A REPORT OF FOUR WEEK TRAINING

at

HB INFOTECH SOLUTIONS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(Computer Science and Engineering)

JUNE-JULY, 2025

SUBMITTED BY:

JIYA GARG

2104131

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GURU NANAK DEV ENGINEERING COLLEGE LUDHIANA

(An Autonomous College Under UGC ACT)

CERTIFICATE BY COMPANY

CANDIDATE'S DECLARATION

I **JIYA GARG** hereby declare that I have undertaken four week training at **HB Infotech Solutions** during a period from 23 June,2025 to 23 July,2025 in partial fulfilment of requirements for the award of degree of B.Tech (Computer Science and Engineering) at GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA. The work which is being presented in the training report submitted to Department of Computer Science and Engineering at GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA is an authentic record of training work.

Signature of the Student

The four week industrial training Viva–Voce Examination of_____ has been held on _____ and accepted.

Signature of Internal Examiner Signature of External Examiner

ABSTRACT

The MERN Stack—consisting of MongoDB, Express.js, React.js, and Node.js—is a powerful full-stack development framework widely used for building dynamic and scalable web applications. This training program offered a detailed exploration of how these technologies work together to create modern, high-performance systems. It emphasized the importance of using JavaScript across all layers of development, making the MERN Stack both efficient and developer-friendly.

The curriculum covered essential concepts such as building interactive user interfaces with React, developing backend logic and APIs using Node.js and Express.js, and managing data efficiently through MongoDB. Hands-on sessions allowed me to practice component-based development, state management, routing, and asynchronous communication between client and server. Real-time exercises reinforced the theoretical understanding of RESTful API design and database integration.

Overall, the training strengthened my full-stack development skills by providing practical experience with the MERN architecture. The program enhanced not only my technical capabilities but also my analytical and problem-solving skills, equipping me to confidently design, develop, and deploy complete web applications. The comprehensive approach ensured that I gained a solid foundation for future learning and professional growth in web development.

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to all those who contributed to the success of this training program. First and foremost, I extend my sincere thanks to the trainer Mr Manpreet Singh and mentors of HB Infotech Solutions for their invaluable guidance, expertise, and support throughout the course. Their dedication and passion for web development inspired and motivated me to push my boundaries. The discussions, shared experiences, and teamwork greatly enriched the learning process. Their commitment to fostering professional development is truly commendable.

Furthermore, I extend my heartfelt thanks to the faculty members of Guru Nanak Dev Engineering College, whose continuous encouragement and support helped me undertake this training with confidence. Their mentorship has been pivotal in my academic growth, and I am grateful for the foundation they provided, which allowed me to fully benefit from this practical experience. This training has been an invaluable step in my journey, and I deeply appreciate all the individuals who contributed to its success

JIYA GARG

ABOUT THE INSTITUTE

HB Infotech Solutions is a leading software services company known for its commitment to delivering world-class integrated business solutions tailored for the IT industry. They specialize in IT consulting services and web application development, consistently striving for top performance through the use of cutting-edge technology, high-quality research, and rigorous quality processes.

Their marketing department is comprised of experts focused on accelerating business growth and innovation. HB Infotech Solutions offers a wide range of specialized software services, ensuring exceptional support and comprehensive access to the best market opportunities.

Their main areas of focus include Quality Consulting, Software Testing, and Outsourcing, all underscored by a commitment to quality and client satisfaction. HB Infotech Solutions provides high quality Web Application Services that all aspects of your software application life-cycle needs. Our support team is efficient enough to handle and deliver the clients run time requirement at any cost. Customized Software Solutions HB Infotech Solutions is dedicated to develop flexible, custom-designed software meeting all the requirements of the customers has helped us earn an unmatched reputation among the customers. The company prioritizes excellence, making it a point to deliver unparalleled quality in every project.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

CHAPTER-1 INTRODUCTION

1.1 Background of the training

In recent years, the digital world has undergone an extraordinary transformation. The rapid growth of online platforms, cloud solutions, automation tools, e-commerce systems, and interactive applications has dramatically changed how users interact with technology. Modern users expect web applications to be fast, intuitive, visually appealing, and capable of processing data in real time. This increasing demand has pushed developers to adopt advanced technologies, efficient workflows, and full-stack development practices that combine both frontend and backend skills.

Full-stack development refers to the ability to build both the client-side (frontend) and server-side (backend) parts of an application. This ensures that developers understand the complete lifecycle of a web application—from designing the user interface to managing databases and server logic. Among the various full-stack technologies, the MERN Stack (MongoDB, Express.js, React.js, and Node.js) has gained prominence due to its simplicity, flexibility, and use of a single language—JavaScript—across the entire application layer.

One of the biggest advantages of the MERN stack is that it eliminates the need to learn multiple programming languages for different layers. Instead, JavaScript is used for building the frontend (React), backend logic (Node.js + Express.js), and even data modeling (MongoDB stores JSON-like documents). This unified approach significantly reduces development complexity and accelerates the overall workflow.

During the training program, I gained practical experience in building end-to-end applications, starting from creating browser interfaces using HTML, CSS, and JavaScript, to developing structured and reusable UI components with React.js. After mastering the frontend, the next stage

involved learning how to build backend servers with Node.js and Express.js—responsible for handling API routing, user requests, authentication, and business logic. Lastly, MongoDB was introduced as the primary database solution to store, modify, and retrieve data efficiently.

This training not only strengthened my technical skills but also deepened my conceptual understanding of essential topics such as state management, component lifecycle, asynchronous programming, API integration, database modeling, server communication, and real-world project deployment. Through hands-on practice with industry-level tools and frameworks, I gained valuable insight into modern software development practices and how professional web applications are built, optimized, and maintained.

1.2 Theoretical Explanation of Technologies

1.2.1 HTML (Hyper Text Markup Language)

HTML (Hyper Text Markup Language) is the standard markup language used to create and structure content on the World Wide Web. It defines the layout and structure of web pages by using a collection of elements known as tags. HTML forms the foundation of all websites, enabling the display of text, images, links, forms, tables, and multimedia.

Characteristics of HTML

- **Markup Language**

  HTML uses markup tags enclosed in angle brackets < >.

  Most tags come in pairs:

  <tagname> content </tagname>

- **Hierarchical Structure**

  HTML document structure follows a fixed format:

```
<!DOCTYPE html>

<html>

  <head> Contains metadata (title, CSS links, scripts) </head>

<body>

<h1>This is a Heading</h1>

<p>This is a paragraph.</p>

</body>

</html>
```

- **Elements in HTML**

  Commonly used elements include:

Table 1.1 Commonly used HTML tags

Tag	**Description**
<p>	Paragraph
<h1>–<h6>	Headings of different levels
<a>	Hyperlink
<img>	Image
, ,	Lists
	Generic container
<form>	User input form
<input>, <textarea>, <button>	Input controls

- **Attributes**

Attributes provide additional information to HTML elements.

Example:

```
<a href="https://example.com">Click Here</a>
```

- **Semantic HTML**

Modern HTML encourages using meaningful elements like:

- <header> <footer> <nav> <section> <article> which improves readability, accessibility, and SEO.

Important HTML Tags Overview

Table 1.2 HTML Tags and their purpose

Tag	**Purpose**
<html>	Root element of document
<head>	Metadata and linked resources
<title>	Page title
<link>	External stylesheet linking
<script>	JavaScript embedding
<style>	Internal CSS styling
<body>	Visible page content
<h1>–<h6>	Headings
<p>	Paragraph
<a>	Hyperlink
<img>	Image
<table>, <tr>, <td>, <th>	Table formatting
<form>, <input>, <button>, <label>, <select>	Form and input controls
<audio>, <video>	Multimedia embedding
<hr>	Horizontal divider

1.2.2 CSS (Cascading Style Sheets)

CSS is a **stylesheet language** used to control the visual styling, layout, and formatting of HTML elements.

It ensures that the structure (HTML) and presentation (CSS) are kept separate.

Purpose of CSS

- **Styling:** Colors, fonts, backgrounds, line spacing.
- **Layout:** Arranging elements using Grid, Flexbox, etc.
- **Responsive Design:** Ensures compatibility across mobile, tablet, and desktop.

How CSS Works

A CSS rule consists of a **selector** and **declaration block**:

```
h1 {
  color: blue;
}
```

Ways to Apply CSS

Table 1.3 Ways to apply CSS

Method	**Description**
Inline CSS	Style written directly in HTML element
Internal CSS	Style written inside <style> tag in <head>
External CSS	Styles placed in .css file and linked using <link>

CSS Selectors

Table 1.4 Types of CSS Selectors

Selector Type	Example	Description
ID Selector	#header	Selects single unique element
Class Selector	.title	Selects elements with same class
Descendant Selector	ul li	Selects nested elements
Child Selector	ul > li	Selects direct children

Common Uses of CSS

- Creating layouts using Grid and Flexbox
- Styling text and typography
- Creating animations and transitions
- Enhancing accessibility
- Ensuring consistent design across devices and browsers

1.2.3 Bootstrap

Bootstrap is a popular open-source CSS framework used for designing responsive, consistent, and mobile-first web interfaces.

Key Features

Table 1.5 Features of Bootstrap

Feature	Description
Grid System	12-column responsive layout system

Pre-built Components	Buttons, Navbars, Forms, Cards, Carousels
Utility Classes	Margin, padding, color, size helpers
JavaScript Plugins	Dropdowns, modals, popovers, sliders
Highly Customizable	Can override and theme styles easily

Advantages of Bootstrap

- Speeds up development
- Ensures cross-browser compatibility
- Mobile-first responsive design
- Large community support

Use Cases

- Corporate websites
- E-commerce platforms
- Admin dashboards
- Blogs and CMS systems
- Prototyping UI quickly

1.2.4 JavaScript

JavaScript is a high-level, dynamic, and event-driven programming language used to add interactivity and dynamic behavior to web pages.

Key Features

- **Client-Side Scripting:** Runs directly in browser.

- **Dynamic Typing:** Variables can hold any data type.
- **Event Driven:** Responds to user actions (clicks, keyboard input).
- **Object-Oriented:** Supports objects and classes.

Common Uses

Table 1.6 Common uses of JavaScript

Use	Description
DOM Manipulation	Update webpage content dynamically
Form Validation	Validate user input before submission
Animations & Effects	Create dynamic UI interactions
Web Applications	Used for SPAs (React, Vue, Angular)
Server-Side Apps	Through Node.js for backend execution

JavaScript Libraries & Frameworks

Table 1.7 JavaScript Libraries and Frameworks

Type	Example	Purpose
Library	jQuery	DOM Interaction
Frontend Framework	React, Angular, Vue	UI Development
Backend Runtime	Node.js	Server & API Development

1.2.5 MERN Stack

The MERN Stack is a popular full-stack development framework used for building dynamic, data-driven web applications. It combines four powerful technologies: MongoDB, Express.js, React.js, and Node.js. The core idea behind MERN is to use JavaScript as the primary

programming language for both frontend and backend, thereby simplifying development and ensuring seamless integration across all layers of the application.

MERN follows a three-tier architecture:

Table 1.8 Three-tier architecture of MERN

Layer	**Technology**	**Purpose**
Frontend (Client Side)	**React.js**	Builds user interface and handles user interactions
Application Layer (Server Side)	**Express.js** & **Node.js**	Handles logic, routing, APIs, and communication
Database Layer	**MongoDB**	Stores and retrieves data in flexible document format

The use of JavaScript across all layers enhances consistency, development speed, and maintainability.

1. MongoDB (Database Layer)

MongoDB is an open-source, NoSQL (Non-Relational) database used for storing large-scale, flexible, and unstructured data. Unlike traditional relational databases that store data in rows and tables, MongoDB uses BSON (Binary JSON) formatted documents, which are stored inside collections. This structure closely resembles JavaScript objects, making MongoDB highly compatible with JavaScript-based technologies such as Node.js and Express.js, forming a natural fit within the MERN Stack.

Document-Oriented Structure

MongoDB stores data in documents rather than tables. A document is similar to a JSON object containing key-value pairs, arrays, or even nested documents. This structure allows developers

to store complex and dynamic data without defining a fixed schema in advance.

Example of a MongoDB Document:

```
{
 "name": "John Doe",
 "email": "john@example.com",
 "age": 22,
 "skills": ["React", "Node.js", "MongoDB"]
}
```

This makes MongoDB flexible, scalable, and easy to modify during development.

Schema Flexibility

Traditional databases require data fields to be defined before inserting data, but MongoDB allows storing data without predefining columns. This schema-less nature enables rapid development and easier updates.

High Performance and Scalability

MongoDB supports horizontal scaling using sharding, where data is distributed across multiple servers.

It also provides indexing and aggregation pipelines, which improve query efficiency and support large-scale applications.

Key Performance Features:

- Indexes for fast searching
- Replication for data backup

- Sharding for distributed data storage
- High read/write performance

Integration with Node.js and Mongoose

In MERN, MongoDB is usually connected using Mongoose, an Object Data Modeling (ODM) library.

Mongoose provides data validation, schema definition, and easy querying methods, making interaction with MongoDB simpler and structured.

Example Mongoose Schema:

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

const studentSchema = new mongoose.Schema({

  name: String,

  email: String,

  department: String

});

module.exports = mongoose.model("Student", studentSchema);
```

This helps manage MongoDB collections in an organized way while maintaining flexibility.

Advantages of MongoDB in MERN

- Uses JSON-like data → easier communication with Node.js / Express
- Flexible schema → faster development and modifications
- No complex joins → simpler queries
- High scalability → suitable for modern web applications

- Supports real-time applications

2. Express.js – The Backend Web Framework

Express.js is a lightweight, fast, and flexible backend web framework built on top of Node.js. It is used to create server-side applications and manage communication between the frontend (React) and the database (MongoDB). Express simplifies tasks such as handling HTTP requests, defining routes, managing middleware, and generating API endpoints.

Purpose of Express.js

Express acts as the **middle layer** in the MERN architecture. It allows the application to:

- Receive requests from the frontend
- Process the request or interact with the database
- Send back the appropriate response

This ensures smooth and structured communication across the system.

Routing in Express.js

Routing refers to defining how a server responds to various URL paths and HTTP methods like:

- **GET** (retrieve data)
- **POST** (create data)
- **PUT** (update data)
- **DELETE** (remove data)

Example Express Route:

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

const app = express();
```

```
app.use(express.json());

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

  res.send("Fetching student data");

});

app.listen(5000, () => {

  console.log("Server running on port 5000");

});
```

This route responds to a GET request at /students.

Middleware

Middleware functions in Express allow you to handle tasks such as:

- Logging user activities
- Validating user inputs
- Authentication
- Error handling

Middleware executes before the final request is processed.

app.use(express.json());

RESTful API Support

Express is commonly used to create **RESTful APIs**, which makes data exchange between frontend and backend standardized and efficient.

In MERN, React calls these APIs to interact with the server and database.

Advantages of Express.js

- Simple and easy to implement
- Highly customizable for small to large applications
- Large ecosystem of plugins and middleware
- Works excellently with Node.js and MongoDB

3. React.js – The Frontend User Interface Layer

React.js is a powerful and widely-used JavaScript library developed by Facebook for building dynamic, interactive, and responsive user interfaces.

It focuses on the view layer of web applications and allows developers to create reusable UI components.

Setting up a React Environment

1. First, make sure you have Node.js installed. You can check by running this in your terminal: node -v
2. If Node.js is installed, you will get a result with the version number: v22.15.0
3. Install Dependencies- As the result above suggests, navigate to your new react application directory: cd my-react-app
4. And run this command to install dependencies: npm install

Understanding the React Project Structure

When the tool creates the project, it generates folders:

node_modules/

- Contains all installed libraries
- Automatically created, never edited manually

public/

- Contains static assets
- The main file inside is **index.html**

src

- Contains all React components
- Main files:
    - **index.js / main.jsx** → entry point
    - **App.js / App.jsx** → root component

package.json

- List of dependencies (React, ReactDOM)
- Scripts for running, building project
- Project metadata

package-lock.json

- Locks the dependency versions

Component-Based Architecture

React follows a **component-based architecture**, which is one of its most powerful and distinguishing features. Instead of building the entire UI as one large structure, React breaks down the interface into small, modular, self-contained components. Each component represents a specific piece of the user interface—for example, a button, navigation bar, form input, or card.

Components in React can be **functional** or **class-based**, but modern React primarily encourages functional components with hooks due to their simplicity and performance benefits.

Example of a Simple React Component

```
function Welcome() {
  return <h1>Hello User!</h1>;
}
export default Welcome;
```

Here:

- The component returns JSX.
- It can be imported and reused anywhere.
- It defines how a specific portion of UI should appear and function.

React applications resemble a tree structure, where the root component (App.js) contains child components such as header, footer, sidebar, forms, etc. This modular architecture leads to clean, scalable, and production-ready applications.

JSX (JavaScript XML)

JSX is a **syntax extension** for JavaScript that allows developers to write HTML-like code directly inside JS files. It improves readability and makes UI development more intuitive.

Why JSX is Powerful

- It combines HTML structure and JavaScript logic in one place.
- It automatically compiles into React.createElement() calls.
- It supports dynamic expressions using {}.
- It improves developer productivity and reduces code complexity.

Example of JSX

```
const element = <h2>Welcome to React</h2>;

const title = <h2>Welcome to React</h2>;
```

Advanced JSX Capabilities

- Embedding JavaScript expressions
- Conditional rendering
- Looping through arrays to render lists
- Inline styling and class usage
- Using JSX fragments (<> </>)

Because JSX is human-readable and extremely expressive, it has become the standard way of writing UI in React.

State and Props

React applications often deal with **dynamic data**, which changes based on user actions or external events. React manages this dynamic data using **state** and **props**, two core concepts essential for understanding component behavior.

State

State refers to data that is **internal to a component**, changeable over time, and triggers re-rendering when updated. It is used to manage interactive features like form inputs, counters, toggles, or API responses.

Example of State

```
import { useState } from "react";

function Counter() {
```

```
  const [count, setCount] = useState(0);

  return (

    <>

      <p>Count: {count}</p>

      <button onClick={() => setCount(count + 1)}>Increase</button>

    </>

  );

}
```

Characteristics of State

- Mutable (can be updated)
- Used for dynamic content
- Causes re-render on change
- Local to the component unless lifted up

Props

Props (short for "properties") are **read-only** values passed from a parent component to a child. They allow data flow throughout the component hierarchy.

Props Example

Main.jsx

```
        import { createRoot } from 'react-dom/client';

        function Car(props) {
```

```
  return (
    <h2>I am a {props.brand}!</h2>
  );
}
createRoot(document.getElementById('root')).render(
  <Car brand="Ford" />
);
```

Index.html

```
<!doctype html>
<html lang="en">
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Key Differences: State vs. Props

Table 1.9 Difference between State and Props

Factor	**State**	**Props**
Mutability	Mutable	Immutable
Ownership	Local to component	Passed from parent

Purpose	Store dynamic data	Pass data between components
Re-render	Yes	Yes (if props change)

Together, state and props help React create dynamic, interactive, predictable user interfaces.

Virtual DOM

The Virtual DOM (VDOM) is one of the key technologies that makes React extremely fast. Instead of updating the entire web page whenever data changes, React updates an in-memory representation of the DOM first and then efficiently synchronizes only the changed parts with the actual DOM.

How It Works

- UI changes → React creates a new virtual DOM tree.
- React compares it with the previous virtual DOM (diffing algorithm).
- Only the changed nodes are updated in the real DOM.

Advantages of Virtual DOM

- **Performance Optimization:** Eliminates unnecessary DOM updates.
- **Improved Rendering Speed:** Only minimal UI parts are updated.
- **Smooth User Experience:** Reduces page flickering or slow UI.
- **Efficient Reconciliation:** React's diffing algorithm ensures fast UI updates.

Why Not Update DOM Directly?

The browser DOM is slow. Every DOM manipulation affects layout, repaint, and reflow cycle—time-consuming processes. VDOM avoids this.

Benefits of React.js

React brings several advantages that have made it one of the most widely adopted frontend technologies:

- **Reusable Components:** Reduces code repetition and makes applications scalable.
- **Strong Ecosystem:** Huge community, third-party libraries, and active development.
- **SEO-Friendly Rendering:** Using frameworks like Next.js, React supports server-side rendering.
- **Easy Backend Integration:** APIs built with Express.js, Node.js, or any backend can be easily consumed in React using Axios, Fetch API and React Query.
- **Clear Code Structure:** JSX + component tree makes code more maintainable.

4. Node.js – The Backend Runtime Environment

Node.js is an open-source, cross-platform, server-side JavaScript runtime environment. It allows developers to run JavaScript outside the web browser, typically on the server.
In the MERN Stack, Node.js is used to build the backend logic and handle the application's server-side operations.

Purpose of Node.js

Traditionally, JavaScript was only used in browsers. Node.js enables developers to:

- Use JavaScript on the server
- Build fast and scalable network applications
- Perform backend tasks such as database interactions, server routing, and API creation

This allows MERN developers to work with one language (JavaScript) on both frontend and backend, making development simpler and more efficient.

Event-Driven & Non-Blocking Architecture

Node.js uses non-blocking I/O and an event-driven model, meaning:

- It does not wait for operations like database queries to finish before moving to the next task.
- It handles multiple requests simultaneously.

This makes Node.js ideal for:

- Real-time applications (e.g., chat apps, notifications)
- High-traffic systems
- Streaming services

npm (Node Package Manager)

Node.js comes with **npm**, the largest ecosystem of libraries and modules in the world.

Developers can install ready-made packages for:

- Authentication
- Database connectivity
- Form validation
- Encryption
- File handling
- API handling

Example:

npm install express

This saves time and reduces the need to write code from scratch.

Asynchronous Patterns: Callbacks, Promises, async/await

- **Callbacks:** Original async style; can lead to "callback hell" if deeply nested.
- **Promises:** Represent eventual values; allow chaining using .then() and .catch().
- **async/await:** Syntactic sugar over promises; enables writing asynchronous code that looks synchronous and is easier to reason about.

Example (async/await):

```
import fs from 'fs/promises';

async function readConfig(path) {
  try {
    const data = await fs.readFile(path, 'utf8');
    return JSON.parse(data);
  } catch (err) {
    throw new Error('Unable to read config: ' + err.message);
  }
}
```

Usage in MERN Stack

Node.js works as the foundation for Express.js:

- **React** sends requests → **Express** handles routes → **Node.js** executes the server logic → **MongoDB** stores or retrieves data.

So Node.js acts as the **engine** powering the backend operation flow.

Advantages of Node.js

Table 1.10 Advantages of Node.js

Feature	**Benefit**
Same language (JavaScript)	Easy full-stack development
Fast performance	Thanks to V8 engine
Handles many users at once	Non-blocking event-driven model
Huge library support	npm ecosystem
Real-time support	Perfect for live chat, tracking, streaming

1.3 Software Tools Learned

Table 1.11 Software Tools Learned

Software / Technology	**Purpose**	**Description**
HTML5	Structure Design	Used to create the basic layout and content of web pages.
CSS3 & Bootstrap	Styling & Layout	Used to enhance visual appearance and ensure responsive and mobile-friendly design.

JavaScript (ES6)	Interactivity	Used for dynamic behavior, events, and logical operations on the client side.
React.js	Frontend Development	Used to build interactive UI components and manage states efficiently.
Node.js	Backend Runtime	Executes JavaScript on the server and handles backend functions.
Express.js	Server Framework	Simplifies API creation, routing, and middleware handling.
MongoDB	Database	Stores and retrieves structured or unstructured data in document format.
Postman	API Testing Tool	Used for testing RESTful API endpoints during backend development.
VS Code	Code Editor	Used as the primary development environment with useful extensions.
Git & GitHub	Version Control	Used for source code tracking and team collaboration.

1.4 Hardware Requirements

Table 1.12 Hardware Requirements

Component	Minimum Requirement	Purpose
Processor	Intel Core i3 / AMD Equivalent	To run development tools smoothly
RAM	4GB (8GB Recommended)	For efficient multitasking and running servers
Storage	10GB Free Space	For installing Node.js, MongoDB, and project files
Operating System	Windows / macOS / Linux	Compatible with MERN Stack tools
Internet Connection	Required	For downloading packages and online documentation

CHAPTER-2 TRAINING WORK UNDERTAKEN

This chapter provides a detailed account of the practical training undertaken during the four-week industrial program on the MERN Stack (MongoDB, Express.js, React.js, and Node.js). It outlines the sequential learning process, methodologies adopted, and hands-on implementation carried out throughout the course. The training emphasized both theoretical reinforcement and practical application, enabling participants to gain comprehensive exposure to full-stack web development. Each week of the training was organized with clear objectives, covering fundamental concepts, environment setup, front-end and back-end development, database management, and full-stack integration. The following sections describe the week-wise activities, tasks performed, and methodologies followed during the course of the training.

2.1 Week 1 – Revision of Web Fundamentals and Core Technologies

The first week of the training was dedicated to revising the core web technologies that form the base of full-stack development. Before proceeding to advanced frameworks and backend tools, we revisited essential frontend concepts to strengthen our understanding of how web applications are structured, styled, and made responsive.

2.1.1 HTML (Hyper Text Markup Language)

We studied the **fundamentals of HTML**, focusing on the structure and layout of web pages using semantic elements such as <header>, <section>, <article>, and <footer>. Emphasis was placed on understanding how a well-organized HTML document forms the backbone of every website.

We further practiced creating **tables** to display tabular data, **lists** (both ordered and unordered) to present information systematically, and **forms** to collect user inputs through elements such as text boxes, checkboxes, and buttons. Practical exercises were carried out to strengthen our understanding of linking, navigation, and maintaining a clear content hierarchy within web pages.

2.1.2 CSS (Cascading Style Sheets)

The practice of CSS involved understanding and implementing different styling techniques, including inline, internal, and external CSS. Various properties such as height, width, colors, margins, paddings, and hover effects were applied to enhance the visual appearance and layout consistency of web pages.

Practical exercises included working with div elements for page structuring, designing web layouts using Flexbox and Grid systems, and experimenting with background images and borders to improve presentation. Responsive web design principles were also introduced and implemented using media queries to ensure that web pages adapted smoothly across different screen sizes and devices.

2.1.3 Bootstrap

We were introduced to Bootstrap as a front-end framework designed for rapid and responsive web development. It helped in building visually appealing layouts with minimal custom styling. The key components explored included the grid system for structured alignment, navigation bars for site navigation, cards and buttons for interactive elements, and modals for dynamic content display.

Practical implementation was carried out through hands-on exercises, where various pre-built components were customized to match specific design requirements. To strengthen our understanding, we practiced developing theme-based pages similar to portfolio and blog-style layouts. This helped in understanding how Bootstrap’s built-in classes and responsive utilities streamline the process of creating professional, mobile-friendly web pages efficiently.

2.1.4 JavaScript

The training progressed with JavaScript, a core language for making web pages interactive and dynamic. The focus was on strengthening programming logic through topics such as variables, data types, operators, loops, conditional statements, arrays, and objects, which enhanced problem-solving and logical reasoning skills. We further explored modern ES6 (ECMAScript 6) features such as let, const, arrow functions, and template literals to improve code readability and efficiency. A major emphasis was placed on understanding events such as mouse and keyboard interactions and how they can be used to trigger specific functions and make web pages more responsive. Additionally, hands-on practice was conducted on DOM (Document Object Model) manipulation, using methods like document.getElementById(), document.querySelector(), and innerHTML to dynamically access, modify, and update web elements, enabling real-time interactivity and functionality within web applications.

To apply these concepts practically, several small interactive programs were implemented as examples:

- **Stopwatch Counter:** Implemented start, stop, and reset functionalities to understand event handling and timer functions.
- **Form Validation Program:** Used DOM manipulation and conditional checks to validate input fields such as email format and password strength.
- **Leap Year Checker:** Applied logical reasoning and conditional statements to determine leap years based on user input.

These exercises collectively strengthened our understanding of programming logic, event handling, and DOM-based interactivity, forming a strong base for advanced front-end and full-stack development.

2.1.5 Mini Practice Activities

At the end of the first week, two static website templates were developed to apply the combined knowledge of HTML, CSS, and Bootstrap. These practice exercises helped in reinforcing the concepts of web page structuring, responsive design, and modern UI layout creation.

- **Learnex Platform Template:** Designed using HTML, CSS, and Bootstrap to practice responsive layouts, structured navigation, and utilization of pre-built components such as cards and buttons.
- **Netflix Clone Template:** Focused on building a fully responsive web page layout using Bootstrap's grid system and media queries, ensuring compatibility across different screen sizes.

These activities provided a practical understanding of how front-end technologies work together to design visually appealing and responsive web pages.

2.2 Week 2 -Environment Setup and Front-End Development

The second week of the training focused on transitioning from basic web technologies to modern front-end development using React.js. This phase involved setting up the development environment, understanding React's component-based architecture, exploring JSX, learning about hooks, and implementing API integration using Axios. The week emphasized hands-on learning through practical exercises, enabling a smooth shift from traditional static development to dynamic, component-driven application building.

2.2.1 Environment Setup for React.js

The week began with creating a proper environment for developing React applications. The setup process included the following key steps:

- **Node.js and npm Installation:** Ensuring that the latest versions of Node.js and npm (Node Package Manager) were installed to manage dependencies efficiently.

- **Creating the React App:** Using the Create React App (CRA) tool to scaffold a new project with the command:

  npx create-react-app my-react-app

  This generated an initialized project structure containing public, src, and configuration files.

- **Understanding Project Structure:** We explored the significance of folders such as:
  - **src folder –** where components, logic, and styling reside.
  - **public folder –** for static assets.
  - **package.json –** to manage dependencies and scripts.

  This setup formed the foundation for building modular and interactive front-end applications.

2.2.2 JSX and Component-Based Architecture

The training then transitioned into understanding JSX (JavaScript XML) and how React uses components to structure UI:

- JSX syntax was practiced to embed HTML-like markup directly within JavaScript.
- We studied the working of functional components and how UI is split into reusable pieces.
- Practical activities included:
  - Creating simple components using .jsx files.
  - Understanding props to pass data between components.
  - Using state to manage dynamic values inside components.

  This component-driven approach enabled the creation of clean, organized, and maintainable code.

2.2.3 React Hooks

React Hooks were introduced as a modern feature that allows functional components to handle state, lifecycle behavior, and logic reuse without using class components. Hooks simplify component development by providing built-in functions that manage data, respond to updates, and perform side effects directly inside functional components.

- **useState:** The useState hook was practiced to manage dynamic and interactive values inside components. It enabled us to store data such as counters, toggle states, and form inputs, and automatically re-render the UI whenever the state changed.
- **useEffect:** The useEffect hook was used to handle side effects like data fetching, updating values after rendering, responding to state changes, and performing cleanup tasks. It helped us understand how lifecycle behavior is managed in functional components.

Through multiple exercises, we learned how hooks make component logic cleaner and more modular. Using hooks improved our understanding of how React handles reactivity, state, and lifecycle events within modern front-end applications.

2.2.4 API Integration Using Axios

To understand how React interacts with external data sources, API integration was practiced using both Fetch API and Axios, two popular methods for making HTTP requests in JavaScript.

- **Fetch API:** The Fetch API was used to perform basic GET requests and understand how promises work in JavaScript. It helped in learning how to send requests, handle JSON responses, and manage asynchronous operations using .then() and async/await. Through practical exercises, we learned how Fetch handles responses, status codes, and error conditions, giving us a foundational understanding of API communication.
- **Axios Library:** Axios was introduced as an advanced and more convenient alternative to Fetch. The library was installed using npm install axios and then integrated into React components to retrieve data from external services. Axios simplifies request handling by

automatically converting JSON responses, offering better error handling, and providing cleaner and more readable syntax.

As part of the hands-on implementation, data was fetched from a COVID-19 public API, and the results were displayed dynamically in React components. This exercise helped reinforce concepts of asynchronous JavaScript, state management, and real-time data rendering in a React application.

2.2.5 Routing Using React Router DOM

We also studied client-side routing to enable multi-page navigation within a single-page React application.

- **React Router DOM was installed using:**

  npm install react-router-dom

- **Key functions practiced:**

  - **BrowserRouter** – for wrapping the whole application with routing capability.
  - **Routes and Route** – to map URLs to components.
  - **Link** – for navigation without page reloads.

Small multi-page structures were created demonstrating smooth page transitions using React Router.

2.2.6 Mini Practice Component Activities

Several small practice tasks were implemented to gain confidence in React fundamentals:

- Creating reusable card components.
- Handling form inputs using controlled components.

- Displaying fetched COVID-19 data in tabular and card formats.
- Navigating between pages using Router links.

These exercises reinforced the understanding of React's component workflow, state management, and routing.

2.3 Week 3 – Back-End Development and Database Management

The third week of the training introduced the server-side development part of the MERN stack using Node.js, Express.js, and MongoDB. This week focused on understanding how backend logic works, how APIs are created, how databases are connected, and how client-server communication takes place. The activities included setting up the backend environment, creating routes, writing controller functions, connecting databases, and testing all operations using Postman.

2.3.1 Environment Setup for Node.js and Express.js

Backend development began with creating a structured environment for building server-side applications.

- **Node.js Installation:** The latest version of Node.js was installed to run JavaScript on the server. Node Package Manager (npm) was used to manage backend dependencies.
- **Initializing the Project:** A new backend project directory was created and initialized using:

  npm init -y

  This generated the package.json file to manage all project dependencies.
- **Installing Express.js:** Express.js, a lightweight framework for building server-side applications, was installed using:

npm install express

- **Basic Server Setup:** The initial server file was created using Express, where a simple GET route was tested to ensure the server was running successfully.

This setup marked the starting point for writing backend routes, controllers, and API logic.

2.3.2 Introduction to Express.js and API Development

After completing the setup, training moved toward creating backend APIs using Express.js.

- The concept of routing was introduced, where different endpoints were created using GET, POST, PUT, and DELETE.
- Separate functions files (controllers) were created to write the logic for each route.
- Basic API structures were implemented to understand how data flows from client to server and back.

Examples of practice APIs included:

- Creating sample routes such as /getData
- Sending JSON responses from the server
- Accepting data from the client using req.body

Through these exercises, the foundation of backend API development was established.

2.3.3 Database Management Using MongoDB

MongoDB was introduced as the NoSQL database used within the MERN stack. Training covered both command-line operations and integration with Node.js.

Basic MongoDB Commands: Using the MongoDB Command Line Interface (CLI), several essential commands were practiced:

- **Creating a database:** use databaseName
- **Creating collections:** db.createCollection("collectionName")
- **Inserting data:** db.collectionName.insertOne({ key: "value" })
- **Updating a Document:** db.collectionName.insertMany([{...}, {...}])
- **Deleting a Document:** db.collectionName.deleteOne({ key: value })
- **Showing All Databases:** show dbs
- **Showing All Collections:** show collections

This helped in understanding how data is structured and managed in MongoDB.

2.3.4 Connecting Node.js Backend with MongoDB

Once the basics were clear, the backend was connected to MongoDB using Mongoose, an ODM (Object Data Modeling) library.

- **Mongoose Installation:**

npm install mongoose

- **Connecting to Database:** A connection file was created where Mongoose was used to connect to the local or cloud MongoDB server using a connection string.
- **Creating Schemas and Models:** Mongoose schemas were defined to structure data like users, entries, or records. These schemas were linked to models that interacted directly with the database.

This integration enabled the backend to store, retrieve, and manage data efficiently.

2.3.5 Async/Await and Handling Database Operations

During API creation, asynchronous operations were handled using async and await. This helped in writing clean and readable code for database queries.

Use cases practiced:

- Saving new entries into the database
- Retrieving lists of documents
- Updating data using IDs
- Deleting records

These operations improved our understanding of handling asynchronous tasks in Node.js.

2.3.6 Testing APIs Using Postman

All developed backend APIs were tested using Postman, a tool used for sending API requests and verifying responses.

- GET, POST, PUT, and DELETE requests were created and tested.
- Request bodies were sent using JSON in the Postman interface.
- Responses were analyzed to verify whether the backend logic and database operations were working correctly.

Testing through Postman helped validate the entire backend workflow from request to database and back to response.

2.3.7 Mini Backend Practice Activities

Several small backend tasks were performed to strengthen backend understanding:

- Creating APIs for adding and retrieving sample data
- Testing CRUD operations through Postman

- Implementing validation checks in controller functions
- Storing submitted form data into MongoDB
- Fetching stored data through GET routes

These activities ensured a strong practical understanding of server-side programming and database communication.

2.4 Week 4 – Full-Stack Integration & Project Work

The fourth week of the training focused on bringing together all the concepts learned in the previous weeks and understanding how frontend, backend, and database layers interact within a complete MERN stack application. The primary objective of this week was to understand full-stack workflows, request–response handling, API communication, and connecting React with Node.js and MongoDB.

2.4.1 Full-Stack Integration

During this phase, we practiced how different components of the MERN stack communicate with each other. Key areas covered included:

- Connecting the frontend (React.js) with the backend (Node.js + Express.js) using REST APIs.
- Sending and receiving data through HTTP methods like GET, POST, PUT, and DELETE.
- Handling JSON data and validating API requests.
- Using Axios and Fetch in React to call backend APIs.
- Testing backend routes using Postman before integrating them with the frontend.

- Establishing a connection with MongoDB using Mongoose and linking database operations with API endpoints.

These tasks helped us gain clarity on how data flows across the client, server, and database layers in a MERN application.

2.4.2 Project Work

The project work phase marked the transition from structured weekly learning to applying all acquired MERN stack concepts in a unified, practical environment. During this phase, development of the **EventSphere** application was initiated an event-management system built using React, Node.js, Express, and MongoDB.

The primary focus of this stage was to set up the complete working foundation of the full-stack application, ensuring that all layers—frontend, backend, and database—were connected and functioning cohesively.

Project Setup Overview

- **Frontend Setup (React):**
    - The React application was initialized using Vite for a faster development environment.
    - Navigation between pages was implemented using react-router-dom.
    - Initial UI screens such as Login, Registration, Home, and Dashboard were created to establish the user interface flow.
- **Backend Setup (Node.js & Express):**
    - A Node.js project was configured with Express.js as the backend framework.
    - The project structure was organized into routes, controllers, and models for maintainability.

    - Basic API endpoints were created to validate server operations and test request–response handling.

- **Database Setup (MongoDB):**
    - A MongoDB database was created for persistent storage of users, events, and related records.
    - Mongoose was used to define schemas and establish a smooth connection between the backend and the database.
    - Initial data flow tests were performed to ensure that the API could successfully create, retrieve, and manage records in the database.

This setup established the core full-stack architecture required for building the EventSphere application. The detailed functioning, modules, and features of the project are presented in Chapter 3.

2.5 Methodology Followed

The training program followed a structured and industry-oriented methodology designed to gradually build full-stack development skills across the MERN stack. The methodology emphasized clarity, incremental learning, continuous practice, full-stack integration, and validation at every stage. The following subsections describe the systematic approach adopted throughout the four-week training period.

2.5.1 Requirement Understanding and Planning

Before the technical sessions began, the overall structure and expectations of the program were reviewed. The objective was to:

- Understand the fundamental technologies involved in the MERN stack.

- Identify the prerequisite knowledge required (HTML, CSS, JavaScript).
- Plan the weekly learning progression from frontend basics to backend development and integration.

This planning ensured a smooth transition from revision-oriented work to advanced full-stack implementation.

2.5.2 Structured Weekly Learning Framework

The entire training was organized using a structured week-wise approach, where each week had clearly defined objectives:

- **Week 1:** Consolidation of core web fundamentals (HTML, CSS, Bootstrap, JavaScript).
- **Week 2:** React environment setup, component architecture, hooks, and frontend API integration.
- **Week 3:** Backend development with Node.js, Express.js, and MongoDB integration.
- **Week 4:** Full-stack communication between React, Node, and MongoDB, followed by project preparation.

This modular structure ensured that new concepts were introduced only after the required foundational skills were strengthened.

2.5.3 Demonstration-Based Concept Introduction

Every topic introduced during the training followed a demonstration-first approach. Instructors showcased:

- Code structure and syntax,
- Step-by-step implementation,
- Real-time output,

- Common errors and their solutions.

This ensured clarity before participants attempted the exercises individually.

2.5.4 Hands-On Practice and Task-Based Reinforcement

The methodology strongly emphasized practical implementation. After each topic:

- Small tasks were assigned (e.g., DOM manipulation exercises, reusable component creation, CRUD APIs).
- Mini templates and component-based activities were completed.
- Backend operations (routing, controllers, database queries) were implemented and tested.
- API integration tasks were performed using Fetch and Axios.

This continuous hands-on practice strengthened problem-solving skills and allowed theoretical concepts to be applied immediately.

2.5.5 Progressive Complexity and Layer-by-Layer Learning

The training followed a bottom-up structure, where the complexity increased gradually:

1. Static structure creation (HTML & CSS).
2. Interactivity through JavaScript and DOM.
3. Component-driven design with React.
4. API development and server logic in Node & Express.
5. Database management and schema creation using MongoDB.
6. Full-stack integration and end-to-end data flow.

This progression ensured that each layer of the MERN stack was understood individually before combining them.

2.5.6 Tool-Based Learning and Environment Familiarization

A practical development environment was maintained throughout:

- VS Code for coding,
- Node and npm for dependency management,
- Postman for backend testing,
- MongoDB Compass/CLI for database visualization,
- Browser DevTools for frontend debugging.

Tools were not taught separately; instead, they were introduced naturally during the development of real tasks.

2.5.7 API-Centric Development Workflow

Once backend concepts were introduced, the methodology followed a consistent API workflow:

1. Build the server route (GET/POST/PUT/DELETE).
2. Write the logic in controller functions.
3. Integrate database queries using Mongoose.
4. Test the API in Postman.
5. Connect the API to React using Axios/Fetch.
6. Render the data dynamically on the frontend.

This workflow provided a complete understanding of request-response lifecycle in full-stack systems.

2.5.8 Iterative Testing and Validation

Testing was integrated into every stage of development:

- DOM updates were verified through console logs.
- React states and components were tested through React DevTools.
- API requests were validated through Postman.
- MongoDB operations were confirmed through Compass and the CLI.

This iterative testing approach ensured reliability and early detection of issues.

2.5.9 Version Control and Code Management

A version-controlled development workflow was followed:

- Initializing Git repositories,
- Making frequent commits after completing modules,
- Pushing updates to GitHub,
- Pulling changes when required.

This promoted clean project management and reflected professional development practices.

2.5.10 Transition to Full-Stack Integration and Project Preparation

In the final week, the methodology shifted to integrating all previously learned components:

- Frontend and backend were connected through REST APIs.
- Database operations were linked with UI components.
- End-to-end flow (React → Node → MongoDB → React) was practiced.
- A structure was prepared for future project development.

CHAPTER-3 RESULT AND DISCUSSION

This chapter outlines the outcomes of the one-month web development training program at HB Infotech Solutions and discusses the knowledge and skills acquired during the training. It also highlights the practical application of these skills through a web development project:

3.1 Key Outcomes

3.1.1 HTML:

- Mastered HTML structure, tags, and syntax.
- Created forms for user input and tables for data display.
- Used semantic tags for better accessibility and SEO.

3.1.2 CSS:

- Learned CSS syntax and layout techniques.
- Styled typography and optimized designs for responsiveness with media queries.

Figure 3.1 Netflix Clone Template

3.1.3 Bootstrap:

- Used Bootstrap's 12-column grid for responsive layouts.
- Implemented components like navigation bars, buttons and forms.
- Customized themes and created responsive tables.
- Designed and implemented a complete Bootstrap template.

Figure 3.2 Learnex Clone Template

3.1.4 JavaScript:

- Gained a solid understanding of **JavaScript syntax**, including variables, operators, data types, and functions.
- Handled **user interactions** using events like **onclick**, **onmouseover**, and **onmouseout**.

- Applied **basic control flow** (if-else, loops)

3.1.5 MERN STACK

1. React.js (Frontend Development)

- Used React Hooks such as useState and useEffect to manage state and lifecycle behavior.
- Implemented routing with React Router to create multi-page applications.
- Built responsive, interactive front-end interfaces and connected them to backend APIs.

1. Node.js (Server-Side Runtime)

- Explored non-blocking, asynchronous programming using callbacks and async/await.
- Used Node's module system to organize backend code efficiently

2. Express,js (Backend Framework)

- Built RESTful APIs to handle frontend requests and perform server-side operations.
- Designed API routes for CRUD operations: Create, Read, Update, Delete.
- Used Express Router to structure backend routes into modular files.

3. MongoDB and Mongoose (Database Layer)

- Created collections and documents using JSON-like structures.
- Performed CRUD operations directly on MongoDB using commands and queries.
- Used Mongoose to define schemas, validate data, and interact with MongoDB efficiently.
- Managed database connections, models, and queries within the backend project structure.

3.2 Project Demonstration

To demonstrate the practical application of the skills acquired during the training period, a full-stack web application titled **EventSphere** was developed. This platform highlights several essential functionalities that reflect modern web development practices. The application showcased the following features:

- **User-Friendly Interface:** A clean and structured layout enabling seamless browsing and interaction.

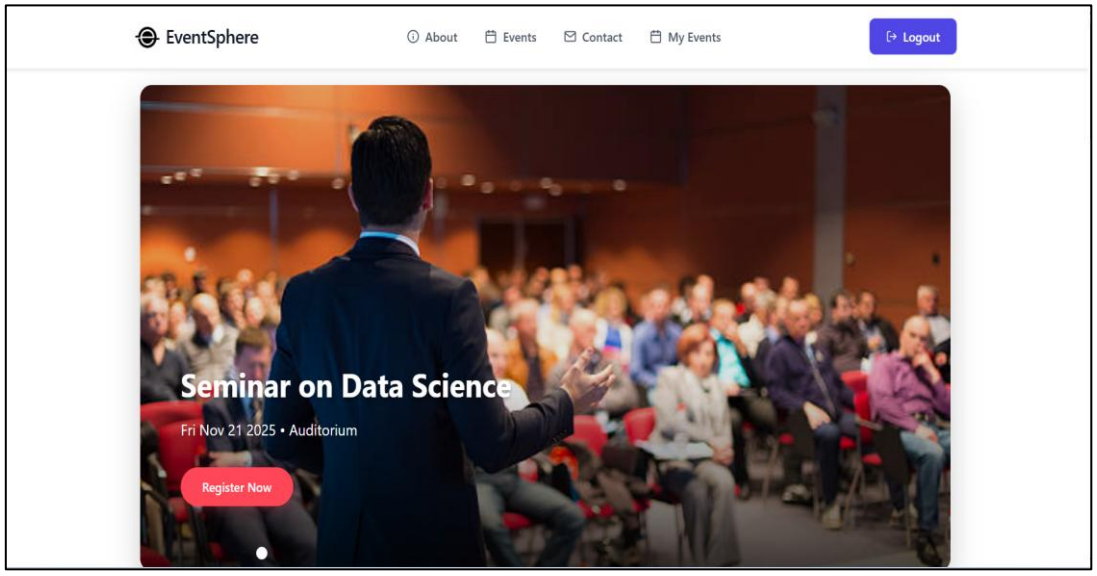

Figure 3.3 EventSphere's User Dashboard

- **Event Display Section:** Categorized event listings such as Ongoing, Upcoming, and Past Events for easy exploration.

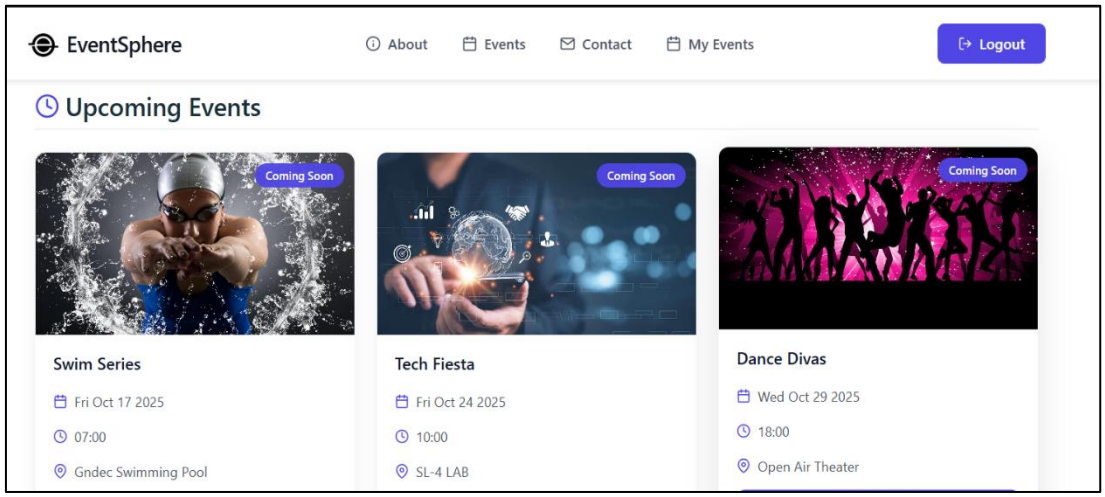

Figure 3.4 EventSphere's event section

- **Secure User Authentication:** A multi-step signup process with OTP verification, login system, and user-specific access.
- **Interactive User Actions:** Users can register for events, set reminders, and view their event history.

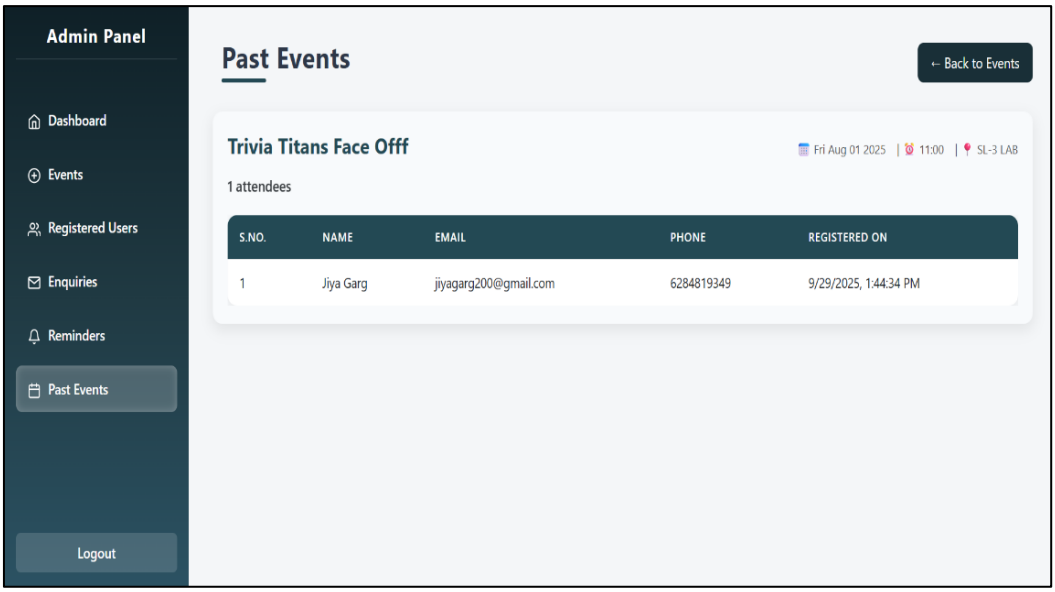

Figure 3.5 EventSphere's Past Events section

- **Contact Form Integration:** Allows users to submit queries, which are stored and visible to the admin.
- **Admin Dashboard:** A dedicated admin panel to manage events, registrations, reminders, and enquiries efficiently.

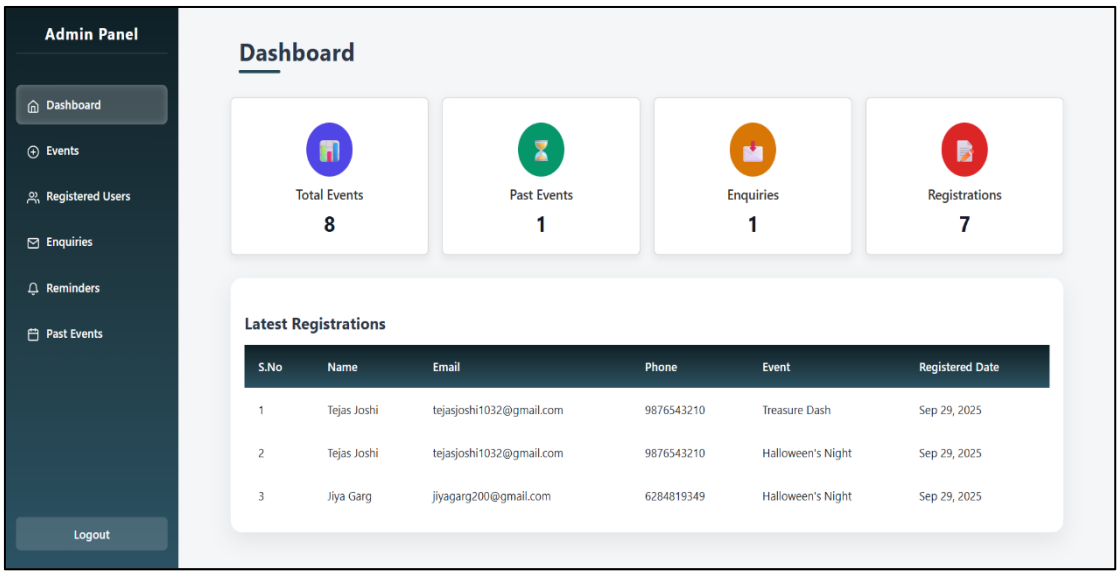

Figure 3.6 EventSphere's Admin Dashboard

- **Event Management Tools:** Admins can add, edit, or delete events using an intuitive form with image and file upload support.

- **Email Reminder System:** Admins can send bulk reminder emails to users who opted for event notifications.

Overall, the project demonstration effectively highlights the integration of frontend and backend technologies to create a fully functional, dynamic event management system. The developed application successfully reflects the practical skills gained during the training period.

3.3 Discussion

The one-month MERN stack training provided a strong introduction to full-stack web development, covering essential concepts in React, Node.js, Express, and MongoDB. Although full mastery of these technologies requires further practice, the trainee gained a clear understanding of component-based UI design, API development, server-side logic, and database operations.

The hands-on project played a key role in reinforcing these skills by demonstrating how frontend and backend systems interact to create a functional application. Through building the project, the trainee improved their problem-solving abilities, debugging skills, and understanding of how data flows between user interfaces, APIs, and databases.

Overall, the training effectively strengthened foundational full-stack development skills and provided practical experience with the MERN stack, establishing a solid base for continued learning and growth in modern web development.

CHAPTER-4 CONCLUSION AND FUTURE SCOPE

4.1 CONCLUSION

The MERN full-stack development training has provided a comprehensive understanding of how modern web applications are built using MongoDB, Express.js, React.js, and Node.js. Through hands-on practice, I gained real-world experience in both frontend and backend development ranging from designing responsive interfaces to building secure APIs and managing databases. The training strengthened my ability to integrate multiple technologies into a complete, functional system and enhanced my problem-solving, debugging, and logical thinking skills. Overall, this training laid a strong foundation in full-stack development and prepared me to participate confidently in industry-level projects, collaborate with development teams, and continue advancing in modern application development.

4.2 FUTURE SCOPE

- **Cloud Deployment & DevOps:** Learning cloud platforms like AWS, Azure, or using tools like Docker and CI/CD pipelines will enable smoother deployment and scaling.
- **Advanced JavaScript & Frameworks:** Exploring technologies such as Next.js, Redux Toolkit, or TypeScript can help build more efficient, large-scale applications.
- **Real-Time Web Applications:** Integrating WebSockets or Socket.io will allow development of real-time features such as live chats, notifications, and dashboards.
- **Enhanced Security Practices:** Implementing OAuth, refresh tokens, encryption techniques, and role-based access control will improve application reliability and user protection.

The future of full-stack development continues to evolve with new tools, faster frameworks, and smarter technologies.

REFERENCES

[1] M. Cantelon, M. Harter, T. Holowaychuk and N. Rajlich, *Node.js in Action*. Shelter Island, NY, USA: Manning Publications, 2017.

[2] R. H. Purcell, *Learning React: Modern Patterns for Developing React Apps*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2021.

[3] Node.js Documentation – OpenJS Foundation, "Node.js Official Documentation." [Online]. Available: https://nodejs.org

[4] A. Banker, *MongoDB in Action*, 2nd ed. Shelter Island, NY, USA: Manning Publications, 2016.

[5] JSON Web Tokens, "JWT Introduction and Documentation." [Online]. Available: https://jwt.io

[6] React Documentation – React.js, "React Official Documentation," Meta Platforms, Inc. [Online]. Available: https://react.dev

[8] Express.js Documentation – OpenJS Foundation, "Express.js Guide." [Online]. Available: https://expressjs.com

[9] MongoDB Inc., "MongoDB Manual and Documentation." [Online]. Available: https://www.mongodb.com/docs