Name: Rubesh Raman

Registration Number: 12115752

WINTER PEP PROJECT

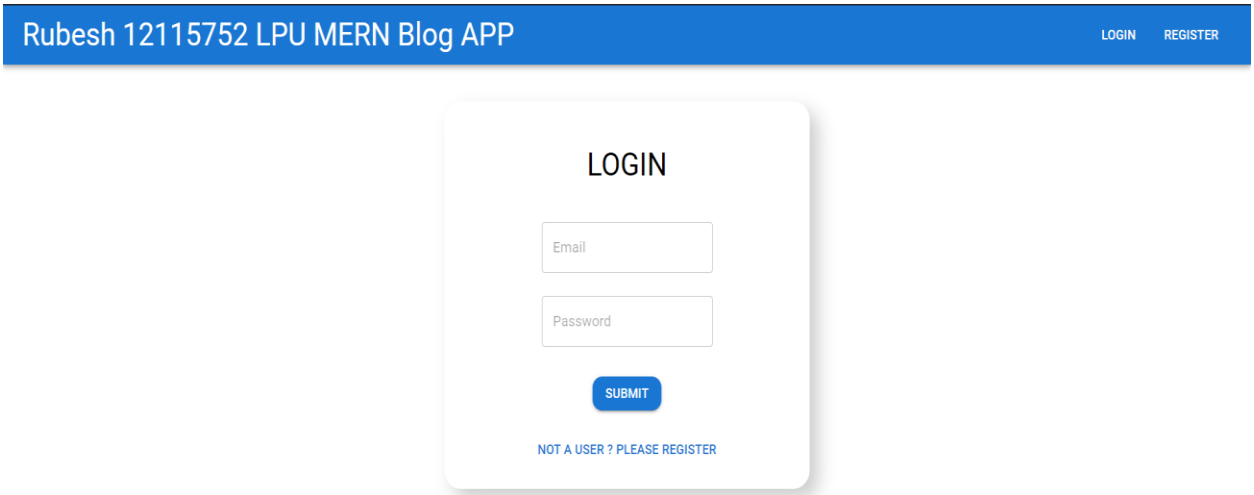
Name of Project: RUBESH 12115752 LPU MERN Blog APP

SUBJECT: MERN STACK

From: 10/01/2024 to 13/02/2024

INTRODUCTION

This project introduces a MERN (MongoDB, Express.js, React.js, Node.js) stack application for creating a robust Rubesh Blogging Website. Users can seamlessly publish and manage blog posts, securely stored in a cloud-based MongoDB database. The backend relies on Node.js and Express.js, while React.js crafts an intuitive frontend. The primary function enables users to effortlessly compose and publish blog content. Once published, the system generates a unique link for easy access to the latest blog post. Users can conveniently share and access their blogs from anywhere with internet connectivity. The combination of MongoDB, Express.js, React.js, and Node.js provides a powerful solution for building and managing a dynamic blogging platform. Each stack component contributes unique capabilities for crafting cutting-edge websites.



HOW TO RUN

BACKEND

\Rubesh Blog New> npm run

```
C:\Rubesh Blog New> npm run
```

```
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
Server Running on undefined mode port no 8080
Congrats Rubesh Connected to Mongodb Database ac-jwmwjtg-shard-00-00.9y4ncly.mongodb.net
```

FRONTEND

\Rubesh Blog New

cd client

\Rubesh Blog New\client

\Rubesh Blog New\client> npm start

```
PS C:\Rubesh Blog New> cd client
PS C:\Rubesh Blog New\client> npm start

Compiled successfully!
```

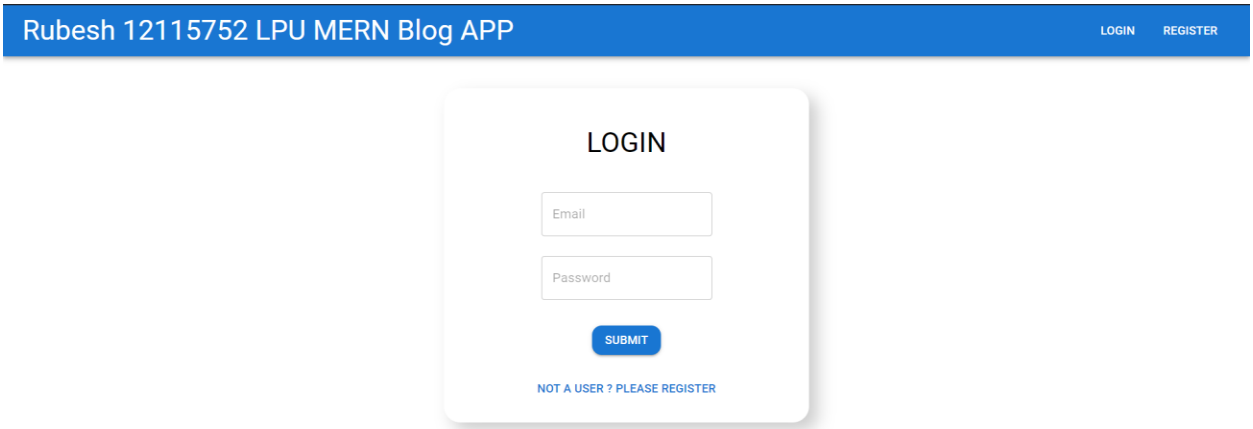
```
  Local:            http://localhost:3000
  On Your Network:  http://192.168.25.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

FRONT-END DEVELOPMENT

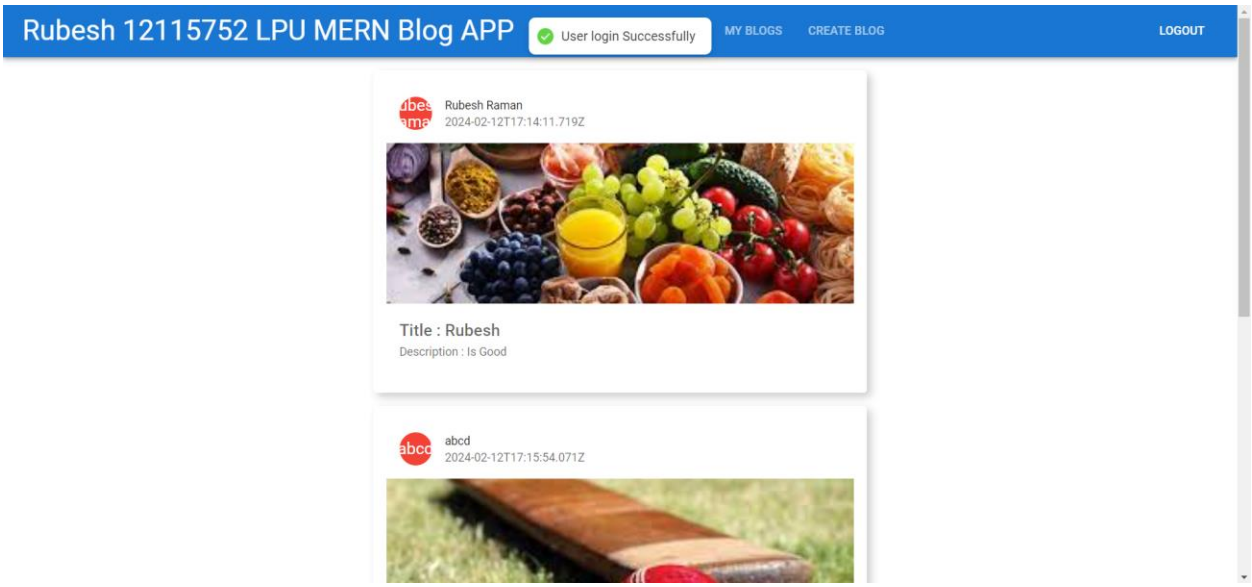
Front-End Developer is important to make and interactive websites and web applications. This focuses on the responsive functionality of a website; Front-End Development is also popular for the aesthetics and good user experience. It consists of constructing the user interface that user sees and interact. This involves employing languages like HTML, CSS, and JavaScript to ensure seamless navigation and engaging design.

I have made the Frontend using React, for UI Experience I have used Material UI.

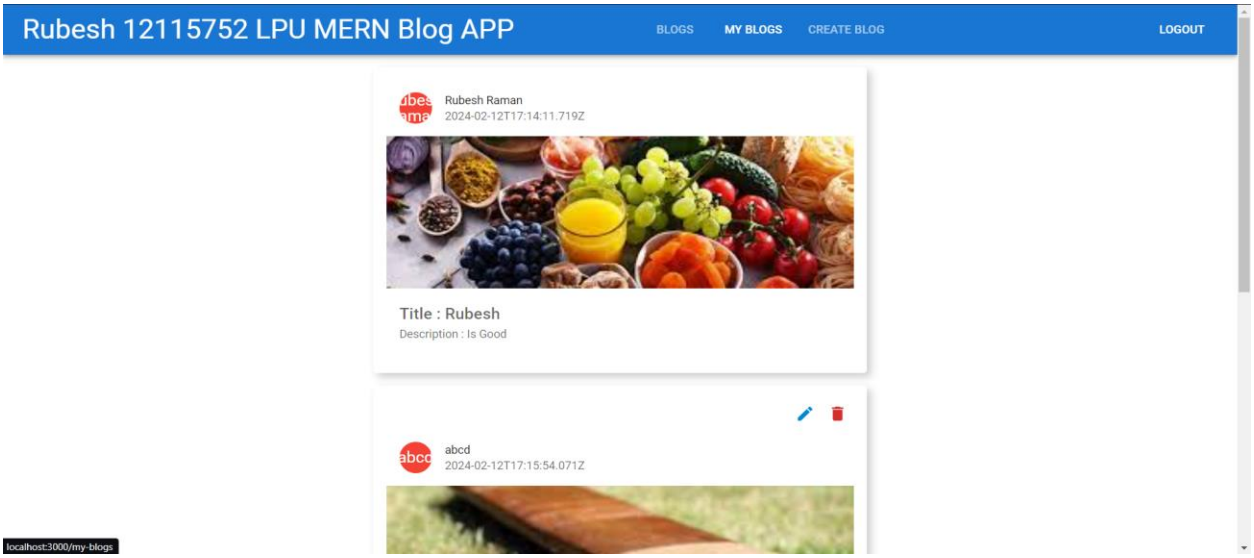
Login Page



Main Page



My Blogs



Create Blog

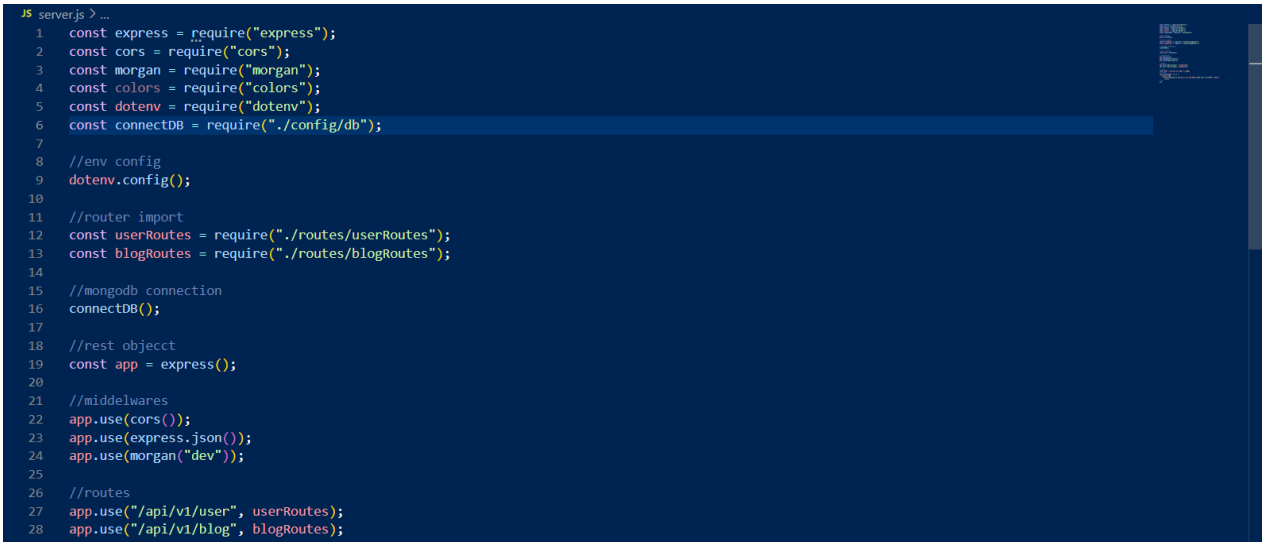
BACK-END DEVELOPMENT

The backend is like an administration room of a website. It handles data, ensures everything on the visible part works smoothly, but this all works on the back part of the website. User can't directly interact with it.

Back-end frameworks are toolkits for building the hidden parts of websites. They come with ready-made tools, rules, and shortcuts that make it easier for developers to create powerful and reliable server using codes snippets without having to build everything from scratch. These frameworks handle common jobs like talking to databases, making sure users are verified, and deciding how different parts of a website should connect.

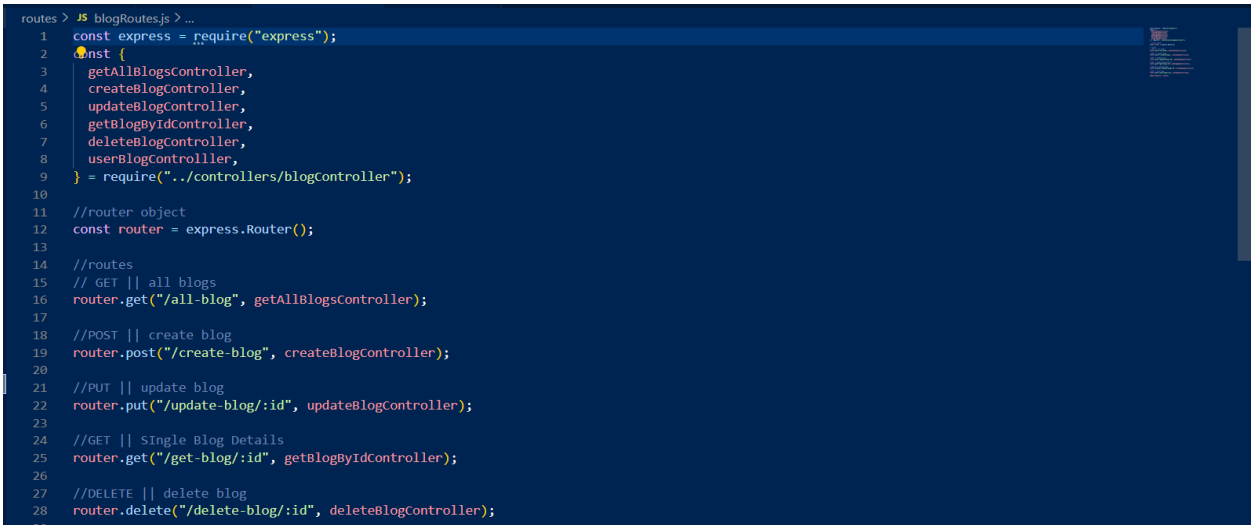
I have made a file server.js has all the important dependencies and function for our server to work. There are two objects in collection one is User and another is Blog. So that the information of Blog and User are saved independently.

Server.js



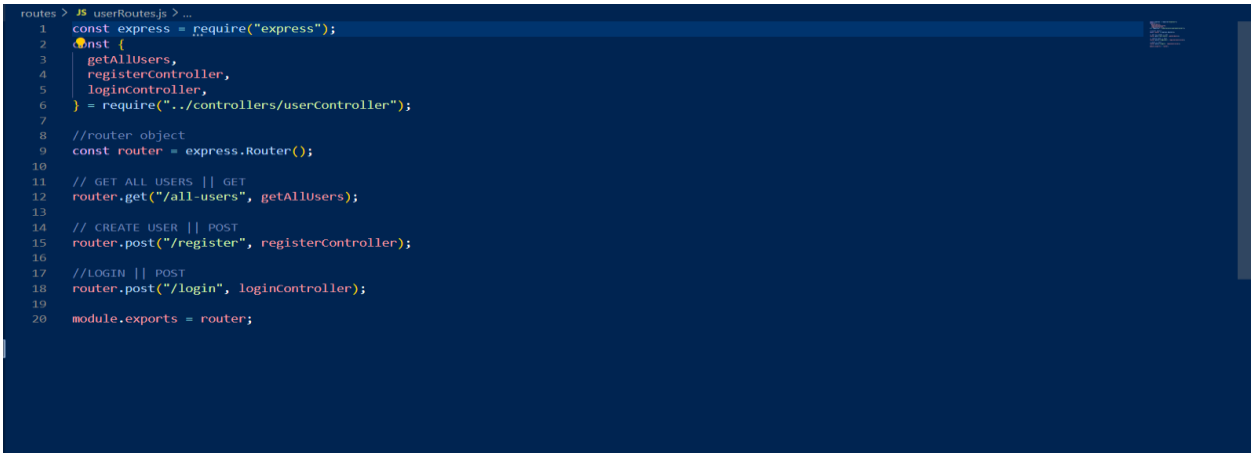
```
server.js > ...
const express = require("express");
const cors = require("cors");
const morgan = require("morgan");
const colors = require("colors");
const dotenv = require("dotenv");
const connectDB = require("./config/db");

//env config
dotenv.config();

//router import
const userRoutes = require("./routes/userRoutes");
const blogRoutes = require("./routes/blogRoutes");

//mongodb connection
connectDB();

//rest objecct
const app = express();

//middelwares
app.use(cors());
app.use(express.json());
app.use(morgan("dev"));

//routes
app.use("/api/v1/user", userRoutes);
app.use("/api/v1/blog", blogRoutes);
```

blogRoutes.js



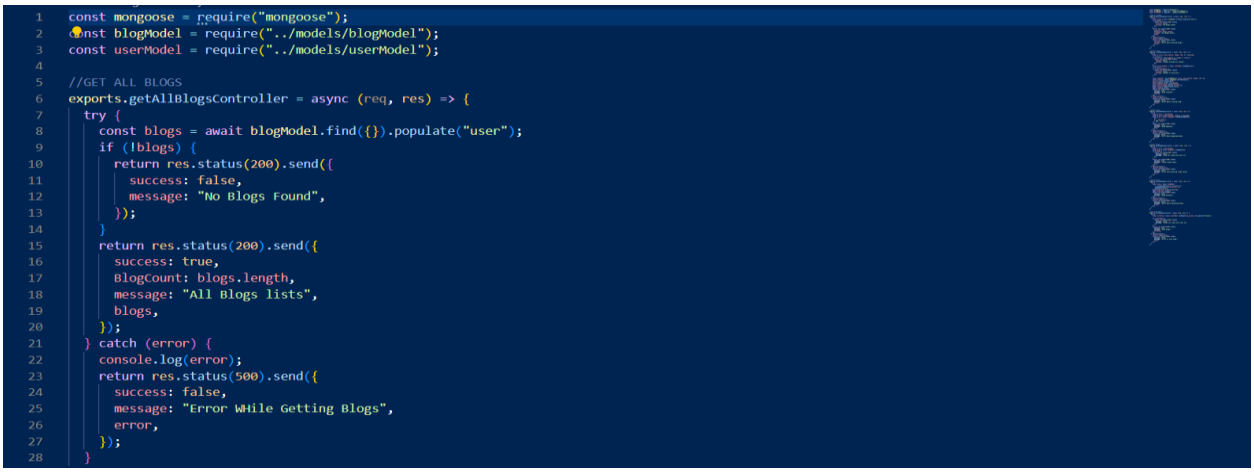
```
routes > blogRoutes.js > ...
const express = require("express");
const {
  getAllBlogsController,
  createBlogController,
  updateBlogController,
  getBlogByIdController,
  deleteBlogController,
  userBlogControlller,
} = require("../controllers/blogController");

//router object
const router = express.Router();

//routes
// GET || all blogs
router.get("/all-blog", getAllBlogsController);

//POST || create blog
router.post("/create-blog", createBlogController);

//PUT || update blog
router.put("/update-blog/:id", updateBlogController);

//GET || SIngle Blog Details
router.get("/get-blog/:id", getBlogByIdController);

//DELETE || delete blog
router.delete("/delete-blog/:id", deleteBlogController);
```

userRoutes.js



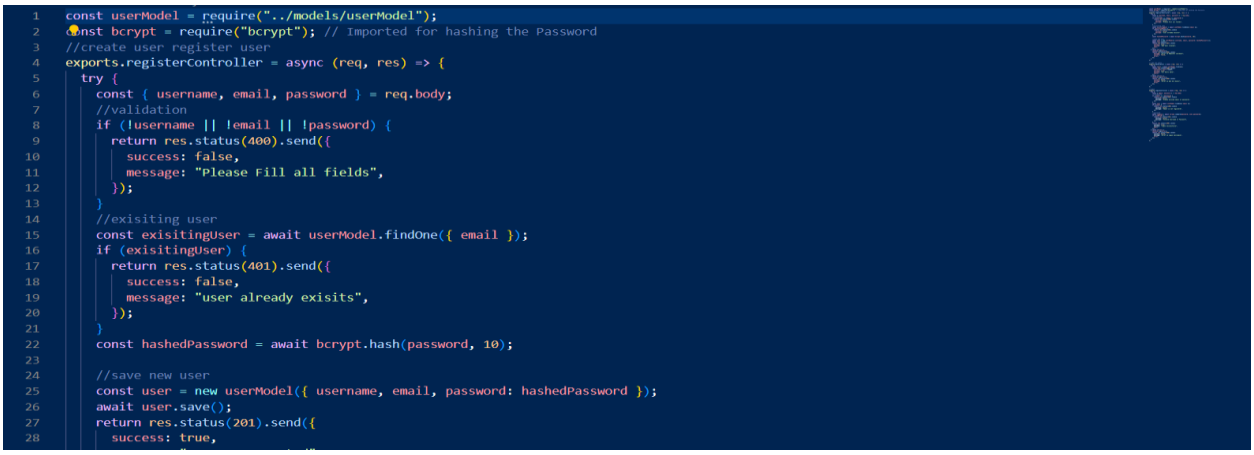
```
routes > JS userRoutes.js > ...
const express = require("express");
const {
  getAllUsers,
  registerController,
  loginController,
} = require("../controllers/userController");

//router object
const router = express.Router();

// GET ALL USERS || GET
router.get("/all-users", getAllUsers);

// CREATE USER || POST
router.post("/register", registerController);

//LOGIN || POST
router.post("/login", loginController);

module.exports = router;
```

blogController.js



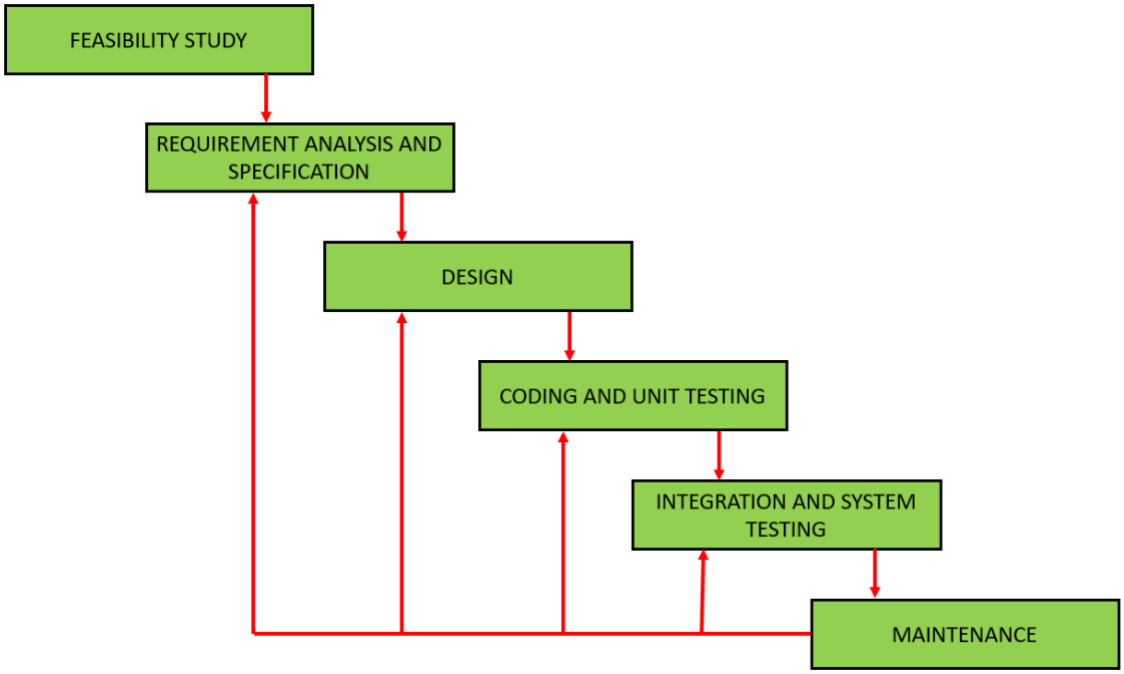
```
const mongoose = require("mongoose");
const blogModel = require("../models/blogModel");
const userModel = require("../models/userModel");

//GET ALL BLOGS
exports.getAllBlogsController = async (req, res) => {
  try {
    const blogs = await blogModel.find({}).populate("user");
    if (!blogs) {
      return res.status(200).send({
        success: false,
        message: "No Blogs Found",
      });
    }
    return res.status(200).send({
      success: true,
      BlogCount: blogs.length,
      message: "All Blogs lists",
      blogs,
    });
  } catch (error) {
    console.log(error);
    return res.status(500).send({
      success: false,
      message: "Error WHile Getting Blogs",
      error,
    });
  }
```

userController.js

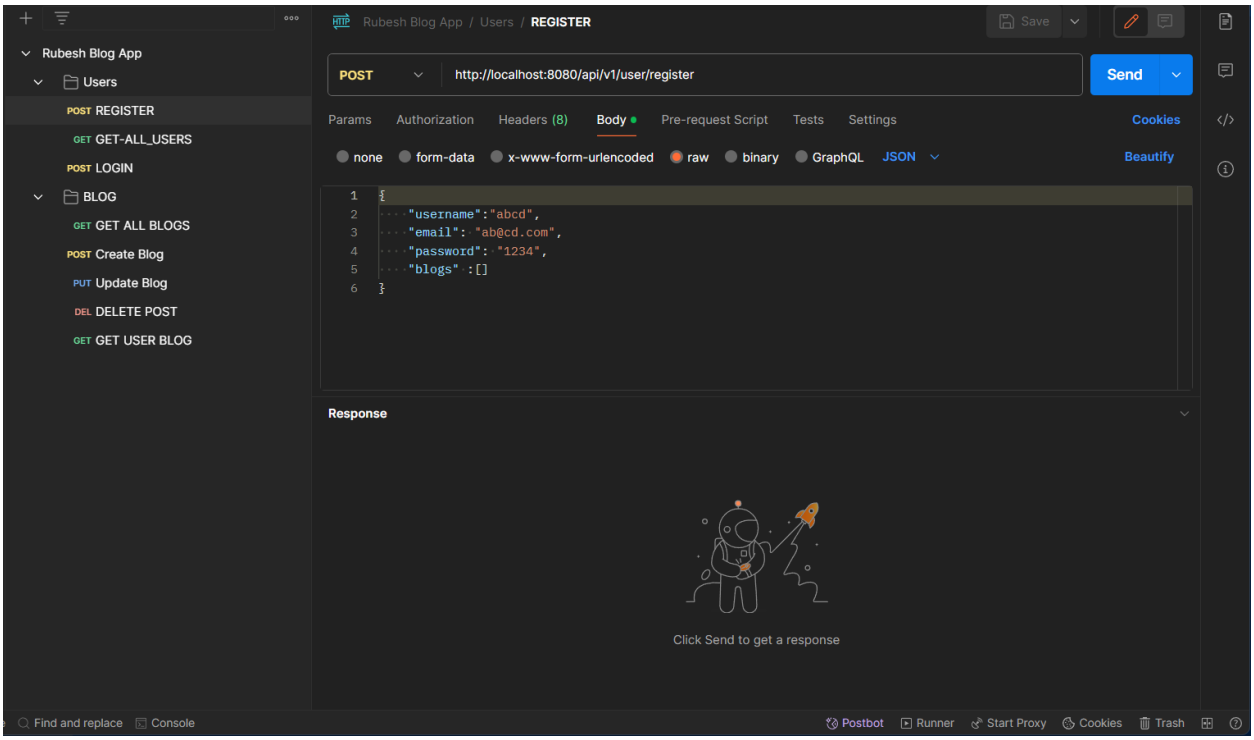


```
const userModel = require("../models/userModel");
const bcrypt = require("bcrypt"); // Imported for hashing the Password
//create user register user
exports.registerController = async (req, res) => {
  try {
    const { username, email, password } = req.body;
    //validation
    if (!username || !email || !password) {
      return res.status(400).send({
        success: false,
        message: "Please Fill all fields",
      });
    }
    //exisiting user
    const exisitingUser = await userModel.findOne({ email });
    if (exisitingUser) {
      return res.status(401).send({
        success: false,
        message: "user already exisits",
      });
    }
    const hashedPassword = await bcrypt.hash(password, 10);

    //save new user
    const user = new userModel({ username, email, password: hashedPassword });
    await user.save();
    return res.status(201).send({
      success: true,
```

PARADIGM in MERN PROJECT



TESTING



CONCLUSION

In conclusion, this Blogging Website, built on the MERN stack, offers a seamless and efficient platform for users to publish and manage their blog posts. The integration of MongoDB, Express.js, React.js, and Node.js ensures a robust and dynamic experience, allowing users to create and share content effortlessly.

I extend my heartfelt gratitude to CipherSchool for providing invaluable guidance and support throughout this project. Harshit Sir's expertise and Nitesh Sir's mentorship have been instrumental in shaping this endeavor. Their dedication to imparting knowledge and learning environment has been an important behind the completion of this project. I am grateful for the opportunity to learn and grow under their mentorship, and I look forward to applying these skills in future.

Thank you, CipherSchool, Harshit Sir, and Nitesh Sir, for your remarkable guidance and support.

SOURCE CODE

Source Code: https://github.com/Apex-Overlord-5/CipherSchools_12115752_Rubesh_Blogging_Website