NeuNotes: Academic Notes Sharing Platform

Overview

NeuNotes is a comprehensive academic notes sharing platform designed to facilitate knowledge exchange among students and faculty members. The application allows users to share, discover, and organize academic content within an intuitive and collaborative environment.

Table of Contents

Features

Core Features

- **User Authentication**: Secure login and registration system
- **Role-Based Access Control**: Three user roles with different permissions:
    - **Admin**: Full system access and user management
    - **Faculty**: Course creation and content management
    - **Student**: Enrollment in courses and note sharing
- **Course Organization**: Hierarchical structure with categories and subcategories
- **Notes Upload**: Support for PDF and image files
- **Preview Capabilities**: Direct viewing of uploaded content

- **Search Functionality**: Find notes by various criteria

Advanced Features

- **Favorites System**: Bookmark notes for quick access
- **Rating System**: Rate notes on a 5-star scale with average calculations
- **Tagging System**: Categorize notes with custom tags
- **Comments**: Discuss notes with other users
- **Analytics Dashboard**: View statistics about note engagement
- **Activity Logging**: Track user actions throughout the system
- **Enrollment Management**: Students can join courses to access content

System Architecture

NeuNotes follows a client-server architecture with:

- **Frontend**: React with Vite for the user interface
- **Backend**: Node.js and Express for the application server
- **Database**: MySQL for data persistence
- **API Layer**: RESTful endpoints for communication between client and server
- **Authentication**: Session-based authentication for security

Database Schema

The database consists of **12 tables**, exceeding the required 9 tables for a 2-person team. All tables are normalized to 3NF:

User Management

- **users**: Stores user credentials and profile information
    - Primary user entity with role-based access control
    - Supports different academic statuses (Undergraduate/Graduate)

Content Organization

- **categories**: Represents courses/subjects
    - Each category has an auto-generated unique code
    - Includes metadata like creation time and creator
- **subcategories**: Represents sections within courses
    - Hierarchical relationship with categories
    - Enables granular organization of content

User Engagement

- **enrollments**: Tracks student enrollment in courses
    - Many-to-many relationship between students and categories
    - Contains enrollment timestamps
- **uploads**: Stores uploaded files
    - Central table for note management
    - Contains metadata and statistical counters
- **file_metadata**: Contains details about uploaded files
    - Separates file metadata from upload management
    - Stores file names, types, sizes, and URLs

Social Features

- **comments**: Stores user comments on uploads
    - Enables discussions about academic content
    - Timestamps for chronological ordering
- **activities**: Logs user actions throughout the system
    - Comprehensive activity tracking for auditing
    - Records actions with timestamps
- **favorites**: Tracks users' favorite notes
    - Implements bookmarking functionality
    - Enables personalized content collections

Content Classification

- **tags**: Stores available tags for categorization
    - Flexible content classification system
    - User-generated taxonomy
- **note_tags**: Junction table connecting notes to tags
    - Many-to-many relationship implementation
    - Enables multi-dimensional categorization

Quality Assessment

- **ratings**: Stores user ratings for notes
    - Five-star rating system
    - Tracks individual ratings and updates aggregates

Database Programming Objects

Functions

- **generate_category_code**: Automatically generates unique 6-character category codes
    - Combines first 3 letters of name with 3 random digits

    - Ensures uniqueness and readability

Stored Procedures

- **create_category**: Handles category creation with automated code generation
    - Centralizes business logic for category creation
    - Performs activity logging
    - Returns generated ID and code
- **add_or_update_rating**: Manages rating creation and updates
    - Validates rating value (1-5 range)
    - Handles both new ratings and updates to existing ones
    - Logs user activity
    - Uses output parameters to indicate if rating is new
- **add_comment**: Manages comment creation with validation
    - Validates comment content
    - Creates comment record
    - Logs activity
    - Returns the new comment ID

Triggers

- **after_subcategory_insert/after_subcategory_delete**: Maintain subcategory counts
    - Automatically updates category's subcategory count
    - Ensures data consistency without application code
- **after_rating_insert/update/delete**: Manage rating statistics
    - Automatically calculates and updates average ratings
    - Maintains accurate rating counts
    - Ensures data consistency across the application
- **after_comment_insert/delete**: Track comment counts
    - Automatically updates comment counts on uploads
    - Ensures accurate comment statistics

Installation

Prerequisites

- Node.js (v16.0 or higher)

- MySQL (v8.0 or higher)
- npm or yarn

Setup Steps

Database Setup

The project includes a complete SQL file (schema.sql) with all **12 tables**, stored procedures, functions, and triggers already defined:

1. **Use the MySQL Data Import Feature**:
    - Open MySQL Workbench or your preferred MySQL client
    - Connect to your MySQL server
    - Create a new database named "neunotes"
    - Select the "neunotes" database
    - Go to "Server" > "Data Import" (in MySQL Workbench)
    - Select "Import from Self-Contained File" and browse to the provided `neunotes_schema.sql` file
    - Click "Start Import"
2. **Verify Installation**:
    - Ensure all 12 tables are created
    - Verify stored procedures, functions and triggers are properly installed

Backend Setup

1. **Clone the repository:**

```
git clone https://github.com/your-username/neunotes.git
cd neunotes
```

2. **Install backend dependencies:**

```
cd backend
npm install
```

3. **Configure environment variables:** Create a `.env` file in the backend directory:

```
PORT=9090
DB_HOST=localhost
DB_PORT=3306
DB_USER=root
DB_PASSWORD=your_mysql_password
DB_NAME=neunotes
SESSION_SECRET=your_secret_key
FRONTEND_URL=http://localhost:5173
```

4. **Start the backend server:**

```
npm run dev
```

Frontend Setup

1. **Install frontend dependencies:**

```
cd ../frontend
npm install
```

2. **Configure environment variables:** Create a `.env` file in the frontend directory:

```
VITE_API_URL=http://localhost:9090/api
```

3. **Start the frontend development server:**

```
npm run dev
```

4. **Access the application:** Open your browser and navigate to `http://localhost:5173`

Usage Guide

User Registration and Login

1. Click "Sign Up" to create a new account
2. Fill in your details (name, email, password, academic status)
3. Log in with your credentials

For Students

1. Browse available courses on the home page

2. Enroll in courses you're interested in
3. View course content by clicking on a course card
4. Upload notes to subcategories within enrolled courses
5. Rate, comment on, and favorite notes
6. Add tags to notes for better organization
7. View your favorites, uploads, and activity in your profile

For Faculty

1. Create new courses (categories) using the "Create Category" button
2. Create subcategories within your courses
3. Upload notes to any subcategory
4. Manage the content you've created

For Admins

1. Access the Admin Dashboard to manage users
2. Create, edit, and delete any content
3. View system-wide activity logs
4. Change user roles as needed

API Documentation

Authentication

- `POST /api/auth/signup`: Register a new user
- `POST /api/auth/login`: Log in a user
- `POST /api/auth/logout`: Log out the current user
- `GET /api/auth/me`: Get the current user's information

Categories

- `GET /api/category`: Get all categories
- `GET /api/category/:id`: Get a specific category
- `POST /api/category`: Create a new category
- `PUT /api/category/:id`: Update a category
- `DELETE /api/category/:id`: Delete a category

Subcategories

- `GET /api/subcategory/by-category/:categoryId`: Get subcategories for a category
- `POST /api/subcategory`: Create a new subcategory
- `DELETE /api/subcategory/:id`: Delete a subcategory

Uploads

- `GET /api/upload/by-subcategory/:subcategoryId`: Get uploads for a subcategory
- `GET /api/upload/:id`: Get upload details
- `GET /api/upload/my-uploads`: Get current user's uploads
- `POST /api/upload`: Upload a new note
- `DELETE /api/upload/:id`: Delete an upload

Comments

- `GET /api/comment/by-upload/:uploadId`: Get comments for an upload
- `POST /api/comment`: Add a comment
- `DELETE /api/comment/:id`: Delete a comment

Favorites

- `GET /api/favorites/my-favorites`: Get user's favorites
- `GET /api/favorites/check/:uploadId`: Check if user has favorited an upload
- `POST /api/favorites`: Add a favorite
- `DELETE /api/favorites/:id`: Remove a favorite

Ratings

- `GET /api/ratings/user/:uploadId`: Get user's rating for an upload
- `GET /api/ratings/average/:uploadId`: Get average rating for an upload
- `POST /api/ratings`: Rate an upload
- `DELETE /api/ratings/:id`: Delete a rating

Tags

- `GET /api/tags`: Get all tags
- `GET /api/tags/by-upload/:uploadId`: Get tags for an upload
- `POST /api/tags`: Create a new tag
- `POST /api/tags/note-tag`: Add a tag to a note
- `DELETE /api/tags/note-tag/:id`: Remove a tag from a note

Technology Stack

Frontend

- **React**: UI component library
- **Vite**: Build tool for modern web development

- **Tailwind CSS**: Utility-first CSS framework
- **Zustand**: State management
- **Axios**: HTTP client for API requests
- **React Router**: Client-side routing

Backend

- **Node.js**: JavaScript runtime environment
- **Express**: Web application framework
- **MySQL**: Relational database management system
- **express-session**: Session-based authentication
- **bcrypt**: Password hashing (for production use)
- **multer**: File upload handling
- **cors**: Cross-Origin Resource Sharing support

Project Structure

Backend Structure

```
backend/
├── src/
│   ├── controllers/        # API logic
│   ├── middleware/         # Express middleware
│   ├── routes/             # API routes
│   ├── db.js               # Database connection
│   ├── app.js              # Express app setup
│   └── server.js           # Server entry point
├── uploads/                # Uploaded files
├── database/               # Database scripts
│   └── neunotes_schema.sql # Complete database schema
├── .env                    # Environment variables
└── package.json            # Dependencies
```

Frontend Structure

```
frontend/
├── src/
│   ├── components/         # Reusable UI components
│   ├── contexts/           # State management
│   ├── pages/              # Page components
│   ├── services/           # API services
│   ├── App.jsx             # Main component
│   └── main.jsx            # Entry point
├── .env                    # Environment variables
└── package.json            # Dependencies
```

Security Features

- **Session-Based Authentication**: Secure user sessions with HTTP-only cookies
- **Password Storage**: Hashed and secure password storage (for production)
- **Input Validation**: Server-side validation for all user inputs
- **Role-Based Access Control**: Different permissions based on user roles
- **CORS Protection**: Configured Cross-Origin Resource Sharing for API security
- **Database Constraints**: Proper foreign key constraints and data validation
- **Error Handling**: Comprehensive error management to prevent information leakage

CRUD Operations

Create

- Register new users
- Create categories and subcategories
- Upload notes
- Add comments and ratings
- Create tags
- Add favorites

Read

- View available categories
- Browse subcategories
- View uploaded notes
- Read comments
- View ratings
- See user profiles

Update

- Update user profiles
- Edit categories
- Update ratings
- Modify tags

Delete

- Delete uploaded notes
- Remove comments
- Delete categories and subcategories
- Remove ratings
- Delete tags
- Unfavorite notes

Technical Implementation Details

Authentication

The application uses session-based authentication with cookies, managed through express-session on the backend and a custom auth store (using Zustand) on the frontend.

File Storage

Files are stored on the server filesystem with metadata stored in the database. The application supports PDFs and images with built-in preview capabilities.

Database Queries

The application uses raw SQL queries (no ORM) to interact with the MySQL database, demonstrating direct SQL usage and optimization.

Error Handling

Comprehensive error handling is implemented throughout the application using try-catch blocks and a global error handler middleware.

Security Considerations

- CORS protection
- Session cookies with proper security attributes
- Input validation
- Role-based access control

- Proper error messaging to prevent information leakage

Future Improvements

Possible enhancements for the NeuNotes platform:

1. **Advanced Search**: Implement full-text search for note content
2. **Collaborative Editing**: Real-time collaborative note editing
3. **Version Control**: Track changes to notes over time
4. **Notifications**: Alert users to new notes, comments, etc.
5. **Mobile App**: Develop a mobile application for on-the-go access
6. **Analytics Dashboard**: Provide usage statistics and insights
7. **Content Recommendations**: Suggest notes based on user behavior and interests
8. **Integration with LMS**: Connect with learning management systems

Troubleshooting

Common Issues

Backend Can't Connect to MySQL

- Verify MySQL is running
- Check database credentials in .env file
- Ensure the database exists and is accessible
- Make sure atabase: process.env.DB_NAME || 'yourDBname' is changed in backend/src/db.js

CORS Errors

- Check that frontend URL matches CORS settings in backend
- Verify that credentials: true is set in CORS configuration

File Upload Issues

- Check uploads directory permissions
- Verify file size limits in Multer configuration

Session Issues

- Clear browser cookies
- Check session configuration in app.js
- Verify SESSION_SECRET is set properly

Database Programming Features

NeuNotes extensively uses database programming objects to ensure data integrity and consistency:

1. **Stored Procedures**: Centralize business logic at the database level
2. **Triggers**: Automate statistics updates and ensure data consistency
3. **Functions**: Generate codes and perform data validations
4. **Constraints**: Enforce data integrity rules
5. **Transactions**: Ensure atomicity of complex operations

These features demonstrate a sophisticated understanding of database design principles and showcase the ability to leverage database-level programming for application optimization.

Contributors

- Achyut Katiyar | 002034885
- Anand Pinnamaneni | 002037203
