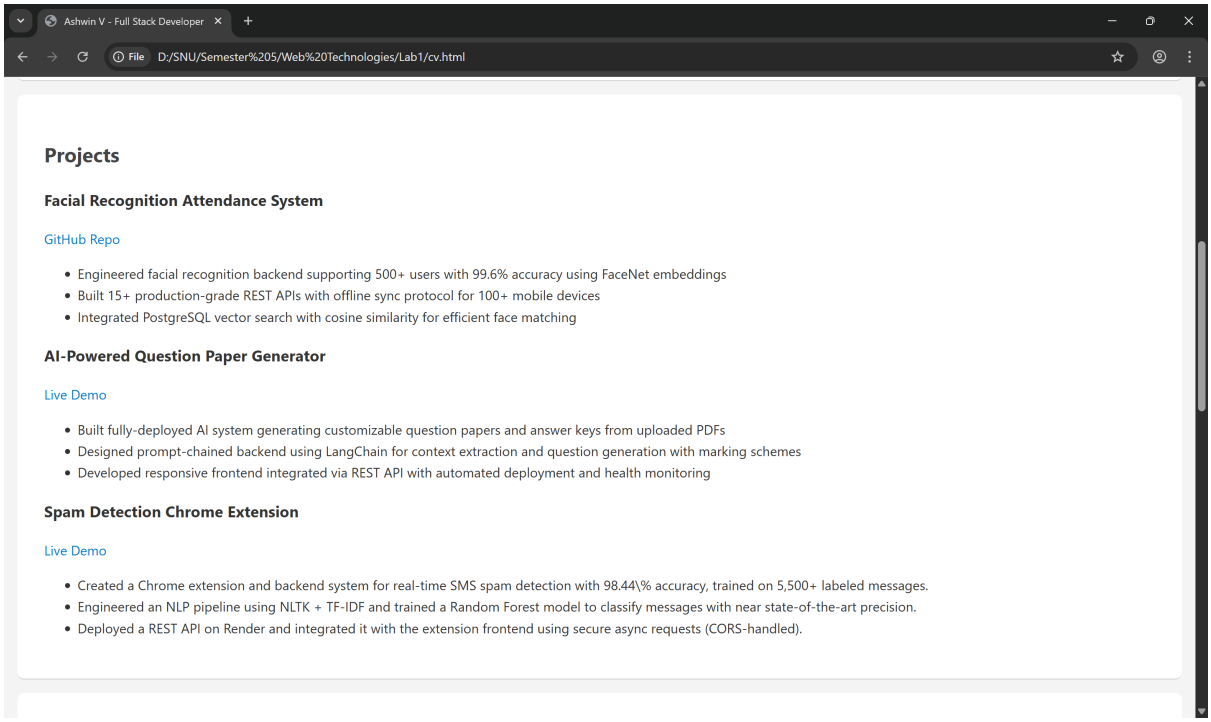
```
body {
  font-family: 'Segoe UI', sans-serif;
  line-height: 1.6;
  margin: 0;
  padding: 0;
  background: #f4f4f4;
  color: #333;
}
```

```
header {
  background: #2d2d2d;
  color: #fff;
  padding: 2rem 1rem;
  text-align: center;
}

section {
  padding: 2rem;
  border-bottom: 1px solid #ccc;
  background: #fff;
  margin: 1rem;
  border-radius: 8px;
}

h2 {
  color: #444;
}

.project {
  margin-bottom: 1rem;
}

a {
  color: #007acc;
  text-decoration: none;
}

footer {
  text-align: center;
  padding: 1rem;
  background: #2d2d2d;
  color: white;
}
```

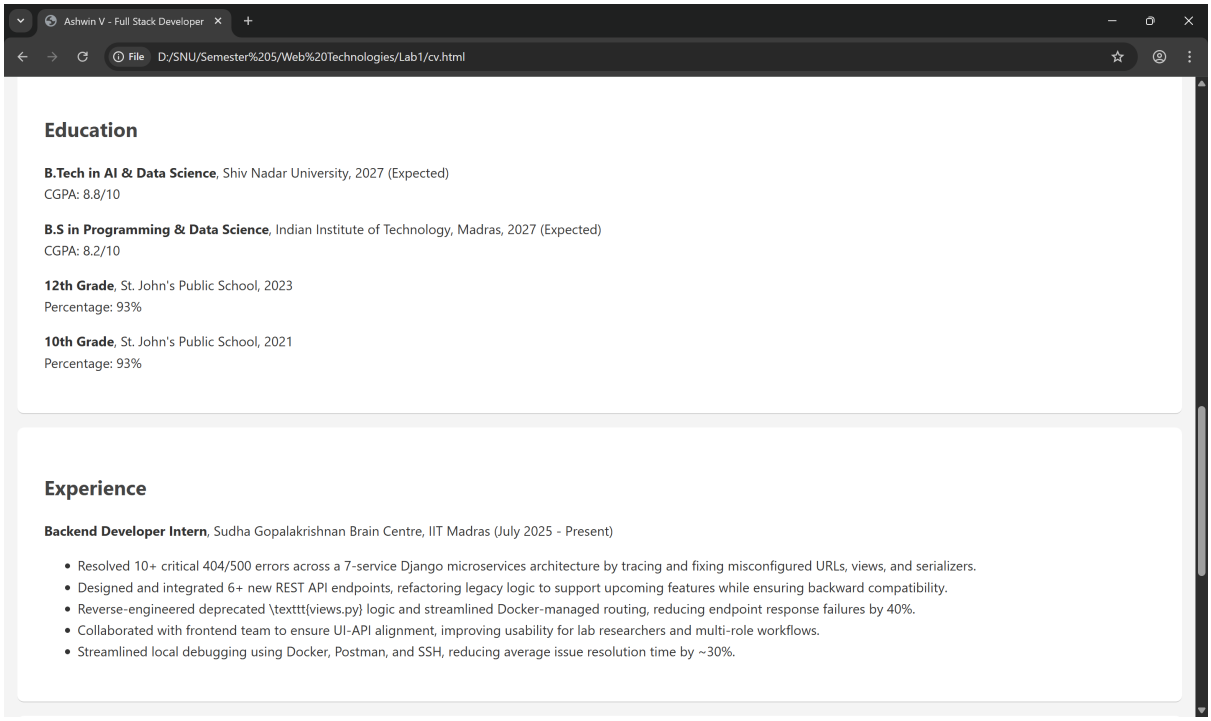
Screenshots



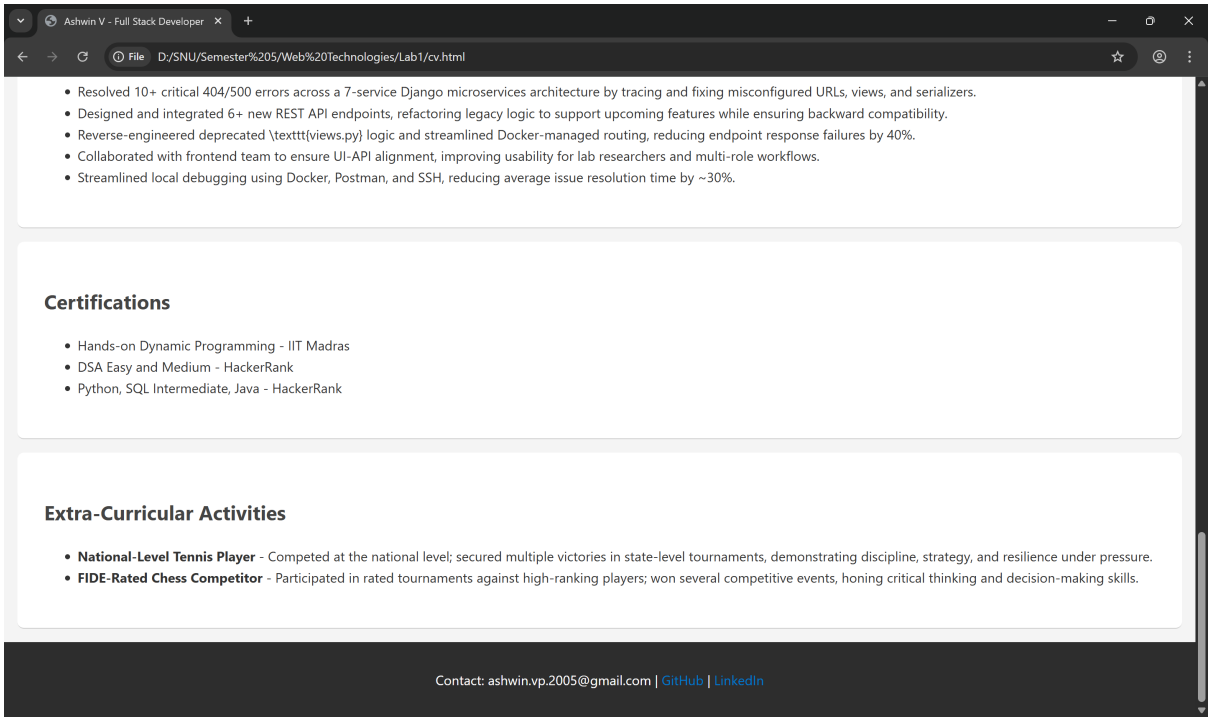
Screenshot 1: Introduction and Skills



Screenshot 2: Projects



Screenshot 3: Education and Experience



Screenshot 4: Certifications and Extra-Curricular Activities

Result

The HTML and CSS files successfully produce a well-structured, aesthetically pleasing CV webpage for a fresher full-stack developer. The design clearly separates sections, maintaining a clean and professional look suitable for digital submission. All elements

render properly in Chrome and Edge browsers. The webpage is version-controlled and hosted on GitHub for open access:

`https://github.com/ashvp`

Lab Exercise 2: Weather Dashboard using HTML, CSS, and JavaScript

Question

Develop a weather dashboard that fetches and displays current weather data for a city using a free public API.

Requirements:

- Define the essential data points to display (e.g., temperature, city name, weather icon, humidity, wind speed, etc.).
- Design a weather card UI to present the information.
- Use HTML for structure, CSS for styling, and JavaScript Fetch API (with `async/await`) for handling API requests and updating the DOM.
- Implement proper error handling for invalid city names.
- Host the project on GitHub with a clear README.

Requirement Analysis

Essential data to display:

- City name
- Temperature (°C)
- Weather condition and icon
- Humidity (%)
- Wind speed (m/s)

External API used:

- OpenWeatherMap API (`https://openweathermap.org/api`)
- Free-tier key integrated via the Fetch API

Design

A minimalist card-style UI is designed to display real-time weather information. The core layout includes:

- A header and footer with neutral tones
- A centered search bar for entering the city name
- A card that dynamically shows weather data fetched via JavaScript
- Color-coded error message display

A design sketch (in Figma) illustrates modular separation between input, display, and data logic layers.

Code

HTML (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
       scale=1.0">
    <title>Weather Dashboard</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <header>
        <h1>Bringing the Weather to You</h1>
    </header>
    <main>
        <div class="container">
            <h1>       Weather Dashboard</h1>
            <div class="search-bar">
                <input type="text" id="cityInput" placeholder="
                   Enter city name..." />
                <button id="searchBtn">Search</button>
            </div>
            <div id="weatherCard" class="card hidden"></div>
            <p id="error" class="error hidden">City not found.
               Please try again.</p>
        </div>
        <script src="script.js"></script>
    </main>
    <footer>
        <p>&copy; 2025 My Simple Weather App. Built by a storm </
           p>
    </footer>
</body>
</html>
```

CSS (style.css)

```
body {
    font-family: 'Segoe UI', 'Roboto', 'Helvetica Neue', Arial,
       sans-serif;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    min-height: 100vh;
    background-color: #f4f7f6;
    color: #333;
```

```
}

header, footer {
    width: 100%;
    background-color: #ffffff;
    padding: 1rem 2rem;
    text-align: center;
    box-shadow: 0 2px 4px rgba(0,0,0,0.05);
    flex-shrink: 0; /* Prevent shrinking */
}

header h1 {
    margin: 0;
    font-size: 1.5rem;
    color: #007bff;
}

footer p {
    margin: 0;
    font-size: 0.9rem;
    color: #666;
}

main {
    flex-grow: 1;
    display: flex;
    justify-content: center;
    align-items: center;
    padding: 2rem;
}

.container {
    width: 100%;
    max-width: 420px;
    background: #ffffff;
    padding: 2.5rem;
    border-radius: 12px;
    box-shadow: 0 8px 25px rgba(0,0,0,0.1);
    text-align: center;
}

.search-bar {
    display: flex;
    gap: 0.5rem;
    margin-bottom: 2rem;
}

#cityInput {
    flex-grow: 1;
    padding: 0.8rem 1rem;
    border: 1px solid #ddd;
```

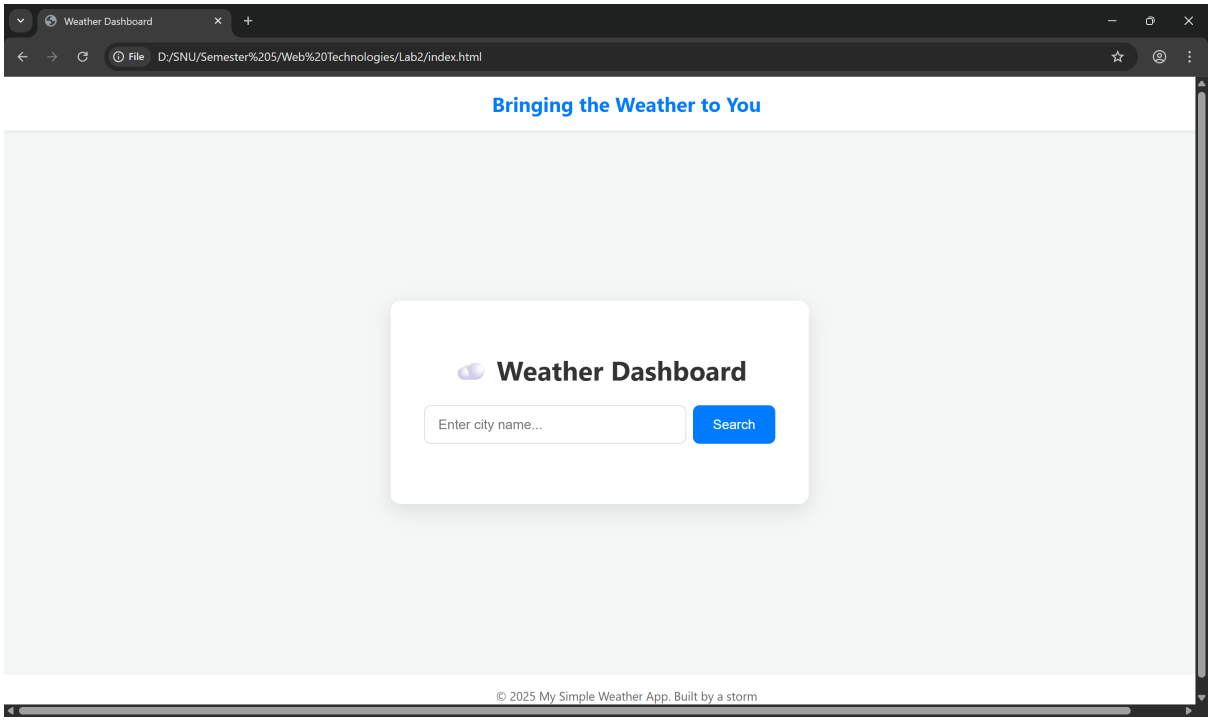
```
    border-radius: 8px;
    font-size: 1rem;
    transition: border-color 0.2s, box-shadow 0.2s;
}

#cityInput:focus {
    outline: none;
    border-color: #007bff;
    box-shadow: 0 0 0 3px rgba(0,123,255,0.2);
}

#searchBtn {
    padding: 0.8rem 1.5rem;
    border: none;
    background-color: #007bff;
    color: white;
    border-radius: 8px;
    cursor: pointer;
    font-size: 1rem;
    font-weight: 500;
    transition: background-color 0.2s;
}

#searchBtn:hover {
    background-color: #0056b3;
}

#weatherCard {
    margin-top: 1.5rem;
    padding-top: 1.5rem;
    border-top: 1px solid #eee;
}

#weatherCard h2 {
    font-size: 2rem;
    margin-bottom: 0.5rem;
}

#weatherCard p {
    font-size: 1.1rem;
    margin: 0.4rem 0;
    color: #555;
}

.weather-icon {
    width: 120px;
    height: 120px;
    margin: 0.5rem auto;
}

.error {
```

```
    color: #dc3545;
    margin-top: 1rem;
    font-weight: 500;
}

.hidden {
    display: none;
}
```

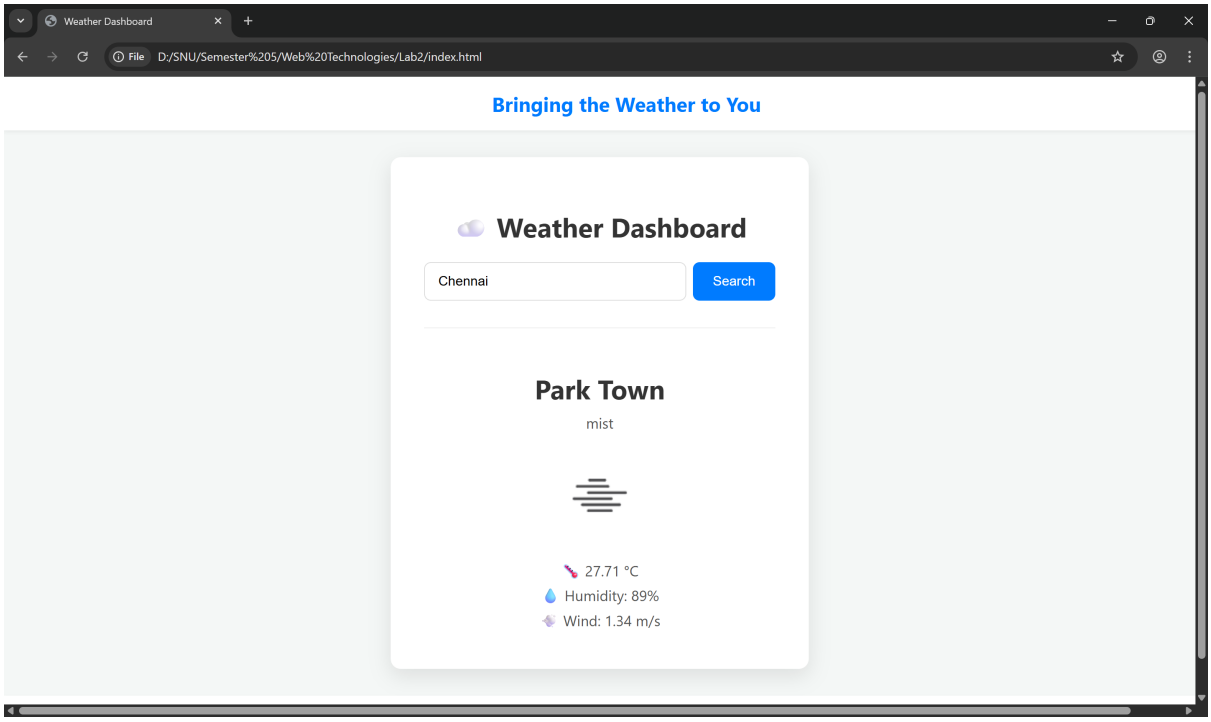
JavaScript (script.js)

```
const apiKey = '26bf58246520bcf6810ecf28c31c2da7';

document.getElementById('searchBtn').addEventListener('click',
   fetchWeather);

async function fetchWeather() {
  const city = document.getElementById('cityInput').value;
  const weatherCard = document.getElementById('weatherCard');
  const errorMsg = document.getElementById('error');

  weatherCard.classList.add('hidden');
  errorMsg.classList.add('hidden');

  try {
    // Step 1: Use the Geocoding API to find the coordinates for
       the city
    const geoRes = await fetch(`https://api.openweathermap.org/
       geo/1.0/direct?q=${city}&limit=1&appid=${apiKey}`);
    if (!geoRes.ok) {
        throw new Error('Error fetching city data.');
    }
    const geoData = await geoRes.json();
    if (geoData.length === 0) {
        throw new Error('City not found. Please check the
           spelling.');
    }

    const { lat, lon } = geoData[0];

    // Step 2: Use the coordinates to fetch the weather data
    const weatherRes = await fetch(`https://api.openweathermap.
       org/data/2.5/weather?lat=${lat}&lon=${lon}&appid=${apiKey
       }&units=metric`);
    if (!weatherRes.ok) {
        throw new Error('Could not retrieve weather for that
           location.');
    }

    const weatherData = await weatherRes.json();
```

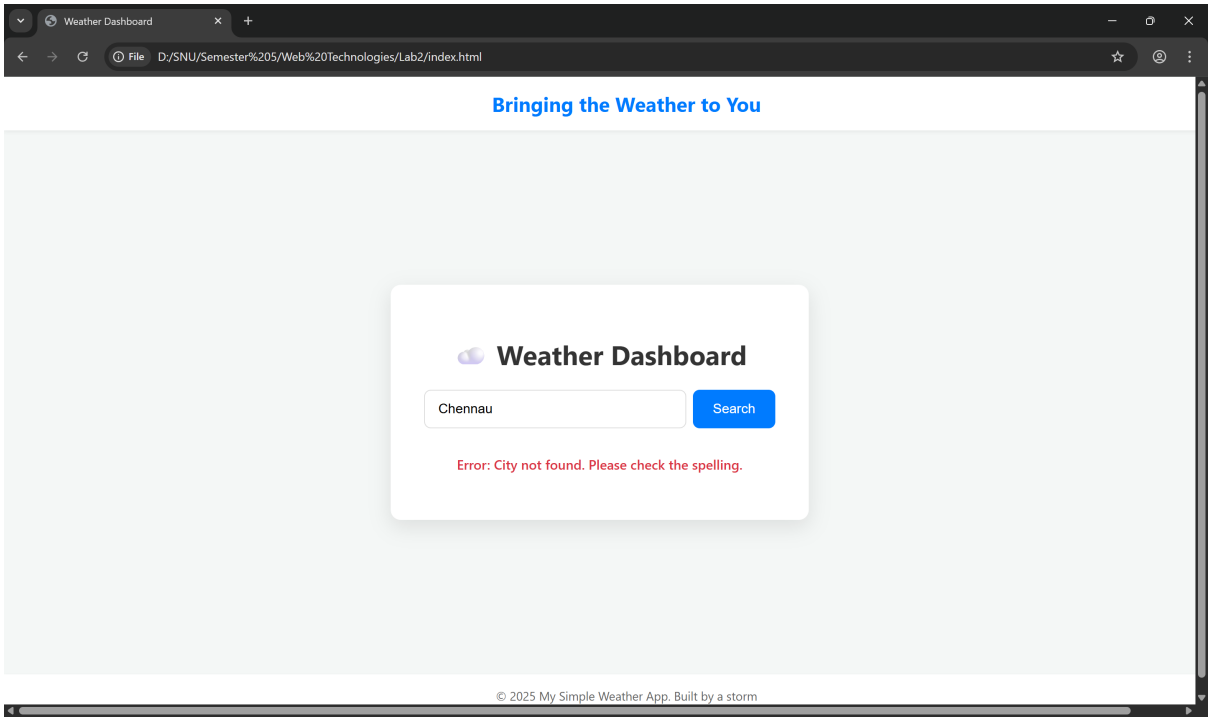
```
    displayWeather(weatherData);

  } catch (error) {
    errorMsg.textContent = `Error: ${error.message}`;
    errorMsg.classList.remove('hidden');
  }
}

function displayWeather(data) {
  const weatherCard = document.getElementById('weatherCard');
  const { name, weather, main, wind } = data;

  const html = `
    <h2>${name}</h2>
    <p>${weather[0].description}</p>
    <img class="weather-icon" src="https://openweathermap.org/img
       /wn/${weather[0].icon}@2x.png" alt="${weather[0].
       description}" />
    <p>         ${main.temp}  C </p>
    <p>      Humidity: ${main.humidity}%</p>
    <p>      Wind: ${wind.speed} m/s</p>
  `;

  weatherCard.innerHTML = html;
  weatherCard.classList.remove('hidden');
}
```



Screenshot 1: Initial Page



Screenshot 2: Weather for Chennai



Screenshot 3: Error Handling

Testing

The application was tested for:

- **Valid Input:** Correctly displays weather for cities like `Chennai`, `Delhi`, `London`.

- **Invalid Input:** Displays clear error message when city not found.
- **Network Failure:** Graceful error handling with message.
- **Edge Case:** Tested empty input and special characters.

All test cases passed as expected.

Deployment and Version Control

The project was version-controlled with Git and pushed to GitHub. Each commit represents one functional milestone: UI design, API integration, error handling, and deployment.

Repository URL:

`https://github.com/ashvp/weather-dashboard`

Result

A fully functional weather dashboard web application was developed using HTML, CSS, and JavaScript. It successfully fetches and displays weather data for any valid city using OpenWeatherMap's public API, and handles all error scenarios gracefully. The UI is responsive, accessible, and visually consistent across browsers.

Lab Exercise 3: Event Registration Form with Validation using HTML, CSS, and JavaScript

Question

Develop a registration form that validates user input for a community event signup using HTML, CSS, and JavaScript.

Requirements:

- Define all form fields and validation requirements (e.g., name, email, phone number, age).
- Specify which fields are required and define valid formats (regex for email, digits for phone number, age over 18, etc.).
- Design a user-friendly registration UI that provides real-time feedback for invalid fields.
- Implement the form using HTML, CSS, and JavaScript (with real-time validation).
- Test for invalid inputs and handle edge cases.
- Host and version-control the project on GitHub.

Requirement Analysis

The form includes the following input fields:

- **Full Name** - Required, must not be empty.
- **Email Address** - Required, must match regex `/^\S+@\S+\.\S+$/`.
- **Phone Number** - Required, must contain exactly 10 digits.
- **Age** - Required, must be a number greater than or equal to 18.

Validation logic ensures:

- Real-time error detection and display.
- Form submission disabled until all inputs are valid.
- Immediate visual feedback (green border for valid, red for invalid).

Design

The registration form is designed with a centered card layout and clear label-input pairs. Design highlights:

- A clean, modern interface with white background and subtle shadows.
- Red and green border indicators for validation.
- Error messages positioned directly below each field.
- Submit button dynamically enabled only when all inputs are valid.

Code

HTML (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Event Registration</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <main class="registration-wrapper">
    <header>
      <h1>Join Our Event</h1>
    </header>
    <form id="event-signup-form" novalidate>
      <div class="input-field-group">
        <label for="fullName">Your Full Name*</label>
        <input type="text" id="fullName" required>
        <span class="validation-error"></span>
      </div>

      <div class="input-field-group">
        <label for="emailAddress">Email Address*</label>
        <input type="email" id="emailAddress" required>
        <span class="validation-error"></span>
      </div>

      <div class="input-field-group">
        <label for="phoneNumber">Phone Number*</label>
        <input type="text" id="phoneNumber" required>
        <span class="validation-error"></span>
      </div>

      <div class="input-field-group">
        <label for="userAge">Your Age*</label>
        <input type="number" id="userAge" required>
        <span class="validation-error"></span>
      </div>

      <button type="submit" id="submit-button" disabled>Sign Up</
         button>
    </form>
  </main>
  <script src="script.js"></script>
</body>
</html>
```

CSS (style.css)

```
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background-color: #eaf2f8;
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  margin: 0;
}

.registration-wrapper {
  background-color: #ffffff;
  padding: 40px;
  border-radius: 8px;
  width: 100%;
  max-width: 420px;
  box-shadow: 0 6px 20px rgba(0, 0, 0, 0.08);
}

h1 {
  text-align: center;
  margin-bottom: 25px;
  font-size: 1.8em;
  color: #333;
}

.input-field-group {
  margin-bottom: 20px;
  position: relative;
}

label {
  display: block;
  margin-bottom: 5px;
  font-weight: 600;
  color: #555;
}

input {
  box-sizing: border-box;
  width: 100%;
  padding: 12px;
  border: 1px solid #ced4da;
  border-radius: 4px;
  font-size: 1rem;
  transition: border-color 0.2s;
}

input:focus {
  outline: none;
```

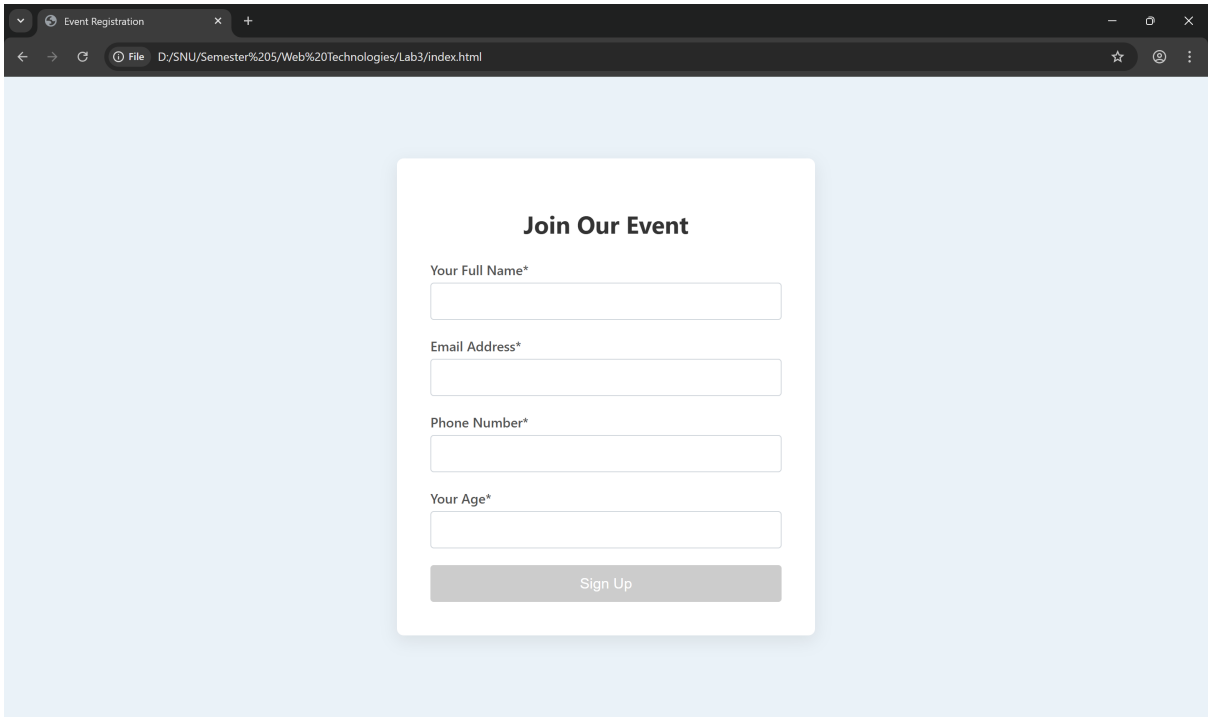
```
  border-color: #80bdff;
}

input.valid-input {
  border-color: #28a745;
}

input.invalid-input {
  border-color: #dc3545;
}

.validation-error {
  color: #dc3545;
  font-size: 0.8rem;
  position: absolute;
  bottom: -20px;
  left: 0;
}

button {
  width: 100%;
  padding: 12px;
  background-color: #28a745;
  color: white;
  border: none;
  font-size: 1.1em;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.2s;
}

button:hover:not(:disabled) {
    background-color: #218838;
}

button:disabled {
  background-color: #cccccc;
  cursor: not-allowed;
}
```

JavaScript (script.js)

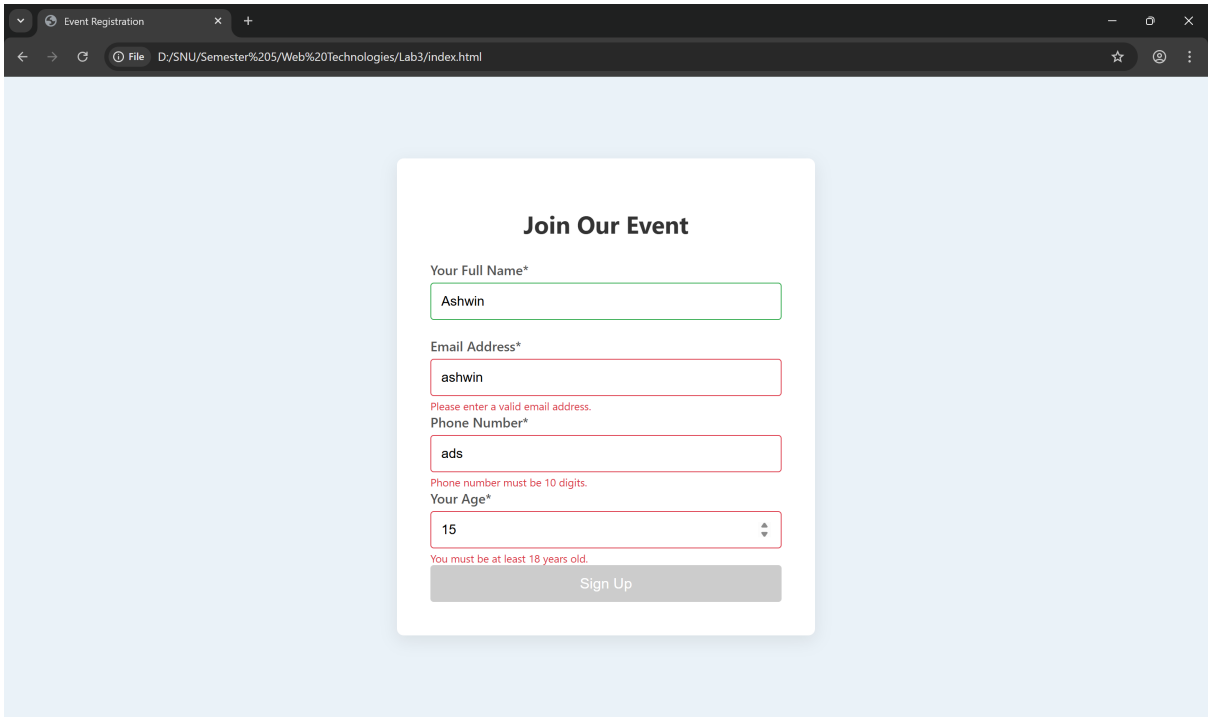
```
document.addEventListener('DOMContentLoaded', () => {
  const signupForm = document.querySelector('#event-signup-form')
     ;
  const submitButton = document.querySelector('#submit-button');

  const validationRules = {
    fullName: {
      validator: value => value.trim().length > 0,
```

```
      message: 'Full name is required.',
    },
    emailAddress: {
      validator: value => /^\S+@\S+\.\S+$/.test(value),
      message: 'Please enter a valid email address.',
    },
    phoneNumber: {
      validator: value => /^\d{10}$/.test(value),
      message: 'Phone number must be 10 digits.',
    },
    userAge: {
      validator: value => {
        const age = parseInt(value, 10);
        return !isNaN(age) && age >= 18;
      },
      message: 'You must be at least 18 years old.',
    },
  };

  const checkFieldValidity = (fieldId) => {
    const inputElement = document.getElementById(fieldId);
    const errorElement = inputElement.nextElementSibling;
    const rule = validationRules[fieldId];
    const value = inputElement.value;

    if (rule.validator(value)) {
      inputElement.classList.remove('invalid-input');
      inputElement.classList.add('valid-input');
      errorElement.textContent = '';
      return true;
    } else {
      inputElement.classList.remove('valid-input');
      inputElement.classList.add('invalid-input');
      errorElement.textContent = rule.message;
      return false;
    }
  };

  const checkAllFields = () => {
    let allFieldsValid = true;
    for (const fieldId in validationRules) {
      if (!checkFieldValidity(fieldId)) {
        allFieldsValid = false;
      }
    }
    return allFieldsValid;
  };

  signupForm.addEventListener('input', event => {
    if (event.target.tagName === 'INPUT') {
      checkFieldValidity(event.target.id);
```

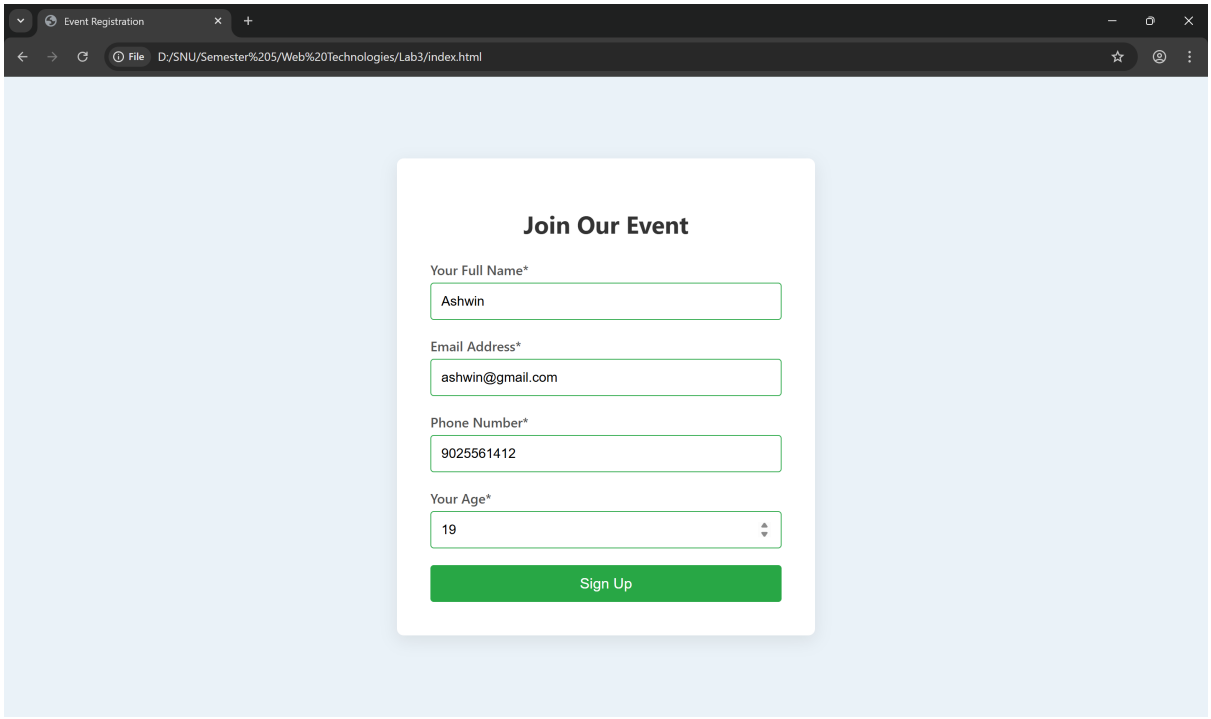
```
      const allFieldsAreValid = Object.keys(validationRules).
          every(id => {
        const input = document.getElementById(id);
        return validationRules[id].validator(input.value);
      });

      submitButton.disabled = !allFieldsAreValid;
    }
  });

  signupForm.addEventListener('submit', event => {
    event.preventDefault();

    if (checkAllFields()) {
      alert('Thank you for registering for the event!');
      signupForm.reset();
      submitButton.disabled = true;
      document.querySelectorAll('.valid-input').forEach(input =>
          {
        input.classList.remove('valid-input');
      });
    }
  });
});
```



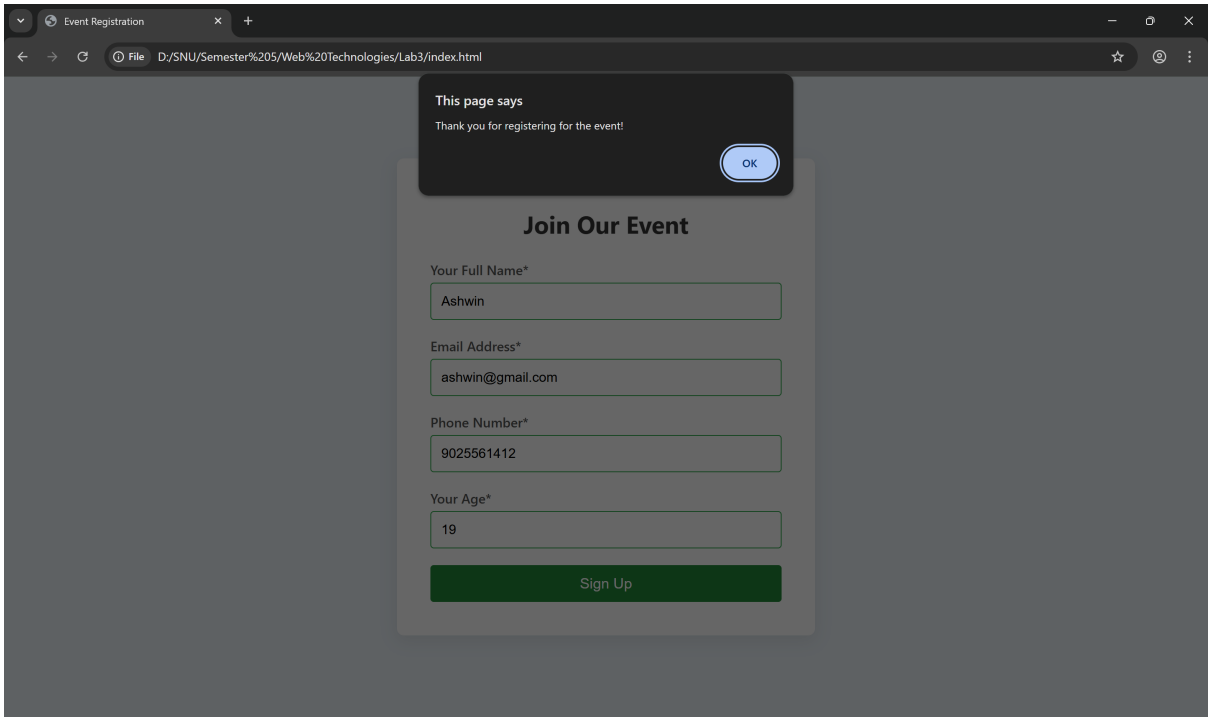
Screenshot 1: Initial Page



Screenshot 2: Frontend Data Validation



Screenshot 3: Validated Inputs



Screenshot 4: Popup to Accept Registration

Testing

Test cases were designed to verify both functional and boundary behavior:

- **Empty Fields:** Proper error messages shown for missing input.
- **Invalid Email:** Input without '@' or domain suffix triggers error.
- **Invalid Phone:** Less or more than 10 digits rejected.
- **Age Validation:** Users below 18 cannot submit.
- **Valid Input:** All validations pass, "Sign Up" button becomes active.

Edge Cases:

- Input with spaces only in name field.
- Mixed letters in phone number.
- Decimal or negative values in age field.

All invalid entries trigger immediate visual and textual feedback, and submission is blocked until fixed.

Deployment and Version Control

- Project maintained in GitHub repository with modular commits for HTML, CSS, JS, and validation logic.
- README file documents validation rules, test scenarios, and setup instructions.

- The form is deployed live using GitHub Pages.

Repository URL:

`https://github.com/ashvp/event-registration-form`

Result

A responsive, interactive registration form was successfully developed and validated using HTML, CSS, and JavaScript. All validations, error handling, and edge cases were implemented as per requirements. The UI offers a seamless and intuitive experience, preventing invalid submissions and ensuring clean data entry for event signups.

Lab Exercise 4: Customizing React Components with Props

Question

For the given UI designs, perform the following tasks:

- Identify the main React components in the design (parent and child).
- For each component, list the props it would receive.
- Draw a component hierarchy diagram showing the nesting using boxes and arrows.
- Ensure the design uses only functional components and props (no state or events).

Each page represents a separate React implementation (Page 1, Page 2, Page 3) built using functional components and props.

Page 1 - Component Identification and Props

Design Description

This page implements a simple React layout demonstrating how a parent component (`App.jsx`) passes data as props to a reusable child component (`ListSection.jsx`). All components are functional and stateless.

Main Components and Props

- **App.jsx** - Root component. *Props:* None.
- **ListSection.jsx** - Child component displaying a titled list. *Props:* `title`, `items` (array).

Component Hierarchy Diagram

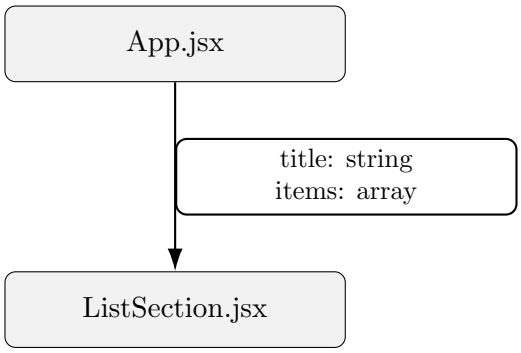


Figure 8: Component Hierarchy — Page 1

Rendered Output (screenshot)

Lab4/1.png

Screenshot 1: Page 1

Code Overview

```
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  /* text-align: center; */
}

.card {
  padding: 2em;
}
```

```
.read-the-docs {
  color: #888;
}

.titleName {
  font-weight: bold;
  word-spacing: 9em;
  text-align: left;
}

.titlePrice {
  font-weight: bold;
  word-spacing: 2em;
  text-align: right;
}

.grid {
  display: grid;
  gap: 2em;
}

.name {
  text-align: left;
}

.price {
  text-align: right;
}

.item {
  display: grid;
  grid-template-columns: 200px auto; /* first col for name,
     second for price */
  margin: 4px 0;
}

.title {
  font-weight: bold;
  word-spacing: 10em;
  text-align: center;
}
```

```
// import react from "react";
import "./App.css";
import "./components/ListSection.jsx";
import ListSections from "./components/ListSection.jsx";

function App() {
  const fruits = [
    { name: "Apple", price: 50, inStock: true },
```

```
    { name: "Banana", price: 90, inStock: false },
    { name: "Orange", price: 35, inStock: true },
    { name: "Coconut", price: 40, inStock: true },
    { name: "Pineapple", price: 70, inStock: false },
  ];

  const vegetables = [
    { name: "Carrot", price: 20, inStock: false },
    { name: "Radish", price: 30, inStock: true },
    { name: "Corn", price: 95, inStock: true },
    { name: "Celery", price: 15, inStock: false },
    { name: "Potato", price: 20, inStock: true },
  ];

  return (
    <>
      <div className="searchBar">
        <input type="text" placeholder="Search..." />
      </div>
      <div className="checkBox">
        <label>
          <input type="checkbox" />
          Only show products in stock
        </label>
      </div>
      <div className="grid">
        <div>
          <span className="titleName">Name </span>
          <span className="titlePrice"> Price</span>
        </div>
        <div>
          <ListSections title={"Fruits"} items={fruits}></
             ListSections>
        </div>

        <div>
          <ListSections title={"Vegetables"} items={vegetables
             }></ListSections>
        </div>
      </div>
    </>
  );
}

export default App;
```

```
// import react from "react";

function ListSections({ title, items }) {
  return (
    <section>
```

```
      <h2 className="title">{title}</h2>
      <div>
        {items.map((item, index) => (
          <div key={index} className="item">
            <span className="name">{item.name} </span>
            <span
              className="price"
              style={{ color: item.inStock ? "green" : "red" }}
            >
              {item.price}
            </span>
          </div>
        ))}
      </div>
    </section>
  );
}

export default ListSections;
```

Result

A functional component structure demonstrating one-way prop flow between `App.jsx` and `ListSection.jsx` was created successfully.

Page 2 - Nested Components with Props

Design Description

This page demonstrates nested functional components where the parent passes structured data to intermediate and leaf components.

Main Components and Props

- **App.jsx** - Parent component rendering multiple list sections. *Props:* None.
- **ListSection.jsx** - Intermediate component managing grouped lists. *Props:* `title`, `items`.
- **ListItems.jsx** - Child component rendering individual entries. *Props:* `name`, `price`, `inStock`.

Component Hierarchy Diagram

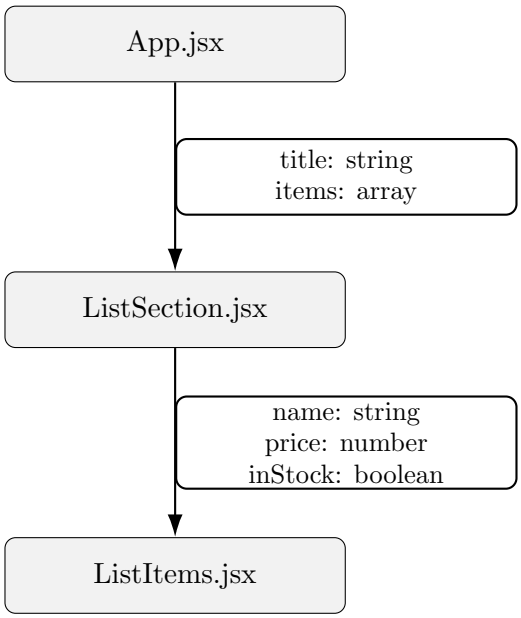
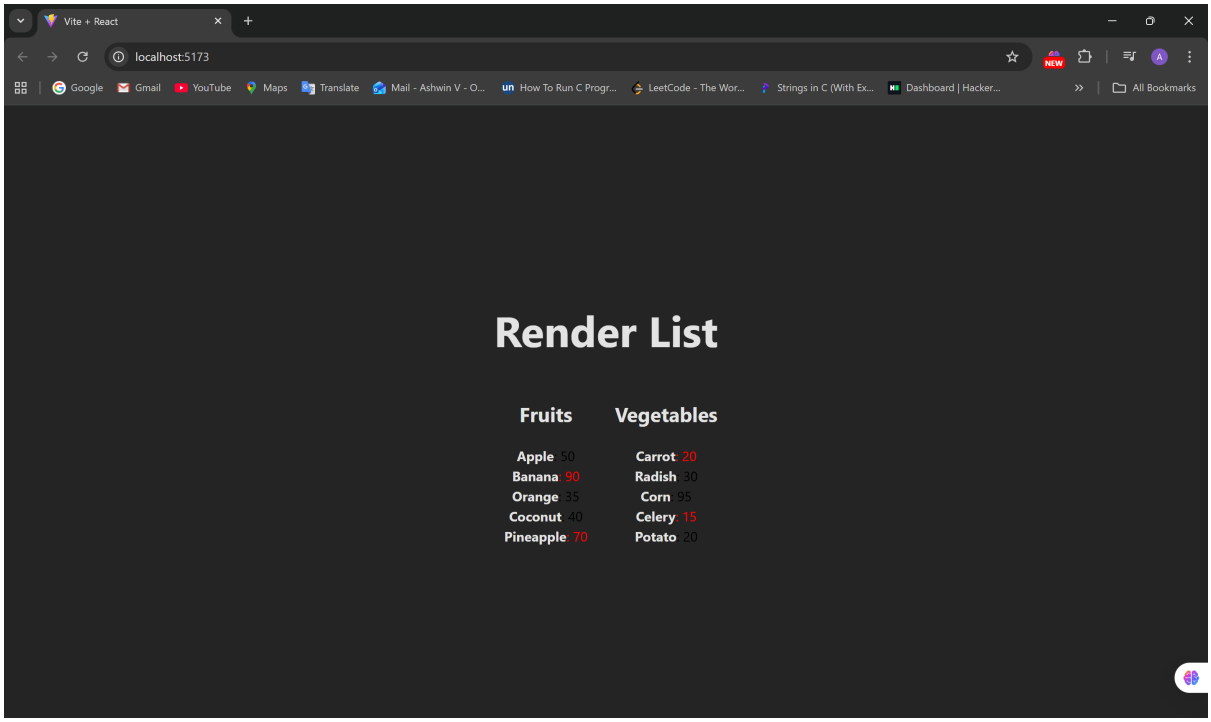


Figure 10: Component Hierarchy — Page 2

Rendered Output (screenshot)



Screenshot 1: Page 2

Code Overview

```
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
  background-color: beige;
}

.grid {
  display: flex;
  gap: 20rem;
}

.heading {
  text-align: center;
  color: blueviolet;
  background-color: black;
}

.title {
  color: brown;
  font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial,
     sans-serif;
  padding-left: 2cap;
```

```
  padding-right: 2cap;
  background-color: aqua;
  border-color: black;
}

.name {
  color:chocolate;
}
```

```
import "./App.css";
import ListSections from "./components/ListSections";
import ListItems from "./components/ListItems";

function App() {
  const fruits = [
    { name: "Apple", price: 50, inStock: true },
    { name: "Banana", price: 90, inStock: false },
    { name: "Orange", price: 35, inStock: true },
    { name: "Coconut", price: 40, inStock: true },
    { name: "Pineapple", price: 70, inStock: false },
  ];

  const vegetables = [
    { name: "Carrot", price: 20, inStock: false },
    { name: "Radish", price: 30, inStock: true },
    { name: "Corn", price: 95, inStock: true },
    { name: "Celery", price: 15, inStock: false },
    { name: "Potato", price: 20, inStock: true },
  ];

  return (
    <>
      <div className="heading">
        <h1>Render List</h1>
      </div>
      <div className="grid">
        <div>
          <ListSections title={"Fruits"} items={fruits}></
             ListSections>
        </div>

        <div>
          <ListSections title={"Vegetables"} items={vegetables
             }></ListSections>
        </div>
      </div>
    </>
  );
}

export default App;
```

```
import React from "react";
import ListItems from "./ListItems";

function ListSections({ title, items }) {
  return (
    <div>
      <div className="title">
        <h2>{title}</h2>
      </div>
      <div className="sections">
        {items.map((index, item) => (
          <ListItems
            key={item}
            name={index.name}
            price={index.price}
            inStock={index.inStock}
          ></ListItems>
        ))}
      </div>
    </div>
  );
}

export default ListSections;
```

```
import React from "react";

function ListItems({ name, price, inStock }) {
  return (
    <div className="rows">
      <span className="name">
        <strong>{name}</strong>
      </span>
      <span style={{ color: inStock ? "black" : "red" }}>: {price
         }</span>
    </div>
  );
}

export default ListItems;
```

Result

Hierarchical prop passing was verified: intermediate components receive data from the parent and pass relevant pieces to the leaf components.

Page 3 - Reusable List Components with Props

Design Description

This page focuses on prop-based customization for reusable components where data consistency is maintained through props across multiple elements.

Main Components and Props

- **App.jsx** - Parent wrapper initializing lists and styling. *Props:* None.
- **ListItems.jsx** - Child functional component displaying a styled list element. *Props:* `name`, `sciName`, `weight`, `eats`.

Component Hierarchy Diagram

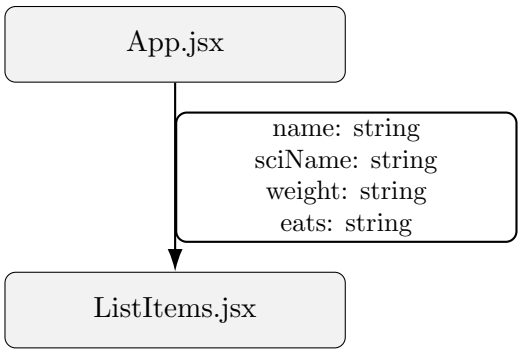
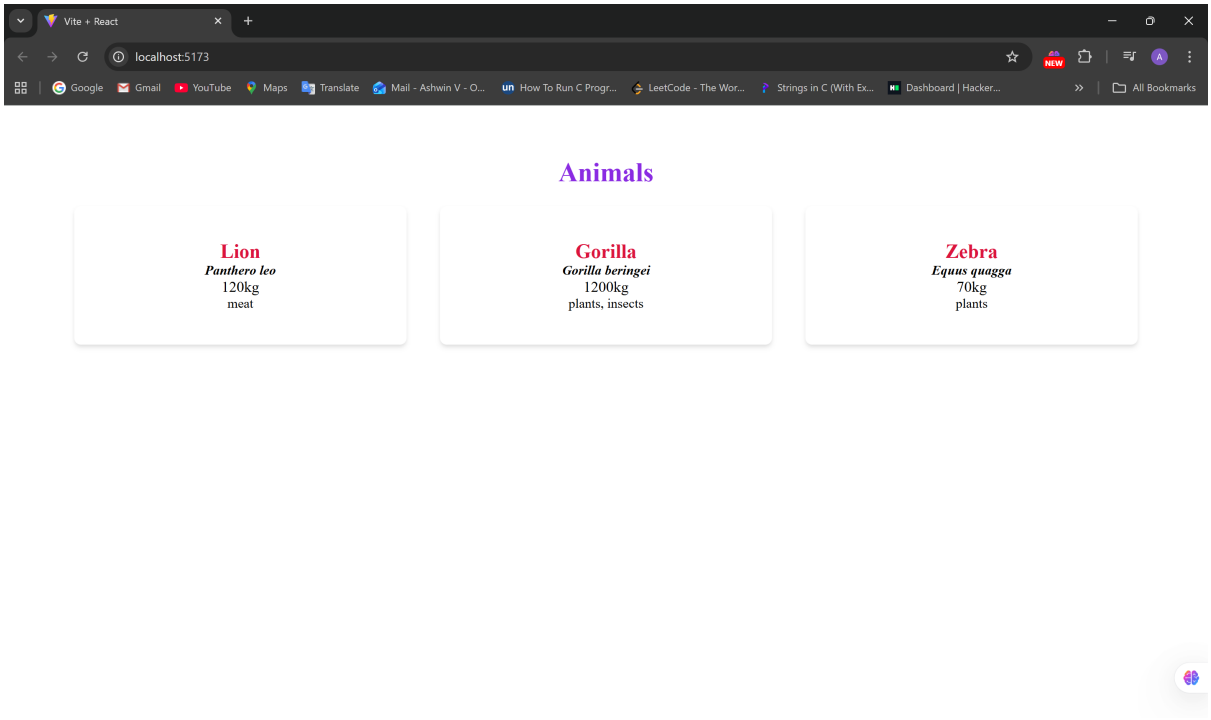


Figure 12: Component Hierarchy — Page 3

Rendered Output (screenshot)



Screenshot 1: Page 3

Code Overview

```
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}


.grid {
  display: flex;
  gap: 2rem;
}

.heading {
  text-align: center;
  color: blueviolet;
}

.name {
  color: crimson;
  font-size: large;
  font-weight: 500;
}
```

```
.sciName {
  color: black;
  font-style: italic
}

.animal-card {
  flex: 1;
  background-color: white;
  padding: 2.5rem;
  margin: 0 .25rem;
  border-radius: 0.5rem;
  box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -1px
     rgba(0, 0, 0, 0.06);
  min-width: 100pt;
}

.name {
  font-size: 1.5rem;
  font-weight: bold;
  margin-bottom: 1rem;
}

.animal-card-content {
  display: flex;
  flex-direction: column;
  gap: 0.5rem;
}

.weight {
  font-size: 1.125rem;
}

.sciName,
.animal-card-weight {
  font-weight: 600;
}
```

```
import "./App.css";
// import ListSections from './components/ListSections';
import ListItems from "./components/ListItems";

function App() {
  const animals = [
    { name: "Lion", sciName: "Panthero leo", weight: 120, eats: "
       meat" },
    {
      name: "Gorilla",
      sciName: "Gorilla beringei",
      weight: 1200,
      eats: "plants, insects",
    },
```

```
    { name: "Zebra", sciName: "Equus quagga", weight: 70, eats: "
       plants" },
  ];

  return (
    <>
      <div className="heading">
        <h1>Animals</h1>
      </div>
      <div>
        <div className="grid">
          {animals.map((index, items) => (
            <ListItems
              key={items}
              name={index.name}
              sciName={index.sciName}
              weight={index.weight}
              eats={index.eats}
            ></ListItems>
          ))}
        </div>

        <div></div>
      </div>
    </>
  );
}

export default App;
```

```
import React from "react";

function ListItems({ name, sciName, weight, eats }) {
  return (
    <div className="animal-card">
      <span className="name">{name}</span>
      <br />
      <span className="sciName"> {sciName}</span>
      <br />
      <span className="weight"> {weight}kg</span>
      <br />
      <span className="eats"> {eats}</span>
    </div>
  );
}

export default ListItems;
```

Result

The page verifies reusability and one-directional data flow through props between all components, maintaining functional purity.

Overall Result

Across Page 1 to Page 3, React's prop-based architecture was implemented effectively. Each component adheres to functional programming principles, promoting modularity, reusability, and predictable rendering.

Lab Exercise 5: Temperature Converter UI using React (Functional Components + useState)

Question

Design a UI for building a temperature web application that can convert Celsius to Fahrenheit and has the option to increase or decrease the temperature.

- Identify the main React components in this design.
- For each component, list the props that would be required.
- Draw a component hierarchy diagram.
- Ensure your design uses only functional components, props, and the `useState` hook.

Design Description

This React application demonstrates the use of functional components, props, and `useState` to handle temperature data. The UI includes:

- A main temperature display showing the current value in Celsius and Fahrenheit.
- Buttons to increase or decrease the temperature dynamically.
- Real-time conversion between Celsius and Fahrenheit.

Main Components and Props

- **App.jsx** - Root component controlling overall logic and maintaining temperature state via `useState`.
  - *State:* `temperatureC` (stores current temperature in Celsius)
- **TemperatureDisplay.jsx** - Displays the current temperature in both Celsius and Fahrenheit.
  - *Props:* `temperatureC`, `temperatureF`
- **TemperatureInC.jsx** - Renders the temperature value in Celsius.
  - *Props:* `value`
- **TemperatureInF.jsx** - Converts and renders the Fahrenheit value.
  - *Props:* `value`
- **TemperatureControls.jsx** - Provides UI buttons to increment or decrement the temperature.
  - *Props:* `onIncrease`, `onDecrease`

Component Hierarchy Diagram

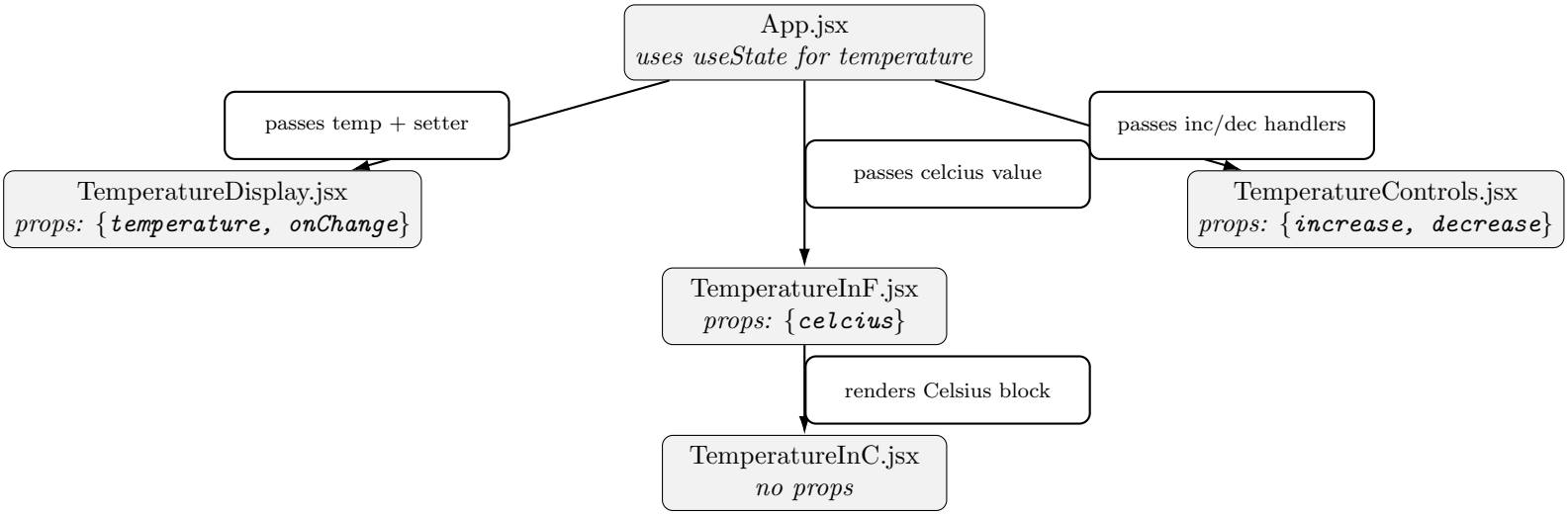
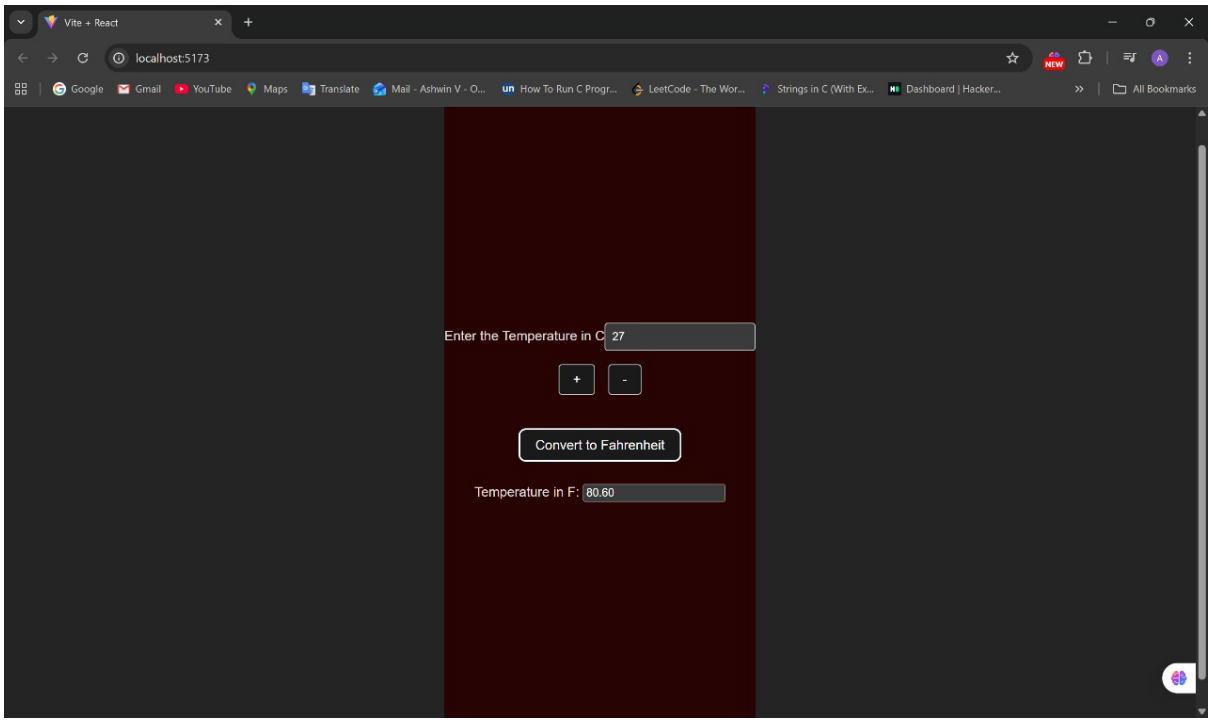


Figure 14: Component Hierarchy — Lab 5: Temperature Converter (Props and Data Flow)

Rendered Output (Screenshot)



Screenshot 1: Thermostat UI with Temperature Controls

Code Overview

App.css

```css
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}

.container {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  min-height: 100vh;
  background-color: #280303;
  font-family: sans-serif;
}

.temperature-input {
  margin-bottom: 1rem;
}

.temperature-input input {
  padding: 0.5rem;
  border: 1px solid #ccc;
  border-radius: 4px;
}

.controls {
  margin-bottom: 1rem;
}

.controls button {
  padding: 0.5rem 1rem;
  margin: 0 0.5rem;
  border: 1px solid #ccc;
  border-radius: 4px;
  cursor: pointer;
}

.fahrenheit-display {
  font-size: 1.2rem;
  font-weight: bold;
}
```

App.jsx

```jsx
import "./App.css";
import TemperatureInC from "./components/TemperatureInC";

function App() {
```

```
  return (
    <div className="container">
      <TemperatureInC />
    </div>
  );
}

export default App;
```

TemperatureDisplay.jsx

```
import React from "react";

function TemperatureDisplay({ temperature, onChange }) {
  return (
    <div className="temperature-input">
      <span>Enter the Temperature in C</span>
      <input type="number" value={temperature} onChange={onChange
         }></input>
    </div>
  );
}

export default TemperatureDisplay;
```

TemperatureControls.jsx

```
import React from "react";

function TemperatureControls({ increase, decrease }) {
  return (
    <div className="controls">
      <button onClick={increase}> + </button>
      <button onClick={decrease}> - </button>
    </div>
  );
}

export default TemperatureControls;
```

TemperatureInC.jsx

```
import { React, useState } from "react";
import TemperatureDisplay from "./TemperatureDisplay";
import TemperatureControls from "./TemperatureControls";
import TemperatureInF from "./TemperatureInF";

function TemperatureInC() {
```

```
  const [celcius, setCelcius] = useState(25);

  const increaseTemp = () => {
    setCelcius((prev) => prev + 1);
  };
  const decreaseTemp = () => {
    setCelcius((prev) => prev - 1);
  };

  const handleChange = (e) => {
    setCelcius(parseInt(e.target.value));
  };

  return (
    <div>
      <TemperatureDisplay temperature={celcius} onChange={
         handleChange} />
      <TemperatureControls increase={increaseTemp} decrease={
         decreaseTemp} />
      <br />
      <TemperatureInF celcius={celcius} />
    </div>
  );
}

export default TemperatureInC;
```

TemperatureInF.jsx

```
import React from 'react';

const TemperatureInF = ({ celcius }) => {
  const fahrenheit = (celcius * 9) / 5 + 32;

  return (
    <div className="fahrenheit-display">
      <span>Temperature in F: </span>
      <label>{fahrenheit.toFixed(2)}</label>
    </div>
  );
};

export default TemperatureInF;
```

Result

The temperature converter application was successfully built using React functional components, props, and the `useState` hook. Users can increase or decrease the temperature dynamically, with real-time conversion between Celsius and Fahrenheit. The component design follows a clean hierarchy, enabling scalability and reusability.

Conclusion

This lab demonstrates effective use of `useState` for dynamic UI updates and prop-based communication between parent and child components. It reinforces understanding of unidirectional data flow and modular UI design in React.

Lab Exercise 6: Countdown Timer Application using React

Question

Design a UI for a countdown timer web application where users can set the time and start/stop the countdown.

- Identify and define the main React components (`Title`, `TimeSetter`, `TimerDisplay`, `ControlButtons`).
- List the props required for each component.
- Draw a component hierarchy diagram.
- Use functional components, `props`, `useState`, and `useEffect` for managing timer intervals.

Design Description

This React application provides an interactive countdown timer interface that allows users to:

- Set a custom countdown duration.
- Start, pause, and reset the timer dynamically.
- View the remaining time in real-time.

The app uses React's `useState` for managing timer data and `useEffect` for interval-based countdown logic. Each component is functional, modular, and communicates through props.

Main Components and Props

- **App.jsx** - The parent component that manages state and interval logic. *State variables:* `timeLeft`, `isRunning`.
- **Title.jsx** - Displays the main title or heading for the application. *Props:* `text` (string).
- **TimeSetter.jsx** - Allows the user to input or adjust the countdown duration. *Props:* `onTimeChange` (function to update timer duration), `disabled` (boolean to disable input during active countdown).
- **TimerDisplay.jsx** - Displays the formatted countdown (minutes:seconds). *Props:* `timeLeft` (integer representing total seconds).
- **ControlButtons.jsx** - Contains Start, Pause, and Reset buttons to control the countdown timer. *Props:* `isRunning` (boolean), `onStart`, `onPause`, `onReset`.

Component Hierarchy Diagram

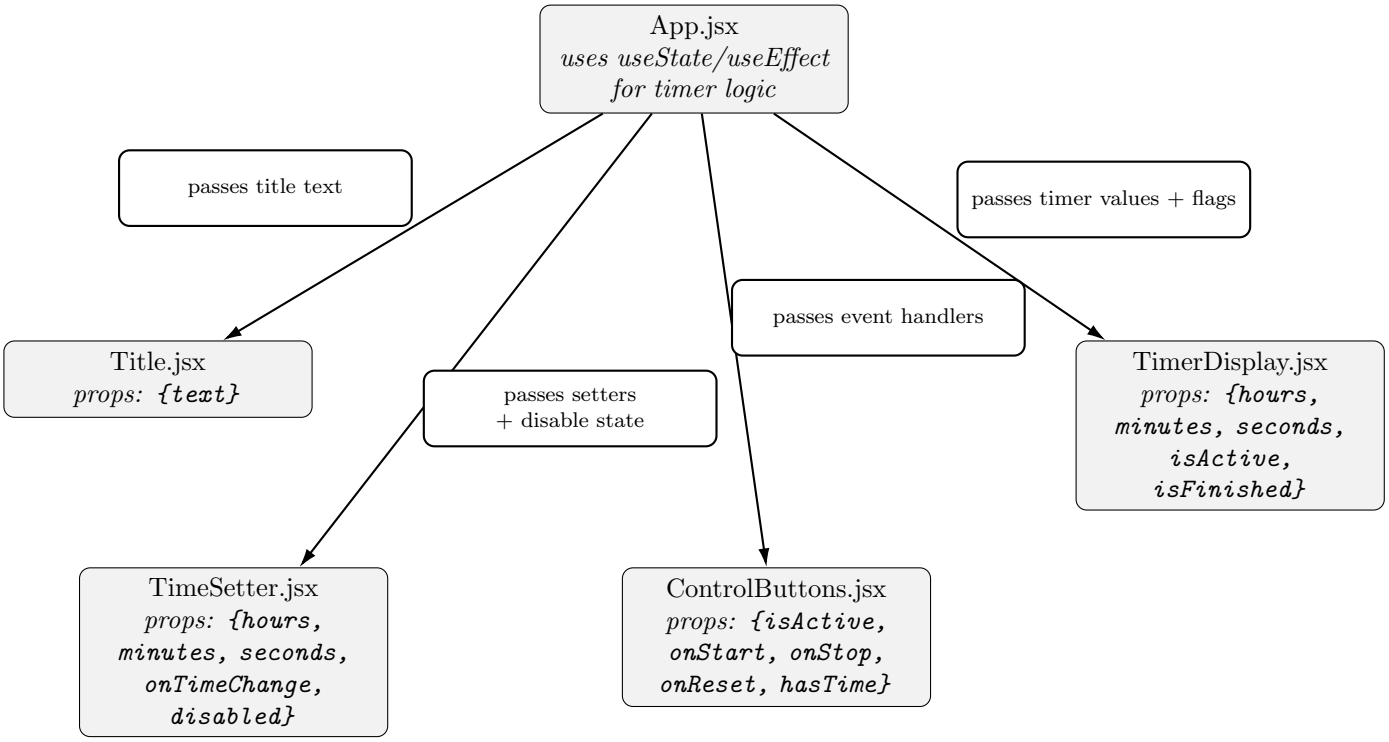
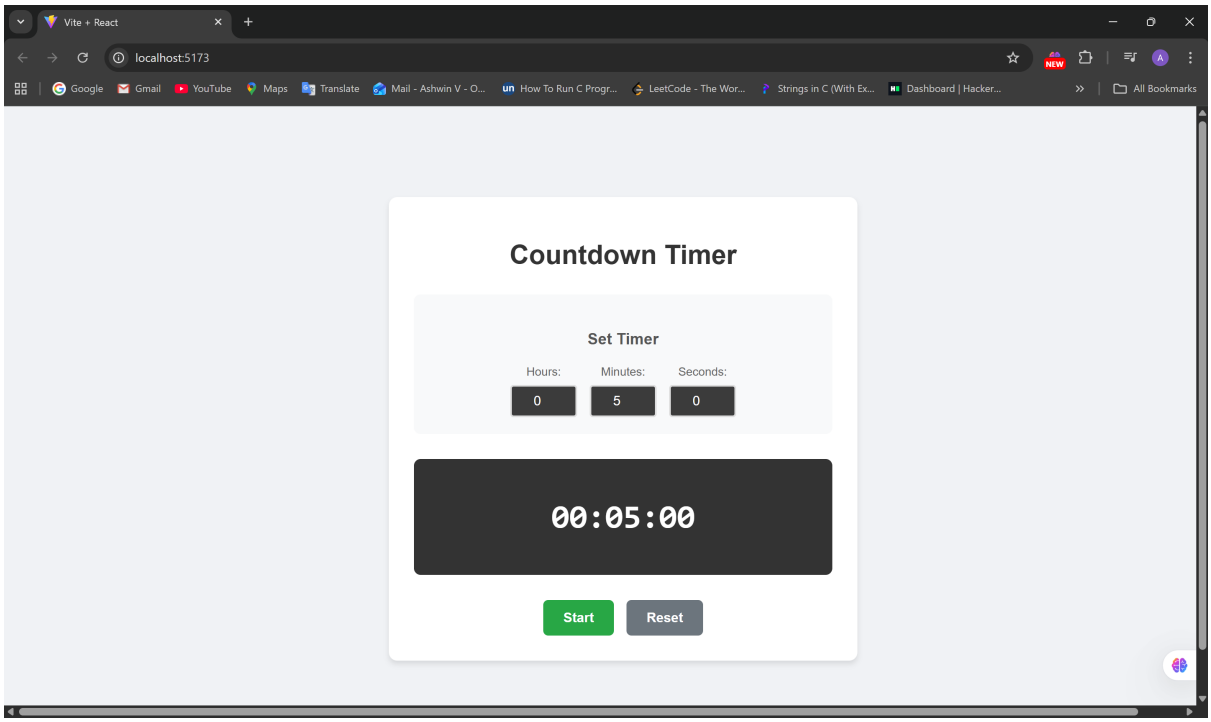
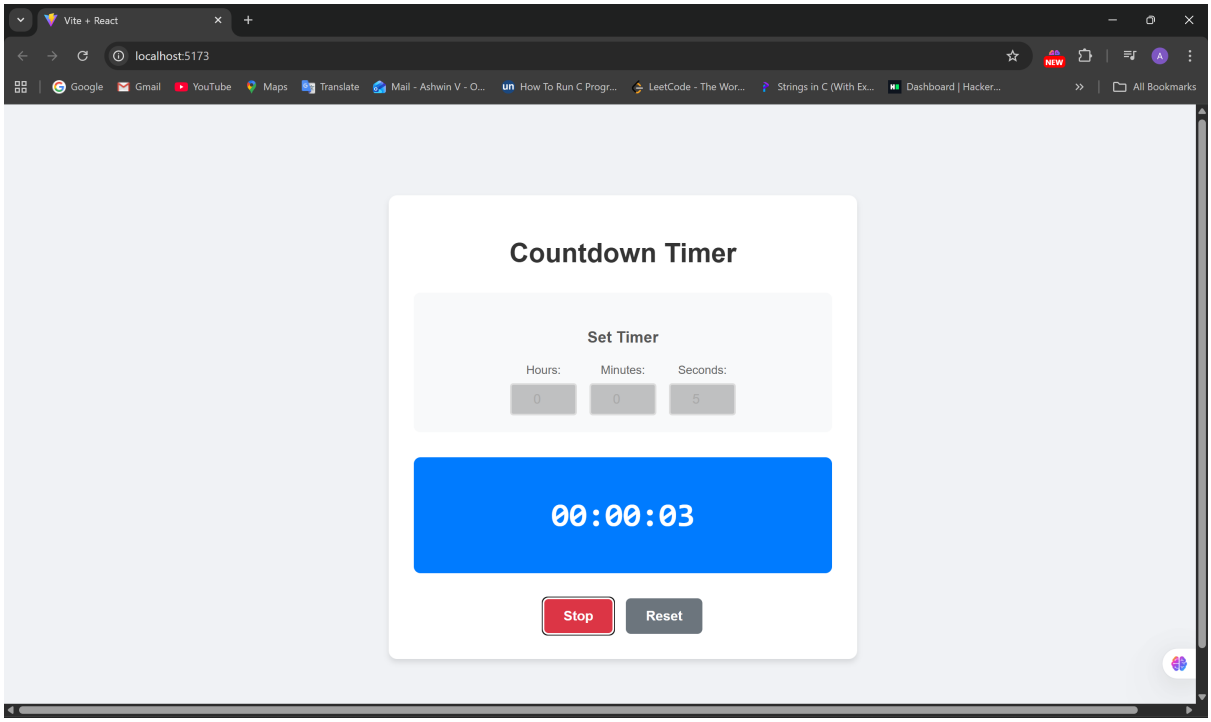


Figure 16: Component Hierarchy — Lab 6: Countdown Timer (Fan-out Structure with Prop Flow)

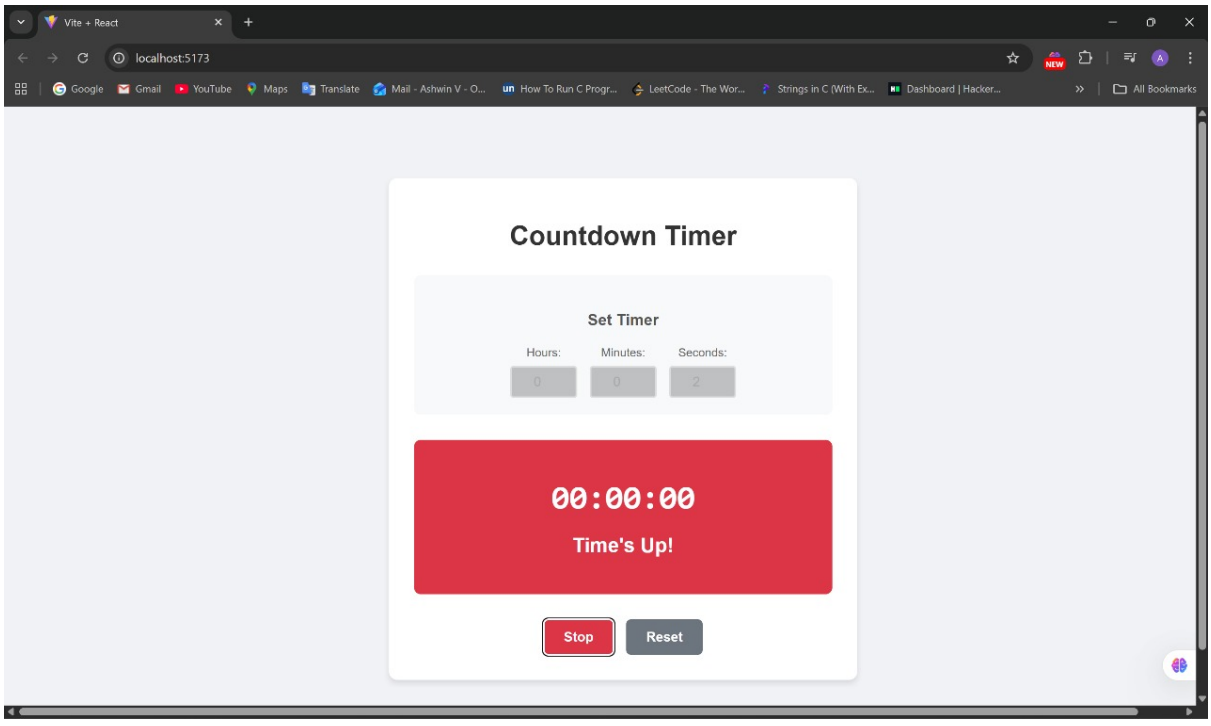
Rendered Output (Screenshot)



Screenshot 1: Countdown Timer UI



Screenshot 2: Timer Running



Screenshot 3: Timer Finished

Code Overview

App.css

```
.app {
```

```
  min-height: 100vh;
  min-width: 100vw;
  background-color: #f0f2f5;
  display: flex;
  justify-content: center;
  align-items: center;
  font-family: Arial, sans-serif;
  padding: 20px;
}
.container {
  background-color: white;
  border-radius: 10px;
  padding: 30px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  text-align: center;
  max-width: 500px;
  width: 100%;
}
.app-title {
  color: #333;
  margin-bottom: 30px;
  font-size: 2em;
}
.time-setter {
  margin-bottom: 30px;
  padding: 20px;
  background-color: #f8f9fa;
  border-radius: 8px;
}
.title {
  color: #555;
  margin-bottom: 15px;
}
.input-group {
  display: flex;
  gap: 15px;
  justify-content: center;
  flex-wrap: wrap;
}
.input-field {
  display: flex;
  flex-direction: column;
  align-items: center;
}
.label {
  margin-bottom: 5px;
  color: #666;
  font-size: 14px;
}
.input {
  width: 60px;
```

```
  padding: 8px;
  text-align: center;
  border: 2px solid #ddd;
  border-radius: 4px;
  font-size: 16px;
}
.timer-display {
  margin: 30px 0;
  padding: 40px;
  background-color: #333;
  color: white;
  border-radius: 8px;
  transition: all 0.3s ease;
}
.timer-display.active {
  background-color: #007bff;
}
.timer-display.finished {
  background-color: #dc3545;
  animation: pulse 1s infinite;
}
.time-text {
  font-size: 3em;
  font-weight: bold;
  font-family: monospace;
}
.finished-text {
  font-size: 1.5em;
  margin-top: 10px;
  font-weight: bold;
}
.control-buttons {
  display: flex;
  gap: 15px;
  justify-content: center;
  flex-wrap: wrap;
}
.button {
  padding: 12px 24px;
  font-size: 16px;
  border: none;
  border-radius: 6px;
  cursor: pointer;
  font-weight: bold;
  transition: all 0.3s ease;
  min-width: 80px;
}
.start-button {
  background-color: #28a745;
  color: white;
}
```

```
.stop-button {
  background-color: #dc3545;
  color: white;
}
.reset-button {
  background-color: #6c757d;
  color: white;
}
.disabled-button {
  background-color: #ccc;
  cursor: not-allowed;
}
```

App.jsx

```
import { useState, useEffect } from "react";
import Title from "./components/Title";
import ControlButtons from "./components/ControlButtons";
import TimerDisplay from "./components/TimerDisplay";
import TimeSetter from "./components/TimeSetter";
import "./App.css";

// Main App Component
function App() {
  const [hours, setHours] = useState(0);
  const [minutes, setMinutes] = useState(5);
  const [seconds, setSeconds] = useState(0);
  const [isActive, setIsActive] = useState(false);
  const [timeLeft, setTimeLeft] = useState({
    hours: 0,
    minutes: 5,
    seconds: 0,
  });

  // Calculate if there's any time set
  const hasTime = hours > 0 || minutes > 0 || seconds > 0;
  const isFinished =
    timeLeft.hours === 0 &&
    timeLeft.minutes === 0 &&
    timeLeft.seconds === 0 &&
    isActive;

  // Handle time changes from TimeSetter
  const handleTimeChange = (type, value) => {
    if (!isActive) {
      switch (type) {
        case "hours":
          setHours(value);
          setTimeLeft((prev) => ({ ...prev, hours: value }));
          break;
```

```
        case "minutes":
          setMinutes(value);
          setTimeLeft((prev) => ({ ...prev, minutes: value }));
          break;
        case "seconds":
          setSeconds(value);
          setTimeLeft((prev) => ({ ...prev, seconds: value }));
          break;
      }
    }
  };

  // Start timer
  const handleStart = () => {
    if (hasTime) {
      setIsActive(true);
      setTimeLeft({ hours, minutes, seconds });
    }
  };

  // Stop timer
  const handleStop = () => {
    setIsActive(false);
  };

  // Reset timer
  const handleReset = () => {
    setIsActive(false);
    setTimeLeft({ hours, minutes, seconds });
  };

  // Timer countdown effect
  useEffect(() => {
    let interval = null;

    if (isActive) {
      interval = setInterval(() => {
        setTimeLeft((prevTime) => {
          const { hours: h, minutes: m, seconds: s } = prevTime;

          if (h === 0 && m === 0 && s === 0) {
            setIsActive(false);
            return prevTime;
          }

          if (s > 0) {
            return { ...prevTime, seconds: s - 1 };
          } else if (m > 0) {
            return { ...prevTime, minutes: m - 1, seconds: 59 };
          } else if (h > 0) {
```

```
            return { ...prevTime, hours: h - 1, minutes: 59,
               seconds: 59 };
          }

          return prevTime;
        });
      }, 1000);
    }

    return () => {
      if (interval) clearInterval(interval);
    };
  }, [isActive]);

  return (
    <div className="app">
      <div className="container">
        <Title text="Countdown Timer" />

        <TimeSetter
          hours={hours}
          minutes={minutes}
          seconds={seconds}
          onTimeChange={handleTimeChange}
          disabled={isActive}
        />

        <TimerDisplay
          hours={timeLeft.hours}
          minutes={timeLeft.minutes}
          seconds={timeLeft.seconds}
          isActive={isActive}
          isFinished={isFinished}
        />

        <ControlButtons
          isActive={isActive}
          onStart={handleStart}
          onStop={handleStop}
          onReset={handleReset}
          hasTime={hasTime}
        />
      </div>
    </div>
  );
}

export default App;
```

Title.jsx

```
function Title({ text }) {
  return <h1 className="app-title">{text}</h1>;
}

export default Title;
```

TimeSetter.jsx

```
// TimeSetter Component
function TimeSetter({
  hours,
  minutes,
  seconds,
  onTimeChange,
  disabled,
}) {
  const handleInputChange = (type, value) => {
    const numValue = Math.max(0, parseInt(value) || 0);
    onTimeChange(type, numValue);
  };

  return (
    <div className="time-setter">
      <h3 className="title">Set Timer</h3>
      <div className="input-group">
        <div className="input-field">
          <label className="label">Hours:</label>
          <input
            type="number"
            min="0"
            max="23"
            value={hours}
            onChange={(e) => handleInputChange("hours", e.target.
               value)}
            disabled={disabled}
            className="input"
          />
        </div>
        <div className="input-field">
          <label className="label">Minutes:</label>
          <input
            type="number"
            min="0"
            max="59"
            value={minutes}
            onChange={(e) => handleInputChange("minutes", e.
               target.value)}
            disabled={disabled}
```

```
            className="input"
          />
        </div>
        <div className="input-field">
          <label className="label">Seconds:</label>
          <input
            type="number"
            min="0"
            max="59"
            value={seconds}
            onChange={(e) => handleInputChange("seconds", e.
               target.value)}
            disabled={disabled}
            className="input"
          />
        </div>
      </div>
    </div>
  );
}

export default TimeSetter;
```

TimerDisplay.jsx

```
// TimerDisplay Component
function TimerDisplay({
  hours,
  minutes,
  seconds,
  isActive,
  isFinished,
}) {
  const formatTime = (time) => time.toString().padStart(2, "0");

  const timerDisplayClasses = [
    "timer-display",
    isActive ? "active" : "",
    isFinished ? "finished" : "",
  ].join(" ");

  return (
    <div className={timerDisplayClasses}>
      <div className="time-text">
        {formatTime(hours)}:{formatTime(minutes)}:{formatTime(
           seconds)}
      </div>
      {isFinished && <div className="finished-text">Time's Up!</
         div>}
    </div>
```

```
  );
}

export default TimerDisplay;
```

ControlButtons.jsx

```
// ControlButtons Component
function ControlButtons({
  isActive,
  onStart,
  onStop,
  onReset,
  hasTime,
}) {
  return (
    <div className="control-buttons">
      {!isActive ? (
        <button
          onClick={onStart}
          disabled={!hasTime}
          className={`button start-button ${!hasTime ? "disabled-
             button" : ""}`}>
          Start
        </button>
      ) : (
        <button
          onClick={onStop}
          className="button stop-button">
          Stop
        </button>
      )}
      <button
        onClick={onReset}
        className="button reset-button">
        Reset
      </button>
    </div>
  );
}

export default ControlButtons;
```

Result

The countdown timer web application was successfully designed and implemented using React functional components. The timer logic was managed using `useState` and `useEffect` hooks to update the countdown in real-time. The design supports modular component-based architecture with clear prop communication.

Conclusion

This exercise demonstrates effective use of React hooks for managing side effects (intervals) and dynamic state updates. By splitting logic across four functional components, the app achieves a clean, maintainable, and scalable design.

Lab Exercise 7: Multi-Page React Application using React Router

Question

Build a React application using React Router with the following routes:

- `/Home`
- `/Education`
- `/Skills`
- `/Projects`
- `/Experience`
- `/Achievements`
- `/Thermostat`
- `/Timer`

The **Home** page should include a short personal summary and an optional profile picture. Routes **Education**, **Skills**, **Projects**, **Experience**, and **Achievements** reflect the student's CV. The **Thermostat** route integrates the Lab 5 temperature-converter logic, and the **Timer** route integrates the Lab 6 countdown timer logic.

The app must use functional components, `props`, `useState`, `useEffect`, and React Router for navigation. A component hierarchy diagram should also be included.

Design Description

This project represents a complete single-page React application using React Router v6. The design consists of eight distinct routes managed by `BrowserRouter`, each rendering its own component tree.

The app demonstrates:

- Multi-page routing with navigation links.
- Component reuse across multiple routes.
- Hook-based state management for dynamic pages (`Thermostat` and `Timer`).
- A clean, scalable folder structure with modular components.

Main Components and Props

Router Setup

- **App.jsx** - Root component defining all routes and shared layout. Uses `BrowserRouter`, `Routes`, and `Route`. *Props:* None.

CV-related Routes

- **Home.jsx** - Displays personal summary and profile picture. *Props:* `name`, `profileImage`, `bio`.
- **Education.jsx** - Lists academic qualifications. *Props:* `degrees`, `institutions`, `years`.
- **Skills.jsx** - Displays technical and soft skills. *Props:* `skillList`.
- **Projects.jsx** - Renders notable project summaries. *Props:* `projects` (array of objects with title, description, and links).
- **Experience.jsx** - Describes internship and professional work. *Props:* `role`, `organization`, `duration`.
- **Achievements.jsx** - Lists personal and academic achievements. *Props:* `achievements`.

Functional Routes (Integrated from Previous Labs)

- **Thermostat.jsx** - Imports components from Lab 5. Uses:
  - `TemperatureDisplay.jsx`
  - `TemperatureInC.jsx`
  - `FahrenheitDisplay.jsx`
  - `TemperatureControls.jsx`
  - `ConversionButton.jsx`

  Props: `temperatureC`, `onIncrease`, `onDecrease`. *Hooks:* `useState` for temperature management.
- **Timer.jsx** - Imports timer logic from Lab 6. Uses:
  - `TimerDisplay.jsx`
  - `TimeSetter.jsx`
  - `ControlButtons.jsx`
  - `Title.jsx`

  Props: None (state managed internally). *Hooks:* `useState`, `useEffect`.

Component Hierarchy Diagram

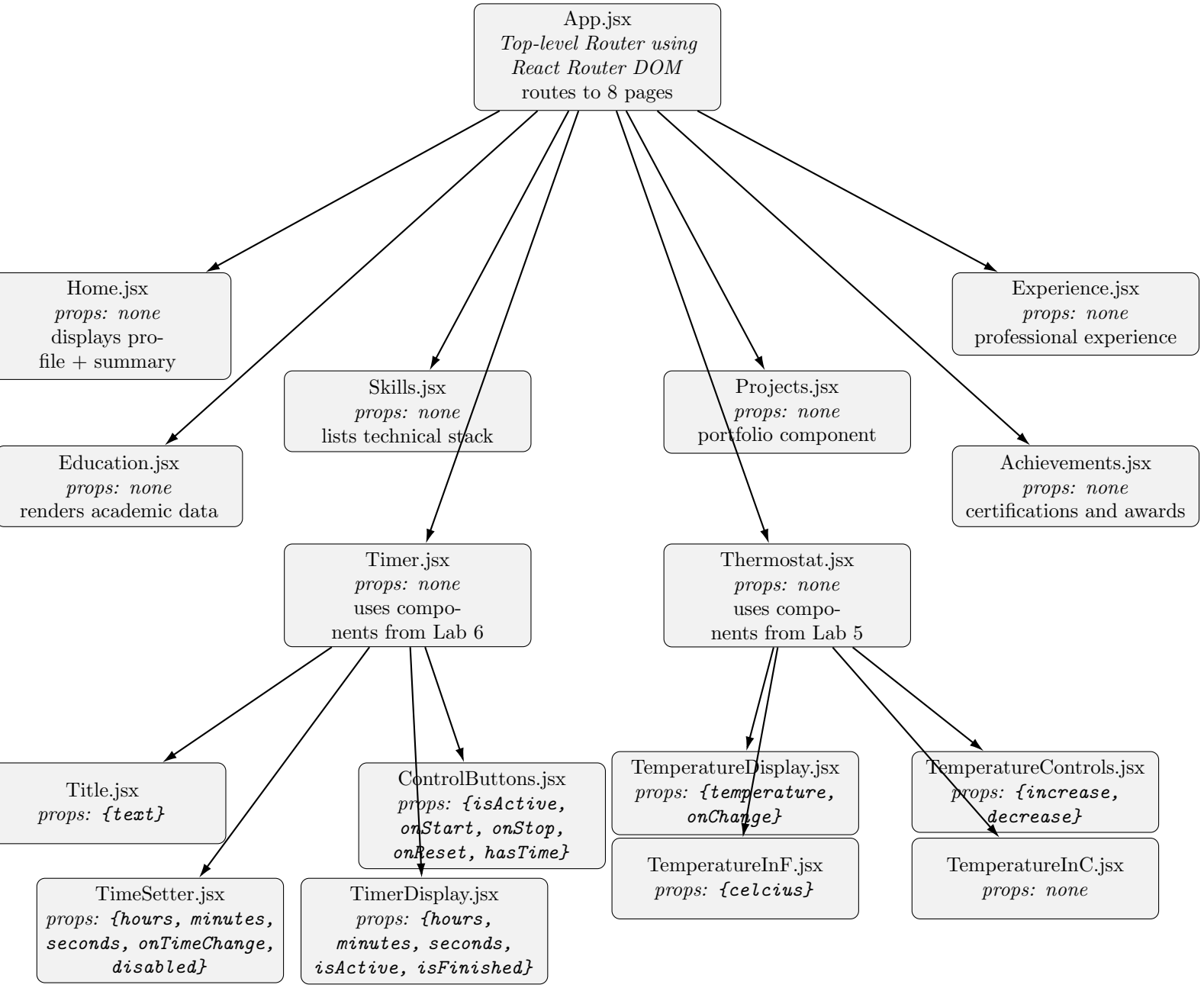
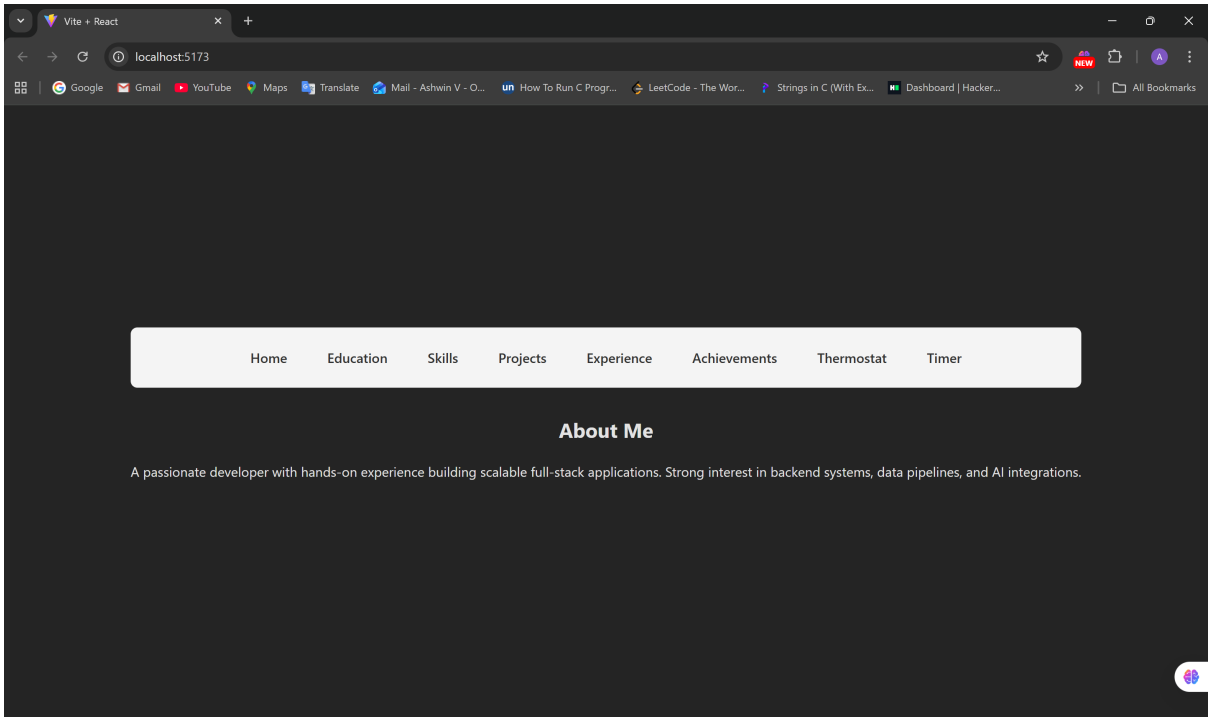
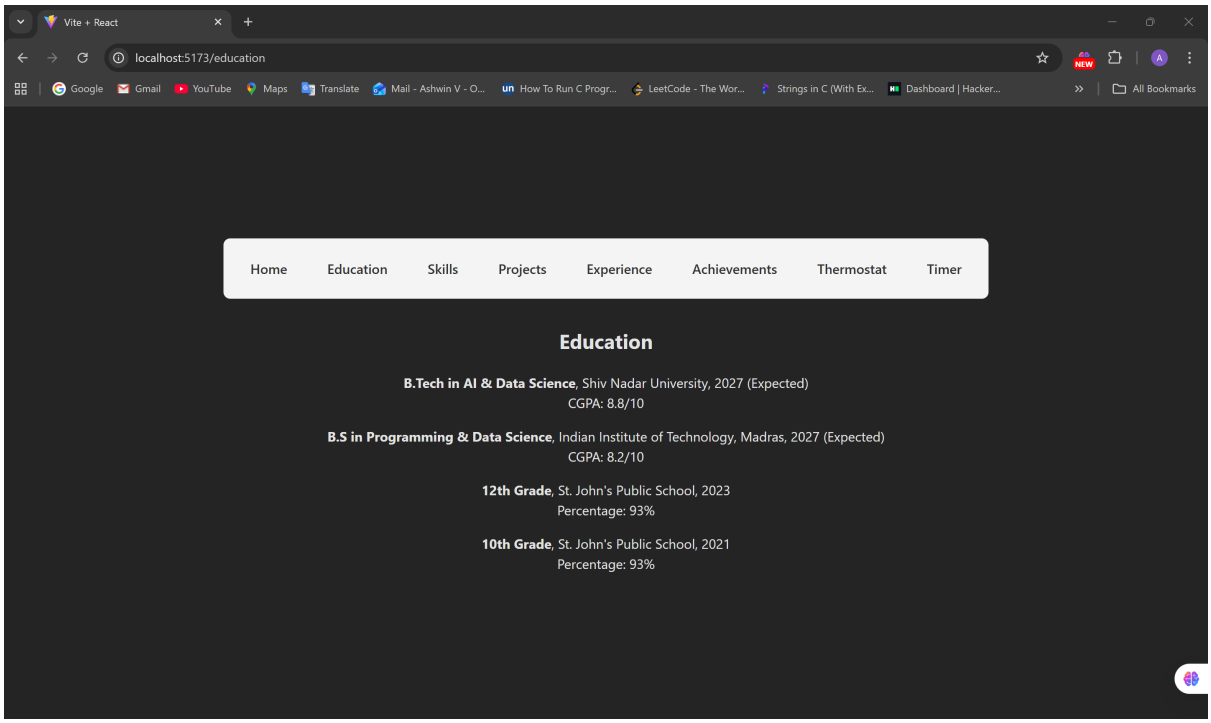


Figure 20: Component Hierarchy — Lab 7: React Router Application (Routes + Nested Thermostat and Timer Components)

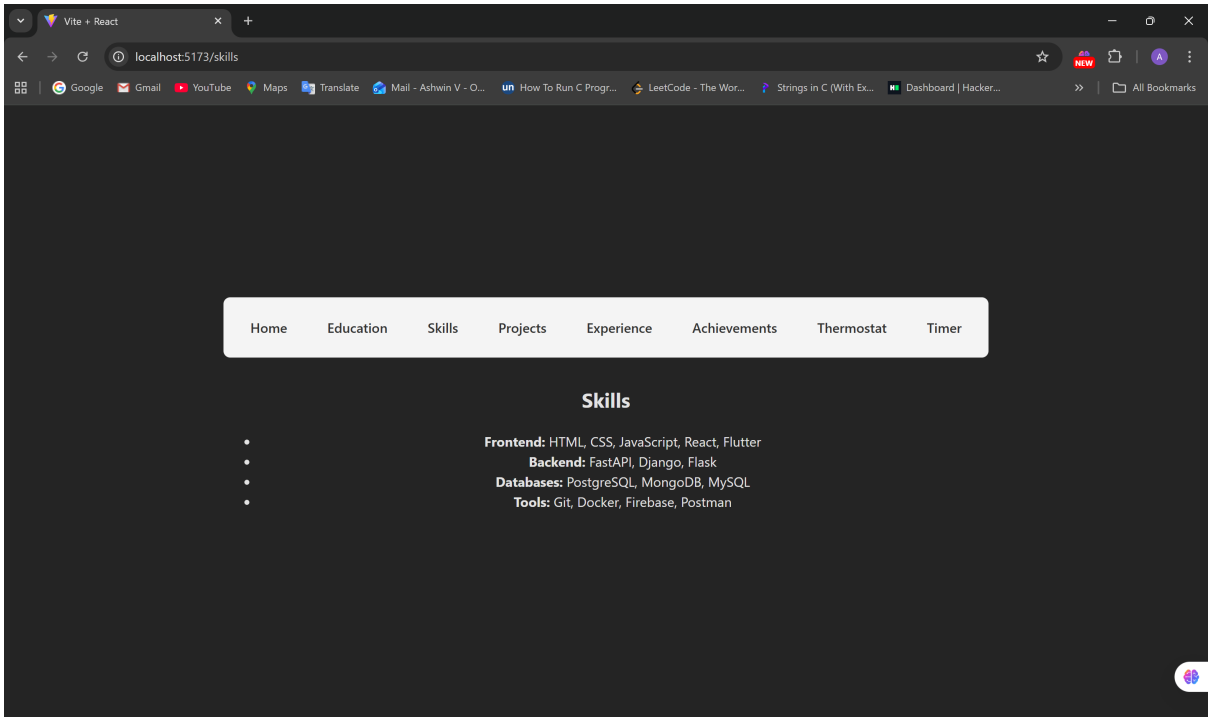
Rendered Output (Screenshot)



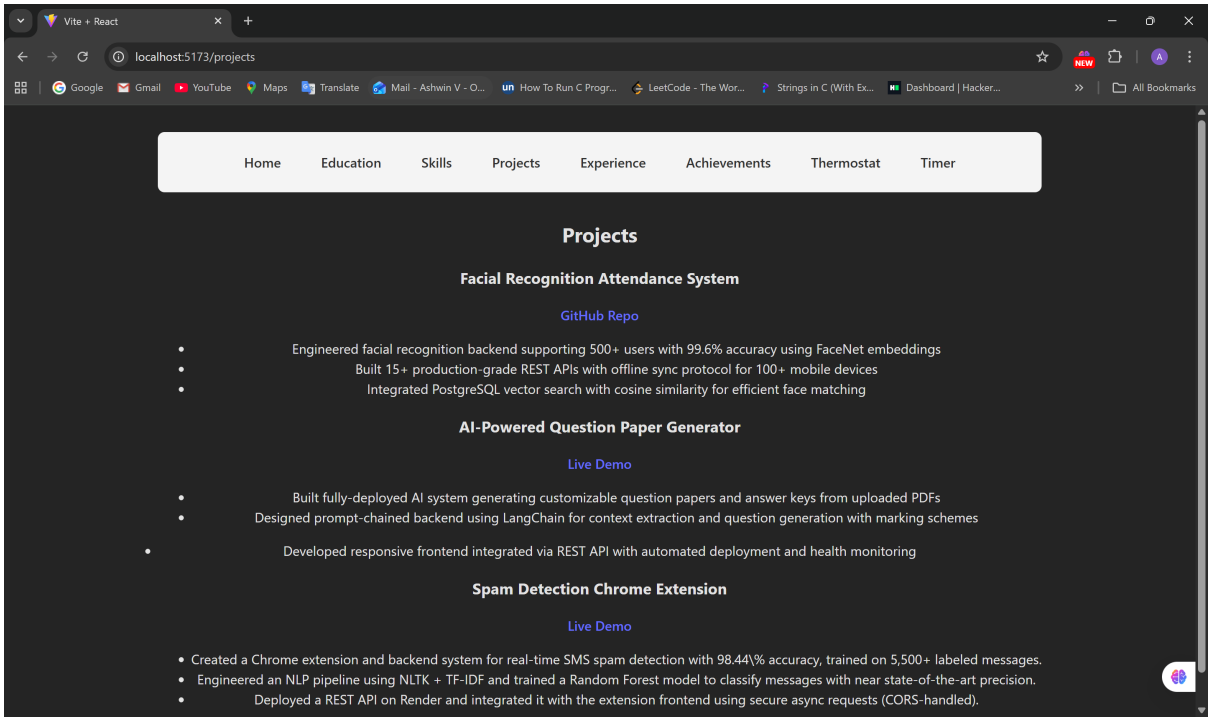
Screenshot 1: Home Page



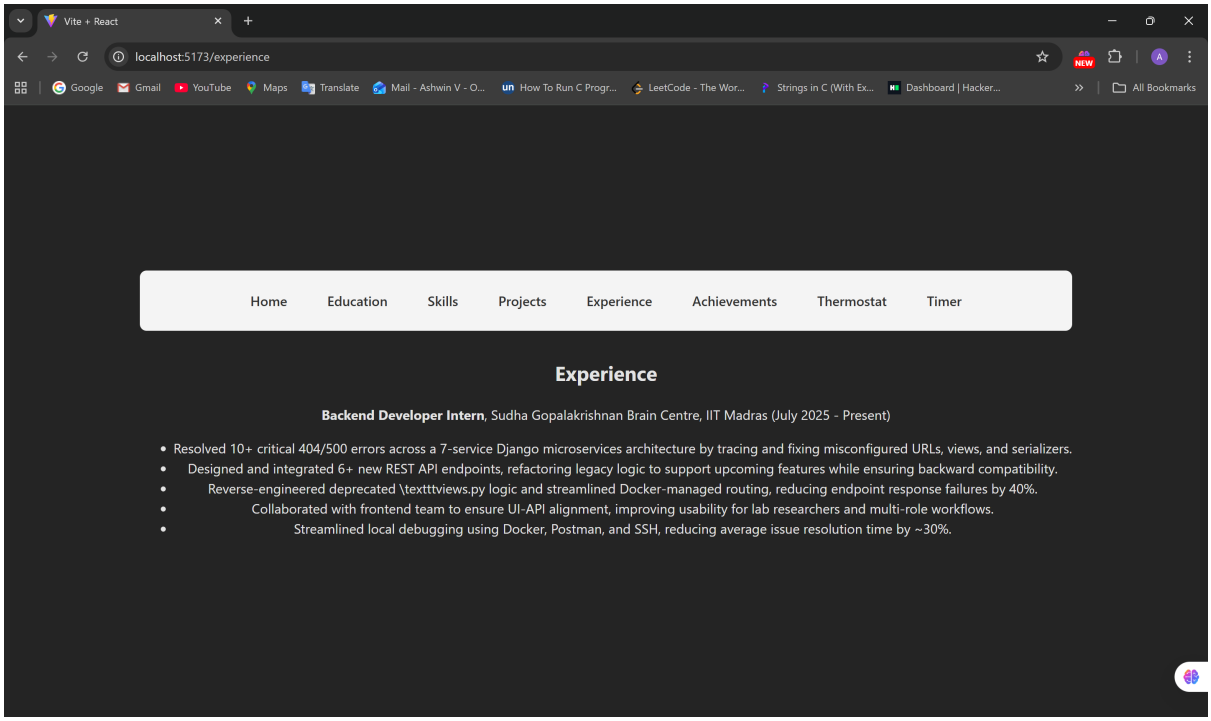
Screenshot 2: Education Page



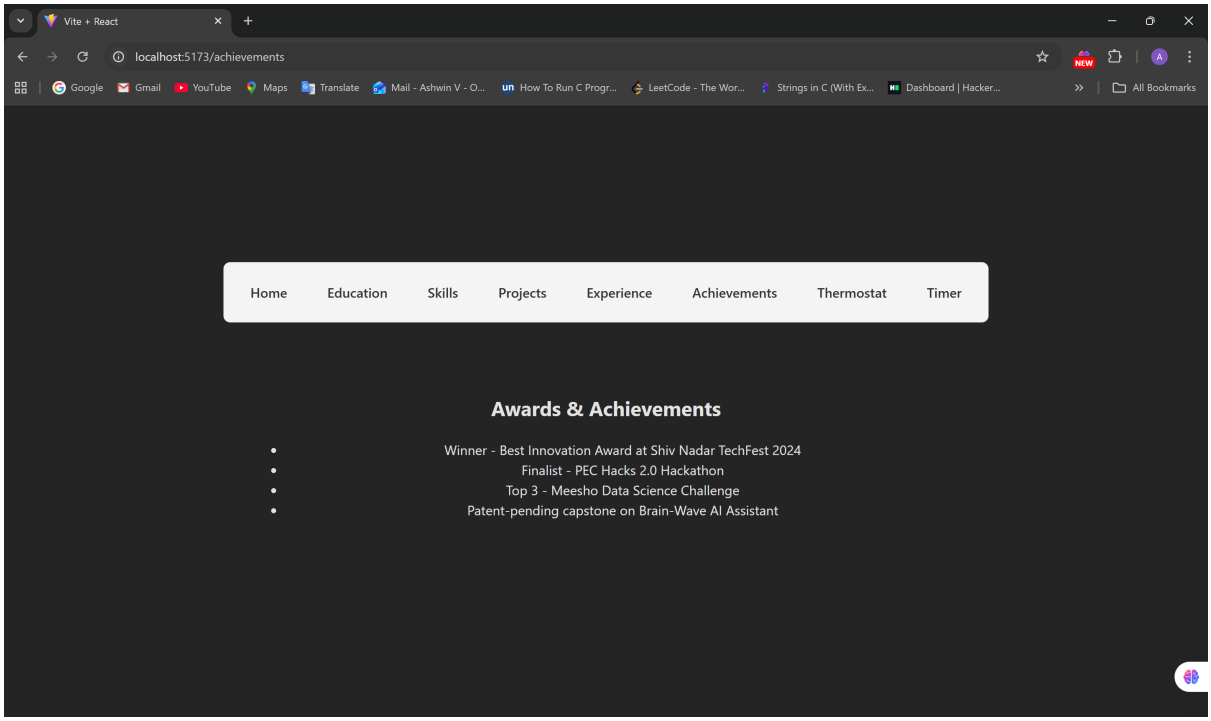
Screenshot 3: Skills Page



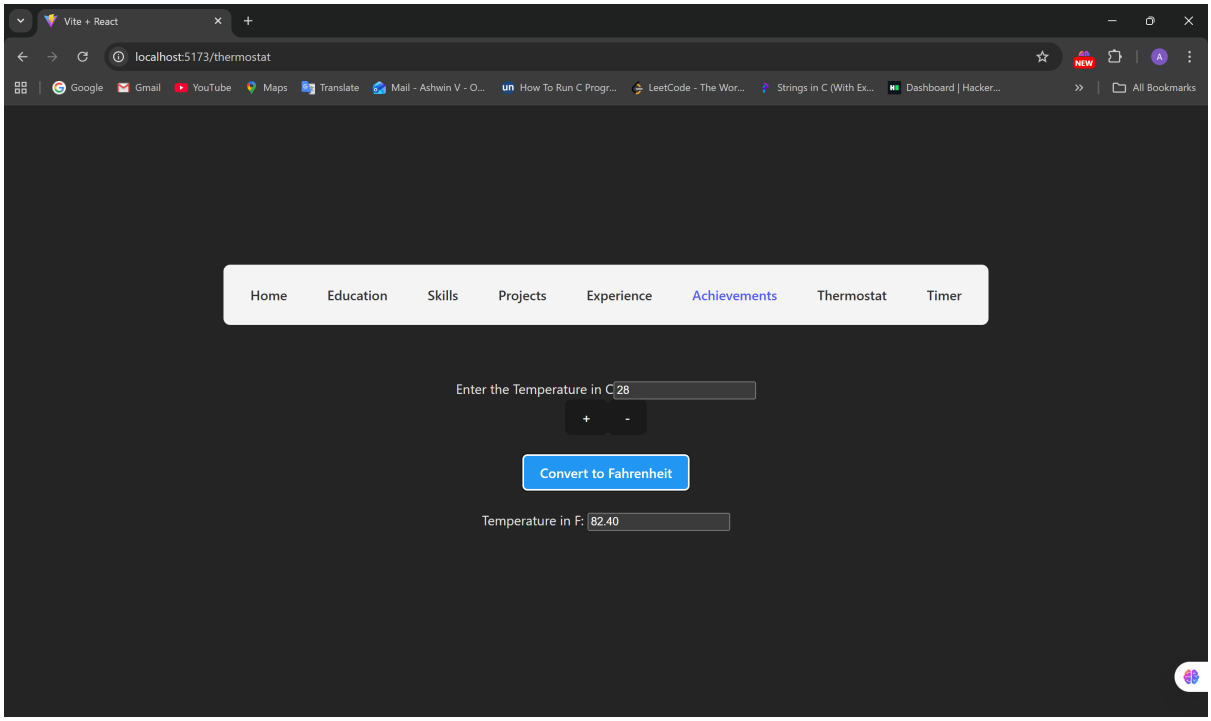
Screenshot 4: Projects Page



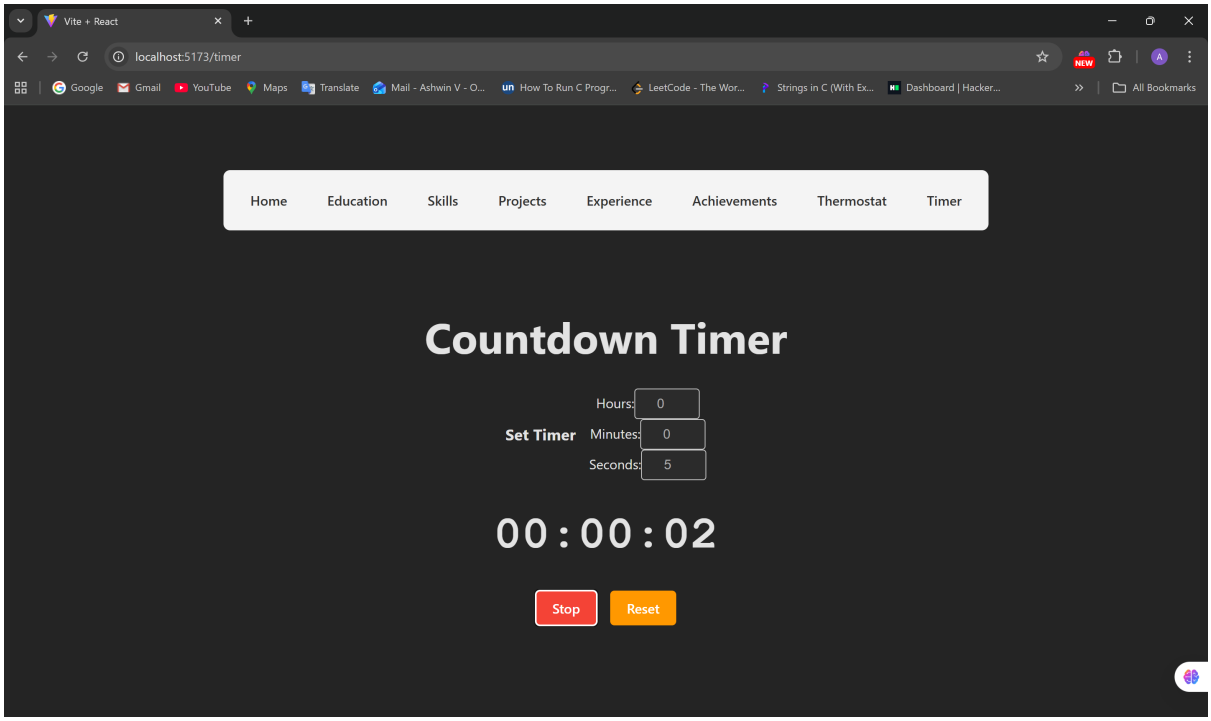
Screenshot 5: Experience Page



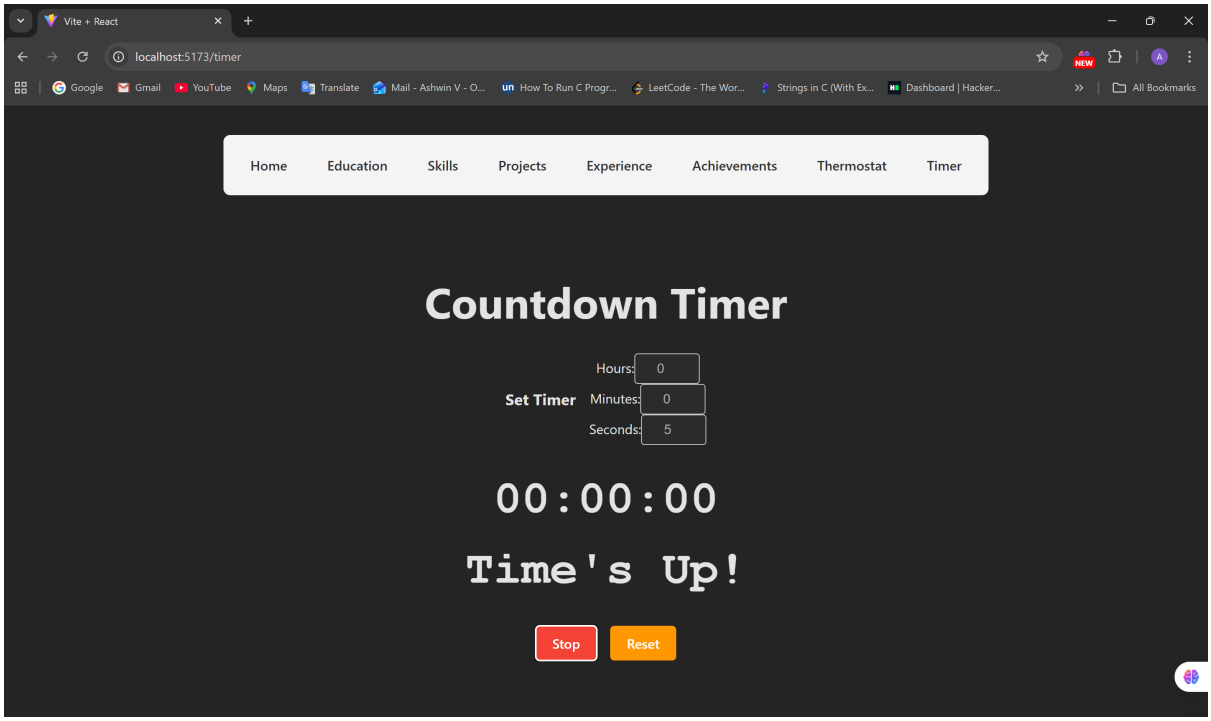
Screenshot 6: Achievements Page



Screenshot 7: Thermostat Page



Screenshot 8: Timer Page Running



Screenshot 9: Timer Page On Finish

Code Overview

App.css

```
/* General App Layout */
#root, .container {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}

/* Navigation Bar */
nav {
  display: flex;
  justify-content: center;
  gap: 1rem;
  padding: 1rem;
  background-color: #f4f4f4;
  border-radius: 8px;
  margin-bottom: 2rem;
}

nav a {
  text-decoration: none;
  color: #333;
  font-weight: 500;
  padding: 0.5rem 1rem;
  border-radius: 5px;
```

```
  transition: background-color 0.3s ease;
}

nav a:hover, nav a.active {
  background-color: #ddd;
}


/* Card Style */
.card {
  padding: 2em;
  border-radius: 12px;
  box-shadow: 0 4px 8px rgba(0,0,0,0.1);
  background-color: #fff;
  margin-bottom: 2rem;
}

.read-the-docs {
  color: #888;
}

/* Base Button Styles */
.button {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-size: 1rem;
  transition: background-color 0.3s ease, transform 0.2s ease;
}

.button:hover {
  transform: translateY(-2px);
}

.button:active {
  transform: translateY(0);
}

.button.disabled-button {
  background-color: #ccc;
  cursor: not-allowed;
}

/* Control Buttons (Start, Stop, Reset) */
.control-buttons {
  display: flex;
  gap: 1rem;
  justify-content: center;
  margin-top: 1rem;
}
```

```
.start-button {
  background-color: #4CAF50; /* Green */
  color: white;
}

.stop-button {
  background-color: #f44336; /* Red */
  color: white;
}

.reset-button {
  background-color: #ff9800; /* Orange */
  color: white;
}

/* Conversion Button */
.conversion-button button {
  background-color: #2196F3; /* Blue */
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-size: 1rem;
  transition: background-color 0.3s ease;
}

.conversion-button button:hover {
  background-color: #0b7dda;
}

/* Thermostat Styles */
.thermostat {
  padding: 2rem;
  border: 1px solid #ccc;
  border-radius: 8px;
  display: inline-block;
}

.temperature-display {
  font-size: 3rem;
  font-weight: bold;
  margin: 1rem 0;
}

.temperature-controls button {
  margin: 0 0.5rem;
  padding: 0.5rem 1rem;
  font-size: 1.2rem;
}
```

```
/* Timer Styles */
.timer {
    padding: 2rem;
    border: 1px solid #ccc;
    border-radius: 8px;
    display: inline-block;
    min-width: 300px;
}

.timer-display {
    font-size: 3.5rem;
    font-weight: bold;
    margin-bottom: 1.5rem;
    font-family: 'Courier New', Courier, monospace;
}

.time-setter {
    display: flex;
    justify-content: center;
    gap: 1rem;
    align-items: center;
    margin-bottom: 1.5rem;
}

.time-setter input {
    width: 60px;
    padding: 0.5rem;
    text-align: center;
    font-size: 1rem;
    border: 1px solid #ccc;
    border-radius: 4px;
}
```

App.jsx (Router Setup)

```
import "./App.css";
import { BrowserRouter as Router, Routes, Link, Route } from "
   react-router-dom";
import Home from "./components/Home";
import Education from "./components/Education";
import Skills from "./components/Skills";
import Projects from "./components/Projects";
import Experience from "./components/Experience";
import Achievements from "./components/Achievements";
import Thermostat from "./components/Thermostat";
import Timer from "./components/Timer";

function App() {
  return (
```

```
    <>
      <Router>
        <nav>
          <Link to="/">Home</Link>
          <Link to="/education">Education</Link>
          <Link to="/skills">Skills</Link>
          <Link to="/projects">Projects</Link>
          <Link to="/experience">Experience</Link>
          <Link to="/achievements">Achievements</Link>
          <Link to="/thermostat">Thermostat</Link>
          <Link to="/timer">Timer</Link>
        </nav>

        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/education" element={<Education />} />
          <Route path="/skills" element={<Skills />} />
          <Route path="/projects" element={<Projects />} />
          <Route path="/experience" element={<Experience />} />
          <Route path="/achievements" element={<Achievements />}
             />
          <Route path="/thermostat" element={<Thermostat />} />
          <Route path="/timer" element={<Timer />} />
        </Routes>
      </Router>
    </>
  );
}

export default App;
```

CV Components

```
function Home() {
  return (
    <>
      <div className="about">
        <h2>About Me</h2>
        <p>
          A passionate developer with hands-on experience
             building scalable
          full-stack applications. Strong interest in backend
             systems, data
          pipelines, and AI integrations.
        </p>
      </div>
    </>
  );
}
```

```
export default Home;
```

```
function Education() {
  return (
    <>
      <div className="education">
        <h2>Education</h2>
        <p>
          <strong>B.Tech in AI & Data Science</strong>, Shiv
             Nadar University,
          2027 (Expected)
          <br />
          CGPA: 8.8/10
        </p>
        <p>
          <strong>B.S in Programming & Data Science</strong>,
             Indian Institute
          of Technology, Madras, 2027 (Expected)
          <br />
          CGPA: 8.2/10
        </p>
        <p>
          <strong>12th Grade</strong>, St. John's Public School,
             2023
          <br />
          Percentage: 93%
        </p>
        <p>
          <strong>10th Grade</strong>, St. John's Public School,
             2021
          <br />
          Percentage: 93%
        </p>
      </div>
    </>
  );
}

export default Education;
```

```
function Skills() {
  return (
    <>
      <div className="skills">
        <h2>Skills</h2>
        <ul>
          <li>
            <strong>Frontend:</strong> HTML, CSS, JavaScript,
               React, Flutter
          </li>
          <li>
```

```
            <strong>Backend:</strong> FastAPI, Django, Flask
          </li>
          <li>
            <strong>Databases:</strong> PostgreSQL, MongoDB,
               MySQL
          </li>
          <li>
            <strong>Tools:</strong> Git, Docker, Firebase,
               Postman
          </li>
        </ul>
      </div>
    </>
  );
}

export default Skills;
```

```
function Projects() {
  return (
    <>
      <div className="projects">
        <h2>Projects</h2>
        <div class="project">
          <h3>Facial Recognition Attendance System</h3>
          <a href="https://github.com/ashvp/face-attendance"
             target="_blank">
            GitHub Repo
          </a>
          <ul>
            <li>
              Engineered facial recognition backend supporting
                 500+ users with
              99.6% accuracy using FaceNet embeddings
            </li>
            <li>
              Built 15+ production-grade REST APIs with offline
                 sync protocol
              for 100+ mobile devices
            </li>
            <li>
              Integrated PostgreSQL vector search with cosine
                 similarity for
              efficient face matching
            </li>
          </ul>
        </div>
        <div class="project">
          <h3>AI-Powered Question Paper Generator</h3>
          <a
            href="https://question-paper-generator-nine.vercel.
```

```
            app/"
          target="_blank"
        >
          Live Demo
        </a>
        <ul>
          <li>
            Built fully-deployed AI system generating
               customizable question
            papers and answer keys from uploaded PDFs
          </li>
          <li>
            Designed prompt-chained backend using LangChain for
                context
            extraction and question generation with marking
               schemes
          </li>
        </ul>
        <li>
          Developed responsive frontend integrated via REST API
              with automated
          deployment and health monitoring
        </li>
      </div>

      <div class="project">
        <h3>Spam Detection Chrome Extension</h3>
        <a
          href="https://question-paper-generator-nine.vercel.
             app/"
          target="_blank"
        >
          Live Demo
        </a>
        <ul>
          <li>
            Created a Chrome extension and backend system for
               real-time SMS
            spam detection with 98.44\% accuracy, trained on
               5,500+ labeled
            messages.
          </li>
          <li>
            Engineered an NLP pipeline using NLTK + TF-IDF and
               trained a
            Random Forest model to classify messages with near
            state-of-the-art precision.
          </li>
          <li>
            Deployed a REST API on Render and integrated it
               with the extension
```

```
              frontend using secure async requests (CORS-handled)
                  .
            </li>
          </ul>{" "}
        </div>
      </div>
    </>
  );
}

export default Projects;
```

```
function Experience() {
  return (
    <>
      <div className="experience">
        <h2>Experience</h2>
        <p>
          <strong>Backend Developer Intern</strong>, Sudha
             Gopalakrishnan Brain
          Centre, IIT Madras (July 2025 - Present)
        </p>
        <ul>
          <li>
            {" "}
            Resolved 10+ critical 404/500 errors across a 7-
               service Django
            microservices architecture by tracing and fixing
               misconfigured URLs,
            views, and serializers.
          </li>
          <li>
            {" "}
            Designed and integrated 6+ new REST API endpoints,
               refactoring
            legacy logic to support upcoming features while
               ensuring backward
            compatibility.
          </li>
          <li>
            {" "}
            Reverse-engineered deprecated \texttt{"views.py"}
               logic and
            streamlined Docker-managed routing, reducing endpoint
                response
            failures by 40%.
          </li>
          <li>
            {" "}
            Collaborated with frontend team to ensure UI-API
               alignment,
```

```
            improving usability for lab researchers and multi-
               role workflows.
          </li>
          <li>
            {" "}
            Streamlined local debugging using Docker, Postman,
               and SSH, reducing
            average issue resolution time by ~30%.
          </li>
        </ul>
      </div>
    </>
  );
}

export default Experience;
```

```
function Achievements() {
  return (
    <>
      <div class="container">
        <h2>Awards & Achievements</h2>
        <ul>
          <li>Winner - Best Innovation Award at Shiv Nadar
             TechFest 2024</li>
          <li>Finalist - PEC Hacks 2.0 Hackathon</li>
          <li>Top 3 - Meesho Data Science Challenge</li>
          <li>Patent-pending capstone on Brain-Wave AI Assistant
             </li>
        </ul>
      </div>
    </>
  );
}

export default Achievements;
```

Thermostat Components (From Lab 5)

```
// import "./App.css";
import TemperatureInC from "./TemperatureInC";

// Main application component
function Thermostat() {
  return (
    <div className="container">
      <TemperatureInC />
    </div>
  );
}
```

```
export default Thermostat;
```

```
import React from "react";

// Displays the temperature in Celsius
function TemperatureDisplay({ temperature, onChange }) {
  return (
    <div className="temperature-input">
      <span>Enter the Temperature in C</span>
      <input type="number" value={temperature} onChange={onChange
         }></input>
    </div>
  );
}

export default TemperatureDisplay;
```

```
import React from "react";

// Controls to increase and decrease temperature
function TemperatureControls({ increase, decrease }) {
  return (
    <div className="controls">
      <button onClick={increase}> + </button>
      <button onClick={decrease}> - </button>
    </div>
  );
}

export default TemperatureControls;
```

```
import React from 'react';

const FahrenheitDisplay = ({ temperature }) => {
  return (
    <div>
      <span>Temperature in F: </span>
      <input type="text" value={temperature} readOnly />
    </div>
  );
};

export default FahrenheitDisplay;
```

```
import React from "react";

// Button to trigger the conversion
const ConversionButton = ({ onClick }) => {
  return (
    <div className="conversion-button">
```

```
      <button onClick={onClick}>Convert to Fahrenheit</button>
    </div>
  );
};

export default ConversionButton;
```

Timer Components (From Lab 6)

```
import { useState, useEffect } from "react";
import Title from "./Title";
import ControlButtons from "./ControlButtons";
import TimerDisplay from "./TimerDisplay";
import TimeSetter from "./TimeSetter";
// import "./App.css";

// Main App Component
function App() {
  const [hours, setHours] = useState(0);
  const [minutes, setMinutes] = useState(5);
  const [seconds, setSeconds] = useState(0);
  const [isActive, setIsActive] = useState(false);
  const [timeLeft, setTimeLeft] = useState({
    hours: 0,
    minutes: 5,
    seconds: 0,
  });

  // Calculate if there's any time set
  const hasTime = hours > 0 || minutes > 0 || seconds > 0;
  const isFinished =
    timeLeft.hours === 0 &&
    timeLeft.minutes === 0 &&
    timeLeft.seconds === 0 &&
    isActive;

  // Handle time changes from TimeSetter
  const handleTimeChange = (type, value) => {
    if (!isActive) {
      switch (type) {
        case "hours":
          setHours(value);
          setTimeLeft((prev) => ({ ...prev, hours: value }));
          break;
        case "minutes":
          setMinutes(value);
          setTimeLeft((prev) => ({ ...prev, minutes: value }));
          break;
        case "seconds":
          setSeconds(value);
```

```
          setTimeLeft((prev) => ({ ...prev, seconds: value }));
          break;
      }
    }
  };

  // Start timer
  const handleStart = () => {
    if (hasTime) {
      setIsActive(true);
      setTimeLeft({ hours, minutes, seconds });
    }
  };

  // Stop timer
  const handleStop = () => {
    setIsActive(false);
  };

  // Reset timer
  const handleReset = () => {
    setIsActive(false);
    setTimeLeft({ hours, minutes, seconds });
  };

  // Timer countdown effect
  useEffect(() => {
    let interval = null;

    if (isActive) {
      interval = setInterval(() => {
        setTimeLeft((prevTime) => {
          const { hours: h, minutes: m, seconds: s } = prevTime;

          if (h === 0 && m === 0 && s === 0) {
            setIsActive(false);
            return prevTime;
          }

          if (s > 0) {
            return { ...prevTime, seconds: s - 1 };
          } else if (m > 0) {
            return { ...prevTime, minutes: m - 1, seconds: 59 };
          } else if (h > 0) {
            return { ...prevTime, hours: h - 1, minutes: 59,
               seconds: 59 };
          }

          return prevTime;
        });
      }, 1000);
```

```
    }

    return () => {
      if (interval) clearInterval(interval);
    };
  }, [isActive]);

  return (
    <div className="app">
      <div className="container">
        <Title text="Countdown Timer" />

        <TimeSetter
          hours={hours}
          minutes={minutes}
          seconds={seconds}
          onTimeChange={handleTimeChange}
          disabled={isActive}
        />

        <TimerDisplay
          hours={timeLeft.hours}
          minutes={timeLeft.minutes}
          seconds={timeLeft.seconds}
          isActive={isActive}
          isFinished={isFinished}
        />

        <ControlButtons
          isActive={isActive}
          onStart={handleStart}
          onStop={handleStop}
          onReset={handleReset}
          hasTime={hasTime}
        />
      </div>
    </div>
  );
}

export default App;
```

```
// TimerDisplay Component
function TimerDisplay({
  hours,
  minutes,
  seconds,
  isActive,
  isFinished,
}) {
  const formatTime = (time) => time.toString().padStart(2, "0");
```

```
  const timerDisplayClasses = [
    "timer-display",
    isActive ? "active" : "",
    isFinished ? "finished" : "",
  ].join(" ");

  return (
    <div className={timerDisplayClasses}>
      <div className="time-text">
        {formatTime(hours)}:{formatTime(minutes)}:{formatTime(
           seconds)}
      </div>
      {isFinished && <div className="finished-text">Time's Up!</
         div>}
    </div>
  );
}

export default TimerDisplay;
```

```
// TimeSetter Component
function TimeSetter({
  hours,
  minutes,
  seconds,
  onTimeChange,
  disabled,
}) {
  const handleInputChange = (type, value) => {
    const numValue = Math.max(0, parseInt(value) || 0);
    onTimeChange(type, numValue);
  };

  return (
    <div className="time-setter">
      <h3 className="title">Set Timer</h3>
      <div className="input-group">
        <div className="input-field">
          <label className="label">Hours:</label>
          <input
            type="number"
            min="0"
            max="23"
            value={hours}
            onChange={(e) => handleInputChange("hours", e.target.
               value)}
            disabled={disabled}
            className="input"
          />
        </div>
```

```
        <div className="input-field">
          <label className="label">Minutes:</label>
          <input
            type="number"
            min="0"
            max="59"
            value={minutes}
            onChange={(e) => handleInputChange("minutes", e.
               target.value)}
            disabled={disabled}
            className="input"
          />
        </div>
        <div className="input-field">
          <label className="label">Seconds:</label>
          <input
            type="number"
            min="0"
            max="59"
            value={seconds}
            onChange={(e) => handleInputChange("seconds", e.
               target.value)}
            disabled={disabled}
            className="input"
          />
        </div>
      </div>
    </div>
  );
}

export default TimeSetter;
```

```
// ControlButtons Component
function ControlButtons({
  isActive,
  onStart,
  onStop,
  onReset,
  hasTime,
}) {
  return (
    <div className="control-buttons">
      {!isActive ? (
        <button
          onClick={onStart}
          disabled={!hasTime}
          className={`button start-button ${!hasTime ? "disabled-
             button" : ""}`}>
          Start
        </button>
```

```
      ) : (
        <button
          onClick={onStop}
          className="button stop-button">
          Stop
        </button>
      )}
      <button
        onClick={onReset}
        className="button reset-button">
        Reset
      </button>
    </div>
  );
}

export default ControlButtons;
```

```
function Title({ text }) {
  return <h1 className="app-title">{text}</h1>;
}

export default Title;
```

Result

A fully functional React application with eight distinct routes was successfully built using React Router v6. Each route loads a unique page component, and the overall app demonstrates:

- Modular React component architecture.
- Routing integration across functional features.
- Hook-based state and effect management.
- Seamless composition of previous labs (Lab 5 + Lab 6) into new routes.

Conclusion

This lab integrates multiple independent React modules into a single navigable application. It reinforces core React concepts such as routing, props, state management, and side-effect handling with `useEffect`. The project demonstrates scalable, modular front-end design suitable for full-stack deployment.

Lab Exercise 8: Blog Dashboard using React Bootstrap

Question

Build a simple React application using React Bootstrap to create a blog dashboard with the following features:

- A responsive **Navbar** (using React Bootstrap's `Navbar` component) with links to "Home" and "Posts".
- A grid of 3-4 **Blog Cards** (using Bootstrap's `Card`, `Row`, and `Col` components).
- Each card displays a post title, a short description, and a `Read More` button (using `Button` component).
- Use functional components and props to pass post data from parent to child components.
- Draw a component hierarchy diagram.

Design Description

The application is built using React and React Bootstrap to create a clean, responsive blog dashboard UI. The dashboard includes:

- A navigation bar at the top.
- A responsive grid layout for blog posts.
- Reusable blog card components that accept post data via props.

All components are built as functional components, with `props` used for communication between the parent (`App.jsx`) and child components (`BlogCard.jsx`).

Main Components and Props

- **App.jsx** - Root component that manages the overall structure and passes post data to the dashboard.
  - *Props:* None
  - *Responsibility:* Acts as parent container, imports all components, and defines post data as an array of objects.
- **NavigationBar.jsx** - Displays the responsive navigation bar using React Bootstrap.
  - *Props:* `brandName`, `links` (array of link names or paths)
  - *Responsibility:* Provides top navigation with responsive collapse functionality.
- **Dashboard.jsx** - Renders the grid layout of all blog posts.

- *Props:* `posts` (array containing post title and description)
- *Responsibility:* Uses Bootstrap's grid system (`Row` and `Col`) to layout BlogCard components responsively.

- **BlogCard.jsx** - A reusable card component displaying each post's title, description, and "Read More" button.
  - *Props:* `title`, `description`
  - *Responsibility:* Displays data in a visually appealing Bootstrap card format.

Component Hierarchy Diagram

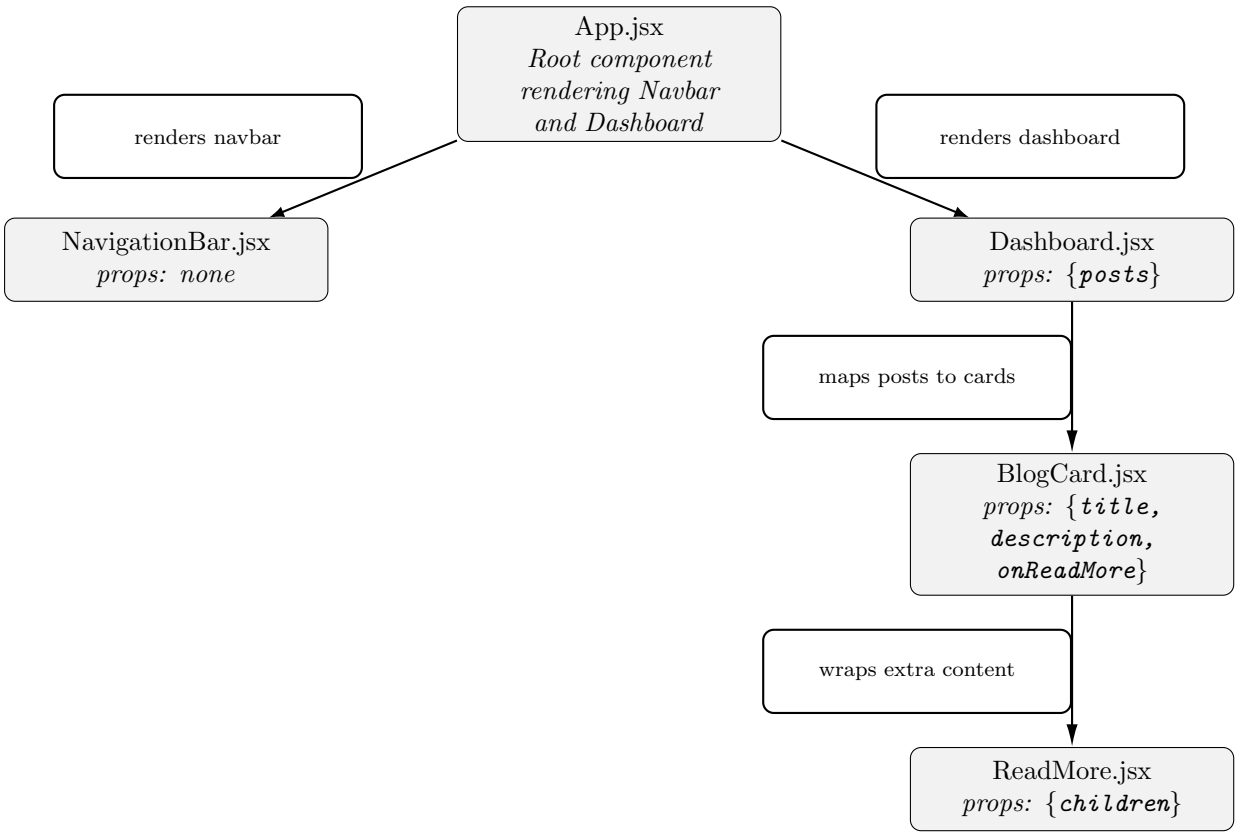
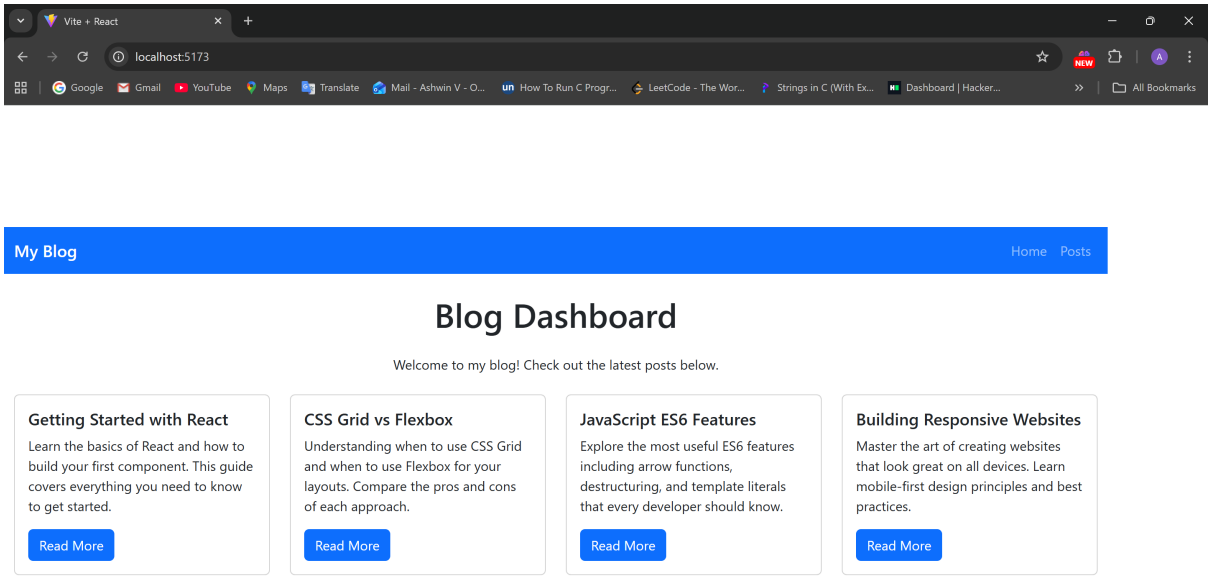
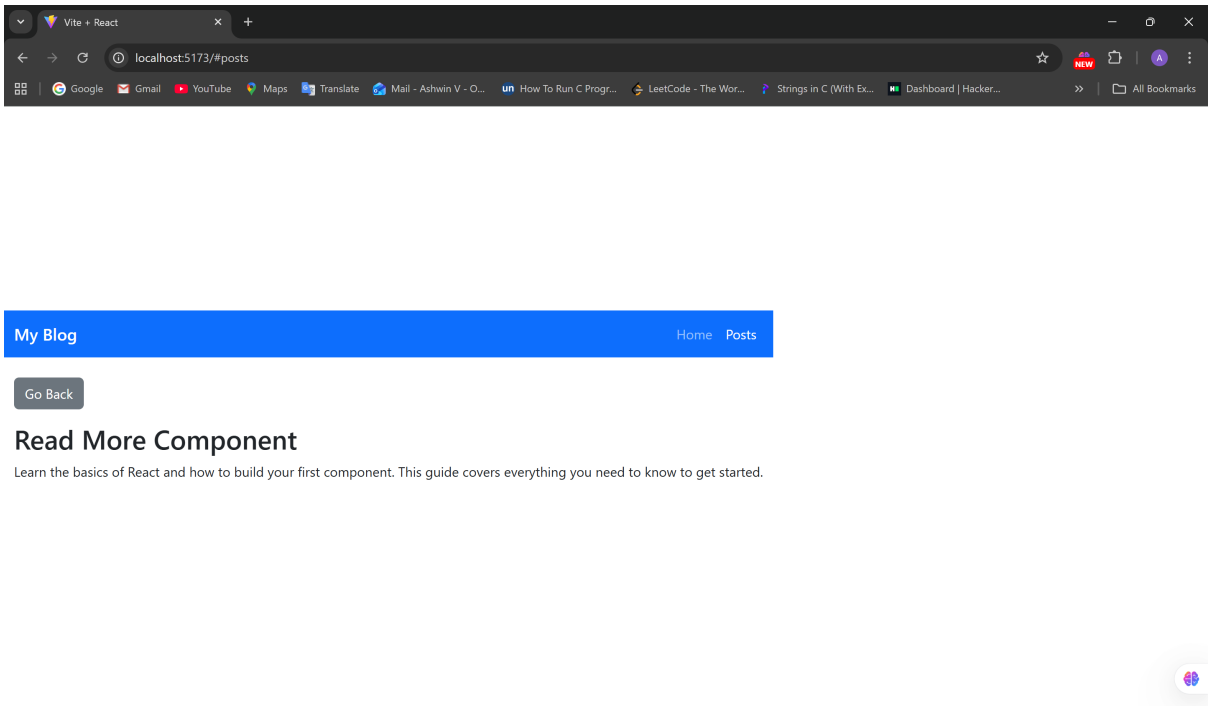


Figure 30: Component Hierarchy — Lab 8: Blog Dashboard (React + Bootstrap with Prop Flow)

Rendered Output (Screenshot)



Screenshot 1: Home Page



Screenshot 2: Read More Button Clicked

Code Overview

App.jsx

```
import React from "react";
import {
  Navbar,
  Nav,
  Container,
  Button,
  Card,
  Row,
  Col,
} from "react-bootstrap";
import NavigationBar from "./components/NavigationBar";
import Dashboard from "./components/Dashboard";

// Function to display the components
function App() {
  // Sample Blog Posts Data
  const blogPosts = [
    {
      id: 1,
      title: "Getting Started with React",
      description:
        "Learn the basics of React and how to build your first
           component. This guide covers everything you need to
           know to get started.",
    },
    {
      id: 2,
      title: "CSS Grid vs Flexbox",
      description:
        "Understanding when to use CSS Grid and when to use
           Flexbox for your layouts. Compare the pros and cons of
            each approach.",
    },
    {
      id: 3,
      title: "JavaScript ES6 Features",
      description:
        "Explore the most useful ES6 features including arrow
           functions, destructuring, and template literals that
           every developer should know.",
    },
    {
      id: 4,
      title: "Building Responsive Websites",
      description:
        "Master the art of creating websites that look great on
           all devices. Learn mobile-first design principles and
```

```
            best practices.",
    },
  ];
  return (
    <>
      <div>
        <NavigationBar />
        <Dashboard posts={blogPosts} />
      </div>
    </>
  );
}

export default App;
```

NavigationBar.jsx

```
import { Nav, Navbar, Container } from "react-bootstrap";

// Function to Display Navigation Bar
function NavigationBar() {
  return (
    <Navbar bg="primary" variant="dark">
      <Container>
        <Navbar.Brand href="#home">My Blog</Navbar.Brand>
        <Nav>
          <Nav.Link href="#home">Home</Nav.Link>
          <Nav.Link href="#posts">Posts</Nav.Link>
        </Nav>
      </Container>
    </Navbar>
  );
}

export default NavigationBar;
```

Dashboard.jsx

```
import { Container, Row } from "react-bootstrap";
import BlogCard from "./BlogCard";

// Function to Display Dashboard with Blog Posts
function Dashboard({ posts }) {
  return (
    <Container className="py-4">
      <h1 className="text-center mb-4">Blog Dashboard</h1>
      <p className="text-center mb-4">
        Welcome to my blog! Check out the latest posts below.
      </p>
```

```
      <Row>
        {posts.map((post) => (
          <BlogCard
            key={post.id}
            title={post.title}
            description={post.description}
          />
        ))}
      </Row>
    </Container>
  );
}

export default Dashboard;
```

BlogCard.jsx

```
import { Card, Col, Button } from "react-bootstrap";

// Function to Display Individual Blog Post Card
function BlogCard({ title, description }) {
  // Function to Handle Button Click
  const onClick = (title) => {
    alert(`You clicked on ${title}`);
  };

  return (
    <Col md={6} lg={3}>
      <Card>
        <Card.Body>
          <Card.Title>{title}</Card.Title>
          <Card.Text>{description}</Card.Text>
          <Button variant="primary" onClick={() => onClick(title)
             }>
            Read More
          </Button>
        </Card.Body>
      </Card>
    </Col>
  );
}

export default BlogCard;
```

Result

A fully functional, responsive React blog dashboard was successfully created using React Bootstrap. The Navbar and Cards adapt automatically to screen sizes, and component-level reusability was achieved through prop-driven data passing.

Conclusion

This lab demonstrates practical usage of React Bootstrap components for building responsive, modular UIs. By combining props, grid layouts, and reusable components, the application achieves scalability, maintainability, and professional front-end design.

Lab Exercise 9: Basic Blog Server using Node.js HTTP Module

Question

Build a simple Node.js blog server using only the built-in `http` module by creating a `server.js` file. Implement basic routing as follows:

- A **GET endpoint** at `/` that returns an HTML home page with inline navigation links to "Home" (`/`) and "Posts" (`/posts`).
- A **GET endpoint** at `/posts` that dynamically generates an HTML page displaying 3-4 blog posts (titles and descriptions) from a hardcoded array using a loop.
- A **404 handler** for unmatched routes, returning a styled HTML error message.

Design Description

This lab demonstrates the creation of a minimal web server in Node.js without using external frameworks like Express. The server handles routing manually using conditional logic on the request URL, serving different pages based on the endpoint.

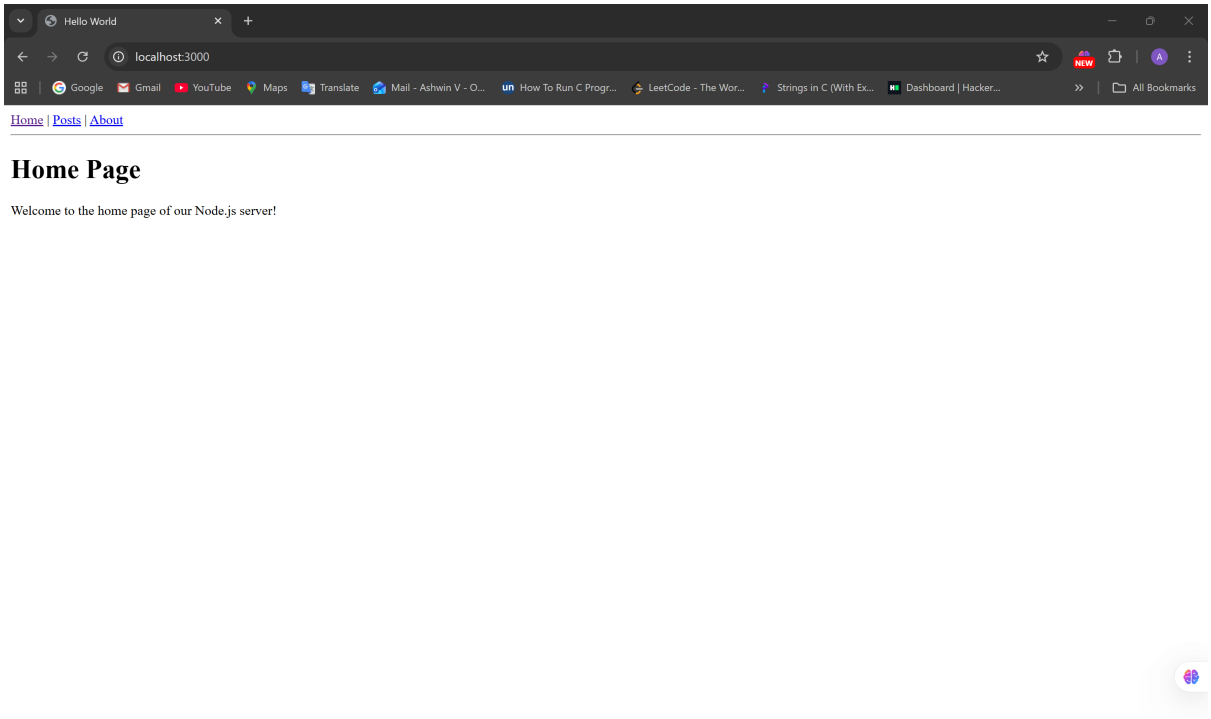
The implementation includes:

- **Home Page:** Static HTML file with navigation links to other routes.
- **Posts Page:** Dynamically generated HTML content using a hardcoded JavaScript array of post objects.
- **Error Handling:** Custom 404 response for undefined routes.

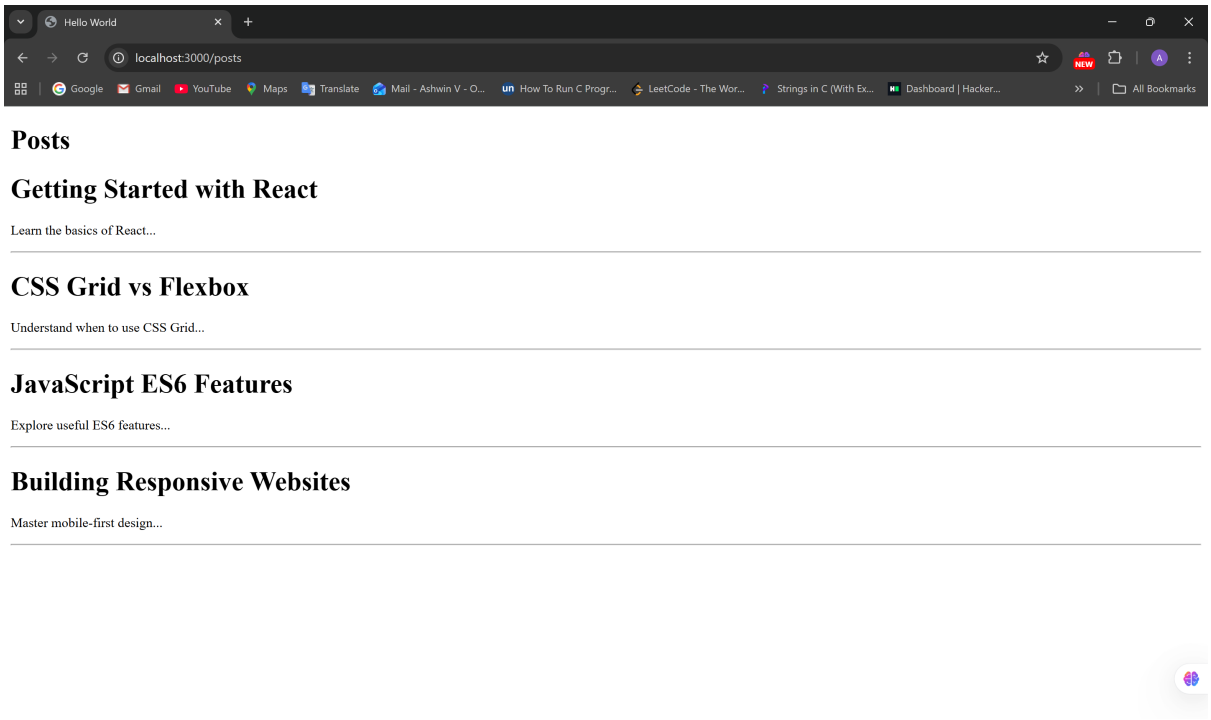
Main Files and Their Roles

- **server.js** - Entry point of the Node application. Creates an HTTP server, handles routing logic, and serves content.
- **components/homePage.html** - Static home page served at the root endpoint (`/`).
- **components/blog.html** - Template HTML structure used to render blog posts dynamically.
- **components/error.html** - HTML error page returned for invalid routes.
- **data/BlogPosts.js** - Exports a hardcoded array of blog post objects, each containing `title` and `description`.
- **routes/home.js** - Handles logic for rendering and returning the home page.
- **routes/posts.js** - Handles logic for looping through the post data and generating the blog page.
- **routes/error.js** - Exports a function to handle 404 errors gracefully.

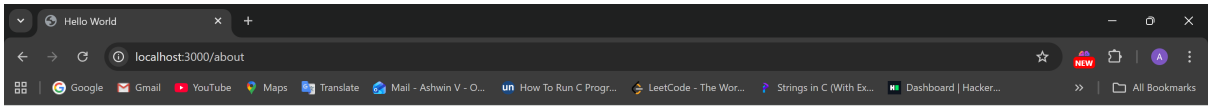
Rendered Output (Screenshots)



Screenshot 1: Home Page



Screenshot 2: Posts Page with Dynamic Blog Posts



Stop Being Nosy

Screenshot 3: Custom 404 Error Page - Invalid Route

Code Overview

server.js

```
const http = require('http');
const homeRoute = require('./routes/home');
const postsRoute = require('./routes/posts');
const errorRoute = require('./routes/error');

const server = http.createServer((req, res) => {
  const { method, url } = req;

  if (method === 'GET' && url === '/') return homeRoute(req, res)
     ;
  if (method === 'GET' && url === '/posts') return postsRoute(req
     , res);

  return errorRoute(req, res);
});

server.listen(3000, () => {
  console.log('Server running on http://localhost:3000');
});
```

routes/home.js

```
const fs = require('fs');
```

```
function homeRoute(req, res) {
  fs.readFile('components/homePage.html', 'utf-8', (err, data) =>
      {
      if (err) {
          res.writeHead(500, { 'Content-Type': 'text/plain' });
          res.end('Internal Server Error');
      }
      else {
          res.writeHead(200, { 'Content-Type': 'text/html' });
          res.end(data);
      }
  });
}

module.exports = homeRoute;
```

routes/posts.js

```
const fs = require('fs');
const blogPosts = require('../data/blogPosts');

function postsRoute(req, res) {
  let str = "";
  blogPosts.map((post) => {
    str += `<h1>${post.title}</h1><p>${post.description}</p><hr
       />`;
  });

  fs.readFile('components/blog.html', 'utf-8', (err, data) => {
      if (err) {
          res.writeHead(500, { 'Content-Type': 'text/plain' });
          res.end('Internal Server Error');
      }
      else {
          const html = data.replace('{posts}', str);
          res.writeHead(200, { 'Content-Type': 'text/html' });
          res.end(html);
      }
  });
}

module.exports = postsRoute;
```

routes/error.js

```
const fs = require('fs');

function errorRoute(req, res) {
  fs.readFile('components/error.html', 'utf-8', (err, data) => {
```

```
    res.writeHead(404, { 'Content-Type': 'text/html' });
    res.end(data || '<h1>404 Not Found</h1>');
  });
}

module.exports = errorRoute;
```

data/BlogPosts.js

```
module.exports = [
  { id: 1, title: "Getting Started with React", description: "
     Learn the basics of React..." },
  { id: 2, title: "CSS Grid vs Flexbox", description: "Understand
      when to use CSS Grid..." },
  { id: 3, title: "JavaScript ES6 Features", description: "
     Explore useful ES6 features..." },
  { id: 4, title: "Building Responsive Websites", description: "
     Master mobile-first design..." },
];
```

components/homePage.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <nav>
      <a href="/">Home</a> |
      <a href="/posts">Posts</a> |
      <a href="/about">About</a>
    </nav>
    <hr />
    <h1>Home Page</h1>
    <p>Welcome to the home page of our Node.js server!</p>
  </body>
</html>
```

components/blog.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hello World</title>
    </head>
    <body>
        <h1>Posts</h1>
```

```
        {posts}

    </body>
</html>
```

components/error.html

```
<!DOCTYPE html>
            <html>
            <head> <title> Hello World </title> </head>
            <body> <h1> Stop Being Nosy </h1> </body>
            </html>
```

Result

The Node.js HTTP server was successfully implemented with basic routing and dynamic content generation. Navigating to `http://localhost:3000/` renders the home page, while `/posts` dynamically displays 3-4 blog posts from the array. Invalid routes correctly return a custom 404 page.

Conclusion

This lab demonstrates how to create a functional web server using only Node.js' built-in `http` and `fs` modules. It showcases:

- Manual routing without frameworks.
- Dynamic HTML generation using JavaScript loops.
- Modular separation of route handlers and data sources.

The project lays the foundation for understanding core Node.js web concepts before introducing Express.js in advanced labs.

Lab Exercise 10: Node.js HTTP Server with URL Routing and Logging

Question

Create an HTTP server using Node.js. Use the `url` module to parse the incoming request URL and extract the pathname for routing decisions.

- For pathname `/`, read and serve the content of `index.html`. If the file doesn't exist, log an error in a log file but still return a basic "Home Page" message.
- For pathname `/about`, read and serve the content of `about.html` similarly, logging an error if the file is missing.
- For any other pathname, return a 404 status with "Page Not Found".
- Create two simple HTML files — `index.html` and `about.html` — with a title and a short paragraph.
- Implement logging of server events and errors into a text file named `Logs.txt`.
- Test the server locally.

Design Description

This lab focuses on building a simple Node.js server without frameworks, utilizing the `http`, `fs`, and `url` modules to serve HTML pages based on URL routes. The design demonstrates:

- Manual routing logic for multiple paths.
- Error handling when files are missing.
- Logging of both successful and failed requests.

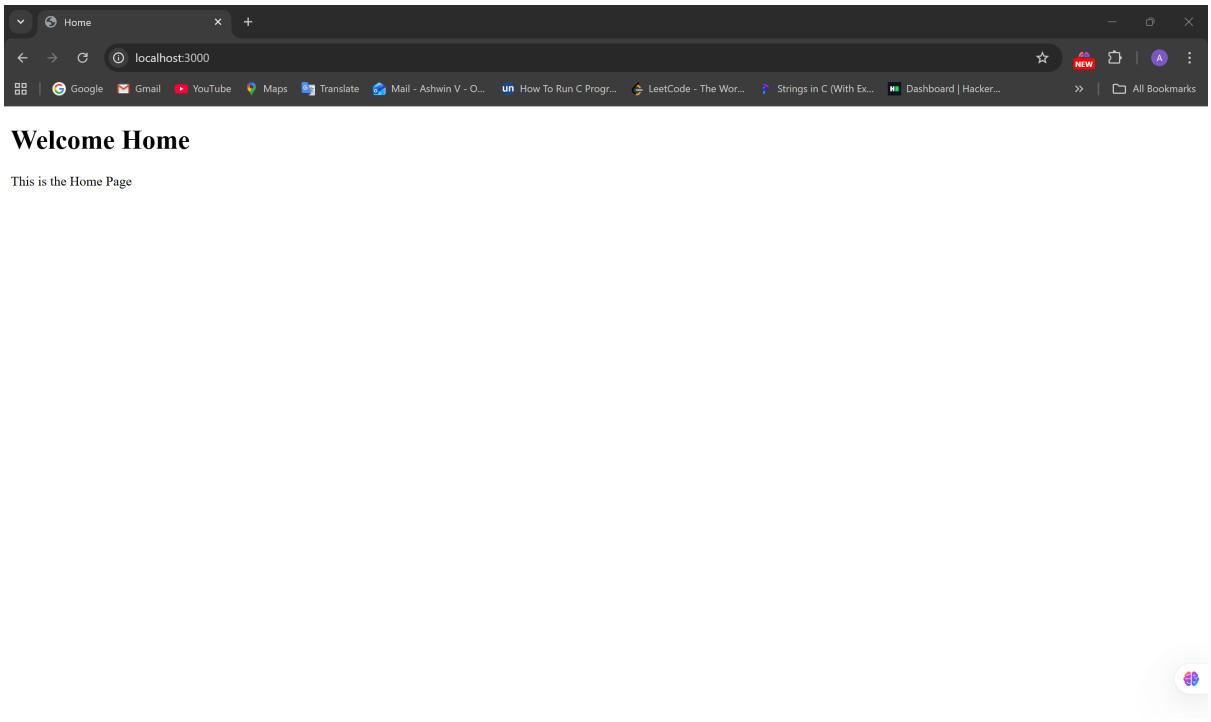
The program reads static HTML files from the local `components/` directory and sends them as responses, while also maintaining logs in a text file for traceability.

Main Files and Their Responsibilities

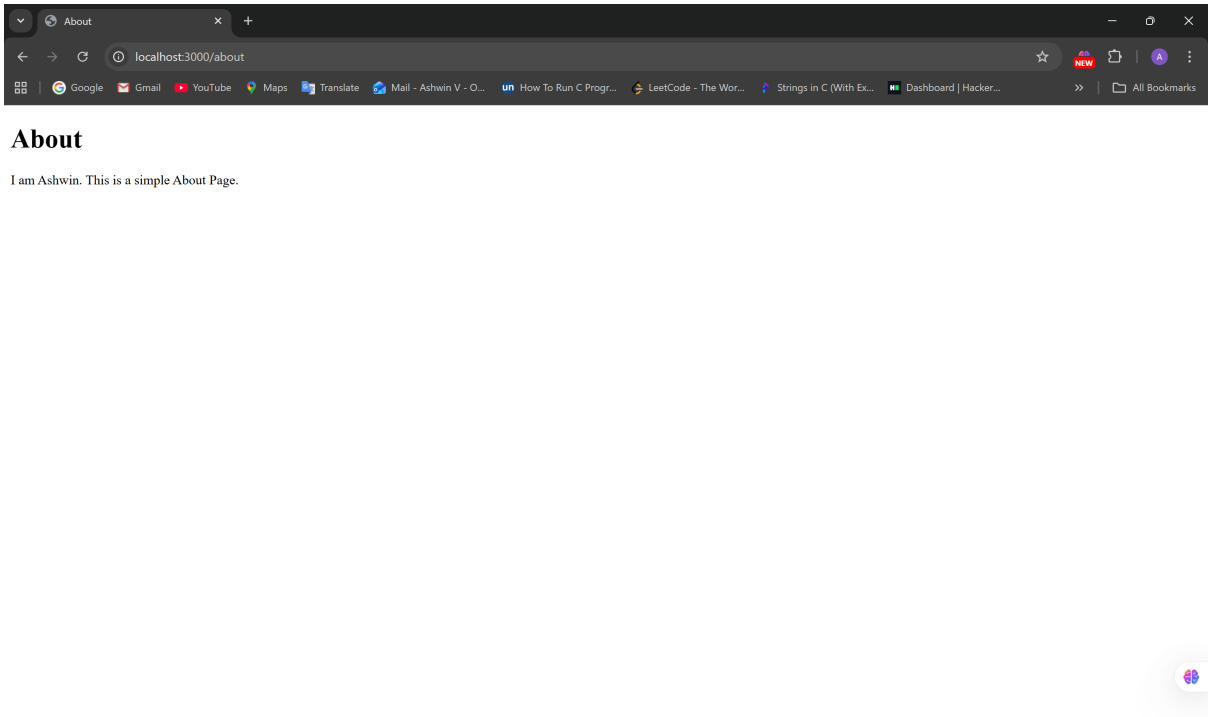
- **server.js** - Entry point that creates the HTTP server, parses URL paths, routes requests, and delegates to route handlers.
- **routes/home.js** - Handles requests to the root path (`/`); serves `home.html`.
- **routes/about.js** - Handles requests to the `/about` path; serves `about.html`.
- **components/home.html** - HTML file for the home page, containing a title and a paragraph.
- **components/about.html** - HTML file for the about page, containing a title and a paragraph.

- **utils/logger.js** - Contains helper functions to log request details and errors to `Logs.txt`.
- **Logs.txt** - Log file that records request events, timestamps, and error messages.

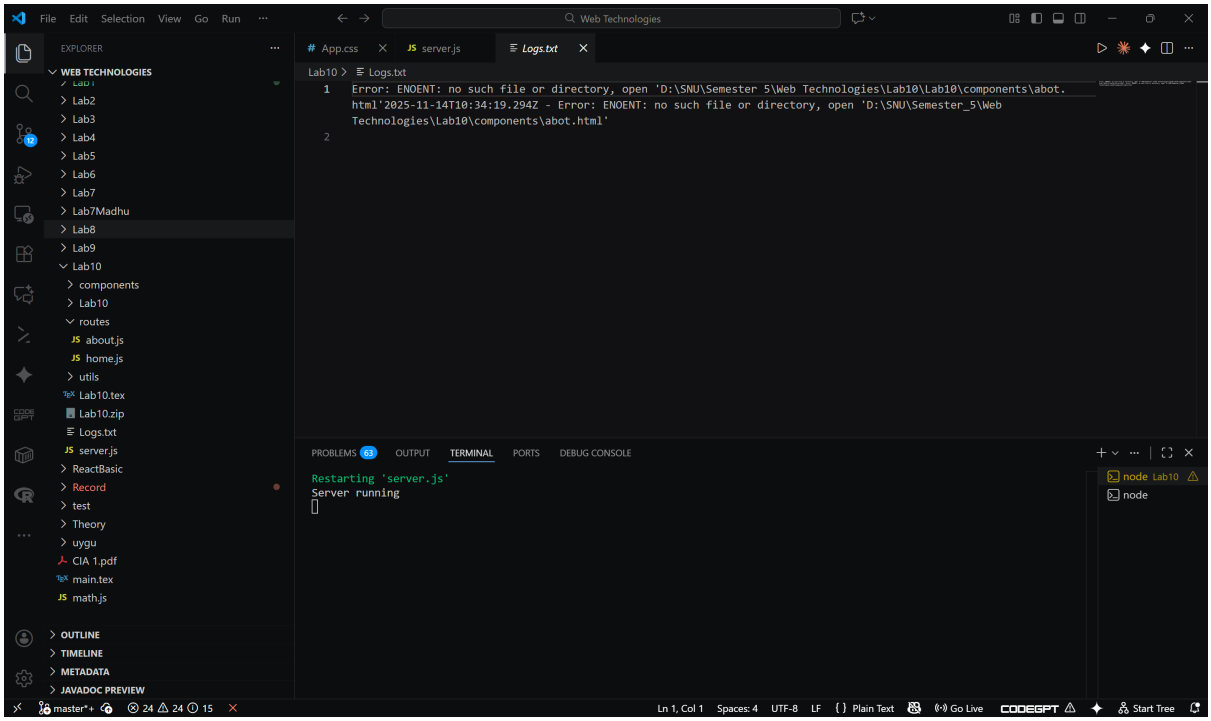
Rendered Output (Screenshots)



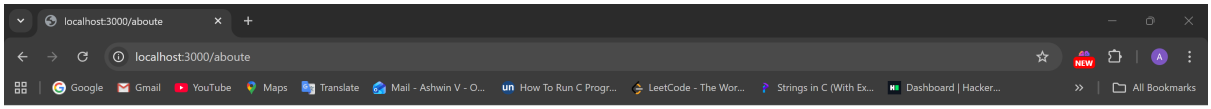
Screenshot 1: Home Page served successfully.



Screenshot 2: About Page Rendered Successfully.



Screenshot 3: Custom 404 Page Not Found response for invalid routes.



Page Not Found

Screenshot 4: Internal Server Error logged when abot.html is missing.

Code Overview

server.js

```
const http = require('http')
const urlLib = require('url')
const homePage = require('./routes/home')
const aboutPage = require('./routes/about')

const server = http.createServer((req, res) => {
    const headers = urlLib.parse(req.url);
    const url = headers.path;
    const method = req.method;

    if(url === "/" && method === "GET") {
        homePage(req, res);
    }

    else if (url === "/about" && method === "GET") {
        aboutPage(req, res);
    }

    else {
        res.writeHead(404, {"Content-Type":"text/html"});
        res.end("<h1> Page Not Found </h1>");
    }

});
```

```
server.listen(3000, () => {
    console.log("Server running");
});
```

routes/home.js

```
const fs = require('fs');
const logError = require('../utils/logger');
function homePage (req, res) {
const htmlContent = fs.readFile('components/home.html', 'utf-8',
   (err, data) => {
    if (err) {
        res.writeHead(500,{"Content-Type":"text/html"});
        res.end("<h1>INTERNAL SERVER ERROR</h1>")
        logError(err);
    }

    else {
        res.writeHead(200, {"Content-Type":"text/html"});
        res.end(data);
    }
})}
module.exports=homePage;
```

routes/about.js

```
const fs = require('fs');
const logError = require('../utils/logger');
function aboutPage (req, res) {
const htmlContent = fs.readFile('components/abot.html', 'utf-8',
   (err, data) => {
    if (err) {
        res.writeHead(500,{"Content-Type":"text/html"});
        res.end("<h1>INTERNAL SERVER ERROR</h1>")
        logError(err);
    }

    else {
        res.writeHead(200, {"Content-Type":"text/html"});
        res.end(data);
    }
})}
module.exports=aboutPage;
```

utils/logger.js

```
const fs = require('fs');

function logError(err) {
  const logMsg = `${new Date().toISOString()} - ${err}\n`;
  fs.appendFile('Logs.txt', logMsg, (writeErr) => {
    if (writeErr) console.error("Failed to write log:", writeErr)
        ;
  });
}

module.exports = logError;
```

HTML Files

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
       scale=1.0">
    <title>Home</title>
</head>
<body>
    <h1>Welcome Home</h1>
    <p>This is the Home Page</p>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-
       scale=1.0" />
    <title>About</title>
  </head>
  <body>
    <h1>About</h1>
    <p>I am Ashwin. This is a simple About Page.</p>
  </body>
</html>
```

Logs.txt (Sample Output)

```
Error: ENOENT: no such file or directory, open 'D:\SNU\Semester
   5\Web Technologies\Lab10\Lab10\components\abot.html'
```

Result

The HTTP server successfully handles multiple routes:

- Accessing `http://localhost:3000/` serves the home page.
- Accessing `http://localhost:3000/about` serves the about page.
- Accessing any other URL returns a custom 404 response.

If any HTML file is missing, the server logs an appropriate error in `Logs.txt` while still responding with a fallback message to ensure reliability.

Conclusion

This lab effectively demonstrates:

- Routing control in Node.js using the `url` module.
- File reading and serving via the `fs` module.
- Error handling and event logging through a custom utility.

It provides a foundational understanding of low-level HTTP server mechanics before transitioning to frameworks like Express.js.