Zewail City of Science, Technology and Innovation
University of Science and Technology
School of Computational Sciences and Artificial Intelligence

CSAI 203 - Fall 2025
Introduction to Software Engineering
|
Software Requirements Specification
for
Adaptive Collaborative Code Learning Lab (ACCL)

Team Number: ***Team 18***
Team Members:
Amr Yasser 202301043
Omar Hazem Ahmed 202300800
Abdelrahman Mohamed 202301645
Hady Emad Saeed 202301707
Representative ID: ***202301043***
Representative Contact info:
s-amr.anwar@zewailcity.edu.eg

Version: 1.0
Organization: Zewail City
Date: ***19/12/2025***

---

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) describes the functional and non-functional requirements for the Adaptive Collaborative Code Learning Lab (ACCL). ACCL is a web-based platform that enables instructors to publish programming exercises, students to submit executable code solutions, and the system to evaluate submissions automatically in a secure sandbox. The SRS is intended for project stakeholders (team members, instructors, and graders) and developers who will implement the Phase-4 prototype and final system.

1.2 Scope

ACCL is a multi-user, web-based application that supports the full assignment lifecycle for programming exercises:

- Role-based authentication and authorization (students, instructors, admins).
- Instructor assignment creation and management (including multiple test cases and rubrics).
- Student code submissions (paste or file upload) with versioning and diffs.
- Secure, isolated sandbox execution of student code with enforced resource limits.
- Automated grading: per-test pass/fail, runtime, stdout/stderr capture, aggregate scoring.
- AI-assisted contextual hints for failing tests, using **pre-trained** models only (no model training).
- Similarity detection (plagiarism advisory) via token / embedding comparisons and highlighted diffs for instructor review.
- Peer review workflow (anonymized reviews with configurable rubric).
- Instructor analytics and exportable reporting (CSV).
- Notifications (in-app; email simulation for prototype) and administrative controls.

Out of scope (explicit):

- Training custom machine learning models from scratch.
- Production-grade horizontal scaling and enterprise deployment beyond course demo requirements (Phase-4 uses SQLite; Phase-5 outlines Postgres migration).
- Payment or billing integrations, and full LMS SSO implementations beyond documented integration points.

1.3 Definitions, Acronyms, and Abbreviations

- ACCL: Adaptive Collaborative Code Learning Lab
- AI: Artificial Intelligence (pre-trained models used for hints and embeddings)
- API: Application Programming Interface
- DB: Database
- Embedding: Vector representation of code or text used for similarity/search
- FR: Functional Requirement
- MVC: Model-View-Controller (architectural pattern)
- NFR: Non-Functional Requirement
- RMS: Result Management Service (logical internal module name)
- SRS: Software Requirements Specification
- SDLC: Software Development Life Cycle
- Sandbox: Isolated execution environment for running user code
- SQL: Structured Query Language
- UI: User Interface
- UML: Unified Modeling Language
- CI/CD: Continuous Integration / Continuous Delivery

1.4 References

- IEEE 830-1998 Software Requirements Specification Standard (template reference)
- Project brief and course handouts (CSAI203 Fall 2025)
- Flask Documentation (for backend constraints).

1.5 Overview

This SRS is structured as follows: Section 2 provides an overall description of the product, users, and environment. Section 3 details specific requirements, including functional and non-functional aspects, use cases, and interfaces. Section 4 includes appendices for data dictionary and glossary. Diagrams (e.g., use case, class) are referenced and stored in the GitHub repository under /docs/diagrams. This document ensures traceability, with FRs linked to use cases and test plans for evaluation in later phases.

---

2. Overall Description

2.1 Product Perspective

ACCL is a standalone web application that interfaces with the existing University Authentication System.

ACCL provides course-level automatic grading and feedback for programming assignments and can optionally integrate with institutional SSO, external AI providers, and object storage via configurable adapters; the sandbox runner is separated from web-facing services to ensure safe, isolated code execution.

2.2 Product Functions

- Authentication & Authorization: The system shall support user registration, secure login, password hashing, role-based access control, and session management.

- Assignment Management: The system shall allow instructors to create, update, and delete assignments; attach visible and hidden test cases; set release and due dates; and define grading rubrics.

- Submission Management: The system shall allow students to submit code (upload or paste), record language metadata and submission versions, and display submission history and diffs.

- Sandbox Execution & Grading: The system shall execute submissions in an isolated sandbox with configurable resource limits, run test suites, and persist structured per-test results for reporting.

- Automated Grading & Reporting: The system shall compute scores from test results, generate test reports, and expose results to instructors and students according to role and timing rules.

- AI Hints: On test failures, the system shall generate up to three progressive contextual hints. Hints shall be cached for repeatable failure fingerprints to reduce repeated model

calls.

- Similarity Detection: The system shall compare submissions against prior submissions and highlight flagged similarities for instructor review.

- Peer Review: The system shall support anonymized peer review assignment, rubric submission, and aggregation of peer scores.

- Analytics & Admin Tools: The system shall provide instructor dashboards (e.g., pass rates, per-assignment summaries), CSV export of data, and user/role management tools.

- Notifications: The system shall provide in-app notifications and a configurable email notification simulation/logging mechanism for key events (e.g., grading completion).

- Documentation & Help: The system shall include quickstart guides for users and developers and in-app contextual help.

2.3 User Classes and Characteristics

User Class	**Primary Responsibilities**	**Typical Tasks**	**Technical Skills**	**Access Method**
Student	Submit solutions, view feedback/hints, participate in peer reviews.	Upload/paste code, review history/diffs, respond to peer reviews.	Basic programming; basic web literacy (forms, uploads).	Web browser (laptop/desktop; mobile optional).
Instructor	Create assignments & test cases, configure rubrics, review analytics & similarity flags.	Create/update assignments, inspect submissions/analytics, resolve flags, export CSVs.	Programming concepts; moderate web literacy.	Web browser; elevated UI for assignments/analytics.
Admin	Manage users/roles,	Create users, assign roles,	System-admin familiarity	Web admin

	configure settings, run exports, monitor security/audit logs.	configure system settings, review logs.	preferred.	console with full privileges.
System (Worker/ Sandbox Runner)	Execute code in isolation, run tests, collect/persist results, return outcomes.	Fetch jobs, execute with limits, capture output/statuses, persist logs.	N/A (system component, non-human service).	Isolated service with restricted privileges.

2.4 Operating Environment

- **Client:** Modern web browsers (Chrome, Firefox, Edge) supporting JS and Bootstrap responsive layouts.
- **Server:** Linux (Ubuntu 20.04+) recommended; Python 3.10+ runtime.
- **Development DB: SQLite** via **SQLAlchemy** (`sqlite:///accl.db`) for Phase-4/demo. Use Write-Ahead Logging (WAL), set a busy timeout, and ensure each process/thread uses its own SQLAlchemy session to reduce contention.
- **Sandbox:** Docker Engine (recommended) for per-submission containers; fallback documented OS-level resource-limits for dev.
- **CI/CD:** GitHub Actions for tests & build pipelines.
- **Storage:** Local filesystem for logs/artifacts in Phase-4.

2.5 Design and Implementation Constraints

- Use **Flask**, App Factory, and Blueprints for modularity.
- HTML/Jinja2 with Bootstrap for responsiveness.
- Sandbox MUST disallow external network access, restrict filesystem writes, and enforce CPU/time/memory limits.

- Passwords must be stored using modern hashing. TLS/HTTPS required for client-server communication.
- Only **pre-trained** AI models permitted; the system must not train new models.
- Database: SQLAlchemy ORM; SQLite for dev, with schema for entities like users, assignments.
- All deliverable documents must follow the course naming convention and be uploaded to Google Classroom by the listed deadlines.

2.6 User Documentation

- In-app: Help sections, tooltips, and FAQ.
- External: User manual (PDF) covering login, submission, grading; technical guide for setup/deployment.
- Repository: README.md with install/run instructions, including docker-compose for demo.

2.7 Assumptions and Dependencies

- Users have reliable internet access.
- Users have basic computer literacy (can use a web browser, paste/upload files).
- Users access ACCL through a modern browser (Chrome, Firefox, Edge).

---

3. Specific Requirements

3.1 Functional Requirements

FR-01: User Authentication & Roles

Statement: The system shall provide secure user registration, authentication, session management, and role-based authorization. Passwords must be stored using a strong hashing algorithm (bcrypt or Argon2). Roles supported: **Student**, **Instructor**, **Admin**.

Actors: New user, Registered user, Admin.

Preconditions: Valid email address supplied at registration; system mail/logging service available for password reset.

Postconditions: Authenticated session established with appropriate role-based permissions.

Primary flow:

1. User submits registration form (name, email, password).
2. System validates input, hashes password, creates account and default role.
3. User logs in with credentials; system issues a session token and enforces expiration.
4. User logs out; session invalidated.

Alternate / error flows:

- Duplicate email → registration rejected with descriptive error.
- Invalid credentials → login rejected; optional lockout after configurable failed attempts.
- Password reset → user requests reset, receives time-limited token, uses token to set new password.

Acceptance criteria:

- No plaintext passwords in storage.
- Protected endpoints return HTTP 403 for unauthorized roles.
- Session expiration and logout behave per configured TTL.
- Password reset tokens are one-time and expire.

Test procedure (high level):

- Create account → inspect DB for hashed password.
- Call protected endpoint with insufficient role → assert 403.

- Simulate token expiry → verify login prevented.
- Execute password reset flow and verify one-time token rules.

FR-02: Assignment Management (Instructor)

Statement: Instructors shall be able to create, edit, and delete assignments; define visible and hidden test cases; specify languages, release and due dates, rubrics, and late policies.

Actors: Instructor, Admin.

Preconditions: Instructor account with course assignment privileges.

Postconditions: Assignment stored and visible per release date and course membership.

Primary flow:

1. Instructor selects “Create Assignment.”
2. Instructor provides metadata (title, description, languages, rubric, release/due dates) and uploads test cases (visible/hidden).
3. System validates input and persists the assignment.

Alternate / error flows:

- Edit after release: changes to hidden tests are versioned and logged.
- Delete assignment with existing submissions: system prompts for soft-delete or archive.

Acceptance criteria:

- Assignment becomes visible to enrolled students at release time.
- Hidden test configuration accepted and stored separately from visible tests.
- Changes are auditable.

Test procedure:

- Create assignment with hidden/visible tests → verify student view before/after release.
- Edit hidden test after release → verify audit record and versioning behavior.

FR-03: Student Code Submission

Statement: The system shall accept student submissions via text paste or file upload, record metadata (language, filenames), maintain version history, and enqueue grading jobs.

Actors: Student, Instructor, System (grader).

Preconditions: Student enrolled; assignment is visible or allowed by policy (late allowances).

Postconditions: Submission persisted; grading job created.

Primary flow:

1. Student selects assignment and submits code (paste or file).
2. System auto-detects language (or accepts user selection), stores submission as version N, and places a job on the grading queue.

Alternate / error flows:

- File size exceeds limit → reject with error.
- Interrupted upload → rollback partial content; display error.

Acceptance criteria:

- Submission record includes timestamp, language, and version.
- Prior versions retrievable and diffable.
- Grading job created and queued.

Test procedure:

- Submit sample code → inspect DB for versioned record and queue job entry.

- Attempt oversized file upload → assert rejection.

FR-04: Secure Test Sandbox Execution

Statement: The system shall execute tests within an isolated sandbox environment per submission, enforcing per-test timeouts (≤ 5 s), CPU and memory limits, no outbound network access, and constrained filesystem scope.

Actors: System (grader), Instructor, Admin.

Preconditions: Grader infrastructure available and configured with sandbox runtime.

Postconditions: Test outputs (pass/fail, logs) persisted; sandbox destroyed.

Primary flow:

1. Grader provisions sandbox, copies submission and test harness, executes tests under resource/time constraints, collects logs, and destroys sandbox.

Alternate / error flows:

- Test exceeds timeout → process killed, test marked errored.
- Attempted network access or filesystem escape → operation blocked and incident logged.

Acceptance criteria:

- Outbound network attempts fail.
- Filesystem access outside workspace denied.
- Processes exceeding timeout are terminated and logged.

Test procedure:

- Execute network call from inside sandbox → assert failure.
- Attempt to read system files → assert permission denied.

- Run long loop → assert termination at ≤ 5 s.

FR-05: Automated Grading & Test Report

Statement: The system shall run instructor-provided test suites for each submission, produce structured per-test results (pass/fail, durations, stdout/stderr), compute weighted scores using the assignment rubric, and present results to students (hiding outputs for hidden tests).

Actors: System (grader), Student, Instructor.

Preconditions: Assignment has test cases and a rubric.

Postconditions: ‘`test_results`’ recorded; student and instructor views updated.

Primary flow:

1. Grader executes test suite; for each test record status, logs, and duration.
2. System computes an aggregate score and stores `test_results`.

Alternate / error flows:

- Test harness crash → tests marked errored; instructor receives failure diagnostics.
- Non-deterministic test behavior → flagged as flaky.

Acceptance criteria:

- Structured `test_results` exist for all tests.
- Student view shows visible test outcomes and aggregate score; hidden test outputs affect score but are not displayed.
- Score matches rubric computation.

Test procedure:

- Submit code that passes known tests → verify `test_results` and computed score.

- Submit code designed to crash harness → assert error handling path.

FR-06: AI-generated Contextual Hints

Statement: The system shall provide up to three progressive, non-solution hints for failing tests using server-hosted pre-trained models and deterministic rule-based fallbacks. Hints must be audit-logged and avoid revealing full solutions.

Actors: Student, System (AI service), Instructor.

Preconditions: Hint feature enabled for the assignment; hint model or fallback available.

Postconditions: Hints delivered and logged.

Primary flow:

1. Student requests hint (or system suggests after repeated failures).
2. System checks cache, queries AI model on cache miss, or uses deterministic fallback; returns hint(s) and logs action.

Alternate / error flows:

- External AI unavailable or slow → deterministic fallback hints returned.
- Repeated hint requests → progressive escalation of hint specificity, constrained to never provide a full solution.

Acceptance criteria:

- Hints delivered within SLA (cache hit ≤ 4 s; AI path ≤ 6 s under demo load).
- Hints are stored in audit logs with metadata.
- Content checks ensure no verbatim full solutions.

Test procedure:

- Trigger hint generation from a failing submission → measure latency and inspect audit log.
- Mock AI provider failure → verify fallback hint delivered within fallback SLA.

FR-07: Similarity / Plagiarism Check

Statement: The system shall compute similarity scores between submissions for the same assignment and language using token/embedding techniques, surface pairwise similarity scores and highlighted matching spans to instructors, and support configurable thresholds for flagging.

Actors: System (similarity engine), Instructor, Admin.

Preconditions: Submission corpus indexed and similarity engine operational.

Postconditions: Similarity records persisted; instructor flags available.

Primary flow:

1. New submission triggers similarity analysis against corpus; system computes composite score and highlights candidate spans.
2. If the score exceeds the configured threshold, submission is flagged.

Alternate / error flows:

- Corpus scale causes deferred batch analysis; preliminary score visible with “pending” status.
- Cross-language comparisons excluded or handled per configuration.

Acceptance criteria:

- Similarity scores stored and visible to instructors.
- Highlighted spans correspond to detected similarities.
- Thresholding flags as configured.

Test procedure:

- Submit known copied pair → assert score exceeds threshold and highlights matching spans.

- Scale test: simulate a larger corpus and verify batch processing behavior.

FR-08: Peer Review Workflow

Statement: The system shall support anonymized peer review: assign peer reviewers, collect rubric scores and comments, aggregate peer scores, and present anonymized feedback to authors.

Actors: Student (reviewer & reviewee), Instructor, System.

Preconditions: Peer review enabled and students enrolled.

Postconditions: Peer reviews stored; aggregated peer score available.

Primary flow:

1. Instructor configures peer review parameters (anonymity, number of reviews).

2. System assigns reviewers and notifies them; reviewers submit rubric scores and comments.

3. System aggregates results and displays anonymized feedback to authors.

Alternate / error flows:

- Reviewer no-show → system reassigns or marks as missing.

- Instructor overrides aggregation results.

Acceptance criteria:

- Reviewer anonymity preserved from reviewee view.

- Expected number of reviews collected per submission.

- Aggregation calculation documented and reproducible.

Test procedure:

- Simulate assignment with peer review → verify anonymity and aggregation correctness.

FR-09: Personalized Remediation Path

Statement: The system shall recommend 1–3 remediation resources (exercises, readings, examples) for students who exhibit repeated failure patterns; recommendations are recorded and made viewable to instructors.

Actors: Student, Instructor, System.

Preconditions: Mapping of failure patterns to remediation resources available (curated or configurable).

Postconditions: Recommendations presented and logged.

Primary flow:

1. The system detects a failure pattern (e.g., repeated failure of the same test).

2. The system selects appropriate remediation resources and presents them to the student; records events.

Alternate / error flows:

- No curated remediation available → present generic help and flag for instructor addition.

Acceptance criteria:

- Recommendations issued when configured pattern thresholds are met.

- Logs contain recommendation details (resource, timestamp).

Test procedure:

- Induce repeated failures → verify recommendation issuance and instructor-facing log.

FR-10: Instructor Dashboard & Analytics

Statement: The system shall provide instructors with dashboards that display pass rates, submission counts, common failing tests, similarity heatmaps, and downloadable CSV exports; dashboard data shall be filterable by assignment, date range, and student group.

Actors: Instructor, Admin.

Preconditions: Submission and grading data available.

Postconditions: Dashboard renders requested data and exports are produced.

Primary flow:

1. Instructor opens dashboard and applies filters; visuals and tables update.
2. Instructor exports selected data as CSV.

Alternate / error flows:

- Very broad queries trigger background export job with notification when ready.

Acceptance criteria:

- Visualizations correspond to the selected filters.
- CSV export contains expected columns and data.

Test procedure:

- Populate sample data and validate dashboard metrics and CSV export content.

FR-11: Versioned Submission History

Statement: The system shall retain all submission versions, expose visual diffs, and permit instructors to re-run any historical version in the sandbox for audit or re-grading.

Actors: Student, Instructor.

Preconditions: Versioning enabled; archive retention policy configured.

Postconditions: Historical versions accessible and re-runnable.

Primary flow:

1. Student submits multiple versions; system stores version metadata.
2. Instructor views version list, inspects diffs, and triggers re-grade of an historical version.

Alternate / error flows:

- Large files may produce summarized diffs.

Acceptance criteria:

- Version list and diffs available for each submission.
- Re-grading of historical versions queues jobs with correct archived content.

Test procedure:

- Submit two distinct versions → validate diff view and re-grade queueing.

FR-12: Assignment Scheduling & Deadlines

Statement: The system shall enforce release and due dates, apply configurable late policies (reject, penalty, instructor override), and allow instructor-level per-student overrides.

Actors: Instructor, Student, Admin.

Preconditions: Assignment scheduling configured with timezone context.

Postconditions: Submissions accepted/rejected per policy and overrides logged.

Primary flow:

1. Assignment visible at release date; submissions accepted until due date.
2. Late policy is enforced automatically; instructors may override per student.

Alternate / error flows:

- Instructor override creates a new effective due date for selected student(s).

Acceptance criteria:

- System enforces configured late policy in automated tests.
- Overrides applied and auditable.

Test procedure:

- Simulate submissions at times relative to due date and confirm policy behavior.

FR-13: Notifications & Email Simulation

Statement: The system shall emit in-app notifications for relevant events (grading completion, deadlines, peer review assignments). For the prototype, email sending is simulated and persisted in a viewable log for admins.

Actors: Student, Instructor, Admin.

Preconditions: Notification subsystem and log available.

Postconditions: Notifications created and persisted; simulated emails logged.

Primary flow:

1. Event triggers notification creation and log entry.
2. User views notification in UI.

Alternate / error flows:

- Notification backlog → system marks delivery as delayed and retries.

Acceptance criteria:

- Notifications are created and visible to intended recipients.
- Simulated email logs contain entries corresponding to triggered events.

Test procedure:

- Trigger grade event and verify notification and email log entry.

FR-14: Admin Controls & Export

Statement: Administrators shall be able to manage users (CRUD), assign roles, and export grades and similarity data as CSV. All admin exports and user management actions shall be audit-logged.

Actors: Admin.

Preconditions: Admin authenticated and authorized.

Postconditions: Admin actions applied and recorded.

Primary flow:

1. Admin accesses user management dashboard to create/update/delete users.
2. Admin requests data export; system generates CSV and logs the export event.

Alternate / error flows:

- Bulk import errors produce row-level failure reports.

Acceptance criteria:

- Admin CRUD functions operate as expected; exports download and include required fields.
- Audit log entries created for each admin action.

Test procedure:

- Perform admin user create/update/delete and verify DB changes and audit entries.
- Execute export and verify CSV content and audit log

FR-15: Submission Auto-Save & Draft Recovery

Statement:
The system shall automatically save in-progress student submissions as drafts at regular intervals and allow students to recover the latest draft after interruption.

Actors: Student, System.

Preconditions:
Student authenticated; assignment open for submission.

Postconditions:
The latest draft persisted and was recoverable.

Primary flow:

1. Students begin editing or uploading code.
2. System auto-saves draft every *N* seconds or on meaningful change.
3. Student resumes submission after refresh/crash and restores draft.

Alternate / error flows:

- Storage failure → warning shown; manual save option offered.

Acceptance criteria:

- Drafts saved without explicit user action.
- Draft recovery restores last auto-saved content.

Test procedure:

- Begin submission → refresh page → verify draft restored.

Why this matters:
Prevents student data loss, reduces support incidents, and dramatically improves UX with minimal backend changes (simple versioned draft table or cache).

FR-16: Submission Validation & Pre-Run Static Checks

Statement:
The system shall perform lightweight, language-specific pre-submission validation (syntax check, forbidden imports, file structure) before enqueueing grading jobs.

Actors: Student, System.

Preconditions:
Submission received.

Postconditions:
Submission either accepted for grading or rejected with actionable feedback.

Primary flow:

1. Students submit code.
2. The system runs fast static checks (≤1 s).
3. If valid, submission is queued for grading.

Alternate / error flows:

- Syntax error → submission rejected with error location.
- Forbidden API/import → submission rejected with policy message.

Acceptance criteria:

- Invalid submissions never enter grading queue.
- Errors are precise and non-revealing.

Test procedure:

- Submit code with syntax error → assert rejection before sandbox.

Why this matters:
Reduces grader load, improves student feedback speed, and is easy to implement using language parsers or linters.

FR-17: Regrade-on-Rubric or Test Update

Statement:
The system shall allow instructors to trigger selective or bulk re-grading of submissions when rubrics or test cases are updated.

Actors: Instructor, System.

Preconditions:
Assignment exists with submissions.

Postconditions:
Selected submissions re-queued and re-graded.

Primary flow:

1. Instructor updates rubric or test cases.
2. Instructor selects re-grade scope (all / subset).
3. System queues re-grading jobs and tracks old vs new scores.

Alternate / error flows:

- Large batch → background job with progress indicator.

Acceptance criteria:

- Historical grades preserved for audit.
- New grades clearly marked as re-graded.

Test procedure:

- Update rubric → trigger re-grade → verify score change.

Why this matters:
Essential for fairness, auditability, and instructor trust. Implementation reuses existing grading queue logic.

FR-18: Rate Limiting & Abuse Protection

Statement:
The system shall enforce configurable rate limits on login attempts, submissions, hint requests, and similarity checks.

Actors: System, Student, Admin.

Preconditions:
Rate limits configured.

Postconditions:
Excessive requests throttled or blocked.

Primary flow:

1. The user performs an action.
2. The system checks rate limit counters.
3. Action allowed or rejected with retry information.

Alternate / error flows:

- Repeated violations → temporary account cooldown logged.

Acceptance criteria:

- Rate limits enforced consistently per endpoint.
- Legitimate usage unaffected.

Test procedure:

- Exceed submission attempts → assert throttling.

Why this matters:
Protects infrastructure, prevents brute force and hint abuse, and is trivial with middleware or Redis-backed counters.

FR-19: System Health & Grading Transparency Status

Statement:
The system shall expose a read-only system status view indicating grader availability, queue backlog, and recent incidents.

Actors: Student, Instructor, Admin.

Preconditions:
Monitoring data available.

Postconditions:
Users informed of system health.

Primary flow:

1. The user opens the system status page.
2. The system displays grader status and queue depth.

Alternate / error flows:

- Monitoring unavailable → degraded status shown.

Acceptance criteria:

- Status reflects real grader conditions.
- Students are informed of delays proactively.

Test procedure:

- Simulate grading backlog → verify status update.

Why this matters:
Reduces confusion, support load, and student anxiety. Implementation is lightweight (aggregated metrics + UI).

3.2 Use Case Model

3.2.1 Use Case Diagram

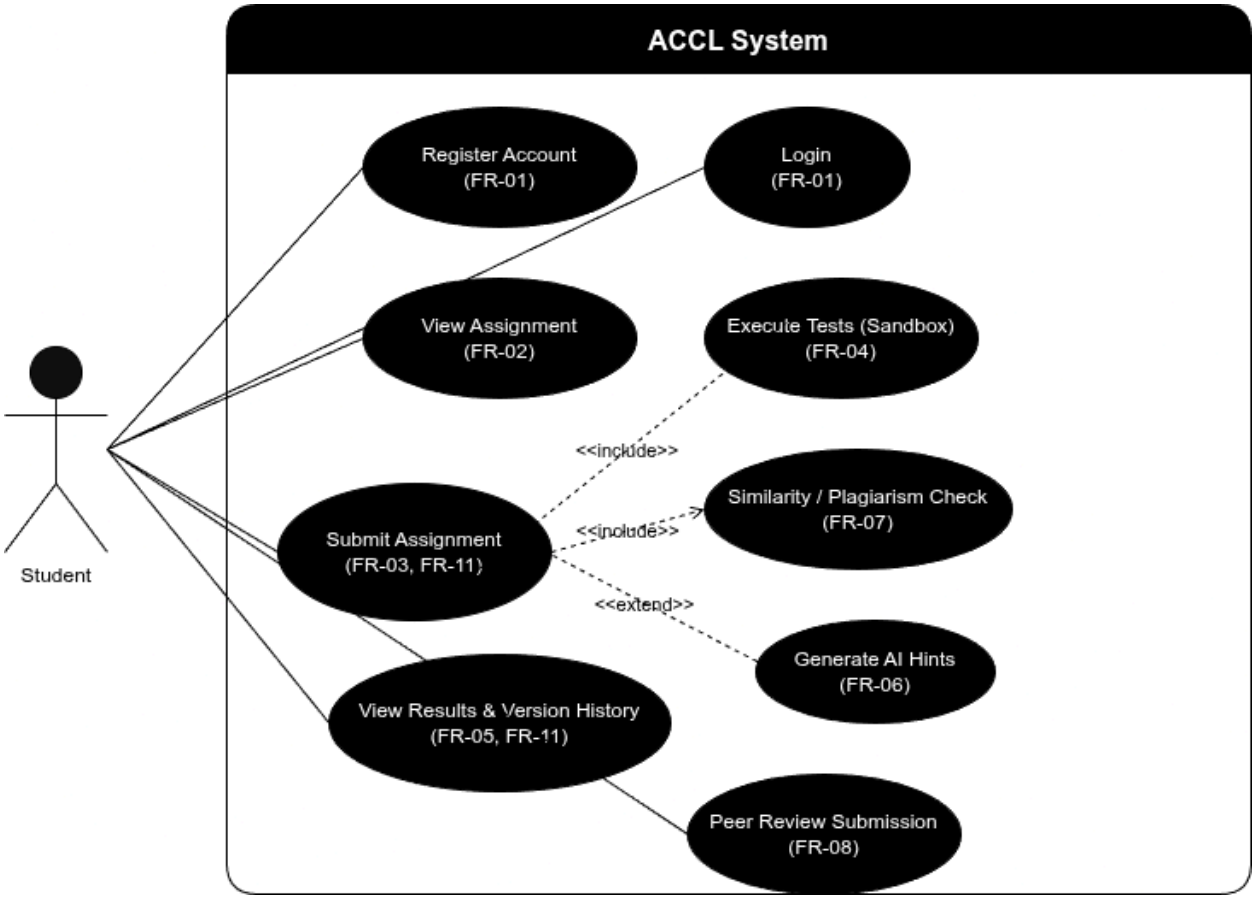


Figure 1. Use Case Diagram – Student Interaction

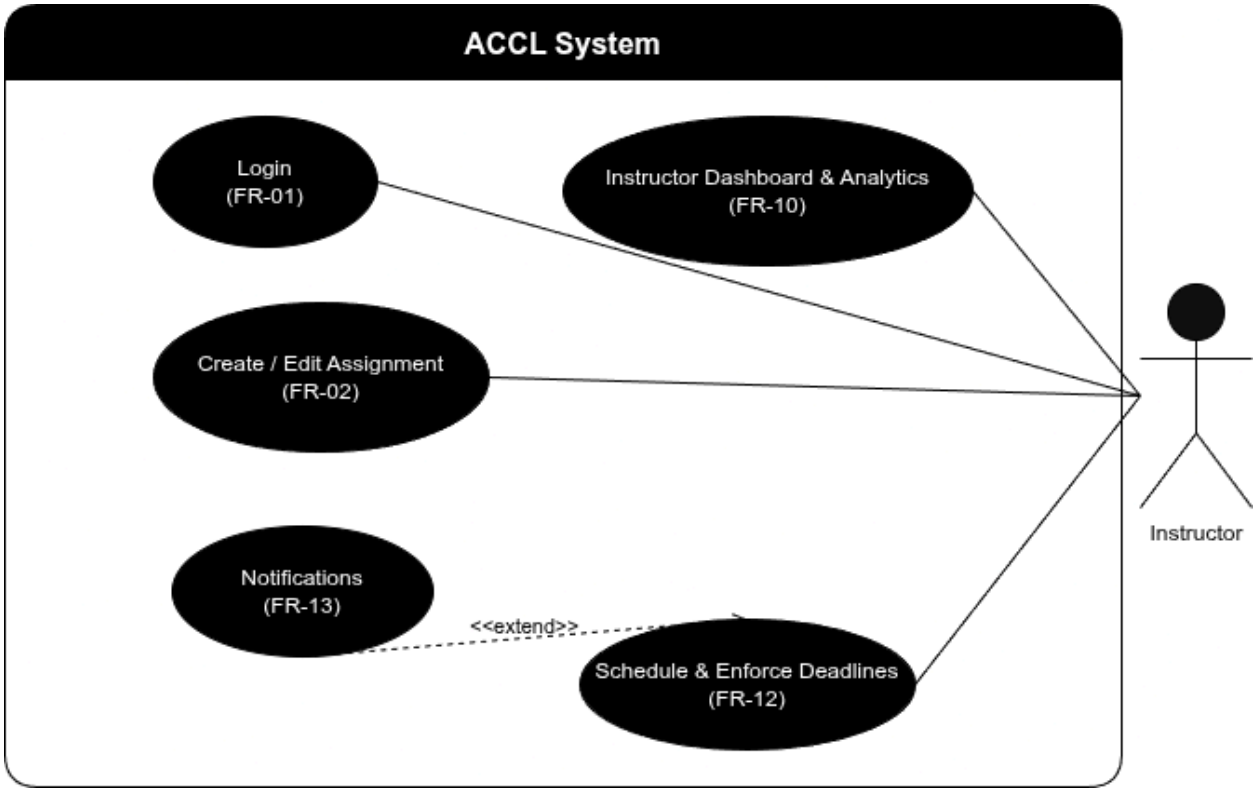


Figure 2. Use Case Diagram – Instructor Interaction

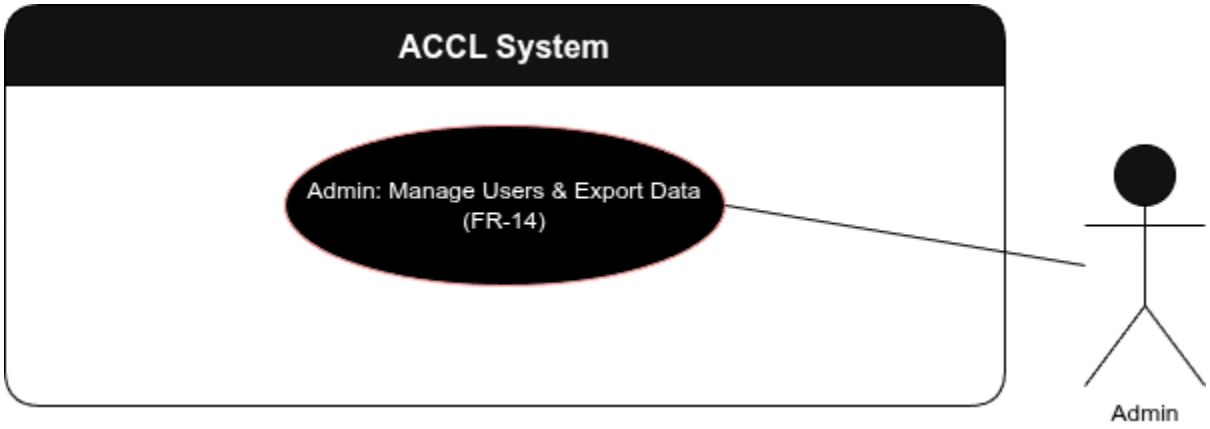


Figure 3. Use Case Diagram – Admin Interaction

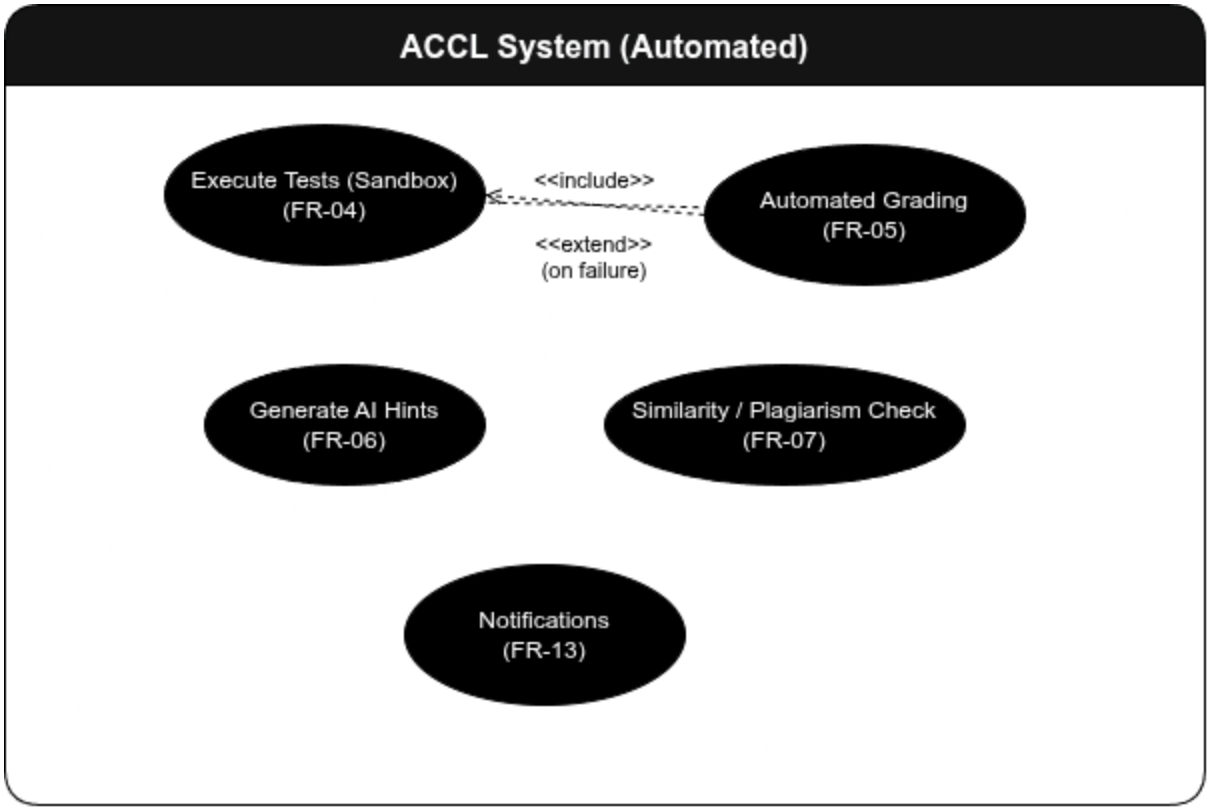


Figure 4. Use Case Diagram – AI/Automation Interaction

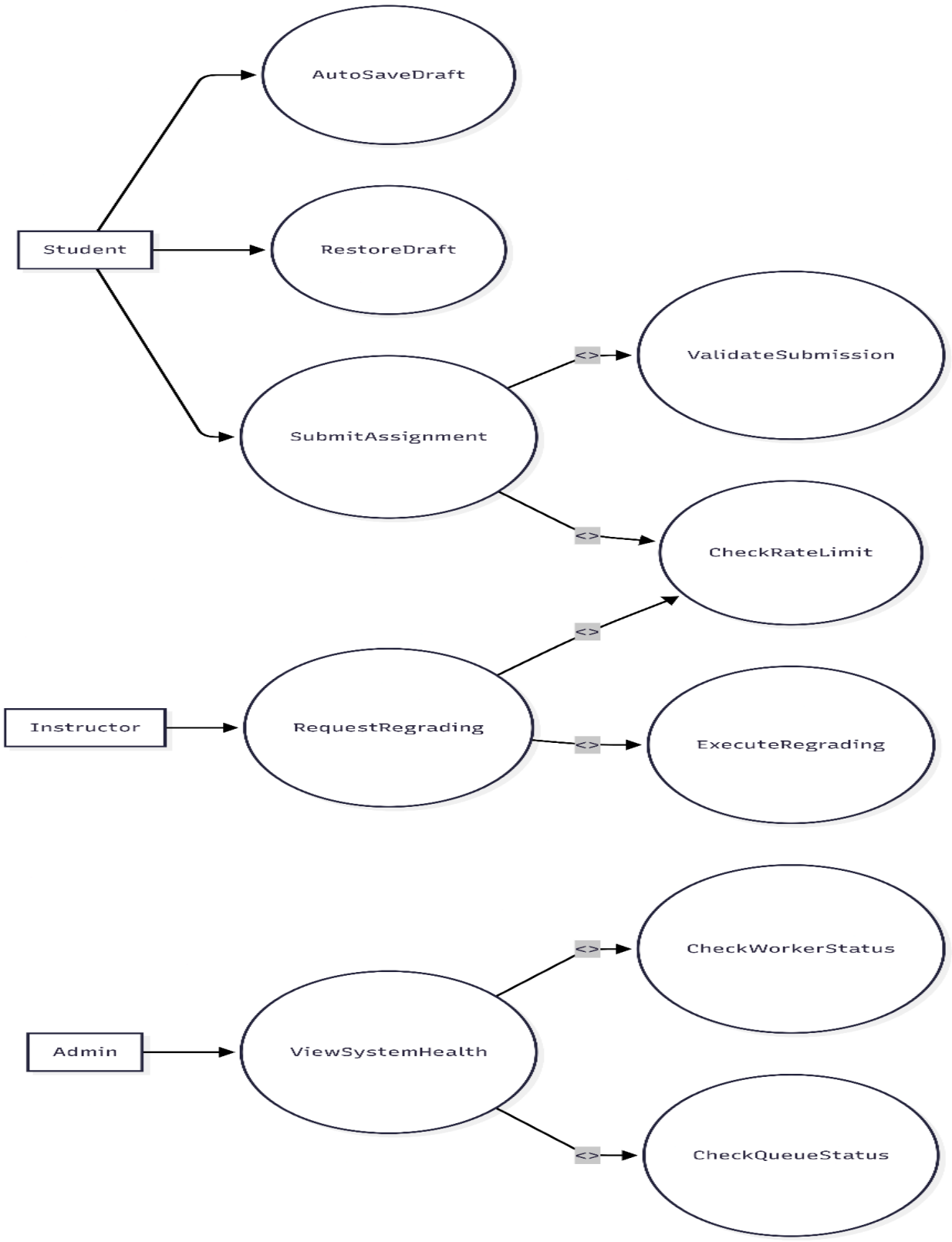


Figure 5. Use Case Diagram

3.2.2 Use Case Descriptions:

Template Section	Purpose / Notes
Use Case ID	UC01
Use Case Name	Register Account
Primary Actor	Student/Instructor/Admin
Stakeholders and Interests	• Student: Wants to create an account to access educational resources and submit assignments securely. • Instructor: Needs an account to manage assignments and monitor student progress. • Admin: Requires an account to oversee user management and system configurations. • University System: Ensures only legitimate users are registered to maintain data integrity and security.
Preconditions	• The registration form is accessible via the web interface. • A valid email address is provided for verification. • Role selection options (Student, Instructor, Admin) are available and restricted based on system policies.
Postconditions	• A new user account is created with a hashed password and assigned role (success). • User is redirected to login page with a confirmation message. • If failure occurs (e.g., duplicate email), an error is displayed, and no account is created.
Trigger	User navigates to the registration page and submits the form.
Main Success Scenario	1. User accesses the registration page. 2. User enters personal details: name, email, password, and selects role. 3. System validates input: checks email uniqueness, password strength (e.g., minimum 8 characters, includes uppercase, number), and valid role. 4. System hashes the password using a secure algorithm. 5. System stores the user data in the database. 6. System displays a success message and redirects to the login page.
Extensions	• 3a: Email already exists → System displays "Email in use" error and suggests login or password reset. • 3b: Password too weak → System displays error with specific

	guidelines (e.g., "Must include at least one uppercase letter and one number") and prompts re-entry. • 3c: Invalid role selected → System restricts options to valid roles and displays "Invalid role" error if tampered. • 5a: Database error → System logs the error, displays a generic "Registration failed, try again" message, and rolls back changes.
Includes / Extends	• Includes: "Validate Input Data" (a sub-use case for checking email and password validity). • Extends: None.
Special Requirements	• Password hashing must use a secure, salted algorithm to prevent breaches. • Email validation must confirm format (e.g., regex check) but no immediate verification email sent in prototype. • UI must be accessible with ARIA labels for form fields.
Assumptions	• User has a stable internet connection to access the web form. • The database is available and properly configured for write operations. • Roles are predefined and no additional approval workflow is needed for registration in the prototype.
Priority	High
Frequency of Use	Occasional (once per user, or rarely for new accounts).

Template Section	**Purpose / Notes**
Use Case ID	UC02
Use Case Name	Login
Primary Actor	Student/Instructor/Admin
Stakeholders and Interests	• Student: Wants quick and secure access to submit assignments and view feedback. • Instructor: Needs access to create assignments and review analytics. • Admin: Requires access to manage users and export data. • University System: Protects against unauthorized access to maintain confidentiality and integrity.
Preconditions	• User account exists in the system with valid credentials.

	• The login page is accessible. • No active session for the user.
Postconditions	• User session is established, and user is redirected to role-specific dashboard (success). • Error message displayed and access denied if credentials invalid (failure).
Trigger	User submits login credentials on the login page.
Main Success Scenario	1. User navigates to the login page. 2. User enters email and password. 3. System validates credentials: compares hashed password and checks account status. 4. System creates a secure session token. 5. System redirects user to their role-based dashboard (e.g., student to assignment list).
Extensions (Alternative Flows)	• 3a: Invalid credentials → System displays "Incorrect email or password" error and increments failed attempt counter. • 3b: Account locked (after 5 failed attempts) → System displays "Account locked, contact admin" and logs the event. • 3c: Role mismatch or inactive account → System denies access with specific error and logs for audit.
Includes / Extends (Relationships)	• Includes: "Validate Credentials" (reused from registration for password comparison). • Extends: "Forgot Password" (optional extension if user clicks the link during login).
Special Requirements	• Session must expire after 30 minutes of inactivity. • Brute-force protection: Limit to 5 attempts per IP/hour. • HTTPS enforced for credential transmission.
Assumptions	• User remembers their credentials or uses password manager. • System clock is synchronized for session timing. • No multi-factor authentication in prototype phase.
Priority	High
Frequency of Use	Daily (multiple times per session for active users).

Template Section	Purpose / Notes
Use Case ID	UC03
Use Case Name	Create/Edit Assignment
Primary Actor	Instructor
Stakeholders and Interests	• Instructor: Wants to define programming exercises with test cases for student evaluation. • Student: Benefits from clear, well-structured assignments. • University System: Ensures assignments align with course objectives and are securely stored.
Preconditions	• Instructor is logged in with appropriate role. • Database is available for storing assignment data.

	• For edit: Existing assignment ID is provided.
Postconditions	• New or updated assignment is stored with associated test cases and rubrics (success). • Changes are reflected in student views post-release (if applicable). • Error if validation fails, no changes saved.
Trigger	Instructor selects "Create New Assignment" or "Edit Existing" from dashboard.
Main Success Scenario	1. Instructor opens the assignment editor form. 2. Instructor enters details: title, description, supported languages, release/due dates. 3. Instructor adds test cases: inputs, expected outputs, timeouts, visibility (hidden/visible). 4. Instructor defines rubrics for partial credit or peer review. 5. System validates all fields (e.g., dates logical, at least one test case). 6. System saves the assignment to the database. 7. System confirms save and returns to dashboard.
Extensions (Alternative Flows)	• 5a: Missing required fields (e.g., no test cases) → System highlights errors and prevents save. • 5b: Invalid dates (due before release) → System displays "Invalid date range" error. • 3a: Adding hidden test → System marks as non-visible to students in results.
Includes / Extends (Relationships)	• Includes: "Validate Assignment Data" (for field checks). • Extends: "Schedule Release" (optional for setting automated release).
Special Requirements	• Support for multiple test cases (up to 20 in prototype). • Dates must handle timezones (e.g., UTC storage). • UI must allow dynamic addition of test case fields.
Assumptions	• Instructor has prepared test cases in advance. • No real-time collaboration on assignment editing. • Assignment size limits (e.g., description < 10KB) enforced implicitly..
Priority	High
Frequency of Use	Weekly (per course assignment cycle).

Template Section	Purpose / Notes
Use Case ID	UC04
Use Case Name	View Assignment
Primary Actor	Student/Instructor
Stakeholders and Interests	• Student: Wants to understand requirements and deadlines for submission. • Instructor: Needs to preview assignments before release. • University System: Ensures content is displayed securely without leaks.
Preconditions	• User is logged in. • Assignment exists in the system. • For student: Assignment is released or user is instructor.
Postconditions	• Assignment details are displayed including description,

	deadlines, and submission interface. • No changes to data occur.
Trigger	User selects an assignment from the list or dashboard.
Main Success Scenario	1. User clicks on assignment link. 2. System checks access: released for student or instructor role. 3. System retrieves and displays: title, description, deadlines, test cases (visible only), rubrics. 4. For students: Displays submission form/editor.
Extensions (Alternative Flows)	• 2a: Assignment not released (student) → System shows "Upcoming Assignment" placeholder with release date. • 2b: No access (e.g., wrong course) → System displays "Access Denied" error. • 3a: No visible test cases → System notes "Hidden tests will be run."
Includes / Extends (Relationships)	• Includes: "Check Access Permissions" (role and release check). • Extends: None.
Special Requirements	• Responsive UI for mobile viewing. • Load time under 2 seconds for standard assignments. • Sanitize description for HTML safety.
Assumptions	• Assignment data is up-to-date in the database. • User has appropriate course enrollment (simulated in prototype). • No caching issues for dynamic content.
Priority	Medium
Frequency of Use	Multiple times daily (students checking deadlines).

Template Section	**Purpose / Notes**
Use Case ID	UC05
Use Case Name	Submit Assignment
Primary Actor	Student
Stakeholders and Interests	• Student: Wants to upload code solutions easily and track versions. • Instructor: Receives submissions for grading and review. • University System: Ensures submissions are stored securely and timestamped.
Preconditions	• Student is logged in. • Assignment is open (released and within deadline). • Submission form is accessible.
Postconditions	• New submission version is stored and queued for grading (success). • Confirmation displayed with version number.

	• Rejection if invalid (e.g., late).
Trigger	Student presses the "Submit" button on the assignment page.
Main Success Scenario	1. Student opens the submission editor on the assignment page. 2. Student pastes code or uploads a file, selects language. 3. System validates: code not empty, language supported, within deadline. 4. System stores code with metadata: timestamp, version (incremented), user/assignment ID. 5. System queues grading job. 6. System displays confirmation and updates submission history.
Extensions (Alternative Flows)	• 3a: Past deadline → System rejects with "Submission closed" (or flags late per policy). • 3b: Invalid file format/language → System displays error and prompts correction. • 4a: Storage failure → System rolls back and notifies user to retry.
Includes / Extends (Relationships)	• Includes: "Validate Submission Data" (checks format and deadline). • Extends: "View Submission History" (after submit, optional view diffs).
Special Requirements	• Support file uploads up to 1MB. • End-to-end latency under 6 seconds for submission. • Versioning must handle up to 50 resubmissions per assignment.
Assumptions	• Student code is in text format (no binaries in prototype). • Queue system is operational for job processing. • No concurrent submissions from same user (handled sequentially).
Priority	High
Frequency of Use	Multiple times per assignment (resubmissions).

Template Section	Purpose / Notes
Use Case ID	UC06
Use Case Name	Execute Tests in Sandbox
Primary Actor	Automated Worker
Stakeholders and Interests	• Student: Expects fair and secure evaluation of code. • Instructor: Relies on accurate test results for grading. • University System: Prevents security risks from untrusted code.
Preconditions	• Submission is stored and queued. • Assignment test cases are available. • Sandbox environment is configured.
Postconditions	• Test results (pass/fail, outputs) are stored per test case. • Submission status updated to "Processed" or "Errored".
Trigger	Worker dequeues a grading job from the queue.

Main Success Scenario	1. Worker retrieves submission code and test cases. 2. Worker spawns an isolated sandbox container. 3. For each test case: Inject input, execute code, capture stdout/stderr/exit code/runtime. 4. Enforce limits: timeout per test (5s), memory (256MB), no network. 5. Worker stores results in database. 6. Worker cleans up sandbox and proceeds to grading.
Extensions (Alternative Flows)	• 4a: Timeout or resource violation → Mark test as errored, log details, kill process. • 4b: Code crash (non-zero exit) → Capture stderr and proceed to next test. • 2a: Sandbox spawn failure → Retry once, then mark submission errored.
Includes / Extends (Relationships)	• Includes: "Enforce Resource Limits" (core security step). • Extends: None.
Special Requirements	• Sandbox must use Docker with network disabled and read-only filesystem. • Logs must be sanitized for sensitive data. • Per-test isolation to prevent state carryover.
Assumptions	• Code is executable in supported language (Python in prototype). • Test cases are well-formed by instructor. • Worker has sufficient resources to spawn containers.
Priority	Critical
Frequency of Use	Per submission (high during deadlines).

Template Section	**Purpose / Notes**
Use Case ID	UC07
Use Case Name	Automated Grading
Primary Actor	Automated Worker
Stakeholders and Interests	• Student: Receives accurate scores and feedback. • Instructor: Gets reliable grading data for analytics. • University System: Ensures consistent evaluation.
Preconditions	• Test results from sandbox execution are available. • Assignment rubrics and weights are defined.
Postconditions	• Aggregate score is computed and stored. • Submission updated with final status and score.
Trigger	Completion of sandbox test execution.
Main Success Scenario	1. Worker aggregates test results: count passed/failed, including hidden tests. 2. Apply weights: e.g., visible tests 60%, hidden 40%.

	3. Compute total score as percentage of max points. 4. Apply partial credit via rubrics if defined. 5. Update submission record with score and status. 6. Trigger notifications and hint generation if failures.
Extensions (Alternative Flows)	• 3a: No tests passed → Score 0, flag for review. • 4a: Rubric not applicable → Use default pass/fail scoring. • 1a: Incomplete results (e.g., errored tests) → Partial score, mark as "Incomplete".
Includes / Extends (Relationships)	• Includes: "Apply Rubric Weights" (for partial credit). • Extends: "Generate AI Hints" (if failures occur).
Special Requirements	• Scoring must be deterministic and auditable. • Handle floating-point precision for runtimes. • Unit tests required for scoring logic.
Assumptions	• Test results are accurate from sandbox. • Weights sum to 100% as set by instructor. • No manual overrides in automated flow.
Priority	High
Frequency of Use	Per submission execution.

Template Section	**Purpose / Notes**
Use Case ID	UC08
Use Case Name	Generate AI Hints
Primary Actor	Automated Worker
Stakeholders and Interests	• Student: Gets helpful, non-spoiler hints for learning. • Instructor: Provides educational feedback without manual effort. • University System: Enhances learning while maintaining fairness.
Preconditions	• Failing test results are available. • Pre-trained AI model API is configured and accessible.
Postconditions	• Up to 3 hints per failing test are generated and stored. • Hints cached for similar failures.
Trigger	Detection of failing tests during grading.

Main Success Scenario	1. Worker identifies failing tests and builds context: error snippet, test input (if safe), code excerpt. 2. Check cache for matching failure fingerprint. 3. If cache miss: Query pre-trained model with prompt for progressive hints. 4. Store hints with tags (e.g., syntax, logic) and confidence. 5. Associate hints with submission and test results. 6. Make hints available for student view.
Extensions (Alternative Flows)	• 3a: Model latency high or unavailable → Use rule-based fallback hints (e.g., common error templates). • 2a: Cache hit → Retrieve and serve cached hints immediately. • 4a: Low confidence hint → Discard and retry or fallback.
Includes / Extends (Relationships)	• Includes: "Build Failure Context" (context assembly). • Extends: None.
Special Requirements	• Hints must not reveal solutions; prompt engineered for nudges. • Response time ≤4s average. • Audit log model calls for cost tracking.
Assumptions	• Pre-trained model provides reliable, educational responses. • Failure fingerprints are unique enough for caching. • No privacy issues in sending anonymized snippets to model.
Priority	Medium
Frequency of Use	Per failing submission (variable).

Template Section	**Purpose / Notes**
Use Case ID	UC09
Use Case Name	Similarity/Plagiarism Check
Primary Actor	Automated Worker
Stakeholders and Interests	• Student: Ensures fair competition. • Instructor: Identifies potential plagiarism for review. • University System: Upholds academic integrity policies.
Preconditions	• New submission is stored. • Prior submissions for the same assignment exist.
Postconditions	• Similarity scores and flags are stored. • Instructor notified if high threshold met.
Trigger	Post-submission processing after storage.
Main Success Scenario	1. Worker normalizes code: strip comments, whitespace,

	optional identifier rename. 2. Compute embeddings or token n-grams. 3. Compare against existing submissions (same assignment/language). 4. Calculate composite score (e.g., cosine + Jaccard). 5. If score ≥ threshold (0.85 high, 0.70 suspicious), flag and generate highlighted diffs. 6. Store flags and evidence in database.
Extensions (Alternative Flows)	• 4a: Low score → No flag, store score only for analytics. • 5a: Matches boilerplate → Downweight common code sections. • 3a: No priors → Skip check, mark as first submission.
Includes / Extends (Relationships)	• Includes: "Normalize Code" (preprocessing step). • Extends: "Notify Instructor" (if flagged).
Special Requirements	• Thresholds configurable by admin. • Privacy: Flags visible only to instructors. • Accuracy tests for composite scoring.
Assumptions	• Embeddings model is pre-trained and efficient. • Submissions are text-based code only. • False positives handled by instructor review.
Priority	Medium
Frequency of Use	Per new submission.

Template Section	**Purpose / Notes**
Use Case ID	UC10
Use Case Name	Peer Review Workflow
Primary Actor	Student (Reviewer)/Instructor
Stakeholders and Interests	• Student (Reviewer): Provides feedback and learns from peers. • Student (Reviewee): Receives constructive reviews. • Instructor: Aggregates peer input for grading. • University System: Promotes collaborative learning.
Preconditions	• Instructor has opened a peer review window for the assignment. • Eligible submissions are anonymized and assigned.
Postconditions	• Reviews are stored, anonymized, and aggregated. • Feedback released to reviewees after window closes.

Trigger	Instructor opens review window, or student accesses assigned reviews.
Main Success Scenario	1. Instructor configures and opens review window (e.g., 2 reviews per submission). 2. System anonymizes submissions and assigns to reviewers. 3. Reviewer opens assigned submission (view-only). 4. Reviewer fills rubric: scores (e.g., correctness 0-5), comments. 5. System stores review and checks completion. 6. After window: Aggregate scores and release feedback.
Extensions (Alternative Flows)	• 5a: Incomplete rubric → Prompt for missing fields. • 2a: Insufficient reviewers → System notifies instructor to extend window. • 6a: Deadline missed → Reassign unfinished reviews.
Includes / Extends (Relationships)	• Includes: "Anonymize Submission" (privacy step). • Extends: "Aggregate Peer Scores" (for final grading).
Special Requirements	• Rubric must support min length for comments. • Anonymity preserved until release. • UI for rubric must be intuitive with progress tracking.
Assumptions	• Students are motivated to complete reviews. • Assignment has enough submissions for pairing. • No conflicts of interest in assignments.
Priority	Medium
Frequency of Use	Per assignment with peer review enabled (occasional).

Template Section	**Purpose / Notes**
Use Case ID	UC11
Use Case Name	Personalized Remediation
Primary Actor	Automated Worker/Student
Stakeholders and Interests	• Student: Gets tailored suggestions to improve skills. • Instructor: Supports student learning without extra effort. • University System: Improves overall educational outcomes.
Preconditions	• Failure patterns from submissions are logged. • Curated remediation resources are available.
Postconditions	• 1-3 suggestions are generated and displayed to student. • Suggestions logged for instructor analytics.

Trigger	Detection of repeated failure patterns post-grading.
Main Success Scenario	1. Worker analyzes submission history: identify patterns (e.g., same error type across versions). 2. Map patterns to curated bundles: exercises, readings, templates. 3. Optionally augment with AI-recommended resources (pre-trained). 4. Store suggestions linked to submission. 5. Display in student results view.
Extensions (Alternative Flows)	• 2a: No matching patterns → Provide generic advice (e.g., "Review basics"). • 3a: AI unavailable → Use only curated mappings. • 1a: First failure → Defer until patterns emerge.
Includes / Extends (Relationships)	• Includes: "Analyze Failure Patterns" (pattern detection). • Extends: None.
Special Requirements	• Suggestions must be relevant and non-repetitive. • Limit to 3 per submission to avoid overload. • Test mapping accuracy with sample data.
Assumptions	• Curated resources are predefined by instructors. • Patterns are detectable from error types/stderr. • Student views results to see suggestions.
Priority	Low
Frequency of Use	Per failing submission with history.

Template Section	**Purpose / Notes**
Use Case ID	UC12
Use Case Name	Instructor Dashboard & Analytics
Primary Actor	Instructor
Stakeholders and Interests	• Instructor: Monitors class performance and identifies issues. • Student: Indirectly benefits from data-driven improvements. • University System: Supports reporting and compliance.
Preconditions	• Instructor is logged in. • Data (submissions, results) exists for the course.
Postconditions	• Analytics visuals and exports are generated. • No data modifications occur.

Trigger	Instructor opens the dashboard page.
Main Success Scenario	1. Instructor navigates to dashboard. 2. System queries DB for aggregates: pass rates, submission counts, failing tests. 3. Display charts: heatmaps for similarity, bar graphs for pass rates. 4. Allow filters: by assignment, student, date range. 5. Provide export option: CSV of grades and flags.
Extensions (Alternative Flows)	• 2a: No data → Show placeholders or "No data yet" message. • 4a: Invalid filter → Reset to defaults with error toast. • 5a: Large export → Queue as background job, notify when ready.
Includes / Extends (Relationships)	• Includes: "Query Analytics Data" (DB aggregation). • Extends: "Export CSV" (optional download).
Special Requirements	• Load time under 3 seconds with caching. • Charts responsive and interactive (e.g., hover tooltips). • Maintainability: Use cached queries for heavy ops.
Assumptions	• Data is current (no real-time needed in prototype). • Instructor has access to all course data. • Export format fixed as CSV.
Priority	Medium
Frequency of Use	Daily/Weekly (monitoring progress).

Template Section	**Purpose / Notes**
Use Case ID	UC13
Use Case Name	Schedule & Enforce Deadlines
Primary Actor	Instructor/Automated Worker
Stakeholders and Interests	• Instructor: Controls assignment timelines. • Student: Knows clear deadlines for planning. • University System: Enforces policy compliance.
Preconditions	• Assignment exists. • Dates are set during creation/edit.
Postconditions	• Deadlines are enforced on submissions. • Late policies applied (e.g., deny or penalty).

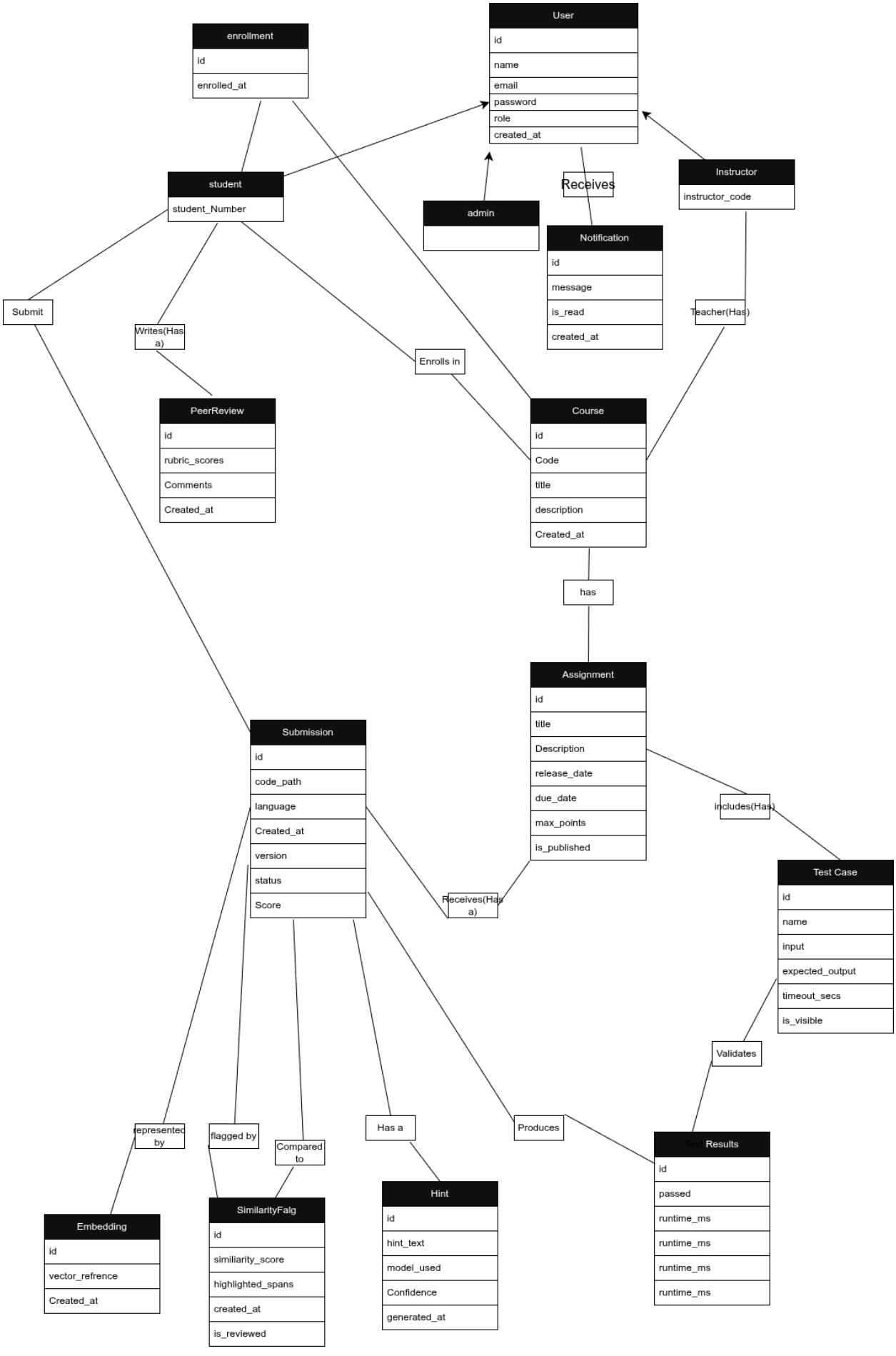
Trigger	Instructor sets dates, or student attempts submit.
Main Success Scenario	1. Instructor sets release and due dates in editor. 2. System validates: release before due, future dates. 3. System stores dates. 4. On submit attempt: Worker checks current time vs. due date. 5. If within: Proceed; else apply policy (e.g., block). 6. Log enforcement for audit.
Extensions (Alternative Flows)	• 4a: Past due, policy allow with penalty → Accept, apply score reduction. • 2a: Invalid dates → Error prompt to correct. • 1a: Extension requested → Instructor updates due date, notifies students.
Includes / Extends (Relationships)	• Includes: "Validate Dates" (logical checks). • Extends: "Notify Students" (on changes).
Special Requirements	• Handle timezones (store in UTC, display local). • Automated checks on submit. • Test for edge cases like midnight deadlines.
Assumptions	• System time is accurate. • Policies predefined (deny default). • No daylight saving issues in prototype.
Priority	Medium
Frequency of Use	Per assignment setup and submission.

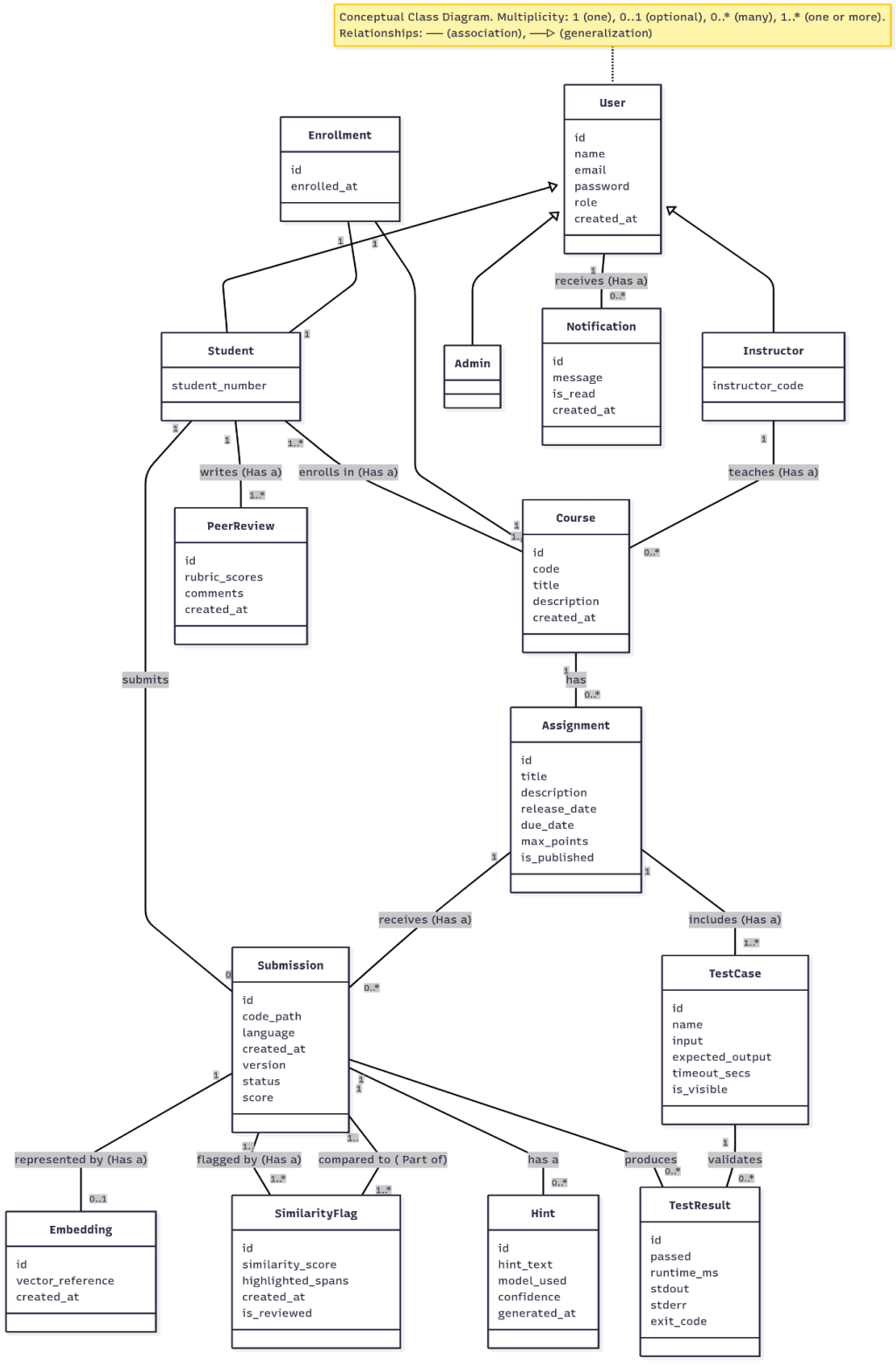
Template Section	**Purpose / Notes**
Use Case ID	UC14
Use Case Name	Notifications & Admin Controls
Primary Actor	System/Admin
Stakeholders and Interests	• User (all roles): Stays informed of events. • Admin: Manages users and data exports. • University System: Ensures communication and oversight.
Preconditions	• Event occurs (e.g., grading done) or admin logged in. • Notification preferences set (default in-app).
Postconditions	• Notification delivered and logged. • Admin actions (user edit, export) completed.

Trigger	System event or admin action initiation.
Main Success Scenario	1. System detects event (e.g., grading complete). 2. System generates notification message. 3. Deliver in-app (visible in user center); simulate/log email. 4. For admin: Open controls, edit users (CRUD), generate exports. 5. System applies changes and logs audit. 6. Confirm action to admin.
Extensions (Alternative Flows)	• 3a: User prefs disable email → In-app only. • 4a: Large export → Queue job, notify when ready. • 5a: Invalid edit (e.g., duplicate user) → Error and rollback.
Includes / Extends (Relationships)	• Includes: "Log Audit Trail" (for all actions). • Extends: "Export CSV" (specific admin function).
Special Requirements	• Immediate delivery for in-app notifications. • Audit logs secure and tamper-proof. • Exports include filters for data selection.
Assumptions	• Email simulation sufficient for prototype (no real SMTP). • Admin has superuser privileges. • Notifications not spammy (rate-limited).
Priority	Medium
Frequency of Use	Per event (frequent for notifications, occasional for admin).

3.3 Domain Model

3.3.1 Conceptual Class Diagram


User
id
name
email
password
role
created_at
enrollment
id
enrolled_at
student
student_Number
admin
Receives
Instructor
instructor_code
Notification
id
message
is_read
created_at
Submit
Writes(Has a)
Enrolls in
Teacher(Has)
PeerReview
id
rubric_scores
Comments
Created_at
Course
id
Code
title
description
Created_at
has
Assignment
id
title
Description
release_date
due_date
max_points
is_published
Submission
id
code_path
language
Created_at
version
status
Score
Receives(Has a)
includes(Has)
Test Case
id
name
input
expected_output
timeout_secs
is_visible
Validates
represented by
flagged by
Compared to
Has a
Produces
Test Results
id
passed
runtime_ms
runtime_ms
runtime_ms
runtime_ms
Embedding
id
vector_refrence
Created_at
SimilarityFalg
id
similiarity_score
highlighted_spans
created_at
is_reviewed
Hint
id
hint_text
model_used
Confidence
generated_at


Conceptual Class Diagram. Multiplicity: 1 (one), 0..1 (optional), 0..* (many), 1..* (one or more).
Relationships: —— (association), ——▷ (generalization)
User
id
name
email
password
role
created_at
Enrollment
id
enrolled_at
receives (Has a)
Notification
id
message
is_read
created_at
Admin
Student
student_number
Instructor
instructor_code
writes (Has a)
enrolls in (Has a)
teaches (Has a)
PeerReview
id
rubric_scores
comments
created_at
Course
id
code
title
description
created_at
submits
has
Assignment
id
title
description
release_date
due_date
max_points
is_published
receives (Has a)
includes (Has a)
Submission
id
code_path
language
created_at
version
status
score
TestCase
id
name
input
expected_output
timeout_secs
is_visible
represented by (Has a)
flagged by (Has a)
compared to (Part of)
has a
produces
validates
Embedding
id
vector_reference
created_at
SimilarityFlag
id
similarity_score
highlighted_spans
created_at
is_reviewed
Hint
id
hint_text
model_used
confidence
generated_at
TestResult
id
passed
runtime_ms
stdout
stderr
exit_code

3.3.2 Class Descriptions

Class Name	Description	Key Attributes	Associations
User	Represents a general registered individual in the ACCL system who can interact with the platform based on their role, such as accessing courses, managing assignments, or administering users. This is the base entity for all user types, encapsulating common identification and authentication details to ensure secure and personalized access to educational resources like coding exercises and feedback mechanisms.	id, name, email, password, role, created_at	Generalizes to Student (inheritance), Generalizes to Instructor (inheritance), Generalizes to Admin (inheritance), Receives Notification (0..* from User to Notification)
Student	Represents a learner enrolled in the ACCL platform who participates in courses by viewing assignments, submitting code solutions for automated evaluation, receiving AI-generated hints and remediation suggestions, and engaging in peer reviews to foster collaborative learning and skill improvement in programming.	student_number	Inherits from User (generalization), Enrolls in Course (1.. Students to 1.. Courses via Enrollment), Submits Submission (1 Student to 0.. Submissions), Writes PeerReview (1 Student to 1.. PeerReviews)
Instructor	Represents a teaching professional in the ACCL system responsible for creating and managing courses, designing programming assignments with test cases, monitoring student submissions through analytics, reviewing similarity flags for plagiarism,	instructor_code	Inherits from User (generalization), Teaches Course (1 Instructor to 0..* Courses)

	and facilitating peer review workflows to guide student learning and ensure academic integrity.		
Admin	Represents a system administrator in the ACCL platform who oversees user management, role assignments, data exports, and overall platform configurations to maintain operational efficiency, security, and compliance with educational policies without directly involvement in teaching or learning activities.	(No additional attributes beyond User)	Inherits from User (generalization)
Course	Represents an educational unit in the ACCL system that groups related programming topics, assignments, and enrolled students under an instructor, providing a structured environment for learning, submission deadlines, and performance tracking to support adaptive code education.	id, code, title, description, created_at	Has Enrollment (0.. Courses to 1.. Enrollments), Taught by Instructor (0.. Courses to 1 Instructor), Has Assignment (1 Course to 0.. Assignments)
Enrollment	Represents the registration of a student in a specific course within the ACCL platform, tracking participation to enable access to assignments, submissions, and notifications, thereby facilitating targeted educational interactions and progress monitoring.	id, enrolled_at	Associates Student and Course (1 Enrollment to 1 Student, 1 Enrollment to 1 Course)
Assignment	Represents a programming exercise or task created by an instructor in the ACCL system, defining requirements, deadlines, test cases for automated grading, and maximum points to evaluate student code submissions, integrate AI hints, and support peer reviews for comprehensive feedback.	id, title, description, release_date, due_date, max_points, is_published	Belongs to Course (0.. Assignments to 1 Course), Includes TestCase (1 Assignment to 1.. TestCases), Receives Submission (1 Assignment to 0..* Submissions)
TestCase	Represents a predefined evaluation criterion	id, name, input, expected_outpu	Included in Assignment (1..

	for an assignment in the ACCL platform, specifying inputs, expected outputs, and constraints to automatically test student code submissions in a secure sandbox, contributing to grading accuracy and hint generation for failing cases.	t, timeout_secs, is_visible	TestCases to 1 Assignment), Validates TestResult (1.. TestCases to 0..* TestResults)
Submission	Represents a student's code solution uploaded or pasted for an assignment in the ACCL system, capturing versions, execution status, and scores after sandbox testing, while enabling similarity checks, hint provision, and peer reviews to promote iterative learning and originality.	id, code_path, language, created_at, version, status, score	Submitted by Student (0.. Submissions to 1 Student), Received by Assignment (0.. Submissions to 1 Assignment), Produces TestResult (1 Submission to 0.. TestResults), Has Hint (1 Submission to 0.. Hints), Flagged by SimilarityFlag (1.. Submissions to 1.. SimilarityFlags), Compared to SimilarityFlag (1.. Submissions to 1.. SimilarityFlags), Represented by Embedding (1 Submission to 0.. Embeddings)
TestResult	Represents the outcome of executing a specific test case on a student submission in the ACCL platform, recording pass/fail status, runtime, and outputs to inform grading, hint generation, and remediation paths for targeted student improvement.	id, passed, runtime_ms, stdout, stderr, exit_code	Produced by Submission (0.. TestResults to 1 Submission), Validated by TestCase (0.. TestResults to 1 TestCase)
Hint	Represents AI-generated or rule-based contextual feedback provided for failing test results in a submission within the ACCL	id, hint_text, model_used, confidence, generated_at	Associated with Submission (0..* Hints to 1 Submission)

	system, offering progressive nudges to help students debug and learn without revealing solutions, thereby enhancing adaptive learning experiences.		
SimilarityFlag	Represents a detected potential plagiarism indicator between submissions in the ACCL platform, storing similarity scores and highlighted code spans for instructor review to uphold academic integrity while allowing for false positive resolutions.	id, similarity_score, highlighted_spans, created_at, is_reviewed	Flags Submission (1.. SimilarityFlags to 1.. Submissions), Compared to Submission (1.. SimilarityFlags to 1.. Submissions)
PeerReview	Represents feedback provided by a student on another anonymized submission in the ACCL system, including rubric scores and comments to support collaborative evaluation, skill development, and aggregated input for final grading.	id, rubric_scores, comments, created_at	Written by Student (1..* PeerReviews to 1 Student)
Notification	Represents system-generated alerts or messages sent to users in the ACCL platform, informing them of events like grading completion, deadlines, or peer review assignments to keep participants engaged and informed throughout the learning process.	id, message, is_read, created_at	Received by User (0..* Notifications to 1 User)
Embedding	Represents a vectorized representation of a submission's code in the ACCL system, used for efficient similarity computations and plagiarism detection to enable scalable comparisons across multiple student works.	id, vector_reference, created_at	Represents Submission (0..1 Embeddings to 1 Submission)

3.4 Non-Functional Requirements

NFR-01 — Security & Isolation (Sandbox)

Requirement: The sandbox shall enforce per-test timeout ≤ **5 seconds**, block all outbound network egress, and restrict filesystem access to the submission workspace. All sandbox security events shall be logged and retained for audit.

Rationale: Prevent code from exfiltrating data, accessing host resources, or performing denial-of-service.

Test approach: Automated sandbox security tests executed for each grader image and in CI: network block, filesystem isolation, and timeout enforcement.

Tools / Environment: Container runtime (Docker/Firecracker), automated test harness (pytest), OS tools for network/socket probing.

Success criteria:

- Network egress attempts from sandbox fail (validated by connecting to known public endpoint).
- Attempts to read files outside workspace return permission or file-not-found errors.
- Long-running processes are terminated at ≤ 5 s and logged.
- Security tests pass in CI for each grader deployment.

NFR-02 — Performance & Latency

Requirement: Define and meet measured latency targets:

- Median end-to-end grading + rule-based hint ≤ **4 s** (serial/demo).
- Average end-to-end with external AI provider ≤ **6 s** under the defined demo load.
- Under **10 concurrent** grading jobs, p95 latency ≤ **12 s**. System must degrade gracefully and fall back to deterministic hints if external AI is slow/unavailable.

Rationale: Provide timely feedback that supports learning workflow.

Test approach: Synthetic load testing and benchmarking using a defined demo dataset and documented hardware profile. Include AI provider simulation for normal and degraded behavior.

Tools / Environment: Locust/JMeter for load; Prometheus/Grafana for metrics; mock AI provider.

Success criteria:

- Benchmark report demonstrating achieved median/mean/p95 metrics with environment details.
- Fallback hints returned within documented fallback SLA when AI provider unavailable.

NFR-03 — Usability & Accessibility

Requirement: Core workflows (register → submit → view results) must be completable within **≤ 3 clicks per step**. UI must be responsive at common mobile widths and include basic accessibility attributes (ARIA labels) for core controls.

Rationale: Ensure the platform is approachable and accessible to a broad student population.

Test approach: Manual workflow click counts and automated accessibility scans (baseline checks).

Tools / Environment: Browser developer tools for responsive testing; axe-core or equivalent for automated accessibility checks.

Success criteria:

- Manual verification: core workflows completed within the click budget on desktop and mobile viewport.
- Automated accessibility scan reports no critical missing labels for core interactive elements.

NFR-04 — Maintainability

Requirement: Codebase shall follow modular architecture (Flask Blueprints + repository pattern), maintain a CI pipeline that runs tests on PRs, and contain at least **five** unit tests covering core modules at prototype delivery.

Rationale: Facilitate collaboration, review, and reliable evolution of the system.

Test approach: Repository inspection, CI verification, and unit test execution.

Tools / Environment: CI service (GitHub Actions/GitLab CI), coverage reporting.

Success criteria:

- Repository structure adheres to agreed pattern.
- CI configured and passing on PRs.
- Unit tests present and passing; test report included in deliverables.

NFR-05 — Privacy & Data Access

Requirement: Student code and similarity detail visibility shall be restricted to the submitting student and authorized instructors. Exports of grades/similarity are restricted to Admin role. All accesses and exports must be audit-logged (user id, timestamp, resource/action).

Rationale: Protect student privacy and provide traceability for sensitive operations.

Test approach: Access control tests and audit log verification.

Tools / Environment: Automated API tests (pytest), log storage (DB or log service).

Success criteria:

- Unauthorized access attempts receive 403 and are logged.
- Admin export action produces an audit log entry with user id, timestamp, and export metadata.
- Logs are queryable and match sample audit queries.

3.5 External Interface Requirements

3.5.1 User Interface

- **Pages:** Login/Register, Dashboard (role dependent), Assignment Editor, Assignment View, Submission Editor/Upload, Submission Results Modal, Version History, Peer Review UI, Admin Panel, Help/FAQ.
- **Design:** Bootstrap 5 for responsive UI, base templates with blocks for extensibility. Provide accessible labels and ARIA attributes for major controls.

3.5.2 Hardware Interface

Standard devices with internet access; no special hardware.

3.5.3 Software Interface

- **DB: SQLite/SQLAlchemy; AI: Pre-trained API wrappers.**

3.5.4 Communication Interface

- **HTTPS**; internal queue for sandbox jobs.

---

4. Appendices

4.1 Appendix A: Data Dictionary

4.2 Appendix B: Glossary

- **Sandbox:** Isolated code runner**.**
- **Embedding:** Code vector for similarity.
- **Rubric:** Review scoring template**.**
