Web-Based Student Note Sharing Platform

Nisa Begüm Burucu (64210025)

Utku Öztekin (64220012)

Elif Serra Uçar (64220058)

Deniz Kenan Ek (64210044)

Kerem Döleksöz (64210022)

Supervisor: Sibel Tarıyan Özyer

(Ankara Medipol University-Computer Engineering Department)

1. Introduction

1.1. Purpose

The purpose of this **Software Requirements Specification (SRS)** document is to clearly define **what** the 'Web-Based Student Note Sharing Platform' will do and **what criteria** it will operate under.

This document serves as the technical blueprint for the project; it lists all functional (what the system will do) and non-functional (such as performance and security) requirements. It is the primary reference source for the project team and the project supervisor, defining the project's scope and success criteria.

1.2. Scope

The platform to be developed will allow students from **any university and department** to upload, rate, comment on, and access lecture notes and related study materials (images, summaries, etc.).

The system will include three distinct user roles (Student, Teacher, Admin) and dedicated panels for these roles. Admins will be responsible for content approval and managing the **general academic hierarchy (University, Faculty, Department, Course)**. Teachers will manage their own course materials, and Students will be responsible for content consumption and contribution.

Out of Scope: This project will *not* include live tutoring, video conferencing, instant messaging, or assignment tracking functionalities.

1.3. Definitions and Acronyms

To ensure the requirements in this document (Section 3) are clearly understood, the following terms and roles are specifically defined:

- **SRS:** Software Requirements Specification.
- **Platform:** The "Web-Based Student Note Sharing Platform" referred to in this document.
- **User:** Any person using the platform (Student, Teacher, or Admin).
- **Admin (Administrator):** A user with full system privileges. Responsible for content approval, user management, and general course catalog management.

- **Teacher:** Academic staff or a verified instructor. An authorized user who can add "approved material" to their courses and moderate comments.
- **Student:** The standard end-user of the platform. Can upload notes (pending approval), download notes, rate, and comment.
- **REST API:** The architecture (Representational State Transfer Application Programming Interface) that facilitates data exchange between the Backend (Java Spring Boot) and Frontend (React.js).

2. Overall Description

2.1. Product Perspective

The platform will be developed as a **standalone**, public web application. It will not be tied to any specific institution; instead, it will offer a dynamic structure that allows users to create their **own university, faculty, and course structures** (with Admin approval). It will consist of a REST API served by the Backend (Java Spring Boot) and a Frontend (React.js) client that consumes this API.

2.2. User Classes and Characteristics

- **Admin (Administrator):**
    - Expected to have technical knowledge.
    - Responsible for the system's overall operation.
    - Key tasks: Approving/rejecting student notes, managing user roles, managing/approving requests for new Universities, Faculties, Departments, and Courses, and deleting inappropriate comments sitewide.
- **Teacher (Academic/Instructor):**
    - Registers on the platform and applies to the Admin for the "Teacher" role, which is then verified.
    - Key tasks: Uploading "Official/Approved Course Material" to registered courses (these materials bypass the approval process), reviewing student notes in their courses, and managing (deleting/editing) comments on those notes.
- **Student:**
    - The main user of the platform. Can register with any email address.
    - Key tasks: Registering, searching for course notes (in the University -> Faculty -> Department -> Course hierarchy), downloading notes, uploading notes (pending Admin approval), rating notes, and commenting.

2.3. Operating Environment

- **Backend:** Will run on any cloud server (e.g., AWS, Azure, Google Cloud) or VPS that supports Java and Spring Boot.
- **Database:** A MySQL or PostgreSQL relational database server is required.
- **Frontend:** The platform will be accessible from any desktop or mobile device with a modern web browser (Google Chrome, Firefox, Safari, Edge).

2.4. Design and Implementation Constraints

- **Technology:** The backend must be developed using Java Spring Boot (including Spring Security, JPA/Hibernate).
- **Technology:** The frontend must be developed using React.js (including HTML, CSS, JavaScript, and Bootstrap/Tailwind CSS).
- **Architecture:** The system must follow a RESTful service-based architecture.
- **Performance:** The system must use efficient database queries and caching mechanisms to handle high concurrent usage.
- **Legal:** A clear usage policy and a "report/takedown" mechanism must be in place to manage the copyright compliance of uploaded content.

3. Specific Requirements

This section details all the functions the system will perform.

3.1. External Interface Requirements

- **3.1.1. User Interface (UI):**
  - The interface will have a modern (Bootstrap/Tailwind) and aesthetic design, aiming to be "visually advanced."
  - The design will be responsive, compatible with mobile and tablet devices.
  - There will be separate, clear, and purpose-driven dashboards for the three user roles (Student, Teacher, Admin).
  - Thumbnail preview support will be available for uploaded images (JPG, PNG).
- **3.1.2. Software Interfaces:**
  - All communication between the Frontend (React.js) and Backend (Spring Boot) will be handled via a **REST API** transmitting data in JSON format over standard HTTP/HTTPS protocols.
  - The API will use a Spring Security-based (e.g., JWT) structure for user authentication.

3.2. Functional Requirements

FR-1: Authentication and User Management

- **FR-1.1:** Users (Students/Teachers) must be able to register on the system with **any valid email**, password, first name, and last name.
- **FR-1.2:** Users must be able to verify their accounts via a link sent to their email address upon registration.
- **FR-1.3:** Registered users must be able to log in to the system with their email and password.
- **FR-1.4:** Users must be able to reset their password using a "Forgot Password" function.
- **FR-1.5:** Users must be directed to different interfaces and permissions based on their roles (Admin, Teacher, Student) (Authorization).

FR-2: Admin Panel Functions

- **FR-2.1 (Content Moderation):** The Admin must be able to list notes uploaded by students that are "Pending Approval."
- **FR-2.2 (Content Moderation):** The Admin must be able to review a pending note and either "Approve" (publish) or "Reject" (notifying the user with a reason).
- **FR-2.3 (Academic Catalog Management):** The Admin must be able to dynamically add, remove, or edit new **Universities**, **Faculties**, **Departments**, and **Courses** in the system.
- **FR-2.4 (User Management):** The Admin must be able to list all users, change their roles (e.g., Student -> Teacher), or suspend their accounts.
- **FR-2.5 (Comment Management):** The Admin must be able to view all comments made on the platform and delete any found to be inappropriate.

FR-3: Teacher Panel Functions

- **FR-3.1 (Authorization):** A user must be able to request the "Teacher" role from the Admin (if necessary, by uploading verification documents).
- **FR-3.2 (Material Upload):** A Teacher must be able to upload "Official Course Material" (PDF, DOCX, PPTX, etc.) to the courses they are responsible for. These uploads *must not* go through the Admin approval process.
- **FR-3.3 (Course Management):** A Teacher must be able to list all notes related to their courses (both their own uploads and student uploads).
- **FR-3.4 (Comment Management):** A Teacher must be able to moderate comments made on their course notes and delete any found to be inappropriate.
- **FR-3.5 (Statistics):** A Teacher must be able to view download and rating statistics for their own materials.

FR-4: Student Panel and Note Functions

- **FR-4.1 (Note Upload):** A Student must be able to upload a note using the upload form (selecting the relevant University, Faculty, Department, Course, Title, Description, File Upload). Uploaded notes will be set to "Pending Approval."
- **FR-4.2 (New Catalog Request):** If the University/Department/Course a student is looking for does not exist, they must be able to request its addition from the Admin.
- **FR-4.3 (Supported Formats):** The system must support PDF, DOCX, TXT, and image (JPG, PNG) formats for note uploads.
- **FR-4.4 (Note Download):** A Student must be able to search for, filter, and download approved notes to their computer.
- **FR-4.5 (My Profile):** A Student must be able to see a list of their own uploaded notes and their approval status (Pending/Approved/Rejected).

FR-5: Interaction Functions (Rating and Comments)

- **FR-5.1 (Rating):** Users (Students/Teachers) must be able to rate a note on a scale of 1 (poor) to 5 (excellent).

- **FR-5.2 (Rating Display):** The arithmetic average of the ratings received by a note must be displayed next to it.
- **FR-5.3 (Commenting):** Users must be able to add text-based comments under notes.
- **FR-5.4 (Comment Deletion):** Users must only be able to delete their own comments. (Admins and relevant Teachers can delete all).

FR-6: Search and Filtering

- **FR-6.1 (Search):** Users must be able to search by note title, course name, course code, or university name.
- **FR-6.2 (Sorting):** Search results and course pages must be sortable by criteria such as "By Date (Newest)," "By Rating (Highest)," and "Most Downloaded."
- **FR-6.3 (Hierarchical Navigation):** Users must be able to navigate hierarchically from the Home Page by selecting University -> Faculty -> Department -> Course to find notes.

3.3. Non-Functional Requirements

- **3.3.1. Performance:**
    - The platform's main page and course listing pages must load in under 3 seconds.
    - The system must be able to handle high concurrent usage (e.g., 100 active users simultaneously) without slowing down.
    - Database queries must be optimized, especially for search and filtering.
- **3.3.2. Security:**
    - User passwords must be stored in the database in an irreversible hashed format (e.g., bcrypt).
    - All API endpoints must be secured with Spring Security; unauthorized access (e.g., a Student trying to access the Admin panel) must be blocked.
    - File upload functions must only accept whitelisted file types to prevent malicious attacks (e.g., script execution) and must store files in a secure location.
    - The system must be protected against basic web attacks such as SQL Injection and XSS.
- **3.3.3. Usability:**
    - The platform must be intuitive to use; a new user should be able to perform basic functions (searching, downloading) without needing a guide.
    - The visual design, aiming for the "advanced" goal, must be orderly, professional, and suitable for an academic context.
    - All error messages (e.g., "Incorrect password") must be displayed to the user in clear and understandable language.

- **3.3.4. Reliability:**
    - The system's uptime must be 99.9%.
    - User-uploaded files (notes, images) must be protected against loss and backed up regularly.
- **3.3.5. Maintainability:**
    - The Backend (Spring Boot) code must be modular, well-commented, and written according to "SOLID" principles.
    - The Frontend (React.js) code must be structured using reusable components.

4.References

"IEEE Recommended Practice for Software Requirements Specifications" (IEEE Std 830-1998)

"IEEE Recommended Practice for Software Design Descriptions" (IEEE Std 1016-2009)

"Software Engineering, 10th Edition" (Ian Sommerville)

"Spring Boot Official Documentation"

"Spring Security Official Documentation"

"JPA/Hibernate Official Documentation"

"Java SE Documentation"

"PostgreSQL 16 Documentation"

"MySQL 8.0 Reference Manual"

"React Official Documentation"

"MDN Web Docs"

"Tailwind CSS Official Documentation"

"Bootstrap 5 Documentation"

"REST API design best practices" (Microsoft Azure Documentation)

"Design Patterns: Elements of Reusable Object-Oriented Software" (Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides)