



L OVELY
P ROFESSIONAL
U NIVERSITY



CipherSchools
*/CipherSchools

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.

Rubesh 12115752 LPU MERN Blog APP

[LOGIN](#) [REGISTER](#)

LOGIN

SUBMIT

[NOT A USER ? PLEASE REGISTER](#)

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

Rubesh 12115752 LPU MERN Blog APP

LOGINREGISTER

LOGIN

Email

Password

SUBMIT

NOT A USER ? PLEASE REGISTER

Main Page

Rubesh 12115752 LPU MERN Blog APP

User login Successfully


MY BLOGSCREATE BLOG

LOGOUT

abes

Rubesh Raman

2024-02-12T17:14:11.719Z




Title : Rubesh

Description : Is Good

abcd

abcd

2024-02-12T17:15:54.071Z



My Blogs

Rubesh 12115752 LPU MERN Blog APP

BLOGS

MY BLOGS


CREATE BLOG

LOGOUT

abes
rma

Rubesh Raman

2024-02-12T17:14:11.719Z




Title : Rubesh

Description : Is Good

abcd

abcd

2024-02-12T17:15:54.071Z



localhost:3000/my-blogs

Create Blog

Rubesh 12115752 LPU MERN Blog APP

BLOGS

MY BLOGS

CREATE BLOG

LOGOUT

Create A Post / Blog

Title

Description

Image URL

SUBMIT

localhost:3000/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

```
JS server.js > ...
1  const express = require("express");
2  const cors = require("cors");
3  const morgan = require("morgan");
4  const colors = require("colors");
5  const dotenv = require("dotenv");
6  const connectDB = require("./config/db");
7
8  //env config
9  dotenv.config();
10
11 //router import
12 const userRoutes = require("./routes/userRoutes");
13 const blogRoutes = require("./routes/blogRoutes");
14
15 //mongodb connection
16 connectDB();
17
18 //rest object
19 const app = express();
20
21 //middlewares
22 app.use(cors());
23 app.use(express.json());
24 app.use(morgan("dev"));
25
26 //routes
27 app.use("/api/v1/user", userRoutes);
28 app.use("/api/v1/blog", blogRoutes);
```

blogRoutes.js

```
routes > JS blogRoutes.js > ...
1  const express = require("express");
2  const {
3    getAllBlogsController,
4    createBlogController,
5    updateBlogController,
6    getBlogByIdController,
7    deleteBlogController,
8    userBlogController,
9  } = require("../controllers/blogController");
10
11 //router object
12 const router = express.Router();
13
14 //routes
15 // GET || all blogs
16 router.get("/all-blog", getAllBlogsController);
17
18 //POST || create blog
19 router.post("/create-blog", createBlogController);
20
21 //PUT || update blog
22 router.put("/update-blog/:id", updateBlogController);
23
24 //GET || Single Blog Details
25 router.get("/get-blog/:id", getBlogByIdController);
26
27 //DELETE || delete blog
28 router.delete("/delete-blog/:id", deleteBlogController);
```

userRoutes.js

```
routes > JS userRoutes.js > ...
1  const express = require("express");
2  const {
3    getAllUsers,
4    registerController,
5    loginController,
6  } = require("../controllers/userController");
7
8  //router object
9  const router = express.Router();
10
11 // GET ALL USERS || GET
12 router.get("/all-users", getAllUsers);
13
14 // CREATE USER || POST
15 router.post("/register", registerController);
16
17 // LOGIN || POST
18 router.post("/login", loginController);
19
20 module.exports = router;
```

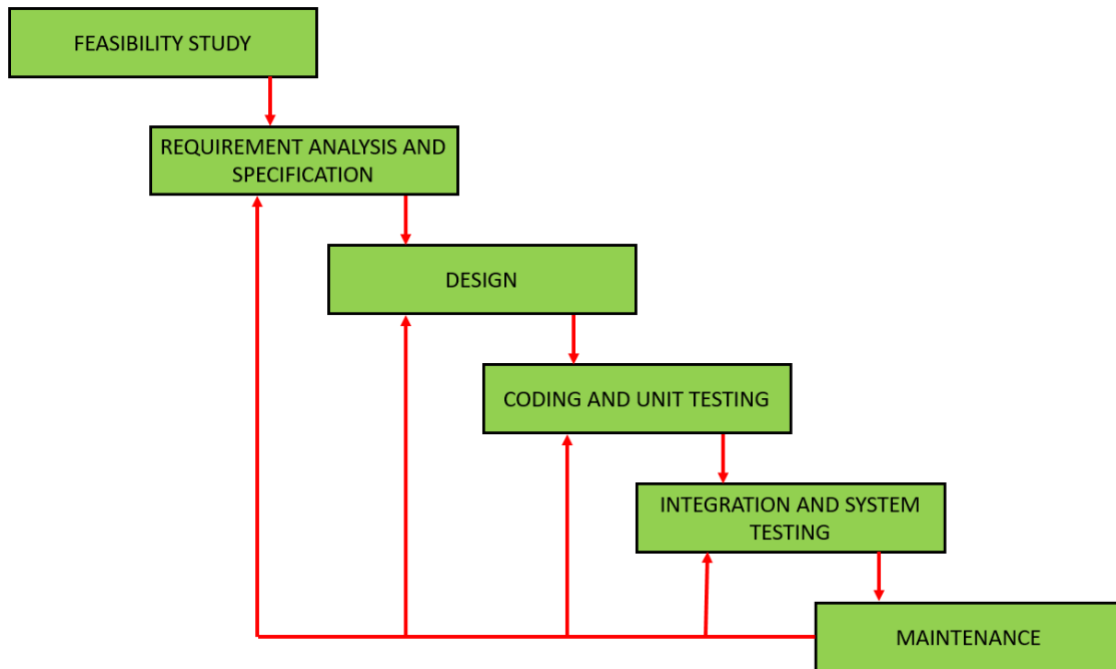
blogController.js

```
1  const mongoose = require("mongoose");
2  const blogModel = require("../models/blogModel");
3  const userModel = require("../models/userModel");
4
5  //GET ALL BLOGS
6  exports.getAllBlogsController = async (req, res) => {
7    try {
8      const blogs = await blogModel.find({}).populate("user");
9      if (!blogs) {
10        return res.status(200).send({
11          success: false,
12          message: "No Blogs Found",
13        });
14      }
15      return res.status(200).send({
16        success: true,
17        blogCount: blogs.length,
18        message: "All Blogs lists",
19        blogs,
20      });
21    } catch (error) {
22      console.log(error);
23      return res.status(500).send({
24        success: false,
25        message: "Error While Getting Blogs",
26        error,
27      });
28    }
29  }
```

userController.js

```
1  const userModel = require("../models/userModel");
2  const bcrypt = require("bcrypt"); // Imported for hashing the Password
3  //create user register user
4  exports.registerController = async (req, res) => {
5    try {
6      const { username, email, password } = req.body;
7      //validation
8      if (!username || !email || !password) {
9        return res.status(400).send({
10          success: false,
11          message: "Please Fill all fields",
12        });
13      }
14      //existing user
15      const existingUser = await userModel.findOne({ email });
16      if (existingUser) {
17        return res.status(401).send({
18          success: false,
19          message: "user already exists",
20        });
21      }
22      const hashedPassword = await bcrypt.hash(password, 10);
23
24      //save new user
25      const user = new userModel({ username, email, password: hashedPassword });
26      await user.save();
27      return res.status(201).send({
28        success: true,
```

PARADIGM in MERN PROJECT



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