



Institute of Engineering & Technology

Final Project Report: Blogging Web Application Using MEAN Stack

Submitted by:

Shubham Sharma (181500696)

Vivek Kumar Singh (181500818)

Submitted to:

Faculty Name: Mr. Akash Kumar Choudhary

Declaration

I the undersigned solemnly declare that the project report BLOGGING WEB APPLICATION USING MEAN STACK is based on my own work carried out during the course of our study under the supervision of Mr. Akash Kumar Choudhary. I assert the statements made and conclusions drawn are an outcome of my work. I further certify that I. The work contained in the report is original and has been done by me under the general supervision of my supervisor. II. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or any other University of India or abroad. III. We have followed the guidelines provided by the university in writing the report. IV. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and giving their details in the references.

Signature of Candidate: Vivek, Shubham

Name of Candidate: Vivek Kumar & Shubham Sharma

Roll. No. :181500818, 181500696

Course: B. Tech. CSE

Year: 2020-21

Semester: VI

Acknowledgement

This project is an acknowledgement to the intensity drive and technical competence of many persons who have contributed to it. I express my heartiest gratitude and deepest thanks to, Project Instructor Mr. Akash Kumar Choudhary for his proper guidance, suggestions and helping me in completing the project and I am very grateful towards the support he showed throughout the whole project. I am highly grateful to my parents who have been the source of money and encouragement during the course of my work. I would also like to thank all those who directly or indirectly supported or helped me in completing my project in time. I would like to express my gratitude towards my parents and members of my college for their kind cooperation and encouragement which helped me in completion of this project. All of them have willingly helped me out with their abilities.

Table of Contents

S No.	Title	Page No.
1	Introduction to the project: -	5
1.1	Reason for selecting the topic	5
1.2	Objectives of the project	5
1.3	Feasibility Study	5
1.4	Future Scope	5
1.5	Modules in Blogging Web Application	6
1.6	Input Data and Validation of Project on Online Blogging System	6
2	About Online Blogging Web Application	6
3	Methodology	7
4	Introduction to Technologies of MEAN Stack	8
5	Hardware and Software Requirements	9
6	Test Technologies	10
7	What contribution would the project make and where?	10
8	Problem Statement	10
9	Scope for Extension into a Major Project	10
10	Software Designing: -	11
10.1	Zero Level DFD for Blogging Web Application	11
10.2	First Level DFD for Blogging Web Application	11
10.3	Second Level DFD for Blogging Web Application	12
10.4	Usecase Diagram for Blogging Web Application	13
10.5	ER Diagram for Blogging Web Application	13
11	Screenshots for Online Blogging Web Application: -	14
11.1	Implementation Screenshots	14
11.2	UI	17
11.3	List of restrictions available in Blogging online systems	
11.4	Existing System	
11.5	Proposed System on Online Blogging System	
11.6	Features on Blogging Application	
12	Conclusion	18
13	References	19

1. Introduction to the Project

1.1. Reason for selecting the topic

We chose the topic because we wanted to build something really dynamic system for posting blogs for the users. The main aim of developing this platform is to create a team blog, allowing multiple bloggers to contribute to a single blog. We select which team members have administrative authority and which are authors only. We can also choose to make our blog private and restrict who can view it.

1.2. Objectives of the project

Publishing content to a blog is frequently seen as a decent method to connect with people with similar interest. This blog is a kind of site that is included sections either made by the understudies or different individuals. A blog is an extraordinary path for an individual to figure out how to make their own site. There are many options available for blog like WordPress, web blog etc. In this we can create only individual blogs on which we can post or share anything that we want which is publicly available. There is no way to create team blogs for any organization, company or institute. This will be done using the mean stack which basically is an open-source JavaScript software for building web sites and web application. *MEAN* is an acronym for MongoDB, ExpressJS, AngularJS and Node. The main objective of the web Blogging System is to manage the small print of Blogs, Topic ,Idea, Content, Entries. It manages all the knowledge regarding Blogs, Views, Entries, Blogs. The project is completely engineered at body finish and so solely the administrator is secure the access. the aim of the project is to create AN application to cut back the manual work for managing the Blogs, Topic, Views, Idea. It tracks all the small print regarding the thought, Content, Entries.

1.3. Feasibility Study

After doing the project blogging system, study and analyzing all the existing or required functionalities of the system, the next task is to do the feasibility study for the project. All projects are feasible – given unlimited resources and infinite time.

- A) Economical Feasibility
- B) Technical Feasibility
- C) Operational Feasibility

1.4. Future Scope

This web application will be utilizing this publishing content to a blog stage educator can undoubtedly advance significant notification identified with the association to their understudies. Understudies can post their recommendations and input for the association. In the event that somebody can't communicate her/his view before individuals those likewise share the things through this contributing to a blog framework.

It can be summarized that the future scope of the project circles around maintaining information regarding:

- 1) Implement the backup mechanism for taking backup.
- 2) Automatic detection of errors like grammar.
- 3) Automatic notification to followers after posting blog

1.5. Modules in Blogging Web Application

- **Blogs Management Module:** Will be used for managing the Blogs details.
- **Views Module:** Will be used for managing the details of views.
- **Content Module:** Will be used for managing the details of content.
- **Login Module:** Will be used for managing the login details.
- **Users Module:** Will be used for managing the users of the system.

1.6. Input Data and Validation of Project on Online Blogging System

- All the fields such as Blogs, Topic, Views are validated and does not take invalid values.
- Each form in the application will not accept the blank value fields.
- Validations for user input will be done.
- Checking of the Coding standards to be maintained during coding.
- Testing the module with all the possible test data.

2. About Online Blogging Web Application

The application is reduced as much as possible to avoid errors while entering the data. It also provides error messages while entering the invalid data. No formal knowledge is needed for the user to use this system. Thus, by this all it provides it is user-friendly. Online Blogging System, as described above, can lead to error free, reliable, secure and fast management system. It can assist the user to concentrate on their other activities rather to concentrate on record keeping. The purpose of Online Blogging Web Application is to manage all the blogs, comments and all other basic information about the blog. Making User able to see the blogs posted by other users. The user can edit their own blogs as per their requirements. The aim is to make adding, updating and removing the posts as efficient as possible as to improve the resource management of the blogs data.

3. Methodology

This web application will be designed using HTML, CSS, JavaScript, AngularJS, mongo dB, NodeJS, and Express. The frontend part of the project will be done by HTML, CSS, JavaScript and the backend part will be done by NodeJS. All the data will be stored in mongoDB.

Mean Stack refers to a collection of JavaScript technologies used to develop web applications. Therefore, from the client to the server and from server to database, everything is based on JavaScript. **MEAN** is a full-stack development toolkit used to develop a fast and robust web applications. MEAN is a user-friendly stack which is the ideal solution for building dynamic websites and applications. This free and open-source stack offers a quick and organized method for creating rapid prototypes for web-based applications.

Components of MEAN Stack

MEAN is comprised of four different technologies:

- MongoDB express is a schema less NoSQL database system
- Express JS is a framework used to build web applications in Node
- AngularJS is a JavaScript framework developed by Google
- Node.js is a server-side JavaScript execution environment

The online blogging system has been developed to override the problems prevailing in the practicing manual system. This software is supported to eliminate and, in some cases, reduce the hardships faced by this existing system. Moreover, this system is designed for the particular need of the company to carry out operations in a smooth and effective manner.

The Application is reduced as much as possible to avoid errors while entering the data. It also provides error message while entering invalid data. No formal knowledge is needed for the user to use this system. Thus, by this all it proves it is user-friendly. Online blogging system. It can Assist the user to concentrate on their other activities rather to concentrate on the record keeping.

HTML: HTML, which stands for Hypertext Mark-Up Language, is the language for describing structured documents as well as the language used to create web pages in the Internet.

CSS: Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

JavaScript: JavaScript is a scripting language and it is used on both client-side and server-side for making web pages interactive. It is a text-based language. Open and cross-platform.

MongoDB: MongoDB is a NoSQL document-oriented database; it is used for storing large amounts of data.

This platform provides an environment for posting or sharing knowledge throughout the organization. This gives detailed information about functional and non-functional requirements of users or students i.e. whatever they want to share then they can easily write on their blog and it can be viewed by the member in the organization. The purpose meets the goal of sharing the information in formal or in secure way.

4. Introduction to Technologies of MEAN Stack

- **M:** - M stands for MongoDB. Working with MongoDB NoSQL database is much easier than working with any relational database. There are no tables in MongoDB. All the data is stored in JSON format i.e. key-value pairs. In JSON, we define a unique key with a value associated with it. These key-value pairs are stored in a document, which in turn is stored in a collection. A collection in MongoDB can have any number of documents and such documents can have any number of key-value pairs. As I mentioned earlier, data in the MongoDB database is stored in BSON. BSON is nothing but extended JSON. It supports more data types than JSON. We store anything like, string, integer, boolean, double, binary data, object, array, javascript code and many more.

These documents are grouped inside a collection. A collection can be equivalent to a table in a relational SQL database. A collection always exists in a database and there is no pre-defined structure of a collection. In SQL, the database contains tables and in MongoDB, the database contains collections.

- **E:** - E stands for Express.js. Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy. Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love. Many [popular frameworks](#) are based on Express.

- **A:** - A stands for Angular. Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your apps.

The architecture of an Angular application relies on certain fundamental concepts. The basic building blocks of the Angular framework are Angular components that are

organized into NgModules. NgModules collect related code into functional sets; an Angular app is defined by a set of NgModules. An app always has at least a root module that enables bootstrapping, and typically has many more feature modules.

- Components define views, which are sets of screen elements that Angular can choose among and modify according to your program logic and data.
 - Components use services, which provide specific functionality not directly related to views. Service providers can be injected into components as dependencies, making your code modular, reusable, and efficient.
-
- **N:** - N stands for Node.js. As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications.

5. Hardware and Software Requirements

1) For VS Code Editor: -

Hardware

- 1.6 GHz or faster processor
- 1 GB of RAM

Platforms

- OS X Yosemite (10.10+)
- Windows 7 (with .NET Framework 4.5.2), 8.0, 8.1 and 10 (32-bit and 64-bit)
- Linux (Debian): Ubuntu Desktop 16.04, Debian 9
- Linux (Red Hat): Red Hat Enterprise Linux 7, CentOS 8, Fedora 24 2)

For MongoDB: - Recommended Platforms

- Amazon Linux 2 • Debian 9 and 10
- RHEL / CentOS 6, 7, and 8 • SLES 12 and 15
- Ubuntu LTS 16.04, 18.04, and 20.04
- Windows Server 2016 and 2019

3) Technologies Used HTML, CSS, JAVASCRIPT, NodeJS, AngularJS, ExpressJS, MongoDB, TypeScript

4) Browser Opera, Chrome, Firefox

6. Test Technologies

For testing the web applications we will use “**burp suite**” to find and flaws in the website like XSS(cross site scripting), broken authentication, Security Misconfiguration and etc. With these testing we will make sure that the web application that we created is safe for the users and any errors will be fixed in real time. Apart from software we will make sure to make some test cases to check the website’s feasibility individually. On the other hand, we will make sure to check OWASP to stay update with new bugs and injections.

7. What contribution would the project make and where?

We think that the project will be best suited for someone who likes to get exposé or want to expand their domain, personally or business wise. Because it will help them to send their target audience a message about themselves. This will help them grow and connect to more people. The field it will be effective the most would be for e-commerce for expanding business.

8. Problem Statement

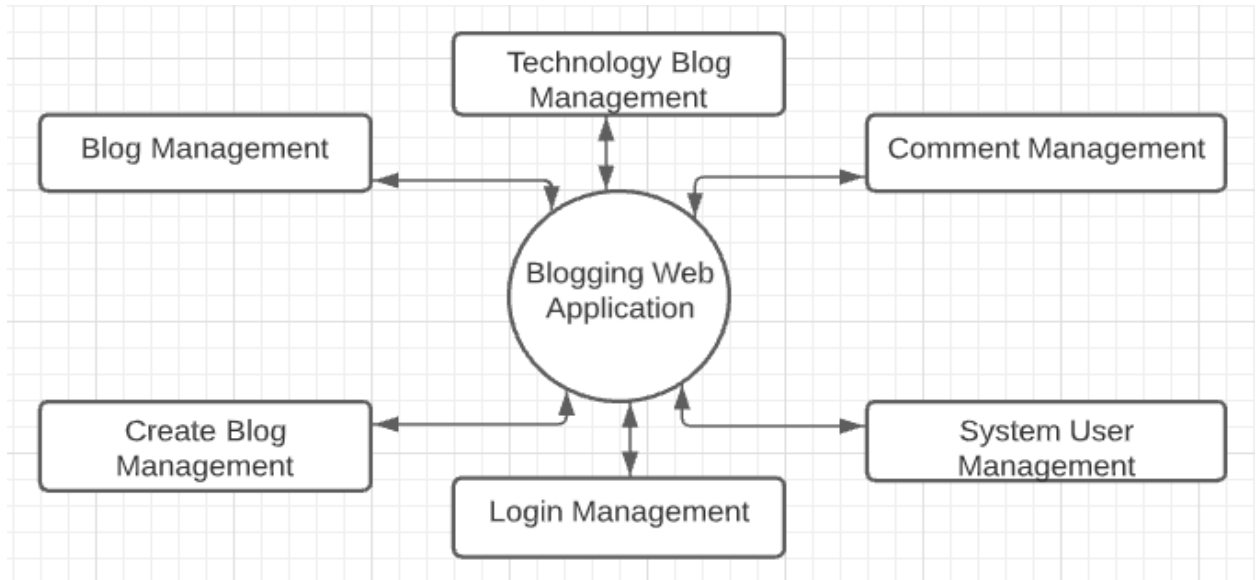
The problems we are going to solve through this project is the common problem that every new blogger face. Through this we are trying to give exposé to the user and convey a message to the target audience that the user need. It will be a place where people can share their knowledge to the world.

9. Scope for Extension into a Major Project

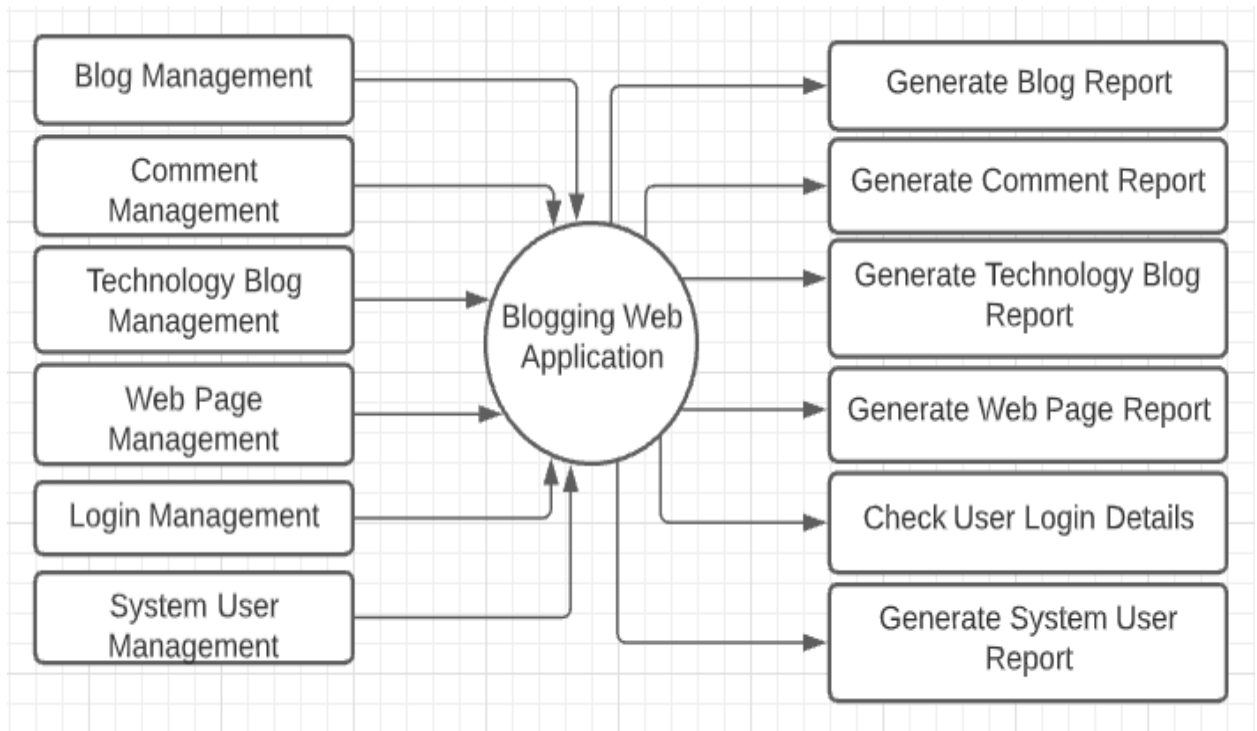
This project can further be extended to major project by adding some of the features which will make it easy for the user to find the most relevant content according to his or her interests by providing a certain set of different categories to the user. Furthermore the people who are following a person will get notified when that person will post some new content. If a user finds something interesting the he can save that post to favorites and at the same time they can leave a like for that post to which the author will get notified if somebody likes their post. Then the user will also be able to share the content to other users.

10. Software Designing

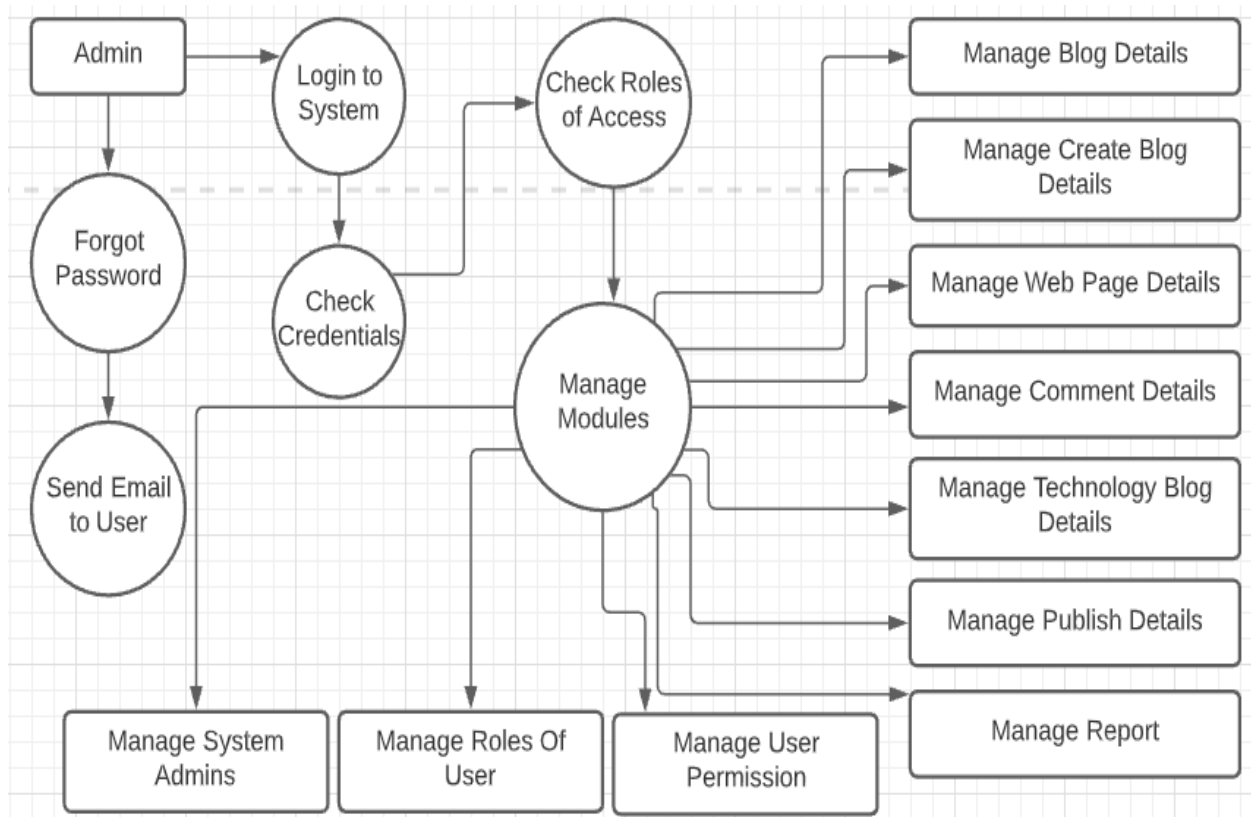
10.1. Zero Level DFD for Blogging Web Application



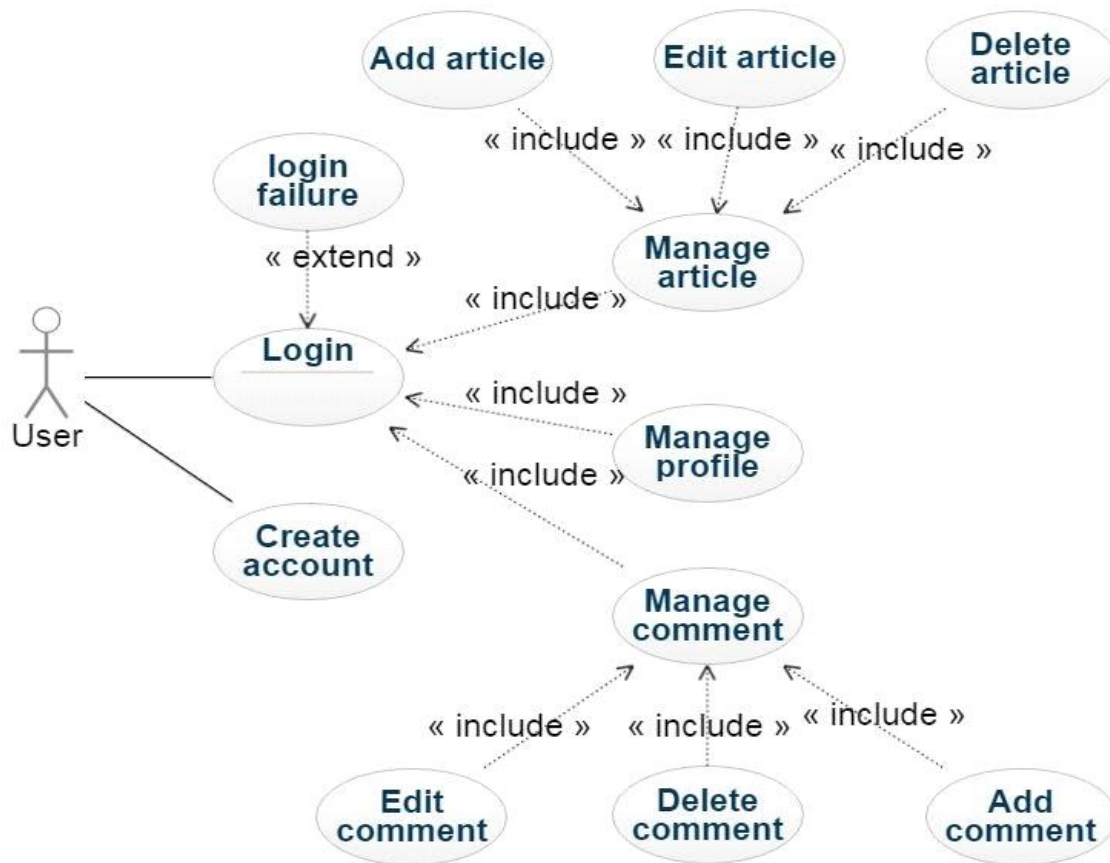
10.2. First Level DFD for Blogging Web Application



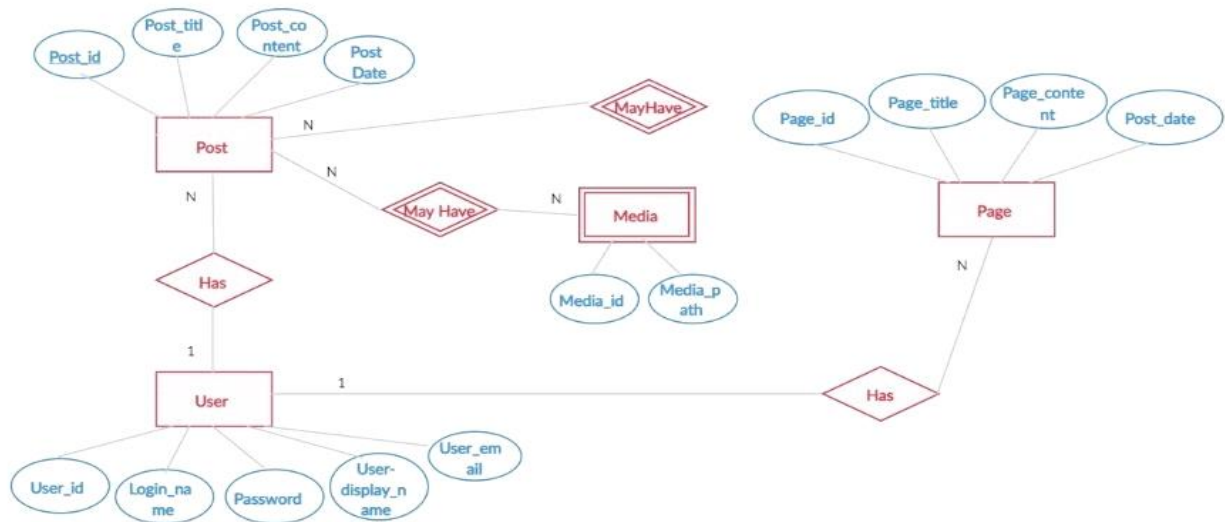
10.3. Second Level DFD for Blogging Web Application



10.4. Usecase Diagram for Blogging Web Application



10.5. ER Diagram for Blogging Web Application



11. Screenshots for Online Blogging Web Application

11.1. Implementation Screenshots

```
app.js
backend > app.js > ...
1 | const path = require("path");
2 | const express = require("express");
3 | const bodyParser = require("body-parser");
4 | const mongoose = require("mongoose");
5 |
6 | const postsRoutes = require("./routes/posts");
7 | const userRoutes = require("./routes/user");
8 |
9 | const app = express();
10 |
11 | mongoose
12 | .connect("mongodb://localhost:27017/BloggingDb")
13 | .then(() => {
14 |   console.log("Connected to database!");
15 | })
16 | .catch(() => {
17 |   console.log("Connection failed!");
18 | });
19 |
20 | app.use(bodyParser.json());
21 | app.use(bodyParser.urlencoded({ extended: false }));
22 | app.use("/images", express.static(path.join("backend/images")));
23 |
24 | app.use((req, res, next) => {
25 |   res.setHeader("Access-Control-Allow-Origin", "*");
26 |   res.setHeader(
27 |     "Access-Control-Allow-Headers",
28 |     "Origin, X-Requested-With, Content-Type, Accept, Authorization"
29 |   );
30 |   res.setHeader(
31 |     "Access-Control-Allow-Methods",
32 |     "GET, POST, PATCH, PUT, DELETE, OPTIONS"
33 |   );
34 | });
```



```
check-auth.js X
backend > middleware > check-auth.js > ...
1  const jwt = require("jsonwebtoken");
2
3  module.exports = (req, res, next) => {
4    try {
5      const token = req.headers.authorization.split(" ")[1];
6      const decodedToken = jwt.verify(token, process.env.JWT_KEY);
7      req.userData = { email: decodedToken.email, userId: decodedToken.userId };
8      next();
9    } catch (error) {
10     res.status(401).json({ message: "You are not authenticated!" });
11   }
12 };
13
```

```
posts.js X
backend > controllers > posts.js > createPost > exports.createPost > post > imagePath
1  const Post = require("../models/post");
2
3  exports.createPost = (req, res, next) => {
4    const url = req.protocol + "://" + req.get("host");
5    const post = new Post({
6      title: req.body.title,
7      content: req.body.content,
8      imagePath: url + "/images/" + req.file.filename,
9      creator: req.userData.userId
10    });
11    post
12      .save()
13      .then(createdPost => {
14        res.status(201).json({
15          message: "Post added successfully",
16          post: {
17            ...createdPost,
18            id: createdPost._id
19          }
20        });
21      })
22      .catch(error => {
23        res.status(500).json({
24          message: "Creating a post failed!"
25        });
26      });
27  });
28
29  exports.updatePost = (req, res, next) => {
30    let imagePath = req.body.imagePath;
31    if (req.file) {
32      const url = req.protocol + "://" + req.get("host");
33      imagePath = url + "/images/" + req.file.filename;
34    }
35    const post = Post.findById(req.params.id);
36    if (!post) {
37      res.status(404).json({ message: "Post not found" });
38    }
39    post.title = req.body.title;
40    post.content = req.body.content;
41    post.imagePath = imagePath;
42    post.creator = req.userData.userId;
43    post
44      .save()
45      .then(updatedPost => {
46        res.status(200).json({
47          message: "Post updated successfully",
48          post: updatedPost
49        });
50      })
51      .catch(error => {
52        res.status(500).json({ message: "Updating a post failed!" });
53      });
54  });
55
56  exports.deletePost = (req, res, next) => {
57    const post = Post.findById(req.params.id);
58    if (!post) {
59      res.status(404).json({ message: "Post not found" });
60    }
61    post
62      .remove()
63      .then(() => {
64        res.status(200).json({ message: "Post deleted successfully" });
65      })
66      .catch(error => {
67        res.status(500).json({ message: "Deleting a post failed!" });
68      });
69  });
70
71  exports.getPost = (req, res, next) => {
72    const post = Post.findById(req.params.id);
73    if (!post) {
74      res.status(404).json({ message: "Post not found" });
75    }
76    res.status(200).json(post);
77  };
78
79  exports.getPosts = (req, res, next) => {
80    Post.find()
81      .then(posts => {
82        res.status(200).json(posts);
83      })
84      .catch(error => {
85        res.status(500).json({ message: "Fetching posts failed!" });
86      });
87  };
88
89  exports.createPostWithImage = (req, res, next) => {
90    exports.createPost(req, res, next);
91    if (req.file) {
92      const url = req.protocol + "://" + req.get("host");
93      const imagePath = url + "/images/" + req.file.filename;
94      const post = Post.findById(req.body._id);
95      if (!post) {
96        res.status(404).json({ message: "Post not found" });
97      }
98      post.imagePath = imagePath;
99      post
100        .save()
101        .then(updatedPost => {
102          res.status(200).json({
103            message: "Post image updated successfully",
104            post: updatedPost
105          });
106        })
107        .catch(error => {
108          res.status(500).json({ message: "Updating post image failed!" });
109        });
110    }
111  };
112
113  exports.updatePostWithImage = (req, res, next) => {
114    exports.updatePost(req, res, next);
115    if (req.file) {
116      const url = req.protocol + "://" + req.get("host");
117      const imagePath = url + "/images/" + req.file.filename;
118      const post = Post.findById(req.params.id);
119      if (!post) {
120        res.status(404).json({ message: "Post not found" });
121      }
122      post.imagePath = imagePath;
123      post
124        .save()
125        .then(updatedPost => {
126          res.status(200).json({
127            message: "Post image updated successfully",
128            post: updatedPost
129          });
130        })
131        .catch(error => {
132          res.status(500).json({ message: "Updating post image failed!" });
133        });
134    }
135  };
136
137  exports.deletePostWithImage = (req, res, next) => {
138    exports.deletePost(req, res, next);
139    if (req.file) {
140      const url = req.protocol + "://" + req.get("host");
141      const imagePath = url + "/images/" + req.file.filename;
142      const post = Post.findById(req.params.id);
143      if (!post) {
144        res.status(404).json({ message: "Post not found" });
145      }
146      post.imagePath = imagePath;
147      post
148        .save()
149        .then(updatedPost => {
150          res.status(200).json({
151            message: "Post image updated successfully",
152            post: updatedPost
153          });
154        })
155        .catch(error => {
156          res.status(500).json({ message: "Updating post image failed!" });
157        });
158    }
159  };
160
161  exports.getPostWithImage = (req, res, next) => {
162    exports.getPost(req, res, next);
163    if (req.file) {
164      const url = req.protocol + "://" + req.get("host");
165      const imagePath = url + "/images/" + req.file.filename;
166      const post = Post.findById(req.params.id);
167      if (!post) {
168        res.status(404).json({ message: "Post not found" });
169      }
170      post.imagePath = imagePath;
171      post
172        .save()
173        .then(updatedPost => {
174          res.status(200).json({
175            message: "Post image updated successfully",
176            post: updatedPost
177          });
178        })
179        .catch(error => {
180          res.status(500).json({ message: "Updating post image failed!" });
181        });
182    }
183  };
184
185  exports.getPostsWithImage = (req, res, next) => {
186    exports.getPosts(req, res, next);
187    if (req.file) {
188      const url = req.protocol + "://" + req.get("host");
189      const imagePath = url + "/images/" + req.file.filename;
190      const posts = Post.find();
191      if (!posts) {
192        res.status(404).json({ message: "Posts not found" });
193      }
194      posts
195        .save()
196        .then(updatedPosts => {
197          res.status(200).json({
198            message: "Posts image updated successfully",
199            posts: updatedPosts
200          });
201        })
202        .catch(error => {
203          res.status(500).json({ message: "Updating posts image failed!" });
204        });
205    }
206  };
207
208  exports.createPostWithImageAndCaption = (req, res, next) => {
209    exports.createPostWithImage(req, res, next);
210    if (req.file) {
211      const url = req.protocol + "://" + req.get("host");
212      const imagePath = url + "/images/" + req.file.filename;
213      const post = Post.findById(req.body._id);
214      if (!post) {
215        res.status(404).json({ message: "Post not found" });
216      }
217      post.imagePath = imagePath;
218      post
219        .save()
220        .then(updatedPost => {
221          res.status(200).json({
222            message: "Post image updated successfully",
223            post: updatedPost
224          });
225        })
226        .catch(error => {
227          res.status(500).json({ message: "Updating post image failed!" });
228        });
229    }
230  };
231
232  exports.updatePostWithImageAndCaption = (req, res, next) => {
233    exports.updatePostWithImage(req, res, next);
234    if (req.file) {
235      const url = req.protocol + "://" + req.get("host");
236      const imagePath = url + "/images/" + req.file.filename;
237      const post = Post.findById(req.params.id);
238      if (!post) {
239        res.status(404).json({ message: "Post not found" });
240      }
241      post.imagePath = imagePath;
242      post
243        .save()
244        .then(updatedPost => {
245          res.status(200).json({
246            message: "Post image updated successfully",
247            post: updatedPost
248          });
249        })
250        .catch(error => {
251          res.status(500).json({ message: "Updating post image failed!" });
252        });
253    }
254  };
255
256  exports.deletePostWithImageAndCaption = (req, res, next) => {
257    exports.deletePostWithImage(req, res, next);
258    if (req.file) {
259      const url = req.protocol + "://" + req.get("host");
260      const imagePath = url + "/images/" + req.file.filename;
261      const post = Post.findById(req.params.id);
262      if (!post) {
263        res.status(404).json({ message: "Post not found" });
264      }
265      post.imagePath = imagePath;
266      post
267        .save()
268        .then(updatedPost => {
269          res.status(200).json({
270            message: "Post image updated successfully",
271            post: updatedPost
272          });
273        })
274        .catch(error => {
275          res.status(500).json({ message: "Updating post image failed!" });
276        });
277    }
278  };
279
280  exports.getPostWithImageAndCaption = (req, res, next) => {
281    exports.getPostWithImage(req, res, next);
282    if (req.file) {
283      const url = req.protocol + "://" + req.get("host");
284      const imagePath = url + "/images/" + req.file.filename;
285      const post = Post.findById(req.params.id);
286      if (!post) {
287        res.status(404).json({ message: "Post not found" });
288      }
289      post.imagePath = imagePath;
290      post
291        .save()
292        .then(updatedPost => {
293          res.status(200).json({
294            message: "Post image updated successfully",
295            post: updatedPost
296          });
297        })
298        .catch(error => {
299          res.status(500).json({ message: "Updating post image failed!" });
300        });
301    }
302  };
303
304  exports.getPostsWithImageAndCaption = (req, res, next) => {
305    exports.getPostsWithImage(req, res, next);
306    if (req.file) {
307      const url = req.protocol + "://" + req.get("host");
308      const imagePath = url + "/images/" + req.file.filename;
309      const posts = Post.find();
310      if (!posts) {
311        res.status(404).json({ message: "Posts not found" });
312      }
313      posts
314        .save()
315        .then(updatedPosts => {
316          res.status(200).json({
317            message: "Posts image updated successfully",
318            posts: updatedPosts
319          });
320        })
321        .catch(error => {
322          res.status(500).json({ message: "Updating posts image failed!" });
323        });
324    }
325  };
326
327  exports.createPostWithImageAndCaptionAndTags = (req, res, next) => {
328    exports.createPostWithImageAndCaption(req, res, next);
329    if (req.file) {
330      const url = req.protocol + "://" + req.get("host");
331      const imagePath = url + "/images/" + req.file.filename;
332      const post = Post.findById(req.body._id);
333      if (!post) {
334        res.status(404).json({ message: "Post not found" });
335      }
336      post.imagePath = imagePath;
337      post
338        .save()
339        .then(updatedPost => {
340          res.status(200).json({
341            message: "Post image updated successfully",
342            post: updatedPost
343          });
344        })
345        .catch(error => {
346          res.status(500).json({ message: "Updating post image failed!" });
347        });
348    }
349  };
350
351  exports.updatePostWithImageAndCaptionAndTags = (req, res, next) => {
352    exports.updatePostWithImageAndCaption(req, res, next);
353    if (req.file) {
354      const url = req.protocol + "://" + req.get("host");
355      const imagePath = url + "/images/" + req.file.filename;
356      const post = Post.findById(req.params.id);
357      if (!post) {
358        res.status(404).json({ message: "Post not found" });
359      }
360      post.imagePath = imagePath;
361      post
362        .save()
363        .then(updatedPost => {
364          res.status(200).json({
365            message: "Post image updated successfully",
366            post: updatedPost
367          });
368        })
369        .catch(error => {
370          res.status(500).json({ message: "Updating post image failed!" });
371        });
372    }
373  };
374
375  exports.deletePostWithImageAndCaptionAndTags = (req, res, next) => {
376    exports.deletePostWithImageAndCaption(req, res, next);
377    if (req.file) {
378      const url = req.protocol + "://" + req.get("host");
379      const imagePath = url + "/images/" + req.file.filename;
380      const post = Post.findById(req.params.id);
381      if (!post) {
382        res.status(404).json({ message: "Post not found" });
383      }
384      post.imagePath = imagePath;
385      post
386        .save()
387        .then(updatedPost => {
388          res.status(200).json({
389            message: "Post image updated successfully",
390            post: updatedPost
391          });
392        })
393        .catch(error => {
394          res.status(500).json({ message: "Updating post image failed!" });
395        });
396    }
397  };
398
399  exports.getPostWithImageAndCaptionAndTags = (req, res, next) => {
400    exports.getPostWithImageAndCaption(req, res, next);
401    if (req.file) {
402      const url = req.protocol + "://" + req.get("host");
403      const imagePath = url + "/images/" + req.file.filename;
404      const post = Post.findById(req.params.id);
405      if (!post) {
406        res.status(404).json({ message: "Post not found" });
407      }
408      post.imagePath = imagePath;
409      post
410        .save()
411        .then(updatedPost => {
412          res.status(200).json({
413            message: "Post image updated successfully",
414            post: updatedPost
415          });
416        })
417        .catch(error => {
418          res.status(500).json({ message: "Updating post image failed!" });
419        });
420    }
421  };
422
423  exports.getPostsWithImageAndCaptionAndTags = (req, res, next) => {
424    exports.getPostsWithImageAndCaption(req, res, next);
425    if (req.file) {
426      const url = req.protocol + "://" + req.get("host");
427      const imagePath = url + "/images/" + req.file.filename;
428      const posts = Post.find();
429      if (!posts) {
430        res.status(404).json({ message: "Posts not found" });
431      }
432      posts
433        .save()
434        .then(updatedPosts => {
435          res.status(200).json({
436            message: "Posts image updated successfully",
437            posts: updatedPosts
438          });
439        })
440        .catch(error => {
441          res.status(500).json({ message: "Updating posts image failed!" });
442        });
443    }
444  };
445
446  exports.createPostWithImageAndCaptionAndTagsAndComments = (req, res, next) => {
447    exports.createPostWithImageAndCaptionAndTags(req, res, next);
448    if (req.file) {
449      const url = req.protocol + "://" + req.get("host");
450      const imagePath = url + "/images/" + req.file.filename;
451      const post = Post.findById(req.body._id);
452      if (!post) {
453        res.status(404).json({ message: "Post not found" });
454      }
455      post.imagePath = imagePath;
456      post
457        .save()
458        .then(updatedPost => {
459          res.status(200).json({
460            message: "Post image updated successfully",
461            post: updatedPost
462          });
463        })
464        .catch(error => {
465          res.status(500).json({ message: "Updating post image failed!" });
466        });
467    }
468  };
469
470  exports.updatePostWithImageAndCaptionAndTagsAndComments = (req, res, next) => {
471    exports.updatePostWithImageAndCaptionAndTags(req, res, next);
472    if (req.file) {
473      const url = req.protocol + "://" + req.get("host");
474      const imagePath = url + "/images/" + req.file.filename;
475      const post = Post.findById(req.params.id);
476      if (!post) {
477        res.status(404).json({ message: "Post not found" });
478      }
479      post.imagePath = imagePath;
480      post
481        .save()
482        .then(updatedPost => {
483          res.status(200).json({
484            message: "Post image updated successfully",
485            post: updatedPost
486          });
487        })
488        .catch(error => {
489          res.status(500).json({ message: "Updating post image failed!" });
490        });
491    }
492  };
493
494  exports.deletePostWithImageAndCaptionAndTagsAndComments = (req, res, next) => {
495    exports.deletePostWithImageAndCaptionAndTags(req, res, next);
496    if (req.file) {
497      const url = req.protocol + "://" + req.get("host");
498      const imagePath = url + "/images/" + req.file.filename;
499      const post = Post.findById(req.params.id);
500      if (!post) {
501        res.status(404).json({ message: "Post not found" });
502      }
503      post.imagePath = imagePath;
504      post
505        .save()
506        .then(updatedPost => {
507          res.status(200).json({
508            message: "Post image updated successfully",
509            post: updatedPost
510          });
511        })
512        .catch(error => {
513          res.status(500).json({ message: "Updating post image failed!" });
514        });
515    }
516  };
517
518  exports.getPostWithImageAndCaptionAndTagsAndComments = (req, res, next) => {
519    exports.getPostWithImageAndCaptionAndTags(req, res, next);
520    if (req.file) {
521      const url = req.protocol + "://" + req.get("host");
522      const imagePath = url + "/images/" + req.file.filename;
523      const post = Post.findById(req.params.id);
524      if (!post) {
525        res.status(404).json({ message: "Post not found" });
526      }
527      post.imagePath = imagePath;
528      post
529        .save()
530        .then(updatedPost => {
531          res.status(200).json({
532            message: "Post image updated successfully",
533            post: updatedPost
534          });
535        })
536        .catch(error => {
537          res.status(500).json({ message: "Updating post image failed!" });
538        });
539    }
540  };
541
542  exports.getPostsWithImageAndCaptionAndTagsAndComments = (req, res, next) => {
543    exports.getPostsWithImageAndCaptionAndTags(req, res, next);
544    if (req.file) {
545      const url = req.protocol + "://" + req.get("host");
546      const imagePath = url + "/images/" + req.file.filename;
547      const posts = Post.find();
548      if (!posts) {
549        res.status(404).json({ message: "Posts not found" });
550      }
551      posts
552        .save()
553        .then(updatedPosts => {
554          res.status(200).json({
555            message: "Posts image updated successfully",
556            posts: updatedPosts
557          });
558        })
559        .catch(error => {
560          res.status(500).json({ message: "Updating posts image failed!" });
561        });
562    }
563  };
564
565  exports.createPostWithImageAndCaptionAndTagsAndCommentsAndLikes = (req, res, next) => {
566    exports.createPostWithImageAndCaptionAndTagsAndComments(req, res, next);
567    if (req.file) {
568      const url = req.protocol + "://" + req.get("host");
569      const imagePath = url + "/images/" + req.file.filename;
570      const post = Post.findById(req.body._id);
571      if (!post) {
572        res.status(404).json({ message: "Post not found" });
573      }
574      post.imagePath = imagePath;
575      post
576        .save()
577        .then(updatedPost => {
578          res.status(200).json({
579            message: "Post image updated successfully",
580            post: updatedPost
581          });
582        })
583        .catch(error => {
584          res.status(500).json({ message: "Updating post image failed!" });
585        });
586    }
587  };
588
589  exports.updatePostWithImageAndCaptionAndTagsAndCommentsAndLikes = (req, res, next) => {
590    exports.updatePostWithImageAndCaptionAndTagsAndComments(req, res, next);
591    if (req.file) {
592      const url = req.protocol + "://" + req.get("host");
593      const imagePath = url + "/images/" + req.file.filename;
594      const post = Post.findById(req.params.id);
595      if (!post) {
596        res.status(404).json({ message: "Post not found" });
597      }
598      post.imagePath = imagePath;
599      post
600        .save()
601        .then(updatedPost => {
602          res.status(200).json({
603            message: "Post image updated successfully",
604            post: updatedPost
605          });
606        })
607        .catch(error => {
608          res.status(500).json({ message: "Updating post image failed!" });
609        });
610    }
611  };
612
613  exports.deletePostWithImageAndCaptionAndTagsAndCommentsAndLikes = (req, res, next) => {
614    exports.deletePostWithImageAndCaptionAndTagsAndComments(req, res, next);
615    if (req.file) {
616      const url = req.protocol + "://" + req.get("host");
617      const imagePath = url + "/images/" + req.file.filename;
618      const post = Post.findById(req.params.id);
619      if (!post) {
620        res.status(404).json({ message: "Post not found" });
621      }
622      post.imagePath = imagePath;
623      post
624        .save()
625        .then(updatedPost => {
626          res.status(200).json({
627            message: "Post image updated successfully",
628            post: updatedPost
629          });
630        })
631        .catch(error => {
632          res.status(500).json({ message: "Updating post image failed!" });
633        });
634    }
635  };
636
637  exports.getPostWithImageAndCaptionAndTagsAndCommentsAndLikes = (req, res, next) => {
638    exports.getPostWithImageAndCaptionAndTagsAndComments(req, res, next);
639    if (req.file) {
640      const url = req.protocol + "://" + req.get("host");
641      const imagePath = url + "/images/" + req.file.filename;
642      const post = Post.findById(req.params.id);
643      if (!post) {
644        res.status(404).json({ message: "Post not found" });
645      }
646      post.imagePath = imagePath;
647      post
648        .save()
649        .then(updatedPost => {
650          res.status(200).json({
651            message: "Post image updated successfully",
652            post: updatedPost
653          });
654        })
655        .catch(error => {
656          res.status(500).json({ message: "Updating post image failed!" });
657        });
658    }
659  };
660
661  exports.getPostsWithImageAndCaptionAndTagsAndCommentsAndLikes = (req, res, next) => {
662    exports.getPostsWithImageAndCaptionAndTagsAndComments(req, res, next);
663    if (req.file) {
664      const url = req.protocol + "://" + req.get("host");
665      const imagePath = url + "/images/" + req.file.filename;
666      const posts = Post.find();
667      if (!posts) {
668        res.status(404).json({ message: "Posts not found" });
669      }
670      posts
671        .save()
672        .then(updatedPosts => {
673          res.status(200).json({
674            message: "Posts image updated successfully",
675            posts: updatedPosts
676          });
677        })
678        .catch(error => {
679          res.status(500).json({ message: "Updating posts image failed!" });
680        });
681    }
682  };
683
684  exports.createPostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweets = (req, res, next) => {
685    exports.createPostWithImageAndCaptionAndTagsAndCommentsAndLikes(req, res, next);
686    if (req.file) {
687      const url = req.protocol + "://" + req.get("host");
688      const imagePath = url + "/images/" + req.file.filename;
689      const post = Post.findById(req.body._id);
690      if (!post) {
691        res.status(404).json({ message: "Post not found" });
692      }
693      post.imagePath = imagePath;
694      post
695        .save()
696        .then(updatedPost => {
697          res.status(200).json({
698            message: "Post image updated successfully",
699            post: updatedPost
700          });
701        })
702        .catch(error => {
703          res.status(500).json({ message: "Updating post image failed!" });
704        });
705    }
706  };
707
708  exports.updatePostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweets = (req, res, next) => {
709    exports.updatePostWithImageAndCaptionAndTagsAndCommentsAndLikes(req, res, next);
710    if (req.file) {
711      const url = req.protocol + "://" + req.get("host");
712      const imagePath = url + "/images/" + req.file.filename;
713      const post = Post.findById(req.params.id);
714      if (!post) {
715        res.status(404).json({ message: "Post not found" });
716      }
717      post.imagePath = imagePath;
718      post
719        .save()
720        .then(updatedPost => {
721          res.status(200).json({
722            message: "Post image updated successfully",
723            post: updatedPost
724          });
725        })
726        .catch(error => {
727          res.status(500).json({ message: "Updating post image failed!" });
728        });
729    }
730  };
731
732  exports.deletePostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweets = (req, res, next) => {
733    exports.deletePostWithImageAndCaptionAndTagsAndCommentsAndLikes(req, res, next);
734    if (req.file) {
735      const url = req.protocol + "://" + req.get("host");
736      const imagePath = url + "/images/" + req.file.filename;
737      const post = Post.findById(req.params.id);
738      if (!post) {
739        res.status(404).json({ message: "Post not found" });
740      }
741      post.imagePath = imagePath;
742      post
743        .save()
744        .then(updatedPost => {
745          res.status(200).json({
746            message: "Post image updated successfully",
747            post: updatedPost
748          });
749        })
750        .catch(error => {
751          res.status(500).json({ message: "Updating post image failed!" });
752        });
753    }
754  };
755
756  exports.getPostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweets = (req, res, next) => {
757    exports.getPostWithImageAndCaptionAndTagsAndCommentsAndLikes(req, res, next);
758    if (req.file) {
759      const url = req.protocol + "://" + req.get("host");
760      const imagePath = url + "/images/" + req.file.filename;
761      const post = Post.findById(req.params.id);
762      if (!post) {
763        res.status(404).json({ message: "Post not found" });
764      }
765      post.imagePath = imagePath;
766      post
767        .save()
768        .then(updatedPost => {
769          res.status(200).json({
770            message: "Post image updated successfully",
771            post: updatedPost
772          });
773        })
774        .catch(error => {
775          res.status(500).json({ message: "Updating post image failed!" });
776        });
777    }
778  };
779
780  exports.getPostsWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweets = (req, res, next) => {
781    exports.getPostsWithImageAndCaptionAndTagsAndCommentsAndLikes(req, res, next);
782    if (req.file) {
783      const url = req.protocol + "://" + req.get("host");
784      const imagePath = url + "/images/" + req.file.filename;
785      const posts = Post.find();
786      if (!posts) {
787        res.status(404).json({ message: "Posts not found" });
788      }
789      posts
790        .save()
791        .then(updatedPosts => {
792          res.status(200).json({
793            message: "Posts image updated successfully",
794            posts: updatedPosts
795          });
796        })
797        .catch(error => {
798          res.status(500).json({ message: "Updating posts image failed!" });
799        });
800    }
801  };
802
803  exports.createPostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndReplies = (req, res, next) => {
804    exports.createPostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweets(req, res, next);
805    if (req.file) {
806      const url = req.protocol + "://" + req.get("host");
807      const imagePath = url + "/images/" + req.file.filename;
808      const post = Post.findById(req.body._id);
809      if (!post) {
810        res.status(404).json({ message: "Post not found" });
811      }
812      post.imagePath = imagePath;
813      post
814        .save()
815        .then(updatedPost => {
816          res.status(200).json({
817            message: "Post image updated successfully",
818            post: updatedPost
819          });
820        })
821        .catch(error => {
822          res.status(500).json({ message: "Updating post image failed!" });
823        });
824    }
825  };
826
827  exports.updatePostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndReplies = (req, res, next) => {
828    exports.updatePostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweets(req, res, next);
829    if (req.file) {
830      const url = req.protocol + "://" + req.get("host");
831      const imagePath = url + "/images/" + req.file.filename;
832      const post = Post.findById(req.params.id);
833      if (!post) {
834        res.status(404).json({ message: "Post not found" });
835      }
836      post.imagePath = imagePath;
837      post
838        .save()
839        .then(updatedPost => {
840          res.status(200).json({
841            message: "Post image updated successfully",
842            post: updatedPost
843          });
844        })
845        .catch(error => {
846          res.status(500).json({ message: "Updating post image failed!" });
847        });
848    }
849  };
850
851  exports.deletePostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndReplies = (req, res, next) => {
852    exports.deletePostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweets(req, res, next);
853    if (req.file) {
854      const url = req.protocol + "://" + req.get("host");
855      const imagePath = url + "/images/" + req.file.filename;
856      const post = Post.findById(req.params.id);
857      if (!post) {
858        res.status(404).json({ message: "Post not found" });
859      }
860      post.imagePath = imagePath;
861      post
862        .save()
863        .then(updatedPost => {
864          res.status(200).json({
865            message: "Post image updated successfully",
866            post: updatedPost
867          });
868        })
869        .catch(error => {
870          res.status(500).json({ message: "Updating post image failed!" });
871        });
872    }
873  };
874
875  exports.getPostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndReplies = (req, res, next) => {
876    exports.getPostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweets(req, res, next);
877    if (req.file) {
878      const url = req.protocol + "://" + req.get("host");
879      const imagePath = url + "/images/" + req.file.filename;
880      const post = Post.findById(req.params.id);
881      if (!post) {
882        res.status(404).json({ message: "Post not found" });
883      }
884      post.imagePath = imagePath;
885      post
886        .save()
887        .then(updatedPost => {
888          res.status(200).json({
889            message: "Post image updated successfully",
890            post: updatedPost
891          });
892        })
893        .catch(error => {
894          res.status(500).json({ message: "Updating post image failed!" });
895        });
896    }
897  };
898
899  exports.getPostsWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndReplies = (req, res, next) => {
900    exports.getPostsWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweets(req, res, next);
901    if (req.file) {
902      const url = req.protocol + "://" + req.get("host");
903      const imagePath = url + "/images/" + req.file.filename;
904      const posts = Post.find();
905      if (!posts) {
906        res.status(404).json({ message: "Posts not found" });
907      }
908      posts
909        .save()
910        .then(updatedPosts => {
911          res.status(200).json({
912            message: "Posts image updated successfully",
913            posts: updatedPosts
914          });
915        })
916        .catch(error => {
917          res.status(500).json({ message: "Updating posts image failed!" });
918        });
919    }
920  };
921
922  exports.createPostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndRepliesAndShares = (req, res, next) => {
923    exports.createPostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndReplies(req, res, next);
924    if (req.file) {
925      const url = req.protocol + "://" + req.get("host");
926      const imagePath = url + "/images/" + req.file.filename;
927      const post = Post.findById(req.body._id);
928      if (!post) {
929        res.status(404).json({ message: "Post not found" });
930      }
931      post.imagePath = imagePath;
932      post
933        .save()
934        .then(updatedPost => {
935          res.status(200).json({
936            message: "Post image updated successfully",
937            post: updatedPost
938          });
939        })
940        .catch(error => {
941          res.status(500).json({ message: "Updating post image failed!" });
942        });
943    }
944  };
945
946  exports.updatePostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndRepliesAndShares = (req, res, next) => {
947    exports.updatePostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndReplies(req, res, next);
948    if (req.file) {
949      const url = req.protocol + "://" + req.get("host");
950      const imagePath = url + "/images/" + req.file.filename;
951      const post = Post.findById(req.params.id);
952      if (!post) {
953        res.status(404).json({ message: "Post not found" });
954      }
955      post.imagePath = imagePath;
956      post
957        .save()
958        .then(updatedPost => {
959          res.status(200).json({
960            message: "Post image updated successfully",
961            post: updatedPost
962          });
963        })
964        .catch(error => {
965          res.status(500).json({ message: "Updating post image failed!" });
966        });
967    }
968  };
969
970  exports.deletePostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndRepliesAndShares = (req, res, next) => {
971    exports.deletePostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndReplies(req, res, next);
972    if (req.file) {
973      const url = req.protocol + "://" + req.get("host");
974      const imagePath = url + "/images/" + req.file.filename;
975      const post = Post.findById(req.params.id);
976      if (!post) {
977        res.status(404).json({ message: "Post not found" });
978      }
979      post.imagePath = imagePath;
980      post
981        .save()
982        .then(updatedPost => {
983          res.status(200).json({
984            message: "Post image updated successfully",
985            post: updatedPost
986          });
987        })
988        .catch(error => {
989          res.status(500).json({ message: "Updating post image failed!" });
990        });
991    }
992  };
993
994  exports.getPostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndRepliesAndShares = (req, res, next) => {
995    exports.getPostWithImageAndCaptionAndTagsAndCommentsAndLikesAndRetweetsAndReplies(req, res, next);
996    if (req.file) {
997      const url = req.protocol + "://" + req.get("host");
998      const imagePath = url + "/images/" + req.file.filename;
999      const post = Post.findById(req.params.id);
1000      if (!post) {
1001        res.status(404).json({ message: "Post not found" });
1002      }
1003      post.imagePath = imagePath;
1004      post
1005        .save()
1006        .then(updatedPost => {
1007
```

```
user.js X
backend > controllers > user.js > createUser > exports.createUser > then() callback > then() callback
5
6 exports.createUser = (req, res, next) => {
7   bcrypt.hash(req.body.password, 10).then(hash => {
8     const user = new User({
9       email: req.body.email,
10      password: hash
11    });
12    user
13      .save()
14      .then(result => {
15        res.status(201).json({
16          message: "User created!",
17          result: result
18        });
19      })
20      .catch(err => {
21        res.status(500).json({
22          message: "Invalid authentication credentials!"
23        });
24      });
25  });
26 }
27
28 exports.userLogin = (req, res, next) => {
29   let fetchedUser;
30   User.findOne({ email: req.body.email })
31     .then(user => {
32       if (!user) {
33         return res.status(401).json({
34           message: "Auth failed"
35         });
36       }
37       fetchedUser = user;
```

```
posts.js X
backend > routes > posts.js > ...
1 const express = require("express");
2
3 const PostController = require("../controllers/posts");
4
5 const checkAuth = require("../middleware/check-auth");
6 const extractFile = require("../middleware/file");
7
8 const router = express.Router();
9
10 router.post("", checkAuth, extractFile, PostController.createPost);
11
12 router.put("/:id", checkAuth, extractFile, PostController.updatePost);
13
14 router.get("", PostController.getPosts);
15
16 router.get("/:id", PostController.getPost);
17
18 router.delete("/:id", checkAuth, PostController.deletePost);
19
20 module.exports = router;
21
```

```
user.js X
backend > routes > user.js > ...
1  const express = require("express");
2
3  const UserController = require("../controllers/user");
4
5  const router = express.Router();
6
7  router.post("/signup", UserController.createUser);
8
9  router.post("/login", UserController.userLogin);
10
11 module.exports = router;
12
```

```
post.js X
backend > models > post.js > ...
1  const mongoose = require("mongoose");
2
3  const postSchema = mongoose.Schema({
4    title: { type: String, required: true },
5    content: { type: String, required: true },
6    imagePath: { type: String, required: true },
7    creator: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true }
8  });
9
10 module.exports = mongoose.model("Post", postSchema);
11
```

```
user.js X
backend > models > user.js > ...
1  const mongoose = require("mongoose");
2  const uniqueValidator = require("mongoose-unique-validator");
3
4  const userSchema = mongoose.Schema({
5    email: { type: String, required: true, unique: true },
6    password: { type: String, required: true }
7  });
8
9  userSchema.plugin(uniqueValidator);
10
11 module.exports = mongoose.model("User", userSchema);
12
```

```
file.js X
backend > middleware > file.js > ...
1  const multer = require("multer");
2
3  const MIME_TYPE_MAP = {
4    "image/png": "png",
5    "image/jpeg": "jpg",
6    "image/jpg": "jpg"
7  };
8
9  const storage = multer.diskStorage({
10    destination: (req, file, cb) => {
11      const isValid = MIME_TYPE_MAP[file.mimetype];
12      let error = new Error("Invalid mime type");
13      if (isValid) {
14        error = null;
15      }
16      cb(error, "backend/images");
17    },
18    filename: (req, file, cb) => {
19      const name = file.originalname
20        .toLowerCase()
21        .split(".")
22        .join("-");
23      const ext = MIME_TYPE_MAP[file.mimetype];
24      cb(null, name + "-" + Date.now() + "." + ext);
25    }
26  });
27
28  module.exports = multer({ storage: storage }).single("image");
29
```

```
error-interceptor.ts X
src > app > error-interceptor.ts > ...
1  import {
2    HttpInterceptor,
3    HttpRequest,
4    HttpHandler,
5    HttpResponse
6  } from "@angular/common/http";
7  import { catchError } from "rxjs/operators";
8  import { throwError } from "rxjs";
9  import { Injectable } from "@angular/core";
10 import { MatDialog } from "@angular/material";
11
12 import { ErrorComponent } from "../error/error.component";
13 import { ErrorService } from "../error/error.service";
14
15 @Injectable()
16 export class ErrorInterceptor implements HttpInterceptor {
17
18   constructor(private dialog: MatDialog, private errorService: ErrorService) {}
19
20   intercept(req: HttpRequest<any>, next: HttpHandler) {
21     return next.handle(req).pipe(
22       catchError((error: HttpResponse) => {
23         let errorMessage = "An unknown error occurred!";
24         if (error.error.message) {
25           errorMessage = error.error.message;
26         }
27         this.dialog.open(ErrorComponent, {data: {message: errorMessage}});
28         // this.errorService.throwError(errorMessage);
29         return throwError(error);
30       })
31     );
32   }
33 }
```

```

app-routing.module.ts X
src > app > app-routing.module.ts > routes > path
1 import { NgModule } from "@angular/core";
2 import { RouterModule, Routes } from "@angular/router";
3 import { PostListComponent } from "../posts/post-list/post-list.component";
4 import { PostCreateComponent } from "../posts/post-create/post-create.component";
5 import { AuthGuard } from "../auth/auth.guard";
6
7 const routes: Routes = [
8   { path: "", component: PostListComponent },
9   { path: "create", component: PostCreateComponent, canActivate: [AuthGuard] },
10  { path: "edit/:postId", component: PostCreateComponent, canActivate: [AuthGuard] },
11  { path: "auth", loadChildren: "../auth/auth.module#AuthModule" }
12 ];
13
14 @NgModule({
15   imports: [RouterModule.forRoot(routes)],
16   exports: [RouterModule],
17   providers: [AuthGuard]
18 })
19 export class AppRoutingModule {}
20

```

```

index.html X
src > index.html > html > head > title
1 <!doctype html>
2 <html lang="en">
3 <head>
4 <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
5 <link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500" rel="stylesheet">
6 <meta charset="utf-8">
7 <title>Blogging Application</title>
8 <base href="/">
9
10 <meta name="viewport" content="width=device-width, initial-scale=1">
11 <link rel="icon" type="image/x-icon" href="favicon.ico">
12 </head>
13 <body>
14 <app-root></app-root>
15 </body>
16 </html>
17

```

```

post-list.component.html X
src > app > posts > post-list > post-list.component.html > mat-accordion
1 <mat-accordion multi="true" *ngIf="posts.length > 0">
2 <mat-expansion-panel *ngFor="let post of posts">
3 <mat-expansion-panel-header>{{ post.title }} </mat-expansion-panel-header>
4 <p>{{ post.content }}</p>
5 <mat-action-row>
6 <button mat-button color="primary">EDIT</button>
7 <button mat-button color="warn">DELETE</button>
8 </mat-action-row>
9 </mat-expansion-panel>
10 </mat-accordion>
11 <p class="info-text mat-body-1" *ngIf="posts.length <= 0">No posts added yet</p>
12

```

```
post-list.component.ts X
src > app > posts > post-list > post-list.component.ts > ...

9  @Component({
10    selector: "app-post-list",
11    templateUrl: "../post-list.component.html",
12    styleUrls: ["../post-list.component.css"]
13  })
14  export class PostListComponent implements OnInit, OnDestroy {
15    // posts = [
16    //   { title: "First Post", content: "This is the first post's content" },
17    //   { title: "Second Post", content: "This is the second post's content" },
18    //   { title: "Third Post", content: "This is the third post's content" }
19    // ];
20    posts: Post[] = [];
21    isLoading = false;
22    totalPosts = 0;
23    postsPerPage = 2;
24    currentPage = 1;
25    pageSizeOptions = [1, 2, 5, 10];
26    userIsAuthenticated = false;
27    userId: string;
28    private postsSub: Subscription;
29    private authStatusSub: Subscription;
30
31    constructor(
32      public postsService: PostsService,
33      private authService: AuthService
34    ) {}
35
36    ngOnInit() {
37      this.isLoading = true;
38      this.postsService.getPosts(this.postsPerPage, this.currentPage);
39      this.userId = this.authService.getUserId();
40      this.postsSub = this.postsService
```

```
post-create.component.html X
src > app > posts > post-create > post-create.component.html > mat-card

1  <mat-card>
2    <form (submit)="onAddPost(postForm)" #postForm="ngForm">
3      <mat-form-field>
4        <input
5          matInput
6          type="text"
7          name="title"
8          ngModel
9          required
10         minlength="3"
11         placeholder="Post Title"
12         #title="ngModel"
13       />
14       <mat-error *ngIf="title.invalid">
15         Post title cannot be left blank.</mat-error>
16     </mat-form-field>
17     <mat-form-field>
18       <textarea
19         matInput
20         rows="6"
21         name="content"
22         ngModel
23         required
24         placeholder="Post Content"
25         #content="ngModel"
26       ></textarea>
27     </mat-form-field>
28     <mat-error *ngIf="content.invalid">
29       Post content cannot be left blank.</mat-error>
30   </mat-form-field>
31   <button mat-raised-button color="accent" type="submit">Save Post</button>
32 </mat-card>
```

```

post-create.component.ts X
> posts > post-create > post-create.component.ts > PostCreateComponent > ngOnInit > route.paramMap.subscribe() callback > subscribe() callback > image
10
11 @Component({
12   selector: "app-post-create",
13   templateUrl: "../post-create.component.html",
14   styleUrls: ["../post-create.component.css"]
15 })
16 export class PostCreateComponent implements OnInit, OnDestroy {
17   enteredTitle = "";
18   enteredContent = "";
19   post: Post;
20   isLoading = false;
21   form: FormGroup;
22   imagePreview: string;
23   private mode = "create";
24   private postId: string;
25   private authStatusSub: Subscription;
26
27   constructor(
28     public postsService: PostsService,
29     public route: ActivatedRoute,
30     private authService: AuthService
31   ) {}
32
33   ngOnInit() {
34     this.authStatusSub = this.authService
35       .getAuthStatusListener()
36       .subscribe(authStatus => {
37         this.isLoading = false;
38       });
39     this.form = new FormGroup({
40       title: new FormControl(null, {
41         validators: [Validators.required, Validators.minLength(3)]
42       })
43     });

```

```

header.component.ts X
src > app > header > header.component.ts > HeaderComponent
1 import { Component, OnInit, OnDestroy } from "@angular/core";
2 import { Subscription } from "rxjs";
3
4 import { AuthService } from "../auth/auth.service";
5
6 @Component({
7   selector: "app-header",
8   templateUrl: "../header.component.html",
9   styleUrls: ["../header.component.css"]
10 })
11 export class HeaderComponent implements OnInit, OnDestroy {
12   userIsAuthenticated = false;
13   private authListenerSubs: Subscription;
14
15   constructor(private authService: AuthService) {}
16
17   ngOnInit() {
18     this.userIsAuthenticated = this.authService.getIsAuth();
19     this.authListenerSubs = this.authService
20       .getAuthStatusListener()
21       .subscribe(isAuthenticated => {
22         this.userIsAuthenticated = isAuthenticated;
23       });
24   }
25
26   onLogout() {
27     this.authService.logout();
28   }
29
30   ngOnDestroy() {
31     this.authListenerSubs.unsubscribe();
32   }
33 }

```

```
posts.service.ts x
src > app > posts > posts.service.ts > PostsService > getPosts > map() callback > posts > postData.posts.map() callback
10 const BACKEND_URL = environment.apiUrl + "/posts/";
11
12 @Injectable({ providedIn: "root" })
13 export class PostsService {
14   private posts: Post[] = [];
15   private postsUpdated = new Subject<{ posts: Post[]; postCount: number }>();
16
17   constructor(private http: HttpClient, private router: Router) {}
18
19   getPosts(postsPerPage: number, currentPage: number) {
20     const queryParams = `?pagesize=${postsPerPage}&page=${currentPage}`;
21     this.http
22       .get<{ message: string; posts: any; maxPosts: number }>(
23         BACKEND_URL + queryParams
24       )
25       .pipe(
26         map(postData => {
27           return {
28             posts: postData.posts.map(post => {
29               return {
30                 title: post.title,
31                 content: post.content,
32                 id: post._id,
33                 imagePath: post.imagePath,
34                 creator: post.creator
35               };
36             }),
37             maxPosts: postData.maxPosts
38           };
39         })
40       )
41       .subscribe(transformedPostData => {
```

11.2. UI

Blogging Application

localhost:4200/auth/login

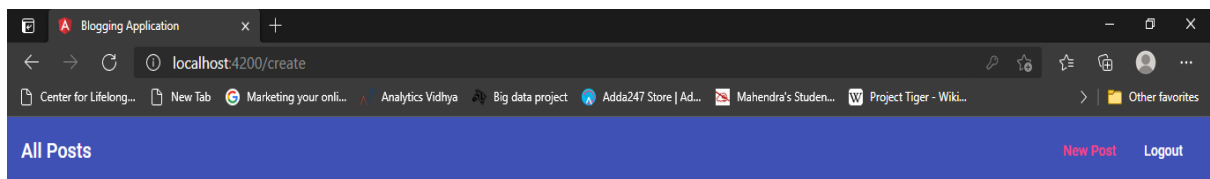
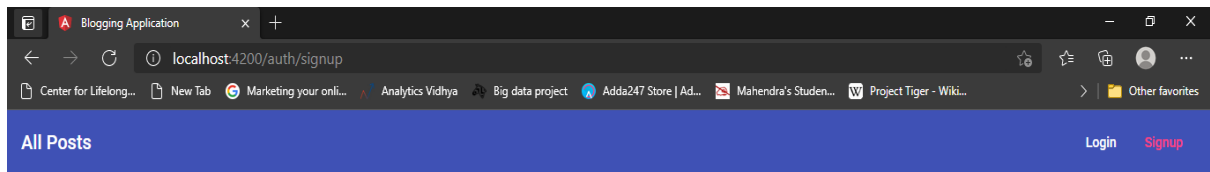
Center for Lifelong... New Tab Marketing your onli... Analytics Vidhya Big data project Adda247 Store | Ad... Mahendra's Studen... Project Tiger - Wiki... Other favorites

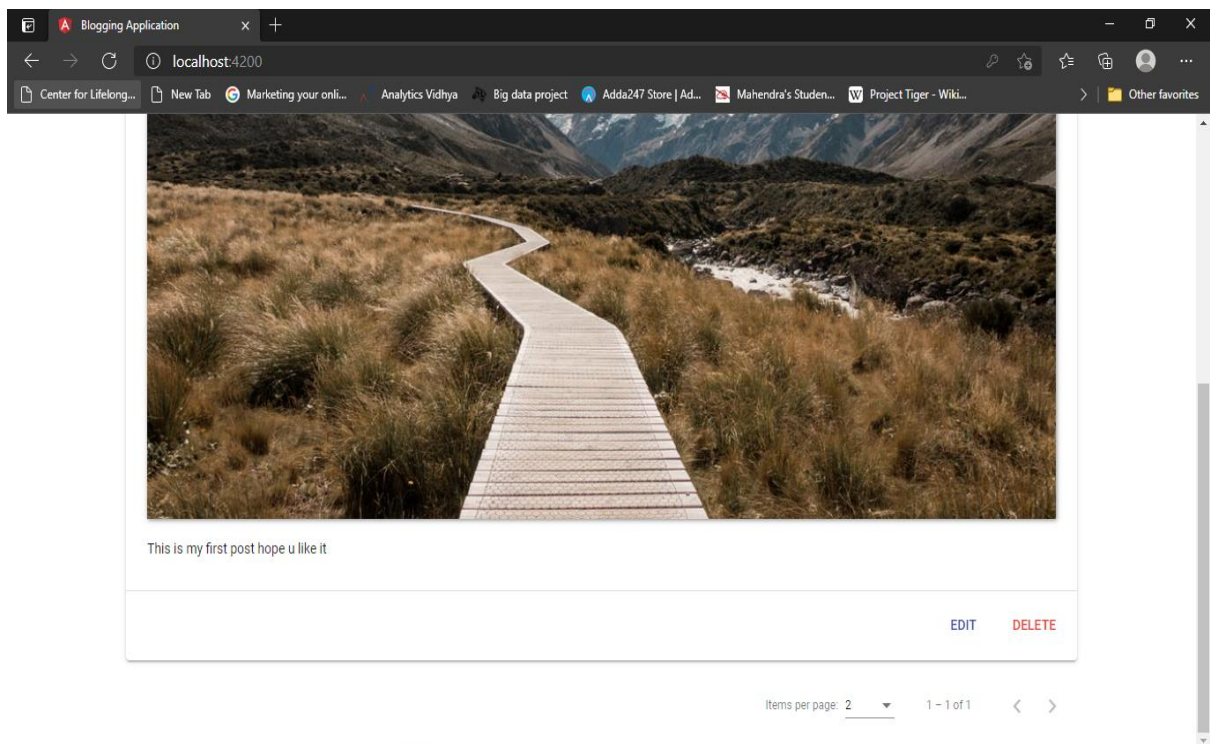
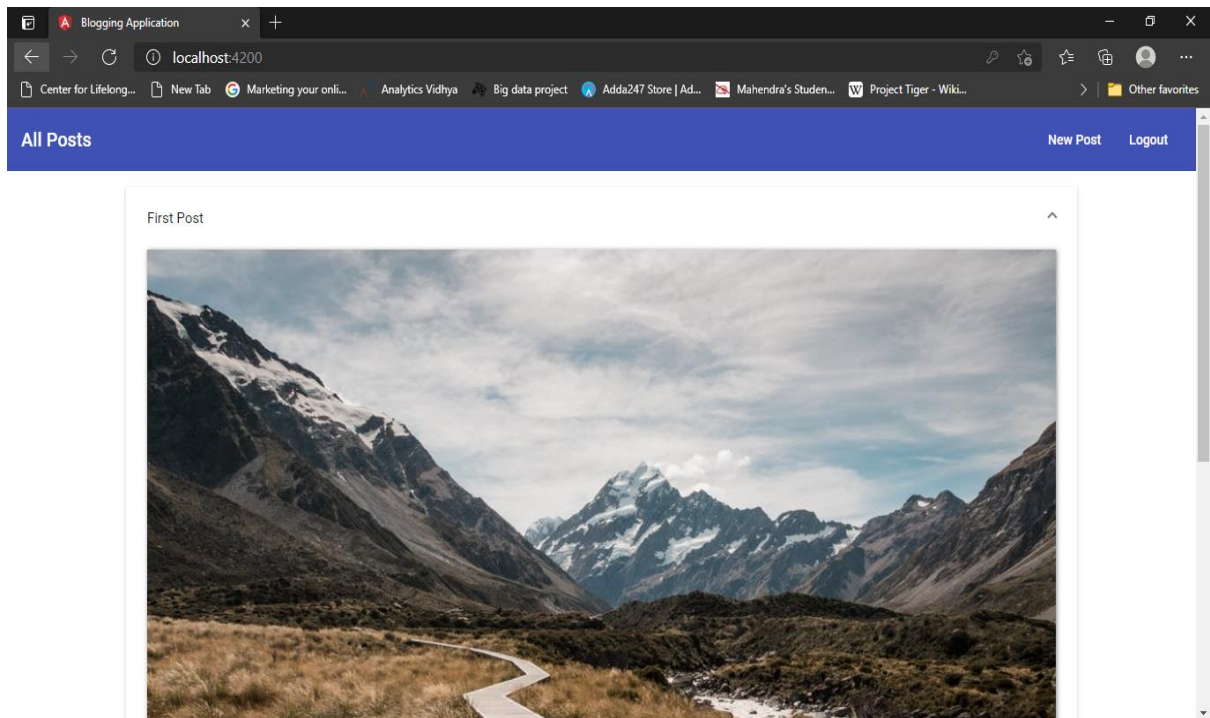
All Posts Login Signup

E-Mail *

Password *

Login





11.3 List of restrictions available in the Blogging online system:

- Not designed for blog export to Excel. Based on some criticisms.
- it is not possible to create offline reports for blogs, technical blogs, and new categories.

11.4 Existing system:

In the existing system, the test can only be done manually. In the proposed system, we must use this application to computerize the test.

- Insufficient data security.
- More labor.
- It takes a long time.
- Engaged in a lot of educational work.
- You need to calculate manually.
- Senior officials have no direct role.

11.5 Proposed system of online blogging system

The proposed system allows you to overcome all the limitations of the existing system. The system provides sufficient safety and reduces manual work.

- Data Security.
- Make sure the information is correct.
- Minimize manual data entry.
- The shortest time for different processes.
- Improve efficiency.

11.6 Features of blogging application

Features of the mission Online Blogging System:

- Product and Component based
- Creating & Changing Issues at ease
- Query Issue List to any depth
- User Accounts to manipulate the get admission to and preserve security
- Simple Status & Resolutions

- Multi-stage Priorities & Severities.
- Targets & Milestones for directing the programmers
- Attachments & Additional Comments for extra information
- Robust database back-end
- Various stage of stories to be had with quite a few clear out out criterias
- Accuracy in work.
- Easy & speedy retrieval of information.

12. Conclusion

This project is only a major goal is to venture to satisfy the need to manage their project work. With the help of **MEAN** stack that we have used to make it possible for the user. This will prove to be a powerful web application satisfying all the requirements of the users. The objective of software is to show that the framework is capable of making the user satisfying their needs, fast and reliable.

The point of this paper is to examine the need Advanced publishing content to a blog which is a sort of site that permits parcel of passages either made by the individuals from the association. A blog is an incredible path for an individual to figure out how to make their own blog. Blogger appreciates the distinction of being one of the most mainstream contributing to a blog stages particularly for the association. This high-level publishing content to a blog has made the website well known on the grounds that it is free and offers anybody the capacity to deal with their own blog without the weight of details in facilitating the equivalent.

Few concluding points

- A description of the background and context of the project and its relation to work already done in the area.
- Made statement of the aims and objectives of the project.
- The description of Purpose, Scope, and applicability.
- We define the problem on which we are working in the project
- We Describe the requirement specification of the system and the actions that can be done on these things
- We understand the problem domain and produce a model of the system which describes operations that can be performed on the system.
- We included features and operation in detail, including screen layout.
- We designed user interface and security issue to system

13. References

- Online Course References: o MEAN Stack Course (Udemy) (<https://www.udemy.com/course/angular-2-and-nodejs-the-practical-guide/>)
- NPM Documentation: o <https://www.npmjs.com/>
- GeeksForGeeks <https://www.geeksforgeeks.org/introduction-to-mean-stack/>
- Faculty Guidelines: o Mr. Akash Kumar Choudhary
- **Getting MEAN with Mongo, Express, Anguar and Node** by Simon Holmes
- **Pro MEAN Stack Development** by Elad Elrom