what “system design” is

System design is the high-level blueprint of a software system. It explains **what parts (components) the system has, how they connect, and why they’re arranged that way**—covering architecture, data models, APIs, security, scalability, deployment, observability, and key flows. Think of it as the map you hand to engineers so they can implement consistently and to stakeholders so they understand capabilities and trade-offs.

Below is a **detailed system design** for your Student Document Management System.

---

1) architecture overview

Style: layered, service-oriented web app
Stack: Angular 19 SPA ←→ ASP.NET Core Web API ←→ SQL Server
Storage: file system (wwwroot or outside) or cloud (Azure Blob, S3)

```
[Browser: Student/Admin]
      |
      ▼
[Angular 19 SPA]
      | HTTPS (JWT)
      ▼
[ASP.NET Core Web API]
  ├─ Auth & RBAC (JWT)
  ├─ Student Service
  ├─ Document Service (validation, virus scan hook)
  ├─ Approval/Workflow Service
  ├─ File Storage Adapter
  └─ Reporting/Dashboard Service
      |
      ├──────────► [SQL Server]   (students, docs, statuses, remarks)
      └──────────► [File Storage] (PDF binaries; path/URI in DB)
```

Key principles

- **Binary/file** stored in storage; **only metadata** (name, path, size, type, status, timestamps) in DB.

- **Secure file serving:** downloads go through an **authorized API** that checks identity/role/ownership before streaming.
- **Clean separation:** controller → service → repository → storage adapter.
- **Extensible storage:** switch local disk ⇄ Azure Blob by config.

---

2) functional decomposition (components)

frontend (Angular 19)

- **Auth module:** login, JWT handling, HTTP interceptors.
- **Student module:** dashboard, profile form, document upload, “my documents”, resubmit.
- **Admin module:** dashboard, student approvals, document reviews, search, download.
- **Shared:** guards (AuthGuard, RoleGuard), UI components (table, progress, badges), file-type pipe, error toaster.

backend (ASP.NET Core)

- **Controllers**
  - `AuthController` (login, refresh)
  - `StudentsController` (register, profile CRUD, list/search for admin)
  - `DocumentsController` (upload, list, download, resubmit)
  - `ApprovalsController` (admin: approve/reject students & docs, remarks)
  - `DashboardController` (counts/metrics)
- **Services**

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

- **Infrastructure**
  - `IStorageProvider` (impls: `LocalStorageProvider`, `AzureBlobStorageProvider`)
  - `IFileTypeValidator`, `IAntivirusScanner` (stub/hook), `IPdfInspector` (MIME/sniffing)
  - Repositories (EF Core)
  - JWT auth & role-based authorization (Student, Admin)
- **Cross-cutting**
  - Logging, exception handling middleware, model validation, rate limiting.

---

3) key API surface (first class endpoints)

All routes under `/api`, secured via JWT unless noted.

auth

- `POST /auth/login` → `{ token, refreshToken, role }`
- `POST /auth/refresh` → new tokens

students (student-facing + admin list)

- `POST /students/register` (public)
- `GET /students/me` (student)
- `PUT /students/me/profile`

- `GET /students` (admin, query: name, course, status, paging)
- `PATCH /students/{id}/approve` (admin) body: `{ registerNumber }`
- `PATCH /students/{id}/reject` (admin) body: `{ reason }`

documents

- `POST /documents/upload` (student)
  multipart/form-data: `file`, `documentTypeId` (optional)
  returns `{ documentId, status: "Pending" }`
- `GET /documents/mine` (student, list) query: paging/sort
- `POST /documents/{id}/resubmit` (student) multipart: `file`
- `GET /documents/{id}/download` (owner student or admin) **streams file**
- `GET /documents` (admin) query: studentName, course, status, date, paging
- `PATCH /documents/{id}/approve` (admin)
- `PATCH /documents/{id}/reject` (admin) body: `{ remarks }`

dashboard

- `GET /dashboard/admin` → totals: students, pending approvals, docs uploaded, incomplete profiles
- `GET /dashboard/student` → registration status, profile completion %, pending/rejected docs, notifications count

Conventions

- Use **problem-details** responses for errors (RFC 7807).
- Use **ETags** for idempotent resubmits if needed.

---

4) security model

- **Auth:** JWT (access + optional refresh), password hashing with ASP.NET Core Identity or custom with PBKDF2/Argon2.
- **RBAC:** Roles: `Student`, `Admin`.
  - Students: CRUD own profile/docs, download own docs.
  - Admins: list/search any, approve/reject, download any.
- **File validation:**
  - Client: accept `.pdf` only, max size (e.g., 10 MB).
  - Server: enforce max size via `RequestSizeLimit`, **MIME sniffing** (read header), **extension whitelist**, optional **PDF signature check**.
- **Virus scanning hook:** pluggable `IAntivirusScanner` (e.g., ClamAV daemon or cloud scanning) before persisting/marking “Pending”.
- **Secure download:**
  - No direct static URLs.
  - `GET /documents/{id}/download` opens stream after **authZ** check (owner or admin), sets `Content-Disposition: attachment; filename="...pdf"`.
  - If using cloud storage, generate **short-lived signed URL** server-side only after authZ.
- **PII protection:** store minimal PII; mask in logs; never log file content or paths with student identifiers in plaintext.
- **Rate limiting & anti-abuse:** per-IP and per-user upload/download limits.
- **CORS:** allow SPA origin only.
- **HTTPS only**; HSTS on gateway; secure cookies when used.

---

5) data flows (sequence-level)

A) student registration → approval

1. Student submits `/students/register`.
2. Status = `PendingApproval`.
3. Admin reviews list → `PATCH /students/{id}/approve` with register number **or** reject with reason.
4. Student sees status/notification on dashboard.

B) upload document (student)

1. Student selects PDF → Angular validates (type & size) → builds `FormData`.
2. `POST /documents/upload` with JWT.
3. API checks role, size, type, scans, writes file via `IStorageProvider`, inserts metadata row (status=`Pending`).
4. Returns doc id & status.
5. SPA refreshes “My Documents” table.

C) review & decision (admin)

1. Admin filters `/documents?status=Pending&...`.
2. Opens a doc (secure download) to verify.
3. Approves or rejects (`PATCH /documents/{id}/approve|reject` with remarks).

4. Student dashboard updates; rejected docs show remarks and **Resubmit** button.

D) resubmit (student)

1. Student clicks resubmit → uploads new PDF to `/documents/{id}/resubmit`.
2. Server versions the file (e.g., `docId/version` path) or supersedes with audit trail.
3. Status back to `Pending`.

---

6) storage strategy

option 1: local file system (simple to start)

- Root: outside web root for safety (e.g., `D:\app-data\docs\`).
- Path pattern: `/student/{studentId}/{documentId}/{yyyy}/{MM}/{randomGuid}.pdf`
- Pros: easy, cheap. Cons: scaling, HA, backup complexity.

option 2: cloud blob (recommended if multi-instance)

- Azure Blob (`container: student-docs`) with server-side encryption.
- Store blob URI in DB; **never** expose raw URI without SAS token.
- Lifecycle rules (versioning, soft delete, retention).

Backups: DB full + differential; storage snapshots; test restore runbooks.

---

7) non-functional requirements

- **Performance:** upload ≤ 10s for 10MB on typical broadband; list queries < 500ms (indexed).
- **Scalability:** stateless API; can scale horizontally. File storage externalized.
- **Availability:** 99.5–99.9% target; health checks + rolling deploys.
- **Observability:** structured logs (request id, user id); metrics (uploads/min, approvals/day); traces (OpenTelemetry).
- **Compliance:** data residency if needed; GDPR-like delete/export upon request.
- **Accessibility:** keyboard nav, ARIA labels, color-contrast for status badges.

---

8) Angular application design (high level)

```
/app
  /core (auth service, interceptors, guards)
  /shared (ui components, directives, pipes)
  /auth (login)
  /student
    /dashboard
    /profile
    /documents (list + upload + resubmit modal)
  /admin
    /dashboard
    /students (list/approve/reject)
    /documents (review table + approve/reject + download)
```

Key pieces

- **HTTP Interceptor:** attach JWT; handle 401 → redirect login.
- **Upload service:** `upload(file: File, documentTypeId?: number): Observable<HttpEvent<any>>` with `reportProgress: true` to power progress

bar.

- **Guards:** `AuthGuard`, `RoleGuard(['Admin'])`.
- **Tables:** server-side paging/sort/search (query params).

---

9) backend implementation notes

- **Controllers** are thin; all logic in Services.
- **DTOs & validation:** FluentValidation or data annotations (`[MaxFileSize]`, `[AllowedExtensions(".pdf")]` custom).
- **Streaming:** `FileStreamResult` for downloads; `Response.Body.WriteAsync` for large files; enable `IIS/IIS Kestrel` request body size limits appropriately.
- **EF Core:** code-first, indexes on `(StudentId, Status)`, `(Status, UploadedOn)`; soft delete column if needed.
- **Concurrency:** use row version (`byte[] RowVersion`) on `StudentDocument`.

---

10) data model (minimum, aligned to your brief)

Student

- `StudentId (PK)`
- `Name, Email (unique), PasswordHash, Course`
- `Status` (Pending, Approved, Rejected)

- `RegisterNumber (nullable)`
- `CreatedOn, UpdatedOn`

StudentDocument

- `DocumentId (PK)`
- `StudentId (FK)`
- `FileName, FilePath/BlobUri, SizeBytes, MimeType`
- `Status` (Pending, Approved, Rejected)
- `Remarks (nullable)`
- `Version (int, default 1)`
- `UploadedOn, ReviewedOn (nullable), ReviewedBy (AdminId, nullable)`

AdminUser

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

---

11) status model & workflow rules

- Student: `Pending → Approved` or `Rejected` (+ reason; can re-apply).
- Document: `Pending → Approved` or `Rejected` (remarks).
  Resubmit sets `Status = Pending`, optionally increments `Version`.

---

12) error handling & UX

- Show **client-side validation** (PDF only, size cap) + server errors (problem-details) in toast/banner.
- Upload: show **progress bar** via `HttpClient` events; disable submit while in progress.
- Table empty states & filter chips; badges: `Pending` (warning), `Approved` (success), `Rejected` (danger).

---

13) deployment & environment

- **Environments:** dev, test, prod; separate DB/storage accounts.
- **Configs:** size limits, allowed types, storage provider, virus scan on/off.
- **CI/CD:** build Angular → static hosting (e.g., Azure Static Web Apps) or serve via Nginx; API on App Service/Container Apps; run EF migrations on deploy.

---

14) risks & mitigations

- **Large files** → chunked upload later; server limits configured.
- **Malicious PDFs** → MIME sniff + AV scan + safe download (no inline view unless demanded).
- **Path traversal** → never trust filename; generate server names; normalize paths.
- **Scaling** → avoid local disk for multi-instance prod; prefer Blob/S3.

---

15) acceptance checklist (ties to your evaluation)

- ✅ Correct file handling: validated on client & server, scanned, stored outside web root / blob, metadata in DB.
- ✅ Secure serving: authZ check per download; no public static links; signed URLs (if cloud).
- ✅ Angular handling: FormData upload with progress; list with download links; resubmit flow; filters; role-guarded routes.

If you'd like, I can turn this into a **one-page system design diagram + sequence flows** you can drop into your slides, and a **concise API spec table** for the appendix.