Software Architecture and Design Specification

Project: Academic Performance Analytics System (APAS)

Version: 1.0

Project / Team ID: 24

Authors:

Diya D Bhat - PES2UG23CS183 (Team Lead)

Dhanya Prabhu - PES2UG23CS169

Eshwar R A - PES2UG23CS188

Delisha Riyona Dsouza - PES2UG23CS166

Submission Date: 30/9/2025

Status: Draft

Revision History

Version	Date	Authors	Change summary
1.0	29-09-2025	Diya D Bhat, Dhanya Prabhu, Delisha Riyona Dsouza, Eshwar R A	Architecture and Design

Approvals

Role | Name | Signature/Date

1. Introduction

1.1. Purpose

This document specifies the architecture and design of the Academic Performance Analytics System (APAS). The purpose of APAS is to provide students, instructors, and administrators with secure, role-based dashboards for monitoring academic performance, analyzing trends, and generating predictive insights.

The architecture ensures that:

- Students can view personalized dashboards, progress, and receive risk alerts.
- Instructors can monitor class-level performance, attendance, and generate reports.
- Administrators can access institution-wide analytics, exports, and compliance logs.

The system integrates with institutional LMS platforms for data ingestion, applies data validation and predictive analytics, and supports secure reporting and notifications. The design emphasizes modularity, scalability, and compliance with institutional security and privacy standards.

1.2. Scope

The Academic Performance Analytics System (APAS) provides a secure and scalable platform for monitoring and improving academic performance within educational institutions. The system covers the following functionalities:

- **Authentication & Access Control**: Secure login role-based dashboards for students, instructors, and administrators.
- **Data Ingestion**: Import of academic records (grades, attendance, assignments) from institutional LMS platforms and CSV uploads.
- **Dashboards**: Visualization of academic trends at individual, course, and institutional levels.
- **Predictive Analytics**: Identification of at-risk students through risk scoring and machine learning models.
- **Reporting & Export**: Generation of PDF/CSV reports with anonymization for privacy compliance.
- **Notifications**: In-app alerts and secure email notifications for students and faculty.
- **Audit & Compliance**: Secure logging of user and admin activities to support institutional audits and regulatory requirements.

This scope ensures APAS addresses the needs of students, faculty, and administrators by providing actionable insights, enabling timely interventions, and supporting data-driven decision-making while maintaining compliance with institutional privacy and security standards.

1.3. Audience

This document is intended for all stakeholders involved in the design, development, validation, and operation of the **Academic Performance Analytics System (APAS)**, including:

- **Developers** – to implement system components according to the defined architecture and APIs.
- **QA Engineers** – to design and execute test cases ensuring that functional and non-functional requirements are met.
- **Security Auditors** – to evaluate compliance with institutional data privacy, authentication, and security standards.
- **Instructors and Administrators** – to understand the system's functional capabilities, dashboards, and reporting features.
- **Maintenance & Operations Teams** – to manage system deployment, monitoring, and long-term sustainability.

This ensures that both technical and non-technical stakeholders have a clear understanding of the system's architecture, scope, and design decisions.

1.4. Definitions

APAS	Academic Performance Analytics System
SSO	Single Sign-On
RBAC	Role-Based Access Control
LMS	Learning Management System

2. Document Overview

3.1. How to use this document

This document serves as a reference for understanding the architecture and design of the Academic Performance Analytics System (APAS). It provides the following deliverables:

- **UML Diagrams** – Component and sequence diagrams to illustrate system structure and interactions.
- **Architecture Decision Records (ADRs)** – Documentation of major design choices and their rationale.

- **Threat Models** – Identification of potential security risks and mitigations based on STRIDE methodology.
- **API Design Specifications** – Details of RESTful endpoints for authentication, data ingestion, reporting, and notifications.
- **Error Handling & Logging Guidelines** – Standards for consistent exception handling, secure logging, and monitoring.
- **UX Design Considerations** – Role-based dashboards, accessibility features, and user interaction flows.

Each section can be referenced independently depending on the reader's role. For example:

- **Developers** should focus on component diagrams, APIs, and design decisions.
- **QA/Test Engineers** should align testing with functional flows and error handling guidelines.
- **Security Auditors** should review the threat models and security architecture.
- **Stakeholders (students, instructors, administrators)** may review the UX and reporting features for functional clarity.

This ensures that the document acts as both a blueprint for implementation and a reference guide for validation and compliance.

3.2. Related Documents

The following documents should be referenced alongside this Software Architecture and Design (SAD) document to gain a complete understanding of the **Academic Performance Analytics System (APAS)**:

- **Software Requirements Specification (SRS)** – Defines the functional and non-functional requirements of APAS, including authentication, dashboards, predictive analytics, reporting, and notifications.
- **Software Test Plan (STP)** – Outlines the testing strategy, scope, approach, tools, and deliverables to ensure that APAS meets its defined requirements and quality standards.
- **Requirements Traceability Matrix (RTM)** – Maps system requirements from the SRS to corresponding design elements and test cases, ensuring complete coverage and validation.

These documents are complementary: the **SRS defines what** the system must do, the **SAD explains how** it is designed to achieve it, and the **STP/RTM ensure validation** through systematic testing.

3. Architecture

3.1. Goals & Constraints

Goals:

1. **Secure Data Handling** – Protect student grades and attendance data using basic encryption (TLS for communication, AES for storage).
2. **Reliable Access** – Ensure the system is available for students and faculty during working hours with minimal downtime.
3. **Role-Based Dashboards** – Provide separate views for students, instructors, and admins to improve usability.
4. **Early Risk Identification** – Implement a simple predictive model (e.g., logistic regression) to flag at-risk students based on grades and attendance.

Constraints:

1. **Limited Resources** – The system will be hosted on a basic university server or cloud free-tier, which restricts performance and scalability.
2. **Dependency on LMS APIs** – Data ingestion depends on external LMS sandbox/test APIs, which may have limited reliability or availability.
3. **Team Skillset** – The project will be built using technologies the student team is familiar with (Java/Spring Boot, MySQL, Python for ML).
4. **Time Boundaries** – Development, testing, and deployment must be completed within the semester timeline, leaving little room for large-scale features

3.2. Stakeholders & Concern

Stakeholders:

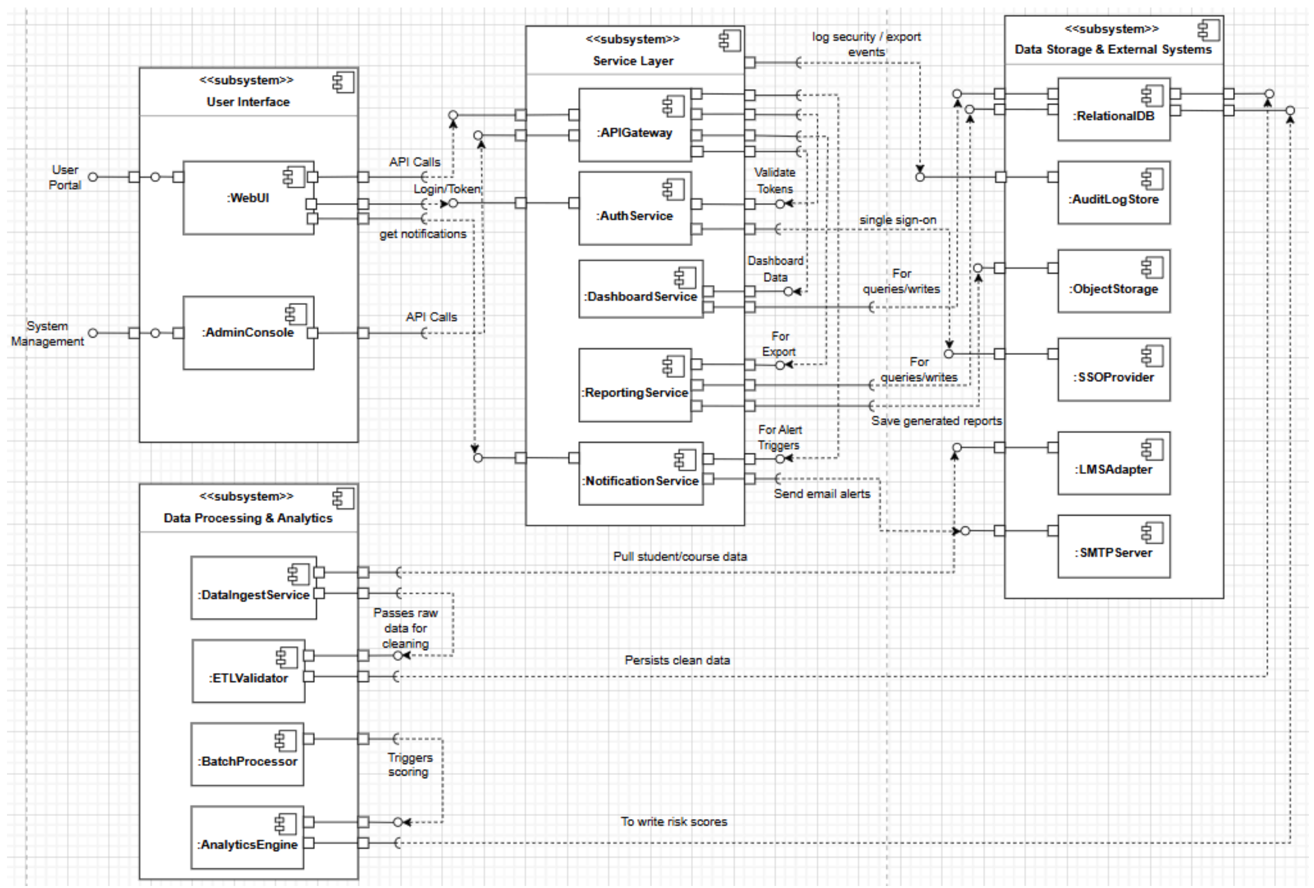
1. **Students** – Use the system to track grades, attendance, and risk alerts for their own performance.
2. **Instructors** – Monitor class progress, identify struggling students, and generate reports.

3. **Administrators** – View institution-wide analytics, compliance reports, and audit logs.
4. **Development & QA Team** – Student developers and testers responsible for building, testing, and maintaining the system as part of the project.

Concerns:

1. **Students** – Want accurate and timely updates on performance, easy-to-use dashboards, and data privacy protection.
2. **Instructors** – Need reliable data ingestion from LMS, clear visualizations, and simple report generation.
3. **Administrators** – Require system compliance with data security policies, and access to anonymized institutional reports.
4. **Development & QA Team** – Concerned with completing the project within the semester timeline, ensuring performance on limited infrastructure, and meeting mentor's evaluation criteria.

3.3. Component (UML) Diagram



3.4. Component Descriptions

User Interface

- **WebUI:** Student/instructor portal for dashboards, reports, and notifications (includes accessibility features & client-side caching).
- **AdminConsole:** Admin portal to manage users, data sources, and monitor system operations.

Service Layer

- **APIGateway:** Unified entry point enforcing TLS, authentication, and routing requests to internal services.
- **AuthService:** Handles login/SSO and validates tokens for secure role-based access.
- **DashboardService:** Serves charts/tables from DB with server-side caching for fast responses.
- **ReportingService:** Generates PDF/CSV exports and applies anonymization before saving to storage.
- **NotificationService:** Sends in-app alerts and email notifications, rate-limited and logged for auditing.

Data Processing & Analytics

- **DataIngestService:** Collects raw LMS/CSV data with retry and queuing for reliability.
- **ETLValidator:** Cleans and normalizes data, raising alerts for repeated errors.
- **BatchProcessor:** Runs scheduled jobs, triggers retraining, and logs system metrics.
- **AnalyticsEngine:** Applies predictive models for risk scoring and stores model artifacts securely.

Data Storage & External Systems

- **RelationalDB:** Stores academic records and transactional data (encrypted at rest).
- **AuditLogStore:** Immutable log for security events, admin actions, and compliance tracking.
- **ObjectStorage:** Holds reports, backups, and ML models with access control.
- **SSOProvider:** External identity provider for institutional single sign-on.

- **LMSAdapter:** Simulated LMS APIs to fetch grades, attendance, and assignments.
- **SMTPServer:** Sends system emails and notifications securely via TLS.

3.5. Chosen Architecture Pattern and Rationale

Layered Architecture chosen for clear separation of concerns as it clearly separates the user interface, services, data processing, and storage. This makes the system easier to understand, test, and maintain. Each layer has a defined role (UI for interaction, services for logic, analytics for scoring, storage for persistence).

Microservices approach rejected since our project scope is moderate and doesn't need the overhead of distributed deployment. A layered design gives modularity and flexibility without unnecessary complexity.

3.6. Technology Stack & Data Stores

Frontend Technologies:

- Streamlit (Python) – main UI framework used for dashboards
- Pandas + Plotly – for interactive charts and data visualizations
- Requests library – secure API communication with backend
- Session management in Streamlit – handles login state and inactivity timeout
- Streamlit Autorefresh – periodic refresh for updated analytics and session expiration
- Responsive layout created using Streamlit columns, containers, and UI widgets

Backend Technologies:

- Python 3.x with Flask – backend framework handling REST API
  Flask-Session – server-side session storage (filesystem-based)
- Flask-CORS – secure CORS enabling cookie-based authentication
- Werkzeug Security – password hashing (PBKDF2)
- ReportLab – PDF generation for admin exports
- MySQL Connector – database connection driver for MySQL
- Fernet Encryption + HMAC (optional) – encryption & deterministic hashing for sensitive data
- TLS Support – HTTPS endpoint using self-signed certificates

Data Storage:

- MySQL 8.0 as primary relational database for academic records, user data, and system metadata
- Filesystem Storage for PDF report and CSV exports, TLS certificates

Analytics & ML:

- Python 3.9 with scikit-learn for predictive analytics and risk scoring models
- Risk scoring is done inside Flask synchronously
- Pandas used for admin analytics summaries

Infrastructure & Security:

- RBAC enforced server-side via decorators
- TLS 1.3 for encrypted HTTPS traffic
- CORS restricted to Streamlit origins only

Monitoring & Logging:

- Audit Log table in MySQL storing all the events happening
- Backend console logs via Python traceback for debugging
- Admin dashboard to view last 500 audit entries

3.7. Risks and Mitigations

Risk Category	***Risk Description***	***Impact***	***Likelihood***	***Mitigation Strategy***
Data Security	Unauthorized access to student academic records	High	Medium	Implement RBAC, encrypt data at rest/transit, regular security audits, multi-factor authentication
System Availability	Service downtime during peak academic periods	High	Low	Load balancing, database replication, automated failover, 99.5% uptime SLA monitoring

Data Quality	Inconsistent or corrupted data from LMS integration	Medium	Medium	Data validation pipelines, error handling with retry mechanisms, data quality monitoring dashboards
Performance	Slow dashboard loading with large datasets	Medium	High	Database indexing, Redis caching, query optimization, pagination for large result sets
Privacy Compliance	FERPA/GDPR violations in data handling	High	Low	Data anonymization for reports, consent management, audit trails, privacy impact assessments
Predictive Model	Inaccurate risk scoring leading to false alerts	Medium	Medium	Cross-validation testing, A/B testing, model retraining schedules, human oversight for critical decisions
Dependency Failure	External LMS API outages affecting data ingestion	Medium	Medium	API circuit breakers, data queuing, fallback to manual CSV uploads, SLA agreements with LMS vendors
Resource Constraints	Limited server capacity during high usage	Low	High	Horizontal scaling with Docker, cloud auto-scaling, resource monitoring, usage analytics

Risk Monitoring:

- Weekly security scans using OWASP ZAP
- Real-time performance monitoring with alerts for response time > 3 seconds
- Daily data quality reports identifying ingestion errors

- Monthly penetration testing by third-party security firm

3.8. Traceability to Requirements

Requirement ID	Requirement Description	Architecture Component	Design Element	Validation Method
FR-001	User authentication via institutional SSO	AuthService, SSOProvider	OAuth 2.0 integration, JWT tokens	Unit tests, SSO integration testing
FR-002	Role-based dashboard access (Student/Instructor/Admin)	WebUI, APIGateway, AuthService	RBAC implementation, route guards	UI testing, permission verification
FR-003	LMS data ingestion (grades, attendance)	DataIngestService, LMSAdapter	REST API clients, data validation	Integration tests with LMS sandbox
FR-004	Predictive analytics for at-risk student identification	AnalyticsEngine, BatchProcessor	ML models (logistic regression), risk scoring	Model accuracy testing, validation datasets
FR-005	PDF/CSV report generation with anonymization	ReportingService, ObjectStorage	Report templates, data masking	Output verification, privacy compliance tests
FR-006	Real-time notifications and alerts	NotificationService, SMTPServer	WebSocket connections, email queues	End-to-end notification testing

FR-007	Audit logging for compliance	AuditLogStore, Elasticsearch	Immutable logs, structured logging	Log integrity verification, query performance
NFR-001	System availability ≥ 99.5%	Load balancers, database replication	Health checks, auto-failover	Uptime monitoring, stress testing
NFR-002	Response time ≤ 3 seconds for dashboards	Redis caching, DashboardService	Query optimization, data aggregation	Performance testing, load testing
NFR-003	Data encryption (TLS 1.3, AES-256)	APIGateway, RelationalDB	SSL certificates, database encryption	Security scanning, encryption verification

3.9 Security Architecture

Threat Modeling (STRIDE Analysis):

Spoofing Identity:

- *Threat:* Unauthorized users impersonating legitimate students/instructors
- *Mitigation:* Secure session cookies, session timeouts, RBAC

Tampering with Data:

- *Threat:* Modification of academic records or predictive model results
- *Mitigation:* Database triggers for audit trails, checksums for data integrity, role-based write permissions, API request signing

Repudiation:

- *Threat:* Users denying actions performed in the system
- *Mitigation:* Immutable audit logs, exporting logs available to admin

Information Disclosure:

- *Threat:* Unauthorized access to sensitive student academic data
- *Mitigation:* AES-256 encryption at rest, TLS 1.3 for data in transit, data masking in non-prod environments, least privilege access

Denial of Service:

- *Threat:* System unavailability during critical academic periods
- *Mitigation:* Lightweight API architecture, simple risk model

Elevation of Privilege:

- *Threat:* Users gaining unauthorized administrative access
- *Mitigation:* RBAC with principle of least privilege, regular permission audits, separation of duties, secure API endpoints

Security Controls Implementation:

Authentication & Authorization:

- Username/password authentication
- Password hashing using Werkzeug
- Auto-expiring session cookies
- Role hierarchy: Student < Instructor < Admin

Data Protection:

- Sensitive fields can be encrypted using Fernet (AES-128)
- Database connection encryption with SSL/TLS
- Deterministic HMAC hashing for IDs where needed

Network Security:

- CORS restricted to Streamlit origins only
- HTTPS supported via self-signed certificate

Compliance Framework:

- FERPA Compliance: Student record access logs, Alerts logged for high-risk students, Audit logs for all access

- GDPR Compliance: Data minimization, Ability to anonymize reports through CSV export, No unnecessary personal data stored

Incident Response:

- Error logging via Stack traces in backend
- Audit logs show recent suspicious behavior
- Ability to monitor access via

Security Testing:

Manual testing of RBAC across all routes

Encryption test suite validates:

- Fernet encryption/decryption
- HMAC consistency
- Invalid token handling
- Key file presence
- TLS certificate validity

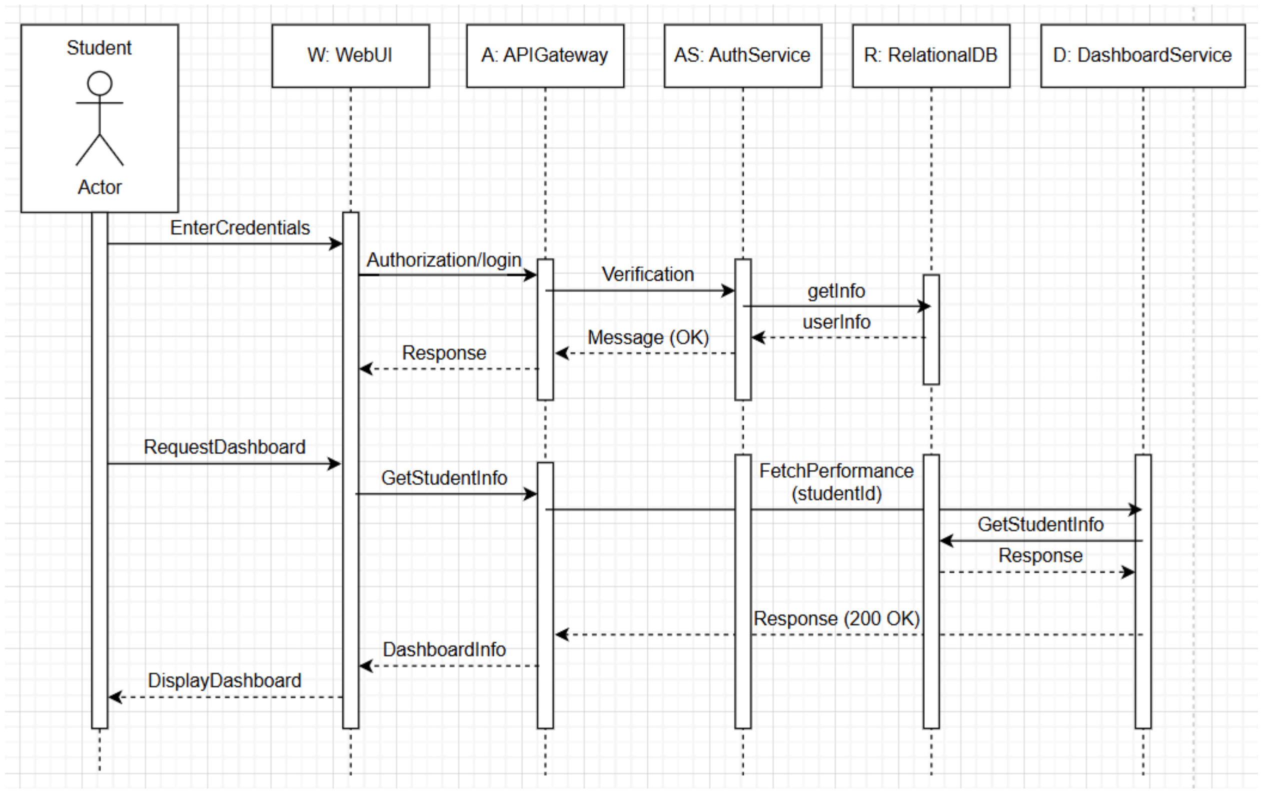
4. Design

4.1 Design Overview

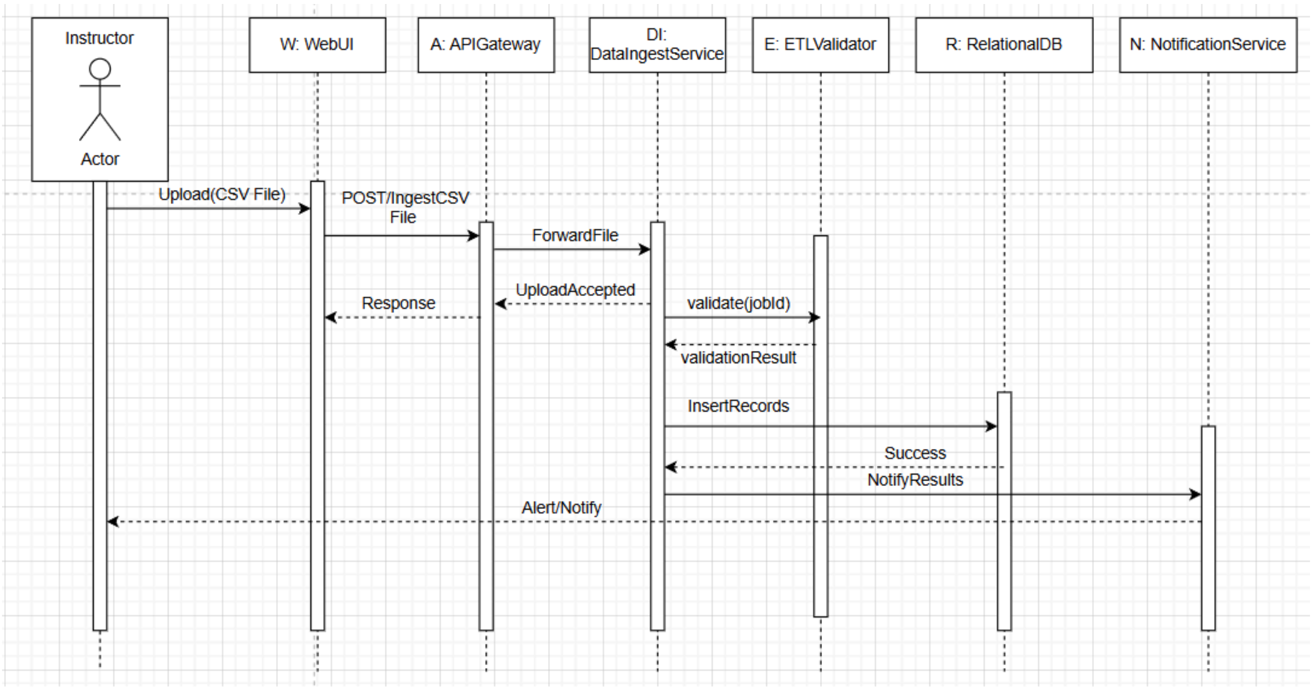
The APAS is designed as a layered, modular web application to enforce separation of concerns, enable independent development/testing, and simplify maintenance

4.2 UML Sequence Diagrams

1) Diagram 1 — Student Login & Dashboard Access



2) Diagram 2 — Instructor Uploads Grades (CSV ingestion)



4.3 API Design

AuthService

- POST /auth/login → Authenticate user.
  - Request: { username, password }
  - Response: { token, roles, expires_in }
  - Errors: 401 Unauthorized, 423 Account Locked
- POST /auth/refresh → Refresh JWT token.
  - Response: { new_token, expires_in }

DataIngestService

- POST /ingest/csv → Upload CSV (grades/attendance).
  - Request: Multipart { file, courseId, term }
  - Response: { status, jobId }
  - Errors: 400 Invalid CSV, 401 Unauthorized, 413 File Too Large
- POST /ingest/api → Push data via LMS API.
  - Request: { courseId, records: [...] }
  - Response: { status, processed, errors }

4.4 Error Handling, Logging & Monitoring

Error Handling: Standard JSON error messages with error codes; no sensitive data in error payloads.

Logging:

- Structured logs (timestamp, component, userId hash, requestId).
- Logged to *AuditLogStore*.
- Sensitive data masked.

Monitoring:

- API latency, dashboard load time, batch job duration.
- Alerts on system availability (<99.5%), ingest failures, or risk scoring errors.
- Metrics tracked: request_duration, ingest_failures, alerts_sent

4.5 UX Design

- Role-based dashboards: Student, Instructor, Admin.
- Accessibility: WCAG 2.1 AA compliance, scalable charts, keyboard navigation.
- Options for high-contrast mode and responsive layout.
- Simple menu structure with consistent navigation.

4.6 Open Issues & Next Steps

Open Issues:

1. Accuracy of the predictive model needs more validation on larger datasets.
2. Stress testing for high user load has not been completed.
3. Data anonymization in exported reports is not fully verified.
4. User acceptance testing (UAT) feedback is still pending.

Next Steps:

1. Run pilot testing with sample student and instructor data.
   Perform basic security and penetration testing.
2. Collect UAT feedback and refine dashboards/reports.
3. Finalize documentation and prepare for project submission.

5. Appendices

5.1 Glossary: APAS, SRS, RTM, SSL, TLS, JWT, PII, WCAG 2.1 AA, RBAC, ADR.

5.2 References: IEEE 42010, OWASP, NIST SP 800-160, WCAG 2.1 AA Guidelines.

5.3 Tools: draw.io