PUNYASHLOK AHILYADEVI HOLKAR SOLAPUR UNIVERSITY

A Report On

"INTERNSHIP"

at

"LeadSoft IT Solutions."

In fulfillment of Vocational Training for Bachelor of Technology (B.Tech) in design by

Punyashlok Ahilyadevi Holkar Solapur University

Submitted by,

Aishwarya Bapu Asabe

Academic Year 2024-2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING SKN SINHGAD COLLEGE OF ENGINEERING, KORTI, PANDHARPUR

Savitribai Phule Shikshan Prasarak Mandal's

SKN SINHGAD COLLEGE OF ENGINEERING, KORTI, PANDHARPUR

CERTIFICATE

This is to certify that the **"Internship"** at **"LeadSoft IT Solutions"** has been successfully done by **"Ms. Aishwarya Asabe"** This report is approved in fulfillment of the Vocational Training for Bachelor of Technology (B. Tech) of the Punyashlok Ahilyadevi Holkar Solapur University, Solapur.

Dr. S.V.Pingale
(HOD)

Prof.S .G.Linge
Internship Coordinator

ACKNOWLEDGMENT

I am very glad to submit this industrial training report, which was satisfactorily completed at LeadSoft IT Solutions in the MEAN Stack Development department. I gratefully acknowledge the inspiring guidance and continuous support of the team at LeadSoft IT Solutions, whose helpful suggestions and encouragement were instrumental in completing this project.

I would also like to express my gratitude to my Head of Department, Prof. S.V. Pingale, and the Training and Placement Coordinator, Prof. S.G. Linge, for their cooperation throughout the training period.

Aishwarya Asabe
B.Tech - Computer Science and Engineering Roll
No: 71

INDEX

Introduction of Course

The internship at LeadSoft IT Solutions focused on MEAN Stack Development, an increasingly popular framework for building full-stack web applications. MEAN Stack is a collection of JavaScript-based technologies — MongoDB, Express, Angular, and Node.js — that provide an efficient, scalable, and flexible structure for both front-end and back-end development.

As a web development intern, my primary responsibility was to gain hands-on experience in designing and developing web applications, focusing on real-world project implementation. The course provided a comprehensive understanding of how to use each component of the MEAN stack to build efficient and scalable web applications.

- MongoDB: A NoSQL database that stores data in a flexible, JSON-like format, allowing for easy scalability and storage of large amounts of unstructured data.
- Express.js: A lightweight framework for building web applications in Node.js, simplifying the process of routing and middleware implementation.
- Angular: A powerful front-end framework maintained by Google, which provides tools for building dynamic, single-page applications with two-way data binding and componentbased architecture.
- Node.js: A server-side platform that allows JavaScript to be used for back-end programming, offering fast and scalable network applications by leveraging non-blocking, event-driven architecture.

This course emphasized both theoretical and practical aspects of web development. While theoretical learning provided a solid understanding of concepts like REST APIs, server-client interaction, and asynchronous programming, the practical application focused on building a blogging website from scratch. This website allowed users to register, create, update, read, and delete posts (CRUD operations), covering the entire lifecycle of web application development.

Abstract

This report discusses my internship at LeadSoft IT Solutions, where I worked on a full-fledged blogging website project. This project involved performing all the essential CRUD (Create, Read, Update, Delete) operations, utilizing the MEAN stack. The goal of the project was to implement the front-end using Angular and the back-end services using Node.js and Express, with MongoDB serving as the database.

Learning Objective

The objectives of this internship were:

- To understand and implement the core components of the MEAN stack.
- To perform all CRUD operations on a web-based application.
- To develop proficiency in back-end and front-end development, including database management.
- To improve problem-solving and debugging skills, especially in a live project environment.
- To learn version control and collaboration using Git.

Technical section

1. Project Overview: Blogging Website

The project involved building a blogging website with features such as:

- **User Registration/Login**: Secure authentication and authorization using JWT.
- **Post Creation**: Users can create new blog posts, which are stored in the MongoDB database.
- **Post Management**: Blog posts can be read, updated, and deleted, ensuring a complete CRUD cycle.
- **Frontend**: Angular was used to create an intuitive and responsive user interface.
- **Backend**: Node.js with Express was utilized to handle API requests and interact with MongoDB.

2. CRUD Operations

- **Create**: Users can create a new blog post with a title, content, and category.
- **Read**: All blog posts are displayed on the homepage, with each post accessible via a unique URL.
- **Update**: Logged-in users can edit their existing blog posts.
- **Delete**: Users can delete their posts, and it will be removed from the database.

1. MongoDB (Database Layer)

MongoDB is a NoSQL database used to store and manage the application's data. In the blogging project, MongoDB was utilized to store user information (such as login credentials) and blog post data. MongoDB's document-oriented structure makes it ideal for applications where data structure can be dynamic and flexible.

- User Schema: Defined the structure for user registration, including fields for username, email, password, and profile details. Passwords were encrypted using hashing techniques for security.
- Blog Post Schema: This schema defined fields for the title, content, author, and timestamps (createdAt and updatedAt). Each blog post was linked to the author, ensuring proper userpost relationships.

Key MongoDB operations included:

- Create: Insert new user and post documents into the database.
- Read: Retrieve posts from the database to display on the homepage or the user's dashboard.
- Update: Allow users to edit their existing posts, ensuring changes are reflected in the database.
- Delete: Remove a post from the database based on user action.

---

2. Express.js (Server Layer)

Express.js was used as the back-end framework to handle API requests and manage communication between the front-end and the database. It simplified the development of RESTful APIs and allowed for smooth handling of routing, middleware, and error handling.

- RESTful APIs: The application followed REST principles to perform CRUD operations. Each operation (create, read, update, delete) was implemented as a separate endpoint using Express.

- POST /register: For user registration.
- POST /login: For user authentication.
- POST /create-post: To create a new blog post.
- GET /posts: To retrieve all blog posts or posts by a specific user.
- PUT /edit-post/:id: To update a specific blog post.
- DELETE /delete-post/:id: To delete a post by its ID.

- Middleware: Middleware functions were employed for tasks like parsing request bodies, handling authentication using JWT (JSON Web Tokens), and managing session data.
- Error Handling: Express.js offered structured error handling, ensuring that invalid API requests were caught and appropriate error messages were returned.

---

3. Angular (Front-End Layer)

Angular was used to build the client-side of the application, providing a dynamic and responsive user interface. The component-based structure of Angular allowed for reusability and modularity, making the application scalable and maintainable.

Key Angular features implemented:

- User Authentication: The login and registration components handled user authentication. JWT tokens were stored in local storage and attached to API requests to verify user access rights.
- Blog Post Components: Various components were created to handle different aspects of the blog:
  - CreatePostComponent: Allowed users to create new blog posts using forms. ○ EditPostComponent: Provided an interface to edit existing blog posts.

- PostListComponent: Displayed a list of blog posts fetched from the server. Pagination was implemented for improved user experience when there are many posts.
- PostDetailComponent: Rendered a single blog post, showing details like the title, content, author, and creation date.

- Routing: Angular's routing module was used to navigate between different pages (e.g., home, login, dashboard, post creation) without reloading the entire application.
- Form Validation: Implemented both client-side and server-side form validation for user input, ensuring the integrity of the data being submitted (e.g., checking if fields are filled correctly, ensuring the email is in the correct format, etc.).
- Data Binding: Two-way data binding was extensively used for handling user inputs and keeping the user interface updated in real-time.

---

4. Node.js (Execution Environment)

Node.js was used to create a server-side runtime environment that is event-driven and nonblocking, ideal for handling I/O-bound tasks such as database operations and API calls. Since both Angular and Node.js are JavaScript-based, it ensured consistency in the development stack, making it easier to switch between front-end and back-end development.

Key Node.js features:

- Non-blocking I/O: This allowed the server to handle multiple requests simultaneously, improving performance, especially in a production environment.
- Asynchronous Programming: Promises and async/await were extensively used to manage asynchronous tasks such as database queries, API requests, and file handling.
- Package Management: npm (Node Package Manager) was used to install and manage various packages, such as bcrypt for password hashing, jsonwebtoken for authentication, mongoose for MongoDB interactions, and others.

5. Authentication & Security

To ensure secure user authentication, JWT (JSON Web Tokens) were used. After a successful login, a token was generated and sent to the client, which was then stored in local storage and used to authenticate future API requests.

Security measures implemented:

- Password Hashing: Before storing user passwords in the database, they were hashed using bcrypt to prevent plain text passwords from being exposed in case of a data breach.
- JWT Authentication: Protected routes were established, where only authenticated users with valid tokens could access certain endpoints (e.g., creating or editing a blog post).
- Cross-Origin Resource Sharing (CORS): Configured CORS to allow secure cross-origin API calls between the client (Angular) and server (Node.js).

---

6. Version Control and Collaboration

The project utilized Git for version control, allowing easy tracking of code changes and collaboration with team members. Key features included:

- Branching: Separate branches were created for feature development (e.g., "featureauthentication", "feature-CRUD"), ensuring that the main branch remained stable.
- Pull Requests & Merging: Code was reviewed and merged into the main branch after testing to maintain code quality and reduce bugs.

---

7. Testing

The application was tested using tools like Postman for API testing and manual testing for user interface components. Postman allowed testing of different API endpoints to ensure they functioned as expected under different scenarios (e.g., invalid inputs, unauthorized access).

8. Deployment

The application was deployed on a cloud-based service for demonstration purposes. Heroku was used to deploy the server (Node.js) and Netlify was used for the front-end (Angular) to ensure seamless hosting and availability.

The blogging website project involved leveraging the MEAN Stack to its full potential, applying best practices, advanced development patterns, and additional tools to optimize both the development process and the performance of the application. The following sections describe the advanced aspects of each layer of the MEAN stack, including optimization techniques, thirdparty tools, and modern design patterns that were implemented during the internship.

1. Advanced MongoDB Techniques (Database Layer)

MongoDB was used as the database solution to handle dynamic data needs. Beyond simple CRUD operations, several advanced MongoDB features and techniques were implemented to enhance performance, scalability, and data integrity:

- Indexing: Indexes were created on frequently queried fields, such as userId and postId, to improve query performance, especially as the database size grows. This reduced the lookup time and increased the overall efficiency of database operations.
- Aggregation Framework: MongoDB's aggregation pipeline was used to handle complex data queries, such as counting the total number of posts by each user and calculating the most active users. This allowed for efficient data transformation and real-time statistics generation without the need for multiple queries.
- Sharding and Replication: While not fully implemented in the internship environment, the project architecture was designed with horizontal scaling in mind. MongoDB's sharding

and replication features were considered for future scalability, ensuring that the application could handle large amounts of traffic and data distribution across multiple servers.

- Data Validation: Leveraging Mongoose's built-in validation, custom validators were created to ensure data integrity, such as checking for unique usernames and properly formatted email addresses. These validations were essential in preventing invalid or duplicate data entries.

---

2. Express.js with Best Practices (Server Layer)

Express.js was used to create the RESTful API, connecting the front-end with the back-end. To ensure maintainability, scalability, and security, several advanced techniques and patterns were employed in the server-side architecture:

- MVC Pattern (Model-View-Controller): The backend was organized using the MVC architecture. This separation of concerns ensured that the application was modular and easier to maintain. The model (handled by Mongoose), controller (handling business logic), and view (JSON responses for the client) were clearly defined:
    - Controllers: Each controller was responsible for handling a specific set of related routes (e.g., AuthController for authentication, PostController for blog posts).
    - Middleware: Middleware functions were extensively used for modularizing code, such as handling user authentication, input validation, and error handling.
- Error Handling Middleware: A global error handler was implemented to catch all runtime errors and ensure that proper error responses were returned to the client. This centralized error handling reduced code duplication and enhanced code readability.
- Rate Limiting: To protect the application from brute-force attacks and DDoS attacks, rate limiting middleware (such as express-rate-limit) was applied to certain routes, like login and registration, limiting the number of requests from a specific IP within a given time frame.

Helmet.js for Security: To secure HTTP headers and protect the application from vulnerabilities such as cross-site scripting (XSS) and clickjacking, helmet.js was integrated into the Express server.

- CORS (Cross-Origin Resource Sharing): CORS was configured to allow the Angular front-end to communicate with the Node.js API securely. This included whitelisting specific domains and methods for additional security.

---

3. Angular with Advanced Features (Front-End Layer)

The Angular front-end was built with advanced features and best practices to ensure performance, maintainability, and a great user experience.

- Lazy Loading: To improve initial load time and performance, lazy loading was implemented. This allowed for the loading of certain modules (such as the post creation page or dashboard) only when needed, reducing the bundle size of the initial load.
- State Management with RxJS and Services: Angular's RxJS was extensively used to manage data streams and asynchronous events. Observable patterns were followed to fetch data from the API, and Angular services were created to maintain state and handle business logic across different components. This separation of concerns ensured that the application remained clean and modular.
- Form Management with Reactive Forms: Angular Reactive Forms were used to handle complex form validation and dynamic form generation. This approach offered greater control over form input and validation processes, including asynchronous validators (e.g., checking if a username already exists) and custom validation rules.
- Reusable Components: A component-based architecture ensured that certain UI elements, such as blog post cards, forms, and navigation bars, were reusable throughout the application. This modular design enhanced maintainability and reduce
- Optimizing Performance with Change Detection: Angular's default change detection mechanism was optimized by using OnPush strategy, which only checks components

when an input reference changes. This optimization improved the performance of the application, especially when dealing with large datasets or frequent user interactions.

---

4. Node.js for High-Performance Server-Side Processing

Node.js, being event-driven and non-blocking, was ideal for building scalable and highperformance applications. Several strategies were employed to further optimize the server-side performance and scalability:

- Asynchronous Programming with Promises and Async/Await: The use of async/await syntax in handling asynchronous operations such as database queries ensured clean, readable code. This approach also made error handling easier and improved the overall responsiveness of the application.
- Clustering for Scalability: Node.js clustering was configured to take advantage of multicore processors. By spawning multiple worker processes, the application was able to handle more simultaneous connections without being constrained to a single thread.
- Memory Management: Proper memory management techniques were implemented to prevent memory leaks, especially when handling large datasets or streaming data. Node.js's built-in garbage collection was closely monitored to ensure efficient memory usage.
- Streaming Large Data: When handling large sets of data (e.g., exporting all blog posts or importing a large dataset), Node.js's streaming capabilities were used instead of loading the entire dataset into memory. This technique reduced memory consumption and improved performance for I/O-bound tasks.

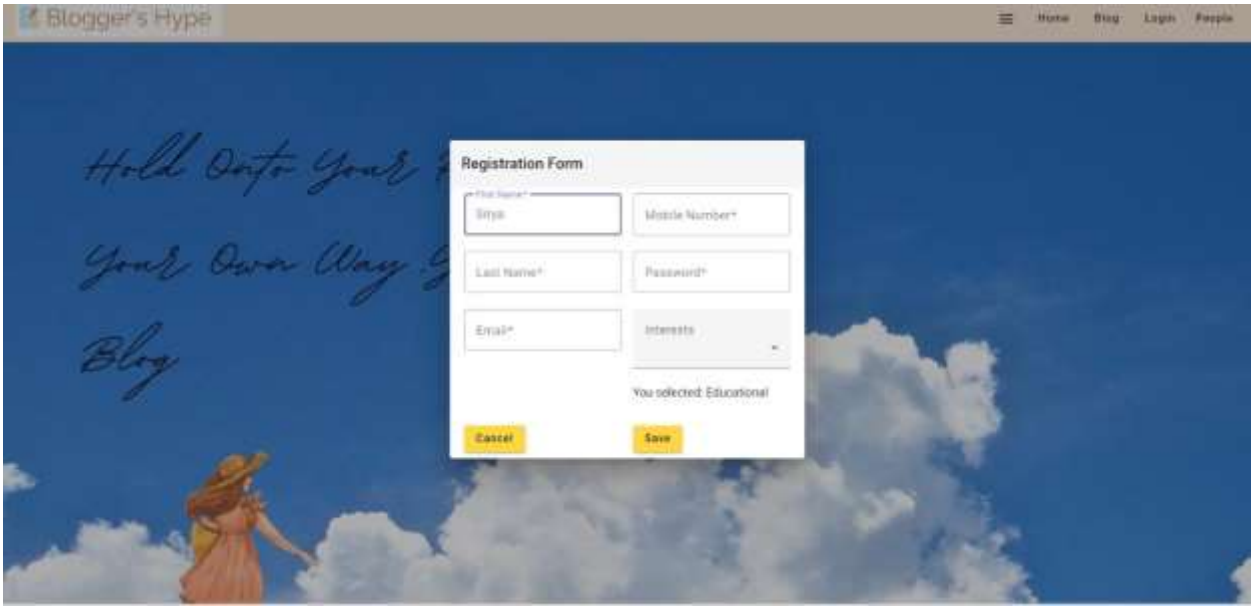
---

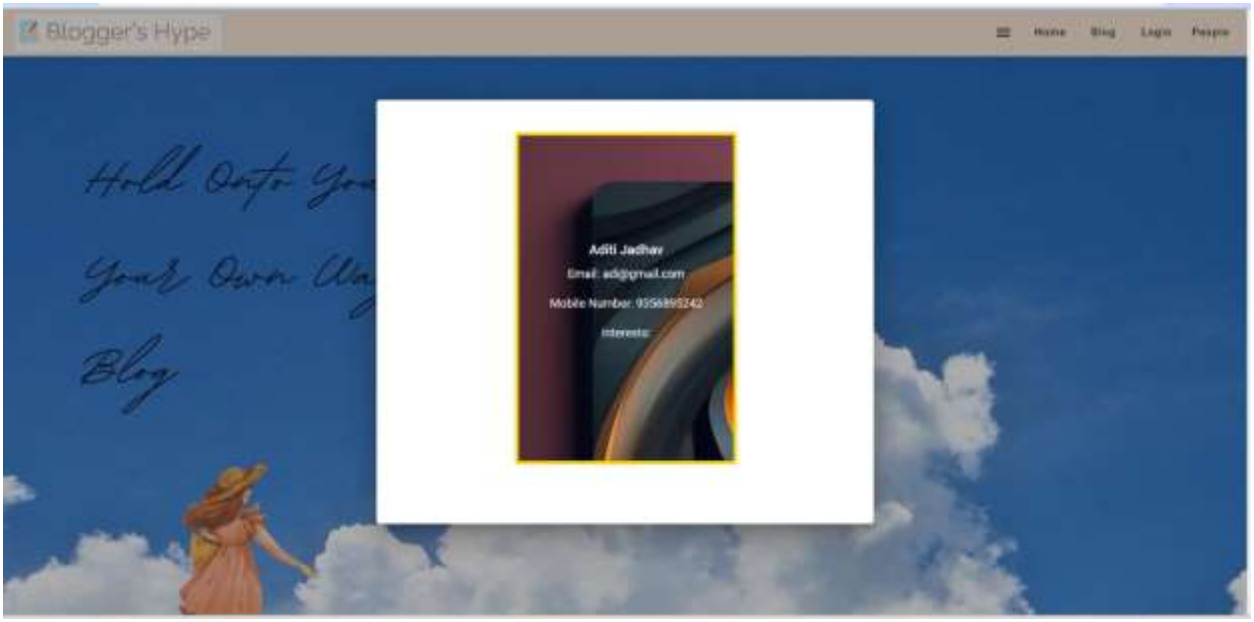
5. Authentication and Authorization

JWT (JSON Web Tokens) was used for secure authentication, allowing users to log in and securely access resources. Additional security features were implemented to ensure the safety of user data:

JWT Token Expiration and Refresh Tokens: To enhance security, tokens were configured to expire after a certain period. Refresh tokens were implemented to allow users to get new access tokens without having to log in again, providing a seamless user experience while maintaining security.

- Role-Based Access Control (RBAC): The application included role-based access control to manage different user permissions. For example, users with the role of “admin” could manage all posts, while regular users were limited to managing their own content.
- Two-Factor Authentication (2FA): For added security, the potential for implementing 2FA using Google Authenticator or SMS verification was explored, though not fully integrated into the current version of the project.


Blogger's Hype
Home
Blog
Login
People
Registration Form
Mobile Number*
Last Name*
Password*
Email*
Interests
You selected: Educational
Cancel
Save


Blogger's Hype
Home
Blog
Login
People
Aditi Jadhav
Email: adi@gmail.com
Mobile Number: 9356895242
Interests:


Create a Blog Post
Menu
Create Blog
See Blogs
Profile
Date:
Thu Nov 30 2023 22:11:09 GMT+0530 (India Standard Time)
Author Name:
Type:
Content:


Outfit Accordance
Date: Oct 27, 2023
Author: saniya Alat
Type: Fashion
Content: In fact, it will age their skin tone dramatically. 'Black gives the illusion of slimming down the wearer, and designers will have us believe that it's the colour of the season and you must not be seen in anything else. But it can also bring out dark circles under the eyes and lines. One reason dark colours have this effect is that they draw the eye away from the body to the head, hands and feet, making you appear taller and slimmer. They also help hide any bumps or bulges. By contrast, lighter colours emphasise your contours, as these are more likely to produce shadows in light-coloured clothing. With black comes a heaviness of heart, which may 'feel appropriate' in the circumstances, but does mean the wearers are likely to be absorbing dangerous levels of sadness and negativity which could lead to increased anxiety, depression and overwhelm.

Outcome

Through this internship, I gained hands-on experience in full-stack web development, specifically:

- Mastered the fundamentals of the MEAN stack, including MongoDB, Express, Angular, and Node.js.
- Built a fully functional blogging website, managing user authentication and implementing CRUD operations.
- Enhanced my understanding of RESTful API development and front-end/back-end integration.
- Developed soft skills such as communication, teamwork, and time management by working in a collaborative environment.

Conclusion

My internship at LeadSoft IT Solutions was a valuable experience that allowed me to apply theoretical knowledge to real-world projects. The exposure to MEAN Stack development, along with the chance to work on a live project, significantly improved both my technical and soft skills. I would highly recommend this internship program to my peers, as it provided practical insights into the web development industry.

Certificate

References:

1)