System Design: Student Document Management System

---

1. Objective

To create a secure, scalable, and maintainable system that allows students to submit their personal and academic documents, enables admins to manage registrations and document approvals, and provides dashboards for all users.

---

2. Problem Statement

Manual handling of student documents is error-prone, insecure, and inefficient. The system aims to:

- Digitally manage student registrations and documents.
- Automate approval workflows for admins.
- Securely store and serve files.
- Provide clear dashboards and notifications for students and admins.

---

3. Scope

- Student registration and login
- Profile management
- Upload, resubmit, and download documents
- Admin approval/rejection workflows

- Role-based dashboards
- Secure file storage and access
- Search using filters

---

4. Technology Stack

Layer	Technology
Frontend	Angular 20, TypeScript, Bootstrap
Backend	ASP.NET Core Web API 8, C#
Database	SQL Server, v19.0
File Storage	Local file system
Authentication	JWT (JSON Web Tokens), ASP.NET Core Identity

---

5. Backend Design

5.1 Architecture

1. **Layered Architecture**
    - Clear separation of concerns:
      Controller → Service → Repository → Storage
    - Controllers handle HTTP requests and responses.
    - Services contain business logic.
    - Repositories manage database operations.
    - Storage layer handles file storage (local).

2. **Service-Oriented Web API**
    - Modular, reusable services for Students, Documents, Approvals, and Dashboard.
    - Each service encapsulates a specific domain of functionality.
    - Enables scalability and easier maintenance.
3. **Secure File Serving**
    - Files are never served via public URLs.
    - Download requests pass through API for **authentication and authorization checks**.
    - Only owners (students) or authorized admins can access files.
4. **Role-Based Authorization**
    - Roles: `Student`, `Admin`.
    - Students can CRUD their profile and own documents.
    - Admins can manage students, view and approve/reject documents.
    - Enforced via JWT tokens and role guards both in backend and frontend.

5.2 Controllers

Controller	Responsibilities
`AuthController`	Login, Registration, Role Creation
`StudentsController`	Register, profile CRUD, list/search for admin
`DocumentsController`	Upload, list, download, resubmit
`ApprovalsController`	Approve/reject students & documents, remarks
`DashboardController`	Admin/student dashboard metrics

5.3 Services

- `StudentService`
- `DocumentService`
- `ApprovalService`
- `DashboardService`

5.4 Infrastructure

IStorageProvider (Local)

- **Purpose:** Abstracts file storage for flexibility.
- **Implementation:** Local disk or cloud (Azure Blob, AWS S3).
- **Responsibilities:** Save files, generate unique paths, retrieve files, optional versioning/soft-delete.

IFileTypeValidator (Security Hook)

- **Purpose:** Ensures only valid files (e.g., PDFs) are uploaded.
- **Security Role:** Prevents malicious uploads; supports MIME checks, PDF inspection, antivirus hooks.

EF Core Repositories

- **Purpose:** Encapsulate database operations.
- **Responsibilities:** CRUD for Students, Documents, Admins, Admissions; filtering, sorting, paging; optional soft delete/versioning.
- **Benefit:** Separates DB access from business logic; easier testing.

JWT Authentication & Role-Based Authorization

- **Authentication:** Users receive JWT with ID and role.
- **Authorization:** Backend checks token and role for each request.
- **Roles:**
    - Students: manage own profile/documents
    - Admins: manage all students/documents
- **Security:** Stateless, scalable, prevents unauthorized access.

---

6. Backend APIs

Module	Endpoint	Method	Description
Auth	/auth/login	POST	Login, return JWT & role
Student	/students/register	POST	Student registration
	/students/me	GET	Get own profile
	/students/me/updateprofile	PUT	Update profile
	/students/getallstudents	GET	Admin: list/search students
	/students/{id}/approve	PATCH	Admin: approve student
	/students/{id}/reject	PATCH	Admin: reject student
Documents	/documents/upload	POST	Student uploads document
	/documents/mydocuments	GET	Student lists own docs
	/documents/{id}/resubmit	POST	Student resubmits
	/documents/{id}/download	GET	Download document (student/admin)
	/documents/student/{id}	GET	Admin: list all documents of a student
	/documents/{id}/approve	PATCH	Admin approves document
	/documents/{id}/reject	PATCH	Admin rejects document

Dashboard	/dashboard/admin	GET	Admin metrics
	/dashboard/student	GET	Student metrics
Admin Ops	/admins	POST	Add new admin

Profile Completion % = `(filledCount / totalFields) * 100`.

7. Frontend Design (Angular)

7.1 Modules & Components

Module	Components
AdminModule	Dashboard, Student Management, Document Management
StudentModule	Dashboard, Profile Management, My Documents
AuthModule	Login, Register
Shared	Layout, Header, Footer, Guards, Pipes, Directives, Toasts

7.2 Services

- `AuthService`: login, logout, authentication
- `StudentService`: profile CRUD, list/search
- `DocumentService`: upload, download, resubmit, approve/reject
- `DashboardService`: metrics

7.3 Guards

- `AuthGuard`: Protect routes for authenticated users
- `RoleGuard`: Restrict routes by role (Student/Admin)

---

8. Security Design

- **Authentication:** JWT access
- **Authorization:** Role-based (Student/Admin)
- **File validation:** Client & server-side PDF only, max size enforced
- **Secure download:** No direct URLs, streamed after authZ checks
- **CORS:** Allow SPA origin only
- **HTTPS:** Enforce TLS + HSTS
- **PII Protection:** Mask identifiers, no logging of file content

---

9. Data Model

9.1 Student

- `StudentId` (PK)
- Name, Email, PasswordHash
- Course, Status (Pending/Approved/Rejected)
- RegisterNumber (nullable)
- CreatedOn, UpdatedOn

9.2 StudentDocument

- `DocumentId` (PK)
- `StudentId` (FK)
- FileName, FilePath/BlobUri, SizeBytes, MimeType

- Status (Pending/Approved/Rejected)
- Remarks, Version
- UploadedOn, ReviewedOn, ReviewedBy

9.3 AdminUser

- `AdminId` (PK)
- Name, Email, PasswordHash, Role

---

10. Data Flow (Sequence)

10.1 Student Registration

1. Student POST `/students/register`
2. Status = Pending
3. Admin PATCH approve/reject
4. Student sees status on dashboard

10.2 Upload Document

1. Student selects PDF, Angular validates
2. POST `/documents/upload` with JWT
3. API validates, stores file & metadata
4. SPA updates "My Documents" table

10.3 Admin Review

1. Admin GET `/documents?status=Pending`
2. Downloads & inspects document
3. PATCH approve/reject with remarks
4. Student dashboard updates

10.4 Resubmit

1. Student uploads new version via `/documents/{id}/resubmit`
2. Version incremented, status = Pending
3. Admin re-review

---

11. Non-functional Requirements

- **Performance:** 10MB upload ≤ 10s
- **Scalability:** Stateless APIs, externalized file storage
- **Availability:** 99.5–99.9%
- **Security:** Encrypted storage, JWT, HTTPS, AV scanning
- **Compliance:** GDPR-like requests, PII masking

Module	Endpoint	Method	Description	Request Example	Response Example
Auth	/auth/login	POST	Login, return JWT & role	`{ "email": "student@example.com", "password": "Password123" }`	`{ "token": "jwt_token_here", "role": "Student", "expiresIn": 3600 }`
Student	/students/register	POST	Student registration	`{ "name":"John Doe","email":"john@example.com","password":"Password123","dob":"2000-01-01","courseApplied":"BSc CS" }`	`{ "studentId": 1, "status": "Pending" }`
	/students/me	GET	Get own profile	Header: `Authorization: Bearer <token>`	`{ "studentId":1,"name":"John Doe","email":"john@example.com","dob":"2000-01-01","courseApplied":"BSc CS","status":"Pending" }`
	/students/me/updateprofile	PUT	Update profile	`{ "name":"John Doe","dob":"2000-01-01","courseApplied":"BSc CS","address":"123 Street" }`	`{ "message": "Profile updated successfully" }`
	/students/getallstudents	GET	Admin: list/search students	Header: `Authorization: Bearer <admin_token>`	`[ { "studentId":1,"name":"John Doe","email":"john@example.com","status":"Pending","courseApplied":"BSc CS" } ]`
	/students/{id}/approve	PATCH	Admin: approve student	`{ "registerNumber": "REG12345" }`	`{ "message": "Student approved" }`

	/students/{id}/reject	PATCH	Admin: reject student	`{ "reason": "Incomplete documents" }`	`{ "message": "Student rejected" }`
Documents	/documents/upload	POST	Student uploads document	Multipart/form-data: file + `documentTypeId`	`{ "documentId": 1, "status": "Pending" }`
	/documents/mydocuments	GET	Student lists own documents	Header: `Authorization: Bearer <token>`	`[ { "documentId":1,"fileName":"marksheet.pdf","status":"Pending","uploadedOn":"2025-08-18T10:00:00Z" } ]`
	/documents/{id}/resubmit	POST	Student resubmits document	Multipart/form-data: file	`{ "message": "Document resubmitted successfully", "status": "Pending" }`
	/documents/{id}/download	GET	Download document	Header: `Authorization: Bearer <token>`	Streams PDF file with `Content-Disposition: attachment`
	/documents/student/{id}	GET	Admin: list all documents of a student	Header: `Authorization: Bearer <admin_token>`	`[ { "documentId":1,"fileName":"marksheet.pdf","status":"Pending","uploadedOn":"2025-08-18T10:00:00Z" } ]`
	/documents/{id}/approve	PATCH	Admin approves document	`{ "remarks": "Verified" }`	`{ "message": "Document approved" }`
	/documents/{id}/reject	PATCH	Admin rejects document	`{ "remarks": "Blurry scan" }`	`{ "message": "Document rejected" }`
Dashboard	/dashboard/admin	GET	Admin metrics	Header: `Authorization: Bearer <admin_token>`	`{ "totalStudents":50,"pendingApprovals":5,"uploadedDocuments":120,"incompleteProfiles":8 }`

	/dashboard/student	GET	Student metrics	Header: `Authorization: Bearer <token>`	`{ "registrationStatus":"Pending","profileCompletion":80,"pendingDocuments":2,"rejectedDocuments":1,"notifications":3 }`
Admin Ops	/admins	POST	Add new admin	`{ "name":"Admin One","email":"admin@example.com","password":"Password123" }`	`{ "adminId":1,"message":"Admin added successfully" }`

Data Flow of the System

1. Student Registration & Login

Frontend (Angular):

- Student fills the registration form → `POST /api/studentaccount/register`.
- Student logs in → `POST /api/studentaccount/login`.

Backend (ASP.NET Core API):

- Saves the student in **AspNetUsers** (Identity table).
- Saves student details in **Students** table (linked by UserId).
- On login → validates credentials → returns JWT token with role.

Database:

- **AspNetUsers:** authentication info (email, password).
- **Students:** personal + academic info (Name, DOB, Marks, CourseApplied).

---

2. Student Registration Approval

Frontend (Angular):

- Admin Approve/Reject Student registration request→ `PATCH/api/student/{id}/approve`.
- Student can check application status → `GET /api/admission/status/{studentId}`.

Backend:

- Saves admission details in **Admissions** table.
- Default status = Pending.

Database:

- **Admissions:** StudentId, CourseId, Status (Pending/Approved/Rejected).
- **Courses:** list of available courses.

---

3. Student Uploads Documents

Frontend:

- Student uploads PDF → `POST /api/documents/upload`.

Backend:

- Validates file (type, size, optional virus scan).
- Saves file in `/uploads/studentId/yyyy/MM/`.
- Saves metadata (file path, type, size, status) in **Documents** table.

Database:

- **Documents:** DocumentId, StudentId, FilePath, Type, Status.

File Storage:

- Physical file storage (local disk).

---

4. Admin Login

Frontend:

- Admin logs in → `POST /api/account/login`.

Backend:

- Checks role (Admin/User).
- Issues JWT token for admin.

Database:

- **AspNetUsers** with Role = Admin.

---

5. Admin Reviews Students

Frontend:

- Admin dashboard → `GET /api/students/all`.
- Shows list with filters/search.

Backend:

- Fetches all students from DB along with admission & document details.

Database:

- Joins **Students + Admissions + Documents**.

---

6. Admin Approves / Rejects

Frontend:

- Admin clicks Approve/Reject → `PUT /api/admission/updateStatus/{studentId}`.

Backend:

- Updates **Admissions.Status** field.

Database:

- Status updated in **Admissions** table.

---

7. Admin / Student Document Access

Frontend:

- Admin views documents → `GET /api/documents/student/{studentId}`.
- Downloads document → `GET /api/documents/download/{docId}`.

Backend:

- Reads file path from DB → streams file to authorized user.

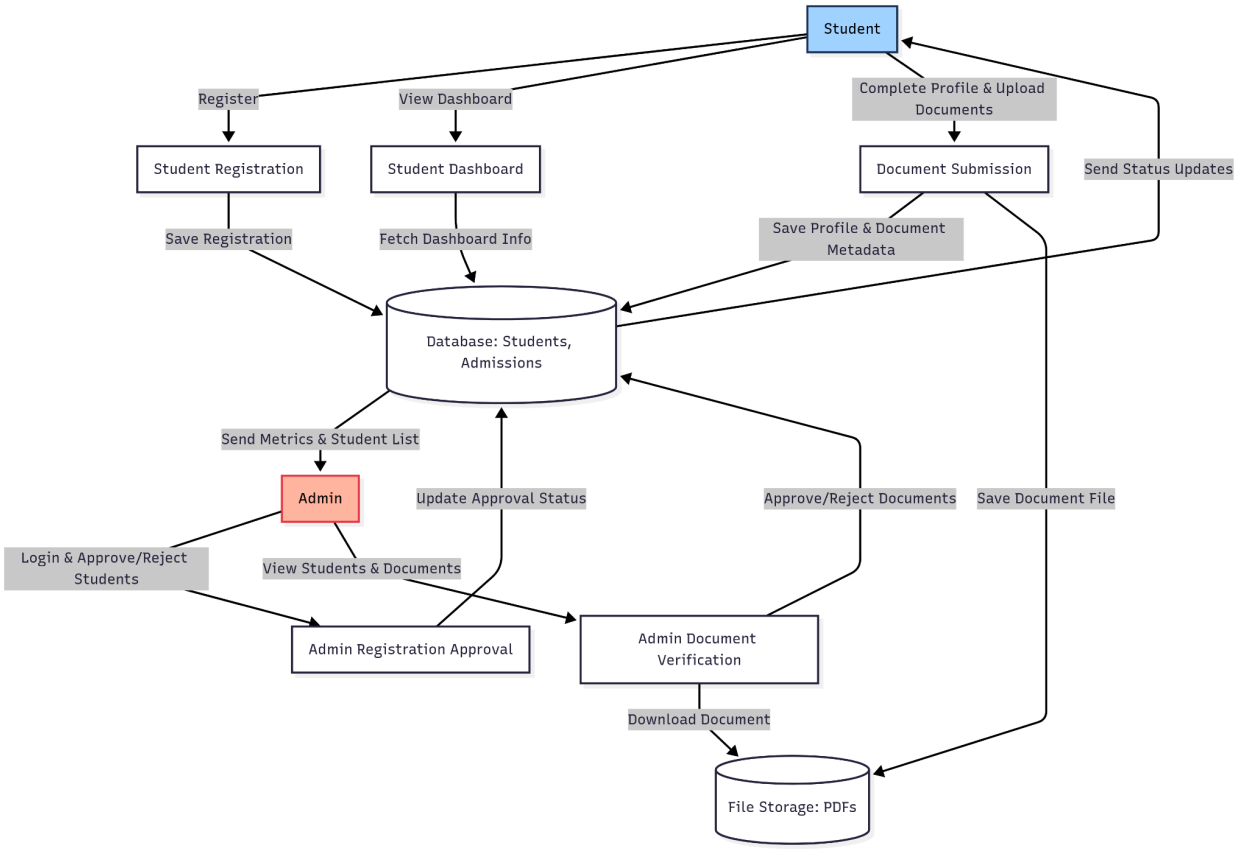
Database:

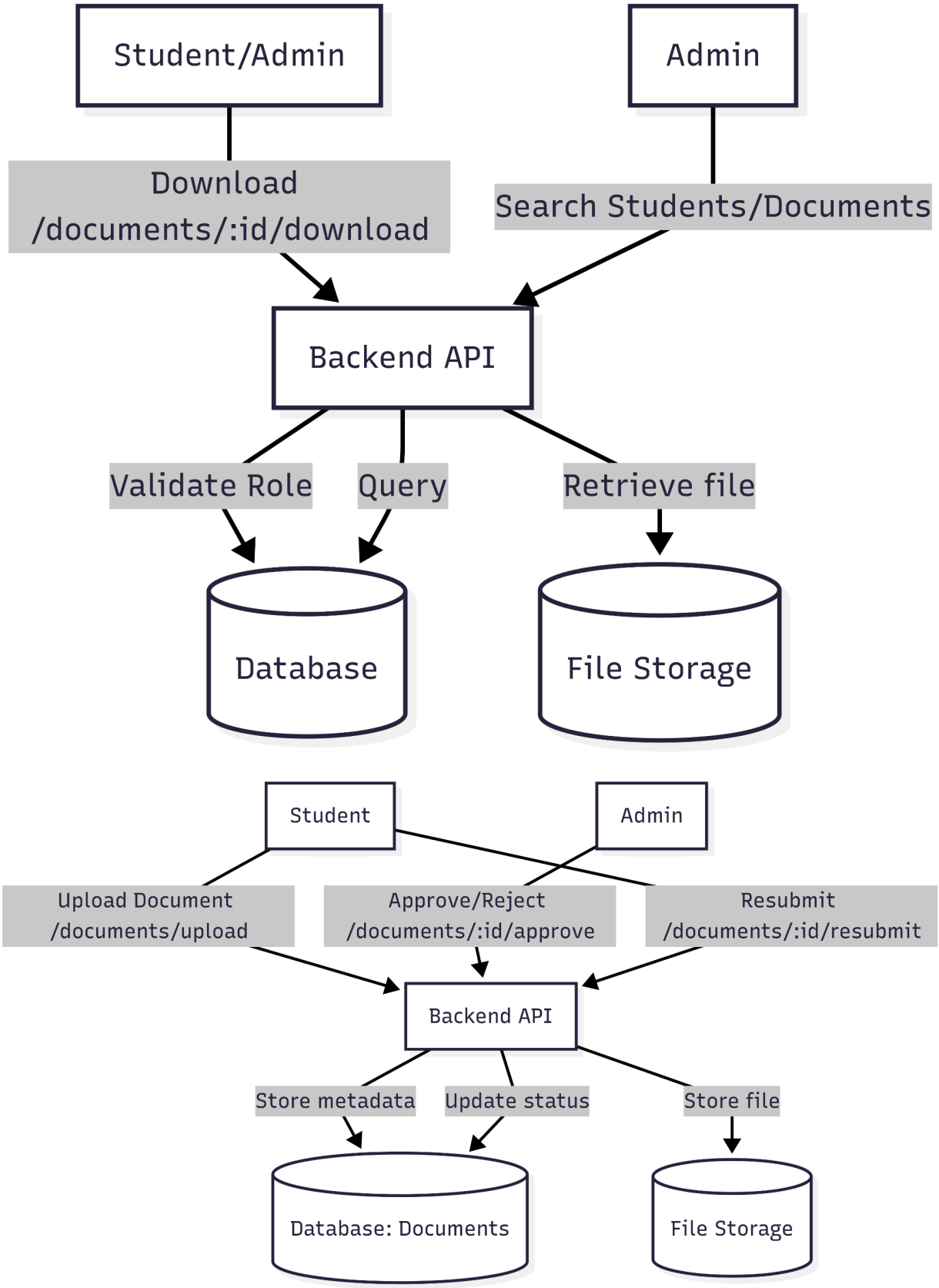
- **Documents** stores only metadata.
- File storage contains the actual file.

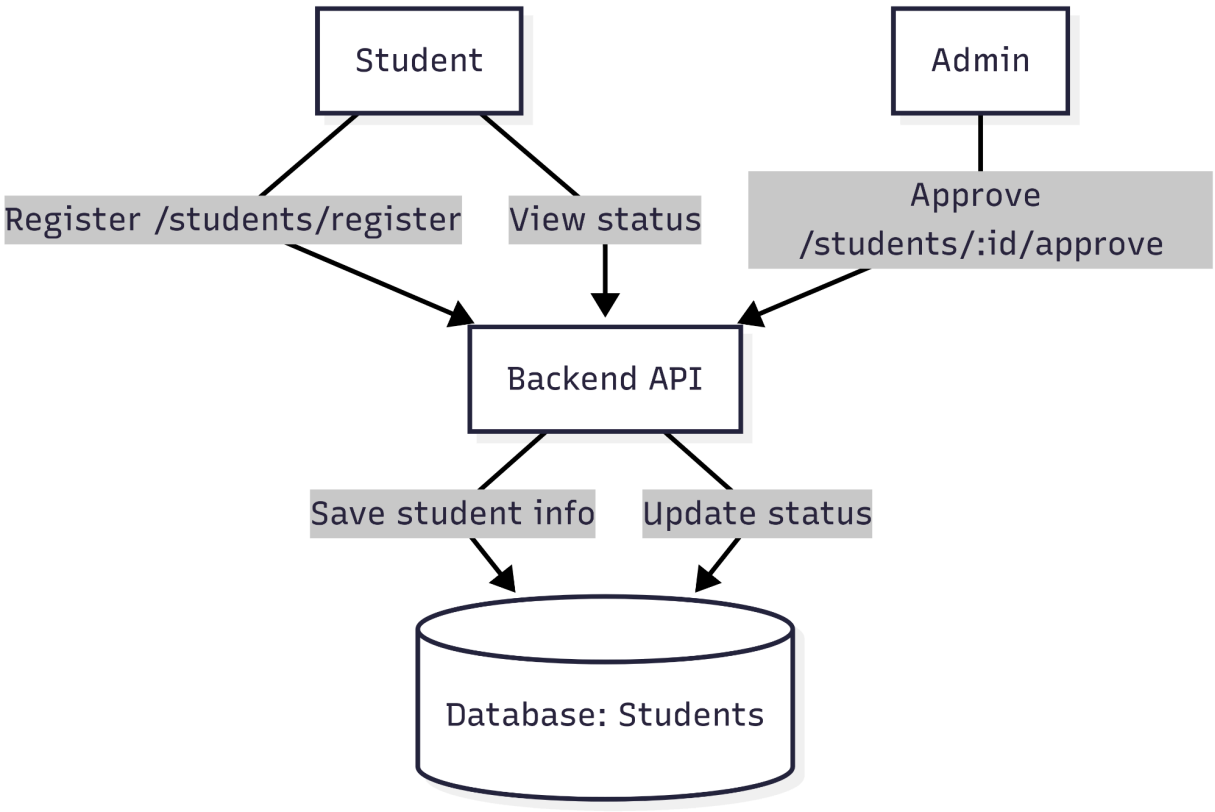
---

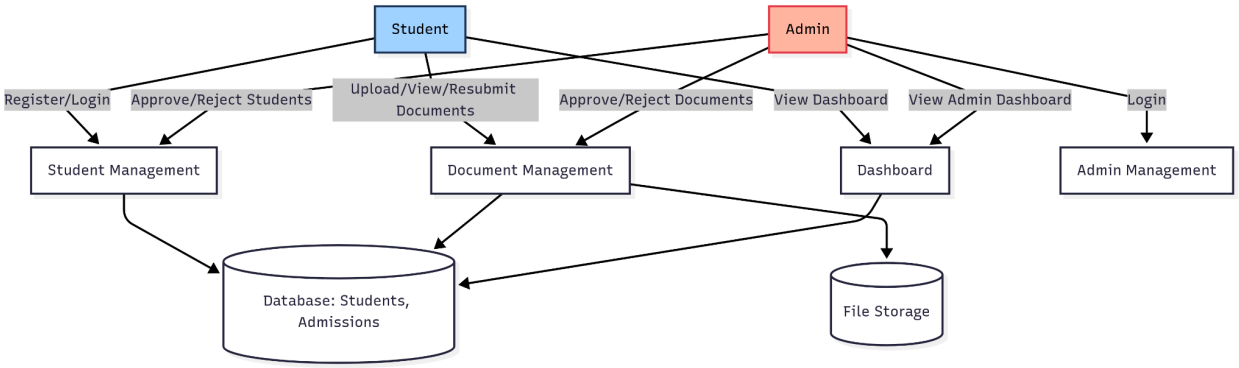
8. Overall Flow (Textual Summary)

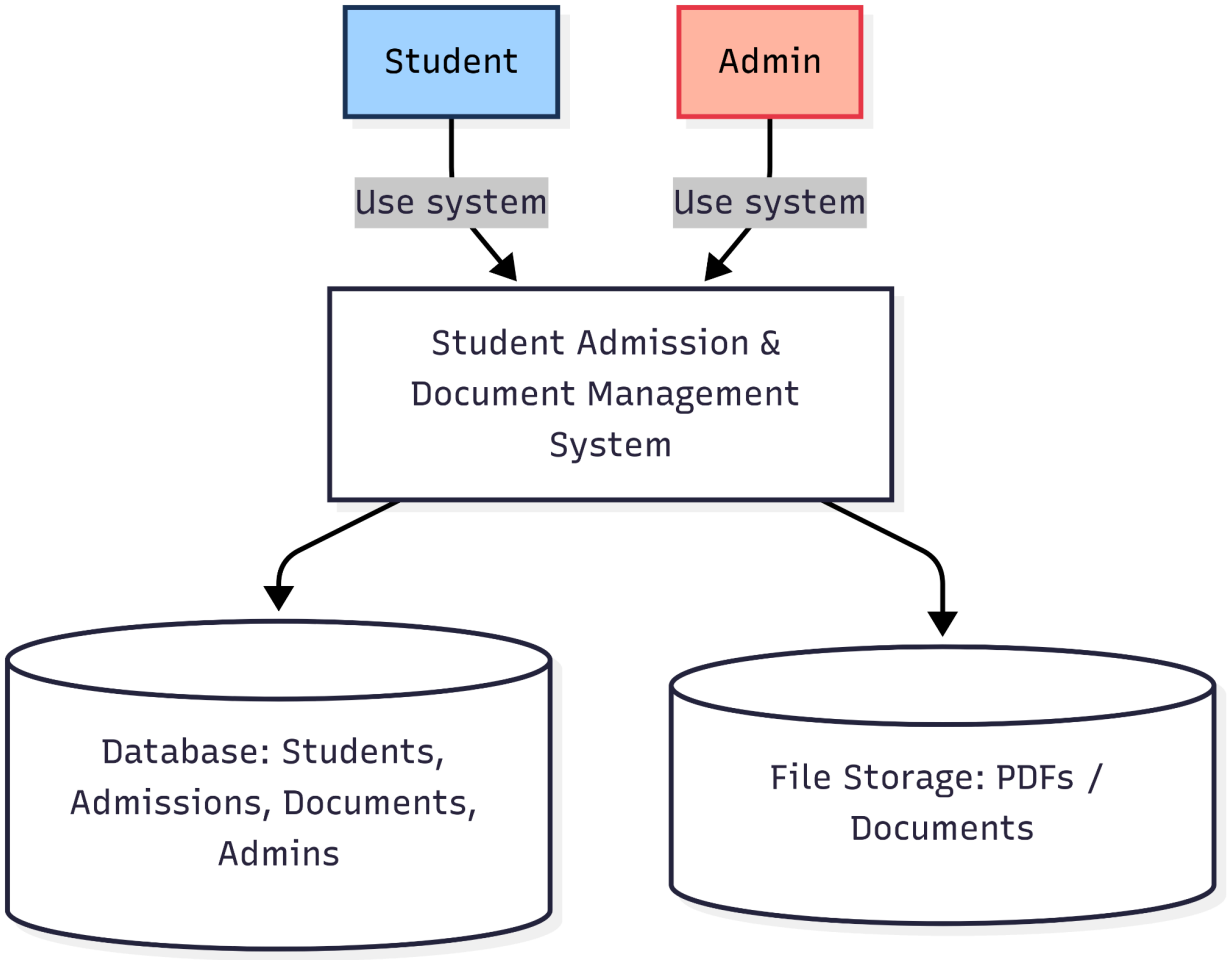
1. **Student App** → Register/Login → **API** → Save in **AspNetUsers** + **Students** → **DB**
2. **Student App** → Apply Admission → **API** → Save in **Admissions** → **DB**
3. **Student App** → Upload Document → **API** → Save file in **File Storage** + metadata in **Documents** → **DB + File System**
4. **Admin App** → Login → **API** → Issue token
5. **Admin App** → View Students → **API** → Fetch from **Students + Admissions + Documents** → **DB**
6. **Admin App** → Approve/Reject Student → **API** → Update **Admissions** → **DB**
7. **Admin App** → View Documents → **API** → Fetch from **Documents** + File Storage → Stream to Admin


Student
Register
View Dashboard
Complete Profile & Upload Documents
Student Registration
Student Dashboard
Document Submission
Send Status Updates
Save Registration
Fetch Dashboard Info
Save Profile & Document Metadata
Database: Students, Admissions
Send Metrics & Student List
Admin
Update Approval Status
Approve/Reject Documents
Save Document File
Login & Approve/Reject Students
View Students & Documents
Admin Registration Approval
Admin Document Verification
Download Document
File Storage: PDFs


Student/Admin
Admin
Download
/documents/:id/download
Search Students/Documents
Backend API
Validate Role
Query
Retrieve file
Database
File Storage
Student
Admin
Upload Document
/documents/upload
Approve/Reject
/documents/:id/approve
Resubmit
/documents/:id/resubmit
Backend API
Store metadata
Update status
Store file
Database: Documents
File Storage


Student
Admin
Register /students/register
View status
Approve
/students/:id/approve
Backend API
Save student info
Update status
Database: Students


Student
Admin
Register/Login
Approve/Reject Students
Upload/View/Resubmit Documents
Approve/Reject Documents
View Dashboard
View Admin Dashboard
Login
Student Management
Document Management
Dashboard
Admin Management
Database: Students, Admissions
File Storage


Student
Admin
Use system
Use system
Student Admission & Document Management System
Database: Students, Admissions, Documents, Admins
File Storage: PDFs / Documents