
FH
University of
Applied Sciences
TECHNIKUM
WIEN
Arc42 documentation

Requirements Overview

Name of Software:
Regional University Management System (RUMS)

Main Purpose:

To streamline and improve the management of academic, administrative, and financial activities in the university.

Main Features:

- Comprehensive Student, Faculty, and Course Management.
- Automated Attendance and Grades Management.
- Integrated Billing and Payment Processing.
- Advanced Reporting and Analytics Tools.
- Robust User Authentication and Authorization.

Stakeholder

Role/Name	Expectations
University Administrator (John Smith)	Efficient management of university operations.
Senior Lecturer (Jane Doe)	Streamlined academic and curriculum management.
Chief Information Officer (Michael Lee)	Secure, scalable, and maintainable system architecture.
Student Representative (Emily Chen)	User-friendly interface and access to necessary academic resources.
Financial Officer (Sarah Johnson)	Accurate and efficient financial management and reporting.

Quality Goals

Priority	Quality	Motivation
1	Usability	Ensure a user-friendly experience for all stakeholders.
2	Security	Protect sensitive academic and financial data.
3	Reliability	Provide consistent and dependable system performance.

Solution Strategy

Goal/Requirements	Architectural Approach
Scalable user management	Microservices for handling different user roles and activities.
Secure financial transactions	Secure API integration with external payment gateways.
Real-time data analytics	Utilize big data technologies for advanced reporting and analytics.

Architecture Decisions

Problem	Considered Alternatives	Decision
System Scalability	Monolith vs. Microservices	Chose Microservices for better scalability and maintainability.
Data Storage	SQL vs. NoSQL	Chose NoSQL (MongoDB) for flexibility and performance in handling large datasets.
Client-Side Framework	Angular vs. React	Chose React for its component-based architecture and efficient UI rendering.

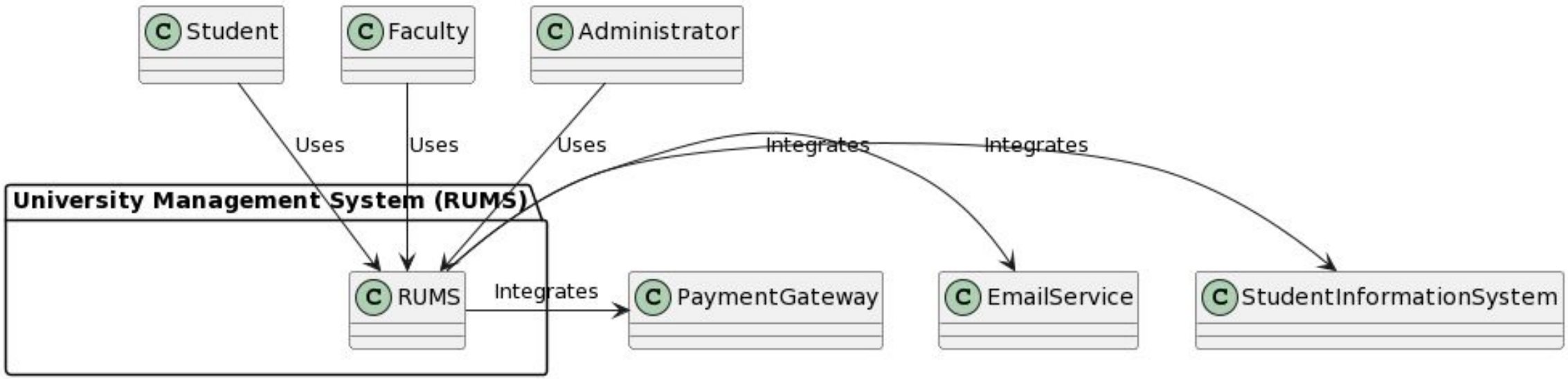
Risks and Technical Debt

Risk/Technical Debt	Description
Integration Complexity	Integrating multiple microservices can lead to complexity.
Data Migration	Migrating existing data to the new system might lead to inconsistencies.
Dependency Management	Managing numerous dependencies in a microservices architecture might increase technical debt.

Architecture Constraints

Constraints	Background and/or motivation
Technology Stack (React, Node.js, MongoDB)	Chosen for their performance, scalability, and community support.
Cloud Infrastructure (AWS/Azure)	Provides scalability, reliability, and advanced services.
Compliance with Data Protection Regulations	Mandatory for protecting user data and privacy.
Integration with Existing Systems	Ensures continuity and minimizes the impact of transition.
Responsive Design	Must be accessible on various devices, enhancing usability.

Business Context



Description:
RUMS is the core system that interacts with various stakeholders and external systems.
It serves students, faculty, and administrators, providing a platform to manage academic and administrative activities.
It integrates with Payment Gateway for financial transactions, Email Service for notifications, and Student Information System for academic data.

Crosscutting Concepts

Development concepts

- Agile methodologies for iterative development and continuous integration/continuous deployment (CI/CD) practices.

Architecture and design patterns

- Microservices architecture for scalability, and MVC pattern for separating concerns in the application.

Safety and security concepts

- Implementation of OAuth for secure authentication, HTTPS for secure communication, and regular security audits.

Decisions

Context	Decision	Consequences
Scalability	We will …adopt a microservices architecture.	More complex orchestration but easier scaling and maintenance.
Data Storage	We will …NoSQL (MongoDB) for our database.	Greater flexibility and performance with large, unstructured datasets.
Frontend Development	We will …use React for the frontend.	Enhanced user interface and component reusability.

Quality Goals and Scenarios

Goal: Ensure system scalability.

- *Scenario* System should handle an increase in user load without performance degradation.

Goal: Maintain high security.

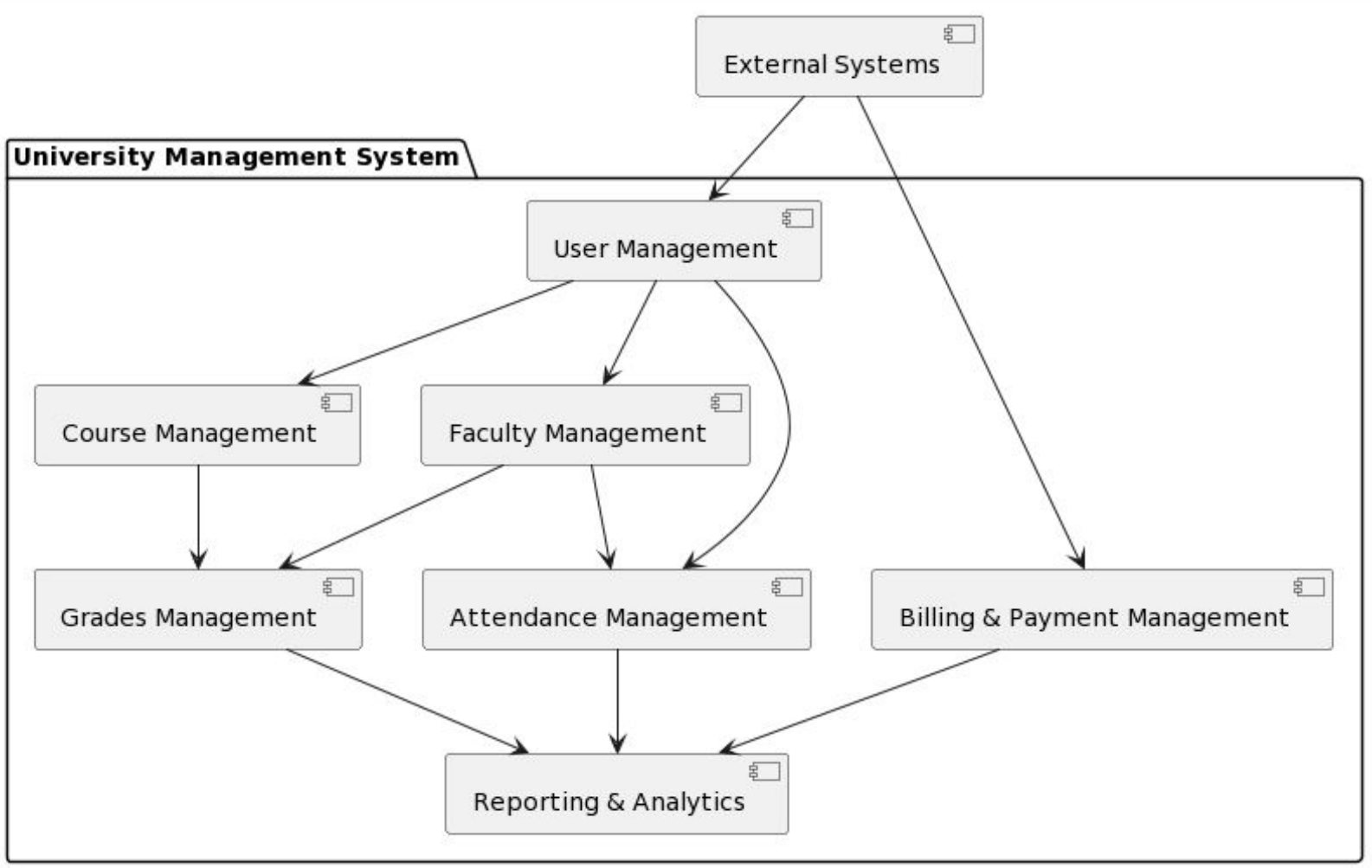
- *Scenario* System should resist XSS and CSRF attacks.

Goal: Ensure data consistency.

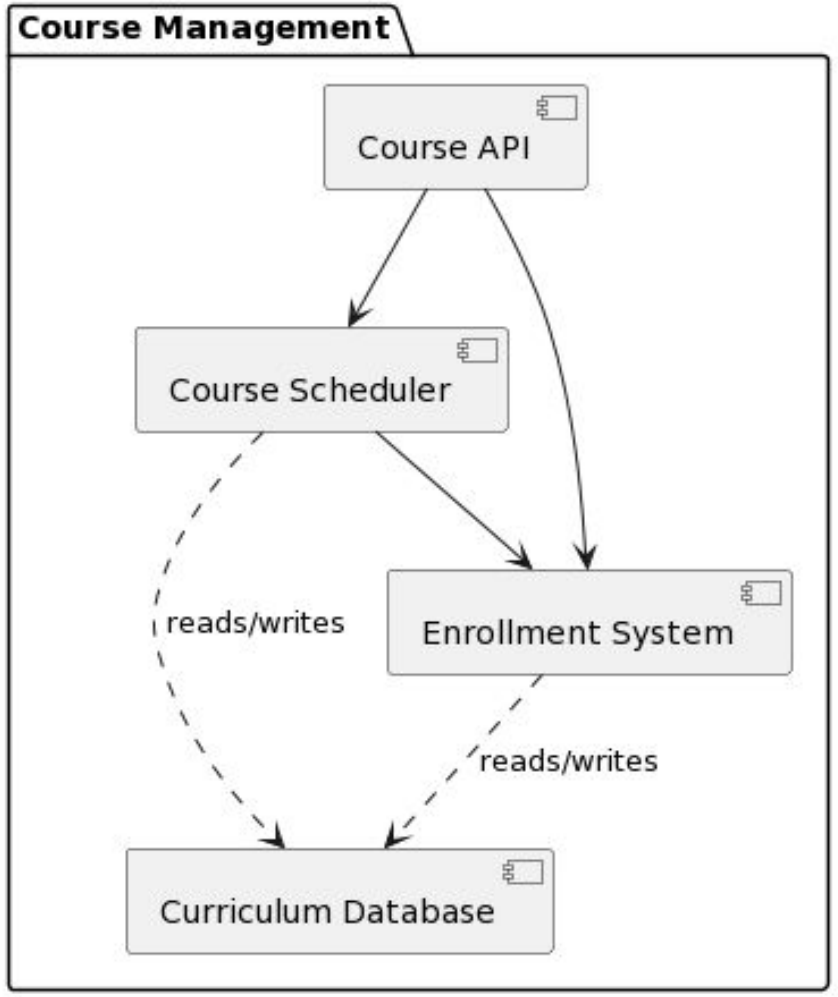
- *Scenario* Updates in one module (e.g., grades) should reflect immediately across all relevant modules (e.g., transcripts, billing).

Quality Tree

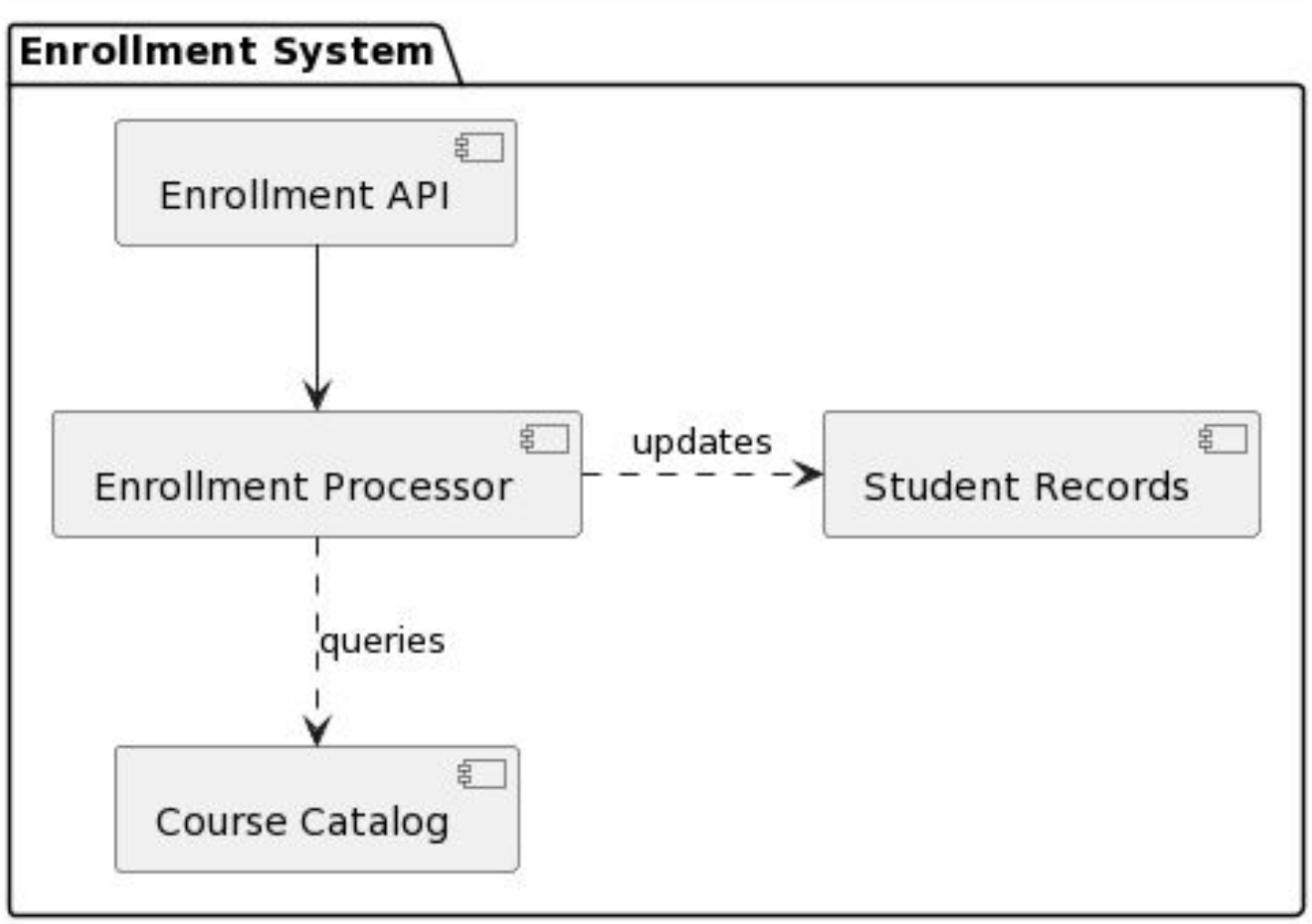
Level 1


External Systems
University Management System
User Management
Course Management
Faculty Management
Grades Management
Attendance Management
Billing & Payment Management
Reporting & Analytics

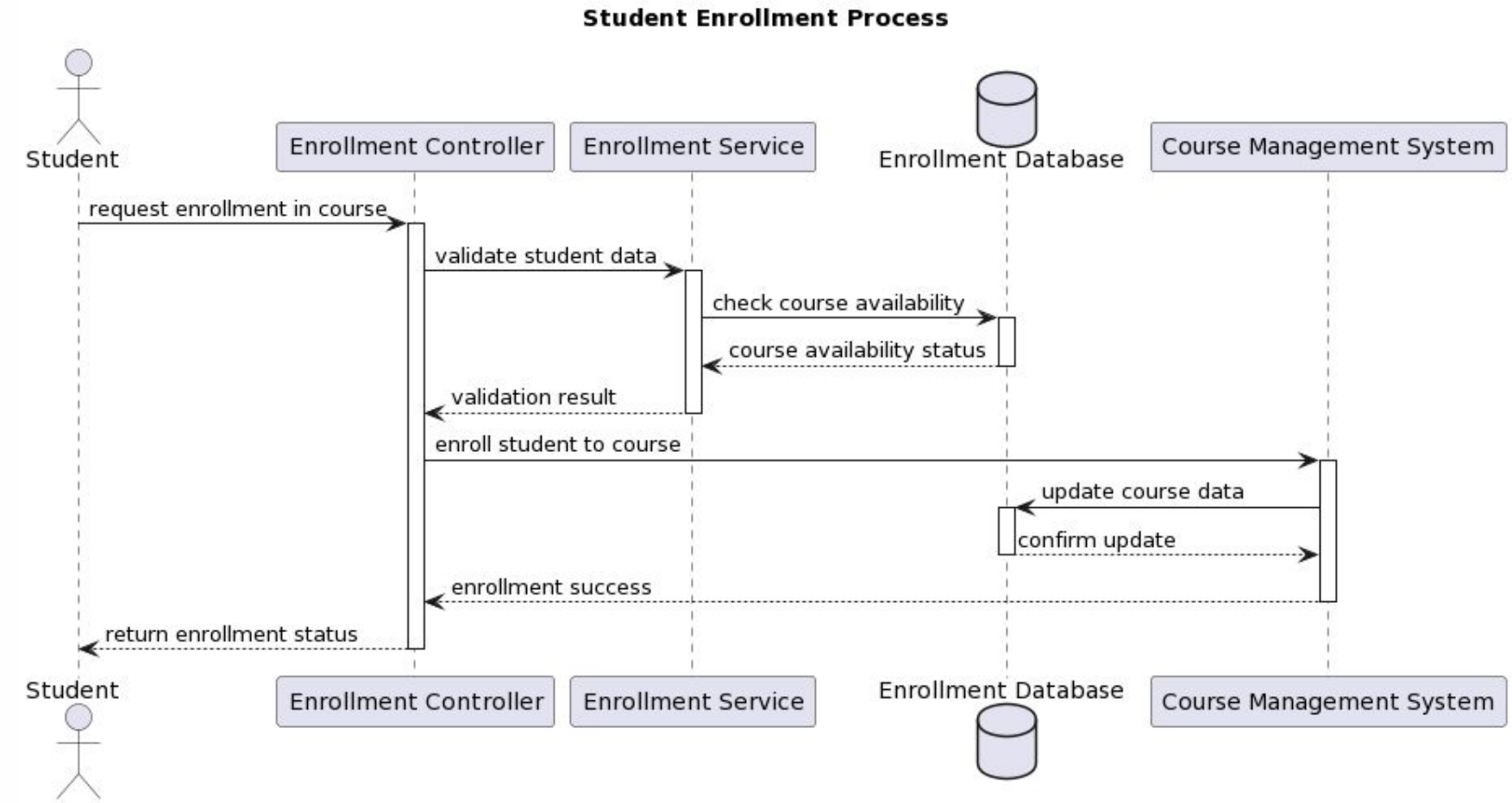
Level 2


Course Management
Course API
Course Scheduler
reads/writes
Enrollment System
reads/writes
Curriculum Database

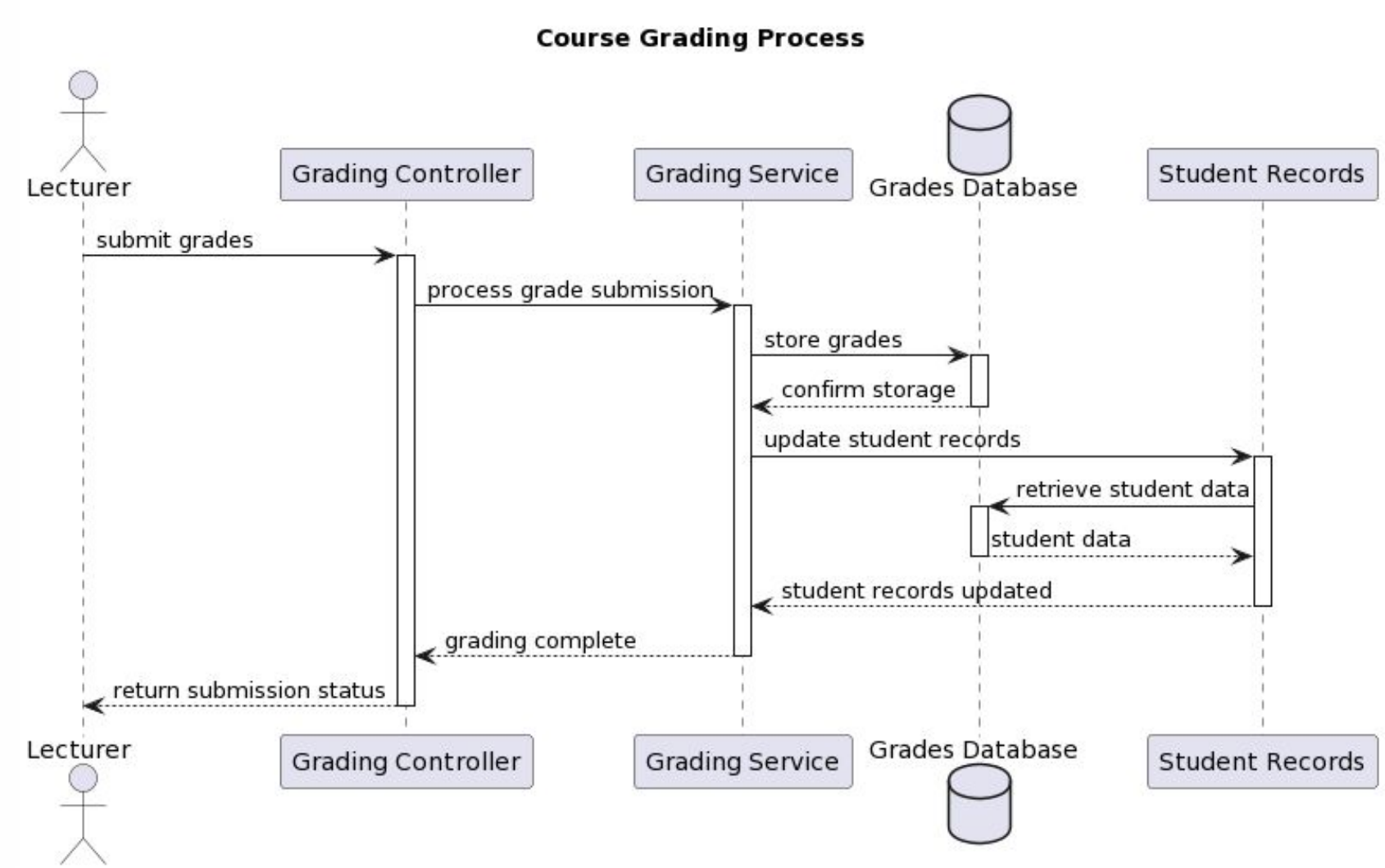
Level 3


Enrollment System
Enrollment API
Enrollment Processor
updates
Student Records
queries
Course Catalog

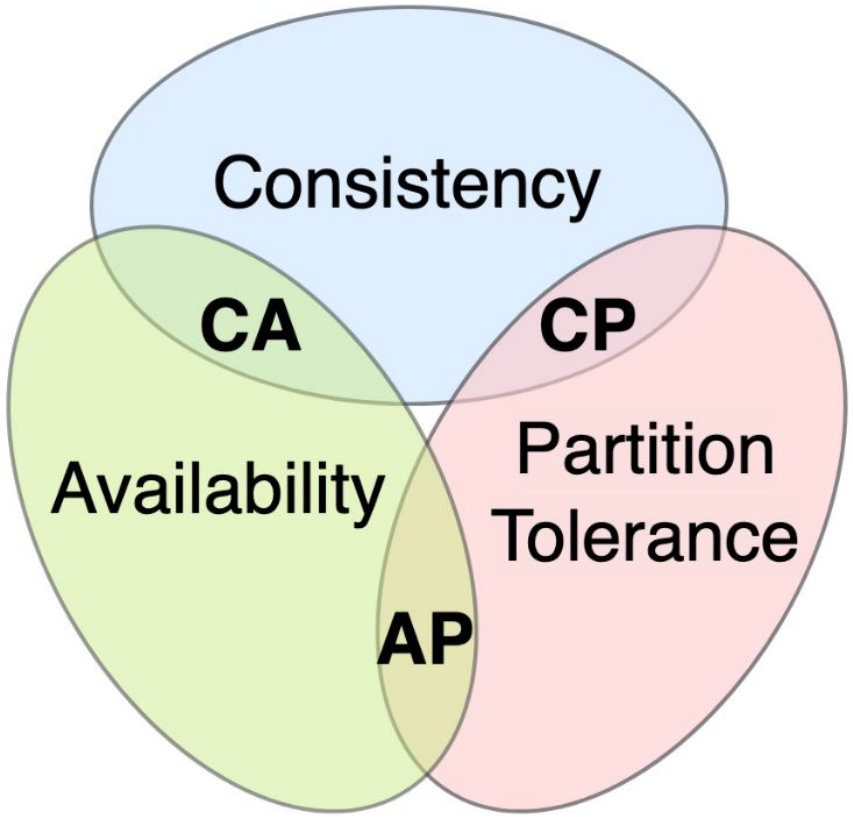
<Functionality 1>



<Functionality 2>



CAP theorem



- Consistency: Essential for academic records and financial data to ensure accuracy across the system.
- Availability: Crucial for providing uninterrupted access to the system for students and faculty, particularly during key academic periods.
- Partition Tolerance: Necessary to maintain system operations during network failures or partitioning events.

Given the need to prioritize, the UMS might:

- Prefer Consistency and Partition Tolerance (CP) for features like enrollment and grade submissions, where data accuracy cannot be compromised.
- Opt for Availability and Partition Tolerance (AP) for less critical features that can tolerate eventual consistency, like accessing course materials or general notifications.

The balance between these properties would be configured to meet the specific needs and usage patterns of the UMS, ensuring the most critical operations maintain integrity while less critical services remain as available as possible.

Deployment diagram