COLLEGE CODE:8203

COLLEGE NAME:A.V.C College of Engineering

DEPARTMENT:ComputerScienceAndEngineering

STUDENTNMID: 2774A1DBA16809C8C764F7BFA755494C

ROLL NO:820323104091

DATE:17-10-2025

Completed the project named Phase 5 TECHNOLOGY

PROJECT NAME :Blogging Platform using Node.js, Express, MongoDB, and JWT

SUBMITTED BY,

NAME: G.Santhosh

MOBILENO: 6369502254

PHASE 5 – PROJECT DEMONSTRATION & DOCUMENTATION

Project Title: Blogging Platform

Technology Stack: Node.js, Express.js, MongoDB, HTML, CSS, JavaScript

Phase: Final Demonstration & Submission

Final Demo Walkthrough:

Overview:

The Blogging Platform is a full-stack web application designed to enable users to create, read, update, and delete blog posts. The platform allows secure user authentication, interactive UI, and real-time post management.

Users can register, log in, and manage their blogs efficiently. Admin users can monitor all posts and moderate inappropriate content.

Key Features:

1. User Authentication: Secure login and registration using JWT (JSON Web Tokens).
2. Create/Edit/Delete Posts: Users can manage their content easily.
3. Comment System: Allows readers to share feedback.
4. Responsive Design: Works seamlessly on desktops, tablets, and mobiles.
5. Search & Filter: Users can search blogs by title, author, or tags.
6. RESTful APIs: Structured endpoints for frontend-backend interaction.
7. Cloud Deployment: Hosted on Render/Glitch/Vercel with MongoDB Atlas.

Project Report:

Introduction:

The purpose of the Blogging Platform is to provide a simple yet scalable content management tool. It helps writers, students, and content creators share thoughts online without technical barriers.

The system was built using Node.js for the backend and MongoDB for the database, ensuring high performance and flexibility.

Objectives:

1. Build a functional blogging system with CRUD operations.
2. Implement secure authentication and session handling.
3. Allow users to upload and manage posts with timestamps.
4. Provide a clean, user-friendly interface.
5. Deploy a stable version on the cloud.

System Architecture:

1. Architecture Type: MVC (Model–View–Controller)
2. Frontend (View): HTML, CSS, JavaScript (EJS templates).
3. Backend (Controller): Node.js + Express.js.
4. Database (Model): MongoDB with Mongoose ODM.

Data Flow:

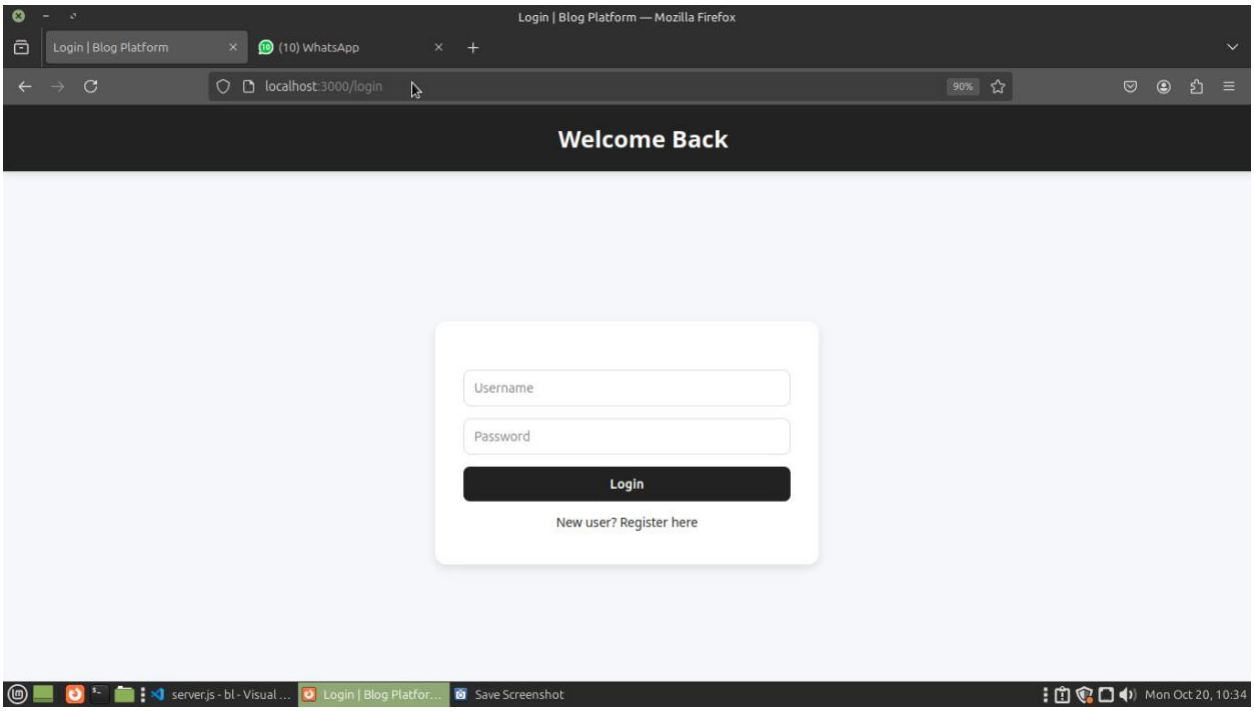
1. User interacts with UI →
2. Request hits Express route →
3. Controller processes →
4. MongoDB handles data →
5. Response rendered to user.

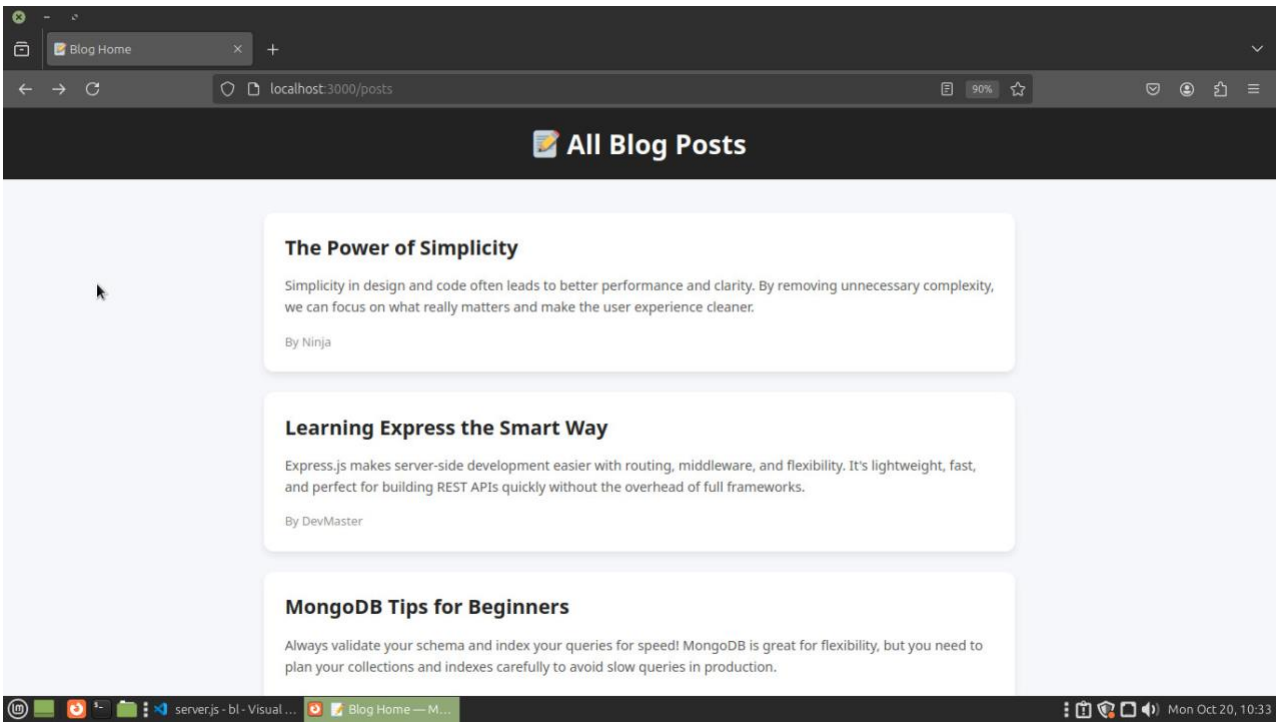
Screenshots / API Documentation:

Screenshots:

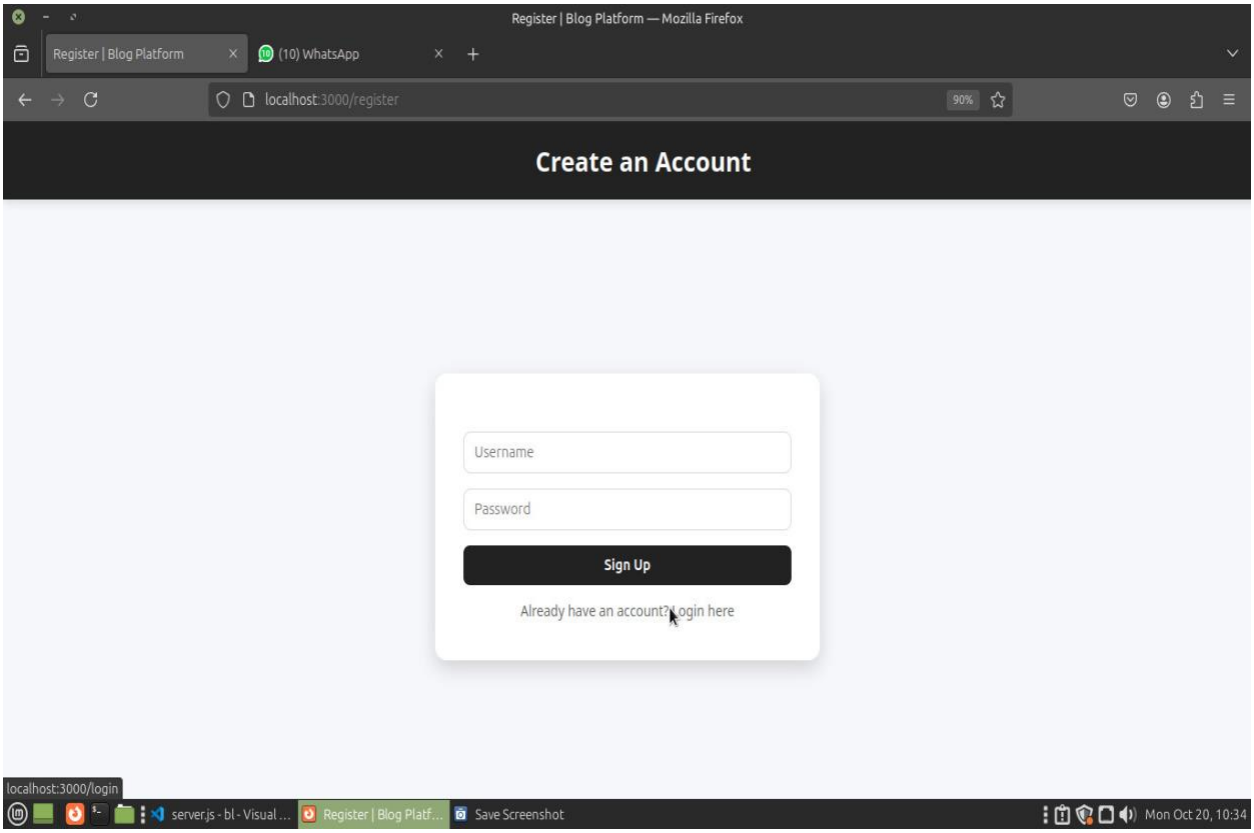
1. Login Page – User enters credentials.
2. Dashboard – Displays user's posts.
3. Create Blog Page – Form to add a new post.
4. View Blog Page – Shows content with comments.
5. Admin Panel – For managing users/posts.

(Insert screenshots of your running application here)





re.)



API Endpoints:

1. POST /api/auth/register Register new user
2. POST /api/auth/login Login and receive JWT
3. GET /api/posts Fetch all posts
4. GET /api/posts/:id Fetch specific post
5. POST /api/posts Create new post
6. PUT /api/posts/:id Update post
7. DELETE /api/posts/:id Delete post
8. POST /api/comments/:postId Add comment

Authentication:

Protected routes use JWT middleware for verification.

Challenges & Solutions:

Challenge:	Description:	Solution Implemented:
1. Authentication Security-	Managing user sessions securely-	Used JWT for token-based authentication
2. Database Connectivity-	Occasional MongoDB Atlas connection timeout-	Implemented retry logic and error handling
3. UI Responsiveness-	Pages broke on smaller devices-	Added CSS Flexbox and media queries
4. API Testing-	Difficulty in debugging REST APIs-	Used Postman for endpoint validation
5. Deployment-	Issues with environment variables-	Used .env file and configured secrets in Render/Vercel

Code Example:

Dashboard .js:

```
// Public/js/dashboard.js


Document.addEventListener(“DOMContentLoaded”, () => {
 Const form = document.querySelector(“#newPostForm”);
 Const postsContainer = document.querySelector(“#userPosts”);


 If (form) {
  Form.addEventListener(“submit”, async € => {
   e.preventDefault();


   const title = form.querySelector(“input[name=’title’]”).value.trim();
   const content = form.querySelector(“textarea[name=’content’]”).value.trim();
```

```
if (!title || !content) {
  alert(“Please fill in all fields!”);
  return;
}

Try {
  Const res = await fetch(“/posts/create”, {
    Method: “POST”,
    Headers: { “Content-Type”: “application/json” },
    Body: JSON.stringify({ title, content }),
  });

  If (res.ok) {
    Const newPost = await res.json();

    // Append new post to dashboard dynamically
    Const div = document.createElement(“div”);
    div.classList.add(“post”);
    div.innerHTML = `<h3>${newPost.title}</h3><p>${newPost.content}</p>`;
    postsContainer.prepend(div);

    // Reset form
    Form.reset();
  } else {
    Alert(“Error creating post”);
```

```
        }
      } catch (err) {
        Console.error(err);
        Alert("Failed to create post");
      }
    });
  }
});
```

Main.js:

```
    //   Public/js/main.js

Document.addEventListener("DOMContentLoaded", () => {
 // Show flash messages and auto-hide
 Const flashMsg = document.querySelectorAll('.flash-message');
 flashMsg.forEach(msg => {
  setTimeout(() => {
   msg.style.display = 'none';
  }, 3000);
 });

 // Basic form validation for login/register
 Const forms = document.querySelectorAll('form');
 Forms.forEach(form => {
  Form.addEventListener('submit', e => {
   Const inputs = form.querySelectorAll('input[required], textarea[required]');
   For (let input of inputs) {
```

```
      If (!input.value.trim()) {
        Alert(` ${input.name || ‘Field’} cannot be empty` );
        e.preventDefault();
        return;
      }
    }
  });
});
```

Css:

```
 body {
 font-family: Arial, sans-serif;
 padding: 20px;
 background-color: #f2f2f2;
}

h1, h2 {
 color: #333;
}

.post {
 background: white;
 padding: 15px;
 margin: 10px 0;
 border-radius: 10px;
 box-shadow: 0 2px 5px rgba(0,0,0,0.1);
```

```
}

input, textarea, button {
  width: 100%;
  padding: 8px;
  margin: 5px 0;
}
```

Package.json:

```
{
  "name": "blogging-platform",
  "version": "1.0.0",
  "description": "Simple blogging platform using Node.js, Express, MongoDB, and EJS",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  },
  "dependencies": {
    "bcrypt": "^5.1.0",
    "body-parser": "^1.20.2",
    "connect-flash": "^0.1.1",
    "dotenv": "^16.4.0",
    "ejs": "^3.1.9",
    "express": "^4.18.2",
    "express-session": "^1.17.3",
    "mongoose": "^8.0.0",
```

```
    "passport": "^0.7.0",
    "passport-local": "^1.0.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.10"
  }
```

Server.js:

```
Const express = require('express');
Const mongoose = require('mongoose');
Const session = require('express-session');
Const flash = require('connect-flash');
Const passport = require('passport');
Const path = require('path');
Require('dotenv').config();

Const dashboardRouter = require('./routes/dashboardRoutes'); // ✅ use require

Const app = express();

// Middleware setup
App.use(express.urlencoded({ extended: true }));
App.use(express.json());

App.use(express.static(path.join(__dirname, 'public')));
App.set('view engine', 'ejs');
App.set('views', path.join(__dirname, 'views'));
```

```
// Session setup
App.use(
  Session({
   Secret: 'blogsecretkey',
   Resave: false,
   saveUninitialized: true,
  })
);
App.use(flash());

// Passport config
Require('./controllers/authController')(passport);
App.use(passport.initialize());
App.use(passport.session());

// Flash messages middleware
App.use((req, res, next) => {
  Res.locals.success_msg = req.flash('success_msg');
  Res.locals.error_msg = req.flash('error_msg');
  Res.locals.error = req.flash('error');
  Next();
});

// ✅ Routes
Const authRoutes = require('./routes/authRoutes');
```

Const postRoutes = require('./routes/postRoutes');

App.use('/', authRoutes);

App.use('/posts', postRoutes);

App.use('/dashboard', dashboardRouter); // ✅ moved here after middleware

// MongoDB connection

Mongoose

.connect(process.env.MONGO_URI || 'mongodb://127.0.0.1:27017/blogging-platform', {})

.then(() => console.log('✅ MongoDB connected'))

.catch(err => console.log(err));

// Start server

Const PORT = process.env.PORT || 5000;

App.listen(PORT, () => console.log(` 🚀 Server running on port ${PORT}`));

}**GitHub README & Setup Guide:**

Installation Instructions:

Requirements:

1. Node.js (v18 or higher)
2. MongoDB Atlas Account
3. Npm package manager

Steps to Run Locally:

1. Clone the repository:

Git clone https://github.com/santhosh120506/blogging-platform-nodejs.git

2. Navigate into project directory

Cd blogging-platform-nodejs

3. Install dependencies

Npm install

4. Create .env file and add:

MONGO_URI=your_mongodb_connection_string

JWT_SECRET=your_secret_key

PORT=5000

5. Start the server

Npm start

Access:

Open browser → http://localhost:3000/posts

Folder Structure:

```
Blogging-platform/
├── server.js
├── package.json
├── routes/
│   ├── authRoutes.js
│   └── postRoutes.js
├── controllers/
│   ├── authController.js
│   └── postController.js
├── models/
│   ├── User.js
│   └── Post.js
```

```
├── views/
│   ├── home.ejs
│   ├── login.ejs
│   ├── dashboard.ejs
└── public/
    ├── css/
    └── js/
```

Deployment Guide:

1. Push code to GitHub repository.
2. Connect GitHub repo to Render/Vercel.
3. Configure environment variables (Mongo URI, JWT).
4. Deploy and test live URL.

Deployed Link:

👉 https://santhosh120506.github.io/blogging-platform-nodejs/

Conclusion:

The Blogging Platform successfully demonstrates CRUD operations, authentication, and deployment practices using the Node.js environment. It provides an easy-to-use, scalable, and responsive solution for content management.

1. Future Enhancements
2. Integrate image uploads for posts.
3. Add role-based access (Admin, Editor, Reader).
4. Include email notifications for comments.
5. Implement analytics dashboard for authors.