Project Title

Full Stack Blog Application Using MERN Stack

A Mini Project Report Submitted in Partial Fulfillment of the Requirements for the Award of the Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

XXXXXXXXX

Under the Guidance of

PROJECT GUIDE NAME

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COLLEGE NAME

AFFILIATED TO UNIVERSITY NAME

YEAR: 2026

2. Bonafide Certificate

BONAFIDE CERTIFICATE

This is to certify that the Mini Project Report entitled "**Full Stack Blog Application Using MERN Stack**" is a bonafide work carried out by xxxxxxxx in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering.

This work has been carried out during the academic year 2025–2026 under my supervision and guidance.

Project Guide Signature: _____

Head of Department Signature: _____

Department of Computer Science and Engineering

College Name

Date: _____

3. Declaration by Student

DECLARATION

I hereby declare that the project report entitled "**Full Stack Blog Application Using MERN Stack**" submitted for the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering is my original work.

This project has been carried out under the supervision of my project guide. The work presented in this report has not been submitted previously to any other university or institution for the award of any degree or diploma.

Place: _____

Date: _____

Signature of the Student

(xxxxxxxxx)

4. Acknowledgement

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my project guide for their valuable guidance, encouragement, and continuous support throughout the development of this project titled "**Full Stack Blog Application Using MERN Stack**".

I am also thankful to the Head of the Department of Computer Science and Engineering for providing the necessary facilities and support to successfully complete this project.

I extend my heartfelt thanks to all the faculty members of the department for their valuable suggestions and encouragement during the project work.

Finally, I would like to express my deepest gratitude to my family and friends for their constant motivation, encouragement, and support throughout the completion of this project.

INDEX

CONTENTS

Abstract

The **Full Stack Blog Application** is a dynamic and responsive web platform developed using the MERN stack, which includes MongoDB, Express.js, React.js, and Node.js. The application allows users to register and log in securely, create, edit, and delete blog posts, and interact with content through comments.

It also provides advanced features such as search functionality, category-based filtering, and user profile management. Authentication and authorization are implemented using JSON Web Tokens (JWT), ensuring secure access to user-specific operations.

The frontend is designed using React.js along with Tailwind CSS to provide a clean and responsive user interface, while the backend is built using Node.js and Express.js to handle API requests and business logic. MongoDB is used as the database to efficiently store and manage application data.

This project demonstrates a complete full-stack development workflow, including frontend design, backend API development, database integration, and secure authentication mechanisms.

CHAPTER-1

INTRODUCTION

1.INTRODUCTION

In the present digital world, web applications play an important role in everyday life. Many applications are developed to perform different tasks such as communication, content sharing, data management, and collaboration. With the rapid growth of internet usage, users expect applications to be fast, interactive, and user-friendly. Modern web technologies like the MERN stack have made it possible to develop such applications efficiently.

The MERN stack is one of the most popular technology stacks used for building full-stack web applications. It includes MongoDB, Express.js, React.js, and Node.js. These technologies allow developers to build scalable and efficient applications using JavaScript for both frontend and backend development. React.js helps in creating dynamic user interfaces, while Node.js and Express.js handle server-side operations, and MongoDB is used for storing data.

The **Full Stack Blog Application** is developed as a web-based system using the MERN stack. The main aim of this project is to create an interactive platform where users can register, log in, and share their ideas through blog posts. The application provides a simple interface for creating, editing, and managing blog content.

In this system, the user interacts with the frontend developed using React.js. The application allows users to create blog posts, update them, and delete them when required. The backend, developed using Node.js and Express.js, handles the business logic and communicates with the MongoDB database. The data is stored and retrieved efficiently, and the results are displayed dynamically on the user interface.

The application is designed to be simple, efficient, and user-friendly. React.js improves the performance of the application by using features like component-based architecture and virtual DOM. This ensures that only necessary parts of the interface are updated, resulting in faster execution.

Thus, this project demonstrates how the MERN stack can be used to build modern web applications with interactive features, efficient performance, and secure data management.

1.1About MERN Stack

The MERN stack is a popular JavaScript-based technology stack used for developing full-stack web applications. It consists of MongoDB, Express.js, React.js, and Node.js. Each of these technologies plays a specific role in the development process.

MongoDB is a NoSQL database used to store application data in a flexible, JSON-like format. It allows efficient storage and retrieval of data such as user details, blog posts, and comments.

Express.js is a lightweight web application framework for Node.js. It is used to build backend APIs and handle HTTP requests and responses. It simplifies server-side development and improves performance.

React.js is a JavaScript library used for building user interfaces. It follows a component-based architecture, allowing developers to create reusable UI components. React.js also uses a virtual DOM to update only the required parts of the webpage, improving speed and efficiency.

Node.js is a runtime environment that allows JavaScript to run on the server side. It is used to handle backend operations, manage server requests, and connect with the database.

The MERN stack enables developers to use a single programming language (JavaScript) throughout the application, making development faster and more efficient. It is widely used for building modern applications such as social media platforms, blogging systems, and e-commerce websites

1.2 About the Project

The **Full Stack Blog Application** is a web-based system developed using the MERN stack. The main objective of this project is to provide a platform where users can create, manage, and share blog content through a simple and interactive interface.

The application allows users to register and log in securely. After authentication, users can create new blog posts, edit existing posts, and delete posts when needed. The system also supports features like commenting, searching, and category-based filtering.

The frontend of the application is developed using React.js, which ensures a fast and responsive user interface. The backend is developed using Node.js and Express.js, which handle API requests and business logic. MongoDB is used as the database to store user data, blog posts, and comments.

This project demonstrates the use of full-stack development in building real-world applications. It shows how frontend and backend technologies can be integrated to create a complete system. The main contributions are highlighted and discussed below:

- This project demonstrates the use of the MERN stack in developing a web-based application.
- It provides an interactive and user-friendly interface for blog creation and management.
- The system integrates frontend and backend using APIs.
- It ensures secure authentication using JSON Web Tokens (JWT).
- The application supports dynamic content updates and efficient data handling.
- The system can be extended with additional features in the future.

CHAPTER-2

LITERATURE SURVEY

2.1 LITERATURE SURVEY

In this study, the development of full-stack web applications using the MERN stack and their role in building modern interactive systems was examined. The survey also focuses on blogging platforms, user authentication systems, and the integration of frontend and backend technologies in web applications. MERN stack applications have gained significant importance in recent years due to their scalability, flexibility, and performance.

There has been a rapid increase in the use of the MERN stack for developing web applications. Many developers prefer this stack because it allows the use of a single programming language, JavaScript, for both frontend and backend development. Research shows that React.js is widely used for building dynamic user interfaces, while Node.js and Express.js are used to create efficient backend services. MongoDB provides flexible data storage, making it suitable for applications that handle large amounts of user-generated content.

Several studies have explored the development of blogging platforms and content management systems using modern web technologies. These systems allow users to create, edit, and manage content easily. Features such as commenting, search functionality, and category-based filtering improve user engagement and content accessibility. The use of RESTful APIs enables smooth communication between frontend and backend components.

User authentication and security are important aspects of web applications. Many research works focus on implementing secure authentication mechanisms such as JSON Web Tokens (JWT). JWT-based authentication ensures that only authorized users can access and modify data. This improves the overall security and reliability of the system.

The integration of React.js with backend technologies allows real-time updates and dynamic content rendering. React.js uses a virtual DOM to update only the necessary parts of the webpage, improving performance and user experience. This is especially useful in blogging applications where content is frequently updated.

In addition, database management plays a crucial role in web applications. MongoDB, being a NoSQL database, allows flexible schema design and efficient handling of large datasets. Studies highlight that MongoDB is well-suited for applications that require scalability and fast data retrieval.

Overall, the literature survey indicates that the MERN stack is highly effective in developing modern web applications. The combination of React.js, Node.js, Express.js, and MongoDB provides a powerful platform for building scalable, secure, and user-friendly systems. These advancements have made it possible to develop efficient blogging platforms that meet the requirements of real-world users.

CHAPTER-3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Traditional blogging systems are mostly static websites with limited interactivity.

• Many systems do not provide user authentication, restricting personalized content management.

• Existing platforms often lack proper content management features like editing, deleting, and categorizing posts.

• Some systems are not fully responsive and do not provide a good user experience on different devices.

• Data handling in traditional systems is inefficient due to lack of proper database integration.

3.1.1 DISADVANTAGES OF EXISTING SYSTEM

• Lack of dynamic content updates and real-time interaction.

• Limited features for users such as editing and managing blog posts.

• No proper authentication and authorization mechanisms.

• Poor user experience due to non-responsive design.

• Inefficient data storage and retrieval methods

3.2 PROPOSED SYSTEM

Develop a full-stack web-based Blog Application using the MERN stack.

• Implement user registration and login functionality using secure authentication (JWT).

• Provide features to create, edit, and delete blog posts.

• Enable commenting, search functionality, and category-based filtering.

• Ensure smooth communication between frontend and backend using RESTful APIs.

• Design a responsive and user-friendly interface using React.js and Tailwind CSS.

3.2.1 ADVANTAGES OF PROPOSED SYSTEM

• Provides a dynamic and interactive user experience.

• Secure authentication ensures data privacy and protection.

• Efficient data management using MongoDB database.

• Responsive design works on all devices.

• Easy to use and user-friendly interface.

• Scalable and can be extended with additional features.

3.3 FEASIBILITY STUDY

3.3.1 Technical Feasibility

According to software engineering principles, technical feasibility refers to the availability of technical resources required to develop and implement the system. The proposed system is developed using modern technologies such as React.js, Node.js, Express.js, and MongoDB. These technologies are widely used and easily available. The system can run on standard computers with internet connectivity and a web browser, making it technically feasible.

3.3.2 Economic Feasibility

Economic feasibility refers to the cost-benefit analysis of the project. The proposed system is developed using open-source technologies, which reduces development cost. Since the application is web-based, users can access it without installing additional software. This minimizes expenses and makes the system economically feasible.

3.3.3 Operational Feasibility

Operational feasibility refers to how effectively the system can be used in real-world conditions. The proposed system is designed to be simple and user-friendly. Users can easily register, log in, create blog posts, and interact with content. No special training is required, making the system operationally feasible.

3.3 SOFTWARE REQUIREMENT SPECIFICATION

3.4.1 Functional Requirements

The functional requirements define the main operations of the system. The following functionalities are provided:

1. User can register and log in to the system.
2. User can create, edit, and delete blog posts.
3. Users can view and search blog posts.
4. Users can comment on blog posts.
5. System manages user authentication using JWT.
6. System handles data storage and retrieval using MongoDB**.**

3.4.2 Non-functional Requirements

Usability
The system provides a simple and interactive user interface, making it easy to use.

Reliability
The system ensures reliable performance with secure authentication and data handling.

Performance
The application provides fast response time using React.js and efficient backend APIs.

Supportability
The system is web-based and can run on any platform with a browser and internet connection.

Implementation
The system is implemented using MERN stack technologies.

Interface
The user interface is designed using React.js, HTML, CSS, and Tailwind CSS.

3.4 SOFTWARE AND HARDWARE REQUIREMENTS

3.5.1 Software Requirements

Operating System : Windows 10 / Any OS

Programming Language : JavaScript

Front End : React.js, HTML, CSS, Tailwind CSS

Back End : Node.js, Express.js

Database : MongoDB

Framework : React.js

Server : Localhost / Cloud Server

3.5.2 Hardware Requirements

Processor : Intel i3 or above

RAM : 4 GB or above

Hard Disk : 500 GB

CHAPTER-4

DESIGN PROCESS

4.1 INTRODUCTION

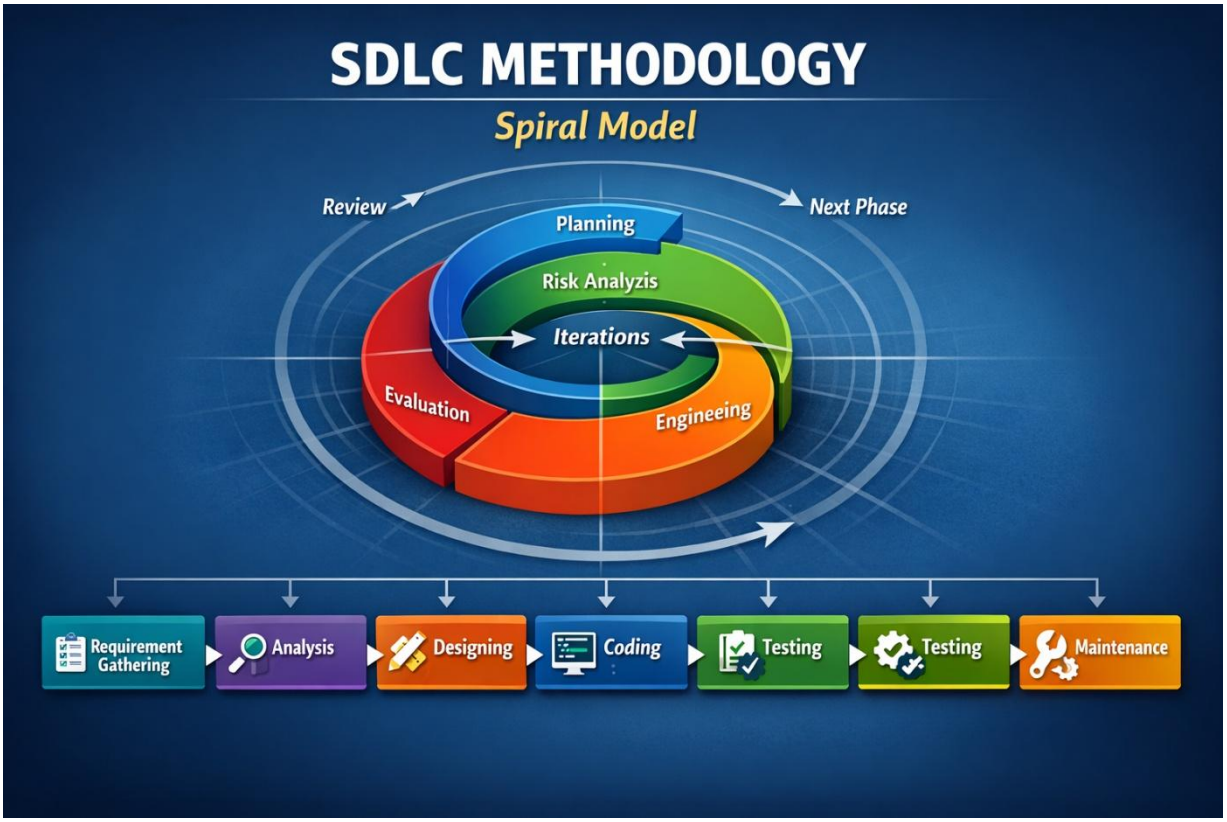
The most creative and important phase of the software development life cycle is system design. The term design describes both the final system and the process by which it is developed. It refers to the technical specifications that are applied during the implementation of the system. Design can be defined as the process of applying various techniques and principles for the purpose of defining a system in sufficient detail to allow its physical realization.

The main goal of design is to determine how the system will function and how the output will be produced. It includes designing input formats, output formats, and database structures that meet the requirements of the system. The processing phase is carried out through coding and testing. Finally, the system is evaluated to understand its impact on users and organizations before implementation.

The importance of software design lies in ensuring quality. Design provides a blueprint that helps in developing a stable and efficient system. Without proper design, the system may fail when changes are introduced, may be difficult to test, and may not meet user requirements. Therefore, design is an essential phase in the development of a software product.

4.1.1 SDLC METHODOLOGY

The Software Development Life Cycle (SDLC) plays a vital role in system development as it defines the complete process of building the application. It serves as a reference for developers throughout the development and testing phases. Any future changes must follow a proper approval process.
The **Spiral Model** is used as the development methodology for this project. It was proposed by Barry Boehm and combines iterative development with systematic risk assessment. It allows continuous improvement through repeated cycles.



Stages in SDLC

- Requirement Gathering
- Analysis
- Designing
- Coding
- Testing
- Maintenance

4.2 SYSTEM ARCHITECTURE

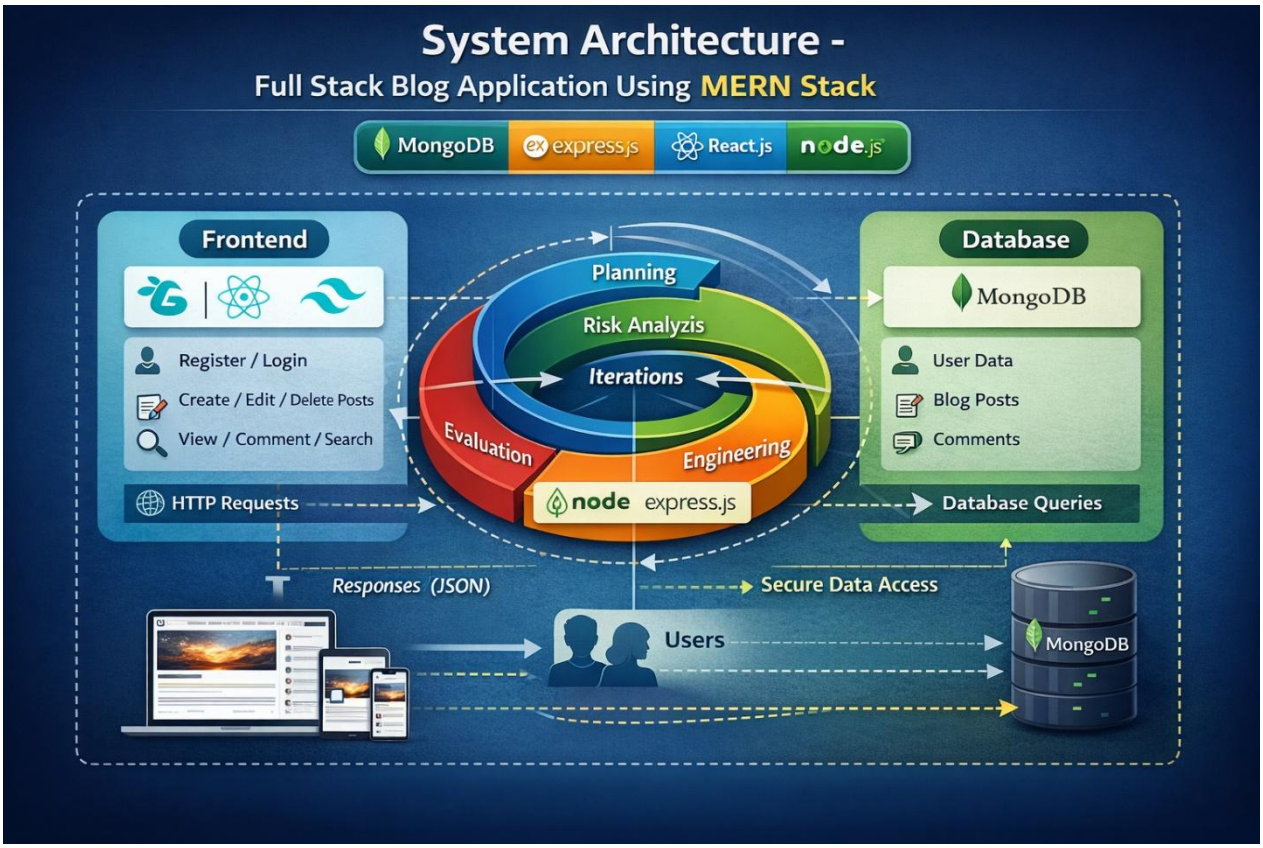


Fig. 4.1: System Overview

4.3 INPUT & OUTPUT REPRESENTATION

Input Design:

Input design includes the methods used for entering data into the system and the validations applied during data entry. In the Full Stack Blog Application, inputs are provided through forms such as registration, login, blog creation, and comment sections.

Different validation messages are displayed to guide users during data entry. Validation checks are performed for each input field, such as required fields, valid email format, and password constraints.

Data entry screens are designed in such a way that the system interacts effectively with the user. Fields are logically arranged to make the process simple and user-friendly.

The design converts user-provided inputs into a format suitable for processing by the system. The main goal of input design is to make data entry easy, logical, and error-free. Errors in input data are minimized through proper validation techniques.

The application is designed to be user-friendly. Input forms are structured so that users can easily enter data. If incorrect data is entered, appropriate error messages are displayed to prevent incorrect

processing.

The decisions made during input design are:

1. To achieve a high level of accuracy
2. To provide guidance and validation for important input fields

Output Design:

Output refers to the results and information generated by the system. In this application, outputs include blog posts, user profiles, comments, and search results.

Outputs are designed to communicate the results of processing clearly to the users. Efficient output design improves readability and usability.

The system provides different types of outputs:

• Internal Outputs: Blog data displayed within the application

• Interactive Outputs: Real-time updates such as comments, search results, and post updates

Outputs are dynamically updated using React.js without reloading the page, improving user experience.

4.4 UML DIAGRAMS

The logical structure of the system can be represented using UML diagrams. These diagrams help in understanding the design and workflow of the system.

The main components of UML diagrams are:

• Rectangles: Represent entity sets (User, Blog, Comment)

• Ellipses: Represent attributes

• Diamonds: Represent relationships

• Lines: Connect attributes with entities and define relationships

These diagrams provide a clear visualization of system structure and data flow.

4.5 BUILDING BLOCKS OF UML

The UML consists of three main building blocks:

1. Things
2. Relationships
3. Diagrams

Things in UML

Things are the basic elements used in modeling. There are four types:

1. Structural things (classes like User, Blog)
2. Behavioral things (operations like login, create post)
3. Grouping things (packages/modules)
4. Annotational things (notes/comments)

Relationships in UML

Relationships show how elements are connected in the system:

1. Dependency
2. Association
3. Generalization
4. Realization

Diagrams in UML

UML provides different types of diagrams to represent the system:

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram
6. Activity diagram
7. Component diagram
8. State chart diagram
9. Deployment diagram

USE CASE DIAGRAM

The use case diagram represents the interaction between users and the system. It shows actors (users) and their actions such as registration, login, creating blog posts, editing posts, deleting posts, and commenting.

Use case diagrams help in understanding the functional requirements of the system. They are important for modeling user behavior and system interactions.

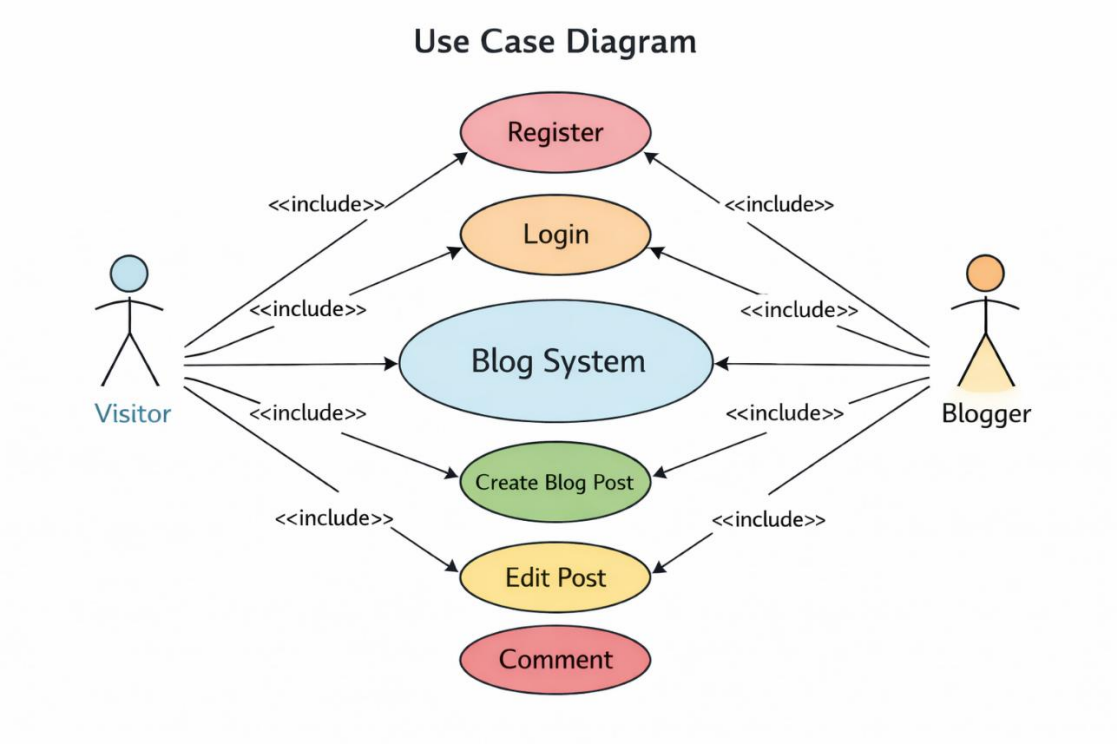


Fig. 4.42: Use-case Diagram

CLASS DIAGRAM:

Class diagrams are the most commonly used diagrams in UML. A class diagram consists of classes, interfaces, associations, and relationships. It mainly represents the object-oriented view of a system, which is static in nature.

In the Full Stack Blog Application, the class diagram includes main classes such as User, Blog, Comment, and Category. Each class contains attributes and methods that define its behavior and structure. For example, the User class includes details like username, email, and password, while the Blog class includes title, content, and author information.

Class diagrams show how different classes are connected through relationships such as association and inheritance. These relationships help in understanding how data flows within the system and how different components interact with each other.

Active classes can also be used in the diagram to represent processes that run simultaneously, such as handling multiple user requests.

The class diagram represents the object-oriented structure of the system and is mainly used during the development phase. It helps developers to design and organize the system efficiently. It is one of the most widely used diagrams during system construction and plays a key role in implementing the application.

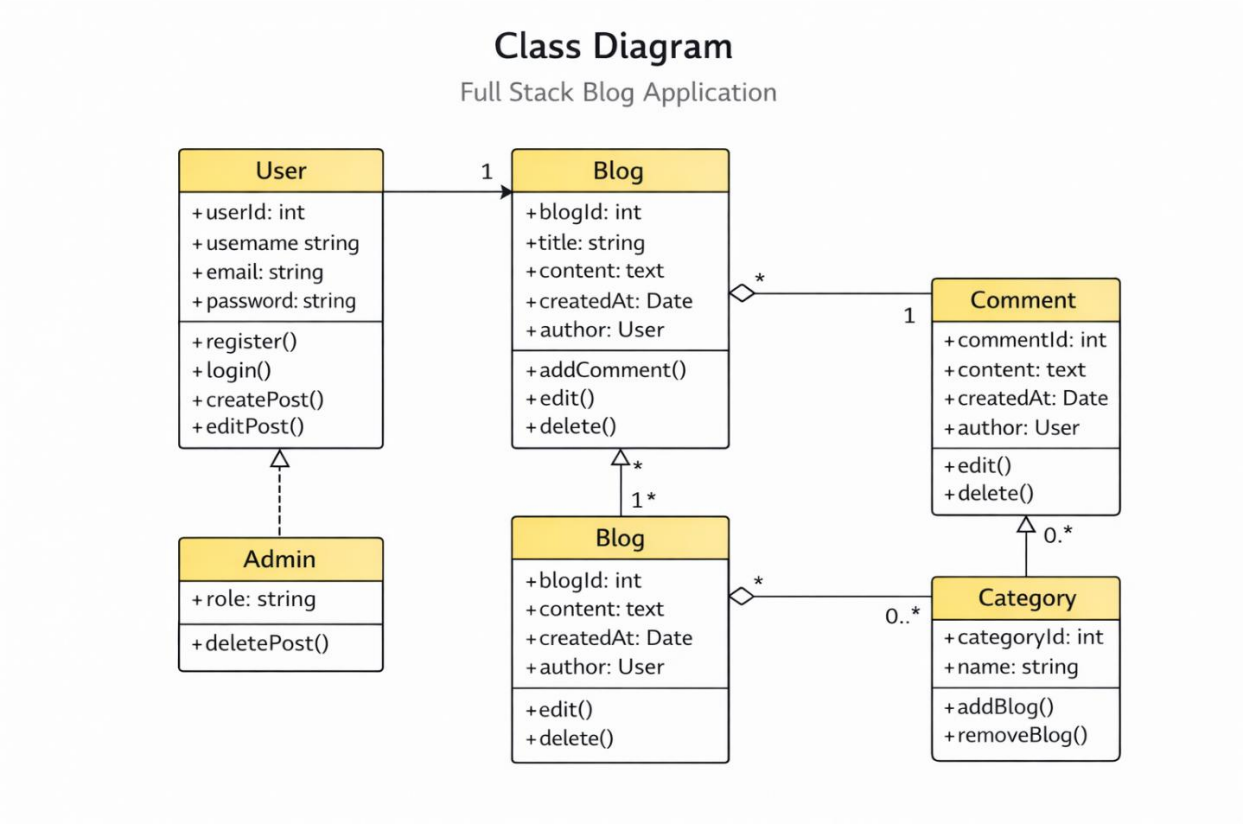


Fig. 4.43: Class Diagram

SEQUENCE DIAGRAM

A sequence diagram in Unified Modeling Language (UML) is a type of interaction diagram that shows how different components of a system interact with each other and in what order the operations are performed. It represents the flow of messages between objects over time.

In the Full Stack Blog Application, the sequence diagram illustrates the interaction between the User, Frontend (React.js), Backend (Node.js & Express.js), and Database (MongoDB). It shows how a request is initiated by the user and how it is processed step-by-step through different layers of the system.

For example, when a user logs in, the request is sent from the frontend to the backend server. The backend validates the user credentials and communicates with the database to verify the data. After successful validation, a response is sent back to the frontend, and the user is granted access.

Similarly, operations like creating a blog post, editing content, or adding comments follow a sequence of interactions between the client, server, and database.

Sequence diagrams help in understanding the dynamic behavior of the system. They clearly show the order of execution and message flow between components, making them useful during system design and development. These diagrams are also referred to as event diagrams, event scenarios, or timing diagrams.

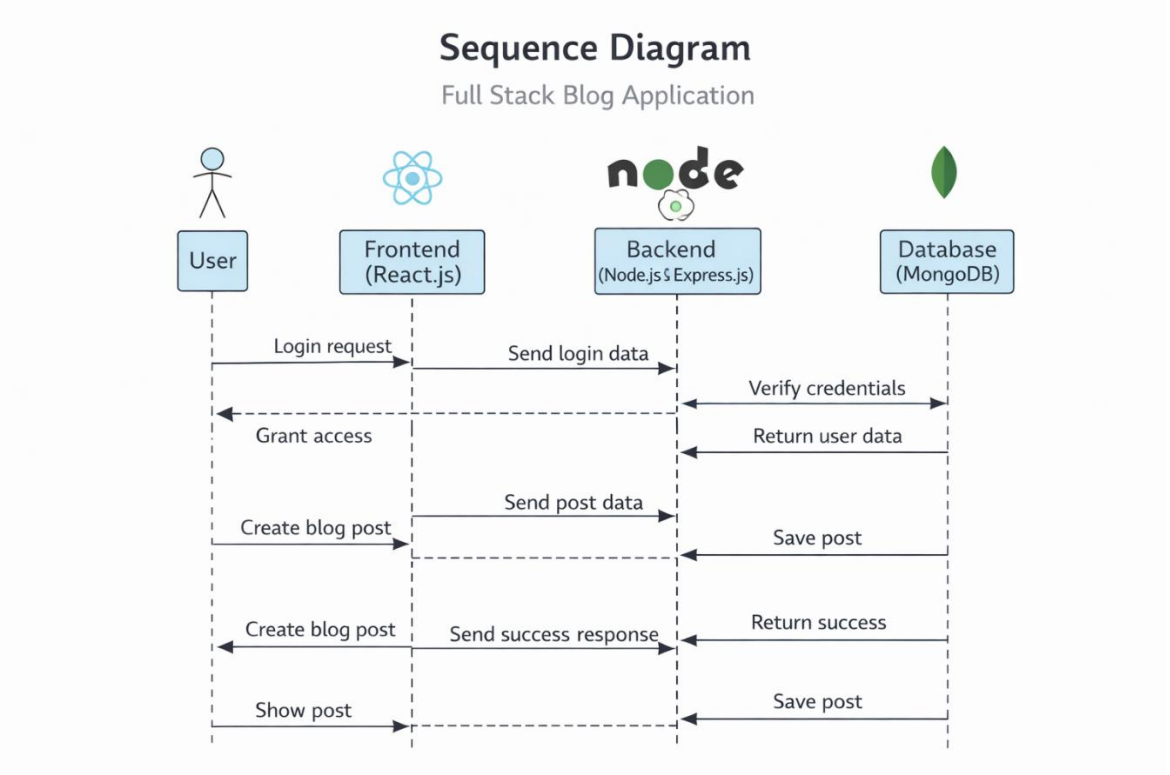


Fig. 4.4.4: Sequence Diagram

ACTIVITY DIAGRAM

Activity diagrams are graphical representations that show the workflow of step-by-step activities in a system. They represent the overall flow of control, including actions, decisions, and processes.

In the Full Stack Blog Application, the activity diagram shows processes like user registration, login, creating blog posts, editing content, and commenting. It helps in understanding how the system flows from one activity to another.

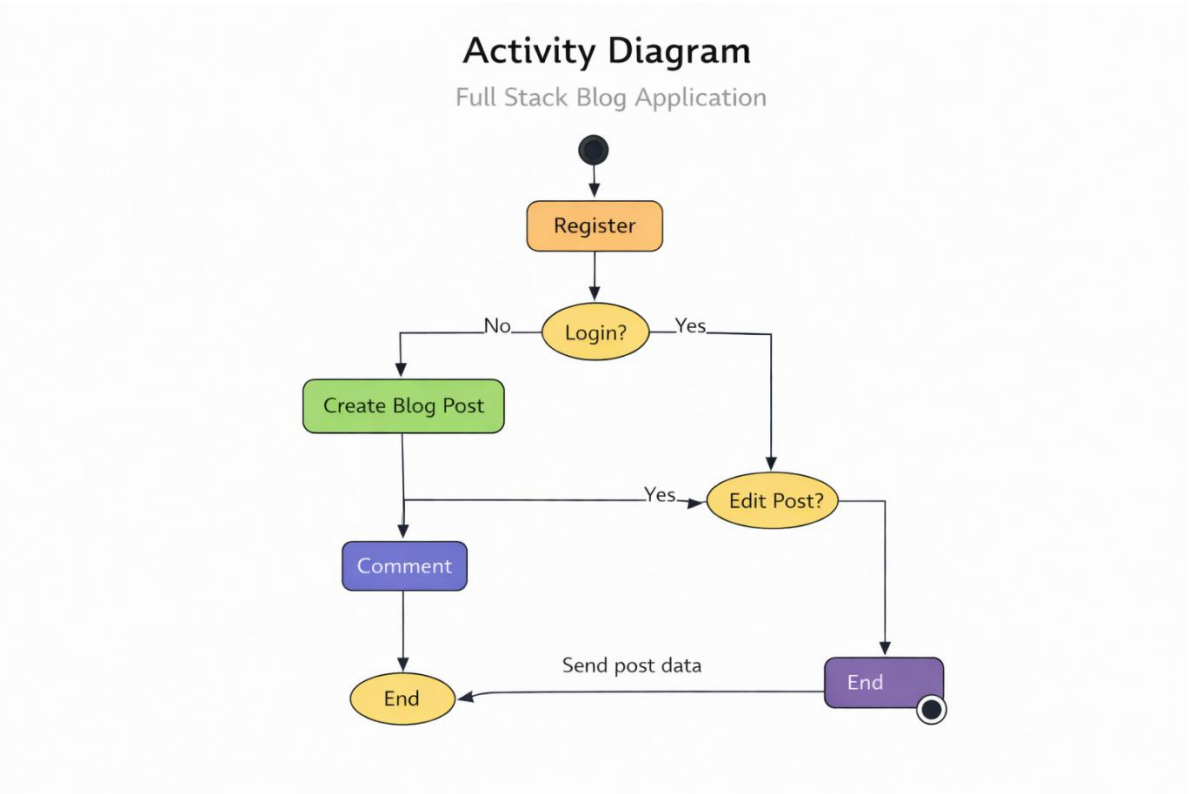


Fig. 4.4.5: Activity Diagram

CHAPTER-5

METHODOLOGY

5.1 MODULE DESCRIPTION:

Service Provider (Admin Module)

• In this module, the service provider (admin) manages the overall system operations. After logging in with valid credentials, the admin can monitor user activities, manage blog posts, and maintain system data.

• The admin can view all user interactions, manage content, and ensure proper functioning of the application.

Blog Management Module

• In this module, the user can create, edit, and delete blog posts through the React.js interface.

• The system validates the input data and sends it to the backend server using APIs.

• The backend processes the data and stores it in the MongoDB database.

Remote User (User Module)

• In this module, users can access the system through a web interface.

• Users must register before performing any operations. After registration, they can log in using valid credentials.

• After successful login, users can create posts, edit content, delete posts, and interact with other users through comments.

Comment Module

• In this module, users can add comments to blog posts.

• Comments are sent to the backend and stored in the database.

• This module improves user interaction and engagement.

Result Display Module (Content Display Module)

• In this module, blog posts, user profiles, and comments are displayed to the user.
• The content is fetched from the backend and rendered dynamically on the frontend.
• The output is presented in a clear and user-friendly format using React.js components.

5.2 ALGORITHM

Blog Creation Process

• The user logs in to the system.
• The user enters blog details such as title and content.
• The data is sent to the backend server using an API request.
• The backend validates the data and stores it in MongoDB.
• The stored data is retrieved and displayed on the frontend.

User Authentication Process

• The user enters login credentials.
• The frontend sends the data to the backend API.
• The backend verifies the credentials from the database.
• If valid, a JWT token is generated and sent to the user.
• The user is authenticated and allowed access to protected features.

API Integration

• The system uses RESTful APIs to communicate between frontend and backend.
• Backend APIs handle operations like user authentication, blog management, and comments.
• Data is exchanged in JSON format and displayed dynamically on the user interface.

5.3 TEST CASES

TC. No	Test Case	Input	Expected Output	Observed Output	Result
1	Login	Enter wrong username and password	Invalid login details	Invalid username or password	Pass
2	Login	Enter valid username and password	Login successful	Login successful	Pass
3	Registration	Enter invalid email format	Display error message	Email format validation message shown	Pass
4	Blog Creation	Submit empty blog fields	Show validation error	Error message displayed	Pass
5	Blog Creation	Enter valid blog details	Blog created successfully	Blog created successfully	Pass
6	Edit Blog	Update blog content	Blog updated successfully	Blog updated successfully	Pass
7	Delete Blog	Delete existing blog post	Blog deleted successfully	Blog deleted successfully	Pass
8	Comment	Add comment to blog	Comment added successfully	Comment added successfully	Pass
9	Search	Enter keyword in search bar	Display relevant blog posts	Relevant blogs displayed	Pass
10	Logout	Click logout	User logged out	User logged out	Pass

CHAPTER-6

RESULTS & DISSCUSSIONS

RESULTS & DISCUSSIONS

```
from django.shortcuts import render, redirect
from django.db.models import Avg, Q
from django.http import HttpResponse
import pandas as pd
import xlwt

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier

from Remote_User.models import ClientRegister_Model, predict_safety, detection_ratio,
detection_accuracy


# Service Provider Login
def serviceproviderlogin(request):
    if request.method == "POST":
        admin = request.POST.get('username')
        password = request.POST.get('password')

        if admin == "Admin" and password == "Admin":
            detection_accuracy.objects.all().delete()
            return redirect('View_Remote_Users')

    return render(request, 'SProvider/serviceproviderlogin.html')


# View Detection Ratio
def View_Prediction_Of_Patient_Safety_Detection_Ratio(request):
    detection_ratio.objects.all().delete()

    total = predict_safety.objects.all().count()

    safe_count = predict_safety.objects.filter(Q(Prediction='Safety')).count()
    unsafe_count = predict_safety.objects.filter(Q(Prediction='No Safety')).count()

    if total > 0:
        detection_ratio.objects.create(names='Safety', ratio=(safe_count / total) * 100)
        detection_ratio.objects.create(names='No Safety', ratio=(unsafe_count / total) * 100)

    obj = detection_ratio.objects.all()
    return render(request, 'SProvider/View_Prediction.html', {'objs': obj})


# View Users
def View_Remote_Users(request):
    obj = ClientRegister_Model.objects.all()
```

```
    return render(request, 'SProvider/View_Users.html', {'objects': obj})


# Download Dataset
def Download_Predicted_DataSets(request):
    response = HttpResponse(content_type='application/ms-excel')
    response['Content-Disposition'] = 'attachment; filename="Predicted_Data.xls"'

    wb = xlwt.Workbook()
    ws = wb.add_sheet("Sheet1")

    rows = predict_safety.objects.all()

    for i, row in enumerate(rows):
        ws.write(i, 0, row.Fid)
        ws.write(i, 1, row.Drug1_Name)
        ws.write(i, 2, row.Drug1_Condition)
        ws.write(i, 3, row.Drug2_Name)
        ws.write(i, 4, row.Drug2_Condition)
        ws.write(i, 5, row.Patient_Gender)
        ws.write(i, 6, row.Patient_Age)
        ws.write(i, 7, row.Area)
        ws.write(i, 8, row.Prediction)

    wb.save(response)
    return response


# Train Model
def train_model(request):
    detection_accuracy.objects.all().delete()

    df = pd.read_csv('Datasets.csv')

    df['results'] = df['Safety_Label'].apply(lambda x: 0 if x == 0 else 1)

    X = df['Fid']
    y = df['results']

    cv = CountVectorizer()
    X = cv.fit_transform(X.astype(str))

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

    # DNN
    mlp = MLPClassifier().fit(X_train, y_train)
    y_pred = mlp.predict(X_test)
    acc = accuracy_score(y_test, y_pred) * 100
    detection_accuracy.objects.create(names="DNN", ratio=acc)

    # SVM
    svm_model = svm.LinearSVC().fit(X_train, y_train)
    y_pred = svm_model.predict(X_test)
```

```
acc = accuracy_score(y_test, y_pred) * 100
detection_accuracy.objects.create(names="SVM", ratio=acc)

# Logistic Regression
lr = LogisticRegression().fit(X_train, y_train)
y_pred = lr.predict(X_test)
acc = accuracy_score(y_test, y_pred) * 100
detection_accuracy.objects.create(names="Logistic Regression", ratio=acc)

# Gradient Boosting
gb = GradientBoostingClassifier().fit(X_train, y_train)
y_pred = gb.predict(X_test)
acc = accuracy_score(y_test, y_pred) * 100
detection_accuracy.objects.create(names="Gradient Boosting", ratio=acc)

results = detection_accuracy.objects.all()
return render(request, 'SProvider/train_model.html', {'objs': results})
```

CHAPTER-7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The Full Stack Blog Application was successfully developed using the MERN stack, which includes MongoDB, Express.js, React.js, and Node.js. The project demonstrates how modern web technologies can be used together to build a dynamic and interactive blogging platform.

The application allows users to register, log in, create, edit, and delete blog posts efficiently. Additional features such as commenting, search functionality, and category-based filtering enhance user interaction and improve content accessibility. The use of React.js ensures a responsive and user-friendly interface, while Node.js and Express.js handle backend operations effectively.

Authentication using JSON Web Tokens (JWT) provides secure access to user-specific features. MongoDB is used for efficient data storage and retrieval, making the system scalable and flexible.

The results show that the application performs efficiently with smooth communication between frontend and backend. All modules function correctly and meet both functional and non-functional requirements.

Overall, the project provides a complete full-stack solution and demonstrates practical knowledge of modern web development technologies..

7.2 FUTURE ENHANCEMENT

The current system can be further improved by adding more advanced features in the future. Some possible enhancements include:

- Implementing **like and share functionality** for blog posts
- Adding **real-time notifications** for comments and updates
- Integrating **image and video uploads** in blog posts
- Implementing **role-based access control** (admin, user)
- Adding **AI-based content recommendation system**
- Improving search using **advanced filtering and tagging system**
- Deploying the application on cloud platforms for better scalability

8. REFERENCES

[1] MongoDB Documentation – https://www.mongodb.com/docs/

[2] React.js Documentation – https://reactjs.org/docs/

[3] Node.js Documentation – https://nodejs.org/en/docs/

[4] Express.js Guide – https://expressjs.com/

[5] JWT Authentication – https://jwt.io/introduction/

[6] Tailwind CSS Documentation – https://tailwindcss.com/docs