# WorkFlow

Students upload activity evidence → client & server validate → system fingerprints file (SHA-256) and stores metadata + file → mentors review/approve with comments → approved records become part of auditable portfolios and one-click PDF exports for reports or accreditation.

---

# Actors & components

**Actors**

- Student (uploader / submitter)
- Mentor / Advisor (reviewer)
- Coordinator / Department Admin (approver/escalation)
- Super Admin (system + policy config)
- External consumer (accreditation body / recruiter)

**System components**

- UI (Thymeleaf server-rendered pages)
- Client-side validation module (browser JS)
- API / Backend (Java 24 + Spring Boot)
- Auth & RBAC (Spring Security)
- Validation pipeline (server checks, virus/format checks)
- Hashing & de-duplication module (SHA-256 + activity-scoped index)
- Storage: metadata DB (MySQL 8) + file store (local or object store)
- PDF generation service (Thymeleaf → PDF renderer)
- Background workers / queue (for heavy tasks: PDF rendering, async validation)
- CI/CD + Flyway migrations and monitoring/logging

---

# End-to-end workflow — step-by-step

## 1. User authentication & landing

1. Student signs in (local credentials or institutional SSO).

2. System applies RBAC — UI shows only allowed actions (submit, view status).
   **What to say on stage:** “Users sign in with secure roles — students submit evidence, mentors review it.”

## 2. Start submission (student)

1. Student selects activity type (workshop, certificate, conference, project).
2. Student fills metadata fields (title, date, description, role played, organizing body).
3. Student uploads file(s) (certificate PDF, jpg, docx, photo).
   **UX note:** show allowed types/size limits and examples inline.

## 3. Client-side validation (fast feedback)

1. Browser JS checks file type, size, and basic rules (e.g., required fields present).
2. If failure → immediate client error message (prevent upload).
   **Why:** reduces wasted uploads and improves UX.

## 4. Upload & server-side initial handling

1. Client initiates multipart upload to backend or directly to object store (pre-signed URL).
2. Backend receives the upload or verifies upload completion; calculates SHA-256 *as the file streams in* (avoid double I/O).
3. Backend performs server-side validation: content-type check, size enforcement, optional virus scan, minimal heuristics (e.g., PDF contains readable text).
4. If validation fails → return clear error (reason + suggested fix), keep the upload lifecycle idempotent so the student can retry.
   **Failure handling:** store a rejected-upload record for audit and user messages.

## 5. Hashing & duplicate detection

1. Compute SHA-256 hash of file bytes.
2. Look up `file_hash` in an **activity-scoped index**: query `WHERE activity_id = ? AND file_hash = ?`.
   - If exists → link to existing evidence (optionally warn student or block duplicate) and notify mentor/admin if needed.
   - If not exists → commit record.
3. Store file in file-store (naming using content-addressable path or unique id) and store metadata in MySQL: user_id, activity_id, file_path, file_hash, upload_time, status=pending_review, checksum_algo.

## 6. Audit & metadata

1. Create an immutable audit entry for the upload action (who, when, ip, metadata).
2. Set submission status: `pending_review`.
   **What to say on stage:** “Every upload creates an auditable record so we can trace provenance.”

## 7. Review workflow (mentor)

1. Mentor gets a notification (email/in-app) or sees pending queue.
2. Mentor opens submission, views file (inline PDF/image) and metadata.
3. Mentor can: comment (request correction), approve, or reject.
4. Approval action records approver_id, timestamp, comments, and status change to `approved`.
5. If `comment` or `request resubmission`, student receives actionable feedback; the student can re-upload a corrected file (new hash & audit record).
   **Escalation:** Coordinator can be added if mentor requests escalation on contested items.

## 8. Post-approval actions & exports

1. Approved items are indexed into the student’s “verified portfolio”.
2. Single or bulk **one-click PDF exports**: backend composes a Thymeleaf template with metadata + approval history + embedded evidence thumbnails or links; renders to PDF via a server-side renderer (worker).
3. PDF can be downloaded, emailed, or attached to institutional reports for accreditation.
   **Scaling note:** large exports are queued; user gets a notification when ready.

---

## Failure modes & mitigations (practical)

- **Bad migration stops startup:** Use Flyway validate in CI and lazy injection for filters to avoid DB access at app bootstrap. Keep repair procedures documented.
- **Large/invalid uploads spike storage:** Client-side size checks + server-side limits + quotas + reject + helpful error messages. Consider resumable & chunked uploads.
- **Duplicate submissions:** SHA-256 + activity-scoped index to detect duplicates and provide human-friendly guidance to students.
- **Forged documents:** Hashing only detects duplicates/tampering after upload — prevention of forgery requires mentor check and future optional verifier integration (DigiLocker / issuer APIs).
- **PDF generation bottleneck:** Offload to worker pool and queue large batches; scale worker pods horizontally.

- **Data corruption:** File-level SHA-256 re-verification, backups, and restore playbooks.

---

## Data design (short & practical)

**Core tables (example)**

- `users` (id, name, email, role, institution_id, sso_id, created_at)
- `activities` (id, user_id, activity_type, title, date, description, created_at)
- `files` (id, activity_id, uploader_id, file_path, file_hash, checksum_algo, size, mime_type, status, uploaded_at)
- `reviews` (id, file_id, reviewer_id, action {approved/rejected/comment}, comment_text, timestamp)
- `audit_logs` (id, entity_type, entity_id, action, performed_by, meta, timestamp)
  **Indexes**
- Unique index: `(activity_id, file_hash)` for quick duplicate detection.
- Indexes on `user_id`, `status`, `uploaded_at` for query performance.

---

## API & endpoint examples (concise)

- `POST /api/v1/auth/login` — authenticate
- `GET /api/v1/activities` — list user activities
- `POST /api/v1/activities/{id}/files` — upload file (returns upload_id)
- `POST /api/v1/files/{upload_id}/finalize` — server validates, computes hash, stores metadata
- `GET /api/v1/reviews/pending` — mentor pending queue
- `POST /api/v1/reviews/{file_id}` — mentor action (approve/comment)
- `POST /api/v1/exports/portfolio/{user_id}` — request PDF export (async)
- `GET /api/v1/exports/{export_id}` — download export when ready

---

## Background & async tasks

- **Validation workers** — run heavier checks (virus scan, OCR heuristics).
- **PDF workers** — render templates to PDF, optimize images.
- **Notification workers** — emails/in-app notifications for mentor/ student.

- **Cleanup/Archival worker** — archive old files per retention policy.

---

## Deployment & operations

- Package as a runnable JAR (embedded Tomcat) — containerize with Docker.
- CI pipeline: build → unit/integration tests → Flyway validate in a staging database → deploy staging → smoke tests → production rollout (blue/green or rolling).
- Monitoring: metrics (submission rate, review latency, export queue length), logs (structured), alerting on worker failures.
- Backups: DB logical + file-store snapshots; test restore procedures quarterly.

---

## Security & compliance checklist

- TLS for all traffic (HTTPS).
- Spring Security with RBAC and optional SSO (SAML/OAuth).
- Store only metadata in DB; files in access-controlled object storage.
- Option for encryption at rest for files or DB fields.
- Audit logs immutable and exportable for accreditation/legal review.
- Admin-configurable retention and data-deletion workflows to meet privacy laws.

---

## Example realistic scenario (narrative)

“Riya, a student, uploads a workshop certificate. Her browser blocks the upload because the file is 120MB (limit 50MB) — she compresses it and retries. The server computes SHA-256 during upload, finds no duplicate, stores the file in the institution’s object store, and creates a `pending_review` file record. Mentor Ali sees a pending item, opens the PDF, leaves a short comment asking for the workshop date to be clearer, and marks it as `needs resubmission`. Riya receives the comment, re-uploads the corrected file; the new hash is different, mentor approves it, and Riya downloads her verified portfolio PDF for placement interviews.”

**What to say on stage:** “Users get clear, actionable feedback at each step — from instant client errors to mentor comments and final verified PDF.”

---

## What to highlight during your presentation (3 bullets)

- **Trust & auditability:** SHA-256 + timestamped approvals = tamper-evident record.
- **Practicality & deployability:** Mature stack (Java/Spring/Thymeleaf/MySQL) — easy to deploy and maintain.
- **Efficiency gains:** Dual validation reduces bad uploads; bulk exports and review queues save coordinator time.