SynapShare

A Collaborative Knowledge Sharing Platform

Chapter 1: Introduction

1.1 Introduction

The exponential growth of digital education and remote learning has underscored the need for robust platforms that facilitate collaborative knowledge sharing among students, educators, and professionals. SynapShare is designed as a comprehensive web-based platform that bridges the gap between isolated note-taking tools and interactive discussion forums. It enables users to upload, share, discuss, and curate academic notes, project ideas (nodes), and collaborative discussions in a secure, user-friendly environment. By incorporating modern authentication, voting, commenting, and moderation features, SynapShare aims to foster a vibrant academic community where knowledge is both accessible and reliable.

1.2 Identification of Problem Domain

Traditional note-sharing solutions, such as Google Drive or Evernote, primarily focus on storage and personal organization, lacking the collaborative and interactive features essential for academic growth. Discussion forums like Stack Overflow or Reddit, while excellent for Q&A and debates, are not tailored for structured academic content sharing and often lack file upload capabilities and content moderation. The absence of a unified platform for authenticated sharing, discussion, and curation of academic content leads to fragmented knowledge, duplication of effort, and potential dissemination of unreliable information. SynapShare addresses these issues by integrating secure authentication, content moderation, and a rich set of features tailored for educational collaboration.

Chapter 2: Literature Review

2.1 Literature Review

2.1.1 Study of Existing Note-Sharing Platforms

Google Classroom, OneNote, Notion, and Evernote are popular tools for note-taking and sharing. While they provide basic sharing functionalities, they lack advanced features such as collaborative discussions, voting, and real-time moderation. Moreover, these platforms often do not enforce strict user verification, leading to potential misuse and content quality issues.

Key Observations:

- Limited to storage and sharing; minimal collaboration.
- No integrated discussion or voting mechanisms.
- Weak content moderation and user verification.

2.1.2 Study of Collaborative Discussion Forums

Stack Overflow, Reddit, and Quora are leading platforms for collaborative discussions and Q&A. They offer voting, commenting, and moderation but are not tailored for academic note sharing and structured content. File uploads are restricted, and there is a lack of categorization for educational resources.

Key Observations:

- Robust discussion and voting features.
- Not designed for structured note sharing or academic content.
- Limited file upload support and content categorization.

2.2 Limitation of Existing System

- **Fragmented Functionality:** No single platform combines note sharing, discussion, and moderation.
- **Inadequate Moderation:** Existing systems lack robust admin controls and content verification.
- **Limited Collaboration:** Absence of features like voting, commenting, and project idea sharing.
- **Security Concerns:** Weak authentication and authorization mechanisms.
- **Poor User Experience:** Non-intuitive interfaces and lack of responsive design.

Chapter 3: Rationale and Process

3.1 Objective

The primary objective of SynapShare is to develop a secure, scalable, and user-friendly platform that enables authenticated users to:

- Share academic notes and project ideas.
- Engage in structured discussions.
- Vote and comment on content to ensure quality.
- Allow administrators to moderate and manage content.
- Support advanced search and categorization for efficient information retrieval.

3.2 Software Model Adapted

Agile Software Development Model

- **Iterative Development:** Features are developed in sprints, allowing for continuous feedback and improvement.
- **Modularization:** The system is divided into modules (authentication, notes, discussions, admin panel) for independent development and testing.
- **Continuous Integration:** Regular code integration and testing ensure that new features do not break existing functionality.
- **User Feedback:** Early and frequent user feedback guides feature prioritization and UI/UX improvements.
- **Documentation:** Each sprint ends with documentation updates for code, APIs, and user guides.

Chapter 4: System Analysis Overview

4.1 Requirement Analysis

4.1.1 Hardware Requirement

- Processor: Intel i3 or higher (or equivalent)
- RAM: Minimum 4GB (8GB recommended for development)
- Storage: At least 2GB free disk space for code, database, and uploads
- Network: Stable internet connection for cloud authentication and database access

4.1.2 Software Requirement

- Operating System: Windows, macOS, or Linux
- Backend: Node.js (v16+), Express.js, MongoDB (local/cloud), Firebase Admin SDK
- Frontend: React.js (v18+), npm/yarn, Tailwind CSS, Axios, React Router
- Other Tools: VS Code or any modern IDE, Git, Postman for API testing, Browser (Chrome/Firefox/Edge)

4.1.3 Functional & Non-functional Requirements

Functional Requirements:

- User registration and login (email/password, Google)
- Email verification and username selection

- CRUD operations for notes, nodes, and discussions
- File upload with validation (images, PDFs, videos)
- Voting and commenting on content
- Admin moderation (delete any content)
- Search functionality across all content types

Non-functional Requirements:

- Secure authentication and authorization
- Responsive, accessible, and user-friendly UI
- High performance and scalability
- Data integrity and regular backups
- Robust error handling and user feedback
- Compliance with privacy and security standards

4.2 Use-Case Diagram & Description

Actors:

- **User:** Can register, log in, upload/view/edit/delete notes, discussions, nodes, comment, vote, and search.
- **Admin:** Has all user privileges plus the ability to moderate (delete) any content.

Use Cases:

- **Register/Login:** Secure user authentication via email/password or Google.
- **Verify Email/Set Username:** Ensures unique, verified users.
- **Upload/View/Edit/Delete Notes:** Manage academic notes with file uploads.
- **Create/View/Edit/Delete Discussions:** Topic-based discussions.
- **Create/View/Edit/Delete Nodes:** Share and manage project ideas.
- **Comment/Vote:** Engage with content to improve quality.
- **Search:** Find content efficiently.
- **Admin Moderation:** Remove inappropriate or low-quality content.

(A UML Use-Case Diagram should be created to visually represent these interactions.)

4.3 Sequence Diagram

Example: Note Upload

1. User fills note upload form and submits.
2. Frontend validates input and sends a POST request to the backend API.
3. Backend verifies authentication via Firebase, processes file upload, and saves note metadata to MongoDB.
4. Backend returns success/failure response.
5. Frontend updates UI and informs the user.

(Similar sequence diagrams should be created for registration, voting, commenting, and admin actions.)

4.4 System Flow Diagram

1. User interacts with the React frontend.
2. Frontend communicates with the Express backend via REST APIs.
3. Backend authenticates requests using Firebase tokens.
4. Backend performs CRUD operations on MongoDB and manages file uploads.
5. Admin actions go through additional role verification.

(A system flow diagram should be included for clarity.)

Chapter 5: System Design Overview

5.1 Data Dictionary

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

5.2 Class Diagram

- **User:** Attributes and methods for authentication, profile management.
- **Note, Discussion, Node:** Inherit common properties (title, content, fileUrl, postedBy, upvotes, downvotes, voters, comments).
- **SavedPost:** Links users to saved notes, discussions, or nodes.
- **Relationships:** One-to-many (User → Notes, Discussions, Nodes, SavedPosts).

(A UML Class Diagram should be created to illustrate these relationships.)

5.3 Data Flow Diagram

- **Level 0:** User → Web App → Server → Database/File Storage
- **Level 1:** Authentication, CRUD operations, voting, commenting, admin moderation.

(DFD diagrams should be created using diagramming tools.)

5.4 Extended E-R Diagram

- **Entities:** User, Note, Discussion, Node, SavedPost
- **Relationships:** User has many Notes, Discussions, Nodes, SavedPosts; content entities have comments and votes.

(An Extended E-R Diagram should be included.)

Chapter 6: Work Plan and System Database Structure

6.1 Time Frame Work

Phase	Duration	Activities
Requirement Analysis	1 week	Stakeholder meetings, requirement gathering
Design	1 week	UI/UX design, database schema, API contracts
Implementation	2 weeks	Frontend and backend development, integration
Testing	1 week	Unit, integration, and system testing
Deployment	1 week	Deployment, documentation, user training

6.2 Design Database Table

User Table:

- **uid:** String (Primary Key), unique user ID from Firebase
- **username:** String, unique username
- **email:** String, user email address
- **isEmailVerified:** Boolean
- **createdAt:** Date

Note Table:

- **_id:** ObjectId (Primary Key)
- **title:** String
- **fileUrl:** String (path to uploaded file)
- **uploadedBy:** String (username)
- **subject:** String
- **upvotes:** Number
- **downvotes:** Number
- **voters:** Array of { username, voteType }
- **comments:** Array of { content, postedBy, createdAt }
- **createdAt:** Date

Discussion Table:

- Similar to Note, with content instead of subject

Node Table:

- Similar to Note, with description and codeSnippet

SavedPost Table:

- **_id:** ObjectId (Primary Key)
- **userEmail:** String
- **postType:** Enum ("note", "discussion", "node")
- **postId:** ObjectId (reference)
- **createdAt:** Date

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