COLLEGE CODE:8203

COLLEGE NAME:A.V.C College of Engineering

DEPARTMENT:Computer Science And Engineering

STUDENTNM-ID: 214A8A7E25B0439839A48AAEA8B33149

ROLL NO:820323104102

DATE:15-09-2025

Completed the project named as Phase2 TECHNOLOGY PROJECT NAME :Blogging Platform using Node.js, Express, MongoDB, and JWT.

SUBMITTED BY,

NAME: R.Sriram

MOBILE NO: 90254 77195

Phase 2 — Solution Design & Architecture:

1. **Introduction**:

Phase 2 focuses on converting the problem understanding and requirements identified in Phase 1 intoa concrete solution. This includes selecting the technology stack, designing the system architecture, data schemas, user interface components, and defining the flow of information between the frontend, backend, and database. The goal is to ensure that the application is scalable, maintainable, and secure while being easy to use for bloggers and readers.

—

2. **Technology Stack Selection:**

Selecting the right tech stack is crucial to building a performant and reliable application. For this blogging platform, a modern MERN-like (MongoDB, Express, React, Node) stack has been selected for both speed of development and scalability.

2.1 Frontend:

- Framework: React.js
- Additional Technologies: HTML5, CSS3, JavaScript (ES6+)
- Justification:
    - React provides a component-based architecture, making the UI modular and maintainable.
    - Fast rendering via virtual DOM and extensive ecosystem support.

2.2 Backend:

- Runtime: Node.js
- Framework: Express.js
- Justification:
    - Node.js is asynchronous and lightweight—suitable for RESTful APIs.
    - Express.js offers routing, middleware, and easy integration with MongoDB.

2.3 Database:

- Database: MongoDB (NoSQL)
- ORM/ODM: Mongoose
- Justification:
    - MongoDB is document-based, making it ideal for handling blogs, users, and comments as nested documents or references.
    - Mongoose allows schema validation and model management.

2.4 Authentication:

- Strategy: JSON Web Tokens (JWT)
- Justification:
    - Stateless, scalable user authentication system.
    - No need for server-side sessions—tokens carry user identity.

—

3. UI Structure and Page Design:

The user interface is divided into logical pages/components based on user workflows. React allows component-based modularity, which helps in the maintainability of the system.

3.1 UI Pages:

- Login / Register Page:
  - Allows users to authenticate and gain access.
  - Displays error messages if authentication fails.
- Dashboard:
  - Displays list of user's blogs and provides access to create, update, or delete blogs.
  - For readers, it shows all blogs from all authors.
- Blog Editor:
  - Rich text form to create or update blog content.
  - Input fields for title, content.
- Blog Detail View:
  - Shows full blog content.
  - Displays user comments and allows new comment submissions.

Each of these components interacts with the backend APIs using Axios or Fetch API for data retrieval and submission.

—

4. API Schema Design (Mongoose Models):

The backend uses Mongoose for defining schemas and models. Below are the main entities involved in the system:

4.1 User Schema:

```
{
username: String,
email: String,
password: String (hashed)
}
```

- Validations:
  - Unique email and username.
  - Password is hashed using bcrypt before storage.

4.2 Blog Schema:

```
{
title: String,
content: String,
author: ObjectId (ref: 'User'),
```

createdAt: Date,
comments: [Comment]
}

- A blog is linked to a specific user via the author field.
- Embedded or referenced comments are stored for performance and modularity.

4.3 Comment Schema:

{
user: ObjectId (ref: 'User'),
text: String,
createdAt: Date
}

- Each comment is tied to a user for attribution and moderation.
- Comments are stored in relation to a blog post.

These schemas are designed with reference relationships and timestamps to ensure data integrity and traceability.

—

5. **Data Handling & Security Practices:**

5.1 Input Validation:

- Middleware is used to validate all incoming data (e.g., express-validator).
- Protects against malformed inputs and injection attacks.

5.2 Password Security:

- All passwords are hashed using bcrypt before being stored in the database.
- Salting is used to enhance security.

5.3 Authentication with JWT:

- On login/register, server issues a signed JWT.
- JWT contains user ID and expiration time.
- Token is required in Authorization headers for protected routes.
- Middleware on backend verifies the JWT on every request.

5.4 Data Storage:

- MongoDB is used for storing all persistent data.
- Indexing is used for optimizing query performance on large datasets (e.g., blogs sorted by date).

—

6. **Component & Module Diagram:**

The architecture is split into frontend and backend modules for separation of concerns.

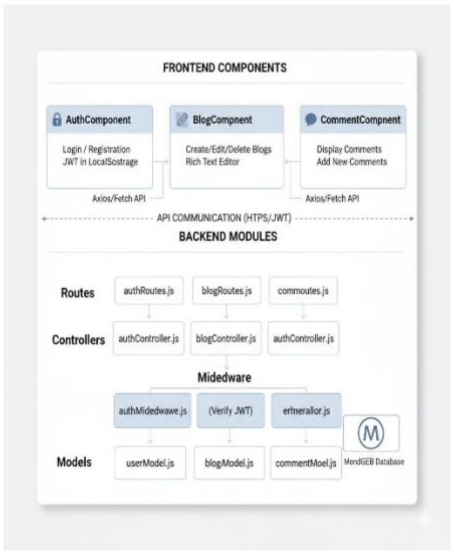
6.1 Frontend Components:

- AuthComponent:
  - Handles login and registration.
  - Manages user session with JWT in localStorage.
- BlogComponent:
  - Handles creation, editing, and deletion of blogs.
  - Uses a rich text editor (like React Quill or Draft.js).
- CommentComponent:
  - Displays existing comments.
  - Allows adding new comments.

6.2 Backend Modules:

- Routes:
  - authRoutes.js – handles login/register.
  - blogRoutes.js – handles blog CRUD operations.
  - commentRoutes.js – handles comment addition.
- Controllers:
  - authController.js, blogController.js, commentController.js – handle business logic.
- Middleware:
  - authMiddleware.js – verifies JWT and attaches user info.
  - errorHandler.js – handles error responses.
- Models:
  - userModel.js, blogModel.js, commentModel.js

This structure promotes separation of concerns, easier testing, and long-term maintainability.



—

7. **Basic Flow Diagram (Described in Text):**

Below is a simplified flow of data through the application:

1. User Authentication:
   - User submits login/register form → API validates → JWT is issued → Stored in localStorage → Used in future API calls.
2. Blog Creation:
   - Authenticated user submits blog form → Blog data sent via POST → Stored in MongoDB → Blog appears on dashboard and global list.
3. Blog Viewing:
   - Reader accesses /blogs → GET request to API → Server returns blog list → Blogs displayed on UI.
4. Adding Comments:
   - Authenticated user types comment → Comment submitted via POST → API validates → Stored under the relevant blog → Displayed in real-time.

—

8. **Conclusion**:

This phase ensures that the platform is not only functionally robust but also scalable, secure, and developer-friendly. With a clearly defined tech stack, UI structure, schema design, and module separation, the system is ready for actual development and testing in the next phases.

By leveraging React for the frontend, Node.js/Express for the backend, MongoDB for flexible data storage, and JWT for authentication, the blogging platform achieves a clean separation of concerns and prepares for scalable deployment. The proposed design is modular, secure, and aligns with modern web development best practices.