SynapShare

A Collaborative Knowledge Sharing Platform

Chapter 1: Introduction

1.1 Introduction

In the digital era, the need for collaborative knowledge sharing platforms has become paramount for students, educators, and professionals. SynapShare is a full-stack web application designed to facilitate the sharing, discussion, and management of study notes, project ideas (nodes), and collaborative discussions. The platform integrates secure authentication, robust content management, and modern UI/UX to provide a seamless user experience.

1.2 Identification of Problem Domain

Despite the proliferation of online learning resources, there is a lack of centralized platforms where authenticated users can securely upload, discuss, and curate academic notes and ideas. Existing solutions often lack integrated discussion, voting, and admin moderation features, leading to fragmented knowledge and poor content quality. SynapShare aims to address these gaps by providing a secure, user-friendly, and feature-rich environment for academic collaboration.

Chapter 2: Literature Review

2.1 Literature Review

2.1.1 Study of Existing Note-Sharing Platforms

Platforms like Google Classroom, Notion, and Evernote provide ways to store and share notes, but often lack collaborative features such as voting, commenting, and real-time discussions. They also may not enforce strict content moderation or user authentication, leading to potential misuse.

2.1.2 Study of Collaborative Discussion Forums

Discussion forums such as Stack Overflow and Reddit offer robust discussion and voting mechanisms but are not tailored for academic note sharing. They may lack structured categorization (notes, nodes, discussions) and seamless file upload capabilities for educational content.

2.2 Limitation of Existing System

- Lack of integrated note sharing and discussion in a single platform.

- Inadequate content moderation and admin controls.
- Limited support for file uploads with type and size restrictions.
- Absence of user verification and role-based access control.
- Poor user experience due to fragmented functionalities.

Chapter 3: Rationale and Process

3.1 Objective

To develop a secure, scalable, and user-friendly platform that enables authenticated users to share, discuss, and manage academic notes and ideas, with features such as voting, commenting, admin moderation, and advanced search.

3.2 Software Model Adapted

The project adopts the Agile Software Development Model, emphasizing iterative development, regular feedback, and adaptive planning. Each module (authentication, notes, discussions, admin panel) is developed in sprints, with continuous integration and testing.

Chapter 4: System Analysis Overview

4.1 Requirement Analysis

4.1.1 Hardware Requirement

- Processor: Intel i3 or higher
- RAM: 4GB minimum
- Storage: 2GB free disk space
- Internet connectivity for cloud-based authentication and database

4.1.2 Software Requirement

- OS: Windows, macOS, or Linux
- Node.js (v16+)
- MongoDB (local or cloud)
- Firebase Project (for authentication)
- React.js (v18+)
- npm/yarn package manager

- Modern web browser (Chrome, Firefox, Edge)

4.1.3 Functional & Non-functional Requirements

Functional:

- User registration/login (email/password, Google)
- Email verification and username selection
- CRUD operations for notes, nodes, and discussions
- File upload (images, PDFs, videos)
- Voting and commenting system
- Admin moderation (delete any content)
- Search functionality

Non-functional:

- Secure authentication and authorization
- Responsive and accessible UI
- Performance and scalability
- Data integrity and backup
- Error handling and user feedback

4.2 Use-Case Diagram & Description

Actors: User, Admin

Use Cases:

- Register/Login
- Verify email & set username
- Upload/View/Edit/Delete Notes
- Create/View/Edit/Delete Discussions
- Create/View/Edit/Delete Nodes
- Comment/Vote on content
- Search content
- Admin: Moderate content, manage users

(Diagram to be created in UML tool; see descriptions above)

4.3 Sequence Diagram

- User Registration: User → Frontend → Backend (Firebase Auth) → DB
- Note Upload: User → Frontend → Backend (API) → DB/File Storage
- Voting: User → Frontend → Backend (API) → DB (update vote count)
- Admin Delete: Admin → Frontend → Backend (API) → DB (remove content)

(Diagrams to be created in UML tool)

4.4 System Flow Diagram

1. User interacts with frontend (React)
2. Frontend sends requests to backend (Express API)
3. Backend authenticates via Firebase, performs DB operations (MongoDB)
4. Files stored on server, metadata in DB
5. Admin actions go through additional role-check middleware

Chapter 5: System Design Overview

5.1 Data Dictionary

Entity	Attribute	Type	Description
User	uid, username, email, isEmailVerified	String, Boolean	User details and verification
Note	title, fileUrl, uploadedBy, subject, upvotes, downvotes, voters, comments	String, Number, Array	Academic notes
Discussion	title, content, fileUrl, postedBy, upvotes, downvotes, voters, comments	String, Number, Array	Topic discussions
Node	title, description, codeSnippet, fileUrl, postedBy, upvotes, downvotes, voters, comments	String, Number, Array	Project ideas
SavedPost	userEmail, postType, postId, createdAt	String, ObjectId, Date	Saved content links

5.2 Class Diagram

- User, Note, Discussion, Node, SavedPost as main classes/entities.
- Relationships: User has many Notes, Discussions, Nodes, SavedPosts.

(Diagram to be created in UML tool)

5.3 Data Flow Diagram

- Level 0: User → Web App → Server → DB/File Storage
- Level 1: Details for authentication, CRUD, voting, admin moderation.

(DFD diagrams to be created in diagramming tool)

5.4 Extended E-R Diagram

- Entities: User, Note, Discussion, Node, SavedPost
- Relationships: One-to-many (User → Notes, Discussions, Nodes, SavedPosts)

Chapter 6: Work Plan and System Database Structure

6.1 Time Frame Work

Phase	Duration
Requirement Analysis	1 week
Design	1 week
Implementation	2 weeks
Testing	1 week
Deployment	1 week

6.2 Design Database Table

User Table:

- uid (PK), username, email, isEmailVerified, createdAt

Note Table:

- _id (PK), title, fileUrl, uploadedBy, subject, upvotes, downvotes, voters, comments, createdAt

Discussion Table:

- _id (PK), title, content, fileUrl, postedBy, upvotes, downvotes, voters, comments, createdAt

Node Table:

- _id (PK), title, description, codeSnippet, fileUrl, postedBy, upvotes, downvotes, voters, comments, createdAt

SavedPost Table:

- _id (PK), userEmail, postType, postId, createdAt

Chapter 7: Implementation & Testing

7.1 Testing Strategy Adapted

- Unit Testing: For backend API endpoints and frontend components.
- Integration Testing: End-to-end flows (e.g., login, note upload).
- Manual Testing: UI/UX, error handling, and edge cases.

7.2 System Testing

- Authentication and authorization tested for all roles.
- File uploads validated for type and size restrictions.
- CRUD operations tested for all entities.
- Voting and commenting tested for correctness.
- Admin moderation tested for all content types.

7.3 Test Cases

Test Case	Input	Expected Output
User Registration	Valid email/password	Success, email verification
Note Upload	Valid file, title, subject	Note created, file stored
Voting	Upvote/downvote action	Vote count updated
Admin Delete Note	Admin action on note	Note deleted
Search	Query term	Relevant results returned
Invalid File Upload	Oversized/invalid file	Error message
Unauthorized Access	No/invalid token	Access denied

Chapter 8: Conclusion and Future Extension

8.1 Conclusion

SynapShare successfully addresses the need for a secure, collaborative, and feature-rich academic content sharing platform. By integrating robust authentication, content management, and moderation features, it enhances the quality and reliability of shared knowledge.

8.2 Future Scope

- Real-time chat and notifications
- AI-based content recommendation
- Mobile app version
- Advanced analytics for admin
- Integration with third-party educational APIs