Bahria University,

Karachi Campus



COURSE: SOFTWARE CONSTRUCTION

TERM: FALL 2024, CLASS: BSE- 5 (A/B)

PROJECT NAME:

BLOGGGING APPLICATION

Engr.Misbah Perveen/Dr. Salahuddin Sheikh / Engr. Hamza

Signed Remarks: Score:

1. **TABLE OF CONTENT**

Contents

[**1.** **INTRODUCTION & PROBLEM:** 3](#_Toc151222627)

[**2.** **TECHNOLOGY:** 3](#_Toc151222627)

[**3.** **PROJECT FUNCTIONALITIES:** 4](#_Toc151222627)

[**4.** **MODULE DISTRIBUTION:** 4](#_Toc151222629)

[**5.** **CODE:** 4](#_Toc151222630)

[**6.** **INTERFACES:** 7](#_Toc151222630)

[**7.** **CONCLUSION:** 10](#_Toc151222630)

1. **INTRODUCTION & PROBLEM**

**INTRODUCTION**:

In the dynamic landscape of web development, the integration of modern technologies has given rise to versatile applications catering to diverse needs. Our Blogging Application, built on the MERN (MongoDB, Express.js, React, Node.js) stack, represents a sophisticated platform designed to streamline content creation, publication, and interaction within a digital community.

**PROBLEM STATEMENT:**

Traditional blogging platforms often face limitations in terms of user experience, flexibility, and real-time engagement. This project addresses these challenges by leveraging the MERN stack to create a robust, feature-rich, and user-friendly blogging application. The goal is to overcome common pitfalls in existing systems, offering an intuitive interface, dynamic content management, and enhanced user interaction. This application seeks to redefine the blogging experience by combining the power of MERN technologies with innovative features for both content creators and readers.

1. **TECHNOLOGY**

The application leverages cutting-edge technologies:

* **Frontend**: React, Redux, Jodit, HTML React Parser
* **Backend**: Node.js, Express.js, MongoDB
* **Authentication**: JWT, bcrypt, Otp Generator, Validator
* **Serverside Libraries**: Node mailer, Formidable, Date-and-Time, JWT-Decode, React-Hot-Toaster, body-parser, cookie-parser , cors , dotenv , mongoose, email js, validator
* **Frontend Libraries**: React, React router dom, react-redux , redux-thunk , moment js , jodit text editor , react hot toast , react helmet , axios , react-apexcharts , react-html-parser , html-to-text

1. **FUNCTIONALITIES & FEATURES**

Blogify encompasses a wide array of functionalities, including:

* User Authentication and Authorization
* Admin Authentication and Authorization
* Admin create blog Category and Tag
* Admin handle all Sub-Admin (Bloggers) and Admin can see Profile of Sub-Admin and do block/unblock Sub-Admin
* Article Publishing, Editing, and Deletion
* Articles are in pending mode unless Admin do not accept and publish them
* User like, dislike blogs and write comment also.
* Search Blog System
* Admin Panel for Comprehensive Content Management
* Real-Time Interaction with Articles
* Multi-Level User Access
* Graphical Insights into User Engagement
* Security Measures with JWT Tokens, bcrypt, and OTP Generation

1. **MODULE DISTRIBUTION**

* **Muhammad Shoaib Akhter Qadri:**

He is working on Frontend, backend with database. He is responsible for working on frontend with backend with database (Mongo DB Atlas) and work on different libraries for authentication etc. He is working to run applications on server side and client side also.

* **Muhammad Yahya:**

He is working on Frontend, and he has responsible to make responsive application and create different pages like home, 404 page, admin and sub-admin panel with different functionalities and feature like category, tags, comments, user, sub-admin and article post pages.

1. **CODE**

const validator = require('validator');

const adminModel = require('../models/adminModel';

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const formidable = require('formidable')

const userModel = require('../models/userModel');

const nodeMiler = require('nodemailer')

const fs = require('fs')

module.exports.admin\_login = async (req, res) => {

const { email, password } = req.body;

const error = {}

if (email && !validator.isEmail(email)) {

error.email = "please provide your valid email"}

if (!email) {

error.email = "please provide your email";}

if (!password) {

error.password = "please provide your password"}

if (Object.keys(error).length > 0) {

return res.status(400).json({ errorMessage: error })

} else {try {

const getAdmin = await adminModel.findOne({ email }).select('+password');

if (getAdmin) {

const matchPassword = await bcrypt.compare(password, getAdmin.password);

if (matchPassword) {

const token = jwt.sign({

id: getAdmin.\_id,

name: getAdmin.adminName,

role: getAdmin.role,

image: getAdmin.image

}, process.env.SECRET, { expiresIn: '7d' });

res.status(200).cookie('blog\_token', token, {

expires: new Date(

Date.now() + process.env.COOKIE\_EXP \* 24 \* 60 \* 60 \* 1000),

httpOnly: true}).json({

successMessage: 'login successful',

token})} else {

return res.status(400).json({ errorMessage: { error: "Password does not match" }});}} else {

return res.status(400).json({ errorMessage: { error: "Email does not exits" } });}} catch (error) {

console.error(error);

return res.status(500).json({ errorMessage: { error: "Internal server error" } });}}}

module.exports.user\_register = async (req, res) => {

const formData = formidable()

formData.parse(req, async (err, fields, files) => {if (err) {

return res.status(500).json({ errorMessage: { error: "form data parse filed" } });} else {

const { name, email, password } = fields

const errorData = {}

if (!name) {

errorData.name = 'Please provide your name'}

if (!email) {

errorData.email = 'Please provide your email'}

if (email && !validator.isEmail(email)) {

errorData.email = 'Please provide your valid email'}

if (!password) {

errorData.password = 'Please provide your password'}

if (Object.keys(files).length === 0) {

errorData.image = 'Please provide your image'}

if (Object.keys(errorData).length === 0) {

try {

const getUser = await userModel.findOne({ email })

if (getUser) {

return res.status(500).json({ errorMessage: { error: "Your email already use" } });} else {

const otp = Math.floor(Math.random() \* 100000 + 345678)

const transporter = nodeMiler.createTransport({

service: 'Gmail',

auth: {

user: process.env.USER\_EMAIL,

pass: process.env.USER\_PASSWORD}})

const mailOption = {

from: process.env.USER\_EMAIL,

to: email,

subject: ' Sending email blog',

html: `<div style={padding:"10px"}>

<h3>Your otp code ${otp}</h3>

</div>`}

transporter.sendMail(mailOption, async (error) => {

if (error) {

return res.status(500).json({ errorMessage: { error: "Somethings else please try again" } });

} else {

const verifyEmailToken = jwt.sign({

email,

name,

password: await bcrypt.hash(password, 10),

imageInfo: files,

otpCode: otp

}, process.env.SECRET, {

expiresIn: process.env.TOKEN\_EXP})

const option = {

expires: new Date(Date.now() + 1 \* 24 \* 60 \* 60 \* 1000)}

res.status(201).cookie('emailVerifyToken', verifyEmailToken, option).json({ successMessage: "Check your email and submit otp" })}})}

} catch (error) {

return res.status(500).json({ errorMessage: { error: "Internal server error" } });}} else {

return res.status(400).json({ errorMessage: errorData })}}})}

module.exports.verify\_email = async (req, res) => {

const { otp } = req.body

const { emailVerifyToken } = req.cookies

if (!otp) {

res.status(404).json({ errorMessage: 'please provide your otp' })

} else {

const { name, email, password, otpCode, imageInfo } = await jwt.verify(emailVerifyToken, process.env.SECRET)

const imageName = Date.now()

+ imageInfo.image.originalFilename;

const disPath = \_\_dirname + `../../../frontend/public/userImage/${imageName}`

try {

if (parseInt(otp) !== otpCode) {

res.status(404).json({ errorMessage: { error: 'please provide your valid otp' } })} else {

fs.copyFile(imageInfo.image.filepath, disPath, async (err) => {

if (!err) {

const createUser = await userModel.create({

userName: name,

email,

loginMethod: 'manually',

password,

image: `http://localhost:3000/userImage/${imageName}`})

const token = jwt.sign({

id: createUser.\_id,

email: createUser.email,

name: createUser.userName,

image: createUser.image,

role: createUser.role,

loginMethod: createUser.loginMethod,

accessStatus: createUser.accessStatus,

createdAt: createUser.createdAt

}, process.env.SECRET, {

expiresIn: process.env.TOKEN\_EXP})

const option = {

expires: new Date(Date.now() + 3 \* 24 \* 60 \* 60 \* 1000)}

res.clearCookie('emailVerifyToken')

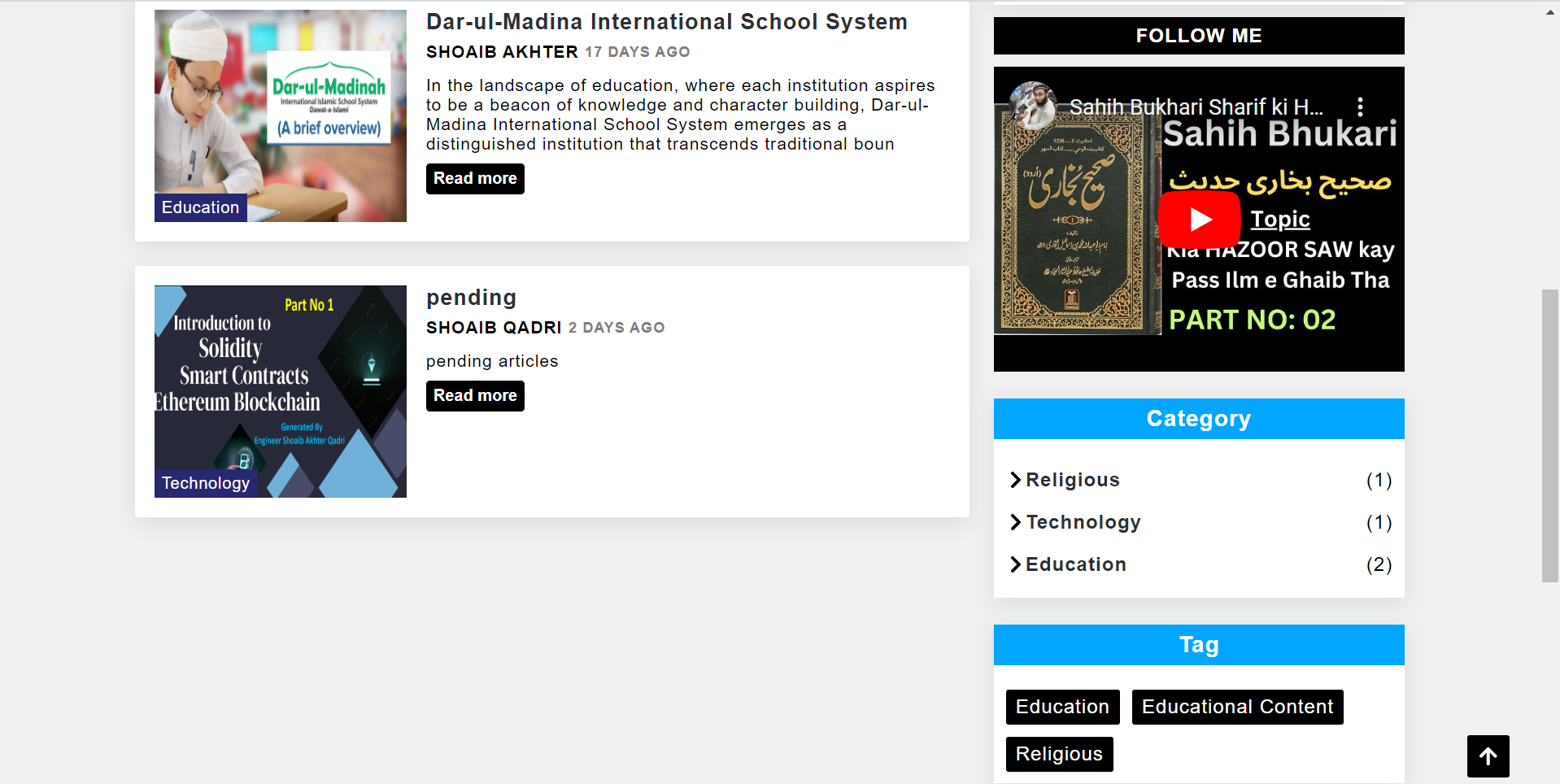
res.status(201).cookie('blog\_token', token, option).json({

successMessage: "Your register successful"

1. **INTERFACES**

**A screenshot of a computer

Description automatically generated**

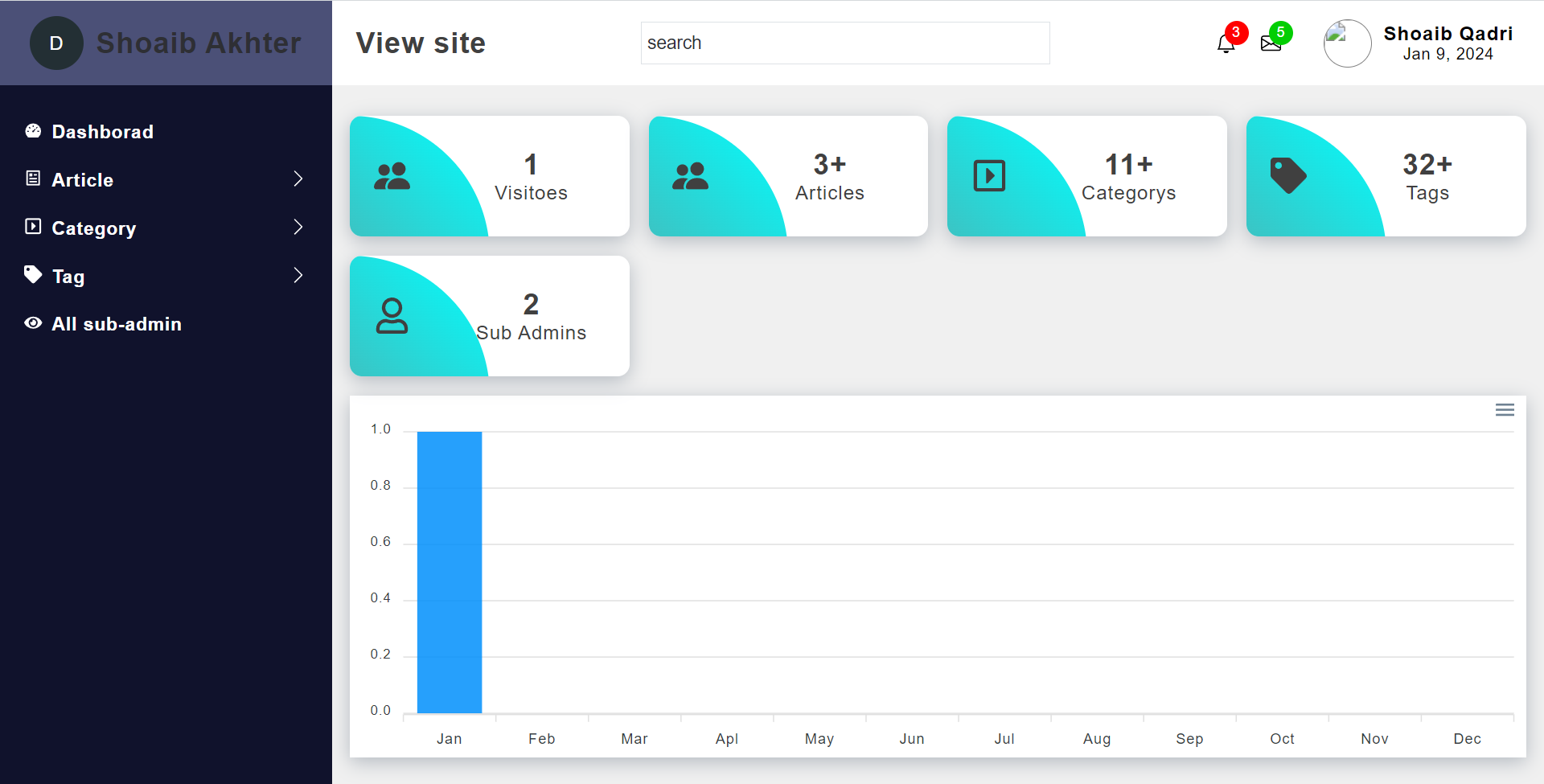
****

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

****

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

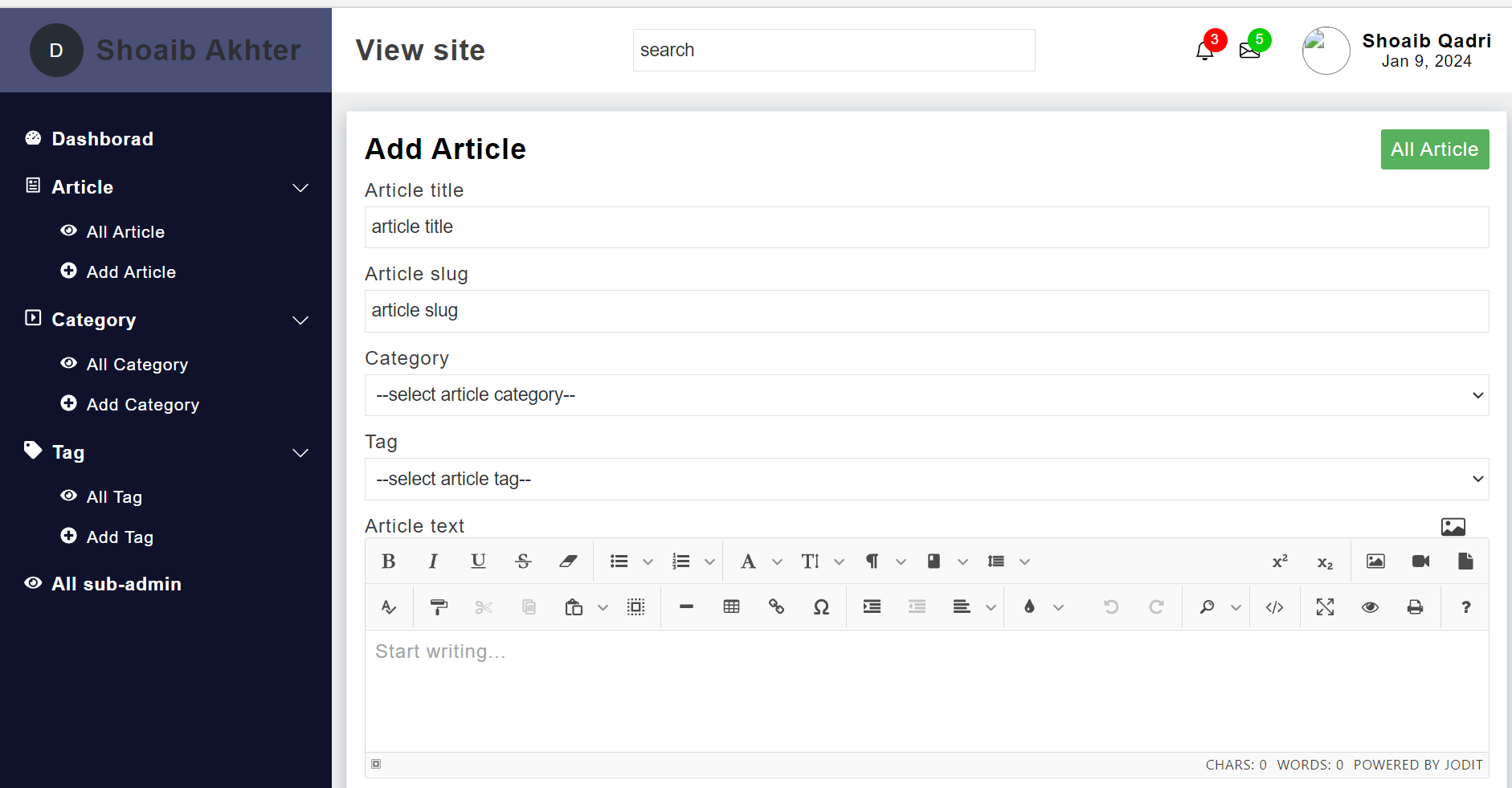
Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

****

1. **CONCLUSION**

Blogify stands as a testament to the capabilities of modern web technologies. Its feature-rich environment, coupled with a secure and user-friendly design, positions it as a top-tier blogging application. The project successfully integrates multiple technologies to deliver a comprehensive solution for content creators and readers, making it a noteworthy contribution to the realm of online publishing.