Secure Web Application: MERN Stack Forum

Submitted by

A ALI AHAMMED PRIOM

ID: 183-35-376

Department of Software Engineering

Daffodil International University

Supervised by

Dr. MARZIA AHMED

Assistant Professor

Department of Software Engineering

Daffodil International University

DECLARATION

I hereby declare that; **Secure Web Application: MERN stack Forum** project has been done by me under the supervision of **Dr. Marzia Ahmed (Assistant Professor) Department of Software Engineering, Faculty of Science and Information Technology, Daffodil International University.** I also declare that neither this project nor any part of this project has been submitted elsewhere for the award of any degree or diploma.

15.09.25

A Ali Ahammed Priom
ID: 183-35-376
Department of Software Engineering
Daffodil International University

Certified by

Marzia
15.9.25

Dr. Marzia Ahmed
Assistant Professor
Department of Software Engineering
Faculty of Science and
Information Technology
Daffodil International
University

APPROVAL

This project titled on "**Secure Web Application: MERN Stack Forum**", submitted by **A Ali Ahammed Priom (ID: 183-35-376)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

BOARD OF EXAMINERS

Fazla Elahe

Chairman

Dr. Md. Fazla Elahe
Assistant Professor & Associate Head
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Marzia
13.9.25

Internal Examiner 1

Dr. Marzia Ahmed
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

13.09.2025

Internal Examiner 2

Dr. Shabnom Mustary
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

13.9.25

External Examiner

Mohammod Abul Kashem
Professor
Department of Computer Science and Engineering
Dhaka University of Engineering & Technology, Gazipur.

ACKNOWLEDGEMENT

I want to start by expressing my deep gratitude to Almighty Allah. I am grateful for His blessings, which gave me the strength, focus, and determination to complete this project.

I am truly thankful to my parents for their constant encouragement and support. Their faith in my abilities has motivated me throughout my academic journey and this achievement.

Then I really grateful to my research supervisor, Dr. Marzia Ahmed who guided me throughout the whole research activities.

I want to extend my heartfelt appreciation to Prof. Dr. Imran Mahmud, Head of the Department of Software Engineering at Daffodil International University. His guidance and ongoing motivation were essential during this project.

I appreciate all the faculty members and lab assistants in the Department of Software Engineering for their help and technical support throughout the development process.

Finally, I am grateful to my classmates and friends for their collaboration, suggestions, and encouragement during this journey. Their support, whether big or small, has meant a lot to me.

ABSTRACT

In today's digital world, people want spaces where they can share ideas, ask questions, and connect with others easily. The **Secure Web Application: MERN Stack Forum** is built to make that possible—a modern, all-in-one platform that replaces old, scattered forums and social media threads with a smooth, user-friendly experience.

For users, it's simple to sign up, start discussions, comment on posts, and follow topics they care about. They can keep track of their activity, engage with others, and build a presence in the community. Moderators have the tools they need to keep conversations healthy, enforce guidelines, and manage posts efficiently. Administrators can oversee everything—from managing user roles and moderating content to tracking activity and customizing settings to fit the community's needs.

This isn't just a forum—it's a space that encourages learning, collaboration, and meaningful conversations. Built with the MERN stack, it's fast, scalable, and works seamlessly on computers, tablets, or phones.

The forum aims to make online communities more engaging, reduce clutter and spam, and give everyone—from casual users to administrators—a better, smoother experience. And with room for future improvements like real-time chat and AI-powered moderation, it's designed to grow with its community.

Table of Contents

Chapter 1: Introductions

1.1 Project Planning and Initiation

Secure Web Application: MERN Stack Forum is initiated to address the growing need for secure, interactive, and community-driven discussion platforms. The aim was to create a fully responsive forum using the MERN stack (MongoDB, Express, React, Node.js) with strong security features, real-time interaction, and role-based access controls. The planning involved selecting modern tools, defining milestones, and incorporating best practices for secure web development.

1.2 Feasibility Study

Phase 1: Preliminary Analysis & Project Scope

Project Gap

Category	Gap	Real-World Example
Verification & Validation	Lack of Input Validation and User Verification	T-Mobile Data Breach (2021) - Malicious input led to unauthorized access and exposure of user data.
	No Password and User Verification	Equifax Data Breach (2017) - Inadequate password verification and lack of user validation led to massive data breach.
	Insecure JWT Implementation (Token Hijacking)	Uber's JWT Token Hijacking (2022) - Session tokens were stolen, leading to unauthorized access.
Authentication & Authorization	Lack of Admin and User Verification for JWT and API	Capital One Data Breach (2019) - Lack of proper verification of user and admin roles in API led to unauthorized data access.
Deployment & Production Security	Exposed Environment Variables	GitHub Credential Leak Incidents - API keys stolen from public repos led to database takeovers.

Project Scope

No.	Scope Item
1	Develope a secure and interactive forum platform using the MERN stack.
2	Ensure data privacy through secure API access.
3	Implement secure deployment using environment variables.
4	Provide user authentication and role-based authorization.
5	Implement JWT authentication and role-based access control.
6	Ensure secure API communication between client and server.
7	Provide user engagement through posts, comments, upvotes, and notifications.
8	Build an application with user authentication and authorization.
9	Implement input verification and validation on client and server sides.
10	Enhance security by encrypting sensitive data.
11	Secure API endpoints against unauthorized access.
12	Implement MongoDB’s built-in encryption features.
13	Deploy the application on a secure production environment.

Phase 2: Market Feasibility

Criteria	Details
Identified Gaps in Market	Weak or outdated authentication systems - Insecure user data handling - No proper role-based access control in many existing forums
Proposed Solutions	Strong JWT authentication - Secure API access and encrypted data - Role-based access (Admin/User)
Modern Features Added	Real-time post/comment updates - Voting and reaction system Admin dashboard with analytics - Security-first approach
Target Audience	Cybersecurity enthusiasts - Academic and industry researchers Professionals seeking secure and organized discussions
Market Demand	High demand for privacy-aware, niche communities - Growing focus on secure communication platforms
Competitive Advantage	Combines interactivity (real-time) with robust security - Tailored to cybersecurity and research-based communities

Phase 3: Technical Feasibility

Technology/Tool	Purpose & Contribution
MERN Stack (MongoDB, Express, React, Node.js)	Enables scalable, modular, and maintainable architecture - Efficient for full-stack JavaScript development
Firebase + JWT	Provides secure and scalable authentication - Manages sessions and token validation effectively
MongoDB Atlas	Offers cloud-based database solution with horizontal scalability - Flexible schema supports evolving data models
Tailwind CSS	Facilitates responsive UI design - Rapid styling with utility-first CSS approach
Hosting & Deployment	Frontend hosted on Vercel/Netlify - Backend deployed via Render/Railway with environment variable support
Security Considerations	HTTPS, input validation, token expiration, and encrypted credentials ensure data protection

Phase 4: Financial Feasibility

1.3 Target User Profile and Tentative Elicitation Process

User Type	Description	Goals	Pain Points
Cybersecurity Enthusiasts	Learners exploring secure system design, ethical hacking basics, and CTF competitions.	- Gain practical security knowledge - Participate in forums for knowledge sharing - Test tools in safe environments	- Lack of reliable learning platforms - Information overload - Difficulty differentiating real vs. fake security advice
University Students & Academic Researchers	Students conducting research and academic projects in cybersecurity.	- Access reliable, updated knowledge - Collaborate with peers on research - Publish/validate findings securely	- Limited access to real-world security case studies - Difficulty verifying authenticity of shared resources
Tech Professionals	Developers, system admins, and security engineers seeking secure knowledge-sharing.	- Share best practices	- Time constraints - Risk of leaking sensitive data

		- Discuss vulnerabilities and fixes - Stay updated with industry standards	- Lack of user-friendly platforms
Casual Social Media Users	Regular internet users active on Facebook, Instagram, etc.	- Protect accounts from hacking - Avoid phishing & identity theft - Simple privacy controls	- Weak passwords - Easily tricked by scams - Low awareness of privacy settings
Online Shoppers	Regular buyers from Amazon, Daraz, eBay, etc.	- Secure online payments - Avoid fraud/scam websites - Get alerts for suspicious activities	- Fear of card fraud - Difficulty identifying secure websites - Low awareness of 2FA
Remote Workers / Freelancers	Work remotely using laptops, cloud apps, and collaboration tools.	- Secure data storage & sharing - Safe use of public Wi-Fi - Encrypted communication (VPN, tools)	- Public Wi-Fi vulnerabilities - Data leaks in shared tools - Lack of cybersecurity training

Everyday Internet Users	General users for browsing, email, online banking, entertainment.	- Safe browsing - Protection from malware/viruses - Easy-to-follow security tips	- Weak/reused passwords - Downloads unsafe files - No backup/update habits

1.4 System Requirements (Users)

1.4.1 Hardware Requirement

Component	Specification
Device	Smartphone, Tablet, Laptop, or Desktop
RAM	Minimum 2 GB (for smooth performance on mobile and basic devices)
Processor	Dual-core or higher (common in most smartphones and computers)
Display	Minimum 5-inch screen (mobile) or standard resolution (1024x768+)
Internet Access	Required for browsing, posting, and receiving real-time updates

1.4.2 Software Requirements

Software	Specification
Web Browser	Latest version of Chrome, Firefox, Safari, Edge, or any modern browser
Operating System	Android 8.0+, iOS 12+, Windows 10+, macOS 10.13+, or Linux
No Installation Needed	Fully web-based application; accessible via browser

1.5 System Requirements (Developer)

1.5.1 Hardware Requirement

Component	**Specification**
RAM	Minimum 4 GB (8 GB or more recommended for smooth development)
Processor	Multi-core processor (Intel i5/Ryzen 5 or better recommended)
Storage	At least 5 GB free (for project files, dependencies, and database)
Internet Access	Required for installing packages, API/database testing, and deployment
Display	13" or larger screen with standard resolution (for coding comfortably)

1.5.2 Software Requirements

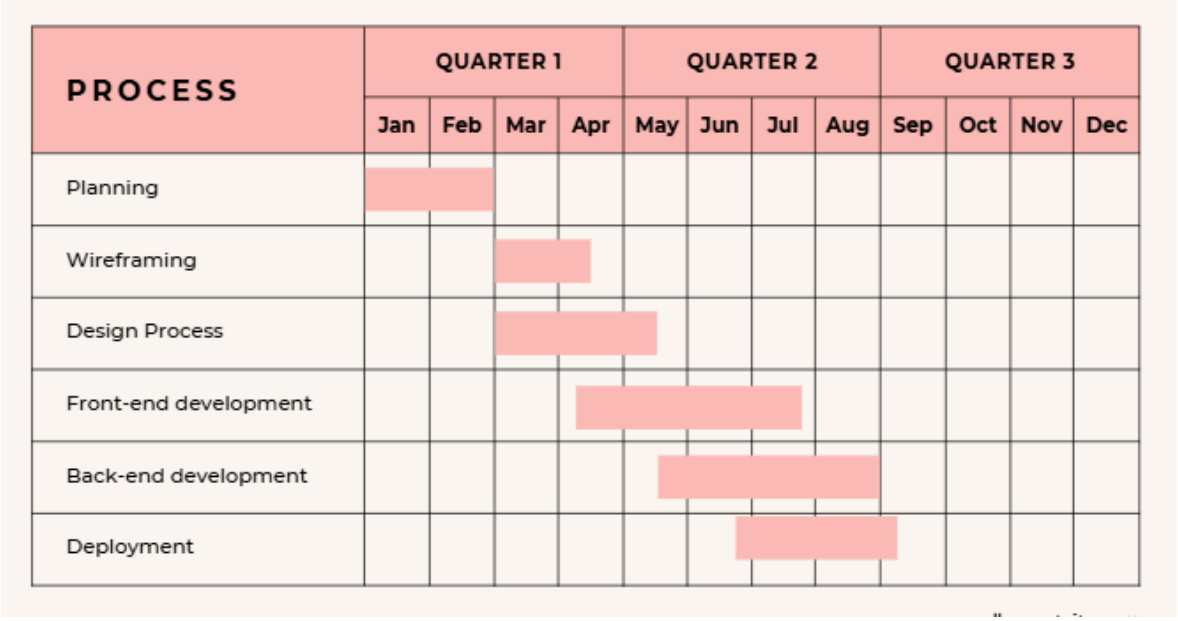
Tool / Platform	Purpose
Node.js & npm	Backend runtime and package manager
React	Frontend framework
Express.js	Backend API framework
MongoDB Atlas	Cloud-hosted NoSQL database
Firebase	Authentication and session management
Git	Version control and collaborative development
Code Editor	VS Code or any modern IDE
Postman / Thunder Client	API testing tool
Vercel / Netlify	Frontend deployment platforms
Render / Railway	Backend hosting and deployment
Browser	Chrome, Firefox, or Edge (for testing frontend)

1.6 Project Scheduling

Phase	What to Do	Duration
1. Initial Setup	Set up your project with React, Node.js, Express.js, MongoDB, etc.	1 weeks
2. User Login & Access	Set up secure login and user roles (Admin/User).	1 weeks
3. Data & Database Protection	Set up MongoDB with secure credentials and handle sensitive user data properly.	4 weeks
4. User Features Security	Secure post creation, comments, and voting actions.	3 weeks

5. Admin Panel Security	Protect the Admin panel from unauthorized access.	2 weeks
6. API & Session Management	Protect API routes and manage user sessions securely.	3 weeks
7. Social Logins	Use Firebase for social media logins (Google).	2 weeks
8. Encryption & Sensitive Data	Encrypt sensitive data when it's stored or being transferred (MongoDB, HTTPS).	3 weeks
9. Deployment Security	Deploy your app securely to platforms like Netlify or Vercel.	2 weeks
10. Monitoring & Alerts	Set up error tracking and security monitoring.	2 weeks

1.7 Gantt Chart



1.8 Risk Management

Risk	Impact	Mitigation Strategy
Hosting failures	Medium	Mirror deployment on alternate platforms
Token hijacking / session misuse	High	Implement JWT refresh logic & secure cookie policies
Input validation issues	High	Use strong sanitization & schema validation
Unauthorized admin access	Critical	Use Jwt token for admin route protection
Deployment leaks	High	Use .gitignore, keep secrets out of public repos
Social login exploits	Medium	Enable CSRF protection and HTTPS only cookies

Chapter 2: Design and Implementation

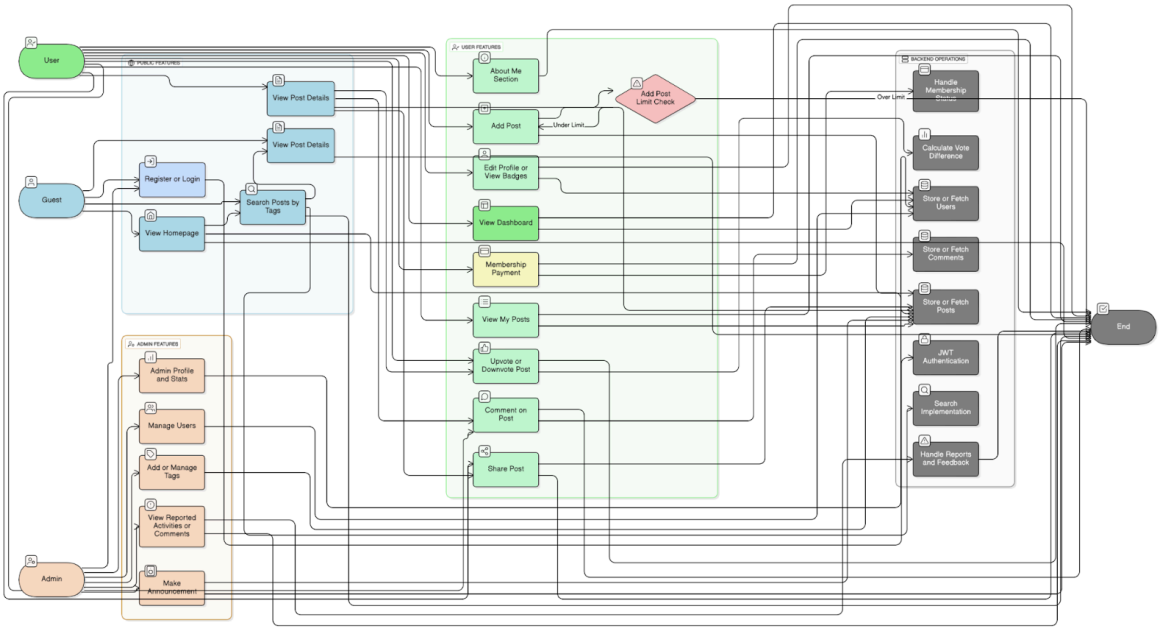
2.1 Functional Requirements

Feature	Description
User Registration & Login	Users register/login via Firebase Authentication. JWT is used for secure session management across frontend/backend.
Authentication & Role Management	Users are assigned roles: Regular User or Admin. Role-Based Access Control restricts sensitive endpoints to Admins only.
Post Management	Users can create, edit, and delete their posts. Posts can be tagged/categorized. Admins can moderate or remove content violating platform policies.
Comment System	Registered users can comment, edit, or delete their own comments. Admins can review and take action reported comments.
Voting System	Users can upvote/downvote posts based on quality/relevance. Helps rank valuable content and promote meaningful discussions.
Tag-Based Post Filtering	Users can browse and filter posts by tags or categories to improve discoverability of niche topics.
Search Functionality	Users can search posts by title, tags, or keywords. Filters help refine search results efficiently.
Announcement System	Admins can publish platform-wide announcements such as updates, rules, or maintenance notices, with control over visibility and expiration.
Admin Dashboard & Analytics	Admin-only dashboard showing activity logs, content statistics, and engagement trends via charts and graphs.
User Management	Admins can view users, edit roles, deactivate/delete accounts, and monitor/report suspicious activities.
Content Moderation	Tools for reporting inappropriate content, admin review of flagged items, and warning/banning of users violating community guidelines.

2.2 Non-Functional Requirements

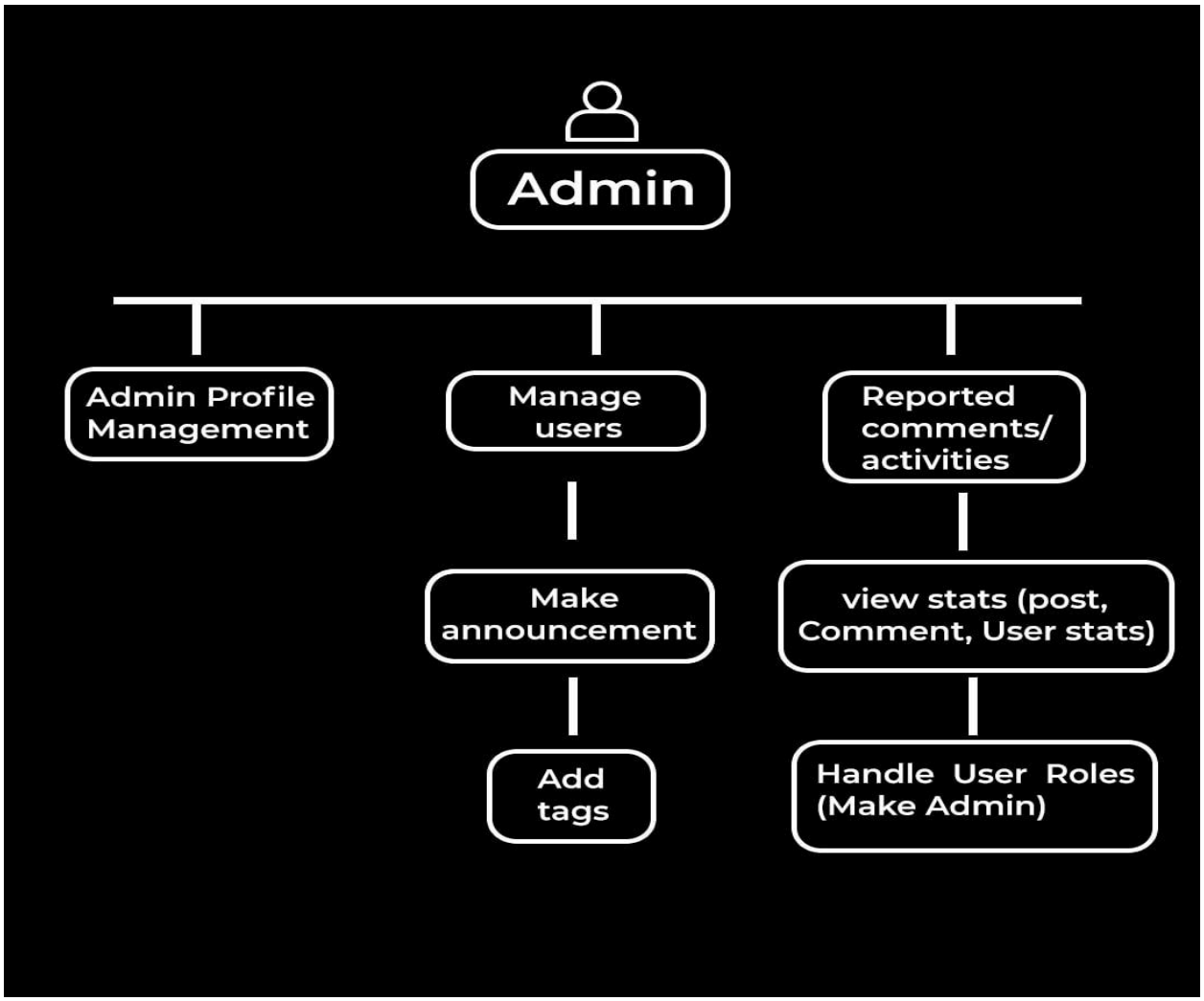
Aspect	Description
Security	Utilizes JWT for session control, Firebase Authentication for user login, HTTPS for secure data transfer, and rigorous data validation to prevent malicious input.
Performance	Optimized for fast load times using TanStack Query for efficient data fetching and pagination for large datasets.
Scalability	Scalable cloud deployment using Vercel for frontend and MongoDB Atlas for handling growing database demands.
Responsiveness	Ensures consistent and adaptive UI across devices (mobile, tablet, desktop) using Tailwind CSS.
Searchability	Dynamic search and filtering based on tags and keywords improve discoverability of content.
Maintainability	Built with modular and reusable components using React.js, making it easy to update, test, and scale.

2.3 Object-Oriented System Design Using UML

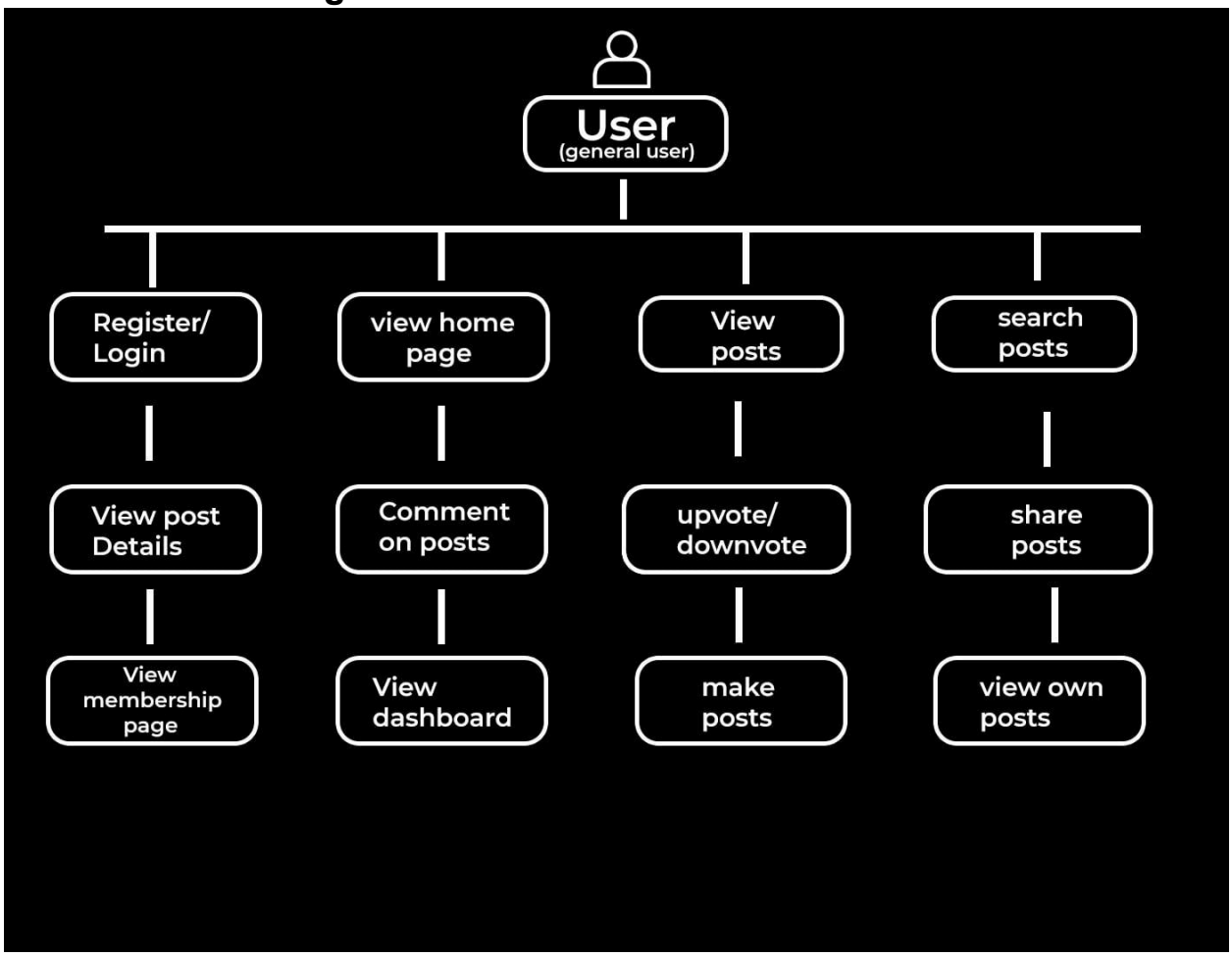


eraser

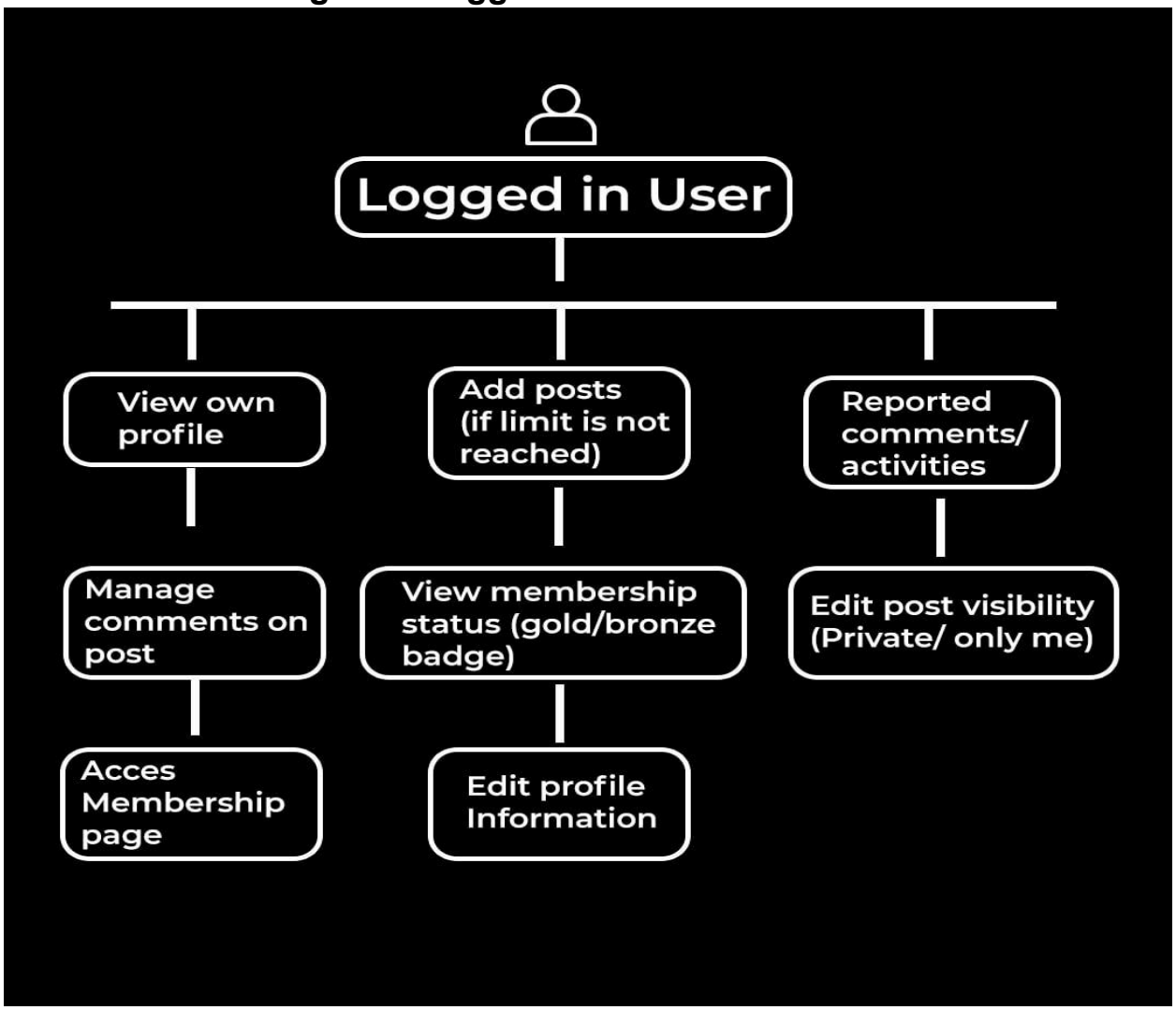
2.3.1 Use Case Diagram – Admin



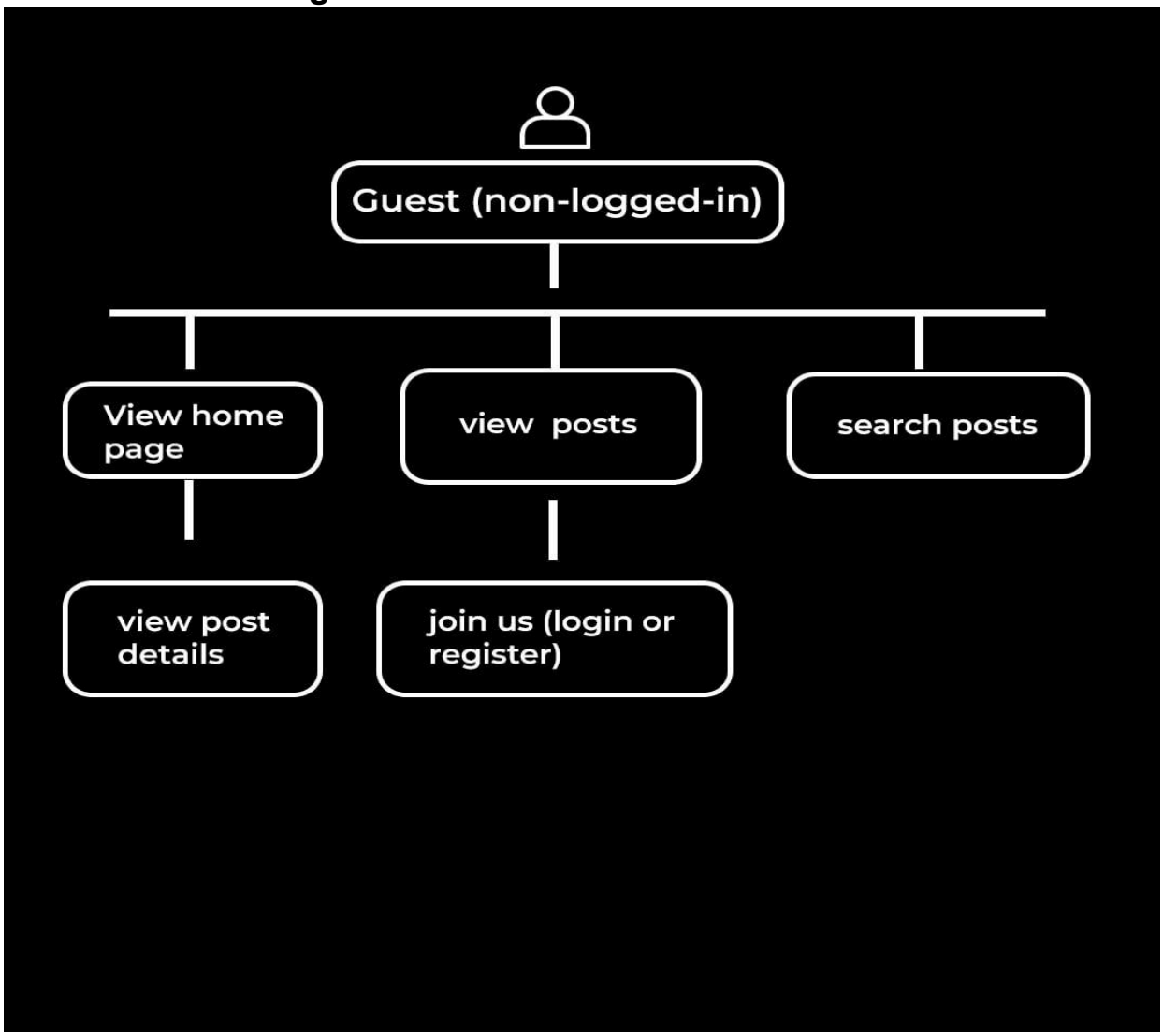
2.3.2 Use Case Diagram – User



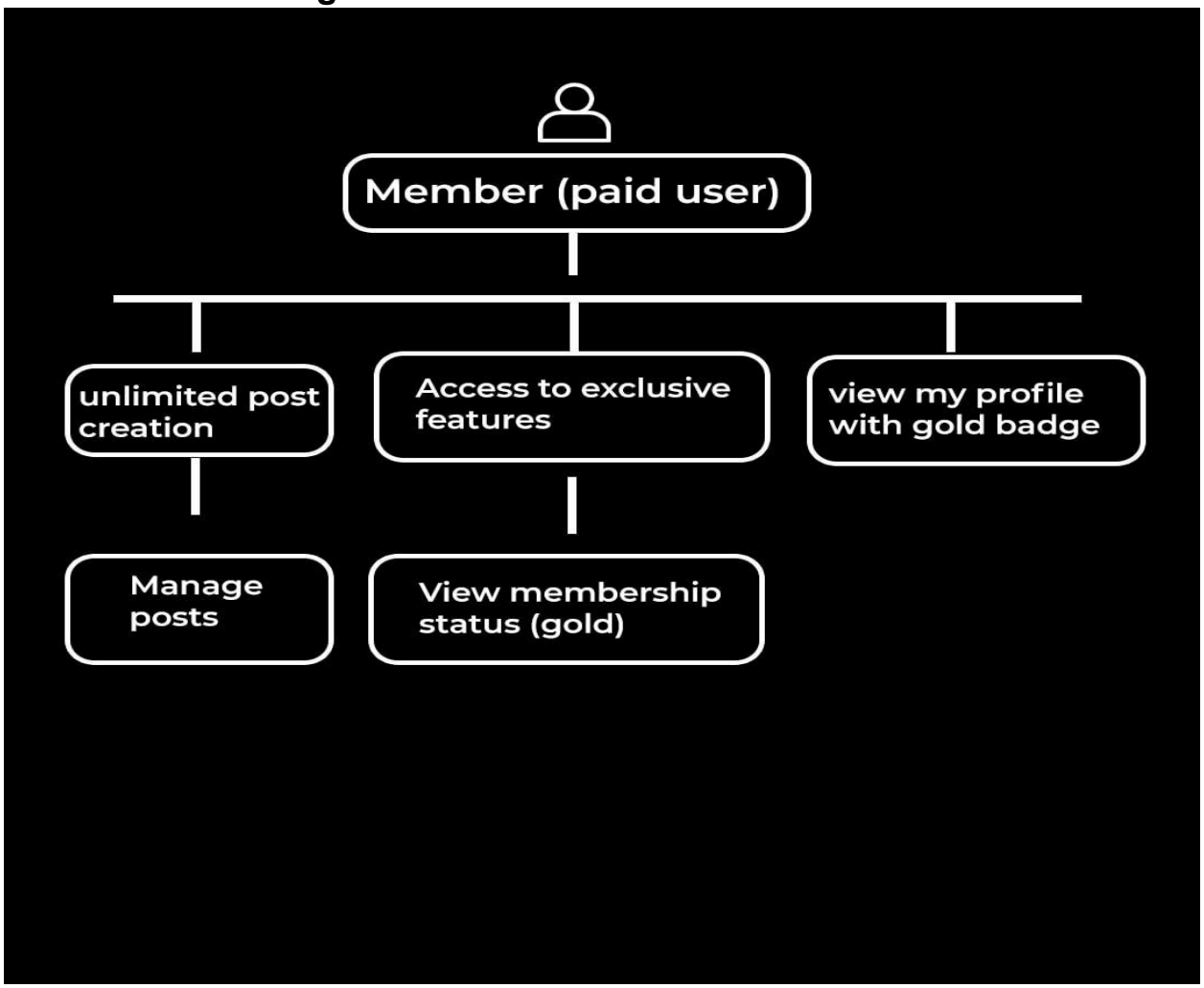
2.3.3 Use Case Diagram –Logged in User



2.3.4 Use Case Diagram – Guest



2.3.5 Use Case Diagram – Paid User



2.4 Use Case Description

Use Case Table

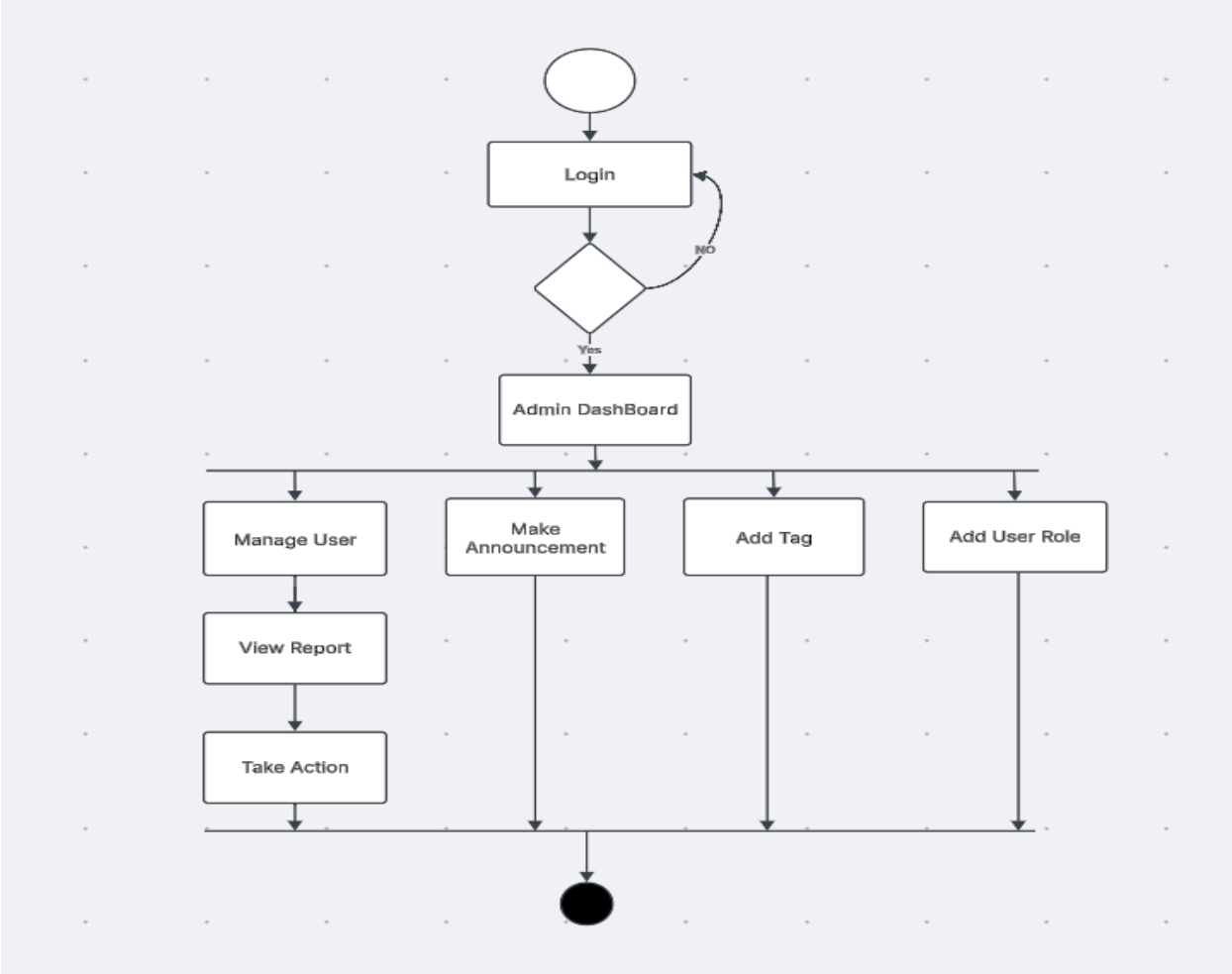
Use Case ID	Use Case Name	Actor	Pre-Conditions	Main Flow	Post Conditions	Exceptions
UC-01	User Registration	Guest	The user is not authenticated	1. Navigate to sign-up page 2. Enter details 3. Submit form 4. System validates and registers user	New user account is created	Validation fails or email already exists
UC-02	User Login	Guest	Must have registered account	1. Navigate to login page 2. Enter credentials 3. System verifies and logs in	The user is authenticat edand receives JWT	Invalid credentials or server error
UC-03	View Posts	Guest, User	None	1. Visit the homepage. 2. System fetches and displays posts	Posts are displayed	The backend fails to fetch data
UC-04	View Post Details	Guest, User	Post ID must exist	1. Click on a post 2. System displays full post with comments	Post and comments are shown	Invalid post ID or system error
UC-05	Create New Post	Regist ered User	The user must be authenticated	1. Navigate to Add Post 2. Fill in the form. 3. Submit form 4. System validates and stores 5. Post is listed	The post is created and visible	Validation fails or session expired

UC-06	Comment on Post	Registered User	Authenticated and post exists	1. Enter comment 2. Submit 3. System validates and saves	The comment appears under the post.	Empty comment or session expired
UC-07	Edit/Delete Own Post	Registered User	Must be post owner	1. Navigate to post 2. Click edit/delete 3. System updates or deletes	The post is updated or deleted	Not authorized or post not found
UC-08	Upvote/Downvote a Post	Registered User	User must be authenticated	1. Click upvote/downvote 2. System updates vote count	Vote count updated	Already voted or session expired
UC-09	Access Dashboard	Registered User	User must be authenticated	1. Click dashboard 2. System loads user info and stats	Dashboard is displayed	Session expired or data fetch failed
UC-10	Membership Upgrade	Registered User	User must be authenticated	1. Open membership page 2. Choose plan 3. Complete payment4. Status updated	New privileges activated	Payment or system error
UC-11	Admin Login	Admin	Valid admin credentials	1. Enter admin credentials 2. System verifies and grants access	Admin session begins	Invalid credentials or unauthorized access
UC-12	Manage Users	Admin	Admin must be authenticated	1. View user list 2. Assign/revoke roles or delete user	User roles updated or users removed	Not authorized or user not found

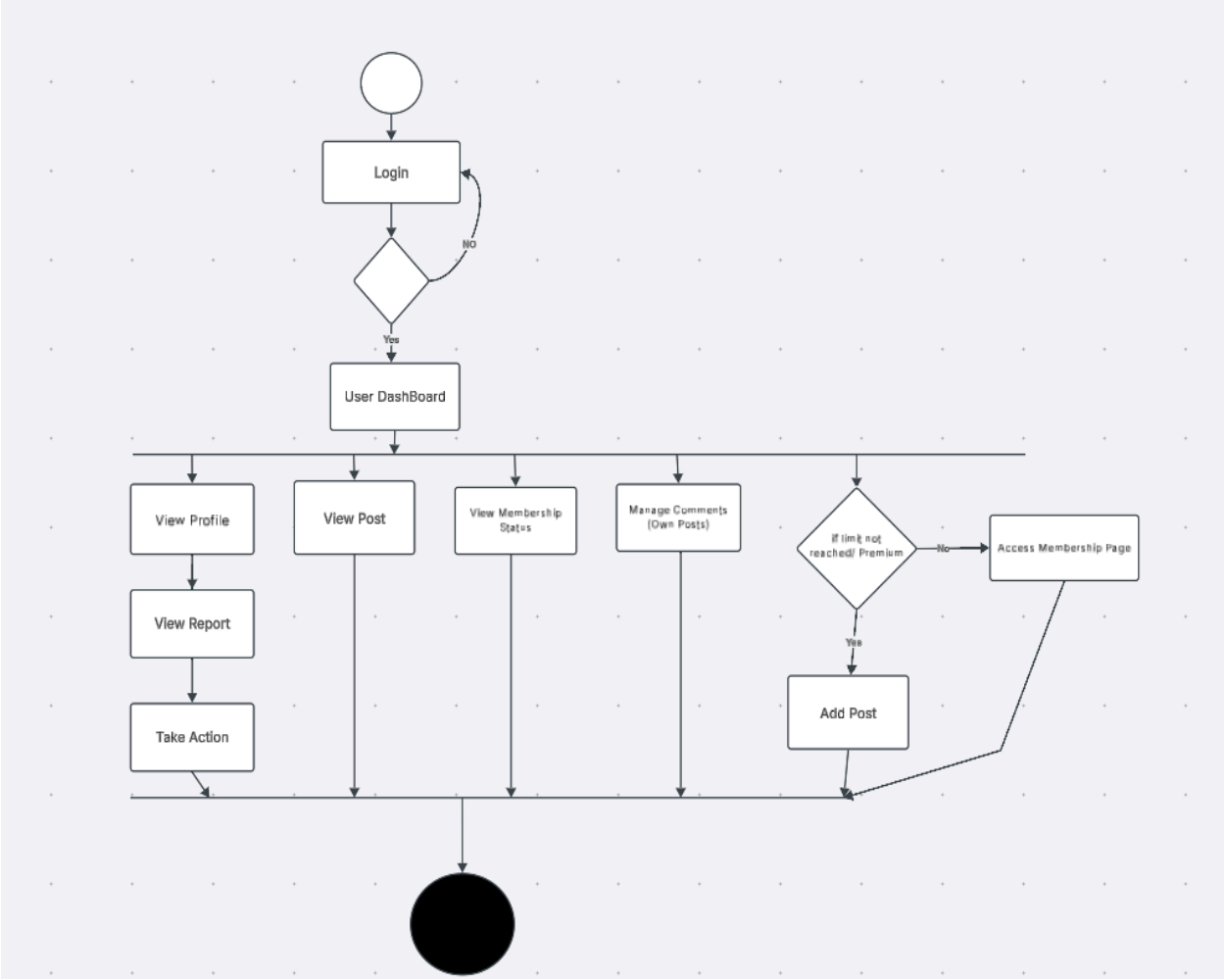
UC-13	Post Announcements	Admin	Admin must be authenticated	1. Navigate to announcement page 2. Create announcement 3. Submit	Announcement is published	Validation or system error
UC-14	Moderate Content	Admin	Admin must be authenticated	1. View reported content 2. Take action (delete/ignore) 3. System executes action	Content is moderated	Content not found or action failed

2.5 Activity Diagram

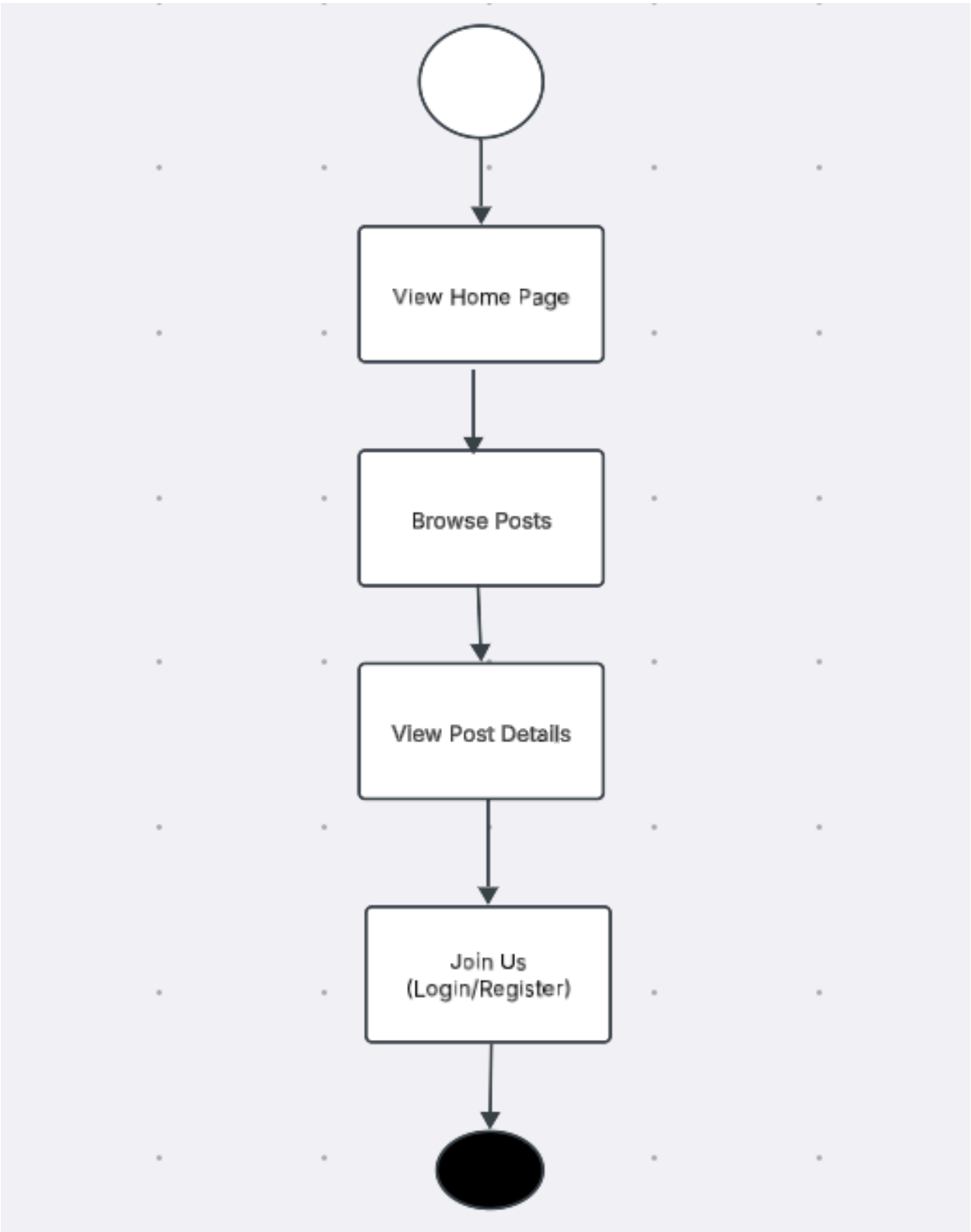
2.5.1 Activity Diagram – Admin



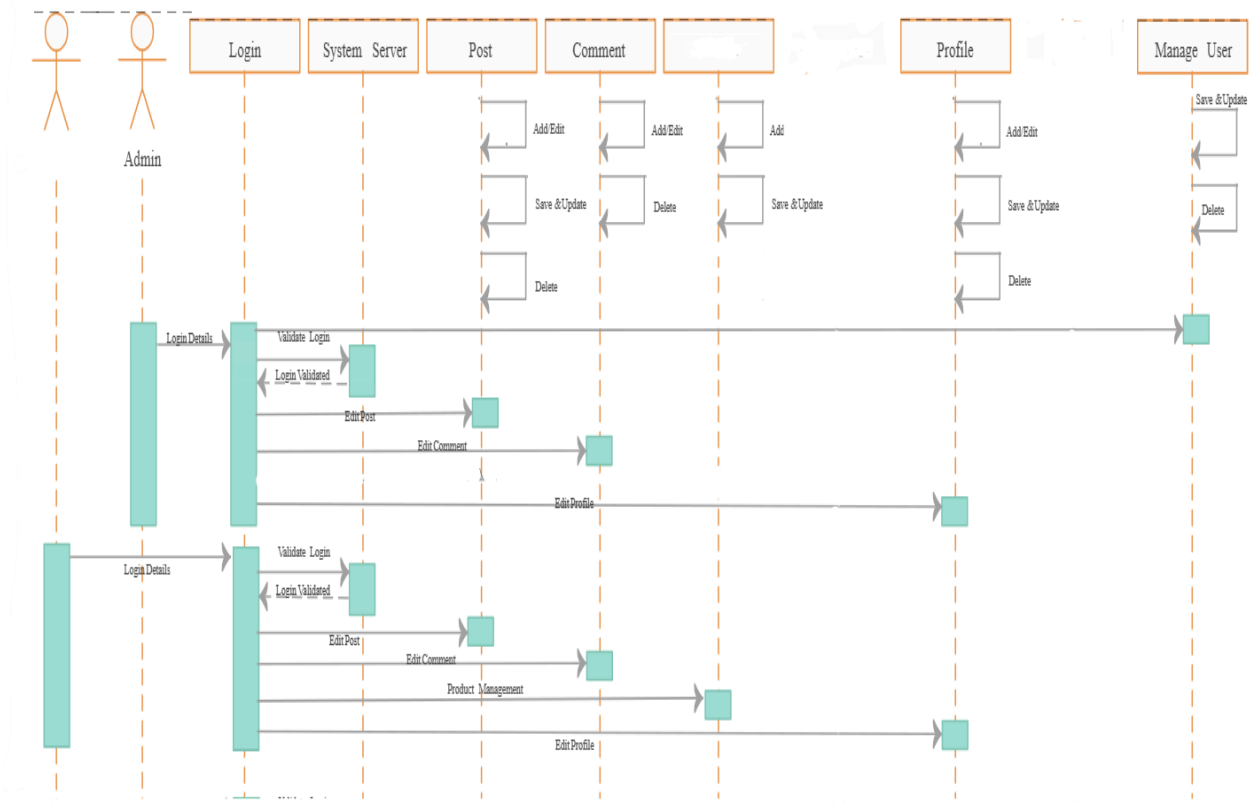
2.5.2 Activity Diagram – User



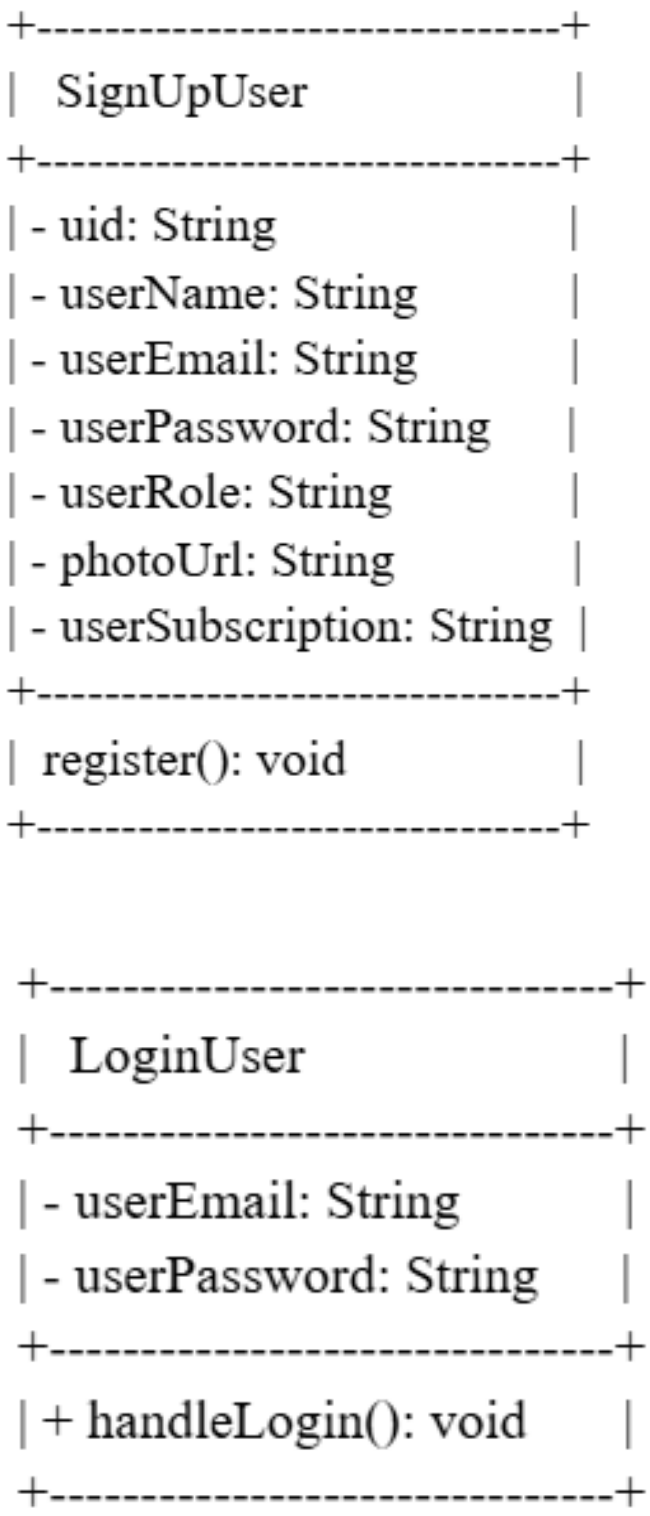
2.5.3 Activity Diagram – Guest

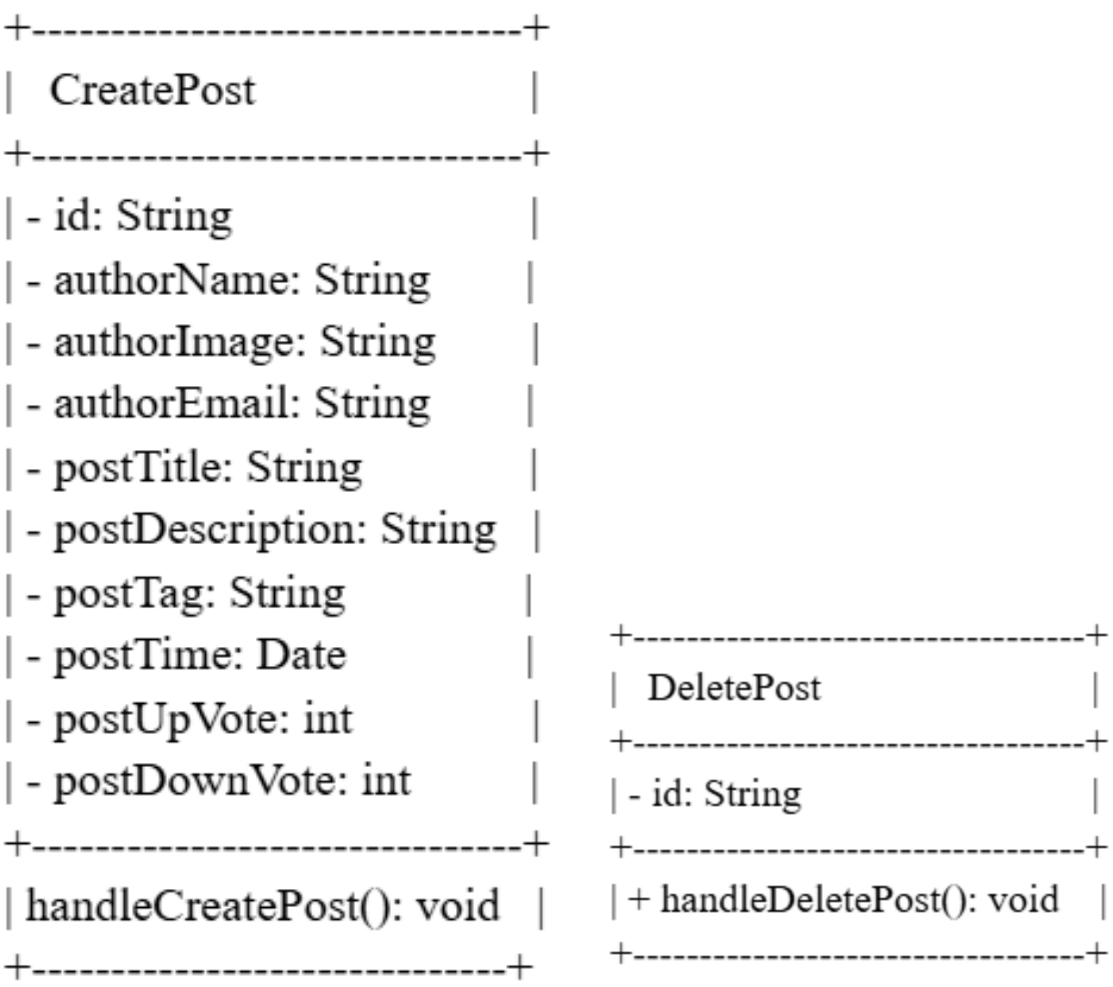


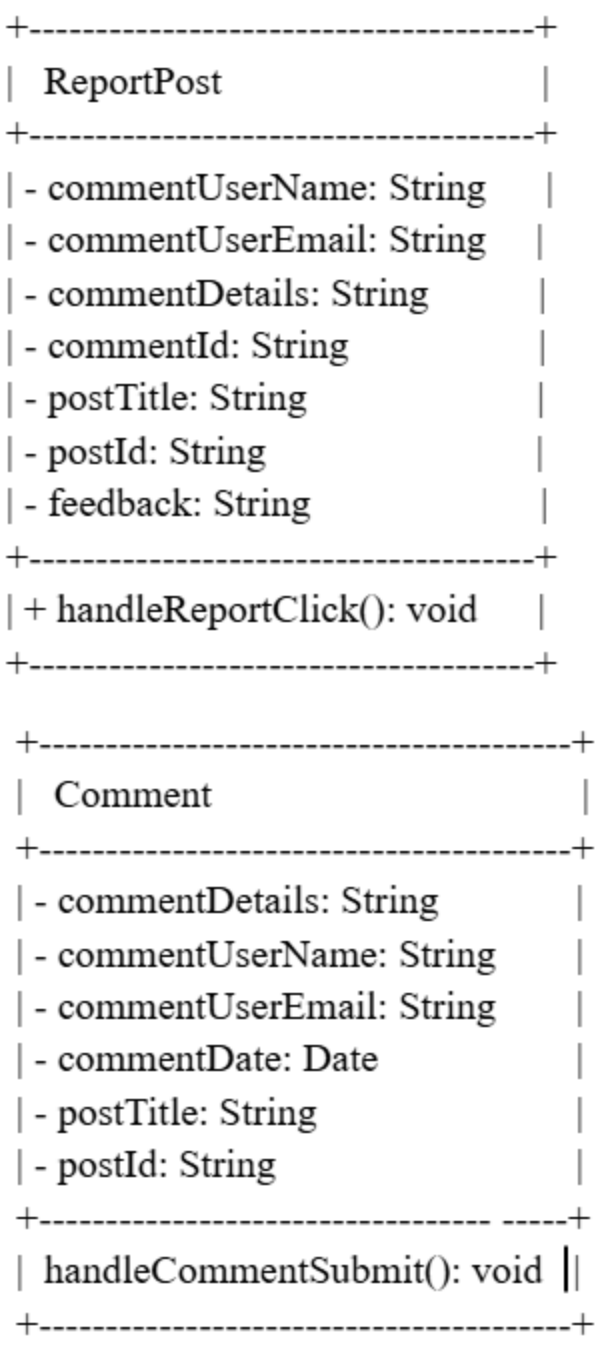
2.6 Sequence Diagram



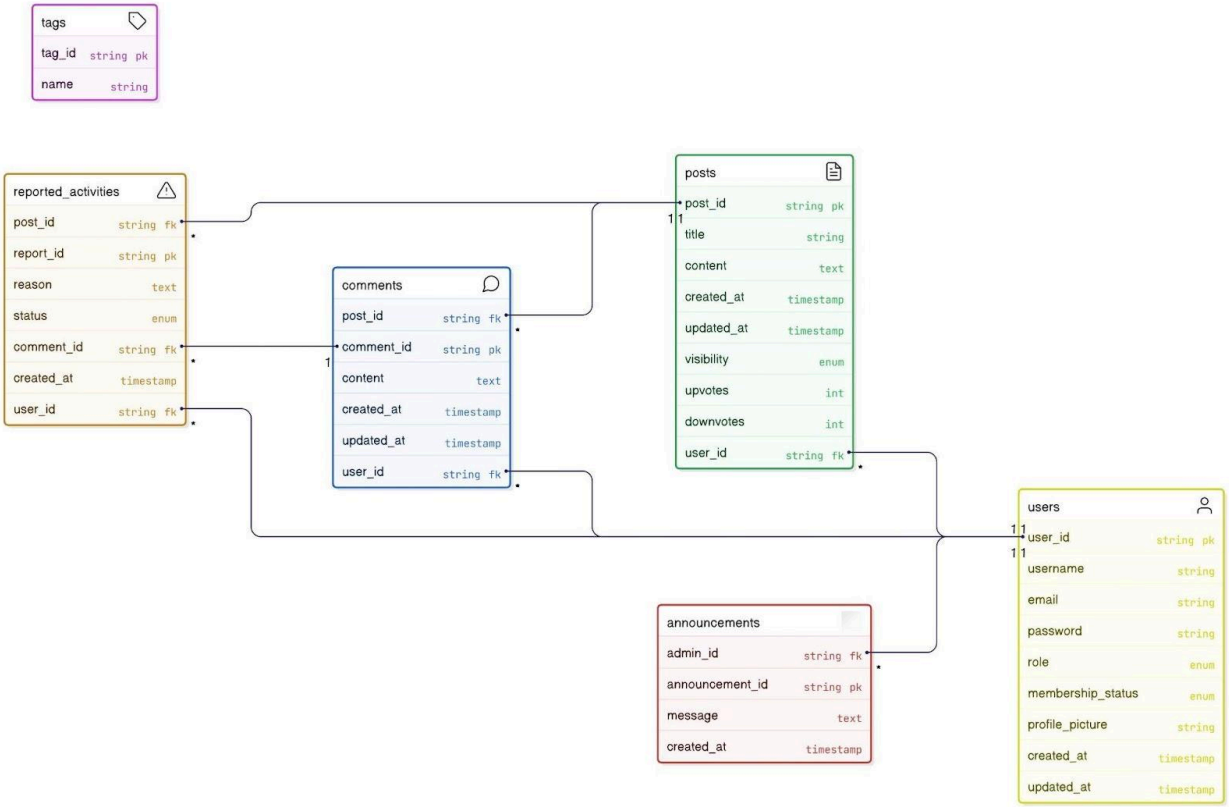
2.7 Class Diagram







2.8 ER Diagram



2.9 Code Snippets

2.9.1 FrontEnd Code

Function Details Copy Function

src/Provider/AuthProvider.jsx:22-26

handleSignInWithEmailAndPassword()

Parameters

email password

Code Preview

```
22  handleSignInWithEmailAndPassword = (email, password) => {
23          setLoading(true)
24          return signInWithEmailAndPassword(auth, email, password)
25
26      }
```

Called by (1)

handleLogin line

Calls (2)

setLoading line
signInWithEmailAndPassword line

Function Details Copy Function

src/Provider/AuthProvider.jsx:28-31

handleCreateUserWithEmailAndPassword()

Parameters

email password

Code Preview

```
28  handleCreateUserWithEmailAndPassword = (email, password) => {
29          setLoading(true);
30          return createUserWithEmailAndPassword(auth, email, password)
31      }
```

Called by (1)

onSubmit line

Calls (2)

setLoading line
createUserWithEmailAndPassword line

Function Details

Copy Function

src/Provider/AuthProvider.jsx:17-20

handleGoogleSignInWithPopup()

Code Preview

```
17  handleGoogleSignInWithPopup = () => {
18          setLoading(true)
19          return signInWithPopup(auth, provider)
20      }
```

Called by (1)

handleGoogleLoginButton line

Calls (2)

setLoading line

signInWithPopup line

useAxiosSecure()

Code Preview

```
6  useAxiosSecure = () => {
7      // const navigate = useNavigate();
8      // const {handleSignOut} =useAuth();
9      axiosSecure.interceptors.request.use((config) => {
10         const token = localStorage.getItem('access_token')
11         config.headers.authorization = `Bearer ${token}`
12         // console.log("request stop by interceptors")
13         // console.log(config)
14         return config
15     }, (error) => {
16         return Promise.reject(error)
17     })
18     axiosSecure.interceptors.response.use((response) => {
19         return response
20     }, (error) => {
21         const status = error.response.status;
22         if (status === 401 || status === 403) {
23             // handleSignOut();
24             // navigate('/error')
25             // alert("error")
26         }
27         return Promise.reject(error)
28     })
29     return axiosSecure;
30 }
```

Called by (14)

Calls (3)

useAxiosPublic()

Code Preview

```
5  useAxiosPublic = () => {
6      return  axiosPublic;
7  }
```

Called by (5)

Calls (0)

useAdmin()

Code Preview

```
5  useAdmin = () => {
6      const { user, loading } = useAuth();
7      const axiosSecure = useAxiosSecure();
8      const { data: isAdmin, isLoading: isAdminLoading } = useQuery({
9          queryKey: [user?.email, 'isAdmin'],
10         enabled: !loading,
11         queryFn: async () => {
12             const res = await axiosSecure.get(`/users/admin/${user?.email}`)
13             // console.log(res.data.isAdmin)
14             return res.data.isAdmin
15         }
16     })
17     return [isAdmin, isAdminLoading]
18 }
```

Called by (3)

MyProfile line
Navbar line
AdminRoutes line

Calls (3)

useAuth line
useAxiosSecure line
useQuery line

2.9.2 BackEnd Code

```
    //  db collection
    const postsCollections = client.db("thinkloopDB").collection("posts");
    const commentsCollections = client.db("thinkloopDB").collection("comments");
    const announcementsCollections = client.db("thinkloopDB").collection("announcements");
    const usersCollections = client.db("thinkloopDB").collection("users");
    const usersVotesCollections = client.db("thinkloopDB").collection("usersVotes");
    const tagsCollections = client.db("thinkloopDB").collection("tags");
    const reportsCollections = client.db("thinkloopDB").collection("reports");
    const paymentCollections = client.db("thinkloopDB").collection("payments");


    // jwt api
    app.post('/jwt', async (req, res) => {
      const user = req.body;
      const token = jwt.sign(user, process.env.ACCESS_TOKEN_SECTRET, { expiresIn: '1h' });
      res.send({ token });
    })

    const verifyToken = (req, res, next) => {
      // console.log("inside Verify token middleware")
      if (!req.headers.authorization) {
        return res.status(401).send({ message: 'unauthorized access for null token' })
      }
      const token = req.headers.authorization.split(' ')[1];
      // console.log(token)
      jwt.verify(token, process.env.ACCESS_TOKEN_SECTRET, (err, decoded) => {
        if (err) {
          return res.status(403).send({ message: 'unauthorized access for invalid token' })
        }
        req.decoded = decoded;
        // console.log(req.decoded.email)

        next();
      });

    }
```

```
// milldeware for verify admin after verify token
const verifyAdmin = async (req, res, next) => {
  const email = req.decoded.email;
  const query = { userEmail: email };
  const user = await usersCollections.findOne(query);
  const isAdmin = user.userAdmin === true
  // console.log(query)
  if (!isAdmin) {
    return res.status(403).send({ message: 'frobidden access for invalid token' })
  }
  next();
}
// admin check
app.get("/users/admin/:email", verifyToken, async (req, res) => {
  // console.log("inside Verify token middleware")

  const email = req.params.email;
  if (email !== req.decoded.email) {
    return res.status(403).send({ message: 'forbidden access for invalid token & email' })
  }
  const query = { userEmail: email };
  const result = await usersCollections.findOne(query);
  let isAdmin = false;
  if (result) {
    isAdmin = result.userAdmin === true;
  }
  // console.log(isAdmin)
  res.send({ isAdmin });

})

```

```
101    // posts api
102 >  app.get('/posts', verifyToken, async (req, res) => {···
138    })
139 >  app.get('/publicposts', async (req, res) => {···
177    })
178
179    // single post by id api
180 >  app.get('/post/:id', async (req, res) => {···
186    })
187
188    // single post by email api
189 >  app.get('/posts/:email', async (req, res) => {···
212    })
213
214    // add new post api
215 >  app.post('/posts', verifyToken, async (req, res) => {···
220    })
221
222    // edit  post
223 >  app.patch('/posts/:id', verifyToken, async (req, res) => {···
233    })
234    // delete post
235 >  app.delete('/posts/:id', verifyToken, async (req, res) => {···
241    })
242
```

```
242
243    // comments api
244 >  app.get('/comments', async (req, res) => {···
247    })
248 >  app.get('/comments/:id', async (req, res) => {···
257    })
258    // add comment api
259 >  app.post('/comments', verifyToken, async (req, res) => {···
266    })
267
268    // comment Count api
269 >  app.get(`/comments-count/:postid`, async (req, res) => {···
281    })
```

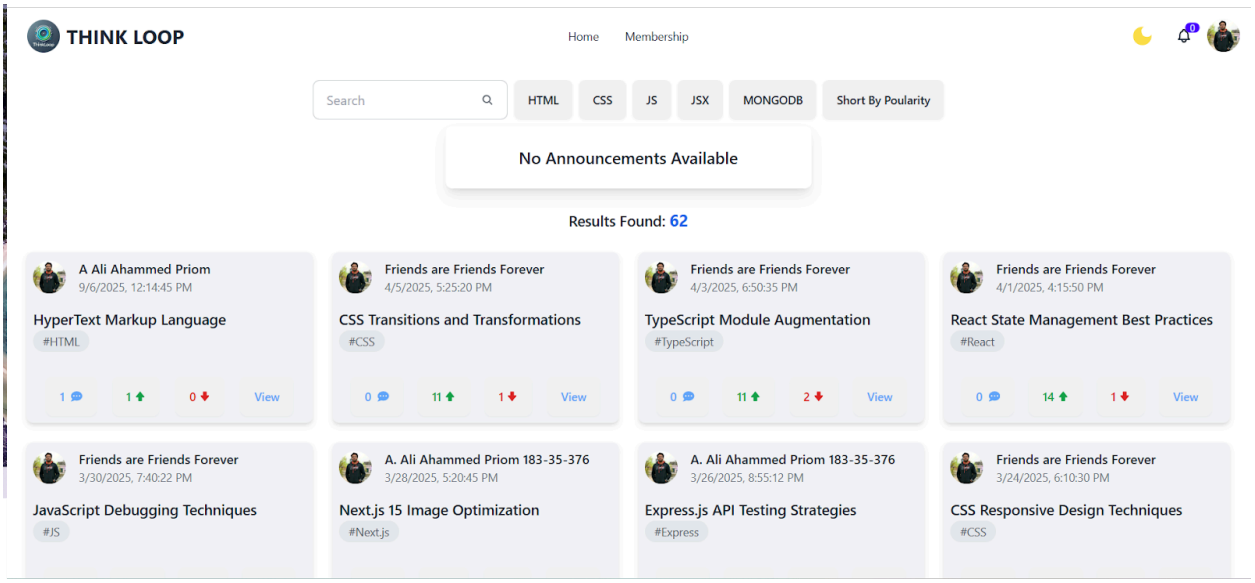
```
283   // report  api
284 > app.get('/reports', verifyToken, verifyAdmin, async (req, res) => {···
290   })
291 > app.post('/reports', verifyToken, async (req, res) => {···
309   })
310 > app.delete('/reports-post/:id', verifyToken, verifyAdmin, async (req, res) => {···
328   })
329
```

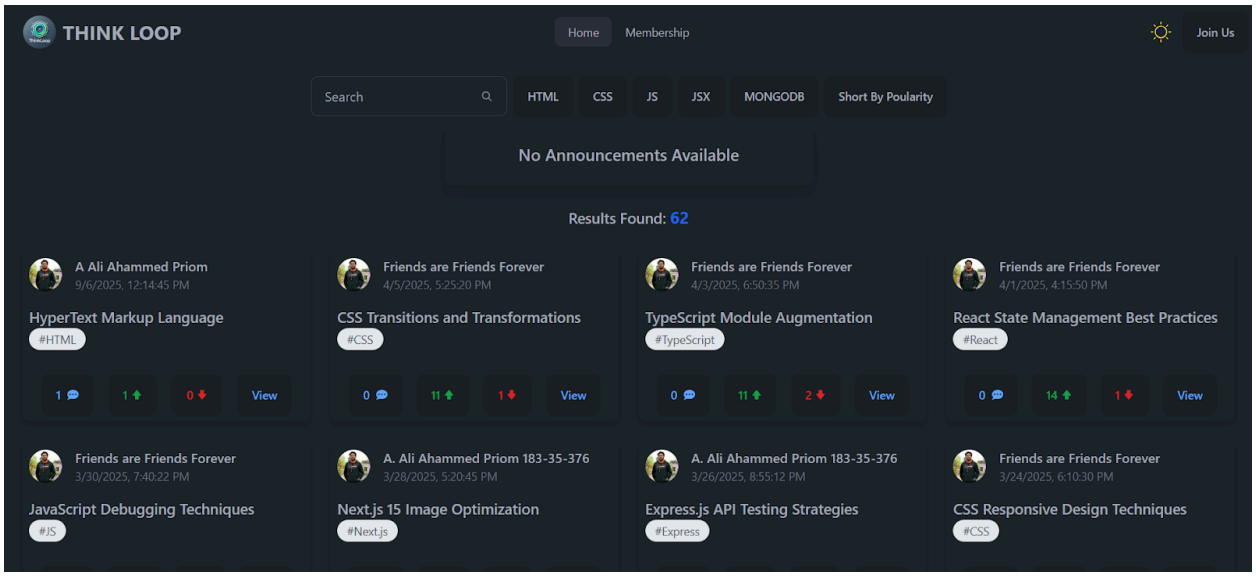
```
330   // user vote api
331 > app.get('/votes', verifyToken, async (req, res) => {···
334   })
335
336   // upVote api
337 > app.post('/userupvote', verifyToken, async (req, res) => {···
377   })
378
379   // downVote api
380 > app.post('/userdownvote', verifyToken, async (req, res) => {···
420   })
```

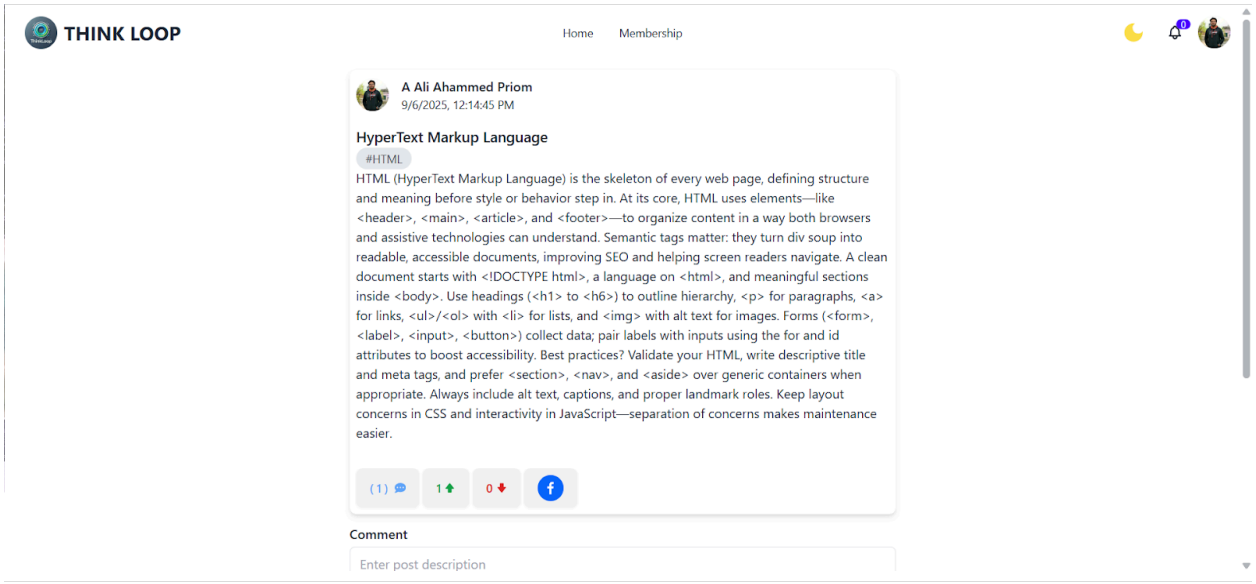
```
422   // tags api
423
424 > app.get('/tags', async (req, res) => {···
427   })
428 > app.post('/tags', verifyToken, verifyAdmin, async (req, res) => {···
432   })
433
434
435   // announcements api
436   // announcements get api
437 > app.get('/announcements', async (req, res) => {···
440   })
441   // announcements post api
442 > app.post('/announcements', verifyToken, verifyAdmin, async (req, res) => {···
447   })
448
449
450   // users api
451 > app.get("/users", verifyToken, verifyAdmin, async (req, res) => {···
465   })
466   // users by email api
467 > app.get("/users/:email", verifyToken, async (req, res) => {···
473   })
474
475   // users add api
476
477 > app.post("/users", async (req, res) => {···
489   })
```

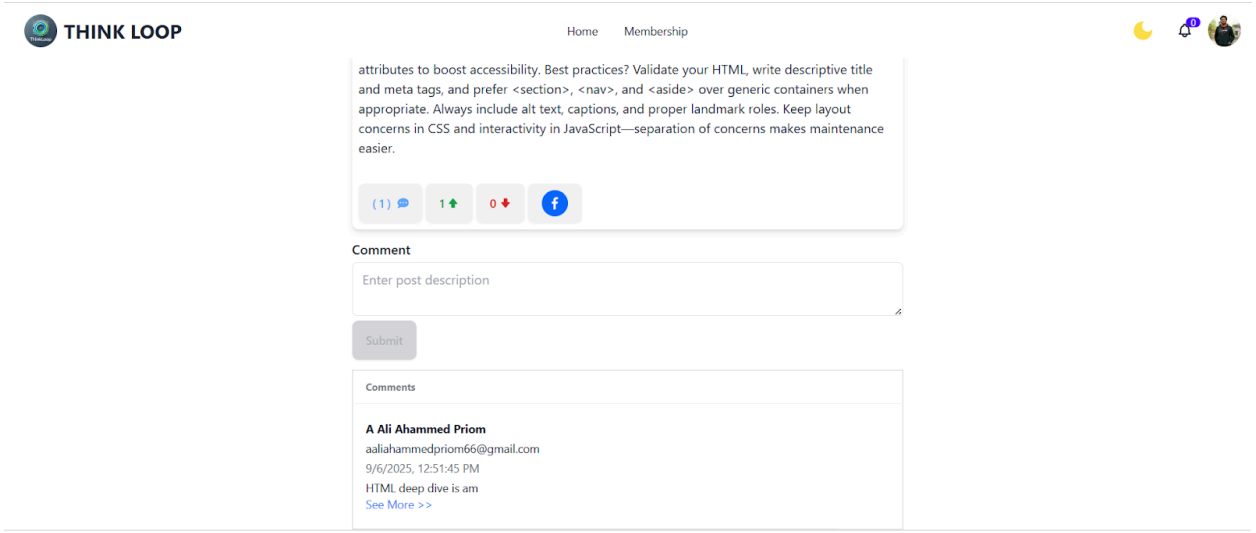
```
491     // add admin
492 >   app.patch('/users/:id', verifyToken, verifyAdmin, async (req, res) => {···
504     })
505 >   app.patch('/subscribe', verifyToken, async (req, res) => {···
511     })
512
513     // payment intent stripe
514 >   app.post('/create-payment-intent', async (req, res) => {···
524     })
525
526
527     // payment and order history
528 >   app.get('/payments/:email',verifyToken, async(req, res)=>{···
536     })
537
538     // stats or analytics
539 >   app.get('/admin-stats', verifyToken, verifyAdmin, async (req, res) => {···
545     })
```

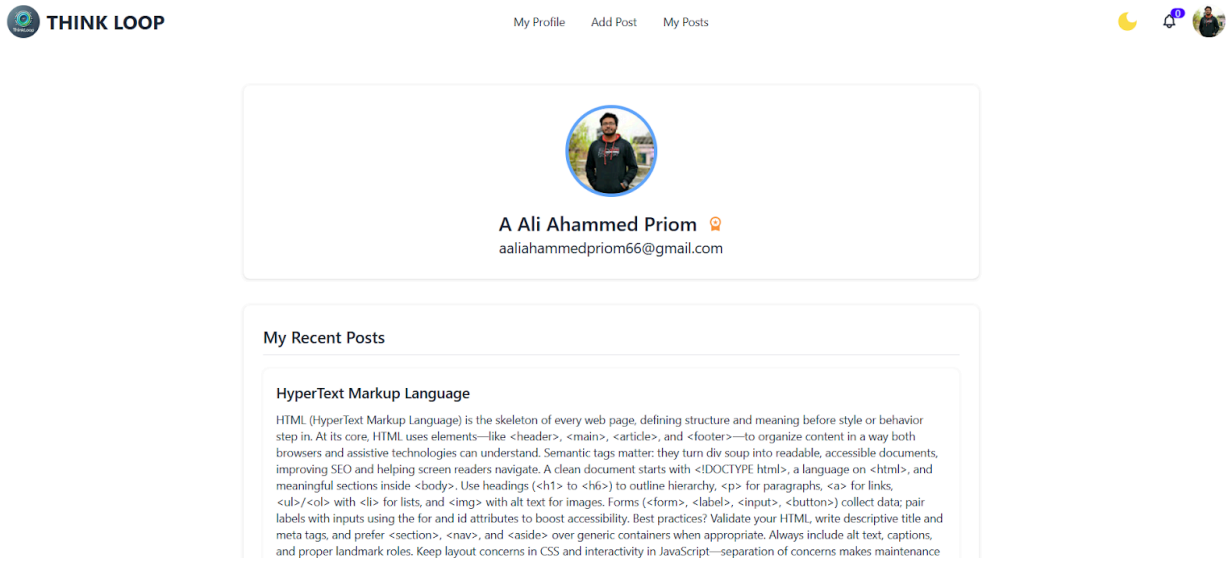
2.10 Ui Snippets

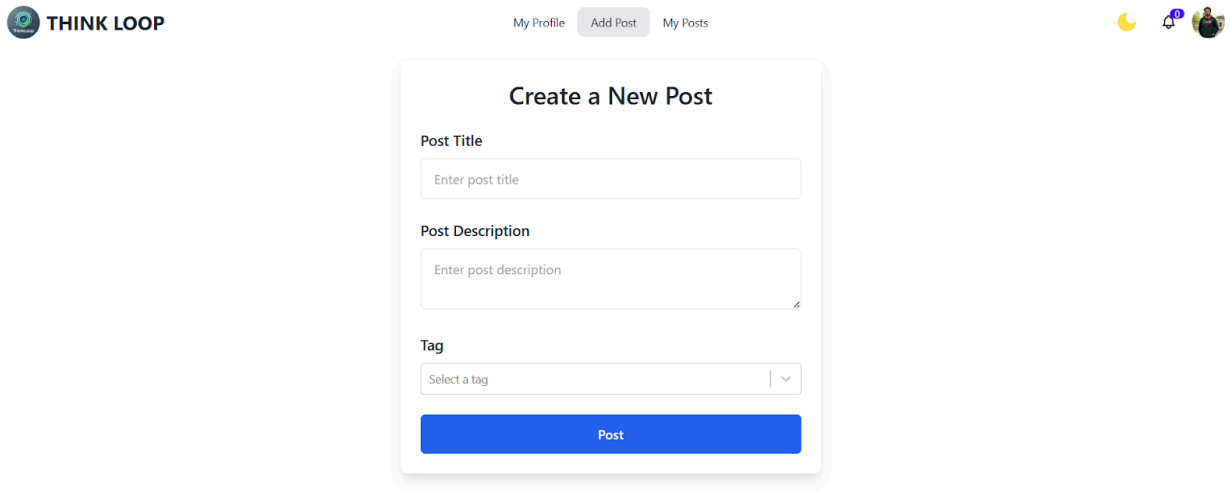


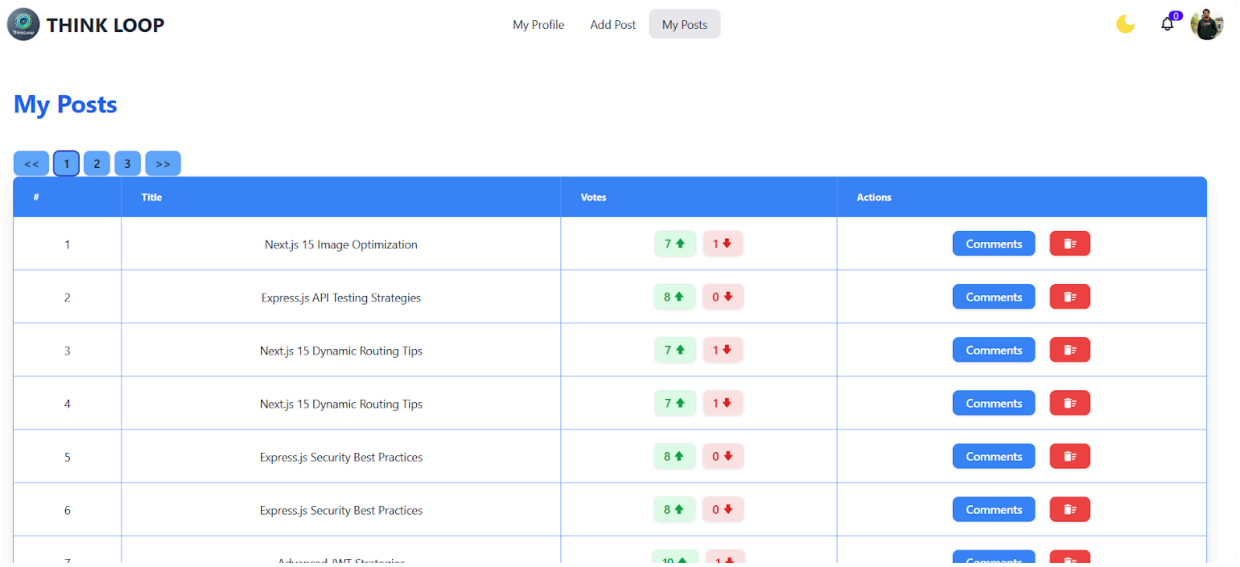


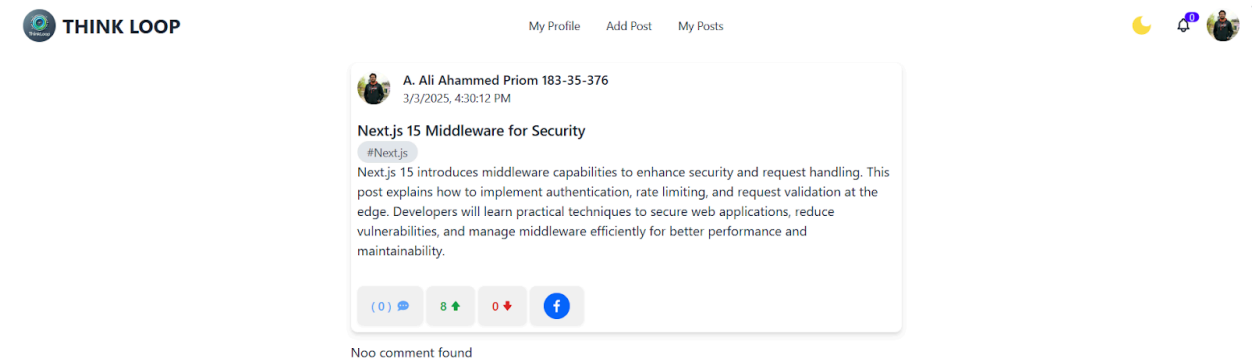


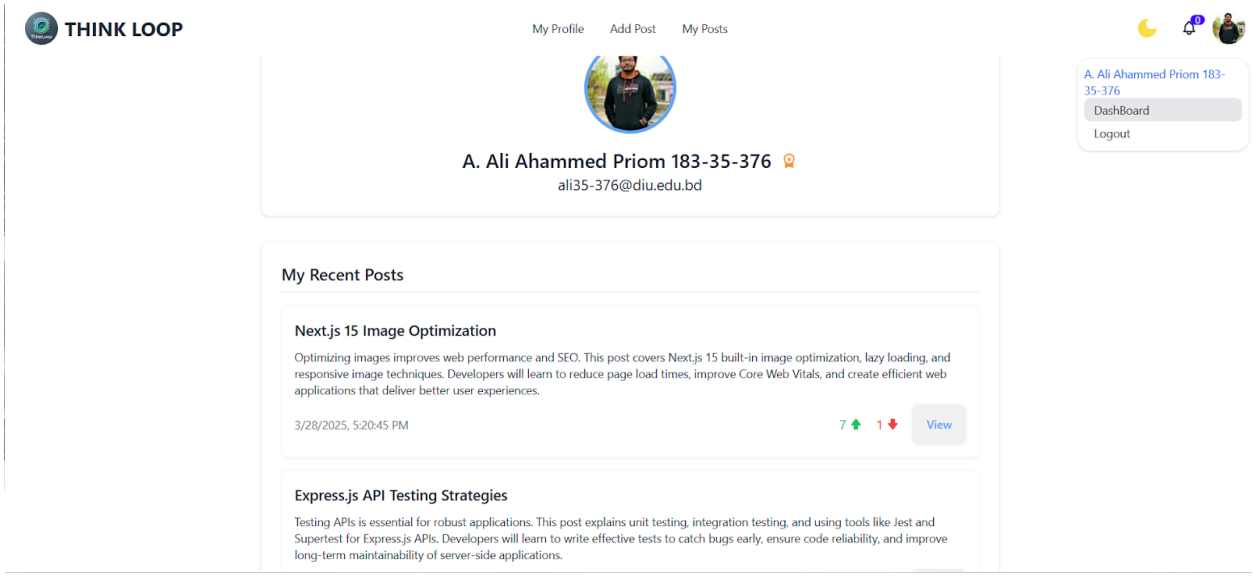


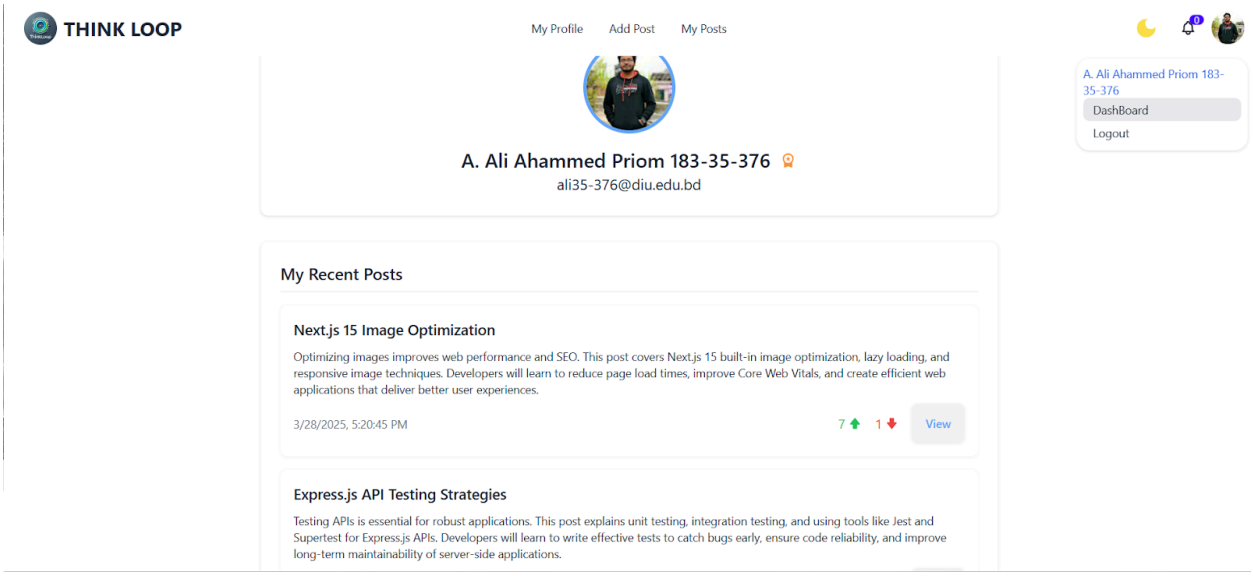


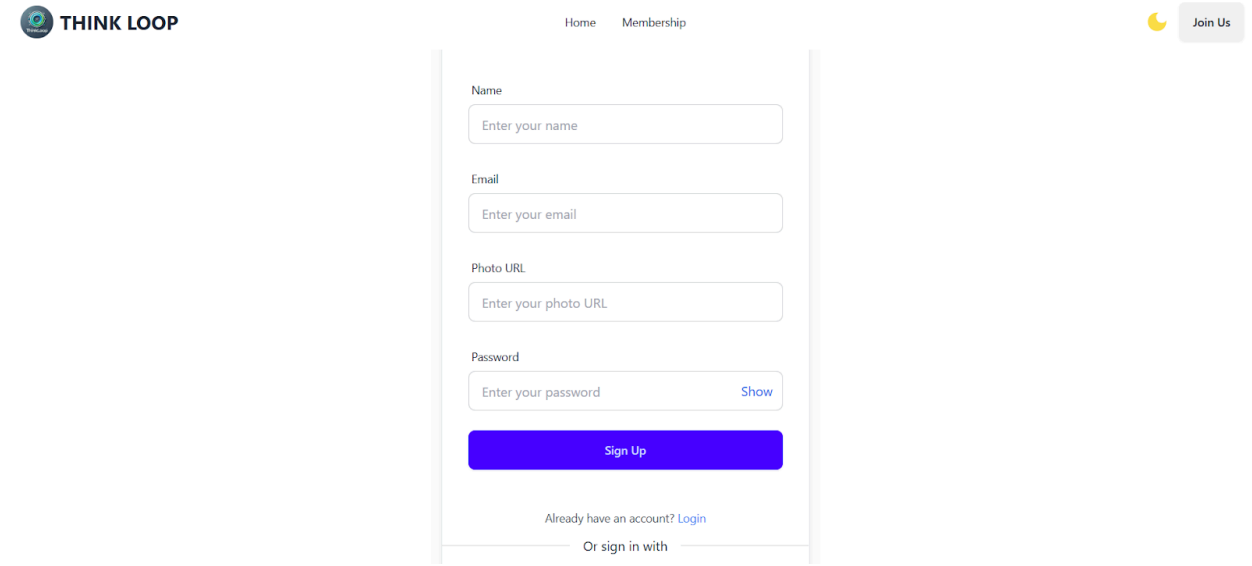


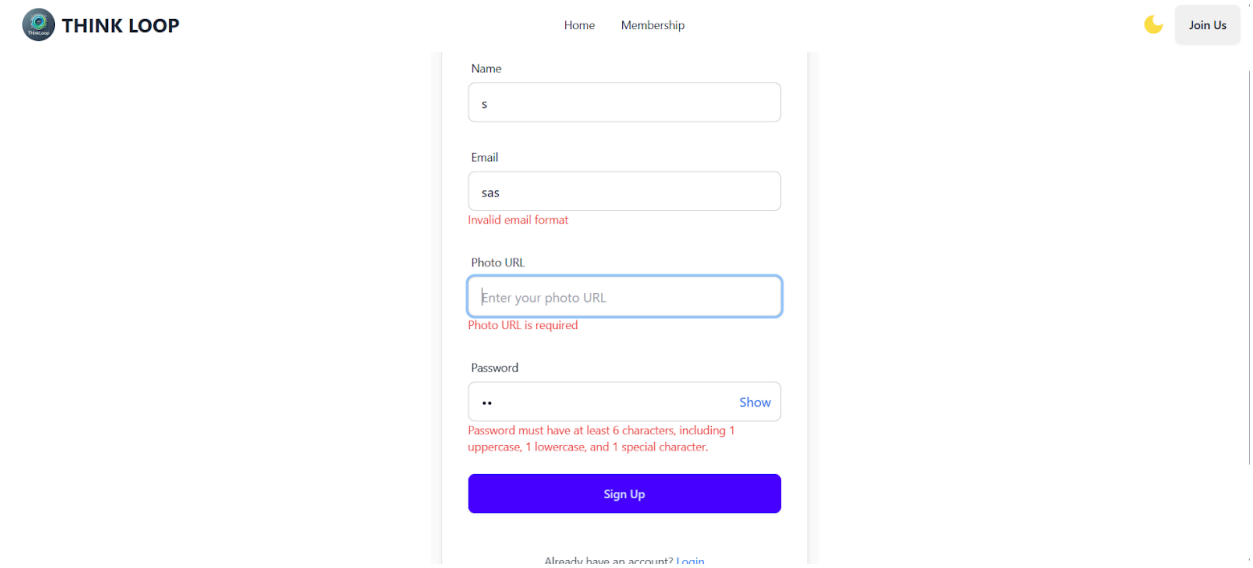


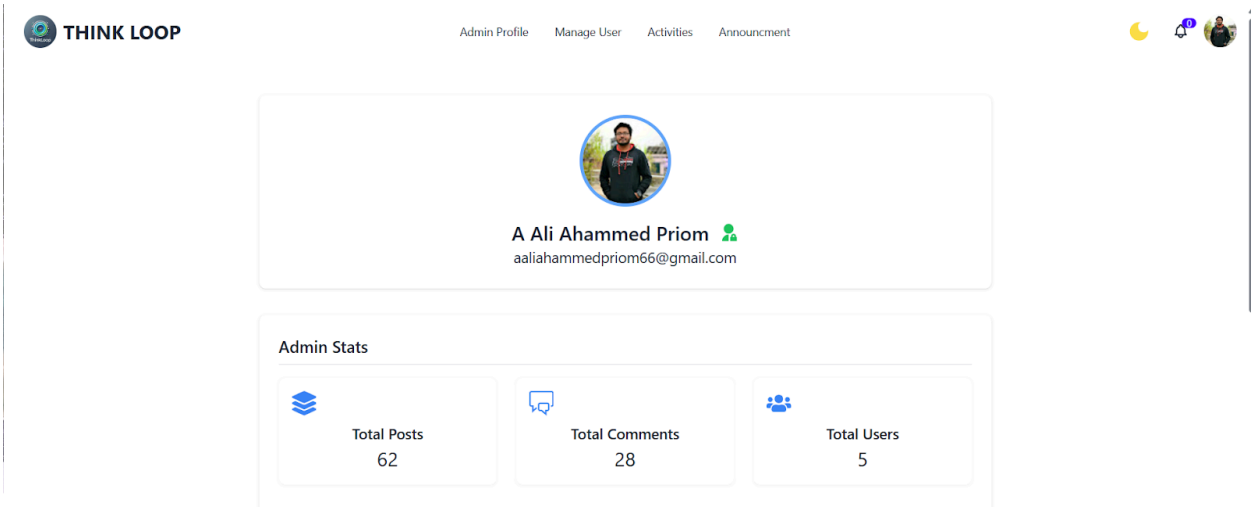


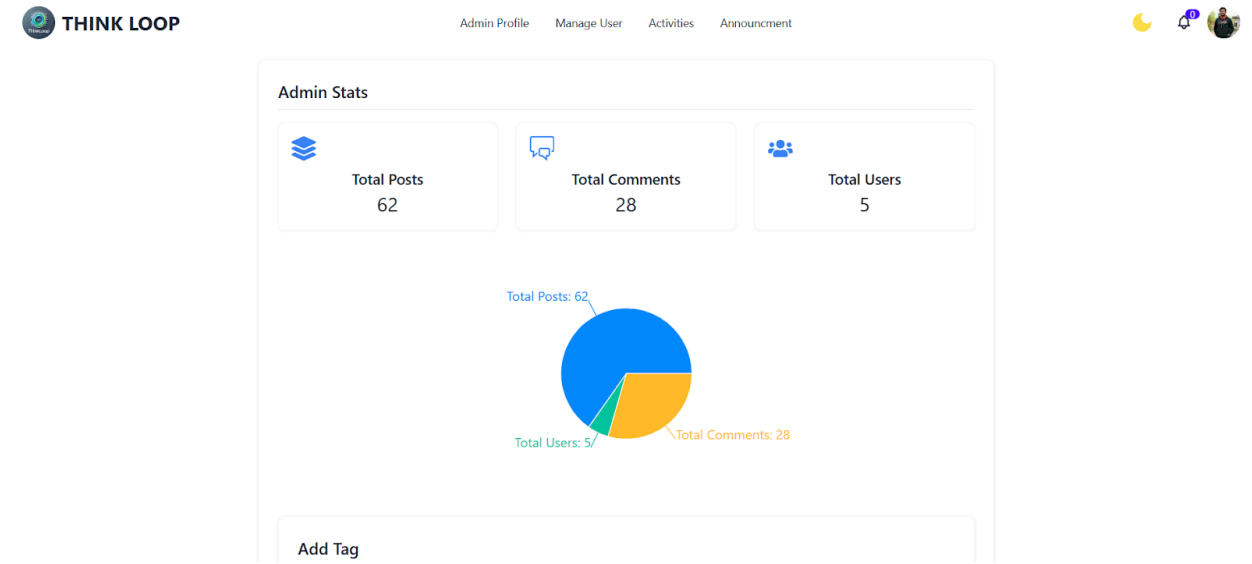


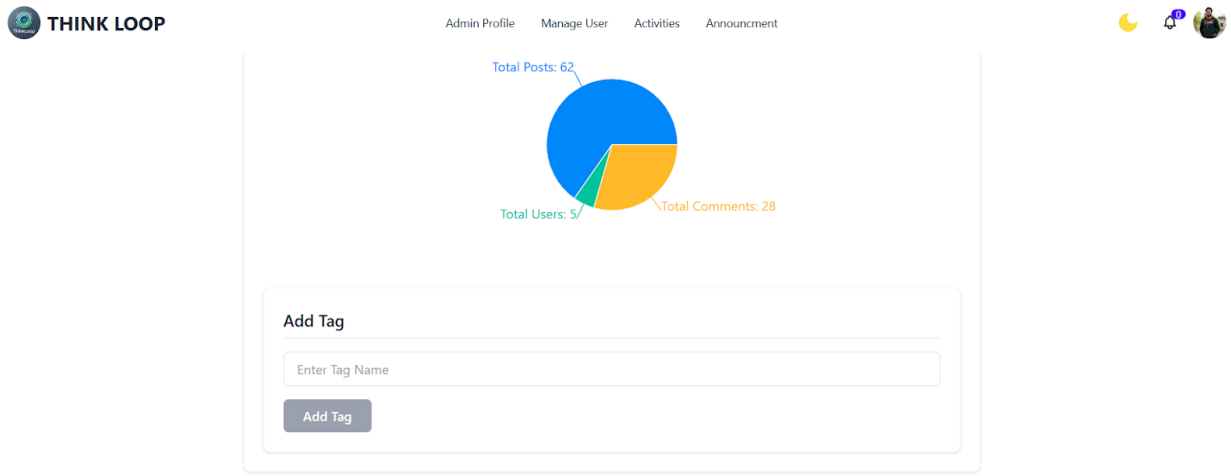


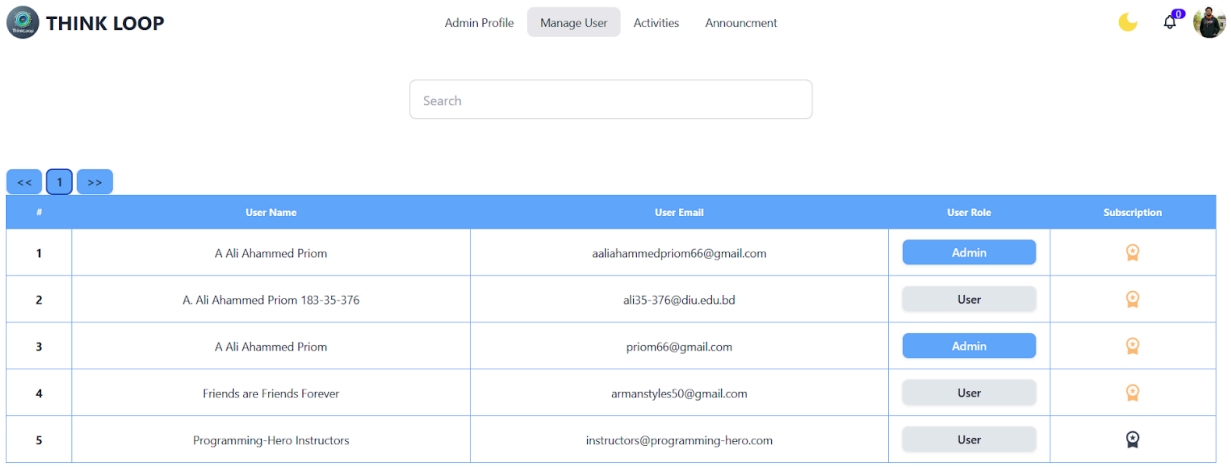


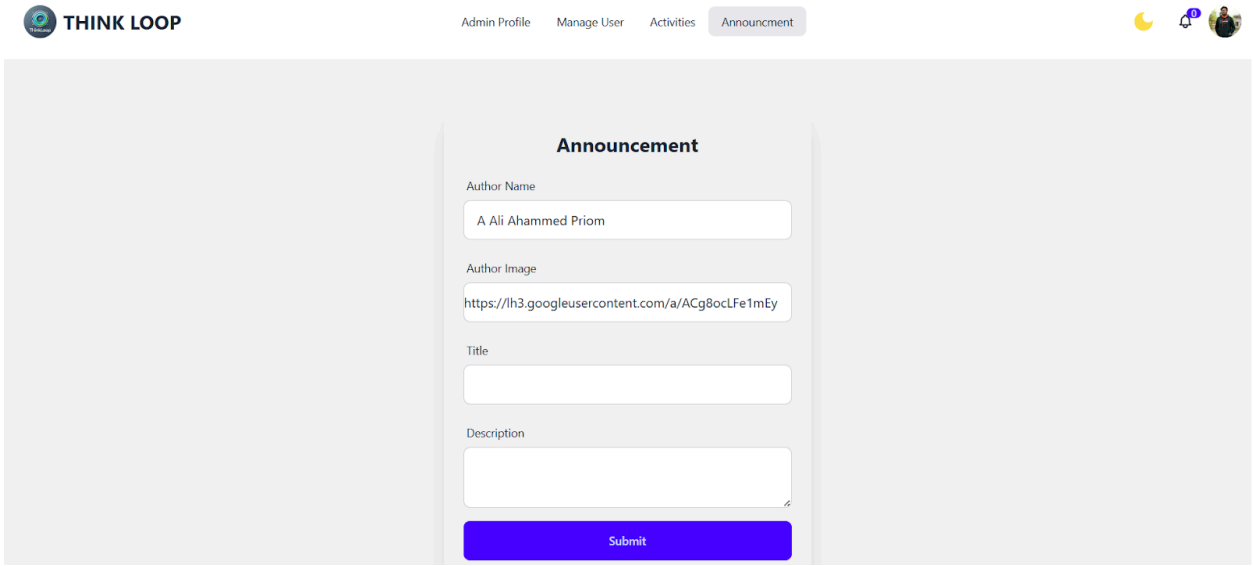


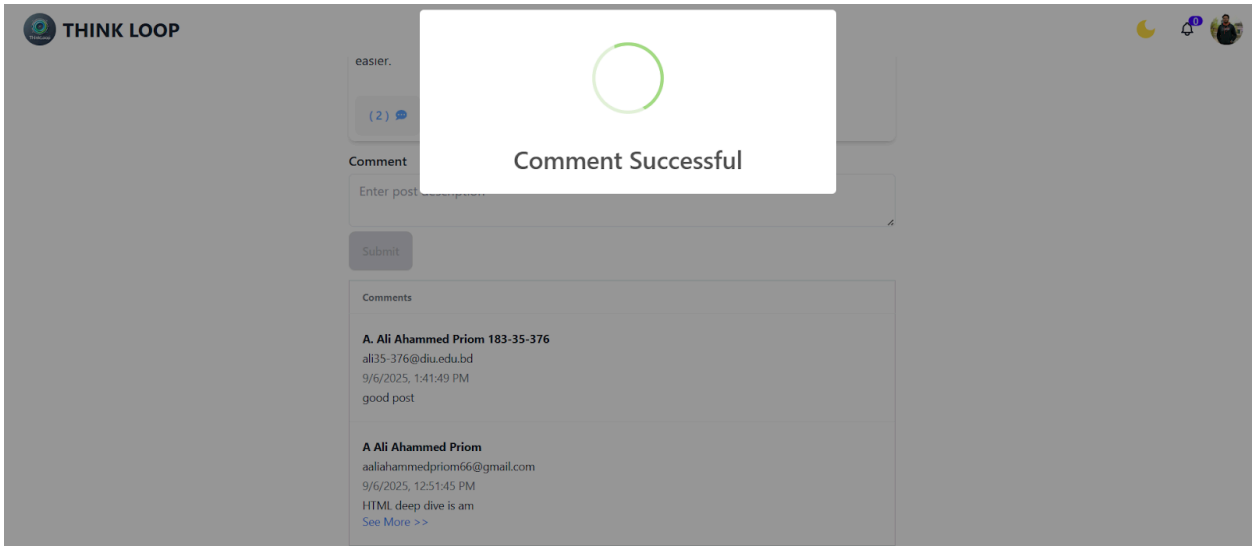


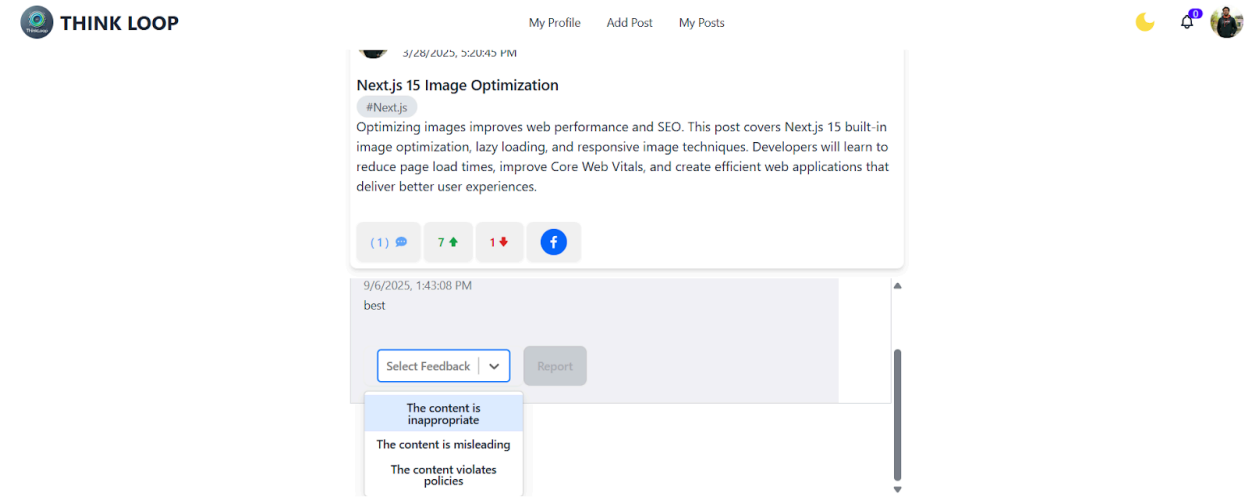


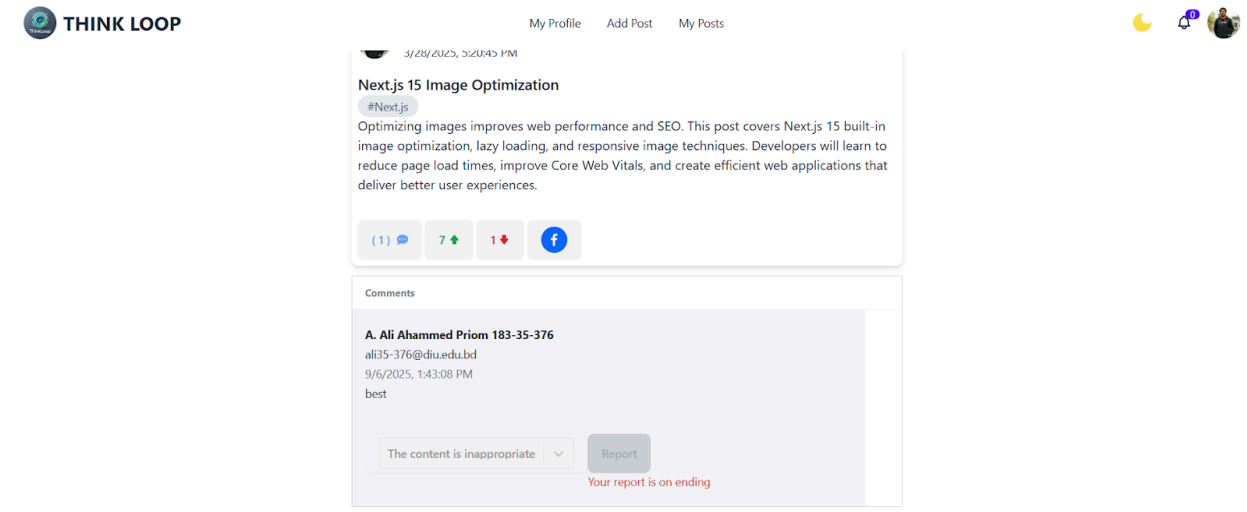


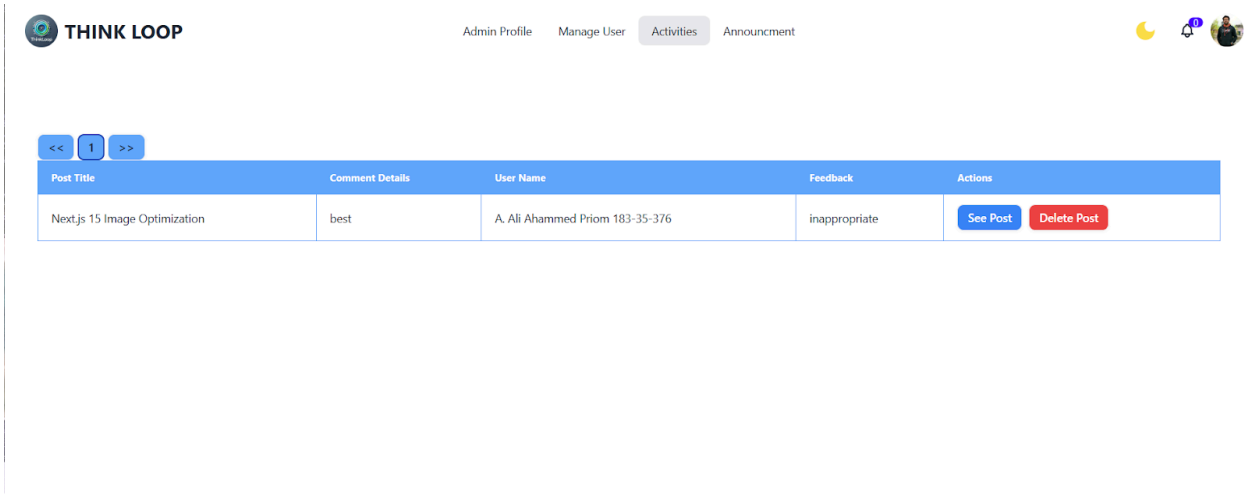




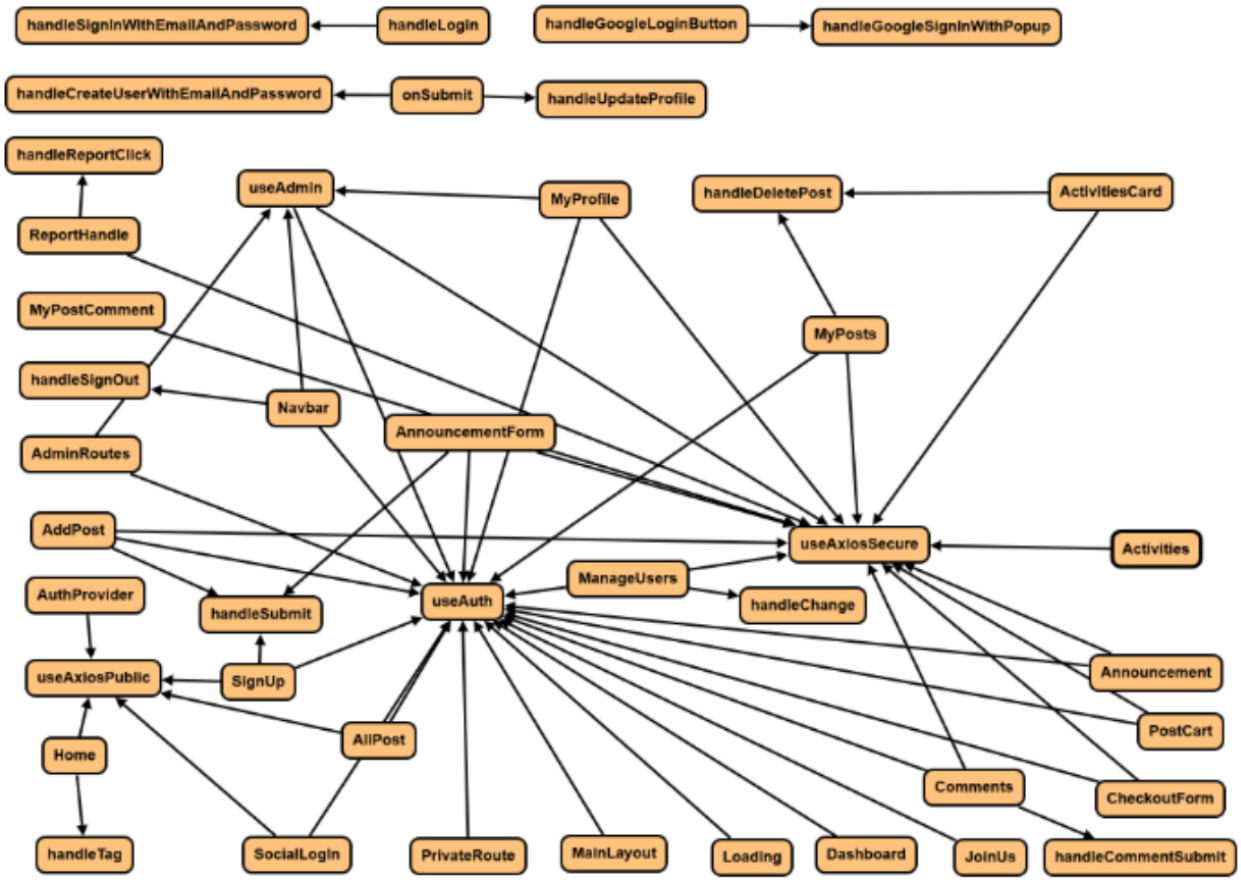








2.11 Functional Calls



Chapter 3: Software Testing

3.1 Testing Features

Feature Category	Description
User Authentication	Login via Email and Social accounts (Google)
JWT Token Validation	Secure token-based access to protected routes
Session Management	Maintains user session, auto-logout on token expiry
Role-based Access Control	Access restrictions for Admin and User roles
Post/Comment Creation	Users can create, edit, and delete posts and comments
Admin Announcement Posting	Admins can post global announcements
Dashboard & Analytics	Visual stats for user activity and post engagement
Search, Tag Filter, Pagination	Search posts, filter by tags, and paginate large datasets
Error Handling & Input Validation	Validates inputs and shows user-friendly error messages

3.2 Testing Strategies

Testing Strategy	**Description**
Unit Testing	Tested backend functions like user registration and token generation. Used Jest and Supertest for route-level testing.
Integration Testing	Verified communication between frontend and backend. Focused on API requests and MongoDB responses.
Manual Testing	Conducted UI testing for post creation, voting, comment deletion, and admin actions. Performed cross-browser testing (Chrome, Firefox, Edge) and device testing (desktop, tablet, mobile).
UI/UX Testing	Observed form validation, feedback messages, loading states. Verified consistent styling with Tailwind CSS.
Security Testing	Tested JWT expiration handling. Checked input sanitization to prevent XSS. Verified RBAC middleware to prevent privilege escalation.

3.3 Test Case Format

Test Case ID	Description	Input	Expected Output	Actual Result	Status
TC001	Login with invalid credentials	Email + wrong password	“Invalid credentials” msg	As Expected	Pass
TC002	JWT token validation	Expired token	Access denied (401)	As Expected	Pass
TC003	Post creation	Valid post data	Post saved to DB	As Expected	Pass
TC004	Unauthenticated post creation	No JWT token	403 Forbidden	As Expected	Pass
TC005	Upvote once per user	Repeated upvote	No multiple votes allowed	As Expected	Pass

TC006	Admin-only route access	Regular user token	Access denied (403)	As Expected	Pass
TC007	Responsive layout	Resize browser	UI adapts without overflow	As Expected	Pass
TC008	Search by tag	“#security” tag entered	Filtered results shown	As Expected	Pass
TC009	Comment deletion (admin only)	User tries to delete	Error message shown	As Expected	Pass
TC010	Firebase social login	Google account	Logged in, token issued	As Expected	Pass

3.4 Bug Fixes Log (Sample)

Issue No	Description	Severity	Status	Fix Summary
1	Token not refreshing on logout/login	High	Resolved	Added refresh token handler in auth flow
2	Upvotes counted twice	Medium	Resolved	Added DB constraint for unique vote per user
3	Mobile layout breaks at 320px	Low	Fixed	Adjusted Tailwind breakpoints
4	Admin user was not verified properly	Critical	Fixed	Refined `verifyAdmin` middleware
5	Comment deletion failed for post owner	Medium	Fixed	Updated permission check to include post owners
6	Pagination broke after filtering by tag	High	Resolved	Recomputed total pages after filtered query
7	Social login not working on Safari	Medium	Fixed	Adjusted OAuth callback handler for Safari support

8	Unescaped HTML rendered in post titles	Critical	Resolved	Implemented server-side sanitization before render
9	Duplicate usernames allowed on registration	High	Resolved	Added unique index and validation on username field
10	Dashboard analytics not updating in real-time	Low	Fixed	Added polling interval and cache invalidation logic

3.5 Tools Used for Testing

Tools	**Purpose**
Jest	Unit testing (Node backend)
Supertest	API endpoint testing
Firebase Console	Social login monitoring
Postman	API testing manually
Browser DevTools	Mobile view / console errors

3.6 Test Coverage Summary

Module	**Test Coverage**
Authentication	100%
User Dashboard	95%
Admin Dashboard	90%
Post/Comment APIs	100%
Voting Mechanism	100%
UI Responsiveness	80% (manual)
Security (XSS/JWT)	90% tested

Chapter 4: Deployment and Maintenance

4.1 Agile Sprint Plan

Sprint #	Duration	Goals/Features	Tasks	Outcome
Sprint 1	2 weeks	Setup & Initial Deployment	Setup repo, basic backend API, frontend scaffold, MongoDB Atlas setup	Basic user authentication implemented
Sprint 2	2 weeks	Core Features—Posts & Comments	CRUD for posts, comments, pagination, JWT authentication, basic role-based access control	Working posts and comments features with user authentication and role-based route protection
Sprint 3	2 weeks	Admin Features & Analytics	Admin dashboard, user management, post moderation, pie chart analytics	Admin can manage users, moderate posts, and view basic analytics
Sprint 4	2 weeks	UI/UX Improvements & Security Enhancements	Responsive styling with Tailwind, input validation, backend validation, enhanced security	Improved UI responsiveness, validated inputs, more secure authentication and authorization
Sprint 5	2 weeks	Social Features & Final Testing	Social sharing, upvote/downvote system, real-time updates testing	Fully functional voting system, social share, ready for production staging
Sprint 6	1 week	Production Deployment & Bug Fixes	Final deployment to Netlify/Vercel, bug fixing	Stable production release with monitoring and rollback plans ready
Ongoing	Continuous	Maintenance & Minor Feature Updates	Address bugs, performance improvements, add enhancements	Continuous project improvement following Agile principles

4.2 Release Schedule

Release Version	Features Included	Notes
v0.1 (Alpha)	Basic login, post creation, comment system	Internal testing only
v0.5 (Beta)	Admin dashboard, voting system, pagination	Beta testing with limited users
v1.0 (Production)	Full feature set: social sharing, analytics	Public launch, full documentation
v1.1	Bug fixes and minor UI enhancements	Patch release
v1.2	Performance optimizations, backup enhancements	Maintenance release

4.3 Risk Management Table

Risk Description	Likelihood	Impact	Mitigation Strategy	Status
Server downtime causing user impact	Medium	High	Use reliable hosting (Render, Netlify), set up monitoring	Planned
Security breach (unauthorized access)	Low	High	Use JWT auth, HTTPS, role-based access control, secure env vars	Implemented
Data loss due to database failure	Low	High	Automated backups with MongoDB Atlas, test recovery process	Implemented
Bugs causing crashes or data issues	Medium	Medium	Agile sprint bug fixes, thorough testing in staging	Ongoing

Performance degradation under load	Medium	Medium	MongoDB aggregation for optimized queries, pagination	Planned
Feature creep delaying releases	Medium	Low	Strict sprint scope, prioritize MVP features	Managed
Environment variable exposure	Low	High	Use .env files, never commit sensitive info to GitHub	Implemented

Chapter 5: User Manual

5.1 Getting Started

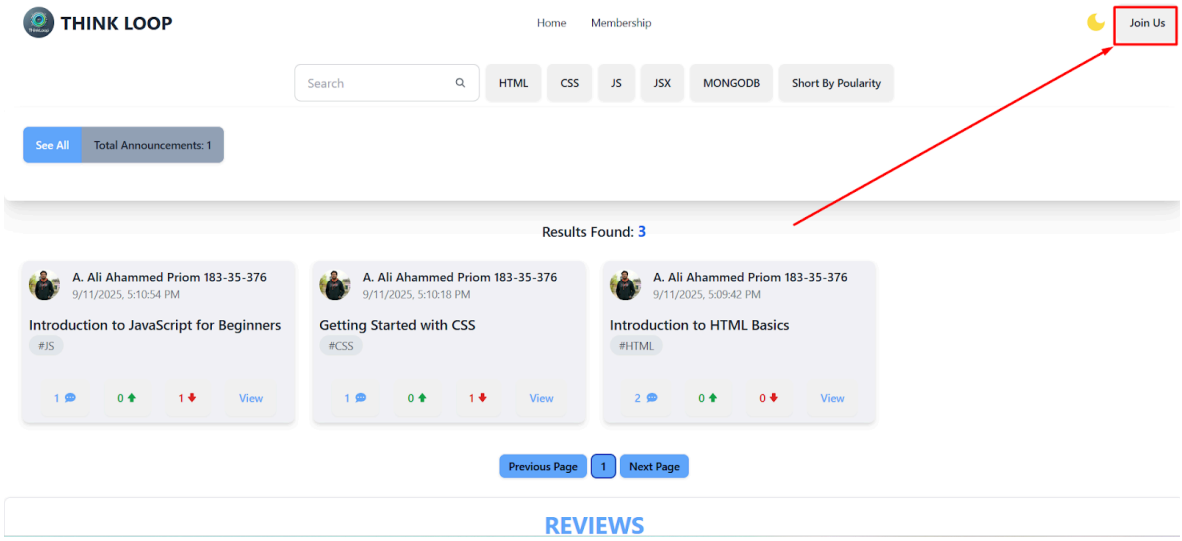
5.1.1 Accessing the Website

- Open the live site URL in any modern browser.
  https://mellow-mandazi-3e1f97.netlify.app/

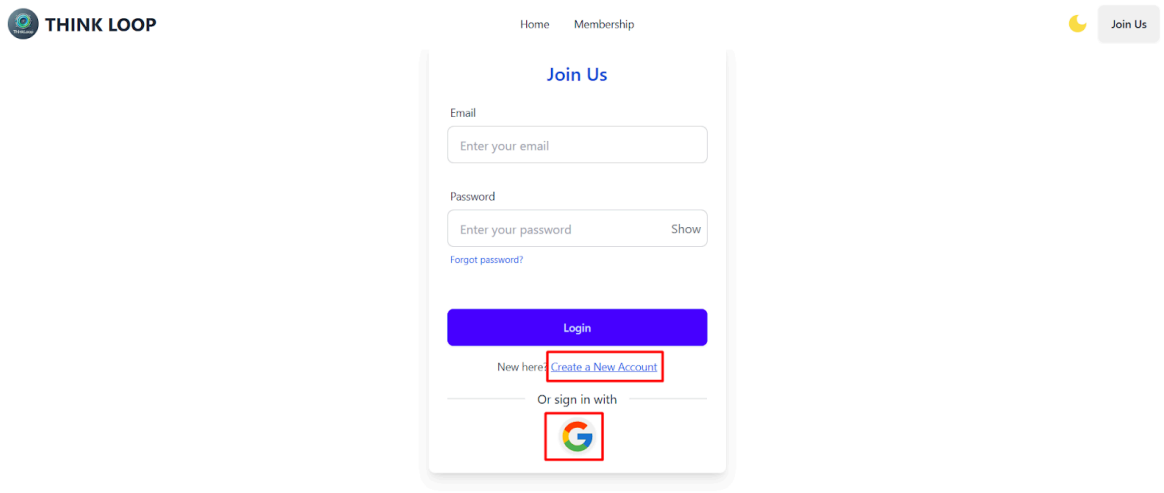
- The platform supports both desktop and mobile devices.

5.1.2 Creating an Account

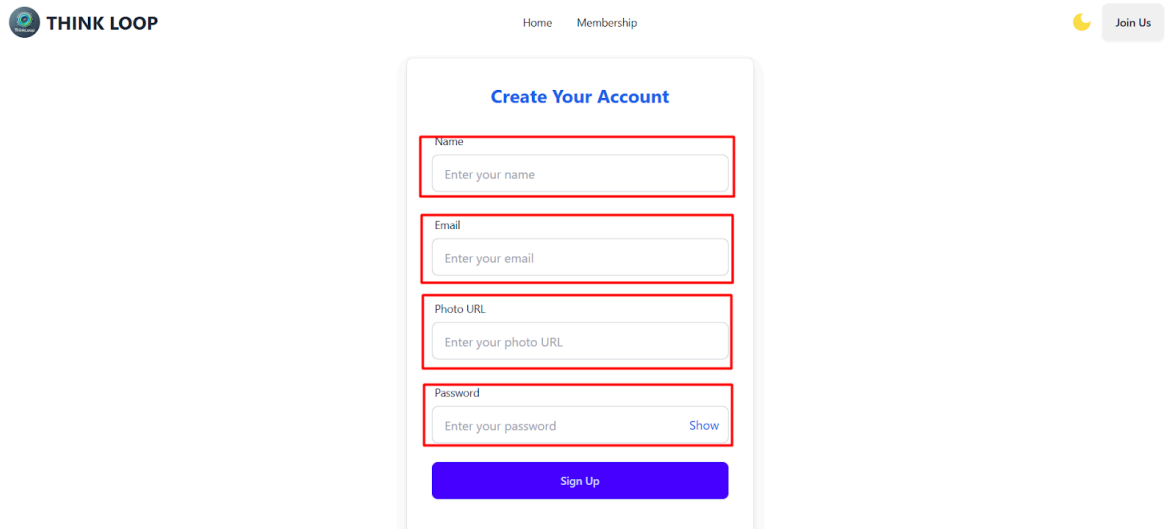
- Click "Join Us" on the homepage.



- Choose a registration method:
  - Email and password
  - Social login (e.g., Google)



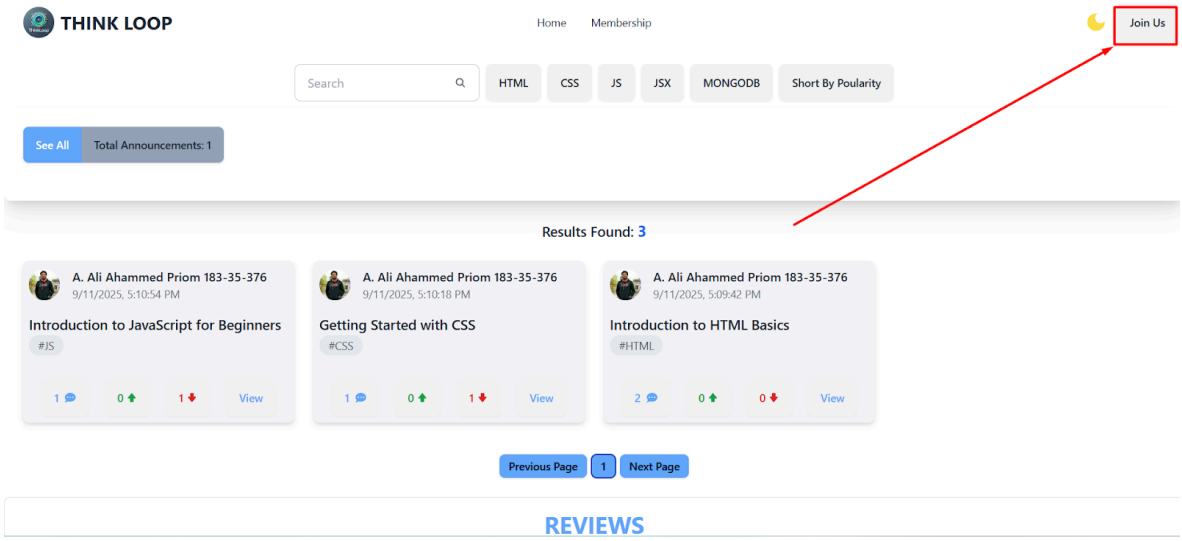
- Fill in the required fields and submit the form.



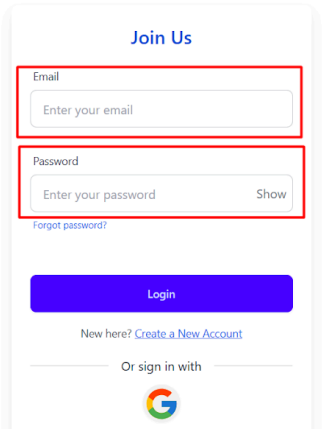
- If prompted, verify your email via the verification link sent to your inbox.

5.1.3 Logging In

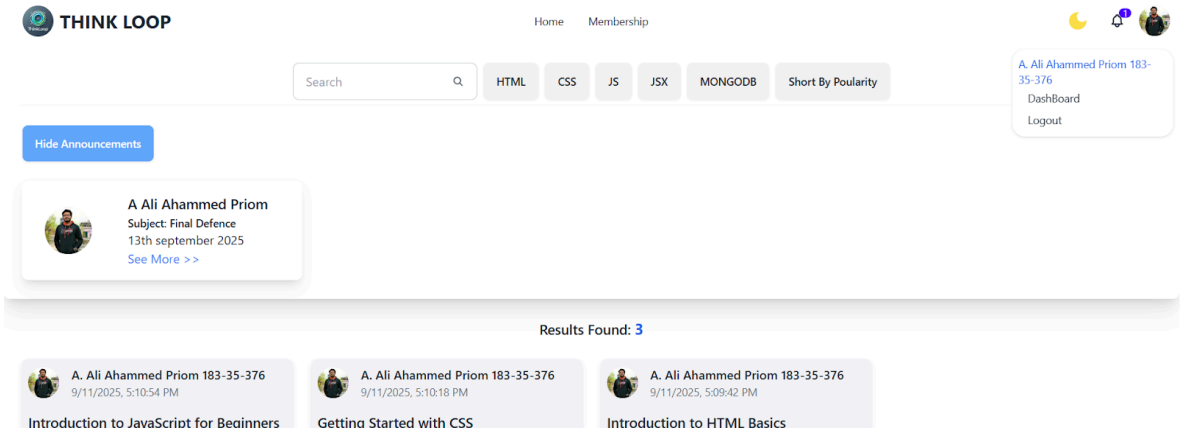
- Navigate to the "Join Us" (login) page.



- Enter your registered email and password, or choose a social login method.



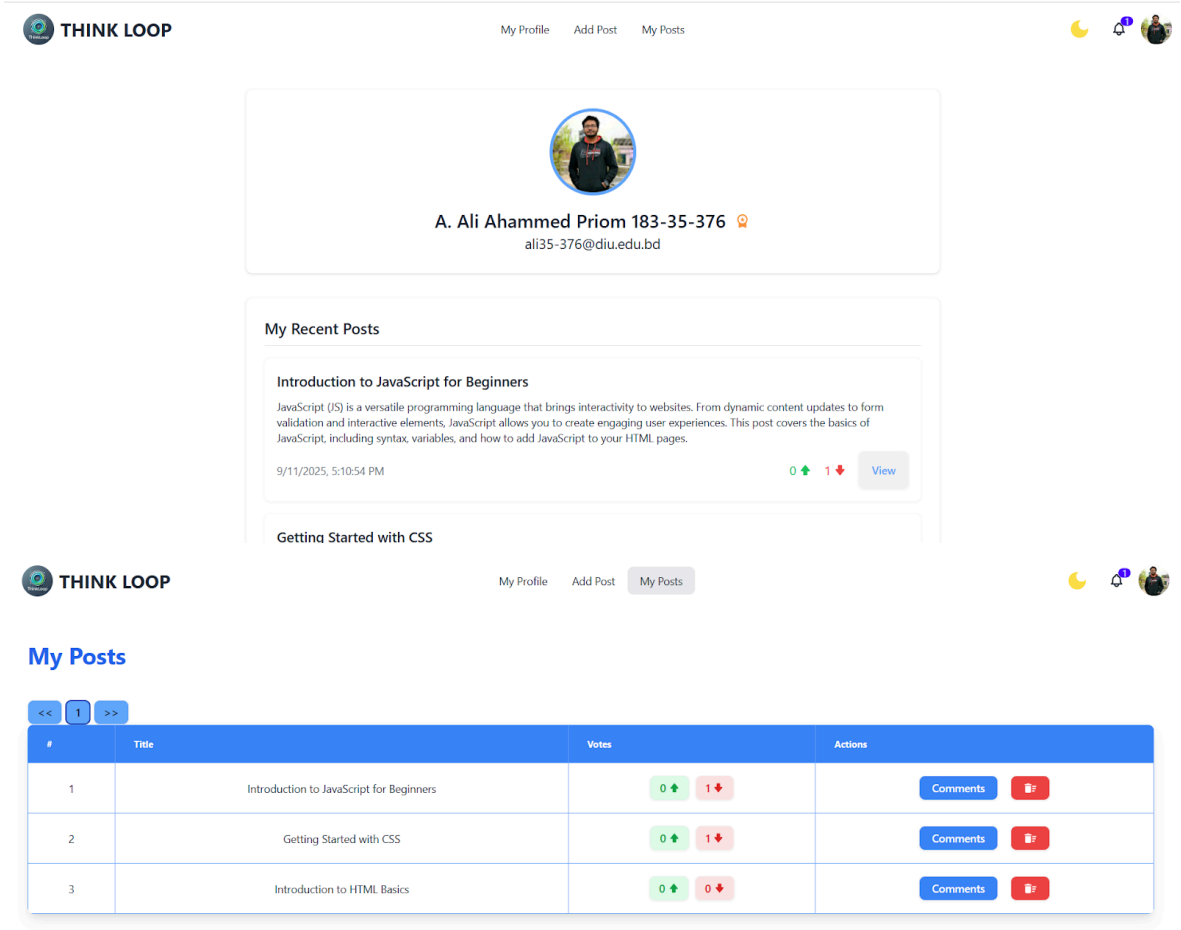
- Upon successful authentication, you will be redirected to your user dashboard.



5.2 User Features

5.2.1 Dashboard

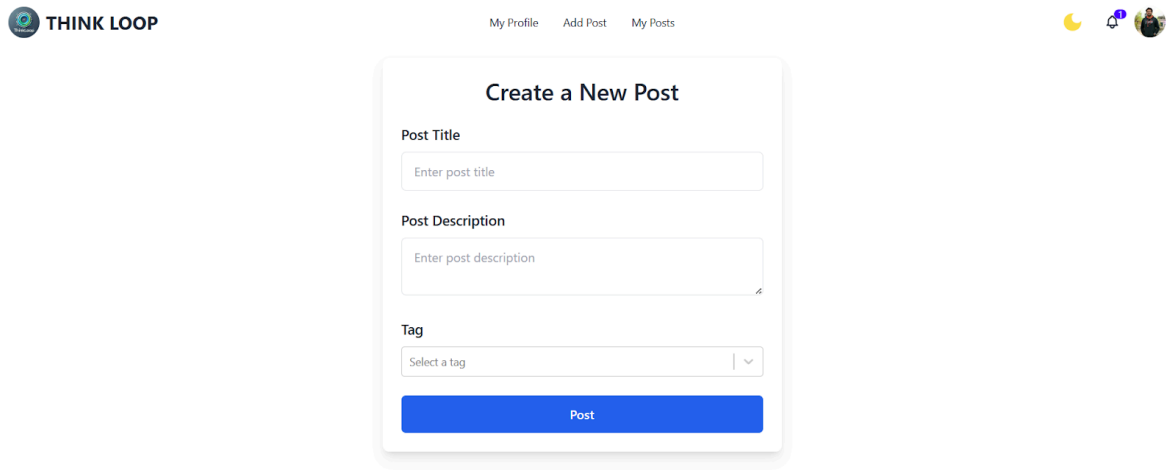
- View your profile details, earned badges, and membership status.



- Manage your own posts and comments directly from the dashboard.

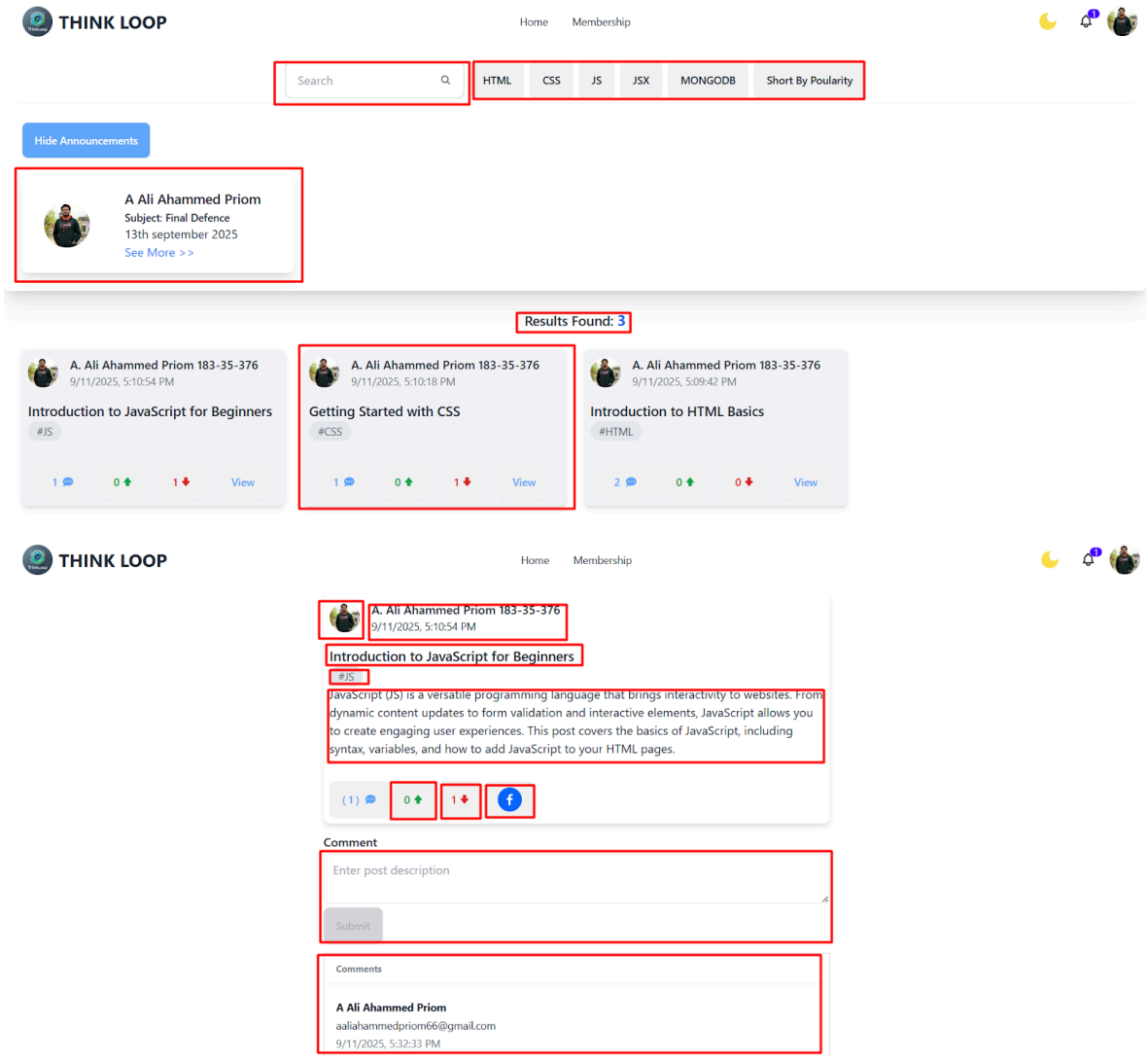
5.2.2 Creating Posts

1. Click the "Add Post" button.
2. Enter the title and content, and select relevant tags.
3. Click Submit to publish your post to the public forum.



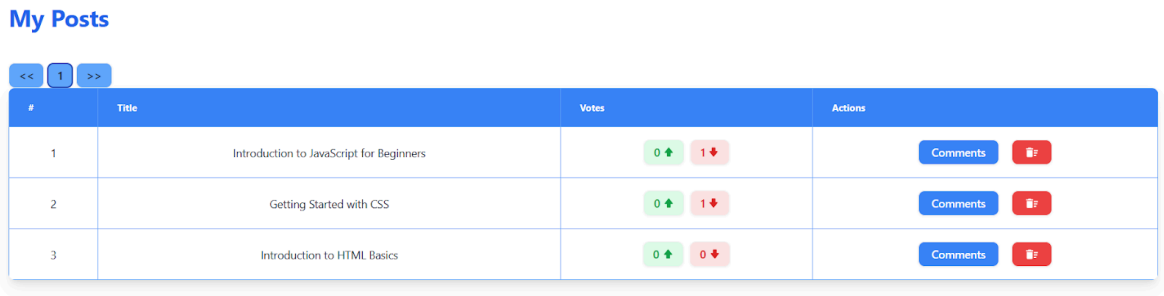
5.2.3 Viewing and Interacting with Posts

- Browse posts on the homepage, or use search and tag filters.
- Sort posts by popularity, date, or tags.
- Upvote or downvote posts to share your opinion.
- Add comments to participate in discussions.



5.2.4 Managing Your Posts and Comments

- Go to "My Posts" to delete your posts.
- Navigate to the "Comments" section to manage your interactions.

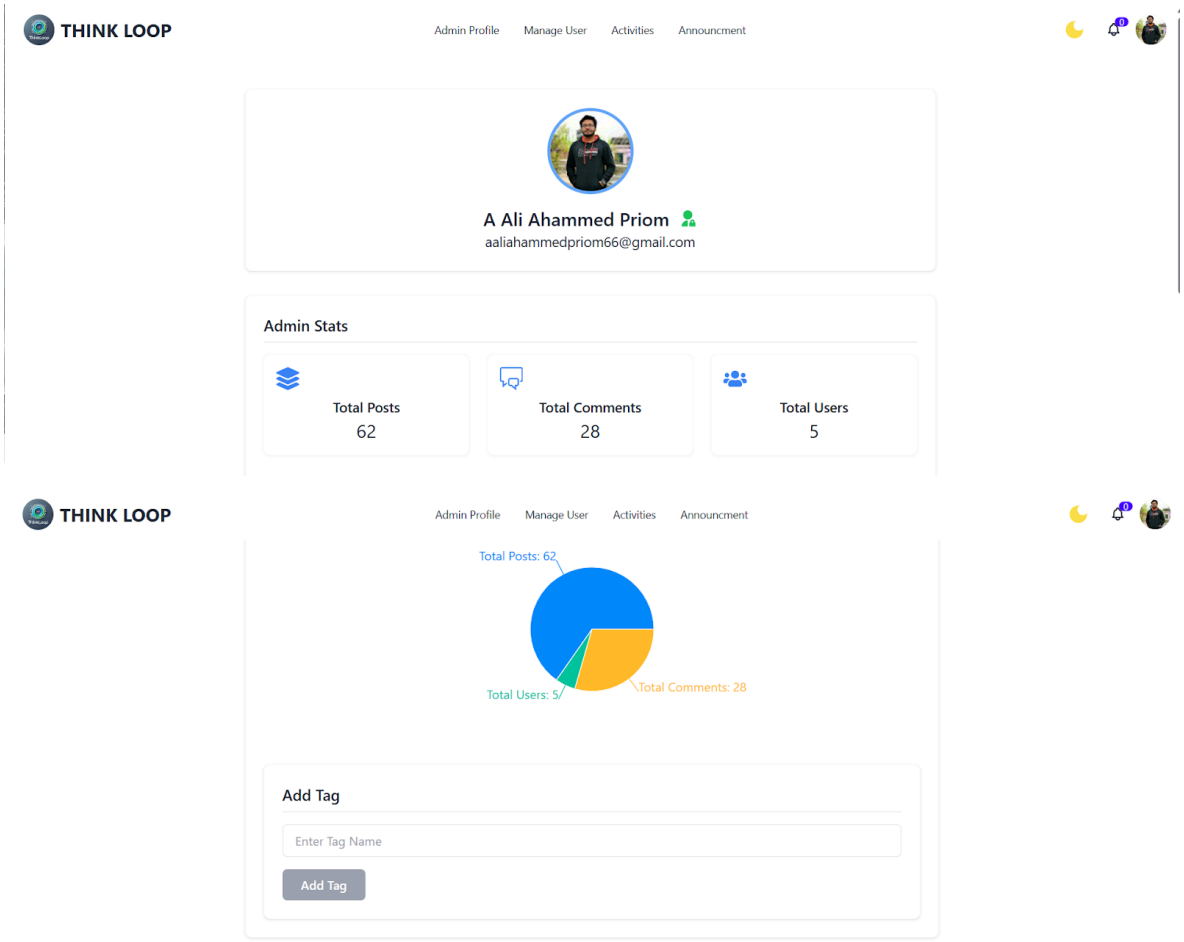


5.3 Admin Features

5.3.1 Admin Dashboard

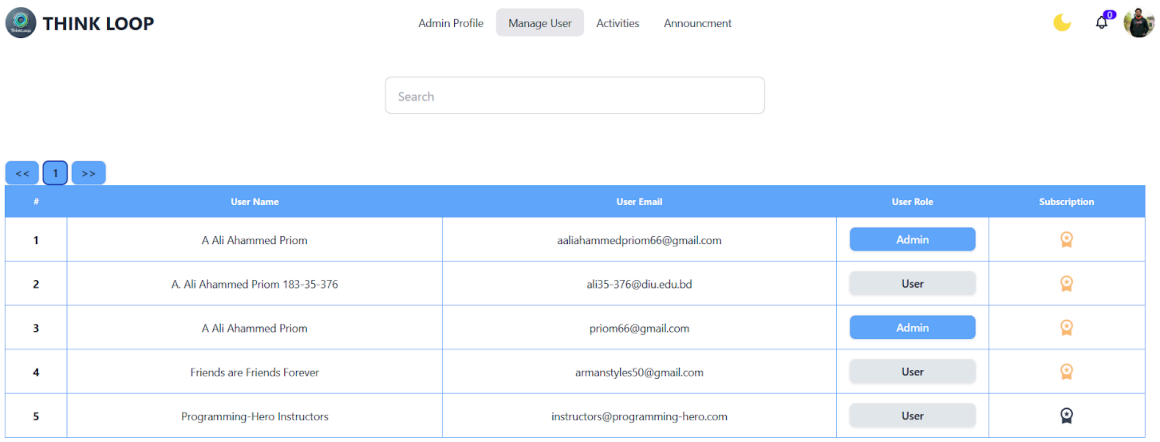
- Access centralized tools for user management, site announcements, and platform analytics.

- View a list of registered users, assign or revoke admin roles, and moderate reported content.



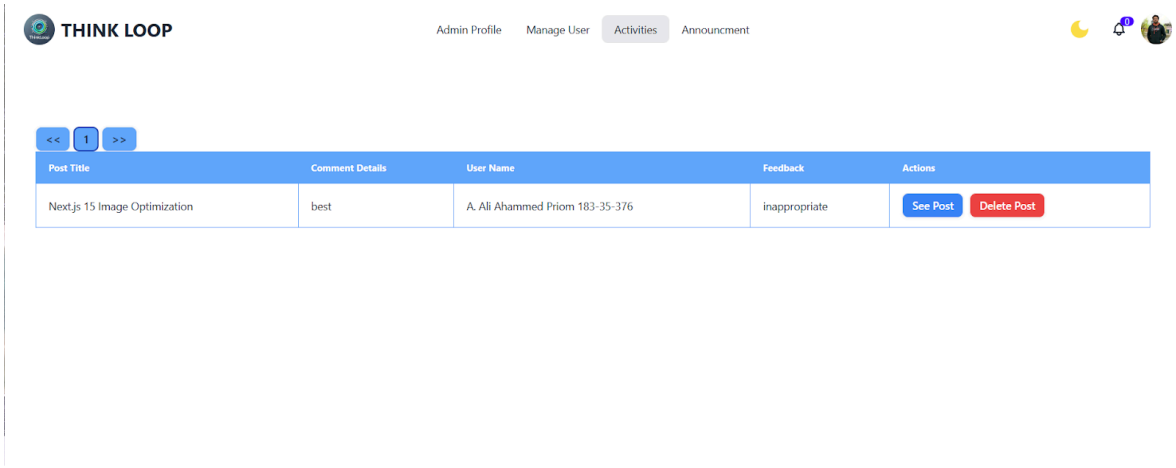
5.3.2 Managing Users

- Search and filter users.
- Promote or demote users to/from admin status.
- Remove users who violate community guidelines.



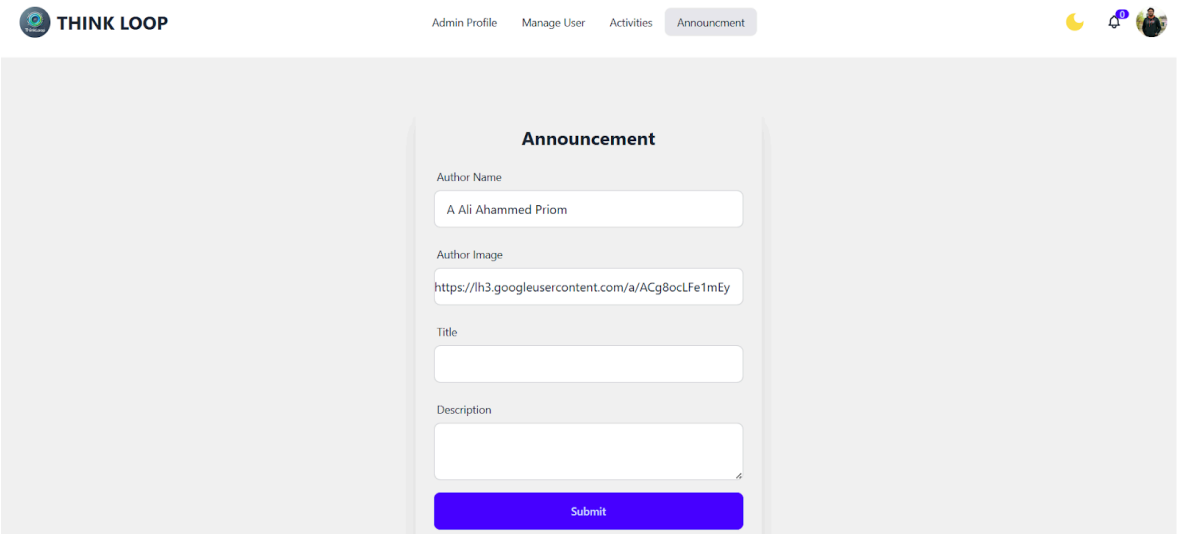
5.3.3 Moderation

- Review reported posts or comments.
- Delete or take action on inappropriate or harmful content.



5.3.4 Announcements

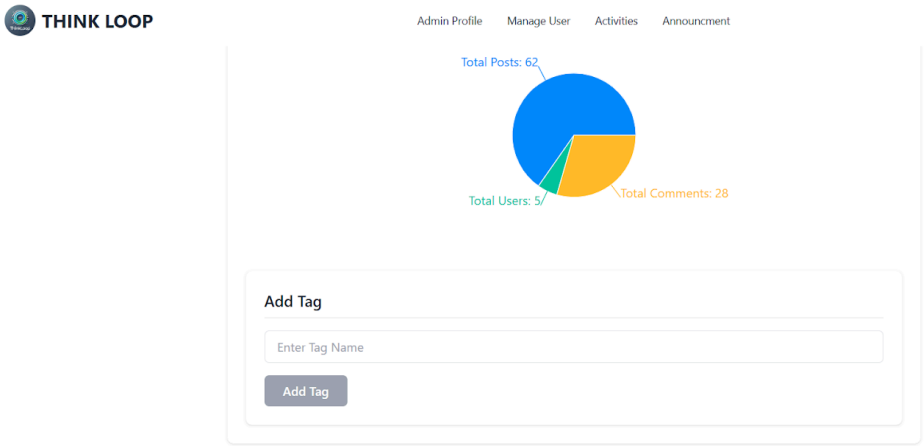
- Create new announcements for the entire community.
- Publish notices that appear on the homepage or dashboard of all users.



5.3.5 Analytics

- View interactive charts and summary reports.

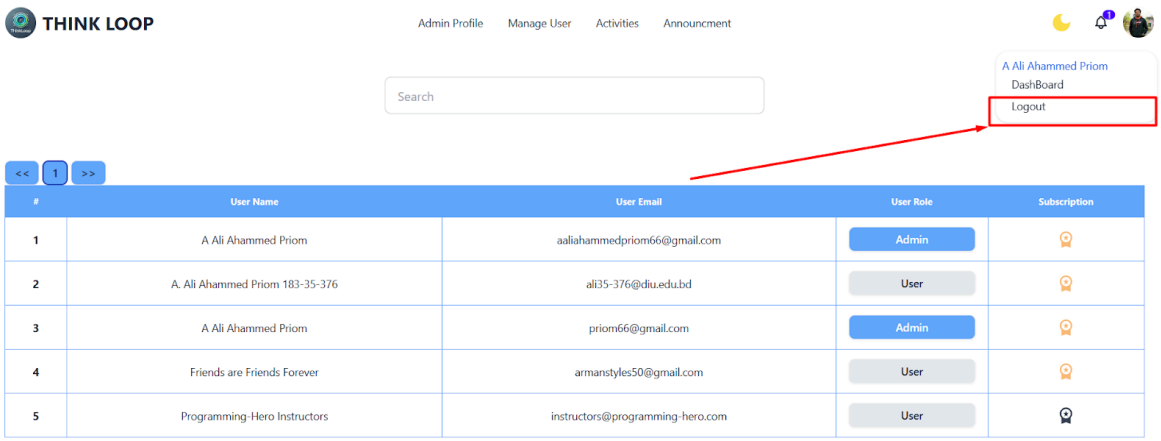
- Monitor user activity, post engagement, and overall platform performance using visual dashboards (e.g., pie charts, line graphs).



5.4 Security & Account Management

5.4.1 Logout

Use the logout button in your profile or dashboard to end your session securely.



5.4.2 Session Security

Sessions use JWT tokens to ensure secure, authenticated access.

5.5 Troubleshooting & Support

5.5.1 Common Issues

- Login problems
  Ensure that your email and password are correct. Check for typos or try resetting your password if needed.

- Post submission errors
  Make sure all required fields (title, content, tags) are filled in before submitting.

- Slow loading
  Try refreshing the page, or clear your browser cache. Also check your internet connection.

5.5.2 Contact Support

If the issue persists, you can reach out for support via :

- GitHub Issues (for bug reports or feature requests):
  https://github.com/aaliahammedpriom/think-loop-client

- Email Support (for general queries or direct help):
  ali35-376@diu.edu.bd

Chapter 6: Developer Manual

6.1 Live Resources

- **Live Site:** https://mellow-mandazi-3e1f97.netlify.app/
- **Git Repositories:**
    - https://github.com/aaliahammedpriom/think-loop-client
    - https://github.com/aaliahammedpriom/think-loop-server

6.2 Installation & Setup

6.2.1 Clone Repositories

Frontend: git clone https://github.com/aaliahammedpriom/think-loop-client

Backend: git clone https://github.com/aaliahammedpriom/think-loop-server

6.2.2 Backend Setup

1. npm install
2. Create .env file with:
3. PORT=5000
4. MONGO_URI=<your_mongo_uri>
5. JWT_SECRET=<your_jwt_secret>
6. FIREBASE_API_KEY=<your_firebase_api_key>
7. npm start

6.2.3 Frontend Setup

1. npm install
2. Create .env file with: REACT_APP_FIREBASE_API_KEY=<your_firebase_api_key>
3. npm start

6.2.4 Admin Access

- **Email:** priom66@gmail.com
- **Password:** priom66@gmail.comA

6.2.5 Folder Structure FrontEnd

Directory structure:

```
├── README.md
├── eslint.config.js
├── firebase
│   └── firebase.config.js
├── index.html
├── package.json
├── postcss.config.js
├── public
│   └── _redirects
├── src
│   ├── Hooks
│   │   ├── useAdmin.jsx
│   │   ├── useAuth.jsx
│   │   ├── useAxiosPublic.jsx
│   │   └── useAxiosSecure.jsx
│   ├── Layouts
│   │   ├── Dashboard.jsx
│   │   └── MainLayout.jsx
│   ├── Pages
│   │   ├── Activities
│   │   │   ├── Activities.jsx
│   │   │   └── ActivitiesCard.jsx
│   │   ├── Add Post
│   │   │   └── AddPost.jsx
│   │   ├── Announcement Form
│   │   │   └── AnnouncementForm.jsx
│   │   ├── Comments
│   │   │   ├── CommentDescription.jsx
│   │   │   └── Comments.jsx
│   │   ├── Home
│   │   │   ├── AllPost.jsx
│   │   │   ├── Announcement.jsx
│   │   │   ├── Banner.jsx
│   │   │   ├── ContactUs.jsx
│   │   │   ├── ForumFaq.jsx
```

```
│   │   │   ├── ForumTrivia.jsx
│   │   │   ├── Home.jsx
│   │   │   ├── Legal.jsx
│   │   │   └── ReviewSlider.jsx
│   │   ├── Join Us
│   │   │   ├── Log In
│   │   │   │   └── JoinUs.jsx
│   │   │   └── Sign Up
│   │   │       └── SignUp.jsx
│   │   ├── Manage Users
│   │   │   └── ManageUsers.jsx
│   │   ├── Membership
│   │   │   └── Membership.jsx
│   │   ├── My Post Comment
│   │   │   └── MyPostComment.jsx
│   │   ├── My Posts
│   │   │   └── MyPosts.jsx
│   │   ├── My Profile
│   │   │   └── MyProfile.jsx
│   │   ├── Post Cart
│   │   │   └── PostCart.jsx
│   │   ├── Report Handle
│   │   │   └── ReportHandle.jsx
│   │   ├── Shared
│   │   │   ├── Error.jsx
│   │   │   ├── Footer.jsx
│   │   │   ├── Loading.jsx
│   │   │   └── Navbar.jsx
│   │   ├── payment
│   │   │   ├── CheckoutForm.jsx
│   │   │   └── Payment.jsx
│   │   └── social login
│   │       └── SocialLogIn.jsx
│   ├── Provider
│   │   └── AuthProvider.jsx
│   ├── Routes
│   │   ├── AdminRoutes.jsx
│   │   ├── PrivateRoute.jsx
│   │   └── Router.jsx
│   ├── index.css
│   └── main.jsx
├── tailwind.config.js
└── vite.config.js
```

6.2.6 Folder Structure BackEnd

Directory structure:

```
├── README.md
├── index.js
├── package.json
└── vercel.json
```

Chapter 7: Project Summary

7.1 Project Overview

The MERN Forum Project is a full-stack web application designed to facilitate user discussions through posts and comments. Built using MongoDB, Express.js, React.js, and Node.js, the project offers a scalable, secure, and user-friendly platform with real-time features, role-based access, and admin moderation.

7.2 Achievements

- Developed a fully functional forum with user registration, login, and authentication secured via JWT and Firebase.
- Implemented core forum features such as post creation, commenting, voting, and tag-based search with pagination.
- Designed a responsive UI with Tailwind CSS ensuring accessibility across devices.
- Created an admin dashboard for user management, content moderation, announcements, and analytics.
- Utilized MongoDB aggregation for efficient data retrieval and analytics visualization.
- Deployed the frontend and backend on modern cloud platforms with environment variable security.

7.3 Challenges Faced

- Implementing secure role-based access control to protect admin routes and sensitive data.
- Managing pagination and sorting efficiently on large datasets using MongoDB aggregation pipelines.
- Ensuring seamless integration between React frontend and Express backend with real-time updates.
- Balancing feature completeness with deployment and maintenance requirements under project timelines.

7.4 Lessons Learned

- Agile development and continuous integration improve project adaptability and quality.
- Proper environment and secret management are critical for security in deployed apps.
- Building reusable React components and centralized state management facilitates scalability.
- Effective backend validation complements frontend checks to maintain data integrity.
- Thorough testing across development and staging environments reduces production issues.

7.5 Future Work

- Add real-time notifications using WebSocket technology for instant user updates.
- Expand comment reporting with analytics for better community moderation.
- Introduce advanced admin analytics with deeper insights into user engagement.
- Enhance mobile experience with a dedicated React Native app.
- Implement further performance optimizations and security audits.

7.6 Conclusion

This project successfully demonstrates the full development lifecycle of a modern web forum, from planning through deployment and maintenance. It highlights the power of the MERN stack combined with agile practices to build scalable, maintainable, and user-focused applications.

References

- MongoDB Docs: https://www.mongodb.com/docs/
- Mongoose Guide: https://mongoosejs.com/docs/guide.html
- Express.js Docs: https://expressjs.com/
- React.js Docs: https://react.dev/
- React Router: https://reactrouter.com/
- Node.js Docs: https://nodejs.org/en/docs/
- JWT Introduction: https://jwt.io/introduction/
- Node.js + Express JWT Tutorial: https://www.digitalocean.com/community/tutorials/nodejs-jwt-expressjs
- Firebase Auth: https://firebase.google.com/docs/auth
- bcrypt npm: https://www.npmjs.com/package/bcrypt
- Tailwind CSS Docs: https://tailwindcss.com/docs
- Material-UI Docs: https://mui.com/material-ui/getting-started/overview/
- React Hook Form: https://react-hook-form.com/
- Formik Docs: https://formik.org/docs/overview
- Nodemailer: https://nodemailer.com/about/
- Nodemailer Tutorial: https://www.digitalocean.com/community/tutorials/nodejs-send-email
- SendGrid Docs: https://docs.sendgrid.com/
- Axios Docs: https://axios-http.com/docs/intro
- Fetch API Guide: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
- REST API Best Practices: https://restfulapi.net/
- File Upload MERN Tutorial: https://www.geeksforgeeks.org/file-uploading-in-react-node-js-and-mongodb/
- Socket.io Docs: https://socket.io/docs/v4/
- Render Deployment Docs: https://render.com/docs/deploy-node-express-app
- Vercel Docs: https://vercel.com/docs
- Netlify Docs: https://docs.netlify.com/
- FreeCodeCamp MERN Tutorial: https://www.freecodecamp.org/news/mern-stack-tutorial/
- Traversy Media MERN Crash Course: https://www.youtube.com/watch?v=-0exw-9YJBo
- JavaScript Info: https://javascript.info/

Plagiarism

183-35-376

ORIGINALITY REPORT

14%	4%	1%	11%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Daffodil International University Student Paper	8%
2	Submitted to UNITEC Institute of Technology Student Paper	1%
3	dokumen.pub Internet Source	<1%
4	Phenikaa University Publication	<1%
5	github.com Internet Source	<1%
6	intranet.cs.man.ac.uk Internet Source	<1%
7	123dok.com Internet Source	<1%
8	Submitted to NCC Education Student Paper	<1%
9	Submitted to The Robert Gordon University Student Paper	<1%
10	etd.aau.edu.et Internet Source	<1%
11	Submitted to Manipal University Jaipur Online Student Paper	<1%

12	Submitted to The Open University Student Paper	<1%
13	Submitted to University of Greenwich Student Paper	<1%
14	Submitted to University of Wales Institute, Cardiff Student Paper	<1%
15	files.eric.ed.gov Internet Source	<1%
16	sakshiz9.github.io Internet Source	<1%
17	www.everydaywinningtips.com.ng Internet Source	<1%
18	mewe.com Internet Source	<1%
19	in.jooble.org Internet Source	<1%
20	www.coursehero.com Internet Source	<1%
21	Aparna Datta, Sanchayan Sen. "chapter 9 A Security and Privacy Validation Methodology for E-Health Systems Using Cloud Storage", IGI Global, 2024 Publication	<1%
22	Ton Duc Thang University Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

Library Clearance

Account Clearance