Chapter 1

Introduction

1.1 Background

In the ever-accelerating realm of Information Technology (IT), the relentless march of technological progress has brought forth an era of unprecedented transformation. The 21st century has witnessed the convergence of data, connectivity, and innovation, rewriting the rules of business, education, communication, and daily life. With each dawn, a new paradigm emerges, underpinned by advancements like artificial intelligence, blockchain, cloud computing, and the Internet of Things.

Yet, as the IT landscape morphs into uncharted territory, a pressing challenge looms large: the challenge of keeping pace with the relentless torrent of innovation. For IT professionals, students, and enthusiasts, staying informed has never been more critical, nor has the need for meaningful connections within the global IT community been more pronounced.

The "Pioneering Hub for Information and Networking in the Latest IT Innovations" is born out of this recognition—a recognition of the indispensable role that knowledge and collaboration play in navigating the ever-shifting sands of IT. It stands as a response to the voracious appetite for information, the thirst for networking, and the yearning for meaningful engagement within a world redefined by technology.

1.2 Objectives

At the core of our project lies a dual set of objectives, underpinning our commitment to advancing the cause of IT innovation and community:

1. Knowledge Dissemination: Our primary aim is to construct a dynamic and accessible digital platform that acts as a conduit for the latest IT advancements and insights. We aspire to be the trusted wellspring from which IT professionals, students, and enthusiasts draw their insights, enabling them to navigate the tumultuous waters of rapid technological transformation.
2. Community Cultivation: Our vision extends beyond the mere sharing of information. We are steadfast in our pursuit of fostering a thriving community—a global congregation of like-minded individuals united by their passion for IT. We aim to create an environment where knowledge sharing and collaborative endeavours flourish, where professional networks are nurtured, and where the collective genius of the IT sector finds its voice.

1.3 Purpose, scope, and applicability

1.3.1 Purpose

The purpose that animates our project is anchored in a profound understanding of the symbiotic relationship between IT innovation and those who seek to harness its power. We acknowledge the transformative potential of IT on industries, economies, and societies. We recognize that the IT sector is more than an industry; it is a catalyst for change, a beacon of innovation, and a conduit for progress.

The purpose that drives us is to equip IT enthusiasts, professionals, and experts with the tools, resources, and opportunities they need to thrive within this ever-evolving landscape. It is to empower individuals with the knowledge that fuels innovation, to facilitate connections that transcend borders, and to facilitate the collective growth of the IT sector.

Our purpose extends beyond the digital realm. We aspire to create real-world impact—to be a catalyst for the development of groundbreaking technologies, the launchpad for innovative startups, and the source of solutions to the world's most pressing challenges. We are driven by a profound belief in the power of community-driven knowledge, collaboration, and innovation.

1.3.2 Scope

The scope of our project is as vast as the horizons of IT itself. It encompasses the development of a sophisticated, multifaceted web platform and a suite of companion mobile applications that collectively serve as the beating heart of the IT community. Within this digital ecosystem, a myriad of features coalesces to create a vibrant, interconnected space where innovation thrives, connections are forged, and knowledge is shared.

Our platform will provide real-time updates on IT innovations, deliver insights from the luminaries of the industry, host interactive and enlightening discussion forums, orchestrate informative webinars and workshops, and facilitate networking opportunities that transcend geographical boundaries. Additionally, we are pioneering the integration of an intelligent chatbot for swift user support and a live open chat feature that nurtures spontaneous and enriching interactions. The project's reach extends to IT professionals, students, and tech enthusiasts worldwide, uniting them under a common banner of progress.

1.3.3 Applicability

The applicability of our platform is comprehensive, designed to cater to the diverse needs and aspirations of a multifaceted audience within the IT sector. We have meticulously engineered our digital domain to effortlessly accommodate the requirements of professionals, students, and enthusiasts, each of whom has their unique journey within the world of IT.

For professionals who seek to stay at the vanguard of industry trends, our platform is a source of timely and relevant information, a wellspring of expert insights, and a hub for networking with peers, mentors, and potential collaborators. It serves as an indispensable tool for career advancement, a space for skill development, and a portal to explore innovative opportunities within the IT sector.

For students embarking on their journey toward careers in IT, our platform is a trusted guide—a repository of knowledge that supplements their formal education, a platform for connecting with industry professionals, and a launchpad for internships and job opportunities. It is a source of inspiration, mentorship, and practical insights that help bridge the gap between academia and industry.

For passionate tech enthusiasts whose curiosity knows no bounds, our platform is a treasure trove of discoveries—a gateway to the latest innovations, a space for engaging with like-minded individuals, and a canvas for sharing their own insights, projects, and experiments. It offers a vibrant community where they can explore their interests, cultivate their skills, and embark on journeys of lifelong learning.

In essence, our platform is a versatile and all-encompassing resource that serves as a lighthouse guiding the ships of IT professionals, students, and enthusiasts through the uncharted waters of technological innovation.

1.4 Achievements

As we embark on this groundbreaking endeavour, we anticipate an array of transformative achievements that will underscore the resounding impact of our platform. Foremost among these is the realization of a flourishing IT community—a global congregation of like-minded individuals unified by their shared curiosity and passion for innovation. This community will be characterized by an ethos of unfettered knowledge sharing, collaborative synergy, and a commitment to perpetually exploring the frontiers of IT.

Our platform's ability to foster meaningful connections between IT professionals, students, and enthusiasts will be an enduring hallmark. By providing a dedicated and welcoming space for users to engage, interact, and forge bonds, we envision a tapestry of relationships that transcend the digital realm, spurring opportunities for mentorship, collaboration, and even entrepreneurship.

Moreover, we anticipate a significant upsurge in the collective knowledge repository of the IT sector, attributable to the proliferation of expert insights, user-generated content, and collaborative projects hosted on our platform. This veritable treasure trove of wisdom will stand as a testament to the power of community-driven knowledge creation.

Furthermore, our platform's potential to expedite career growth and professional development within the IT sector cannot be overstated. By seamlessly connecting job seekers with employers, fostering mentorship relationships, and offering a plethora of career resources, we aspire to become a catalyst for individual and collective progress.

The cumulative achievements of our project will culminate in a palpable contribution to the growth and development of the IT sector. The solutions, ideas, and innovations birthed within our community will have far-reaching implications, potentially reshaping industries, solving complex problems, and powering technological advancements that have yet to be conceived.

1.5 Organisations of reports

This expansive report is structured to provide a detailed and comprehensive exploration of the multifaceted journey undertaken by the "Pioneering Hub for Information and Networking in the Latest IT Innovations" project. Within these pages, you will traverse the intricate labyrinth of project planning, design and development, feature integration, user engagement strategies, security considerations, and a visionary roadmap for future enhancements. Each section represents a meticulously crafted facet of our unwavering commitment to pioneering excellence within the IT sector.

1.5 Summary

In summary, our project, the "Pioneering Hub for Information and Networking in the Latest IT Innovations," seeks to transcend the boundaries of a conventional web platform. It endeavours to become a transformative force—a beacon illuminating the path to uncharted IT innovations, a haven for vibrant networking and collaboration, and a cornerstone of knowledge dissemination within the global IT community. As you delve deeper into the following pages, we invite you to embark on this journey with us, to explore our vision, and to partake in the boundless possibilities of IT innovation and community.

Chapter 2

Survey of Technology

2.1 Why MERN Stack?

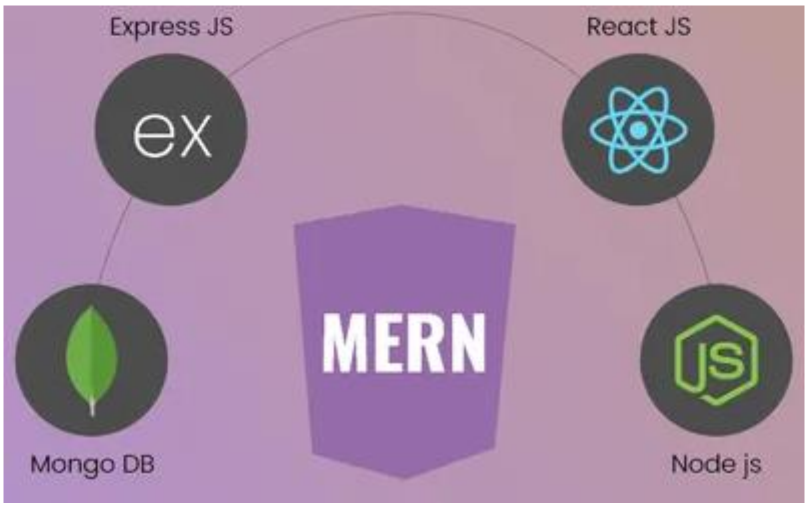


Figure 2.1: MERN

The choice of the MERN (MongoDB, Express.js, React.js, Node.js) stack for PHINLIN's development represents a strategic decision rooted in the stack's capabilities and synergy. The MERN stack offers a full-fledged JavaScript-based development environment, ensuring a consistent and streamlined workflow throughout the project's lifecycle.

MongoDB, a NoSQL database, is particularly well-suited for our project's requirements. Its flexible schema, scalability, and ability to handle large volumes of unstructured data make it an ideal choice. MongoDB's compatibility with JSON data and dynamic queries facilitates the storage and retrieval of diverse content on the platform.

Express.js, a minimalistic Node.js web application framework, is an integral part of our backend development. It provides a robust and efficient way to handle HTTP requests, enabling rapid development of RESTful APIs and ensuring the seamless flow of data between the server and client.

Node.js, known for its non-blocking, event-driven architecture, serves as the foundation of our backend. Its exceptional performance, scalability, and extensive library support make it the preferred choice for real-time applications and data-intensive tasks.

React.js, a JavaScript library for building user interfaces, powers our front-end development. Its component-based architecture, virtual DOM, and declarative approach simplify the creation of responsive and interactive user interfaces. React's extensive ecosystem of libraries and a vibrant community ensure the availability of resources and support for PHINLIN's ongoing development.

2.2 Database

2.2.1 Why MongoDB?

Figure 2.2: MongoDB

MongoDB emerges as the preferred database solution for PHINLIN due to its exceptional suitability for handling diverse and evolving data. MongoDB's NoSQL architecture, based on collections and documents, provides the flexibility needed to accommodate the diverse types of content, user-generated data, and real-time updates that our platform demands.

The schema-less nature of MongoDB is particularly advantageous, as it allows us to adapt and evolve our data structures without the constraints of predefined schemas. This flexibility is vital in an environment where data types, structures, and relationships can evolve rapidly.

Scalability is another key factor driving our choice of MongoDB. With its support for horizontal scaling through sharding, MongoDB can effortlessly handle the anticipated growth in data and users on PHINLIN. This scalability ensures that our platform remains responsive and efficient even as the user base expands.

Furthermore, MongoDB's robust querying capabilities, support for geospatial data, and powerful aggregation framework equip us with the tools needed to deliver features like real-time chat, user-generated content, and personalized recommendations. Its built-in replication and high availability features guarantee data durability and accessibility, essential for the uninterrupted operation of our platform.

2.2 Server Side

2.3.1 Why Node Js?

Figure 2.4: Node js

Node.js, renowned for its non-blocking, event-driven architecture, emerges as the backbone of PHINLIN's backend development. Its selection is rooted in its exceptional performance, scalability, and the inherent advantages it brings to real-time and data-intensive applications.

Node.js excels in handling concurrent connections and asynchronous tasks, making it ideal for the real-time features that PHINLIN incorporates, such as chat functionality and instant data updates. This event-driven nature ensures that our platform remains responsive and can seamlessly accommodate numerous users concurrently.

Scalability is a critical consideration for PHINLIN, given our projection for rapid user growth. Node.js's horizontal scalability through load balancing and clustering enables us to distribute incoming requests efficiently and accommodate increased traffic without sacrificing performance.

Moreover, Node.js boasts a vast and active open-source ecosystem, with a wealth of libraries and modules that accelerate development and simplify the creation of RESTful APIs, one of the core components of our backend. This extensive library support, combined with a thriving community, ensures that we have access to the tools and resources needed for ongoing development and maintenance.

2.3.2 Why Node js Express?

Figure 2.5: Express js

Node.js Express serves as the foundational framework for PHINLIN's backend development, playing a pivotal role in routing and handling HTTP requests. The choice of Express.js is guided by its minimalist and unopinionated design, which empowers developers to design and structure applications according to project-specific requirements.

Express.js excels in its ability to create RESTful APIs, a core component of our backend architecture. Its straightforward routing mechanisms and middleware support simplify the creation of routes, handling of HTTP requests, and implementation of authentication and authorization mechanisms, ensuring efficient data flow between the server and client.

Scalability is a critical consideration for PHINLIN, and Express.js's lightweight design contributes to efficient resource utilization, enabling the platform to scale gracefully as user traffic increases. Its flexibility allows us to integrate additional modules and libraries seamlessly to address evolving requirements.

Moreover, Express.js is renowned for its active and vibrant community, ensuring access to a wealth of third-party middleware and plugins. This extensibility enhances our development efficiency and enables us to leverage pre-built solutions for common backend functionalities.

2.3 Clint Side

2.4.1 Why React Js?

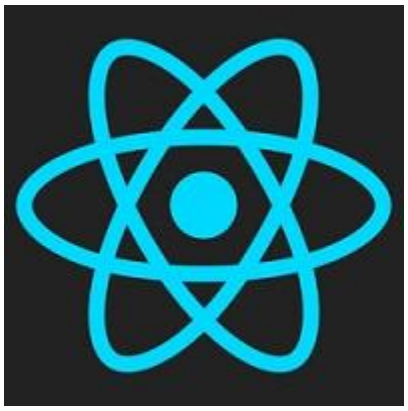


Figure 2.6: React js

React.js stands at the forefront of PHINLIN's front-end development, driving the creation of responsive and interactive user interfaces. The selection of React.js is underpinned by its exceptional capabilities in building dynamic, component-based user interfaces, its extensive community support, and its alignment with modern web development practices.

React's component-based architecture is a game-changer in crafting user interfaces. It allows us to break down complex UIs into reusable and manageable components, enhancing maintainability and code reusability. This modularity enables our development team to work efficiently, as components can be developed in isolation and seamlessly integrated into the overall application structure.

Furthermore, React's virtual DOM (Document Object Model) efficiently updates only the parts of the user interface that have changed, minimizing page reloads and ensuring a lightning-fast user experience. This performance optimization is critical for PHINLIN's real-time features and ensures smooth interactions for users, whether they're engaging in discussions, attending webinars, or participating in open chat sessions.

React's extensive ecosystem of libraries and tools, including React Router for routing and Redux for state management, provides a comprehensive toolkit for addressing various frontend development challenges. This ecosystem streamlines development, accelerates feature implementation, and ensures that our platform remains maintainable and scalable as it grows.

Additionally, React boasts a vibrant and active community that continuously contributes to its improvement and offers a wealth of resources, tutorials, and support. This ensures that our development team has access to the latest best practices and solutions, accelerating our development cycle and enabling us to deliver a cutting-edge user experience.

Chapter 3

Requirements and Analysis

3.1 Problem Definition

The "Pioneering Hub for Information and Networking in the Latest IT Innovations (PHINLIN)" aims to address several challenges within the IT sector, including:

Information Overload: Streamlining the influx of information by offering real-time updates on IT innovations and trends, ensuring users stay informed without feeling overwhelmed.

Fragmented Networking: Creating a unified platform to connect IT professionals, students, and enthusiasts globally, fostering meaningful connections and collaboration opportunities.

Limited Knowledge Sharing: Facilitating interactive discussion forums, webinars, and workshops to encourage knowledge sharing, problem-solving, and collaborative learning within the IT community.

Career Advancement: Providing a space for professionals and students to explore career opportunities, mentorship, and industry insights, addressing challenges related to career growth within the IT sector.

Lack of Personalization: Offering personalized dashboards and content recommendations to tailor the user experience, ensuring that individuals receive relevant and customized information based on their preferences.

Absence of Real-Time Interaction: Introducing features like an integrated chatbot and live open chat to enable instant assistance, answer queries, and foster real-time interactions, enhancing user engagement and support.

Sparse User-Generated Content: Encouraging the contribution and publication of user-generated tech-related content, addressing the need for diverse and community-driven content within the IT space.

In essence, PHINLIN endeavours to streamline information consumption, enhance networking opportunities, promote knowledge sharing, and address various challenges faced by individuals and professionals in navigating the ever-evolving IT landscape.

3.2 Requirements Specification

3.2.1 System Requirements :

Server-Side Requirements:

Web Server:

Use a web server capable of running Node.js applications (e.g., Nginx, Apache).

- **Node.js:**

  Minimum version: 14.x

  Utilize the latest stable LTS version for production deployments.

- **Express.js:**

  Incorporate the Express.js framework for building the server-side application.

  Version compatibility with Node.js version used.

  Database:

  MongoDB is the preferred database.

  Ensure compatibility with the selected Node.js and Express.js versions.

  Establish a scalable and well-optimized database architecture.

- **Security:**

  Implement secure communication using HTTPS.

Employ security best practices for Node.js and Express.js applications.

  Ensure proper authentication and authorization mechanisms.

- **Scalability:**

Design the architecture with scalability in mind, considering potential increases in user base and data volume.

  Utilize load balancing and horizontal scaling techniques.

- **Monitoring and Logging:**

  Implement robust monitoring and logging solutions (e.g., Stack driver for GCP).

Monitor application performance, server health, and error logs.

Client-Side Requirements:

- **Web Browser:**

  Support major modern browsers, including Chrome, Firefox, Safari, and Edge.

  Ensure compatibility with the latest browser versions.

- **Frontend Framework:**

  React.js is the chosen frontend library.

  Utilize the latest stable version for development.

- **Responsive Design**:

  Implement a responsive design to ensure optimal user experience across various devices and screen sizes.

- **User Interface (UI):**

  Develop an intuitive and user-friendly UI for easy navigation and engagement.

  Ensure accessibility standards are met.

Other Considerations:

- **User Authentication:**

  Implement secure user authentication mechanisms.

  Integrate OAuth or other secure authentication protocols.

- **User Data and Privacy:**

  Comply with data protection regulations (e.g., GDPR) and prioritize user data privacy.

  Implement secure data storage and transmission practices.

- **Content Delivery:**

  Utilize Content Delivery Networks (CDNs) for efficient content delivery.

  Optimize media and static file delivery for performance.

- **Backup and Recovery:**

  Implement regular backup procedures for databases and critical data.

  Establish a disaster recovery plan to minimize downtime in case of failures.

- **Testing:**

  Conduct thorough testing, including unit testing, integration testing, and end-to-end testing. Implement automated testing processes.

- **Documentation:**

  Maintain comprehensive documentation for the system architecture, APIs, and deployment procedures. Include user guides and developer documentation.

3.3 Planning and Scheduling

3.3.1 GANTT CHART

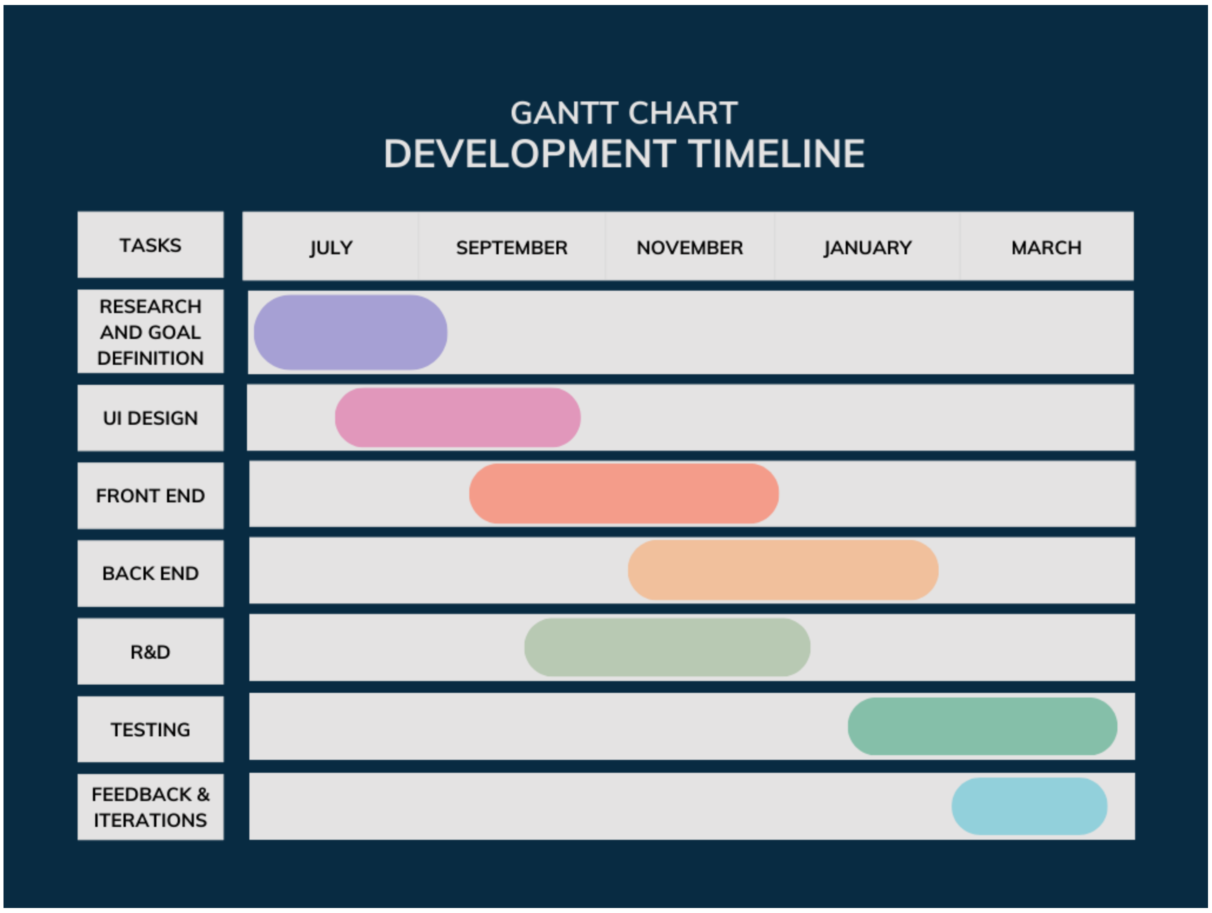


Figure 3.1: Gant Chart

3.4 Software and Hardware Requirements

Hardware Requirement:

o Considerable disk space for storing website data.

o Suggested RAM capacity for the server: 16GB or higher.

o Reliable network infrastructure with sufficient bandwidth.

o High-performance servers for hosting Coupons99 (for future enhancements).

· **Software Requirements:**

o Server operating system compatible with web hosting and database management (e.g., Linux, Windows Server).

o PostgreSQL 42.6.0 for database management.

o MongoDB for specific data storage.

o Google Cloud as the web hosting platform.

o HTML, CSS, and JavaScript (JS) for frontend development.

o Backend development tools, frameworks, and libraries.

o Optional user interface frameworks for enhanced design and interactivity if required, (e.g., React, Angular).

o Security tools, including firewalls and encryption protocols.

o Version control system (e.g., Git).

o Integrated Development Environment (IDE) and testing tools.

o Optional communication and collaboration tools (e.g., Slack, Microsoft Teams).

o Specific libraries or frameworks for backend development.

o Optional cloud-based services for scalability, storage, and infrastructure.

o Web browsers for development and testing.

o Privacy and security compliance measures.

o Implementation of multilingual support.

o Payment gateway integration (for future enhancements).

3.5 Preliminary Product Description

Real-Time Knowledge Feed:

Stay informed with a dynamic news feed providing real-time updates on the latest IT innovations, trends, and industry insights.

Comprehensive User Profiles:

Create detailed user profiles showcasing skills, expertise, and professional achievements, facilitating meaningful connections within the IT community.

Interactive Discussion Forums:

Engage in vibrant discussion forums, encouraging knowledge sharing, problem-solving, and collaborative learning among IT enthusiasts.

Webinars and Workshops:

Participate in and host interactive webinars and workshops featuring IT thought leaders, fostering a culture of continuous learning and skill development.

Networking Tools:

Utilize robust networking tools to connect with peers, mentors, and potential collaborators, expanding professional circles and creating valuable industry connections.

User-Generated Content:

Contribute and publish tech-related content, including articles, tutorials, and case studies, to share expertise and insights with the global IT community.

Personalized Dashboards:

Enjoy a personalized user experience with customized dashboards, providing tailored content recommendations based on user preferences and interests.

Integrated Chatbot:

Access instant assistance, get answers to queries, and receive guided support through an intelligent chatbot, enhancing user engagement and experience.

Live Open Chat:

Engage in real-time, spontaneous interactions and discussions with peers through live open chat sessions, fostering a sense of community and camaraderie.

Goal:

PHINLIN aims to revolutionize the IT landscape by providing a centralized hub where professionals and enthusiasts can seamlessly access the latest information, connect with like-minded individuals, and collaboratively contribute to the advancements in technology.

Target Audience:

IT professionals, students, researchers, and tech enthusiasts seeking a comprehensive platform for staying informed, networking, and actively participating in the exchange of knowledge within the IT community.

3.6 Conceptual Model

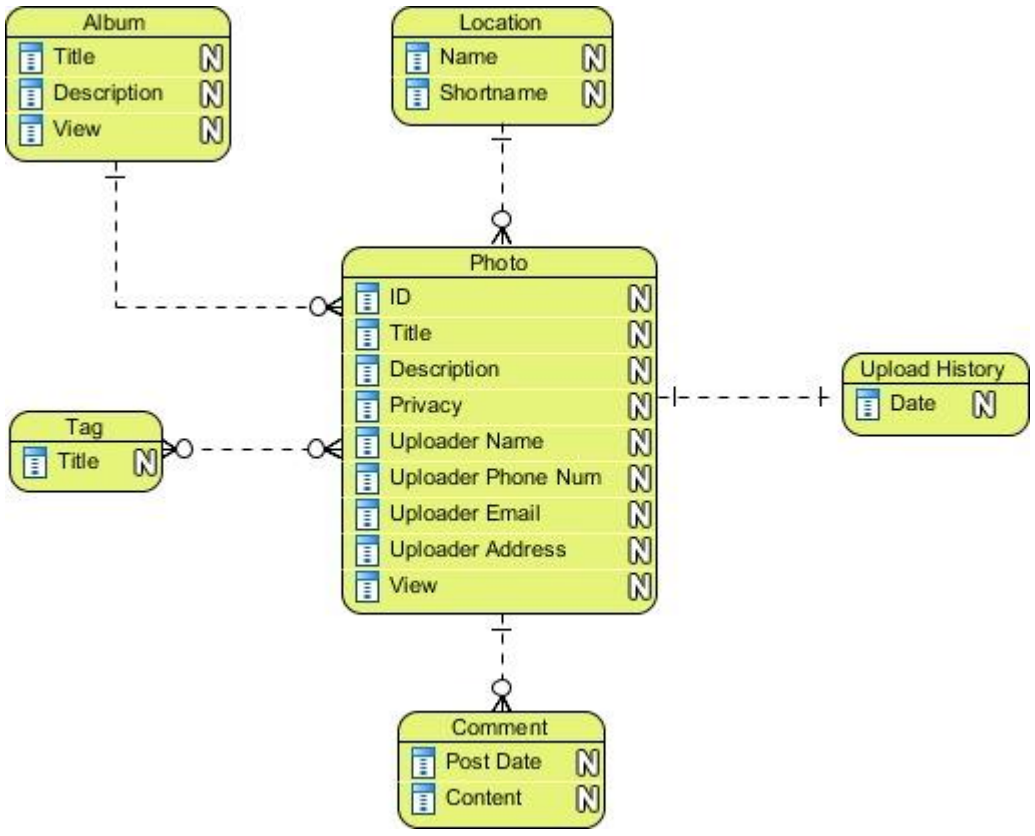


Figure 3.2: Conceptual Model

Chapter 4

System Design

4.1 Module Division

User Management Module:

Responsible for user authentication, registration, and profile management.

Includes functionalities like login, logout, user registration, profile editing, and password recovery.

Content Management Module:

Manages the creation, editing, and deletion of various content types.

Sub-modules for managing articles, forum posts, webinars, and user-generated content.

Communication Module:

Facilitates real-time communication among users.

Includes features such as chat messaging, notifications, and alerts.

Networking Module:

Handles functionalities related to networking and user connections.

Manages friend requests, followers, and other networking features.

Knowledge Sharing Module:

Focuses on features that promote knowledge sharing within the community.

Includes forums, discussion boards, and collaborative spaces.

Webinar Module:

Manages the scheduling, hosting, and participation in webinars.

Includes features for webinar creation, scheduling, and attendee management.

Analytics and Reporting Module:

Provides insights into user activities, content engagement, and platform usage.

Generates reports and analytics to help administrators understand platform performance.

Search and Recommendation Module:

Implements search functionalities for users to find relevant content and connections.

Includes recommendation algorithms for personalized content suggestions.

Security Module:

Ensures the overall security of the platform.

Implements measures such as encryption, secure authentication, and protection against common security threats.

Integration Module:

Manages integrations with external services or APIs. Includes integrations with external databases, authentication providers, and third-party services.

Admin Panel Module:

Provides administrators with tools for managing and moderating the platform.

Includes user management, content moderation, and analytics dashboard.

Settings and Preferences Module:

Allows users to customize their experience on the platform.

Manages user settings, preferences, and notification configurations.

Responsive UI/UX Module:

Ensures a seamless and user-friendly experience across various devices.

Focuses on responsive design and optimal user interface interactions.

Testing and Quality Assurance Module:

Includes unit testing, integration testing, and quality assurance processes.

Ensures the reliability and stability of the entire platform.

Documentation and Help Module:

Provides user guides, FAQs, and help documentation.

Ensures users and developers have access to necessary information.

Deployment and Infrastructure Module:

Manages the deployment process and infrastructure requirements.

Includes server setup, cloud platform configuration, and continuous integration.

4.2 Data Dictionary

A Data Dictionary for the "Pioneering Hub for Information and Networking in the Latest IT Innovations (PHINLIN)" project outlines the details of the data entities, their attributes, and their relationships. Below is a simplified example to illustrate the structure of a Data Dictionary. Actual details will depend on the specific requirements and design of your project.

Entities: -

User:

Attributes:

UserID (Primary Key)

Username

Email

Password

UserType (e.g., Admin, Moderator, Regular User)

JoinDate

LastLogin

Article:

Attributes:

ArticleID (Primary Key)

Title

Content

AuthorID (Foreign Key referencing User)

PublicationDate

Likes

ForumPost:

Attributes:

PostID (Primary Key)

Content

AuthorID (Foreign Key referencing User)

PostedDate

Likes

Replies

Webinar:

Attributes:

WebinarID (Primary Key)

Title

Description

HostID (Foreign Key referencing User)

Schedule

Attendees

Chat:

Attributes:

ChatID (Primary Key)

SenderID (Foreign Key referencing User)

ReceiverID (Foreign Key referencing User)

Message

Timestamp

Relationships: -

User - Article (One-to-Many):

Foreign Key:

AuthorID in Article

User - ForumPost (One-to-Many):

Foreign Key:

AuthorID in ForumPost

User - Webinar (One-to-Many):

Foreign Key:

HostID in Webinar

User - Chat (One-to-Many):

Foreign Keys:

SenderID and ReceiverID in Chat

Article - Webinar (Many-to-Many):

Association Entity:

ArticleWebinar (ArticleWebinarID, ArticleID, WebinarID)

Data Types:

UserID: Integer

Username: String

Email: String

Password: String (Hashed)

UserType: Enum (Admin, Moderator, Regular User)

JoinDate: Date

LastLogin: Date

ArticleID: Integer

Title: String

Content: Text

AuthorID: Integer (Foreign Key)

PublicationDate: Date

Likes: Integer

PostID: Integer

PostedDate: Date

Replies: Integer

WebinarID: Integer

Description: Text

HostID: Integer (Foreign Key)

Schedule: DateTime

Attendees: Integer

ChatID: Integer

SenderID: Integer (Foreign Key)

ReceiverID: Integer (Foreign Key)

Message: Text

Timestamp: DateTime

4.3 ER Diagram

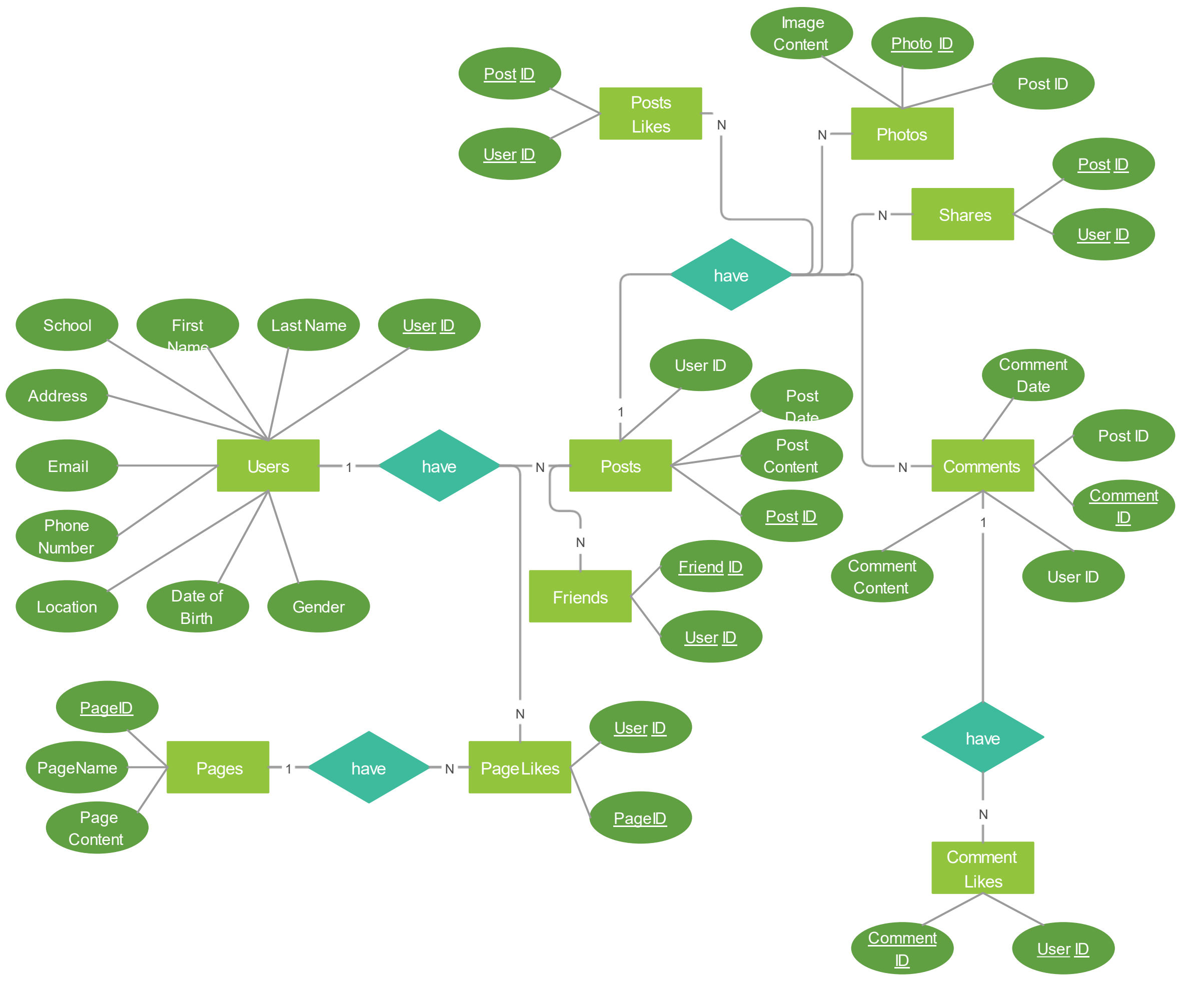


Figure 4.1: ER Digram

4.4 Relational Diagram

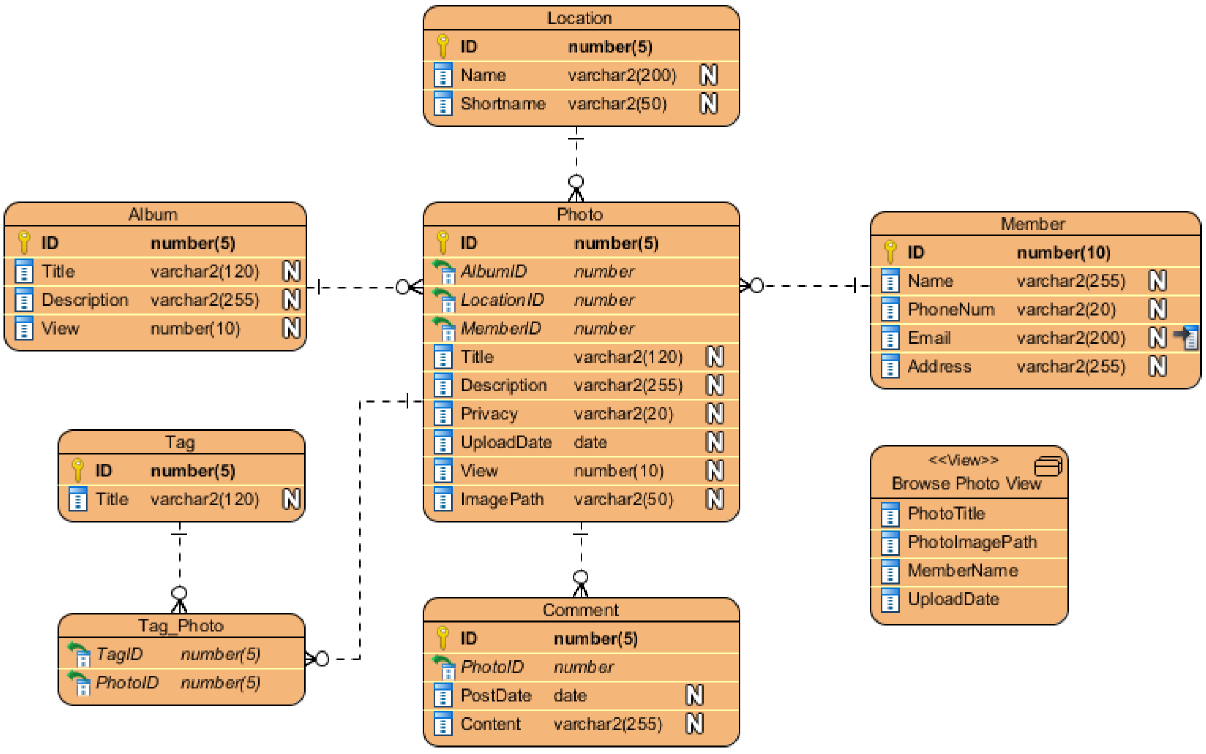


Figure 4.2: Relational Diagram

4.5 Use case

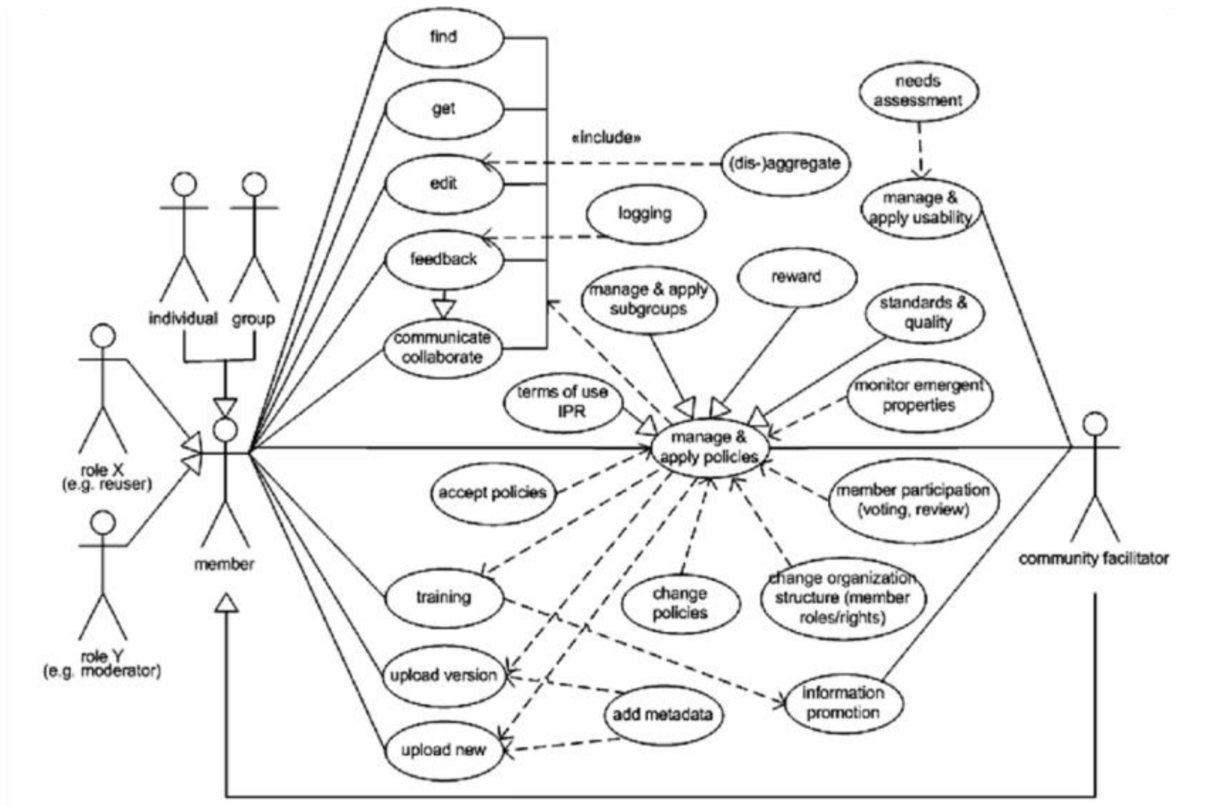
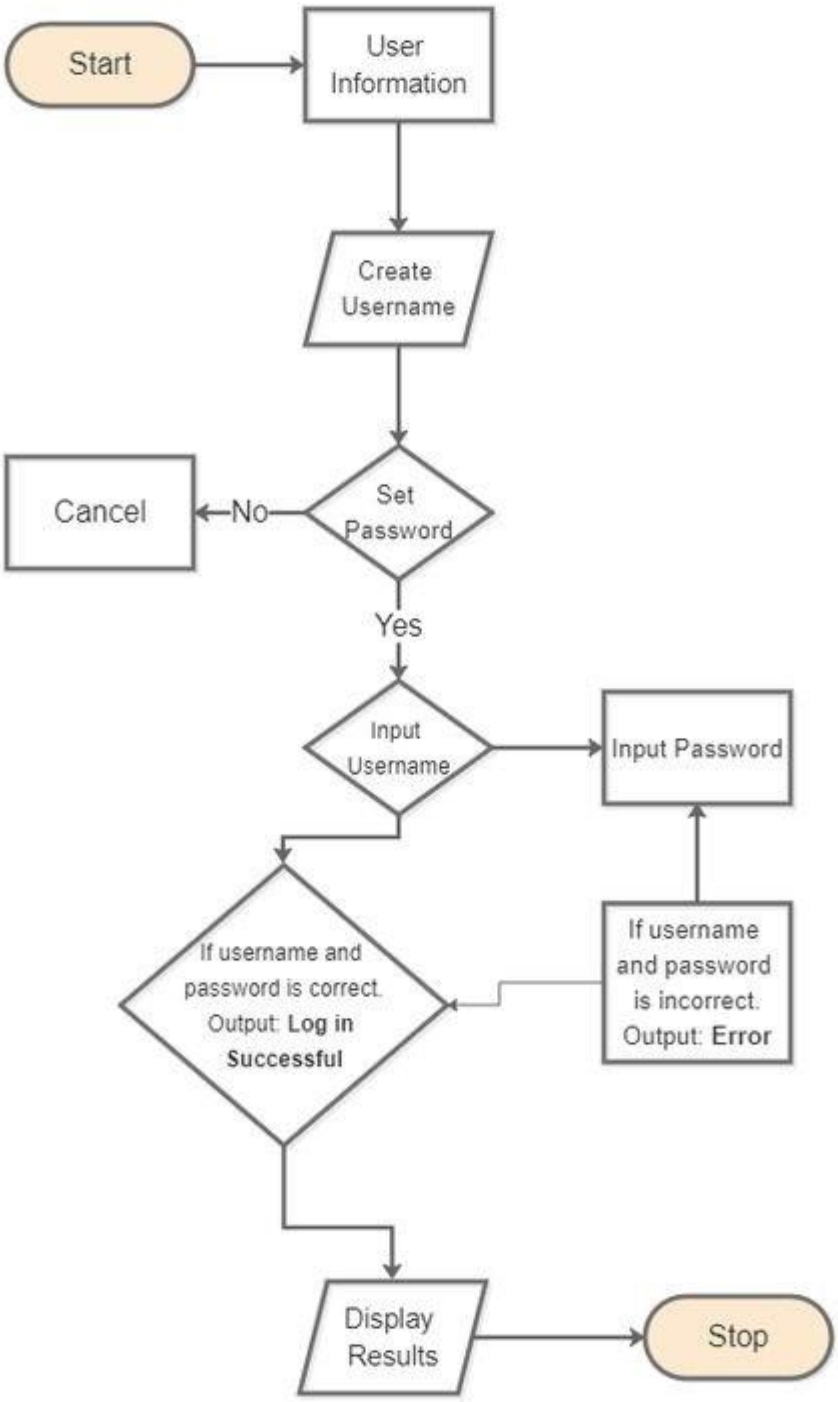


Figure 4.3: Use case

4.6 Procedural Design

4.6.1 Login



Start
User Information
Create Username
Set Password
No
Cancel
Yes
Input Username
Input Password
If username and password is correct. Output: Log in Successful
If username and password is incorrect. Output: Error
Display Results
Stop

Figure 4.4: Login

4.7 Data Structure

User:

UserID (Primary Key)

Username

Email

Password (Hashed)

UserType (Admin, Moderator, Regular User)

JoinDate

LastLogin

Content:

ContentID (Primary Key)

Title

ContentType (Article, ForumPost, Webinar, ChatMessage, etc.)

AuthorID (Foreign Key referencing User)

PublicationDate

Likes

Replies

ContentDetails (Specific attributes based on content type)

Networking:

ConnectionID (Primary Key)

SenderID (Foreign Key referencing User)

ReceiverID (Foreign Key referencing User)

Status (Pending, Accepted, Declined, etc.)

Knowledge Sharing:

ForumID (Primary Key)

ThreadID (Primary Key)

UserID (Foreign Key referencing User)

Content

Timestamp

Events:

EventID (Primary Key)

Title

Description

HostID (Foreign Key referencing User)

Schedule

Attendees (Array of UserIDs)

Chat:

ChatID (Primary Key)

SenderID (Foreign Key referencing User)

ReceiverID (Foreign Key referencing User)

Message

Timestamp

Relationships:

User - Content (One-to-Many):

A user can create multiple pieces of content, but each piece of content is created by one user.

User - Networking (Many-to-Many):

Users can have multiple connections, and each connection involves multiple users.

User - Knowledge Sharing (One-to-Many):

A user can participate in multiple knowledge-sharing activities, but each activity is associated with one user.

User - Events (One-to-Many):

A user can host multiple events, but each event has one host (user).

User - Chat (One-to-Many):

A user can send multiple chat messages, but each message has one sender (user).

Content - Events (Many-to-Many):

Content (e.g., articles) can be associated with multiple events (e.g., webinars), and an event can be associated with multiple pieces of content.

Data Storage Considerations:

User Data:

Store user data securely with hashed passwords.

Implement user roles and permissions for different user types.

Content Storage:

Use appropriate storage for various content types (e.g., articles, forum posts, chat messages).

Consider a flexible schema to accommodate different content structures.

Networking Data:

Store connection data with information about the relationship status (pending, accepted, declined).

Knowledge Sharing Data:

Forum threads and posts should be stored with timestamps for tracking discussions.

Events Data:

Store event details including the host, schedule, and attendees.

Chat Data:

Store chat messages with sender, receiver, message content, and timestamps.

Data Access Patterns:

User Profiles:

Retrieve user profiles with relevant information for networking and community engagement.

Content Retrieval:

Fetch content based on content type, author, and popularity (likes, replies).

Networking Queries:

Retrieve connections, friend lists, and pending connection requests for users.

Knowledge Sharing Interactions:

Fetch forum threads, posts, and user contributions for knowledge sharing.

Event Management:

Retrieve event details, host information, and participant lists.

Real-Time Chat:

Implement real-time data retrieval for chat messages between users.

Storage Technology:

Database:

Consider a NoSQL database (e.g., MongoDB) for flexibility in handling diverse data structures.

Use appropriate indexing for efficient data retrieval.

Authentication:

Implement secure authentication mechanisms and use JWT (JSON Web Tokens) for user sessions.

Security Measures:

Employ encryption for sensitive data.

Regularly update and patch the database and application for security.

This is a high-level representation, and the actual data structure may evolve based on specific requirements and the chosen technology stack for PHINLIN. Always consider scalability, security, and performance in the design and implementation of the data structure.

4.8 Security Issue

Security is a critical aspect of any web application, and addressing potential security issues is essential to protect user data, maintain system integrity, and prevent unauthorized access. Here are some common security issues and considerations that you should be aware of and address in the context of "Pioneering Hub for Information and Networking in the Latest IT Innovations (PHINLIN)":

Data Encryption:

Issue: Storing sensitive data, such as passwords, in plain text.

Solution: Use strong encryption algorithms (e.g., bcrypt) to hash and salt passwords before storing them.

Authentication and Authorization:

Issue: Weak or flawed authentication mechanisms.

Solution: Implement secure user authentication, use multi-factor authentication where possible, and enforce proper authorization controls to ensure users have appropriate access rights.

Cross-Site Scripting (XSS):

Issue: Allowing the injection of malicious scripts into web pages viewed by other users.

Solution: Implement input validation, sanitize user inputs, and use Content Security Policy (CSP) headers to mitigate XSS attacks.

Cross-Site Request Forgery (CSRF):

Issue: Allowing attackers to perform actions on behalf of authenticated users without their consent.

Solution: Use anti-CSRF tokens in forms and ensure that state-changing requests require proper authentication and authorization.

SQL Injection:

Issue: Allowing attackers to manipulate SQL queries through user inputs.

Solution: Use parameterized queries or prepared statements to prevent SQL injection attacks.

Insecure Direct Object References (IDOR):

Issue: Allowing unauthorized access to resources by manipulating object references.

Solution: Implement proper access controls and validate user permissions before allowing access to resources.

Session Management:

Issue: Insecure session management leading to session hijacking or fixation.

Solution: Use secure session handling practices, employ secure cookies, and regularly rotate session tokens.

Insecure File Uploads:

Issue: Allowing users to upload and execute malicious files.

Solution: Implement strict file type validation, limit file upload size, and store uploaded files in a secure location.

Security Headers:

Issue: Missing security headers, exposing the application to various attacks.

Solution: Implement security headers such as Content Security Policy (CSP), Strict-Transport-Security (HSTS), and X-Content-Type-Options.

Logging and Monitoring:

Issue: Insufficient logging and monitoring, making it difficult to detect and respond to security incidents.

Solution: Implement comprehensive logging, monitor system logs, and set up alerts for suspicious activities.

Dependency Security:

Issue: Using outdated or vulnerable third-party dependencies.

Solution: Regularly update dependencies, perform security audits, and use tools to check for known vulnerabilities in libraries.

Secure Communication:

Issue: Transmitting sensitive data over unsecured channels.

Solution: Use HTTPS to encrypt data in transit and ensure secure communication protocols.

Remember that security is an ongoing process, and it's essential to stay informed about the latest security best practices and vulnerabilities. Regularly conduct security audits, perform penetration testing, and address any identified issues promptly to ensure the continued security of PHINLIN.

Chapter 5

Implementation And Testing

5.1 Code

5.1.1 Client Side:

Index.js:

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";
import { AuthContextProvider } from "./context/AuthContext";
ReactDOM.render(
  <React.StrictMode>
    <AuthContextProvider>
      <App />
    </AuthContextProvider>
  </React.StrictMode>,
  document.getElementById("root")
);
```

App.js:

```
import Home from "./pages/home/Home";
import Login from "./pages/login/Login";
import Profile from "./pages/profile/Profile";
import Register from "./pages/register/Register";
import {
  BrowserRouter as Router,
  Switch,
  Route,
  Redirect,
} from "react-router-dom";
import { useContext } from "react";
import { AuthContext } from "./context/AuthContext";
import Messenger from "./pages/messenger/Messenger";
function App() {
  const { user } = useContext(AuthContext);
  return (
    <Router>
      <Switch>
        <Route exact path="/">
          {user ? <Home /> : <Register />}
        </Route>
        <Route path="/login">{user ? <Redirect to="/" /> : <Login />}</Route>
        <Route path="/register">
          {user ? <Redirect to="/" /> : <Register />}
        </Route>
        <Route path="/messenger">
          {!user ? <Redirect to="/" /> : <Messenger />}
        </Route>
        <Route path="/profile/:username">
          <Profile />
        </Route>
      </Switch>
    </Router>
  );
```

```
}
export default App;
```

apiCalls.js:

```
import axios from "axios";
export const loginCall = async (userCredential, dispatch) => {
  dispatch({ type: "LOGIN_START" });
  try {
    const res = await axios.post("/auth/login", userCredential);
    dispatch({ type: "LOGIN_SUCCESS", payload: res.data });
  } catch (err) {
    dispatch({ type: "LOGIN_FAILURE", payload: err });
  }
};
```

Home page:

home.jsx:

```
import Topbar from "../../components/topbar/Topbar";
import Sidebar from "../../components/sidebar/Sidebar";
import Feed from "../../components/feed/Feed";
import Rightbar from "../../components/rightbar/Rightbar";
import "./home.css"
export default function Home() {
  return (
    <>
      <Topbar />
      <div className="homeContainer">
        <Sidebar />
        <Feed/>
        <Rightbar/>
      </div>
    </>
  );
}
```

home.css:

```
.homeContainer{
    display: flex;
    width: 100%;
}
```

Login page:

Login.jsx:

```
import { useContext, useRef } from "react";
import "./login.css";
import { loginCall } from "../../apiCalls";
import { Link } from "react-router-dom";
import { AuthContext } from "../../context/AuthContext";
import { CircularProgress } from "@material-ui/core";
export default function Login() {
  const email = useRef();
  const password = useRef();
  const { isFetching, dispatch } = useContext(AuthContext);
  const handleClick = (e) => {
    e.preventDefault();
    loginCall(
      { email: email.current.value, password: password.current.value },
      dispatch);};
```

```
return (
  <div className="login">
    <div className="loginWrapper">
      <div className="loginLeft">
        <h3 className="loginLogo">Phinlin</h3>
        <span className="loginDesc">
          Connect with People's and the world around you on Phinlin.
        </span>
      </div>
      <div className="loginRight">
        <form className="loginBox" onSubmit={handleClick}>
          <input
            placeholder="Email"
            type="email"
            required
            className="loginInput"
            ref={email}/>
          <input
            placeholder="Password"
            type="password"
            required
            minLength="6"
            className="loginInput"
            ref={password}
          />
          <button className="loginButton" type="submit" disabled={isFetching}>
            {isFetching ? (
              <CircularProgress color="white" size="20px" />
            ) : (
              "Log In"
            )}
          </button>
          <span className="loginForgot">Forgot Password?</span>
          <Link to="/register" style={{ textDecoration: "none" }}>
          <button className="loginRegisterButton">
            {isFetching ? (
              <CircularProgress color="white" size="20px" />
            ) : (
              "Create a New Account"
            )}
          </button>
          </Link>
        </form>
      </div>
    </div>
  </div>
 );}
```

Login.css:

```
.login {
 width: 100vw;
 height: 100vh;
 background-color: #f0f2f5;
 display: flex;
 align-items: center;
```

```
  justify-content: center;
}
.loginWrapper {
  width: 70%;
  height: 70%;
  display: flex;
}
.loginLeft,
.loginRight {
  flex: 1;
  display: flex;
  flex-direction: column;
  justify-content: center;
}
.loginLogo {
  font-size: 50px;
  font-weight: 800;
  color: #1775ee;
  margin-bottom: 10px;
}
.loginDesc {
  font-size: 24px;
}
.loginBox{
    height: 300px;
    padding: 20px;
    background-color: white;
    border-radius: 10px;
    display: flex;
    flex-direction: column;
    justify-content: space-between;
}
.loginInput{
    height: 50px;
    border-radius: 10px;
    border: 1px solid gray;
    font-size: 18px;
    padding-left: 20px;
}
.loginInput:focus{
    outline: none;
}
.loginButton{
    height: 50px;
    border-radius: 10px;
    border: none;
    background-color: #1775ee;
    color: white;
    font-size: 20px;
    font-weight: 500;
    cursor: pointer;
}
.loginButton:focus{
  outline: none;
```

```
}
.loginButton:disabled{
  cursor: not-allowed;
}
.loginForgot{
    text-align: center;
    color: #1775ee;
}
.loginRegisterButton{
    width: 60%;
    align-self: center;
    height: 50px;
    border-radius: 10px;
    border: none;
    background-color: #42b72a;
    color: white;
    font-size: 20px;
    font-weight: 500;
    cursor: pointer;
}
```

Messenger page

Messenger.jsx

```
import "./messenger.css";
import Topbar from "../../components/topbar/Topbar";
import Conversation from "../../components/conversations/Conversation";
import Message from "../../components/message/Message";
import ChatOnline from "../../components/chatOnline/ChatOnline";
import { useContext, useEffect, useRef, useState } from "react";
import { AuthContext } from "../../context/AuthContext";
import axios from "axios";
import { io } from "socket.io-client";
export default function Messenger() {
  const [conversations, setConversations] = useState([]);
  const [currentChat, setCurrentChat] = useState(null);
  const [messages, setMessages] = useState([]);
  const [newMessage, setNewMessage] = useState("");
  const [arrivalMessage, setArrivalMessage] = useState(null);
  const [onlineUsers, setOnlineUsers] = useState([]);
  const socket = useRef();
  const { user } = useContext(AuthContext);
  const scrollRef = useRef();
  useEffect(() => {
    socket.current = io("ws://localhost:8900");
    socket.current.on("getMessage", (data) => {
      setArrivalMessage({
        sender: data.senderId,
        text: data.text,
        createdAt: Date.now(),
      });
    });
  }, []);
  useEffect(() => {
    arrivalMessage &&
      currentChat?.members.includes(arrivalMessage.sender) &&
      setMessages((prev) => [...prev, arrivalMessage]);
  }, [arrivalMessage, currentChat]);
  useEffect(() => {
    socket.current.emit("addUser", user._id);
    socket.current.on("getUsers", (users) => {
      setOnlineUsers(
        user.followings.filter((f) => users.some((u) => u.userId === f))
      );
    });
  }, [user]);
  useEffect(() => {
    const getConversations = async () => {
      try {
        const res = await axios.get("/conversations/" + user._id);
        setConversations(res.data);
      } catch (err) {
        console.log(err);
      }
    };
    getConversations();
  }, [user._id]);
  useEffect(() => {
    const getMessages = async () => {
      try {
        const res = await axios.get("/messages/" + currentChat?._id);
        setMessages(res.data);
      } catch (err) {
        console.log(err);
      }
    };
    getMessages();
  }, [currentChat]);
  const handleSubmit = async (e) => {
    e.preventDefault();
    const message = {
      sender: user._id,
      text: newMessage,
      conversationId: currentChat._id,
    };
    const receiverId = currentChat.members.find(
      (member) => member !== user._id
    );
    socket.current.emit("sendMessage", {
      senderId: user._id,
      receiverId,
      text: newMessage,
    });
    try {
      const res = await axios.post("/messages", message);
      setMessages([...messages, res.data]);
      setNewMessage("");
    } catch (err) {
      console.log(err);
    }
```

```
};
useEffect(() => {
  scrollRef.current?.scrollIntoView({ behavior: "smooth" });
}, [messages]);
return (
  <>
    <Topbar />
    <div className="messenger">
      <div className="chatMenu">
        <div className="chatMenuWrapper">
          <input placeholder="Search for friends" className="chatMenuInput" />
          {conversations.map((c) => (
            <div onClick={() => setCurrentChat(c)}>
              <Conversation conversation={c} currentUser={user} />
            </div>
          ))}
        </div>
      </div>
      <div className="chatBox">
        <div className="chatBoxWrapper">
          {currentChat ? (
            <>
              <div className="chatBoxTop">
                {messages.map((m) => (
                  <div ref={scrollRef}>
                    <Message message={m} own={m.sender === user._id} />
                  </div>
          ))}
        </div>
        <div className="chatBoxBottom">
          <textarea
            className="chatMessageInput"
            placeholder="write something..."
            onChange={(e) => setNewMessage(e.target.value)}
            value={newMessage}
          ></textarea>
          <button className="chatSubmitButton" onClick={handleSubmit}>
            Send
          </button>
        </div>
      </>
    ) : (
      <span className="noConversationText">
        Open a conversation to start a chat.
      </span>
    )}
  </div>
</div>
<div className="chatOnline">
  <div className="chatOnlineWrapper">
    <ChatOnline
      onlineUsers={onlineUsers}
      currentId={user._id}
      setCurrentChat={setCurrentChat}
```

```
          />
        </div>
      </div>
    </div>
  </>
 );
}
}
}
}
```

Messenger.css

```
.messenger {
  height: calc(100vh - 70px);
  display: flex;
}

.chatMenu {
  flex: 3.5;
}
.chatMenuInput {
  width: 90%;
  padding: 10px 0;
  border: none;
  border-bottom: 1px solid gray;
}
```

```
.chatBox {
  flex: 5.5;
}
.chatBoxWrapper {
  display: flex;
  flex-direction: column;
  justify-content: space-between;
  position: relative;
}
.chatBoxTop {
  height: 100%;
  overflow-y: scroll;
  padding-right: 10px;
}
.chatBoxBottom {
  margin-top: 5px;
  display: flex;
  align-items: center;
  justify-content: space-between;
}
.chatMessageInput {
  width: 80%;
  height: 90px;
  padding: 10px;
}
.chatSubmitButton {
  width: 70px;
  height: 40px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  background-color: teal;
  color: white;
}
.chatOnline {
  flex: 3;
}
.chatMenuWrapper,
.chatBoxWrapper,
.chatOnlineWrapper {
  padding: 10px;
  height: 100%;
}
.noConversationText {
  position: absolute;
  top: 10%;
  font-size: 50px;
  color: rgb(224, 220, 220);
  cursor: default;
}
@media screen and (max-width: 768px) {
  .chatMenu {
    flex: 1;
  }
  .chatMenuInput {
    display: none;
  }
  .chatBox{
    flex: 10;
  }
  .chatOnline{
    flex: 1px;
  }
}
```

Profile page

Profile.jsx

```
import "./profile.css";
import Topbar from "../../components/topbar/Topbar";
import Sidebar from "../../components/sidebar/Sidebar";
import Feed from "../../components/feed/Feed";
import Rightbar from "../../components/rightbar/Rightbar";
import { useEffect, useState } from "react";
import axios from "axios";
import { useParams } from "react-router";
export default function Profile() {
  const PF = process.env.REACT_APP_PUBLIC_FOLDER;
  const [user, setUser] = useState({});
  const username = useParams().username;
  useEffect(() => {
    const fetchUser = async () => {
      const res = await axios.get(`/users?username=${username}`);
      setUser(res.data);
    };
    fetchUser();
  }, [username]);
  return (
    <>
      <Topbar />
      <div className="profile">
        <Sidebar />
        <div className="profileRight">
          <div className="profileRightTop">
            <div className="profileCover">
              <img
                className="profileCoverImg"
                src={
                  user.coverPicture
                    ? PF + user.coverPicture
                    : PF + "person/noCover.png"}
                alt=""/>
              <img
                className="profileUserImg"
                src={
                  user.profilePicture
                    ? PF + user.profilePicture
                    : PF + "person/noAvatar.png"}
                alt=""
              />
            </div>
            <div className="profileInfo">
              <h4 className="profileInfoName">{user.username}</h4>
              <span className="profileInfoDesc">{user.desc}</span>
            </div>
          </div>
          <div className="profileRightBottom">
            <Feed username={username} />
            <Rightbar user={user} />
          </div>
        </div>
      </div>
    </>
  );}
```

Profile.css

```
.profile{
  display: flex;
}
.profileRight{
  flex: 9;
}
.profileCover{
  height: 320px;
  position: relative;
}
.profileCoverImg{
  width: 100%;
```

```
    height: 250px;
    object-fit: cover;
}
.profileUserImg{
    width: 150px;
    height: 150px;
    border-radius: 50%;
    object-fit: cover;
    position: absolute;
    left: 0;
    right: 0;
    margin: auto;
    top: 150px;
    border: 3px solid white;
}
.profileInfo{
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
}
.profileInfoName{
    font-size: 24px;
}
.profileInfoDesc{
    font-weight: 300;
}
```

```
.profileRightBottom{
    display: flex;
}
```

Register page

Register.jsx

```
import axios from "axios";
import { useRef } from "react";
import { Link } from "react-router-dom";
import "./register.css";
import { useHistory } from "react-router";
export default function Register() {
  const username = useRef();
  const email = useRef();
  const password = useRef();
  const passwordAgain = useRef();
  const history = useHistory();
  const handleClick = async (e) => {
    e.preventDefault();
    if (passwordAgain.current.value !== password.current.value) {
      passwordAgain.current.setCustomValidity("Passwords don't match!");
    } else {
      const user = {
        username: username.current.value,
        email: email.current.value,
        password: password.current.value,
      };
      try {
```

```
      await axios.post("/auth/register", user);
      history.push("/login");
    } catch (err) {
      console.log(err);
    }
  }
};
return (
  <div className="login">
    <div className="loginWrapper">
      <div className="loginLeft">
        <h3 className="loginLogo">Phinlin</h3>
        <span className="loginDesc">
          Connect with People's and the world around you on Lamasocial.
        </span>
      </div>
      <div className="loginRight">
        <form className="loginBox" onSubmit={handleClick}>
          <input
            placeholder="Username"
            required
            ref={username}
            className="loginInput"
          />
          <input
            placeholder="Email"
            required
            ref={email}
            className="loginInput"
            type="email"
          />
          <input
            placeholder="Password"
            required
            ref={password}
            className="loginInput"
            type="password"
            minLength="6"
          />
          <input
            placeholder="Password Again"
            required
            ref={passwordAgain}
            className="loginInput"
            type="password"
          />
          <button className="loginButton" type="submit">
            Sign Up
          </button>
          <Link to="/login" style={{ textDecoration: "none" }} >
          <button className="loginRegisterButton">Log into Account</button>
          </Link>
        </form>
      </div>
```

```
      </div>
    </div>
  );
}
```

Register.css

```
.login {
  width: 100vw;
  height: 100vh;
  background-color: #f0f2f5;
  display: flex;
  align-items: center;
  justify-content: center;
}
.loginWrapper {
  width: 70%;
  height: 70%;
  display: flex;
}
.loginLeft,
.loginRight {
  flex: 1;
  display: flex;
  flex-direction: column;
  justify-content: center;
```

```
}
.loginLogo {
  font-size: 50px;
  font-weight: 800;
  color: #B026FF;
  margin-bottom: 10px;
}
.loginDesc {
  font-size: 24px;
}
.loginBox{
    height: 400px;
    padding: 20px;
    background-color: white;
    border-radius: 10px;
    display: flex;
    flex-direction: column;
    justify-content: space-between;
}
.loginInput{
    height: 50px;
    border-radius: 10px;
    border: 1px solid gray;
    font-size: 18px;
    padding-left: 20px;
}
.loginInput:focus{
    outline: none;
}
.loginButton{
    height: 50px;
    border-radius: 10px;
    border: none;
    background-color: #1775ee;
    color: white;
    font-size: 20px;
    font-weight: 500;
    cursor: pointer;
}
.loginForgot{
    text-align: center;
    color: #1775ee;
}
.loginRegisterButton{
    width: 100%;
    align-self: center;
    height: 50px;
    border-radius: 10px;
    border: none;
    background-color: #42b72a;
    color: white;
    font-size: 20px;
    font-weight: 500;
    cursor: pointer;
```

```
}
```

Context:

AuthAction.js

```
export const LoginStart = (userCredentials) => ({
  type: "LOGIN_START",
});
export const LoginSuccess = (user) => ({
  type: "LOGIN_SUCCESS",
  payload: user,
});
export const LoginFailure = () => ({
  type: "LOGIN_FAILURE",
});
export const Follow = (userId) => ({
  type: "FOLLOW",
  payload: userId,
});
export const Unfollow = (userId) => ({
  type: "UNFOLLOW",
  payload: userId,
});
```

AuthContext.js

```
import { createContext, useEffect, useReducer } from "react";
import AuthReducer from "./AuthReducer";
const INITIAL_STATE = {
  user:JSON.parse(localStorage.getItem("user")) || null,
  isFetching: false,
  error: false,
};
export const AuthContext = createContext(INITIAL_STATE);
export const AuthContextProvider = ({ children }) => {
  const [state, dispatch] = useReducer(AuthReducer, INITIAL_STATE);
  useEffect(()=>{
    localStorage.setItem("user", JSON.stringify(state.user))
  },[state.user])
  return (
    <AuthContext.Provider
      value={{
        user: state.user,
        isFetching: state.isFetching,
        error: state.error,
        dispatch,
      }}
    >
      {children}
    </AuthContext.Provider>
  );
};
```

AuthReducer.js

```
const AuthReducer = (state, action) => {
  switch (action.type) {
    case "LOGIN_START":
      return {
        user: null,
        isFetching: true,
        error: false,
      };
    case "LOGIN_SUCCESS":
      return {
        user: action.payload,
        isFetching: false,
        error: false,
      };
    case "LOGIN_FAILURE":
      return {
        user: null,
        isFetching: false,
        error: true,
      };
    case "FOLLOW":
      return {
        ...state,
        user: {
          ...state.user,
          followings: [...state.user.followings, action.payload],
        },
      };
    case "UNFOLLOW":
      return {
        ...state,
        user: {
          ...state.user,
          followings: state.user.followings.filter(
            (following) => following !== action.payload
          ),
        },
      };
    default:
      return state;
  }
};
export default AuthReducer;
```

Component:

chatOnline Component:

chatOnlin,jsx

```
import axios from "axios";
import { useEffect, useState } from "react";
import "./chatOnline.css";
export default function ChatOnline({ onlineUsers, currentId, setCurrentChat }) {
  const [friends, setFriends] = useState([]);
  const [onlineFriends, setOnlineFriends] = useState([]);
  const PF = process.env.REACT_APP_PUBLIC_FOLDER;
  useEffect(() => {
    const getFriends = async () => {
      const res = await axios.get("/users/friends/" + currentId);
      setFriends(res.data);
    };
  getFriends();
}, [currentId]);
useEffect(() => {
  setOnlineFriends(friends.filter((f) => onlineUsers.includes(f._id)));
}, [friends, onlineUsers]);
const handleClick = async (user) => {
  try {
    const res = await axios.get(
      `/conversations/find/${currentId}/${user._id}`
    );
    setCurrentChat(res.data);
  } catch (err) {
    console.log(err); }};
return (
  <div className="chatOnline">
    {onlineFriends.map((o) => (
      <div className="chatOnlineFriend" onClick={() => handleClick(o)}>
        <div className="chatOnlineImgContainer">
          <img
            className="chatOnlineImg"
            src={
              o?.profilePicture
                ? PF + o.profilePicture
                : PF + "person/noAvatar.png" }
            alt=""
          />
          <div className="chatOnlineBadge"></div>
          </div>
          <span className="chatOnlineName">{o?.username}</span>
        </div>
      ))}
    </div>
  );
}
```

chatOnlin.css

```
.chatOnlineFriend {
  display: flex;
  align-items: center;
  font-weight: 500;
  cursor: pointer;
  margin-top: 10px;
}
.chatOnlineImgContainer {
  position: relative;
  margin-right: 10px;
}
.chatOnlineImg {
  width: 40px;
  height: 40px;
  border-radius: 50%;
  object-fit: cover;
  border: 1px solid white;
}
.chatOnlineBadge {
  width: 10px;

  height: 10px;

  border-radius: 50%;

  background-color: limegreen;

  position: absolute;

  top: 2px;

  right: 2px;

}

@media screen and (max-width: 768px) {

  .chatOnlineName {

    display: none;

  }

}
```

closeFriend Component:

CloseFriend.jsx

```
import "./closeFriend.css";

export default function CloseFriend({user}) {

  const PF = process.env.REACT_APP_PUBLIC_FOLDER;

  return (

    <li className="sidebarFriend">

      <img className="sidebarFriendImg" src={PF+user.profilePicture} alt="" />

      <span className="sidebarFriendName">{user.username}</span>

    </li>

  );

}
```

closeFriend.css

```
.sidebarFriend {
    display: flex;
    align-items: center;
    margin-bottom: 15px; }
  .sidebarFriendImg {
    width: 32px;
    height: 32px;
    border-radius: 50%;
    object-fit: cover;
    margin-right: 10px; }
```

Conversations Component:

Conversation.jsx

```
import axios from "axios";
import { useEffect, useState } from "react";
import "./conversation.css";
export default function Conversation({ conversation, currentUser }) {
  const [user, setUser] = useState(null);
  const PF = process.env.REACT_APP_PUBLIC_FOLDER;
  useEffect(() => {
    const friendId = conversation.members.find((m) => m !== currentUser._id);
    const getUser = async () => {
      try {
        const res = await axios("/users?userId=" + friendId);
        setUser(res.data);
      } catch (err) {
        console.log(err); }};
    getUser();
  }, [currentUser, conversation]);
  return (
    <div className="conversation">
      <img
        className="conversationImg"
        src={
          user?.profilePicture
            ? PF + user.profilePicture
            : PF + "person/noAvatar.png"}
        alt=""
      />
      <span className="conversationName">{user?.username}</span>
    </div>);}
```

Conversation.css

```
.conversation {
  display: flex;
  align-items: center;
  padding: 10px;
  cursor: pointer;
  margin-top: 20px;}
.conversation:hover {
  background-color: rgb(245, 243, 243);}
.conversationImg {
  width: 40px;
  height: 40px;
  border-radius: 50%;
  object-fit: cover;
  margin-right: 20px;}
```

```
.conversationName {
  font-weight: 500;}
@media screen and (max-width: 768px) {
  .conversationName {
    display: none;}}
```

Feed Component:

Feed.jsx

```
import { useContext, useEffect, useState } from "react";
import Post from "../post/Post";
import Share from "../share/Share";
import "./feed.css";
import axios from "axios";
import { AuthContext } from "../../context/AuthContext";
export default function Feed({ username }) {
  const [posts, setPosts] = useState([]);
  const { user } = useContext(AuthContext);
  useEffect(() => {
    const fetchPosts = async () => {
      const res = username
        ? await axios.get("/posts/profile/" + username)
        : await axios.get("posts/timeline/" + user._id);
      setPosts(
        res.data.sort((p1, p2) => {
          return new Date(p2.createdAt) - new Date(p1.createdAt);}));};
    fetchPosts();
  }, [username, user._id]);
  return (
    <div className="feed">
      <div className="feedWrapper">
        {(!username || username === user.username) && <Share />}
        {posts.map((p) => (
          <Post key={p._id} post={p} />))}
      </div>
    </div>);}
```

feed.css

```
.feed{
    flex:5.5}
.feedWrapper{
    padding: 20px;}
```

Message Component:

Message.jsx

```
import "./message.css";
import { format } from "timeago.js";
export default function Message({ message, own }) {
  return (
    <div className={own ? "message own" : "message"}>
      <div className="messageTop">
        <img
          className="messageImg"
          src="https://images.pexels.com/photos/3686769/pexels-photo-
3686769.jpeg?auto=compress&cs=tinysrgb&dpr=2&w=500"
```

```
          alt=""
        />
        <p className="messageText">{message.text}</p>
      </div>
      <div className="messageBottom">{format(message.createdAt)}</div>
    </div>
  );
}
```

message.css

```
.message {
  display: flex;
  flex-direction: column;
  margin-top: 20px;}
.messageTop{
    display: flex;}
.messageImg {
  width: 32px;
  height: 32px;
  border-radius: 50%;
  object-fit: cover;
  margin-right: 10px;}
.messageText{
    padding: 10px;
    border-radius: 20px;
    background-color: #B026FF;
    color: white;
    max-width: 300px;}
```

```
.messageBottom{
    font-size: 12px;
    margin-top: 10px;}
.message.own{
    align-items: flex-end;}
.message.own .messageText{
    background-color: rgb(245, 241, 241);
    color: black;}
```

Online Component:

Online.jsx

```
import "./online.css";
export default function Online({user}) {
  const PF = process.env.REACT_APP_PUBLIC_FOLDER;
  return (
    <li className="rightbarFriend">
      <div className="rightbarProfileImgContainer">
        <img className="rightbarProfileImg" src={PF+user.profilePicture} alt="" />
        <span className="rightbarOnline"></span>
      </div>
      <span className="rightbarUsername">{user.username}</span>
    </li>  );}
```

online.css

```
.rightbarFriend {
    display: flex;
    align-items: center;
    margin-bottom: 15px;}
  .rightbarProfileImgContainer {
    margin-right: 10px;
    position: relative;}
  .rightbarProfileImg {
    width: 40px;
    height: 40px;
    border-radius: 50%;
    object-fit: cover;}
  .rightbarOnline{
      width: 12px;
      height: 12px;
      border-radius: 50%;
      background-color: limegreen;
      position: absolute;
      top: -2px;
      right: 0;
      border: 2px solid white; }
  .rightbarUsername{
      font-weight: 500; }
```

Post Component:

Post.jsx

```
import "./post.css";
import { MoreVert } from "@material-ui/icons";
import { useContext, useEffect, useState } from "react";
import axios from "axios";
import { format } from "timeago.js";
import { Link } from "react-router-dom";
import { AuthContext } from "../../context/AuthContext";
```

```
export default function Post({ post }) {
  const [like, setLike] = useState(post.likes.length);
  const [isLiked, setIsLiked] = useState(false);
  const [user, setUser] = useState({});
  const PF = process.env.REACT_APP_PUBLIC_FOLDER;
  const { user: currentUser } = useContext(AuthContext);
  useEffect(() => {
    setIsLiked(post.likes.includes(currentUser._id));
  }, [currentUser._id, post.likes]);
  useEffect(() => {
    const fetchUser = async () => {
      const res = await axios.get(`/users?userId=${post.userId}`);
      setUser(res.data);
    };
    fetchUser();
  }, [post.userId]);
  const likeHandler = () => {
    try {
      axios.put("/posts/" + post._id + "/like", { userId: currentUser._id });
    } catch (err) {}
    setLike(isLiked ? like - 1 : like + 1);
    setIsLiked(!isLiked);
  };
  return (
    <div className="post">
      <div className="postWrapper">
        <div className="postTop">
  <div className="postTopLeft">
    <Link to={`/profile/${user.username}`}>
      <img
        className="postProfileImg"
        src={
          user.profilePicture
            ? PF + user.profilePicture
            : PF + "person/noAvatar.png"
        }
        alt=""
      />
    </Link>
    <span className="postUsername">{user.username}</span>
    <span className="postDate">{format(post.createdAt)}</span>
  </div>
  <div className="postTopRight">
    <MoreVert />
  </div>
</div>
<div className="postCenter">
  <span className="postText">{post?.desc}</span>
  <img className="postImg" src={PF + post.img} alt="" />
</div>
<div className="postBottom">
  <div className="postBottomLeft">
    <img
      className="likeIcon"
```

```
              src={`${PF}like.png`}
              onClick={likeHandler}
              alt="" />
            <img
              className="likeIcon"
              src={`${PF}heart.png`}
              onClick={likeHandler}
              alt="" />
            <span className="postLikeCounter">{like} people like it</span>
          </div>
          <div className="postBottomRight">
            <span className="postCommentText">{post.comment} comments</span>
          </div>
        </div>
      </div>
    </div> );}
```

post.css

```
.post {
  width: 100%;
  border-radius: 10px;
  -webkit-box-shadow: 0px 0px 16px -8px rgba(0, 0, 0, 0.68);
  box-shadow: 0px 0px 16px -8px rgba(0, 0, 0, 0.68);
  margin: 30px 0;
}
.postWrapper {
  padding: 10px;}
.postTop {
  display: flex;
  align-items: center;
  justify-content: space-between;}
.postTopLeft {
  display: flex;
  align-items: center;}
.postProfileImg {
  width: 32px;
  height: 32px;
  border-radius: 50%;
  object-fit: cover;}
.postUsername {
  font-size: 15px;
  font-weight: 500;
  margin: 0 10px;}
.postDate{
    font-size: 12px;}
.postCenter{
    margin: 20px 0;}
.postImg{
    margin-top: 20px;
    width: 100%;
    max-height: 500px;
    object-fit: contain;}
.postBottom{
    display: flex;
    align-items: center;
    justify-content: space-between;}
.postBottomLeft{
    display: flex;
    align-items: center;   }
.likeIcon{
    width: 24px;
    height: 24px;
    margin-right: 5px;
    cursor: pointer;}
.postLikeCounter{
    font-size: 15px;}
.postCommentText{
    cursor: pointer;
    border-bottom: 1px dashed gray;
    font-size: 15px;}
```

Rightbar Component:

Rightbar.jsx

```
import "./rightbar.css";
import { Users } from "../../dummyData";
import Online from "../online/Online";
import { useContext, useEffect, useState } from "react";
import axios from "axios";
import { Link } from "react-router-dom";
import { AuthContext } from "../../context/AuthContext";
import { Add, Remove } from "@material-ui/icons";
export default function Rightbar({ user }) {
  const PF = process.env.REACT_APP_PUBLIC_FOLDER;
  const [friends, setFriends] = useState([]);
  const { user: currentUser, dispatch } = useContext(AuthContext);
  const [followed, setFollowed] = useState(
    currentUser.followings.includes(user?.id)
  );
  useEffect(() => {
    const getFriends = async () => {
      try {
        const friendList = await axios.get("/users/friends/" + user._id);
        setFriends(friendList.data);
      } catch (err) {
        console.log(err);
      }
    };
    getFriends();
  }, [user]);
  const handleClick = async () => {
    try {
      if (followed) {
        await axios.put(`/users/${user._id}/unfollow`, {
          userId: currentUser._id,
        });
        dispatch({ type: "UNFOLLOW", payload: user._id });
      } else {
        await axios.put(`/users/${user._id}/follow`, {
          userId: currentUser._id,
        });
        dispatch({ type: "FOLLOW", payload: user._id });
      }
      setFollowed(!followed);
    } catch (err) {}};
  const HomeRightbar = () => {
    return (
      <>
        <div className="birthdayContainer">
          <img className="birthdayImg" src="assets/gift.png" alt="" />
          <span className="birthdayText">
            <b>Pola Foster</b> and <b>3 other friends</b> have a birhday today.
          </span>
        </div>
        <img className="rightbarAd" src="assets/ad.png" alt="" />
        <h4 className="rightbarTitle">Online Friends</h4>
        <ul className="rightbarFriendList">
          {Users.map((u) => (
            <Online key={u.id} user={u} />
          ))}
        </ul>
      </> );};
  const ProfileRightbar = () => {
    return (
      <>
        {user.username !== currentUser.username && (
          <button className="rightbarFollowButton" onClick={handleClick}>
            {followed ? "Unfollow" : "Follow"}
            {followed ? <Remove /> : <Add />}
          </button>
        )}
        <h4 className="rightbarTitle">User information</h4>
        <div className="rightbarInfo">
          <div className="rightbarInfoItem">
            <span className="rightbarInfoKey">City:</span>
            <span className="rightbarInfoValue">{user.city}</span>
          </div>
          <div className="rightbarInfoItem">
            <span className="rightbarInfoKey">From:</span>
            <span className="rightbarInfoValue">{user.from}</span>
          </div>
          <div className="rightbarInfoItem">
            <span className="rightbarInfoKey">Relationship:</span>
            <span className="rightbarInfoValue">
              {user.relationship === 1
                ? "Single"
                : user.relationship === 1
                ? "Married"
                : "-"}
            </span>
          </div>
        </div>
        <h4 className="rightbarTitle">User friends</h4>
        <div className="rightbarFollowings">
          {friends.map((friend) => (
                  <Link
                    to={"/profile/" + friend.username}
                    style={{ textDecoration: "none" }}
                  >
                    <div className="rightbarFollowing">
                      <img
                        src={
                          friend.profilePicture
                            ? PF + friend.profilePicture
                            : PF + "person/noAvatar.png" }
                        alt=""
                        className="rightbarFollowingImg"
                      />
                      <span className="rightbarFollowingName">{friend.username}</span>
                    </div>
                  </Link>
                ))}
              </div>
            </>
          );
        };
        return (
          <div className="rightbar">
            <div className="rightbarWrapper">
              {user ? <ProfileRightbar /> : <HomeRightbar />}
            </div>
          </div>
```

```
 );
}
```

rightbar.css

```
.rightbar {
  flex: 3.5;
}
.rightbarWrapper {
  padding: 20px 20px 0 0;
}
.birthdayContainer {
  display: flex;
  align-items: center;}
.birthdayImg {
  width: 40px;
  height: 40px;
  margin-right: 10px;}
.birthdayText {
  font-weight: 300;
  font-size: 15px;}
.rightbarAd {
  width: 100%;
  border-radius: 10px;
  margin: 30px 0;}
.rightbarTitle{
    margin-bottom: 20px;}
.rightbarFriendList {
  padding: 0;
  margin: 0;
  list-style: none;}
.rightbarTitle{
  font-size: 18px;
  font-weight: 500;
  margin-bottom: 10px;}
.rightbarInfo{
  margin-bottom: 30px;}
.rightbarInfoItem{
  margin-bottom: 10px;}
.rightbarInfoKey{
  font-weight: 500;
  margin-right: 15px;
  color: #555;}
.rightbarInfoValue{
  font-weight: 300;}
.rightbarFollowings{
  display: flex;
  flex-wrap: wrap;
  justify-content: space-between;}
.rightbarFollowing{
  display: flex;
  flex-direction: column;
  margin-bottom: 20px;
  cursor: pointer;}
.rightbarFollowingImg{
  width: 100px;
  height: 100px;
  object-fit: cover;
  border-radius: 5px;}
.rightbarFollowButton{
  margin-top: 30px;
  margin-bottom: 10px;
  border: none;
  background-color: #B026FF;
  color: white;
  border-radius: 5px;
  padding: 5px 10px;
  display: flex;
  align-items: center;
  font-size: 16px;
  font-weight: 500;
  cursor: pointer;}
.rightbarFollowButton:focus{
  outline: none;}
```

Share Component:

Share.jsx

```
import "./share.css";
import {
  PermMedia,
  Label,
  Room,
  EmojiEmotions,
  Cancel,
} from "@material-ui/icons";
import { useContext, useRef, useState } from "react";
import { AuthContext } from "../../context/AuthContext";
import axios from "axios";
export default function Share() {
  const { user } = useContext(AuthContext);
  const PF = process.env.REACT_APP_PUBLIC_FOLDER;
  const desc = useRef();
  const [file, setFile] = useState(null);
  const submitHandler = async (e) => {
    e.preventDefault();
    const newPost = {
      userId: user._id,
      desc: desc.current.value, };
    if (file) {
      const data = new FormData();
      const fileName = Date.now() + file.name;
      data.append("name", fileName);
      data.append("file", file);
      newPost.img = fileName;
      console.log(newPost);
      try {
        await axios.post("/upload", data);
      } catch (err) {} }
    try {
      await axios.post("/posts", newPost);
      window.location.reload();
    } catch (err) {} };


  return (
    <div className="share">
      <div className="shareWrapper">
        <div className="shareTop">
          <img
            className="shareProfileImg"
            src={
              user.profilePicture
                ? PF + user.profilePicture
                : PF + "person/noAvatar.png"
            }
            alt=""
          />
          <input
            placeholder={"What's in your mind " + user.username + "?"}
            className="shareInput"
            ref={desc}
          />
        </div>
        <hr className="shareHr" />
        {file && (
          <div className="shareImgContainer">
            <img className="shareImg" src={URL.createObjectURL(file)} alt="" />
            <Cancel className="shareCancelImg" onClick={() => setFile(null)} />
          </div> )}
```

```
<form className="shareBottom" onSubmit={submitHandler}>
  <div className="shareOptions">
    <label htmlFor="file" className="shareOption">
      <PermMedia htmlColor="tomato" className="shareIcon" />
      <span className="shareOptionText">Photo or Video</span>
      <input
        style={{ display: "none" }}
        type="file"
        id="file"
        accept=".png,.jpeg,.jpg"
        onChange={(e) => setFile(e.target.files[0])}
      />
    </label>
    <div className="shareOption">
      <Label htmlColor="blue" className="shareIcon" />
      <span className="shareOptionText">Tag</span>
    </div>
    <div className="shareOption">
      <Room htmlColor="green" className="shareIcon" />
      <span className="shareOptionText">Location</span>
    </div>
    <div className="shareOption">
      <EmojiEmotions htmlColor="goldenrod" className="shareIcon" />
      <span className="shareOptionText">Feelings</span>
    </div>
  </div>
  <button className="shareButton" type="submit">
```

```
          Share
        </button>
      </form>
    </div>
  </div>
 );
}
```

share.css

```
.share {
  width: 100%;
  border-radius: 10px;
  -webkit-box-shadow: 0px 0px 16px -8px rgba(0, 0, 0, 0.68);
  box-shadow: 0px 0px 16px -8px rgba(0, 0, 0, 0.68);}
.shareWrapper {
  padding: 10px;}
.shareTop {
  display: flex;
  align-items: center;}
.shareProfileImg {
  width: 50px;
  height: 50px;
  border-radius: 50%;
  object-fit: cover;
  margin-right: 10px;}
.shareInput {
  border: none;
  width: 80%;}
```

```
.shareInput:focus {
  outline: none;}
.shareHr {
  margin: 20px;}
.shareBottom {
  display: flex;
  align-items: center;
  justify-content: space-between;}
.shareOptions{
    display: flex;
    margin-left: 20px;}
.shareOption{
    display: flex;
    align-items: center;
    margin-right: 15px;
    cursor: pointer;}
.shareIcon{
    font-size: 18px;
    margin-right: 3px;}
.shareOptionText{
    font-size: 14px;
    font-weight: 500;}
.shareButton{
    border: none;
    padding: 7px;
    border-radius: 5px;
    background-color: green;
    font-weight: 500;
    margin-right: 20px;
    cursor: pointer;
    color: white;}
.shareImgContainer{
  padding: 0 20px 10px 20px;
  position: relative;}
.shareImg{
  width: 100%;
  object-fit: cover;}
.shareCancelImg{
  position: absolute;
  top: 0;
  right: 20px;
  cursor: pointer;
  opacity: 0.7;}
```

Sidebar Component:

Sidebar.jsx

```
import "./sidebar.css";
import { Link } from "react-router-dom";
import {
  Chat,
  WorkOutline,
  Event,
  School,
} from "@material-ui/icons";
import { Users } from "../../dummyData";
import CloseFriend from "../closeFriend/CloseFriend";

export default function Sidebar() {

  return (

    <div className="sidebar">

      <div className="sidebarWrapper">

        <ul className="sidebarList">

          <Link to="/messenger" style={{ textDecoration: "none" }}>

          <li className="sidebarListItem">

            <Chat className="sidebarIcon" />

            <span className="sidebarListItemText">Chats</span>

          </li>

          </Link>

          <li className="sidebarListItem">

            <WorkOutline className="sidebarIcon" />

            <span className="sidebarListItemText">Jobs</span>

          </li>

          <li className="sidebarListItem">

            <Event className="sidebarIcon" />

            <span className="sidebarListItemText">Events</span>

          </li>

          <li className="sidebarListItem">

            <School className="sidebarIcon" />

            <span className="sidebarListItemText">Courses</span>

          </li>

        </ul>

        <button className="sidebarButton">Show More</button>

        <hr className="sidebarHr" />
        <ul className="sidebarFriendList">
          {Users.map((u) => (
            <CloseFriend key={u.id} user={u} />
          ))}
        </ul>
      </div>
    </div>);}
```

sidebar.css

```
.sidebar {
  flex: 3;
  height: calc(100vh - 50px);
  overflow-y: scroll;
  position: sticky;
  top: 50px;}
::-webkit-scrollbar {
  width: 5px;}
::-webkit-scrollbar-track {
  background-color: #f1f1f1;}
::-webkit-scrollbar-thumb {
  background-color: rgb(179, 179, 179);}
.sidebarWrapper {
  padding: 20px;}
.sidebarList {
  padding: 0;
  margin: 0;
  list-style: none;}
.sidebarListItem {
  display: flex;
  align-items: center;
  margin-bottom: 20px;}
.sidebarIcon {
  margin-right: 15px;}
.sidebarButton {
  width: 150px;
  border: none;
  padding: 10px;
  border-radius: 5px;
  font-weight: 500;}
.sidebarHr {
  margin: 20px 0;}
.sidebarFriendList {
  padding: 0;
  margin: 0;
  list-style: none;}
```

Topbar Component:

Topbar.jsx

```
import "./topbar.css";
import { Search, Person, Chat, Notifications } from "@material-ui/icons";
import { Link } from "react-router-dom";
import { useContext } from "react";
import { AuthContext } from "../../context/AuthContext";
export default function Topbar() {
  const { user } = useContext(AuthContext);
  const PF = process.env.REACT_APP_PUBLIC_FOLDER;
```

```
return (
  <div className="topbarContainer">
    <div className="topbarLeft">
      <Link to="/" style={{ textDecoration: "none" }}>
        <span className="logo">Phinlin</span>
      </Link>
    </div>
    <div className="topbarCenter">
      <div className="searchbar">
        <Search className="searchIcon" />
        <input
          placeholder="Search for friend, post or video"
          className="searchInput"
        />
      </div>
    </div>
    <div className="topbarRight">
      <div className="topbarLinks">
        <span className="topbarLink">Homepage</span>
        <span className="topbarLink">Timeline</span>
      </div>
      <div className="topbarIcons">
        <div className="topbarIconItem">
          <Person />
          <span className="topbarIconBadge">1</span>
        </div>
        <div className="topbarIconItem">
```

```
          <Chat />
        <Link to="/messenger" style={{ textDecoration: "none" }}>
          <span className="topbarIconBadge">2</span>
          </Link>
        </div>
        <div className="topbarIconItem">
          <Notifications />
          <span className="topbarIconBadge">1</span>
        </div>
      </div>
      <Link to={`/profile/${user.username}`}>
        <img
          src={
            user.profilePicture
              ? PF + user.profilePicture
              : PF + "person/noAvatar.png"
          }
          alt=""
          className="topbarImg"
        />
      </Link>
    </div>
  </div>
 );
}
```

topbar.css

```
.topbarContainer {
  height: 50px;
  width: 100%;
  background-color: #B026FF;
  display: flex;
  align-items: center;
  position: sticky;
  top: 0;
  z-index: 999;}
.topbarLeft {
  flex: 3;}
.logo {
  font-size: 24px;
  margin-left: 20px;
  font-weight: bold;
  color: white;
  cursor: pointer;}
.topbarCenter {
  flex: 5;}
.searchbar {
  width: 100%;
  height: 30px;
  background-color: white;
  border-radius: 30px;
  display: flex;
  align-items: center;}
.searchIcon {
  font-size: 20px !important;
  margin-left: 10px;}
.searchInput {
  border: none;
  width: 70%;}
.searchInput:focus {
  outline: none;}
.topbarRight {
  flex: 4;
  display: flex;
  align-items: center;
  justify-content: space-around;
  color: white;}
.topbarLink {
  margin-right: 10px;
  font-size: 14px;
  cursor: pointer;}
.topbarIcons {
  display: flex;}
.topbarIconItem {
  margin-right: 15px;
  cursor: pointer;
  position: relative;}
.topbarIconBadge {
  width: 15px;
  height: 15px;
  background-color: red;
  border-radius: 50%;
  color: white;
  position: absolute;
  top: -5px;
  right: -5px;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 12px;}
.topbarImg {
  width: 32px;
  height: 32px;
  border-radius: 50%;
  object-fit: cover;
  cursor: pointer;}
@media screen and (max-width: 768px)  {
  .topbarLink{
    display: none;  }}
```

Public Component:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <meta name="description" content="Web site created using create-react-app" />
  <link
href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;
0,900;1,500&display=swap"
    rel="stylesheet">
    <link rel="stylesheet" href="http://localhost:3000/assets/style.css">
  <title>Phinlin</title>
</head>
<body>
  <div id="root"></div>
</body>
</html>
```

5.1.2 Server Side:

Index.js

```
const express = require("express");

const app = express();

const mongoose = require("mongoose");

const dotenv = require("dotenv");

const helmet = require("helmet");

const morgan = require("morgan");

const multer = require("multer");

const userRoute = require("./routes/users");

const authRoute = require("./routes/auth");

const postRoute = require("./routes/posts");

const conversationRoute = require("./routes/conversations");

const messageRoute = require("./routes/messages");

const router = express.Router();

const path = require("path");

dotenv.config();

mongoose.connect(

  process.env.MONGO_URL,

  { useNewUrlParser: true, useUnifiedTopology: true },

  () => {

    console.log("Connected to MongoDB");});

app.use("/images", express.static(path.join(__dirname, "public/images")));

//middleware

app.use(express.json());

app.use(helmet());
app.use(morgan("common"));
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "public/images"); },
  filename: (req, file, cb) => {
    cb(null, req.body.name); },});
const upload = multer({ storage: storage });
app.post("/api/upload", upload.single("file"), (req, res) => {
  try {
    return res.status(200).json("File uploded successfully");
  } catch (error) {
    console.error(error);}});
app.use("/api/auth", authRoute);
app.use("/api/users", userRoute);
app.use("/api/posts", postRoute);
app.use("/api/conversations", conversationRoute);
app.use("/api/messages", messageRoute);
app.listen(8800, () => {
  console.log("Backend server is running!");});
```

Models:

Conversation.js

```
const mongoose = require("mongoose");
const ConversationSchema = new mongoose.Schema(
  {
    members: {
      type: Array,
    },
  },
  { timestamps: true }
);
module.exports = mongoose.model("Conversation", ConversationSchema);
```

Message.js

```
const mongoose = require("mongoose");
const MessageSchema = new mongoose.Schema(
  {
    conversationId: {
      type: String, },
    sender: {
      type: String, },
    text: {
      type: String,}, },
  { timestamps: true });
module.exports = mongoose.model("Message", MessageSchema);
```

Post.js

```
const mongoose = require("mongoose");
const PostSchema = new mongoose.Schema(
  { userId: {
      type: String,
      required: true,},
    desc: {
      type: String,
      max: 500,},
    img: {
      type: String, },
    likes: {
      type: Array,
      default: [],},},
  { timestamps: true });
module.exports = mongoose.model("Post", PostSchema);
```

User.js

```
const mongoose = require("mongoose");
const UserSchema = new mongoose.Schema({
    username: {
      type: String,
      require: true,
      min: 3,
      max: 20,
      unique: true,},
    email: {
      type: String,
      required: true,
      max: 50,
      unique: true,},
    password: {
      type: String,
      required: true,
      min: 6,},
    profilePicture: {
      type: String,
      default: "",},
    coverPicture: {
      type: String,
      default: "",},
    followers: {
      type: Array,
      default: [],},
    followings: {
      type: Array,
      default: [],},
    isAdmin: {
      type: Boolean,
      default: false,},
    desc: {
      type: String,
      max: 50,},
    city: {
      type: String,
      max: 50,},
    from: {
      type: String,
      max: 50,},
    relationship: {
      type: Number,
      enum: [1, 2, 3],},},
  { timestamps: true });
module.exports = mongoose.model("User", UserSchema);
```

Routes:

auth.js

```
const router = require("express").Router();
const User = require("../models/User");
const bcrypt = require("bcrypt");
//REGISTER
router.post("/register", async (req, res) => {
  try {
    //generate new password
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(req.body.password, salt);
    //create new user
    const newUser = new User({
      username: req.body.username,
      email: req.body.email,
      password: hashedPassword,
    });
    //save user and respond
    const user = await newUser.save();
    res.status(200).json(user);
  } catch (err) {
    res.status(500).json(err)
  }
});
//LOGIN
router.post("/login", async (req, res) => {
  try {
    const user = await User.findOne({ email: req.body.email });
    !user && res.status(404).json("user not found");
    const validPassword = await bcrypt.compare(req.body.password, user.password)
    !validPassword && res.status(400).json("wrong password")
    res.status(200).json(user)
  } catch (err) {
    res.status(500).json(err)}});
module.exports = router;
```

conversations.js

```
const router = require("express").Router();
const Conversation = require("../models/Conversation");
router.post("/", async (req, res) => {
  const newConversation = new Conversation({
    members: [req.body.senderId, req.body.receiverId],});
  try {
    const savedConversation = await newConversation.save();
    res.status(200).json(savedConversation);
  } catch (err) {
    res.status(500).json(err);}});
router.get("/:userId", async (req, res) => {
  try {
    const conversation = await Conversation.find({
      members: { $in: [req.params.userId] },
    });
    res.status(200).json(conversation);
  } catch (err) {
    res.status(500).json(err);}});
router.get("/find/:firstUserId/:secondUserId", async (req, res) => {
  try {
    const conversation = await Conversation.findOne({
      members: { $all: [req.params.firstUserId, req.params.secondUserId] },});
    res.status(200).json(conversation)
  } catch (err) {
    res.status(500).json(err);}});
module.exports = router;
```

messages.js

```
const router = require("express").Router();
const Message = require("../models/Message");
router.post("/", async (req, res) => {
  const newMessage = new Message(req.body);
  try {
    const savedMessage = await newMessage.save();
    res.status(200).json(savedMessage);
  } catch (err) {
    res.status(500).json(err);}});
router.get("/:conversationId", async (req, res) => {
  try {
    const messages = await Message.find({
      conversationId: req.params.conversationId,});
    res.status(200).json(messages);
  } catch (err) {
    res.status(500).json(err);}});
module.exports = router;
```

posts.js

```
const router = require("express").Router();
const Post = require("../models/Post");
const User = require("../models/User");
router.post("/", async (req, res) => {
  const newPost = new Post(req.body);
  try {
    const savedPost = await newPost.save();
    res.status(200).json(savedPost);
  } catch (err) {
    res.status(500).json(err);}});
router.put("/:id", async (req, res) => {
  try {
    const post = await Post.findById(req.params.id);
    if (post.userId === req.body.userId) {
      await post.updateOne({ $set: req.body });
      res.status(200).json("the post has been updated");
    } else {
      res.status(403).json("you can update only your post");}
  } catch (err) {
    res.status(500).json(err);}});
router.delete("/:id", async (req, res) => {
  try {
    const post = await Post.findById(req.params.id);
    if (post.userId === req.body.userId) {
      await post.deleteOne();
      res.status(200).json("the post has been deleted");
    } else {
      res.status(403).json("you can delete only your post");
    }
  } catch (err) {
    res.status(500).json(err);}});
//like / dislike a post
router.put("/:id/like", async (req, res) => {
  try {
    const post = await Post.findById(req.params.id);
    if (!post.likes.includes(req.body.userId)) {
      await post.updateOne({ $push: { likes: req.body.userId } });
      res.status(200).json("The post has been liked");
    } else {
      await post.updateOne({ $pull: { likes: req.body.userId } });
      res.status(200).json("The post has been disliked");}
  } catch (err) {
    res.status(500).json(err);}});
//get a post
router.get("/:id", async (req, res) => {
  try {
    const post = await Post.findById(req.params.id);
    res.status(200).json(post);
  } catch (err) {
    res.status(500).json(err);}});

//get timeline posts
router.get("/timeline/:userId", async (req, res) => {
  try {
    const currentUser = await User.findById(req.params.userId);
    const userPosts = await Post.find({ userId: currentUser._id });
    const friendPosts = await Promise.all(
      currentUser.followings.map((friendId) => {
        return Post.find({ userId: friendId });}));
    res.status(200).json(userPosts.concat(...friendPosts));
  } catch (err) {
    res.status(500).json(err);}});
//get user's all posts
router.get("/profile/:username", async (req, res) => {
  try {
    const user = await User.findOne({ username: req.params.username });
    const posts = await Post.find({ userId: user._id });
    res.status(200).json(posts);
  } catch (err) {
    res.status(500).json(err);}});
module.exports = router;
```

users.js

```
const User = require("../models/User");
const router = require("express").Router();
const bcrypt = require("bcrypt");

//update user
router.put("/:id", async (req, res) => {
  if (req.body.userId === req.params.id || req.body.isAdmin) {
    if (req.body.password) {
      try {
        const salt = await bcrypt.genSalt(10);
        req.body.password = await bcrypt.hash(req.body.password, salt);
      } catch (err) {
        return res.status(500).json(err);}}
    try {
      const user = await User.findByIdAndUpdate(req.params.id, {
        $set: req.body,});
      res.status(200).json("Account has been updated");
    } catch (err) {
      return res.status(500).json(err);}
  } else {
    return res.status(403).json("You can update only your account!");}});
//delete user
router.delete("/:id", async (req, res) => {
  if (req.body.userId === req.params.id || req.body.isAdmin) {
    try {
      await User.findByIdAndDelete(req.params.id);
      res.status(200).json("Account has been deleted");
    } catch (err) {
      return res.status(500).json(err);}
  } else {
    return res.status(403).json("You can delete only your account!");}});
//get a user
router.get("/", async (req, res) => {
  const userId = req.query.userId;
  const username = req.query.username;
  try {
    const user = userId
      ? await User.findById(userId)
      : await User.findOne({ username: username });
    const { password, updatedAt, ...other } = user._doc;
    res.status(200).json(other);
  } catch (err) {
    res.status(500).json(err);}});
//get friends
router.get("/friends/:userId", async (req, res) => {
  try {
    const user = await User.findById(req.params.userId);
    const friends = await Promise.all(
      user.followings.map((friendId) => {
        return User.findById(friendId);}));
    let friendList = [];
    friends.map((friend) => {
      const { _id, username, profilePicture } = friend;
      friendList.push({ _id, username, profilePicture });});
    res.status(200).json(friendList)
  } catch (err) {
   res.status(500).json(err);}});
//follow a user
router.put("/:id/follow", async (req, res) => {
  if (req.body.userId !== req.params.id) {
    try {
      const user = await User.findById(req.params.id);
      const currentUser = await User.findById(req.body.userId);
      if (!user.followers.includes(req.body.userId)) {
        await user.updateOne({ $push: { followers: req.body.userId } });
        await currentUser.updateOne({ $push: { followings: req.params.id } });
        res.status(200).json("user has been followed");
      } else {
        res.status(403).json("you allready follow this user");}
    } catch (err) {
      res.status(500).json(err);}
  } else {
    res.status(403).json("you cant follow yourself");}});
//unfollow a user
router.put("/:id/unfollow", async (req, res) => {
  if (req.body.userId !== req.params.id) {
    try {
      const user = await User.findById(req.params.id);
      const currentUser = await User.findById(req.body.userId);
      if (user.followers.includes(req.body.userId)) {
        await user.updateOne({ $pull: { followers: req.body.userId } });
        await currentUser.updateOne({ $pull: { followings: req.params.id } });
        res.status(200).json("user has been unfollowed");
      } else {
        res.status(403).json("you dont follow this user");}
    } catch (err) {
      res.status(500).json(err);}
  } else {
    res.status(403).json("you cant unfollow yourself");}});
module.exports = router;
```

5.1.3 Socket:

Index.js

```
const io = require("socket.io")(8900, {
  cors: {
    origin: "http://localhost:3000",},});
let users = [];
const addUser = (userId, socketId) => {
  !users.some((user) => user.userId === userId) &&
    users.push({ userId, socketId });
};
const removeUser = (socketId) => {
  users = users.filter((user) => user.socketId !== socketId);
};
const getUser = (userId) => {
  return users.find((user) => user.userId === userId);
};
io.on("connection", (socket) => {
  //when ceonnect
  console.log("a user connected.");
  //take userId and socketId from user
  socket.on("addUser", (userId) => {
    addUser(userId, socket.id);
    io.emit("getUsers", users);
  });
  //send and get message
  socket.on("sendMessage", ({ senderId, receiverId, text }) => {
    console.log("receiverId:", receiverId);
    const user = getUser(receiverId);
    io.to(user.socketId).emit("getMessage", {
      senderId,
      text,
    });
  });
  //when disconnect
  socket.on("disconnect", () => {
    console.log("a user disconnected!");
    removeUser(socket.id);
    io.emit("getUsers", users);
  });
});
```

5.2 Testing Approach

5.2.1 Unit Testing:

During this first round of testing, the program is submitted to assessments that focus on specific units or components of the software to determine whether each one is fully functional. The main aim of this endeavour is to determine whether the application functions as designed. In this phase, a unit can refer to a function, individual program or even a procedure, and a White box Testing method is usually used to get the job done. One of the biggest benefits of this testing phase is that it can be run every time a piece of code is changed, allowing issues to be resolved as quickly as possible. It's quite common for software developers to perform unit tests before delivering software to testers for formal testing.

5.2.2 Integration Testing:

Integration testing allows individuals the opportunity to combine all the units within a program and test them as a group. This testing level is designed to find interface defects between the modules/functions. This is particularly beneficial because it determines how efficiently the units are running together. Keep in mind that no matter how efficiently each unit is running, if they aren't properly integrated, it will affect the functionality of the software program. In order to run these types of tests, individuals can make use of various testing methods, but the specific method that will be used to get the job done will depend greatly on the way in winch the units are defined.

5.2.3 Registration and Login Testing:

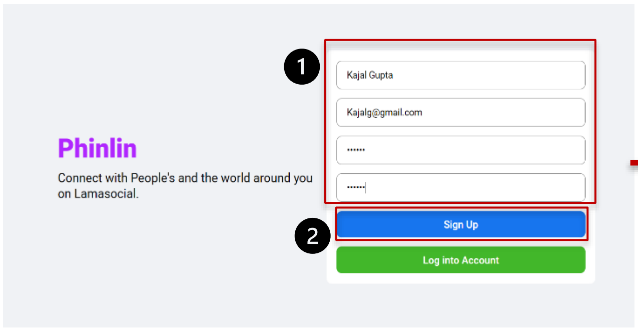

Phinlin
Connect with People's and the world around you on Lamasocial.
1
Kajal Gupta
Kajalg@gmail.com
2
Sign Up
Log into Account

Figure 5.1: Registration Input

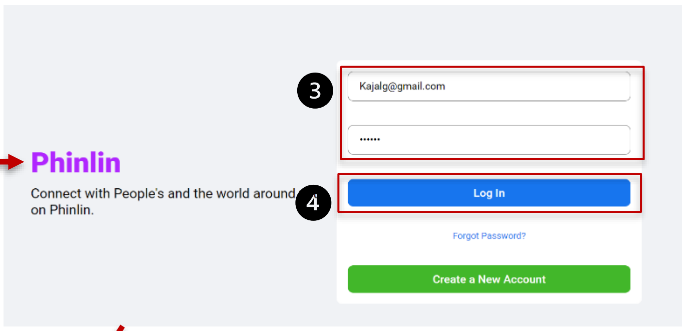


Figure 5.2: Login Input

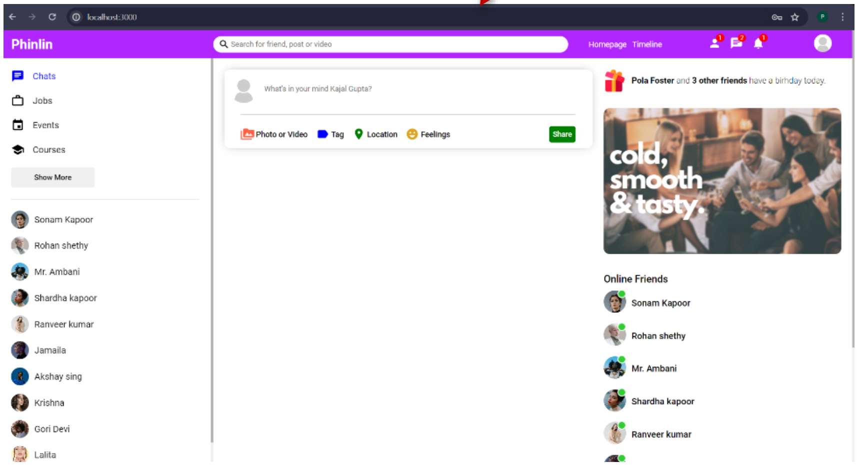
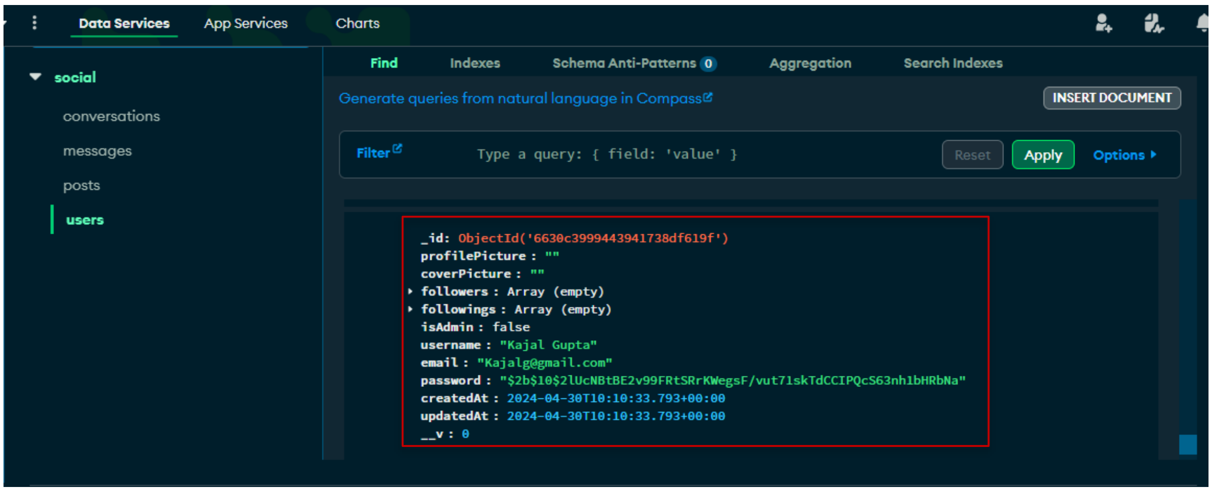


Figure 5.3: Registration and login Output

Description: On the registration/login page, user inputs are seamlessly processed, swiftly granting access to the platform's rich features, ensuring a smooth and secure digital journey.

5.2.3.1 Registration and Login Data:



```
_id: ObjectId('6630c3999443941738df619f')
profilePicture : ""
coverPicture : ""
followers : Array (empty)
followings : Array (empty)
isAdmin : false
username : "Kajal Gupta"
email : "Kajalg@gmail.com"
password : "$2b$10$2lUcNBtBE2v99FRtSRrKWegsF/vut71skTdCCIPQcS63nh1bHRbNa"
createdAt : 2024-04-30T10:10:33.793+00:00
updatedAt : 2024-04-30T10:10:33.793+00:00
__v : 0
```

Figure 5.4: Users DB

Description: User data securely stored in MongoDB Atlas cluster, safeguarding confidentiality and enabling efficient retrieval for personalized experiences.

5.2.4 Post and Like Testing:

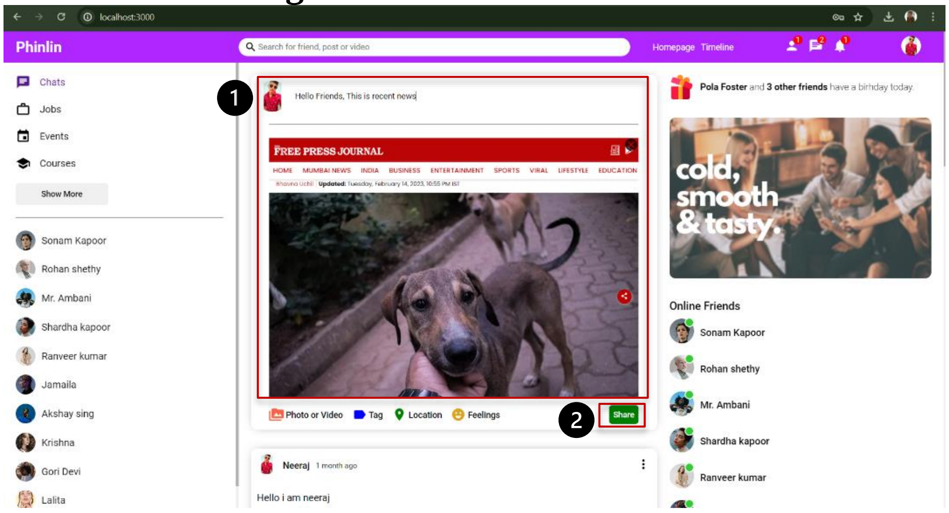


Figure 5.5: Post Input

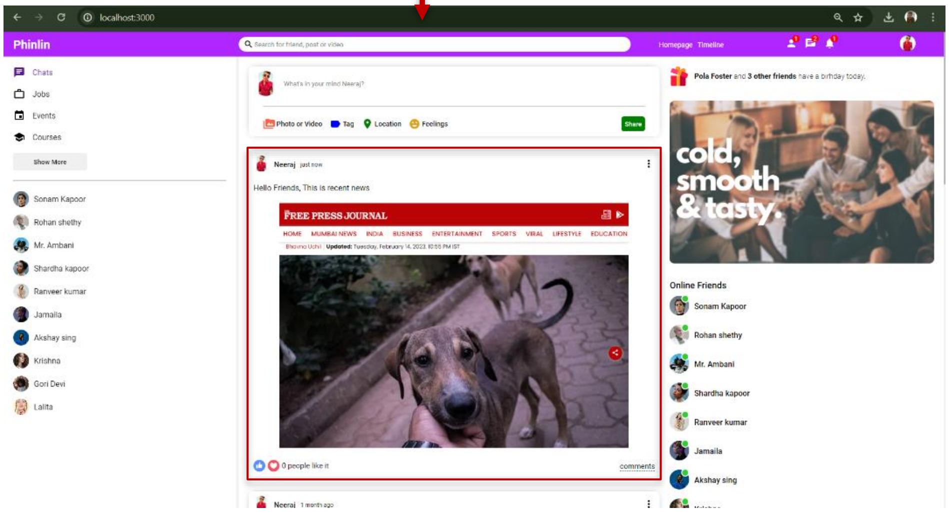


Figure 5.6: Post Output

Description: The post functionality operates flawlessly, allowing users to create, publish, and interact with content effortlessly, fostering a vibrant and dynamic community experience.

5.2.4.1 Post and Like Data:

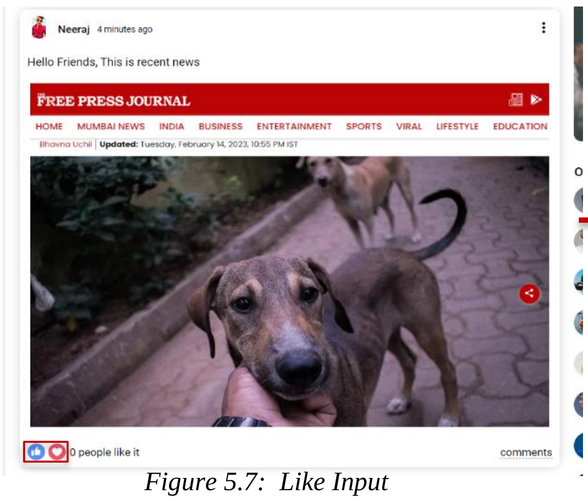

Neeraj 4 minutes ago
Hello Friends, This is recent news
FREE PRESS JOURNAL
HOME MUMBAI NEWS INDIA BUSINESS ENTERTAINMENT SPORTS VIRAL LIFESTYLE EDUCATION
Bhavna Uchil | Updated: Tuesday, February 14, 2023, 10:55 PM IST
0 people like it
comments

Figure 5.7: Like Input

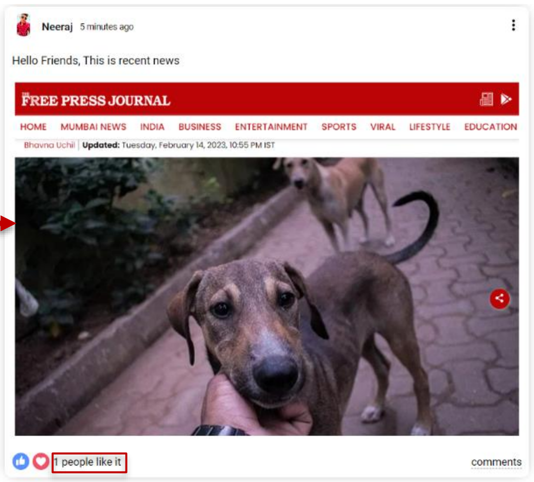
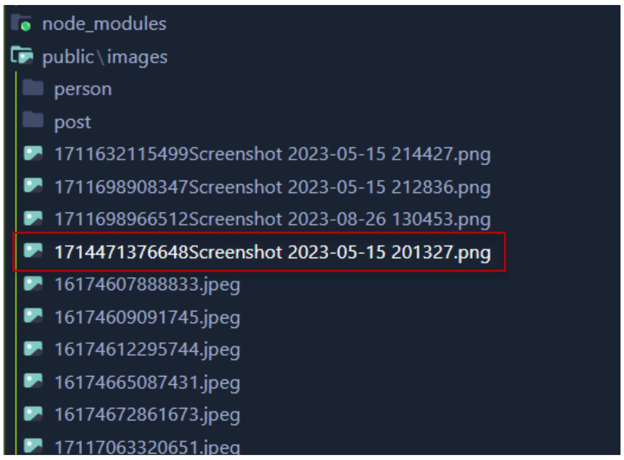

Neeraj 5 minutes ago
Hello Friends, This is recent news
FREE PRESS JOURNAL
HOME MUMBAI NEWS INDIA BUSINESS ENTERTAINMENT SPORTS VIRAL LIFESTYLE EDUCATION
Bhavna Uchil | Updated: Tuesday, February 14, 2023, 10:55 PM IST
1 people like it
comments

Figure 5.8: Like Output

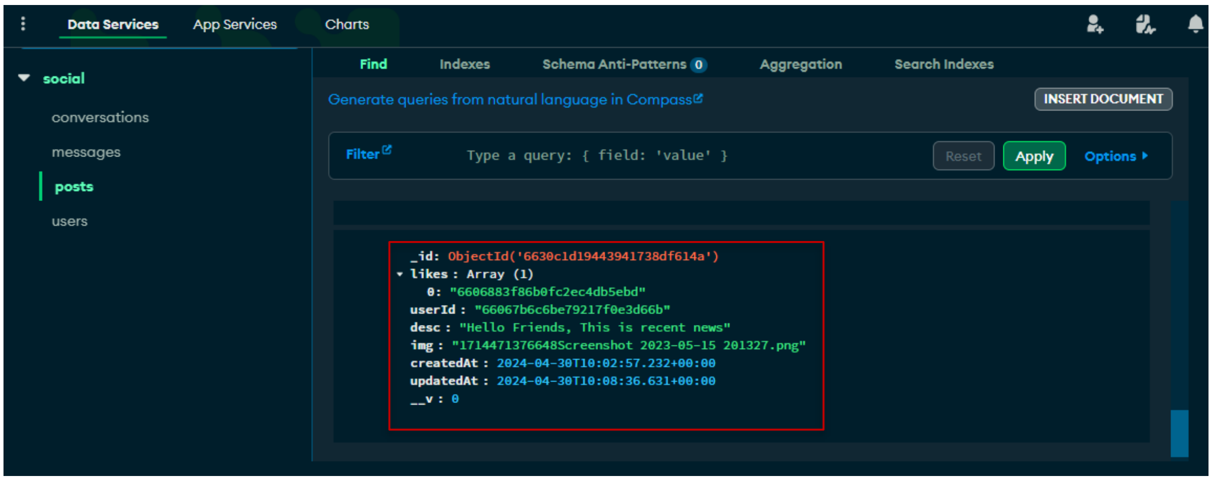
Description: The like button functions seamlessly, instantly updating the count and visually reflecting user interaction, enhancing engagement and content appreciation.

5.2.4.2 Post and Like Data:



```
node_modules
public\images
  person
  post
  1711632115499Screenshot 2023-05-15 214427.png
  1711698908347Screenshot 2023-05-15 212836.png
  1711698966512Screenshot 2023-08-26 130453.png
  1714471376648Screenshot 2023-05-15 201327.png
  16174607888833.jpeg
  16174609091745.jpeg
  16174612295744.jpeg
  16174665087431.jpeg
  16174672861673.jpeg
  17117063320651.jpeg
```

Figure 5.9: Posted Image storing location.

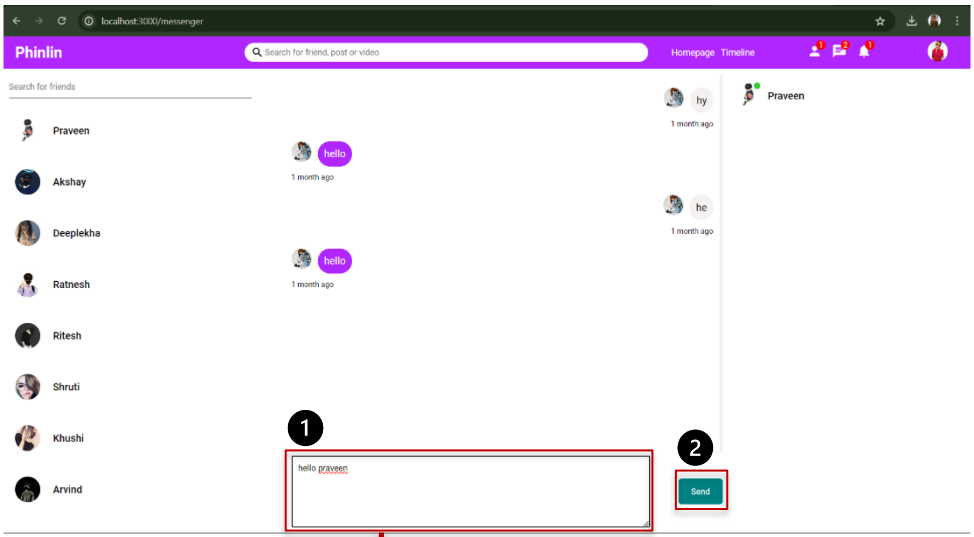


```
  _id: ObjectId('6630c1d19443941738df614a')
▾ likes : Array (1)
    0: "6606883f86b0fc2ec4db5ebd"
  userId : "66067b6c6be79217f0e3d66b"
  desc : "Hello Friends, This is recent news"
  img : "1714471376648Screenshot 2023-05-15 201327.png"
  createdAt : 2024-04-30T10:02:57.232+00:00
  updatedAt : 2024-04-30T10:08:36.631+00:00
  __v : 0
```

Figure 5.10: Post Content stored in DB.

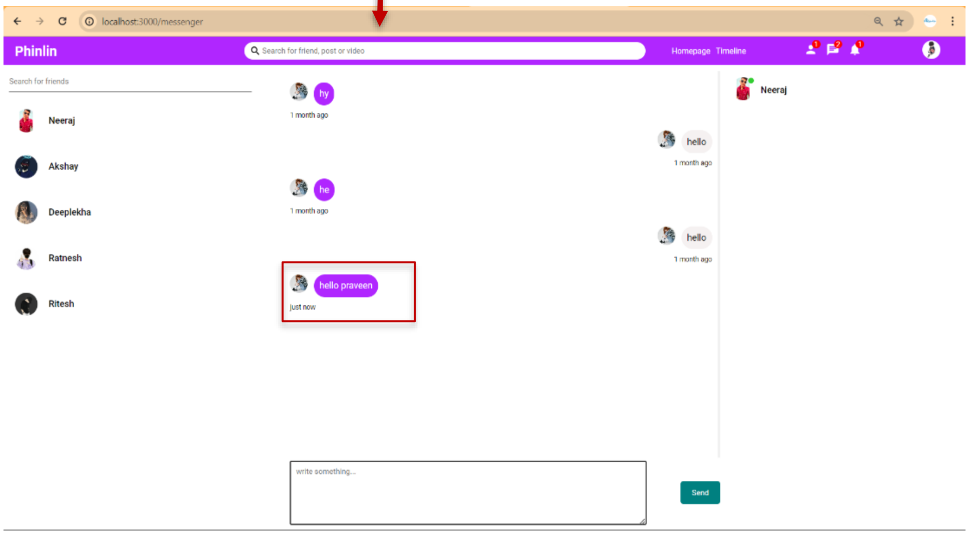
Description: User-posted images are securely stored in local files, ensuring efficient access and preserving data integrity, while post metadata including post ID, like count, and user IDs of likers are stored in MongoDB Atlas cluster.

5.2.5 Chat Testing:



hy
1 month ago
hello
1 month ago
he
1 month ago
hello
1 month ago
hello praveen

Figure 5.11: Chat input.

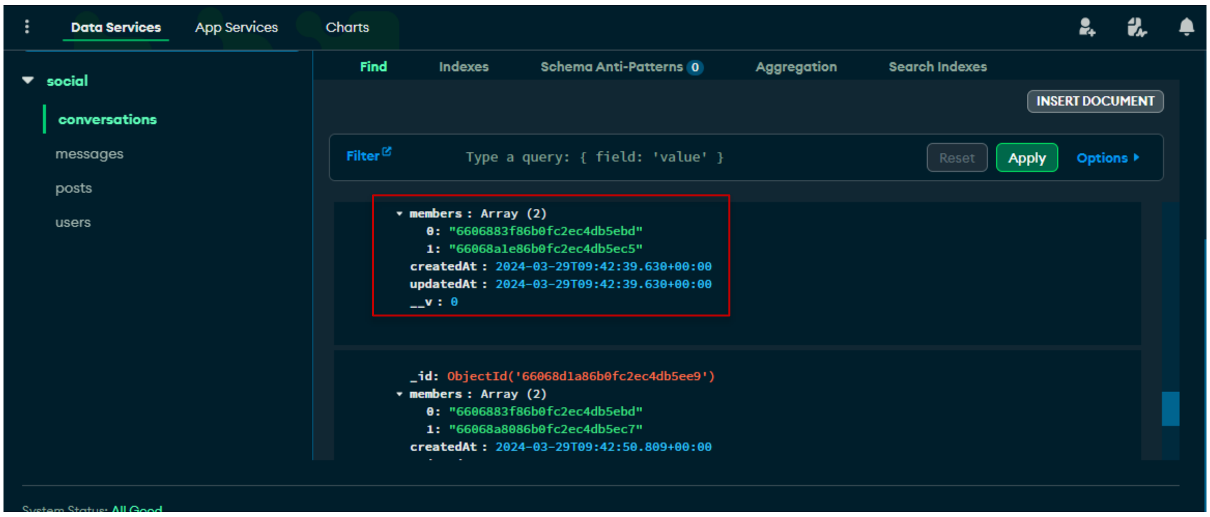


hy
1 month ago
hello
1 month ago
he
1 month ago
hello
1 month ago
hello praveen
just now

Figure 5.12: Chat output.

Description: The live chat feature functions seamlessly, facilitating real-time communication between users with instant message delivery fostering dynamic interactions and efficient support.

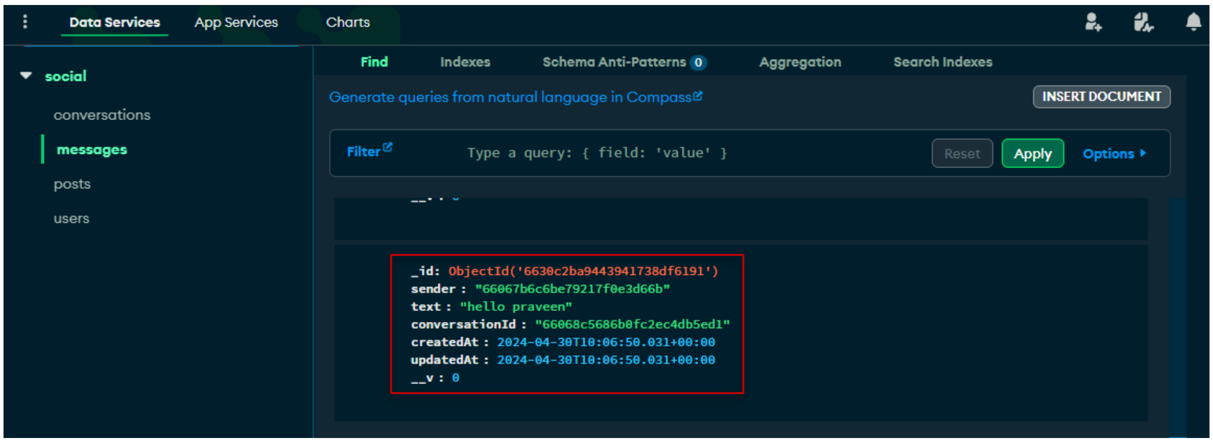
5.2.5.1 Chat Data:



```
▾ members : Array (2)
    0: "6606883f86b0fc2ec4db5ebd"
    1: "66068a1e86b0fc2ec4db5ec5"
  createdAt : 2024-03-29T09:42:39.630+00:00
  updatedAt : 2024-03-29T09:42:39.630+00:00
  __v : 0

  _id: ObjectId('66068d1a86b0fc2ec4db5ee9')
▾ members : Array (2)
    0: "6606883f86b0fc2ec4db5ebd"
    1: "66068a8086b0fc2ec4db5ec7"
  createdAt : 2024-03-29T09:42:50.809+00:00
```

Figure 5.13: Chat Connection UID string Stored Data.



```
_id: ObjectId('6630c2ba9443941738df6191')
sender : "66067b6c6be79217f0e3d66b"
text : "hello praveen"
conversationId : "66068c5686b0fc2ec4db5ed1"
createdAt : 2024-04-30T10:06:50.031+00:00
updatedAt : 2024-04-30T10:06:50.031+00:00
__v : 0
```

Figure 5.14: Chat message stored Data.

Description: Chat conversation messages are securely stored in MongoDB Atlas cluster, preserving message history and ensuring data integrity, with unique connection IDs facilitating streamlined communication and message tracking for seamless user experience.

Chapter 6

Results and Discussions (Output Screen)

6.1 Login

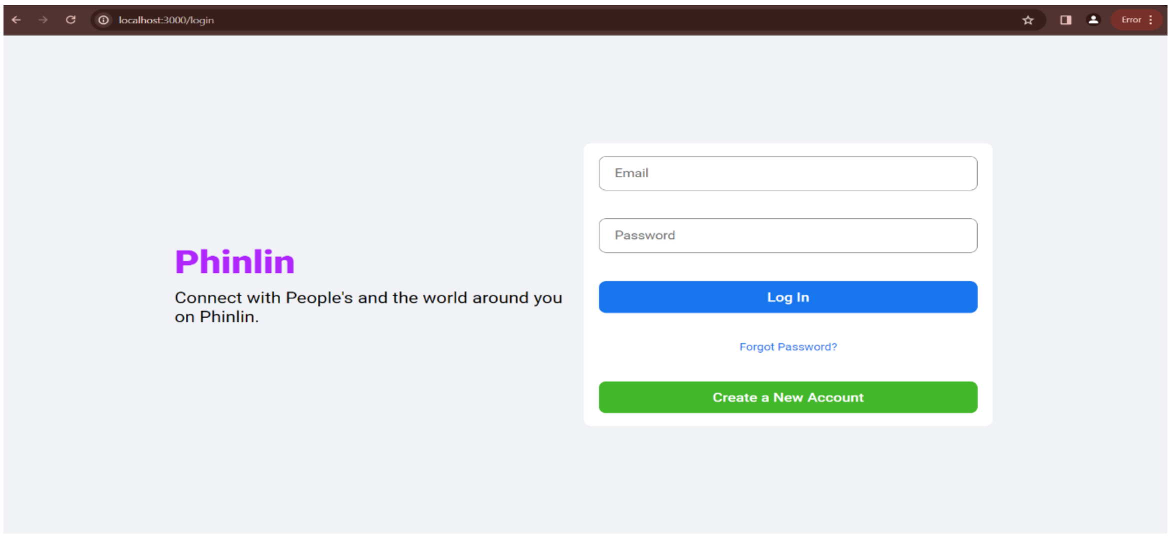


Figure 6.1: Login

Description: Phinlin's login: where simplicity meets security. Access our services seamlessly with user-friendly authentication and robust encryption. Experience hassle-free login and personalized account options tailored to your preferences.

6.2 Register

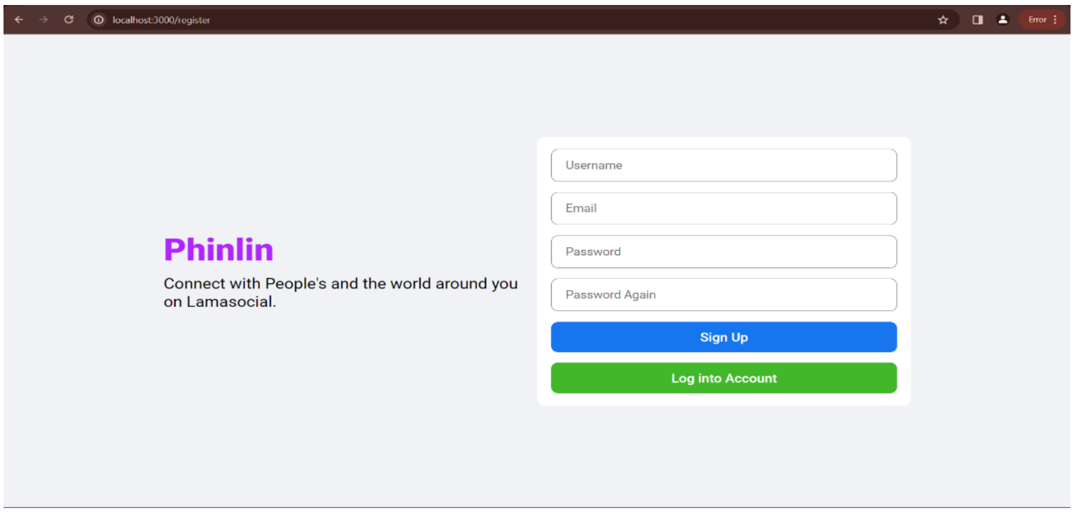


Figure 6.2: Register

Description: Join Phinlin effortlessly with our user-friendly registration page. Experience seamless account creation and personalized options tailored to your needs, all with top-notch security measures in place.

6.3 Home

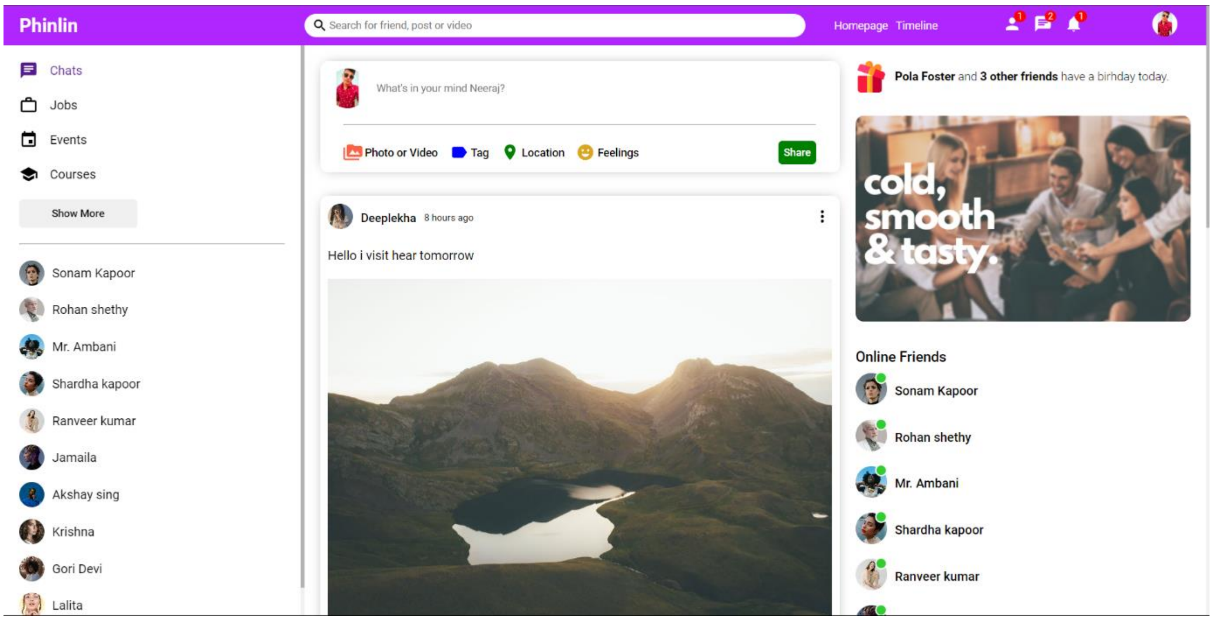


Figure 6.3: Home

Description: Welcome to Phinlin's dynamic home page. Navigate effortlessly with the left-side menu, connect with users through the central feed, and enjoy personalized user options, all while staying updated with notifications and user profiles conveniently located on the right.

6.4 Profile

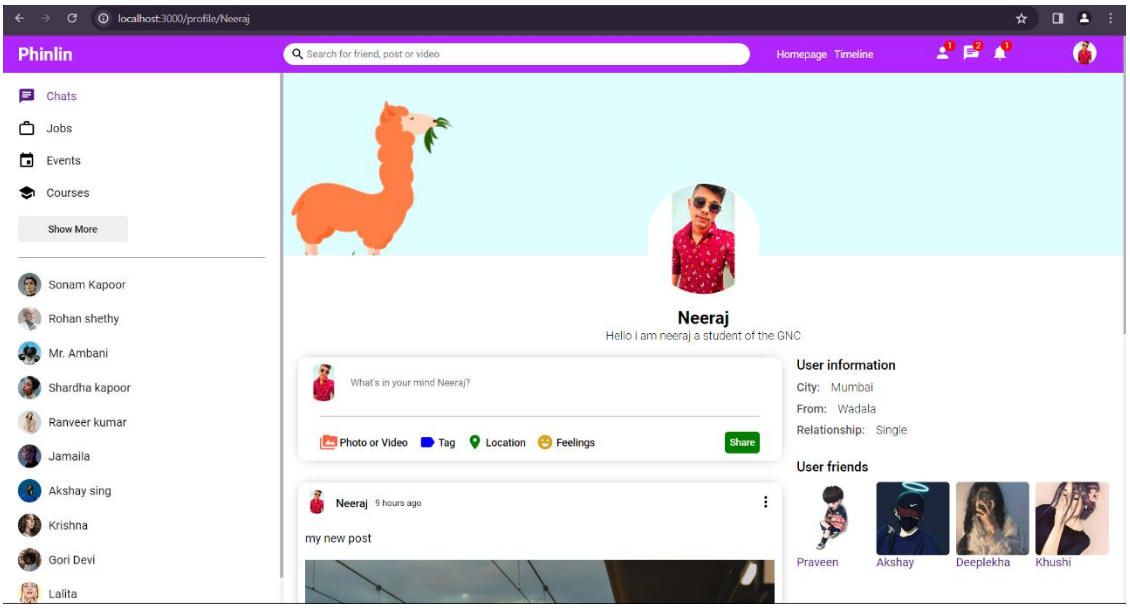


Figure 6.4: Profile

Description: Welcome to your personalized sanctuary on Phinlin's profile page. Explore your content and connections effortlessly, with your information and connections neatly organized on the right side.

6.5 Chat

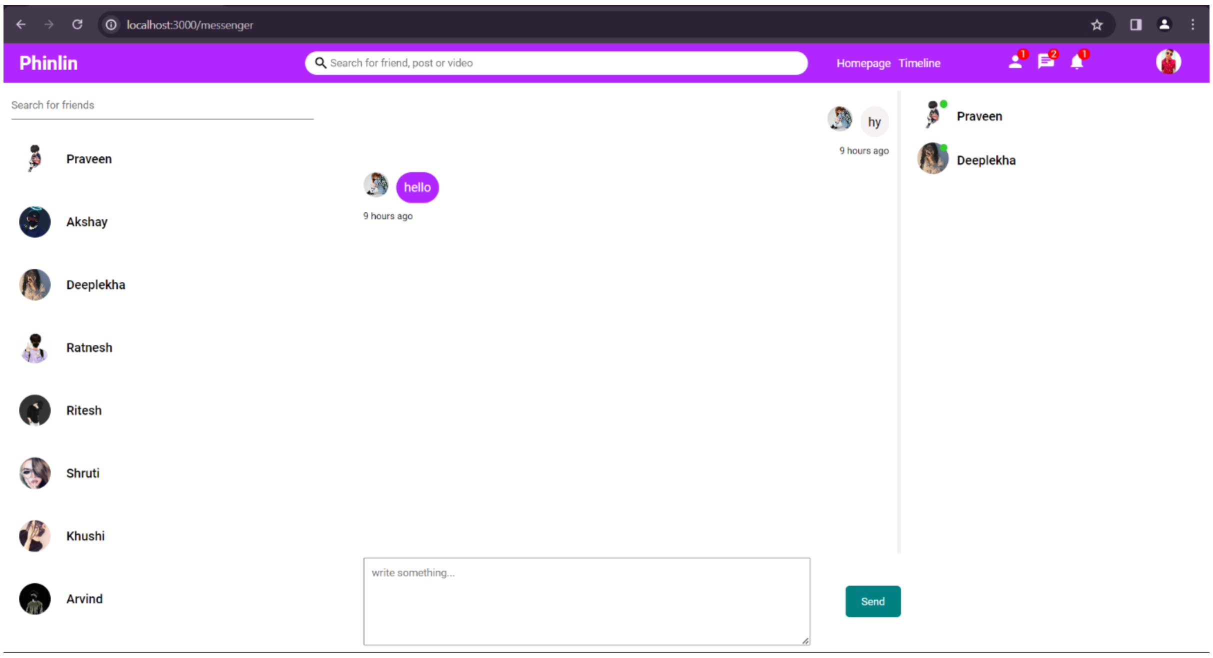


Figure 6.5: Chat

Description: Welcome to Phinlin's chat hub, where your connections are just a click away on the right side. Engage in lively conversations effortlessly, with online users conveniently listed on the left side, ensuring seamless interaction and vibrant community engagement.

6.6 Post

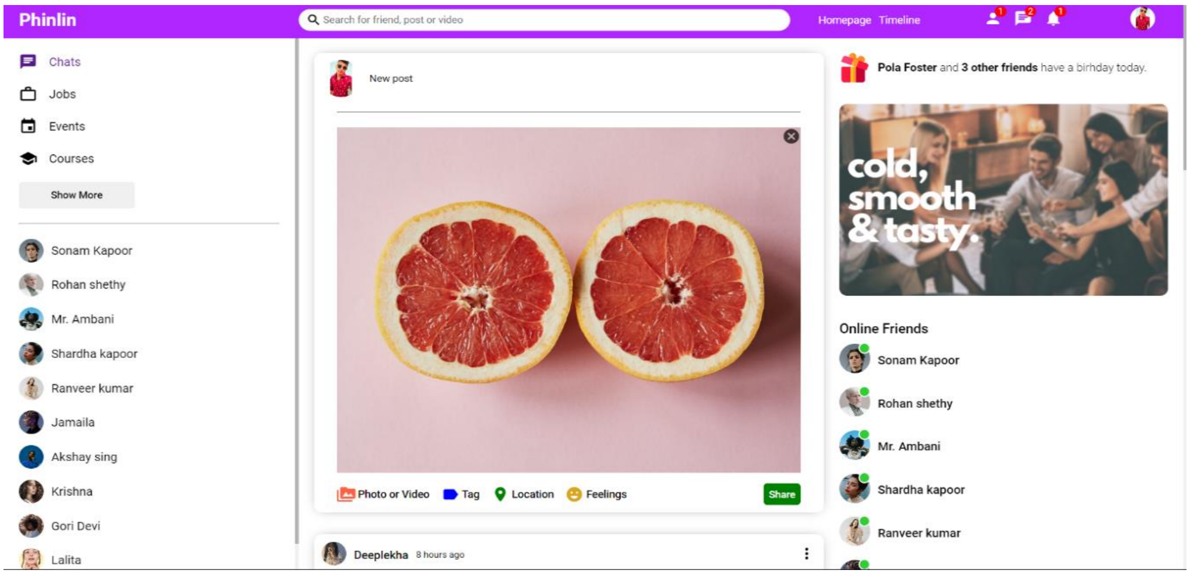


Figure 6.6: Post

Description: Easily share your thoughts, moments, and updates on Phinlin from both the home and profile pages. Whether it's a quick status update or a photo album showcasing your adventures, posting is intuitive and accessible, ensuring your content reaches your audience effortlessly.

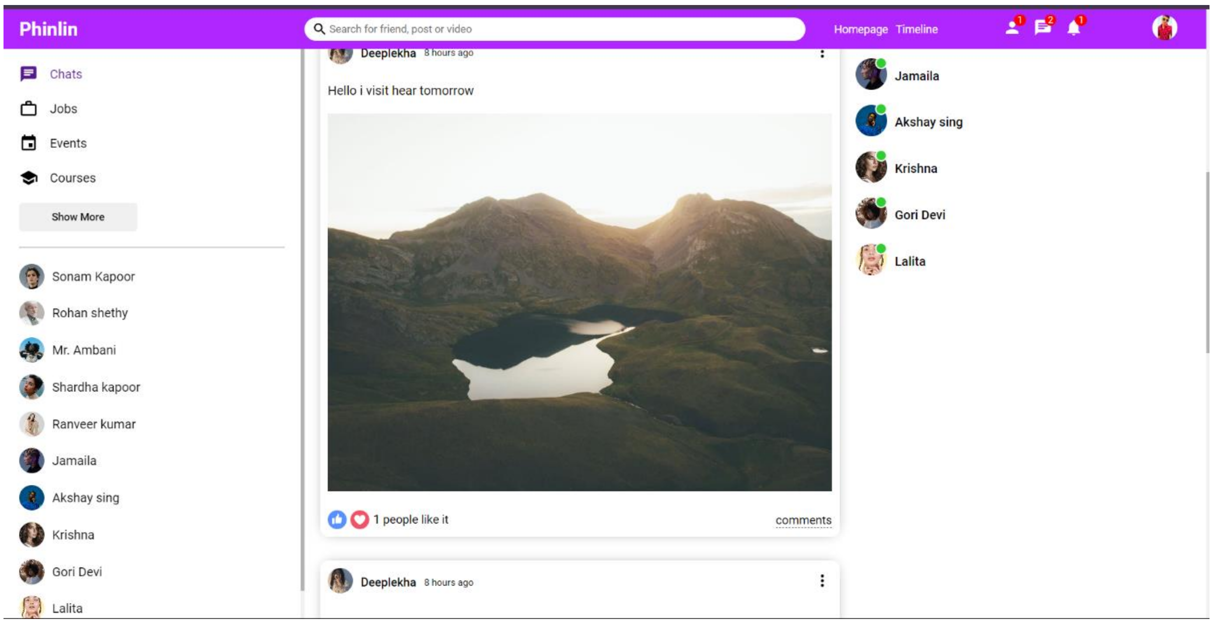


Figure 6.6: Post-2

Description: Express your appreciation and support on Phinlin by liking posts from both the home and profile pages. Whether it's a friend's update or your own memorable moment, liking is a simple yet meaningful gesture that fosters connection and engagement across the platform.

6.7 Friend Post

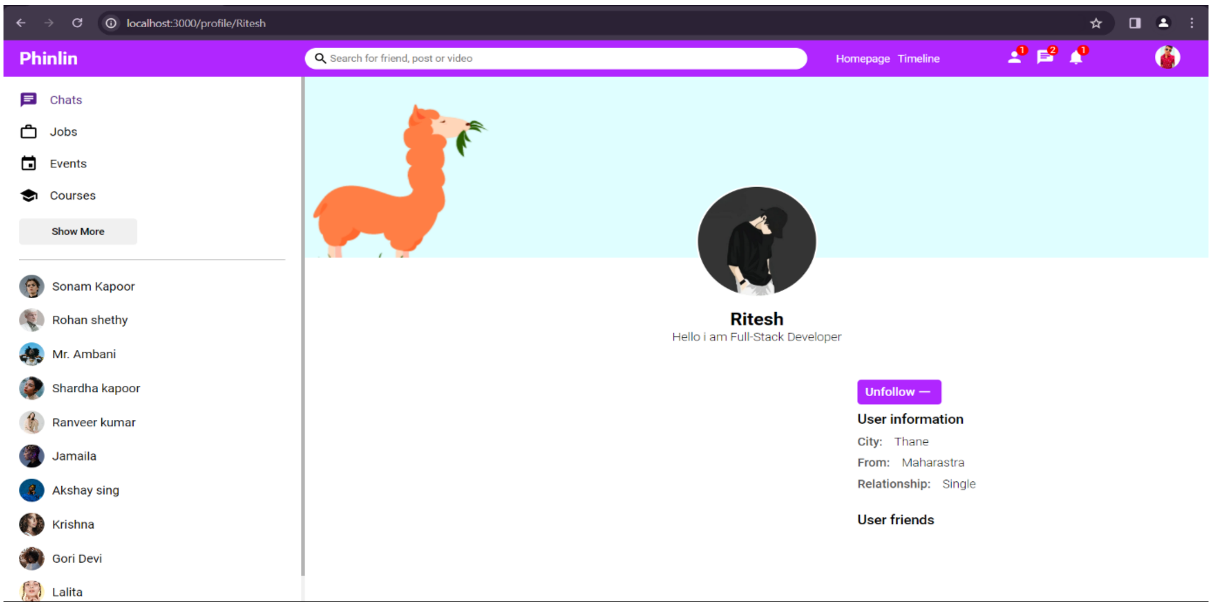


Figure 6.7: Friends Post

Description: On Phinlin's user profile pages, you can effortlessly follow or unfollow other users to tailor your feed to your interests. Stay updated with the latest content from those you follow, enriching your experience with diverse perspectives and meaningful connections.

Chapter 7

Conclusion and Future Work

7.1 Conclusion:

In conclusion, PHINLIN stands as a beacon of innovation and connectivity in the ever-evolving landscape of information technology. As a pioneering hub, it not only provides a platform for users to gather, share, and exchange the latest IT novation’s but also fosters a vibrant community where collaboration thrives. With its user-centric approach, PHINLIN empowers individuals to contribute their insights, connect with like-minded professionals, and stay at the forefront of technological advancements. By embracing the spirit of innovation and collaboration, PHINLIN continues to pave the way for a brighter future in the realm of IT, shaping the landscape of tomorrow, today. Join us in this exciting journey of discovery, connection, and growth at PHINLIN - your gateway to the forefront of IT innovation and networking.

7.2 Future Work:

In our ongoing pursuit to enhance the PHINLIN experience, we envision a plethora of exciting future developments aimed at enriching our users' interactions and expanding the scope of our platform. Here's a glimpse into what lies ahead:

1. Dedicated Pages for Courses: We plan to introduce specialized pages dedicated to IT courses, providing users with comprehensive information on various topics, ranging from programming languages to cutting-edge technologies.

2. Events Calendar: A dynamic events calendar will be integrated into PHINLIN, featuring workshops, seminars, webinars, and conferences related to IT innovation. Users can easily discover and participate in events tailored to their interests and professional goals.

3. Job Board: To facilitate career growth and networking opportunities, we'll introduce a job board where users can explore job openings in the IT sector, post job listings, and connect with potential employers and candidates.

4. AI Bot for Guidance: Harnessing the power of artificial intelligence, we'll introduce an AI-powered chatbot to provide personalized guidance, answer queries, and assist users in navigating the platform effectively.

5. Community Forums: Interactive community forums will be introduced, enabling users to engage in discussions, seek advice, and share insights on various IT-related topics. This collaborative space will foster a sense of belonging and facilitate knowledge exchange among members.

6. Resource Library: A curated resource library will be established, housing a wealth of educational materials, whitepapers, case studies, and research papers on emerging IT trends and technologies. Users can access these resources to stay informed and enhance their skills.

7. Expert Panels and Q&A Sessions: Periodic expert panels and Q&A sessions will be organized, featuring renowned professionals and thought leaders in the IT industry. Users can participate in these sessions to gain valuable insights and guidance from experts.

8. Hackathons and Challenges: We'll organize hackathons, coding challenges, and innovation competitions to inspire creativity and collaboration among our users. These events will provide opportunities to showcase skills, solve real-world problems, and win exciting prizes.

9. Mobile App: To ensure accessibility and convenience, we'll develop a mobile app version of PHINLIN, allowing users to stay connected and engaged on the go.

10. Localized Content and Communities: To cater to diverse audiences worldwide, we'll introduce localized content and communities, providing language-specific resources, events, and forums tailored to different regions and cultures.

These ambitious initiatives represent just a glimpse of our future plans for PHINLIN. As we continue to evolve and grow, we remain committed to empowering our users, fostering innovation, and driving positive change in the ever-expanding realm of information technology. Join us on this exciting journey towards a brighter, more connected future!

Chapter 8

References

React JS Documentation: https://legacy.reactjs.org/docs/getting-started.html

React Router DOM Documentation: https://www.npmjs.com/package/react-router-dom

Express Documentation: https://expressjs.com/en/5x/api.html

MongoDB Documentation: https://www.mongodb.com/docs/atlas/getting-started/

Node JS Documentation: https://nodejs.org/docs/latest/api/documentation.html

Socket.IO Documentation: https://socket.io/docs/v4/

NPM Documentation: https://docs.npmjs.com/

Nodemon Documentation: https://www.npmjs.com/package/nodemon

Bcrypt Documentation: https://www.npmjs.com/package/bcrypt

Mongoos Documentation: https://mongoosejs.com/docs/guide.html

Multer Documentation: https://www.npmjs.com/package/multer

Dotenv Documentation: https://www.npmjs.com/package/dotenv

Helmet Documentation: https://www.npmjs.com/package/helmet