SMART STUDENT HANDBOOK

Title	Name	Surname	Student No
Mr	Takudzwa	Magunda	u22599160
Mr	Reinhard	Pretorius	u22509578
Mr	Mpumelelo	Njamela	u23534932
Mr	Tanaka	Ndhlovu	u22610792
Mr	Junior	Motsepe	u22598473

Software Requirements Specification (SRS)

1 Introduction

Our vision is to create an advanced application that makes it easy for students to create, share, and find notes. The Smart Student application is designed to enhance digital note-taking and academic collaboration for students. It enables users to create and manage notebooks, which can be organized into folders for easy access. Each notebook can be divided into sections, and sections further split into individual pages, offering a structured and personalized way to capture and categorize notes.

Users can write notes from a text editor, unlocking a wide range of formatting options such as headings, bullet lists, tables, images, and other multimedia elements. To improve productivity, the application features a context-aware AI Smart Assist toolbar that appears when a notebook is opened. This assistant suggests relevant notebooks based on content similarity, allowing users to merge, duplicate, or clone them into their workspace seamlessly. The application can also change text to speech for one to listen in when they are doing something that needs their attention.

In addition, users can provide feedback by liking notebooks or providing comments on them, helping highlight high-quality content. They can also add events on the app calendar to keep track of their schedule.

The Figma prototype below illustrates our initial concept for the Smart Student Handbook interface. While still in early development and subject to change, it provides a clear foundation and preview of the intended final product.

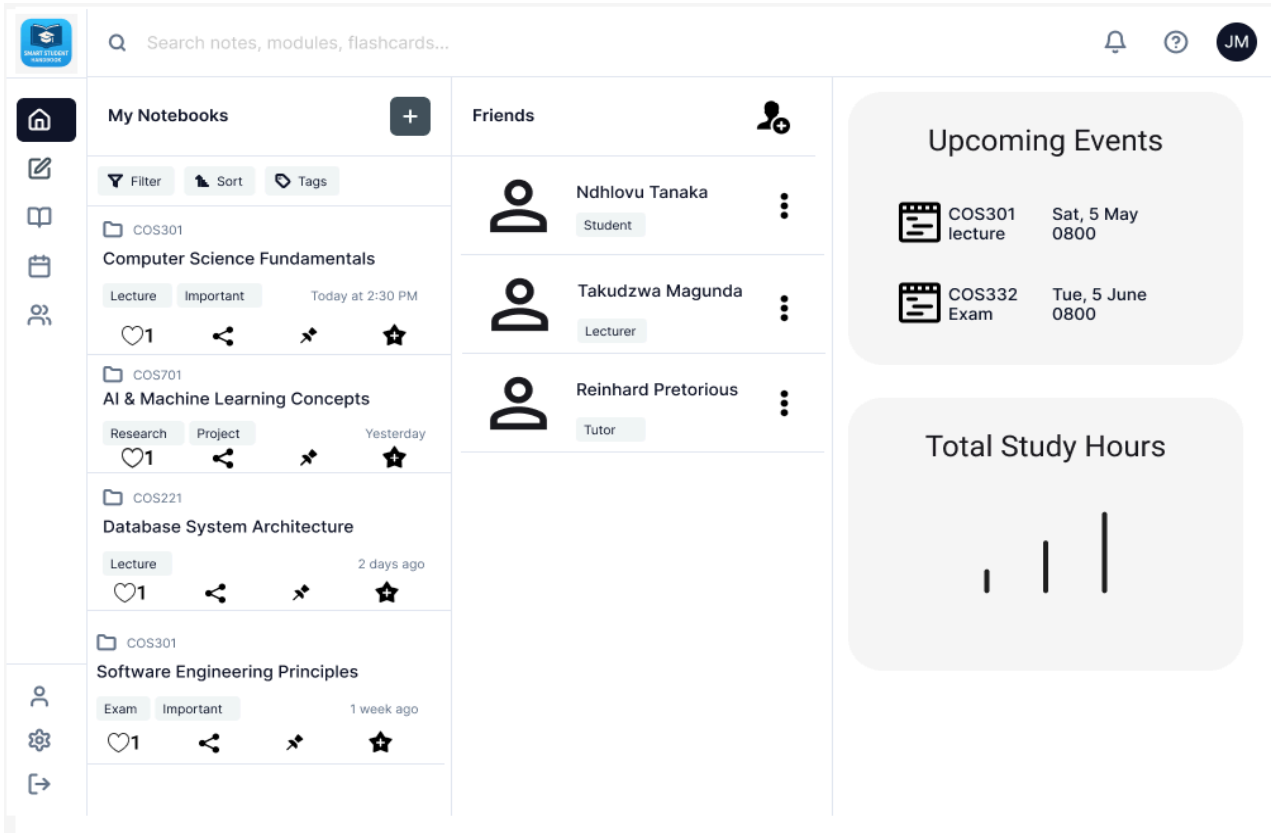


Figure 1: Envisioned Smart Student Handbook User Interface Prototype

2 User Characteristics

The primary users of the Smart Student Handbook are university students who rely on their gadgets(laptops, cellphones, tablets) as their main tool for academic engagement. These users must have an internet connection at first to sign up, but can then use the application in offline mode later.

2.1 USER

- A person who can operate a computer or smartphone
- A person who wants to create notebooks in order to take and organize notes
- A person who wants to collaborate with other students on the same notebook
- A person who wants to improve/make their notes by using other students' notes
- A person who wants to share their notes with other users
- A person who wants to keep track of their times on the app calendar
- A person who wants to create study groups with their friends

3 User Stories

3.1 USER STORY 1

As a user, I want to be able to sign up for an account and securely log in so that I can access and manage my notebooks and other features.

3.2 USER STORY 2

As a user, I want to be able to view and edit my profile details, such as my username and profile picture, so I can personalize my account to reflect my identity.

3.3 USER STORY 3

As a user, I want to be able to create notebooks and type my notes in a text formatter so I can style my notes how I want.

3.4 USER STORY 4

As a user, I want to organize my notebooks into folders (e.g., by year or module) so I can easily find them without scrolling endlessly.

3.5 USER STORY 5

As a user, I want to be able to take notes even when I am offline, so that I can continue studying during load shedding or without data, and have my notes sync when I'm back online.

3.6 USER STORY 6

As a user, I want the application to suggest relevant notebooks or snippets based on what I’m typing, so I can easily access useful and related content.

3.7 USER STORY 7

As a user, I want to rate notebooks or specific snippets and view ratings from others, so I can prioritize high-quality and trusted content.

3.8 USER STORY 8

As a user, I want to collaborate on notebooks with my friends by sharing or co-editing, so we can divide work and improve our notes together.

3.9 USER STORY 9

As a user, I want to search for notebooks related to my topics of interest, so I can explore diverse perspectives and information.

3.10 USER STORY 10

As a user, I want to use a calendar to view upcoming events, add lectures, and schedule study sessions, so I can manage my academic time effectively.

3.11 USER STORY 11

As a user, I want to add friends, view their shared notebooks, and collaborate with them, so I can learn and grow in a community environment.

3.12 USER STORY 12

As a user, I want to track the number of hours I spend studying or working, so I can monitor my productivity and improve time management.

4 USE CASES

1.

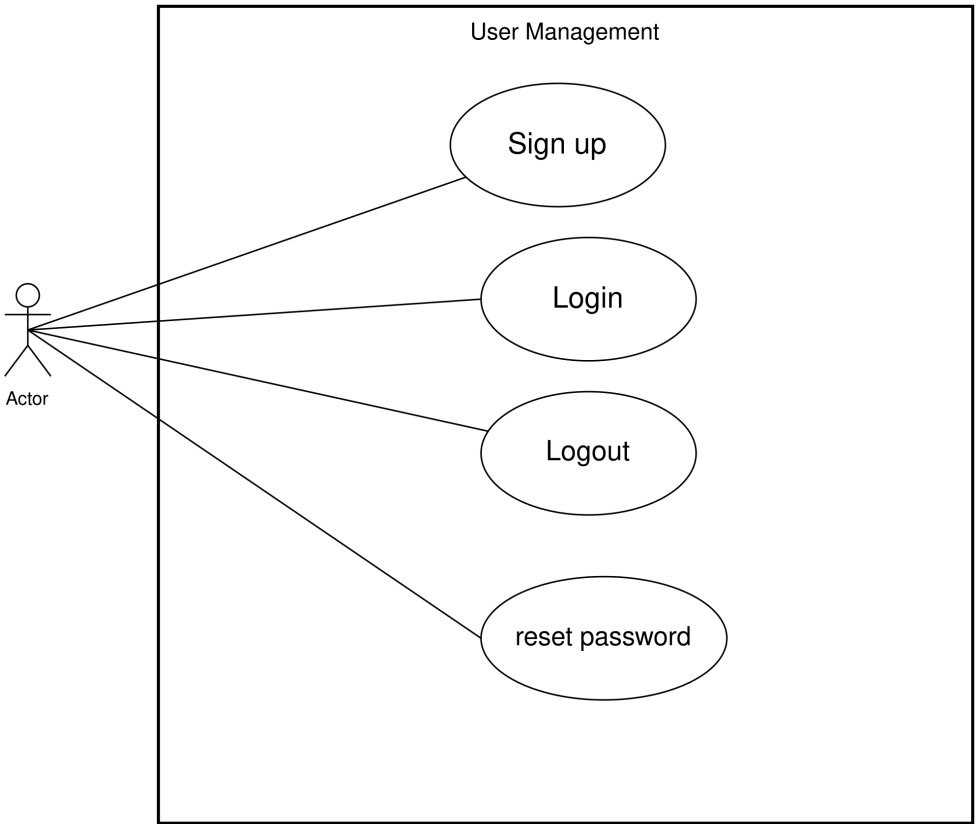


Figure 2: User Management Use Case Diagram

User Management Service Contracts:

Sign Up

- What It Does: Registers a new user in the system.
- Inputs: Username, email, password, role (e.g., "Actor").
- Outputs: Success message or error (e.g., "Email already exists").
- Interacts with: Database (to store user details)

Login

- What It Does: Authenticates a user and grants access to the system.
- Inputs: Email/username, password.
- Outputs: Authentication token or error (e.g., "Invalid credentials").
- Interacts With: Database (to verify credentials).’

Logout

- What It Does: Terminates the user's active session.
- Inputs: Authentication token.
- Outputs: Success message.
- Interacts with: Session management service.

Reset Password

- What It Does: Allows users to reset their password via email.
- Inputs: Email address.
- Outputs: Password reset link (sent to email) or error.
- Interacts with: Email service, database.

2.

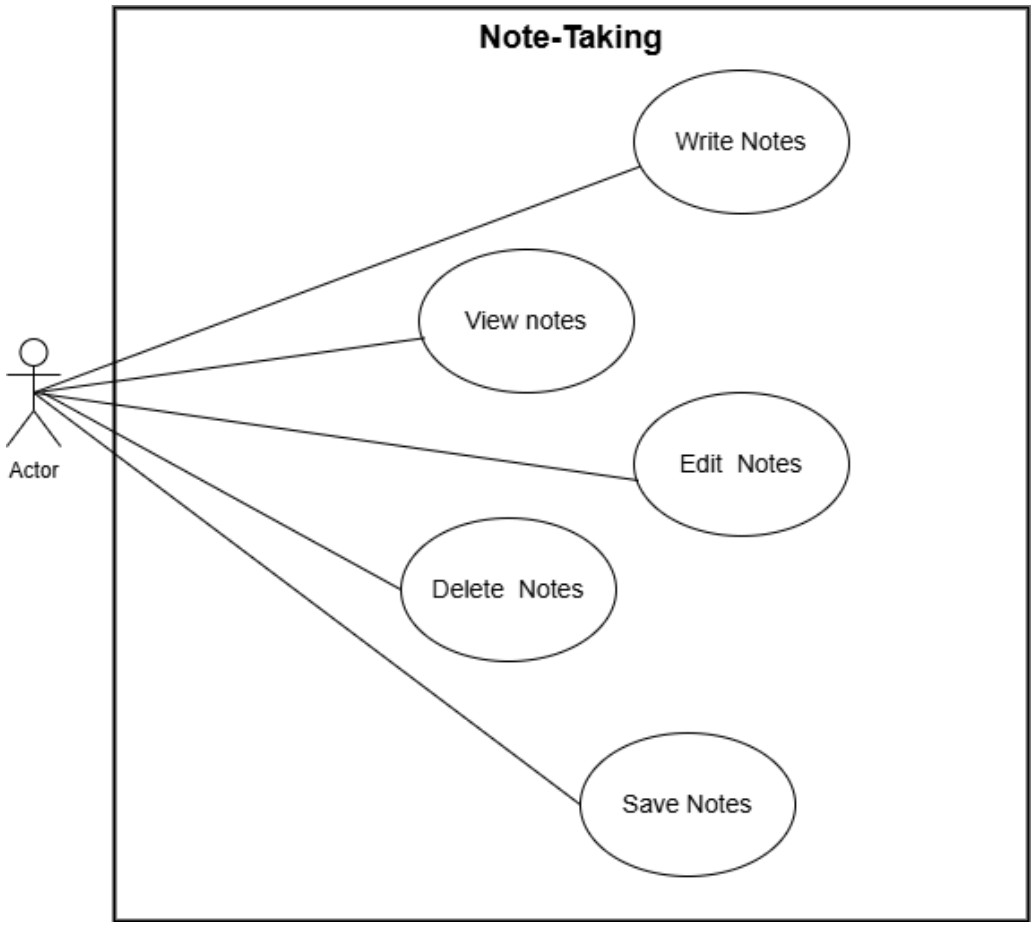


Figure 3: Note-Taking Use Case Diagram

Note-Taking Service Contracts

Write Notes

- What It Does: Creates a new note.
- Inputs: User ID, note title, content.
- Outputs: Note ID or error.
- Interacts with: Database (to save notes).

View Notes

- What It Does: Retrieves notes for a user.
- Inputs: User ID, optional filters (e.g., date).
- Outputs: List of notes or error.
- Interacts with: Database (to fetch notes).

Edit Notes

- What It Does: Updates an existing note.
- Inputs: Note ID, updated content.
- Outputs: Success message or error.
- Interacts With: Database.
-

Delete Notes

- What It Does: Removes a note permanently.
- Inputs: Note ID.
- Outputs: Success message or error.
- Interacts With: Database.

Save Notes

- What It Does: Auto-saves note changes.
- Inputs: Note ID, partial/full content.
- Outputs: Success message or error.
- Interacts With: Database.

3.

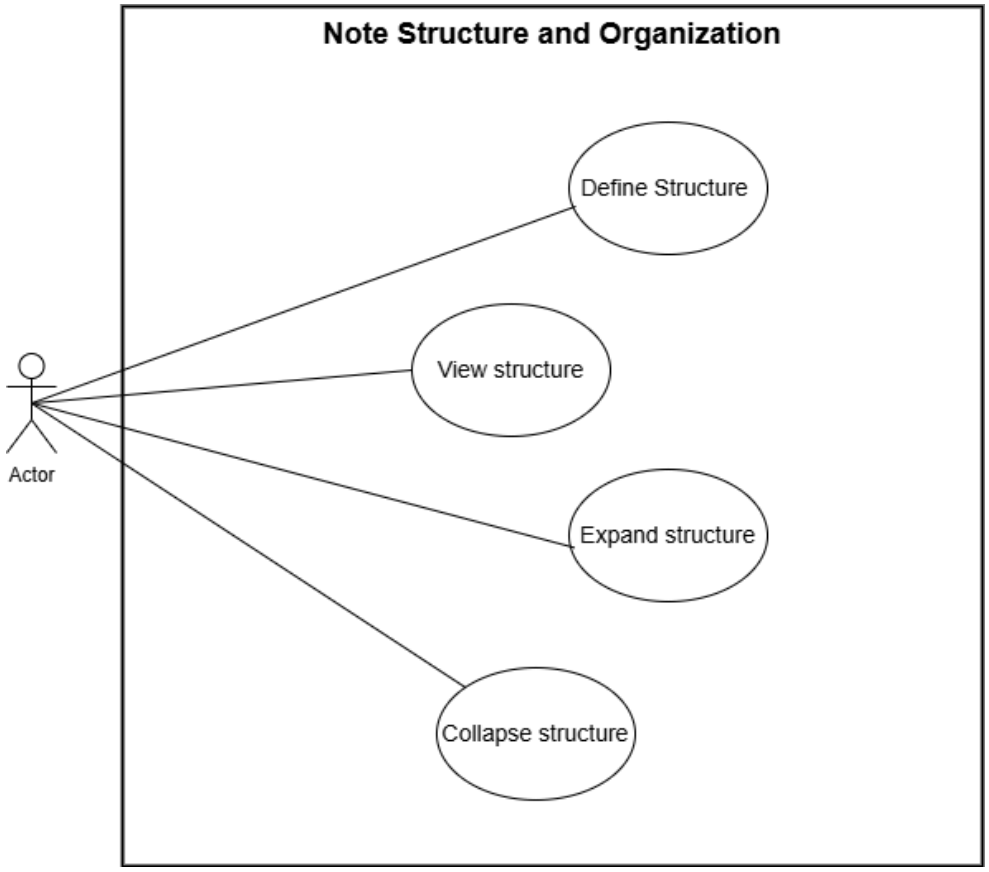


Figure 4: Note Structure and Organisation Use Case Diagram

Note Structure and Organisation Service Contracts

Define Structure

- What It Does: Creates a hierarchical structure for notes (e.g., folders).
- Inputs: User ID, structure name, parent ID (optional).
- Outputs: Structure ID or error.
- Interacts With: Database.

View Structure

- What It Does: Displays the note hierarchy.
- Inputs: User ID.
- Outputs: Tree-like structure of notes or error.
- Interacts With: Database.

Expand/Collapse Structure

- What It Does: Toggles visibility of substructures.
- Inputs: Structure ID, action ("expand" or "collapse").
- Outputs: Updated UI state.
- Interacts With: Frontend components.

4.

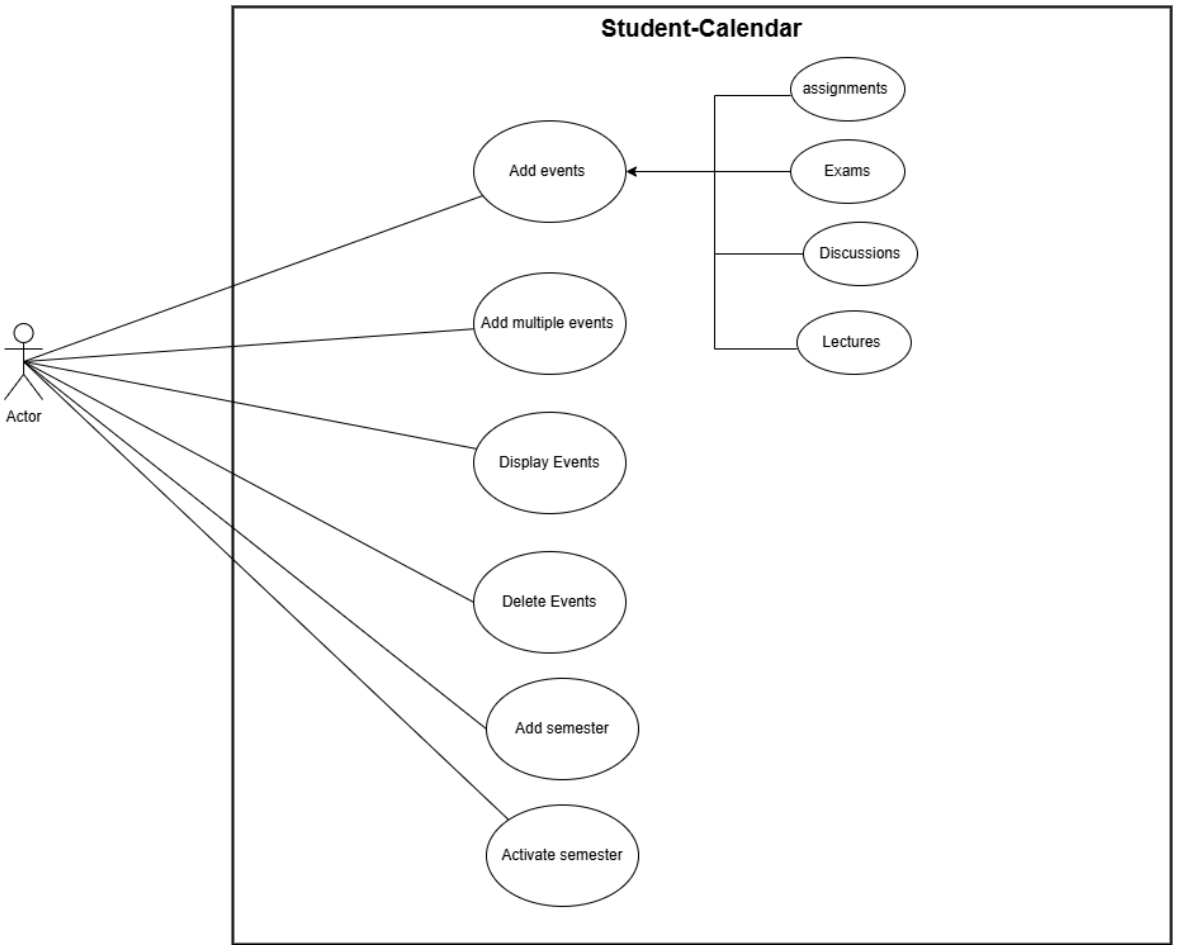


Figure 5: Student-Calendar Use Case Diagram

Student-Calendar Service Contracts

Add Events

- What It Does: Schedules a new event (e.g., lecture).
- Inputs: User ID, event title, date/time, type (e.g., "Exam").
- Outputs: Event ID or error.
- Interacts With: Database.

Display Events

- What It Does: Shows events for a given timeframe.
- Inputs: User ID, date range.
- Outputs: List of events or error.
- Interacts With: Database.

Delete Events

- What It Does: Removes an event.
- Inputs: Event ID.
- Outputs: Success message or error.
- Interacts With: Database.

Add/Advise Semester

- What It Does: Creates or updates semester plans.
- Inputs: User ID, semester name, courses.
- Outputs: Semester ID or error.
- Interacts With: Database.

Figure 6: Profile-Management Use Case Diagram

Profile-Management Service Contracts

View/Edit Details

- What It Does: Displays or updates user profile data.
- Inputs: User ID, updated fields (e.g., bio).
- Outputs: Profile data or error.
- Interacts With: Database.

View Study Hours/Badges

- What It Does: Shows analytics or achievements.
- Inputs: User ID.
- Outputs: Study hours/badges list or error.
- Interacts With: Analytics service.

View/Remove Friends

- What It Does: Manages friend connections.
- Inputs: User ID, friend ID (for removal).
- Outputs: Friend list or success message.
- Interacts With: Social graph service.

Change Password

- What It Does: Updates user password.
- Inputs: User ID, old password, new password.
- Outputs: Success message or error.
- Interacts With: Database.

4 FUNCTIONAL REQUIREMENTS:

- **R1:** The system should allow users to create accounts and sign in
- **R2:** The system should allow users to manage their Profile
  - **R2.1** The system should allow the user to update their personal information(name, surname, profile picture)
  - **R2.2** The system should allow a user to change their password
- **R3:** The system should allow users to create notebooks and notes
  - **R3.1:** The system should allow a user to create a notebook
  - **R3.2:** The system should allow a user to create folders inside a notebook
  - **R3.3:** The system should allow a user to create notes inside folders
- **R4:** The system should allow a user to share notes
- **R5:** The system should allow a user to search and see shared notes
  - **R5.1:** The system should allow a user to clone shared notes
  - **R5.2:** The system should allow a user to like shared notes
- **R6:** The system should allow a user to collaborate with friends on a notebook
- **R7:** The system should allow a user to make and view notes offline
- **R8:** The system should allow a user to access the app calendar
  - **R8.1:** The system should allow the user to see all upcoming events on the calendar
  - **R8.2** The system should allow the user to add an event to the calendar

Architectural Requirements

Quality Requirements

Q1 Performance

1.1 Routine notebook actions (create, read, update, delete) should complete in under one second, even under heavy load.

1.2 Synchronizing data across devices must finish within five seconds by batching updates efficiently.

1.3 More complex database queries must return results in under ten seconds by leveraging optimized, indexed queries.

Q2 Scalability

2.1 The platform must handle a minimum of 55,000 concurrent active users, each can have multiple notebooks, and the system should not slow down.

2.2 It should support seamless scaling both by upgrading server resources and by adding additional instances to meet demand spikes

2.3 Auto-scaling mechanisms must dynamically adjust capacity as the number of users increases or decreases.

Q3 Storage

3.1 Our system must handle ever-increasing volumes of notes and attachments without large data costs or hurting performance.

3.2 Before saving, all user-uploaded images and large files will be automatically compressed to reduce storage use and speed up access.

3.4 We reserve at least **10 to 15 %** of our total database space to be able to absorb unexpected spikes in data.

Q4 Availability

Q4.1 Annual downtime must stay under 2.2 hours and should not increase by more than 8.5% each year

Q4.2 During planned maintenance, at least 80 % of services must continue running so students can still access their notes.

Q5 Reliability

5.1 The application should run stably without unhandled crashes; any error must trigger a captured report with context, allowing us to quickly fix the error.

5.2 User notebooks must be protected from data loss—automatic version backups and rollback procedures should aid in this.

5.3 Service deployments must eliminate single points of failure through redundancy.

Q6 Security

6.1 Access controls must ensure that only authenticated and authorised users can read or write a given notebook unless given access through collaboration.

6.2 User data should be encrypted to ensure that users' anonymity is kept.

6.3 Mechanisms to reset all user passwords if a breach is required.

Architectural Patterns

Layered Architecture:

- **Presentation Layer:** React/Next.js frontend for user interaction and Shadcn for structured components.
- **Application Layer:** Spring Boot/NestJS backend for business logic
- **Data Layer:** Firebase for dynamic structure of notes
- **AI Layer:** Lightweight LLM integration for smart assist features

Microservices Pattern:

- Authentication Service: User management and security
- Note Management Service: Core note operations
- AI Processing Service: Machine learning and recommendations
- Collaboration Service: Real-time sharing and editing
- Search Service: Advanced search and filtering capabilities

Constraints

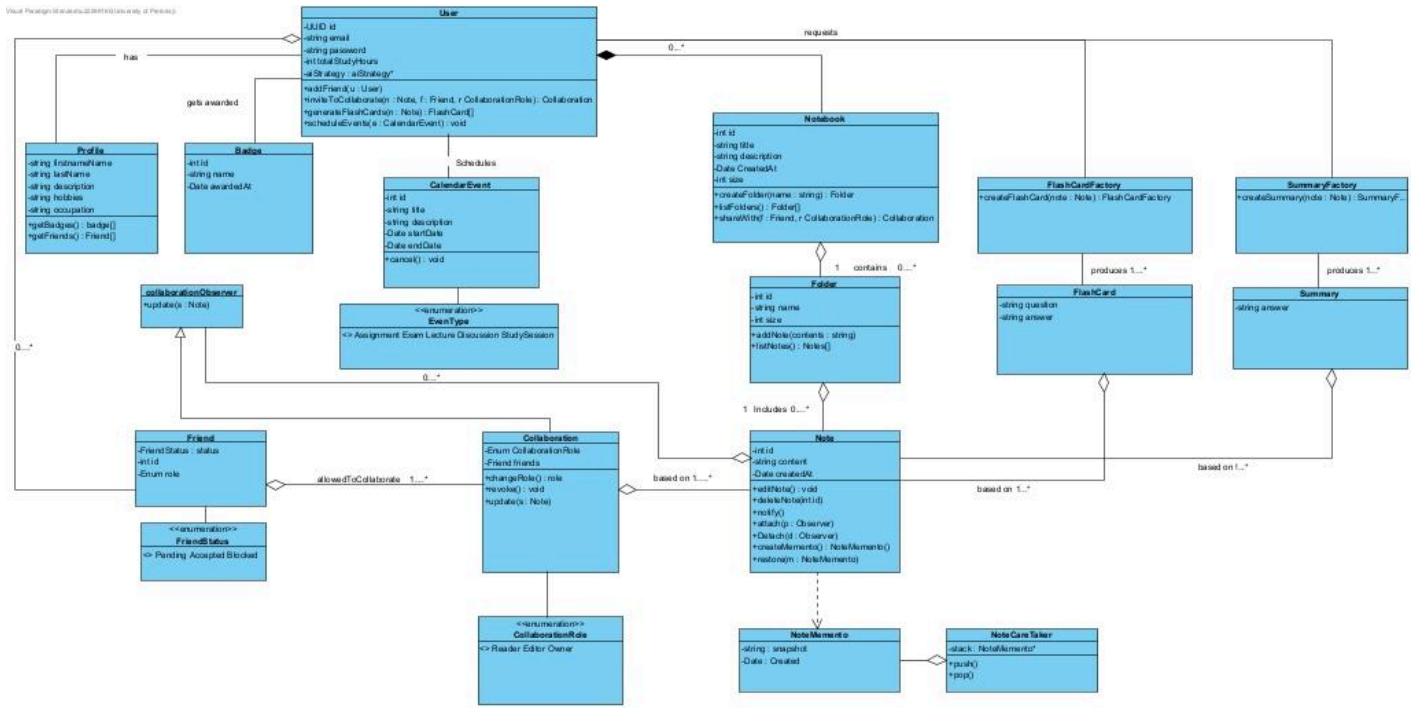
Technical Constraints:

- Must integrate with existing university authentication systems
- AI processing must remain within budget constraints using free-tier services
- Mobile responsiveness is required for cross-device compatibility
- Offline functionality is limited to read-only operations initially

Business Constraints:

- Development timeline: 12 weeks (academic semester)
- Budget limitations: Utilize free-tier cloud services where possible
- Team expertise: Student-level experience requiring learning curve consideration
- Regulatory compliance: Must adhere to educational data privacy requirements

Domain Model



DomainModel.jpg

Design Patterns

1.Composite (nested notebooks and folders)
Notebook, Folder, and Note all implement a common Component interface so operations such as size calculation, sharing or exporting work exactly the same whether a student selects a single note or an entire notebook. The whole structure forms one uniform tree, eliminating type checks and duplicate logic.

2.Observer (real-time collaboration updates)
A Note plays the role of subject. When the note changes, it calls notify(), which immediately pushes updates to collaborators without any hard-wired dependencies inside the editing code.

3.Memento (safe undo and redo)
Before a note is modified, it captures its current state in a NoteMemento object and pushes that snapshot onto a stack managed by a caretaker. Later, restore(memento) rolls the content back while still keeping the internals of Note encapsulated. This gives users unlimited, trustworthy undo and redo.

4.Factory Method(generating study aids)
A shared ContentFactory interface hides the concrete creation of study aids like FlashCard or Summary. When we want to introduce new study aids such as mind-maps, we simply provide another factory class. The user interface and service layers continue to work unchanged.

5.State
The Friend class stores its current FriendStatus, which can be Pending, Accepted, or Blocked. Behaviour changes internally based on that status.

Technology Requirements

Frontend Technologies

- **Framework:** Next.js with React for server-side rendering and SEO optimization
- **Language:** TypeScript for type safety and maintainable code
- **Styling:** TailwindCSS for rapid, responsive UI development
- **Component Library**: Shadcn for prebuilt, customizable UI components
- **State Management:** Redux Toolkit for complex state management

Backend Technologies

- **Framework:** Spring Boot (Java) or NestJS (TypeScript) for robust API development
- **API Design:** RESTful API with OpenAPI documentation
- **Authentication:** JWT tokens with a refresh token mechanism
- **Real-time:** WebSocket for collaborative features

Database Technologies

- **Primary Database:** Firebase for dynamic structured notes
- **File Storage:** Firebase Storage

AI/ML Technologies

- **ML Framework:** Hugging Face Transformers for pre-trained models
- **Model:** DistilBERT for lightweight NLP processing
- **Processing:** Python-based AI service for recommendation generation
- **Integration:** REST API for AI service communication

Cloud Infrastructure

- **Platform:** AWS for scalable cloud infrastructure
- **Compute:** EC2 instances with auto-scaling capabilities
- **Serverless:** Lambda functions for event-driven processing
- **CDN:** CloudFront for global content delivery

Development Tools

- **IDE:** Visual Studio Code with relevant extensions
- **Containerization:** Docker for consistent development environments
- **Version Control:** Git with GitHub for code management
- **CI/CD:** GitHub Actions for automated testing and deployment
- **Testing:** Jest for unit testing, Cypress for end-to-end testing
- **Design:** Figma for UI/UX design and prototyping

Monitoring and Analytics

- **Application Monitoring:** AWS CloudWatch for system metrics
- **Error Tracking:** Sentry for error monitoring and debugging
- **Analytics:** Custom analytics dashboard for user engagement tracking
- **Performance:** Lighthouse for web performance optimization