SOFTWARE DESIGN DOCUMENT

for

Academia – Course Registration Portal

Version 1.0

Prepared by :

1. Akshat Batra (IMT2023025)
2. Harshita Bansal (IMT2023035)

Submitted to :

Prof. Sujit Kumar Chakrabarti
Professor, IIITB

November 17, 2025

Contents

Chapter 1

Introduction

This Software Design Document (SDD) describes the architecture and detailed design for the Course Registration Portal. It contains design goals, component and deployment diagrams, data models, class and sequence diagrams, DFDs, state and activity diagrams, API specification, test plan, and traceability.

Chapter 2

Design Goals and Constraints

Design goals, assumptions, constraints as discussed: correctness, file-based persistence, concurrency with locks, extensibility.

2.1 Component Diagram

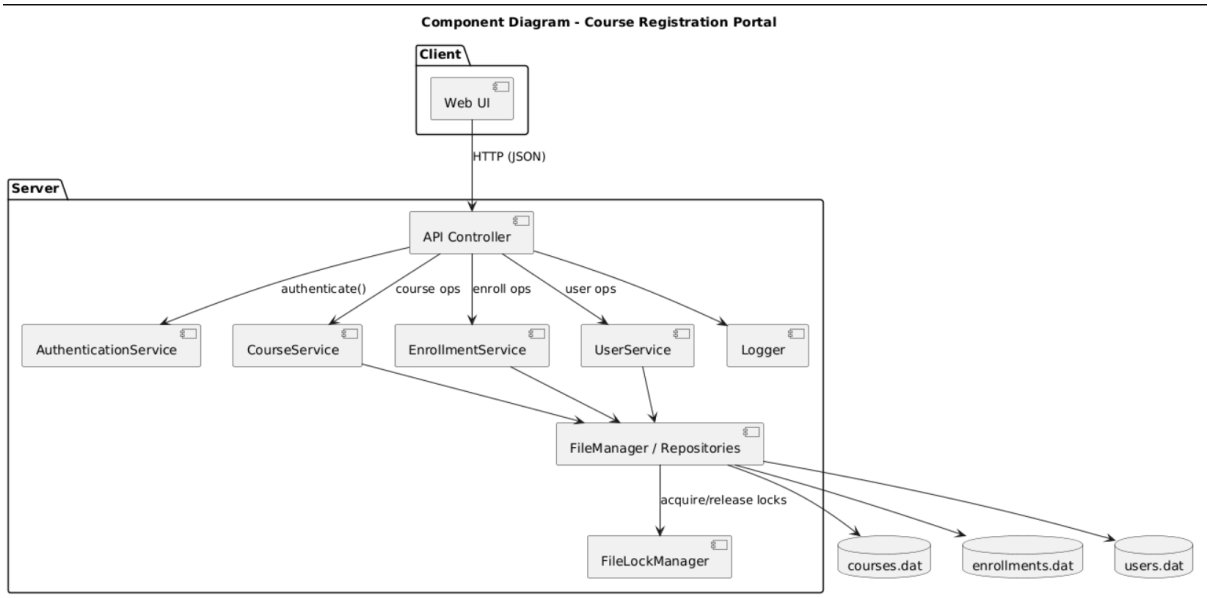

Figure 2.1: Component Diagram - Course Registration Portal

Explanation: The component diagram shows the high-level modular decomposition of the system into client-facing UI, API controller, and service components (AuthenticationService, CourseService, EnrollmentService, UserService). It also shows repositories and the FileLockManager for safe file-based persistence, and how services interact with the data files and logger.

Chapter 3

Deployment

3.1 Deployment Diagram

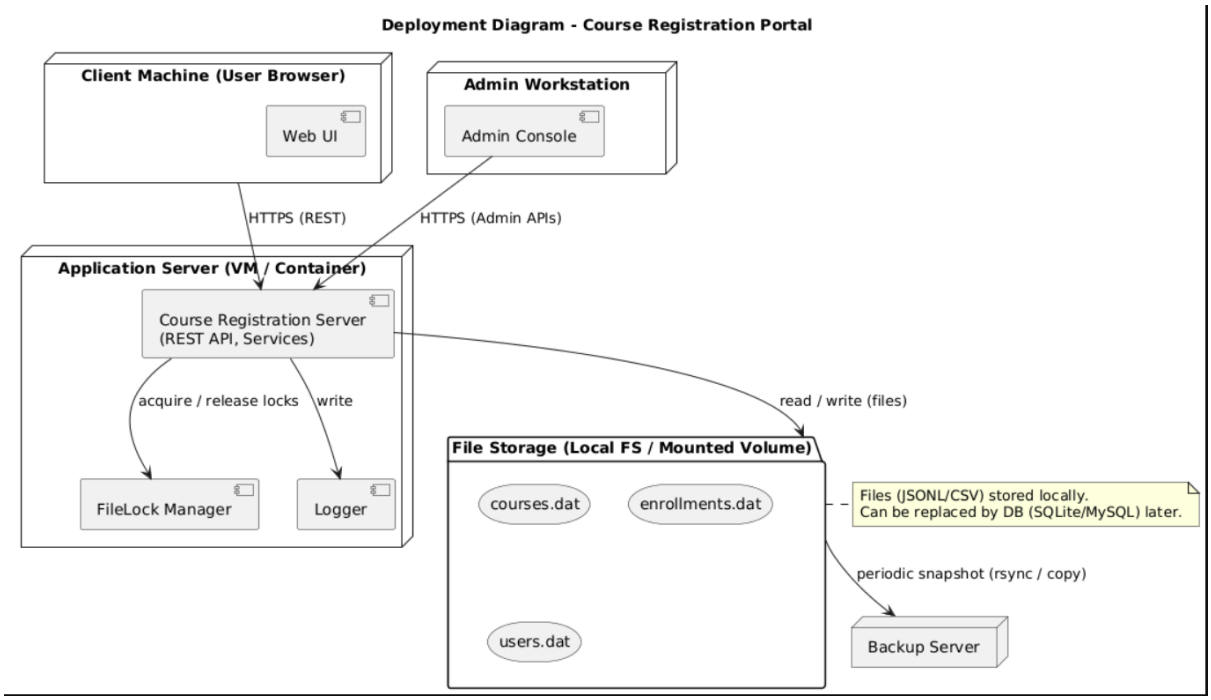

Figure 3.1: Deployment Diagram - Course Registration Portal

Explanation: The deployment diagram describes the runtime topology: user browsers, application server (VM/container) hosting the REST API and lock manager, and file storage (local/mounted volume) holding the data files. It also indicates administrative access and an external backup server for periodic snapshots.

Chapter 4

Use Cases

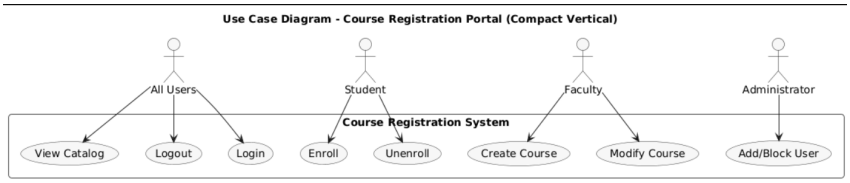

Figure 4.1: Use Case Diagram - Course Registration Portal

Explanation: The use case diagram summarizes primary user interactions with the system. It identifies actors (Students, Faculty, Administrators, and All Users) and top-level use cases such as login, view catalog, enroll/unenroll, course creation/modification, and administrative user management.

Chapter 5

Data Model

5.1 ER Diagram

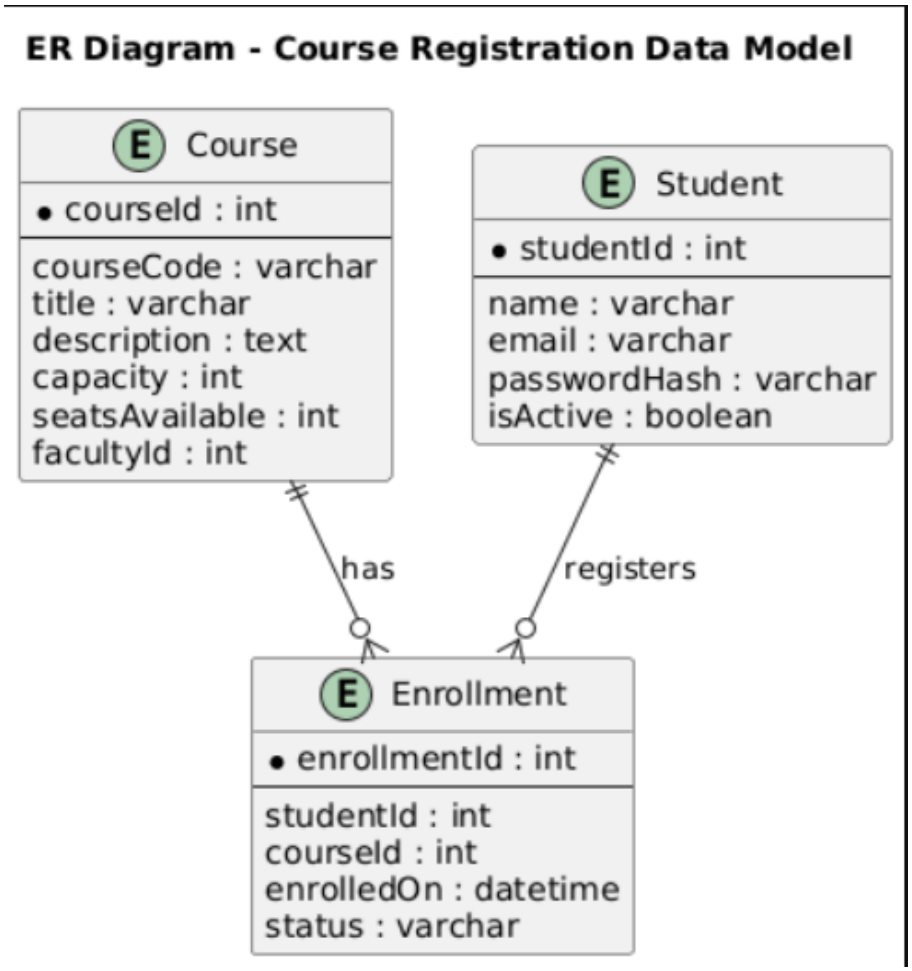

Figure 5.1: ER Diagram - Course Registration Data Model

Explanation: The ER diagram presents the logical data model. It shows the Course, Student and Enrollment entities, primary keys and key attributes, and relationships: a

Course has many Enrollments and a Student can register in many Enrollments. This model guides the file/record layout used by the repository layer.

Chapter 6

Structural Design

6.1 Class Diagrams

6.1.1 Compact Class Diagram

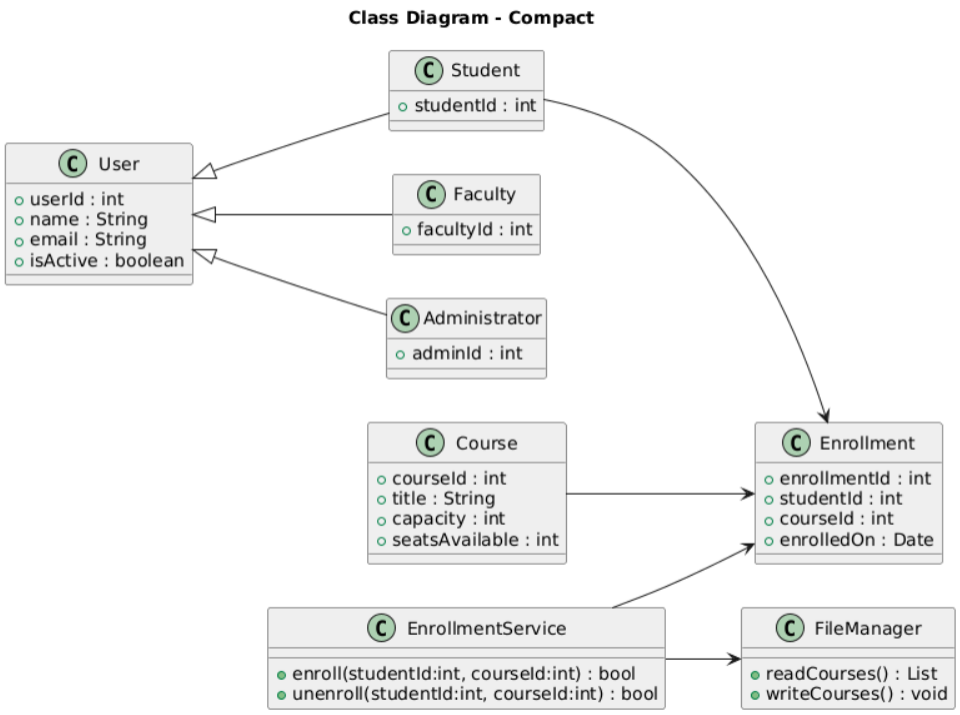

Figure 6.1: Compact Class Diagram - Course Registration Portal

Explanation: The compact class diagram gives an overview of core classes and their relationships (User hierarchy, Course, Enrollment, FileManager, EnrollmentService). It focuses on key methods and attributes relevant to service operations and persistence.

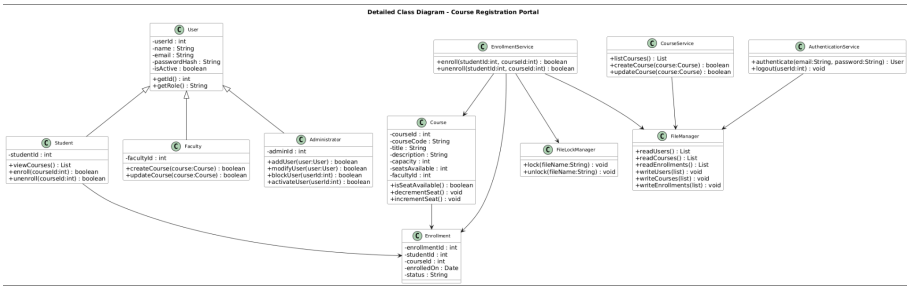

Figure 6.2: Detailed Class Diagram - Course Registration Portal

6.1.2 Detailed Class Diagram

Explanation: The detailed class diagram expands the compact view with full class attributes and public methods, showing service dependencies (EnrollmentService → FileManager, FileLockManager), and helper classes like AuthenticationService and CourseService. Use this diagram when implementing class interfaces and method signatures.

Chapter 7

Behavioral Design

7.1 Sequence Diagrams

7.1.1 Enroll Sequence

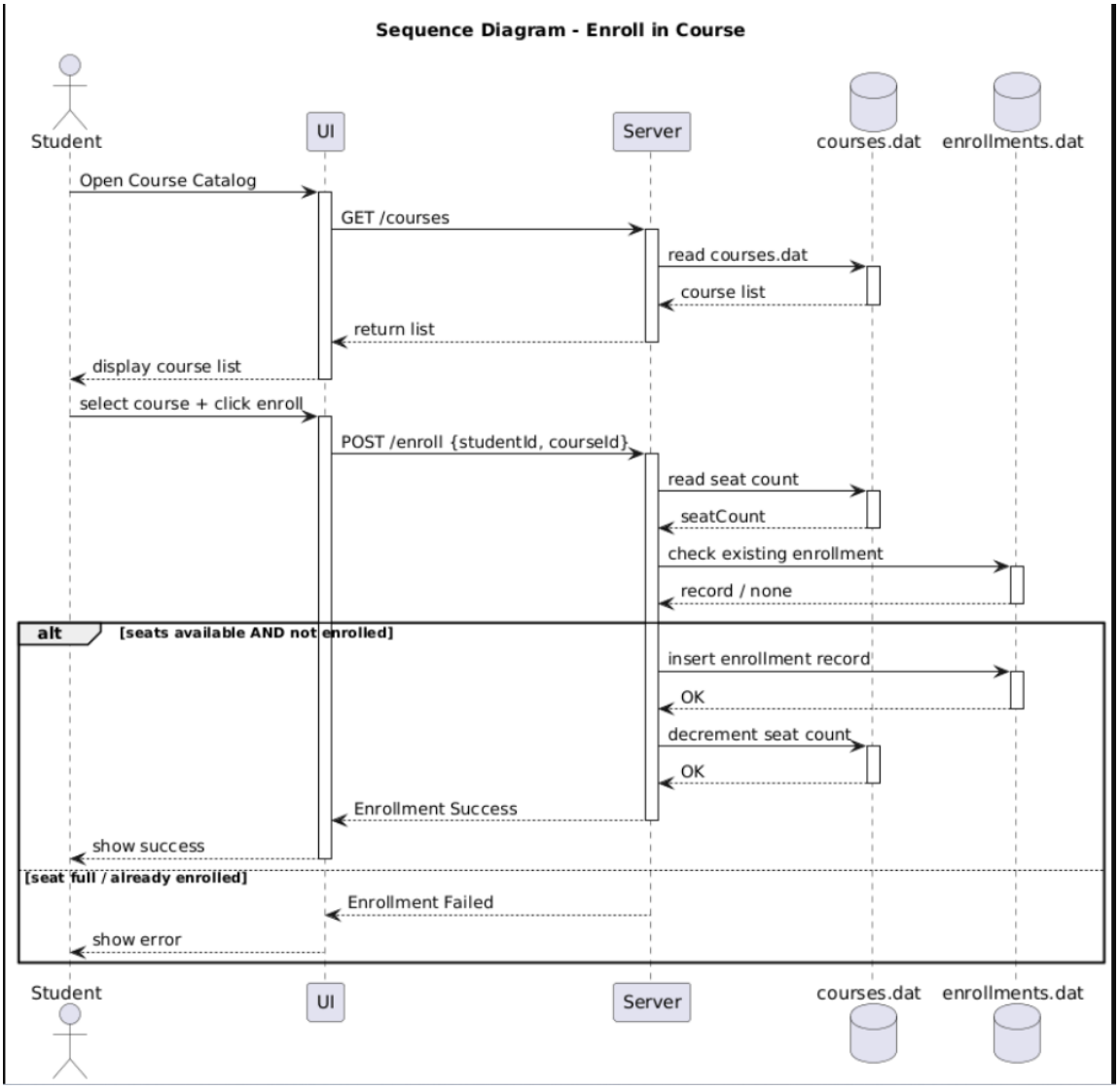

Figure 7.1: Sequence Diagram – Enroll in Course

Explanation: The enroll sequence diagram shows the runtime interaction for enrolling a student: the UI requests the course list, the student selects a course, the server checks seat availability and existing enrollment, inserts an enrollment record, decrements seat count, and returns success or failure.

7.1.2 Unenroll Sequence

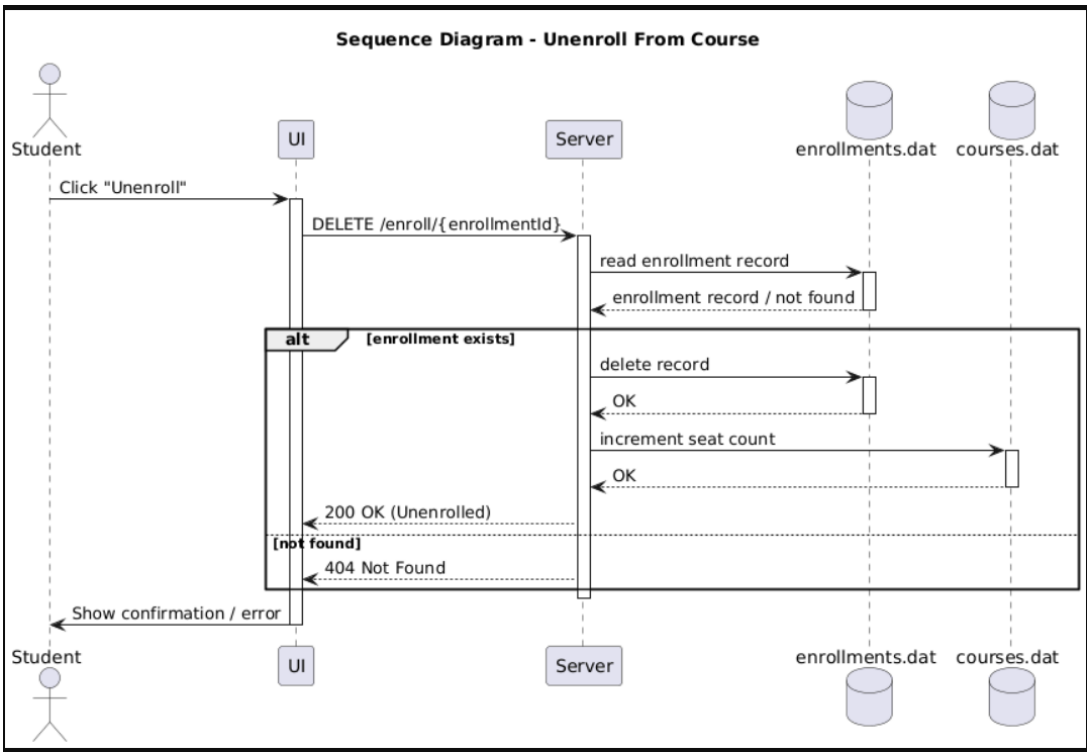

Figure 7.2: Sequence Diagram – Unenroll From Course

Explanation: The unenroll sequence diagram describes removing a student from a course: the UI issues a DELETE request, the server locates and deletes the enrollment record, increments seat count, and responds with confirmation or not-found.

7.1.3 Login Sequence

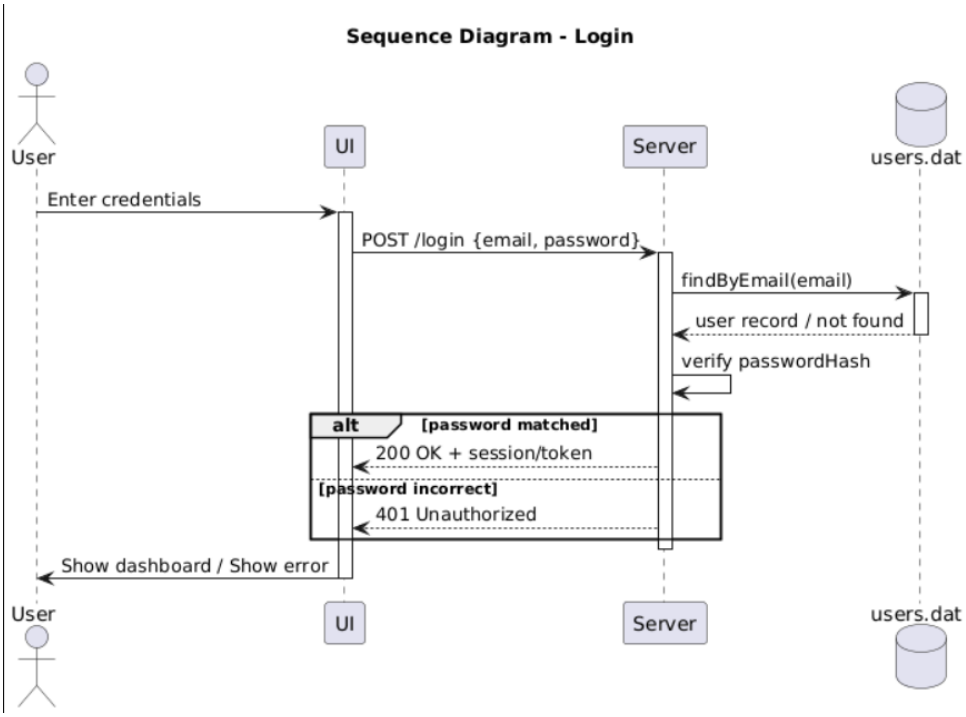

Figure 7.3: Sequence Diagram – Login

Explanation: The login sequence diagram captures authentication flow: the UI sends credentials to the server, which looks up the user record, verifies the password hash, and issues a session or token on success (or returns an unauthorized response on failure).

7.2 Activity Diagram

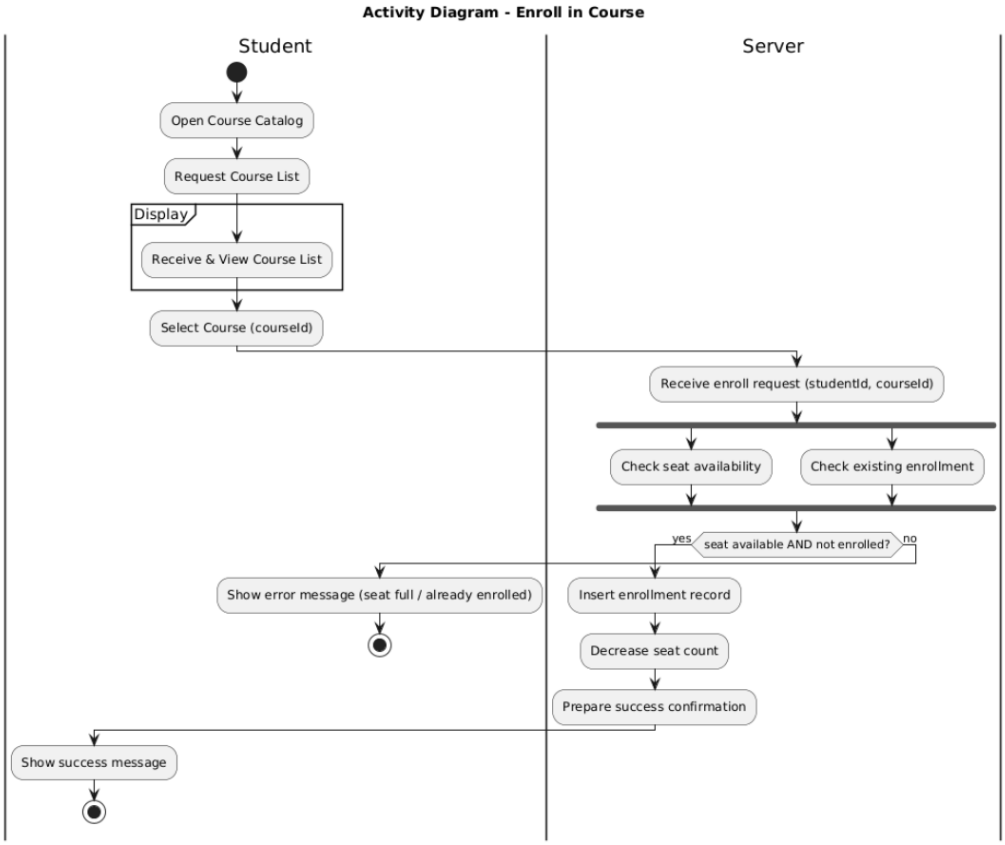

Figure 7.4: Activity Diagram – Enroll in Course

Explanation: The activity diagram outlines the internal decision flow for enrollment: after selecting a course, the server concurrently checks seat availability and prior enrollment; if both checks pass, it records the enrollment and decreases the seat count; otherwise it reports the appropriate error.

7.3 State Machine

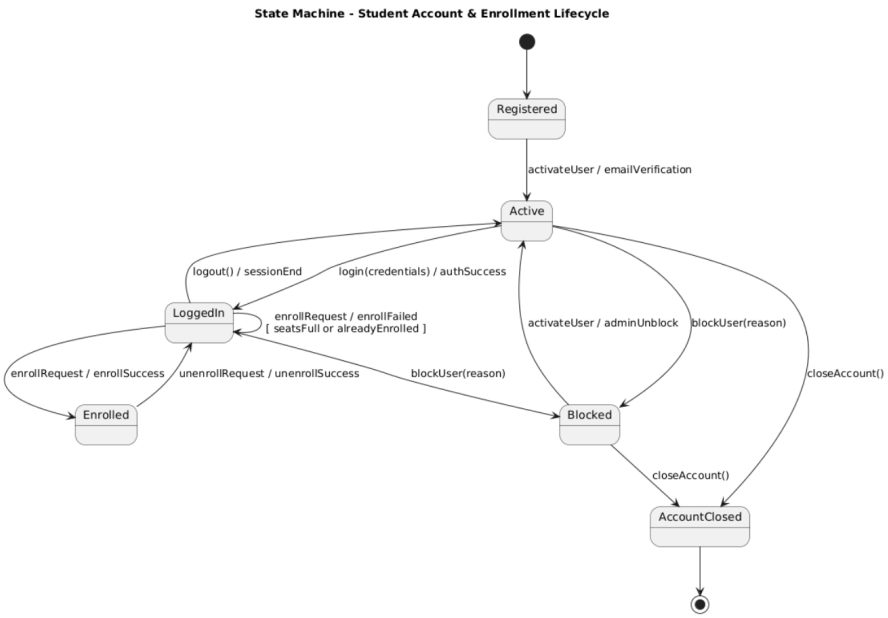

Figure 7.5: State Machine – Student Account & Enrollment Lifecycle

Explanation: The state machine shows lifecycle states for a student account (Registered → Active → LoggedIn → Enrolled → AccountClosed) and transitions triggered by actions like activation, login, enrollment, blocking and account closure. It documents allowed transitions and error states for the account lifecycle.

7.4 Object Diagram

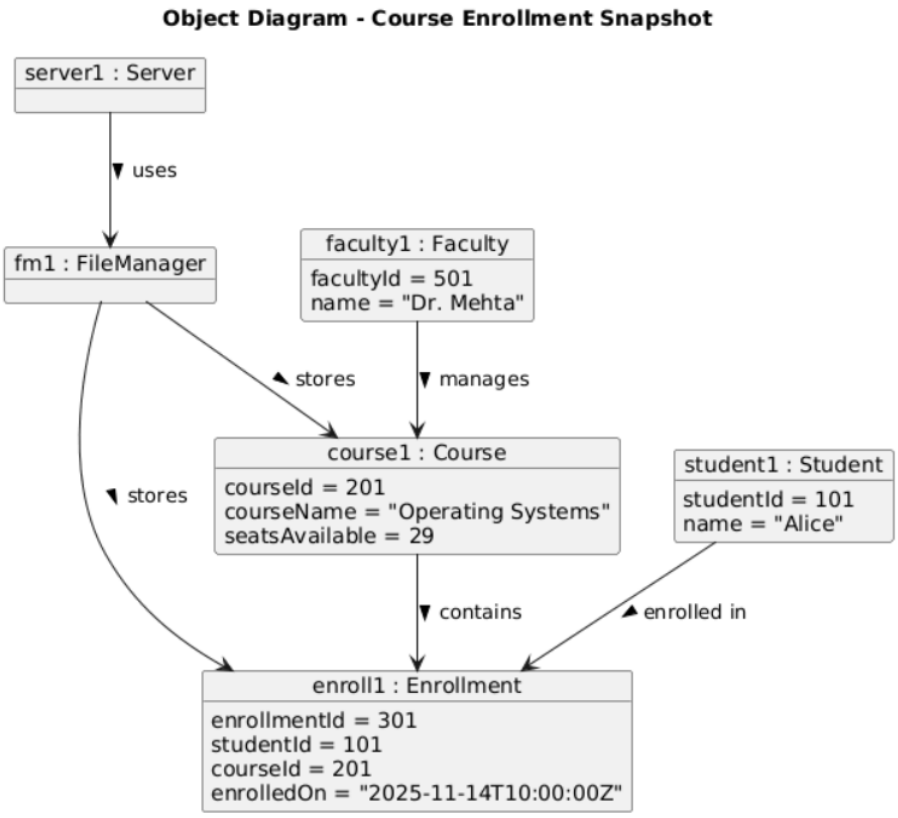

Figure 7.6: Object Diagram – Course Enrollment Snapshot

Explanation: The object diagram provides a concrete snapshot of runtime objects and their attribute values for a sample enrollment (student Alice enrolled in Operating Systems). It is useful to illustrate an example data instance and to validate the ER/class models.

Chapter 8

Data Flow Diagrams

8.1 DFD Level 0

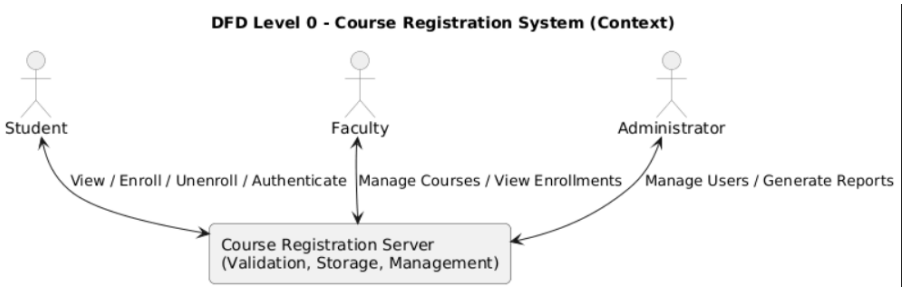

Figure 8.1: DFD Level 0 – Course Registration System (Context)

Explanation: The level-0 DFD (context diagram) shows external actors (Student, Faculty, Administrator) and the system boundary, summarizing high-level data flows such as view/enroll/unenroll, course management, and administrative functions.

8.2 DFD Level 1

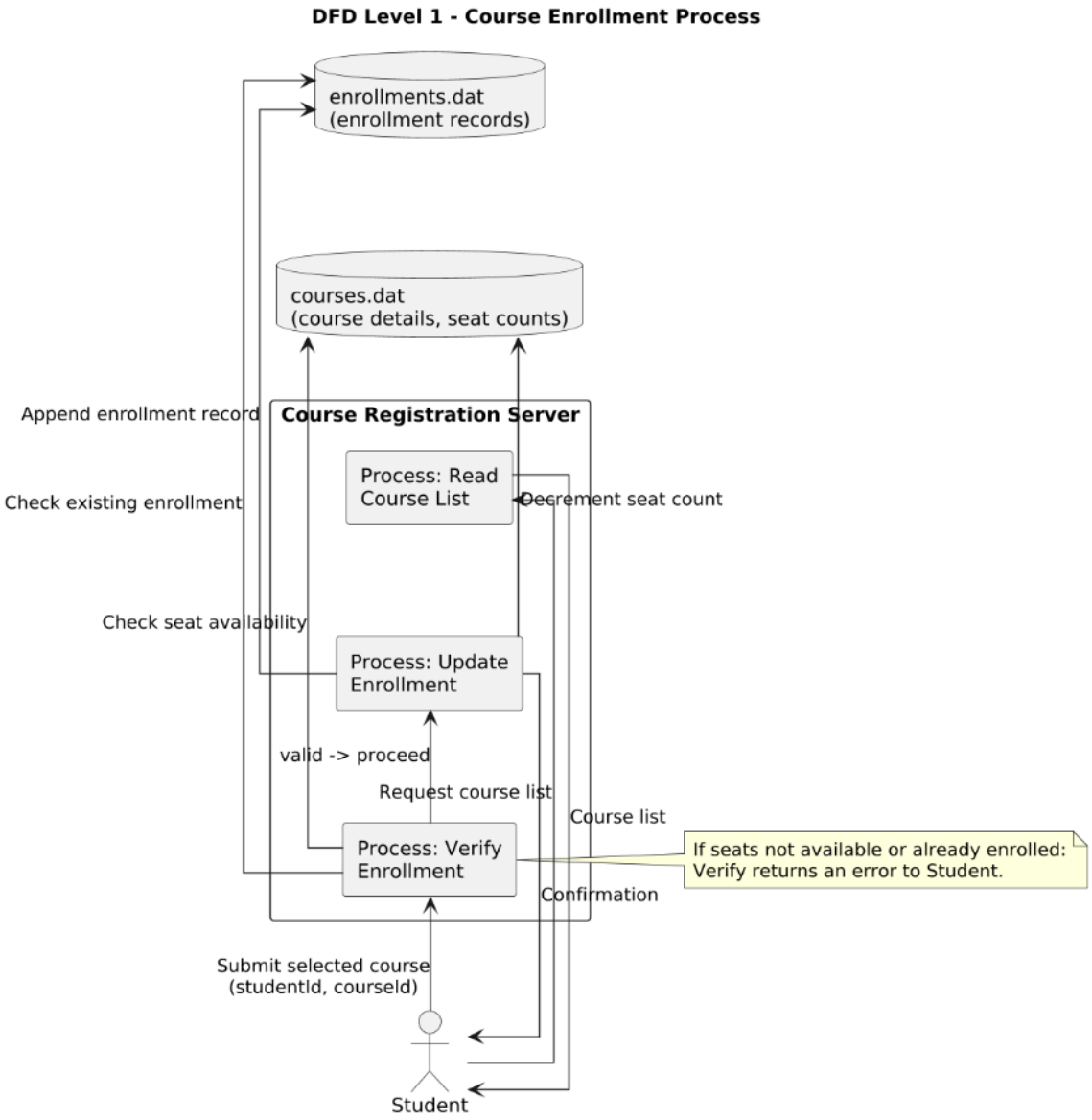

Figure 8.2: DFD Level 1 – Course Enrollment Process

Explanation: The level-1 DFD breaks down the enrollment process into subprocesses (Read Course List, Verify Enrollment, Update Enrollment) and shows data stores (courses.dat, enrollments.dat). It clarifies validation steps, updates and error handling for the enroll workflow.

Chapter 9

Architectural Style and Justification

The Course Registration Portal follows a combination of architectural styles:

9.1 Client–Server Architecture

The system uses a clear separation between the client (browser-based UI) and the server (API + services). All communication happens over HTTP using JSON. This matches the call-and-return architectural style discussed in class.

9.2 Layered Architecture

The server internally follows a layered structure:

- **Presentation Layer:** API controller that exposes endpoints.
- **Application Layer:** Services such as AuthenticationService, CourseService and EnrollmentService.
- **Data Layer:** FileManager and repositories that manage persistent data.

A layered architecture provides modularity, easier maintenance, and low coupling, aligning with the styles described in the lecture.

9.3 Data-Centered Repository Architecture

The system stores all persistent data (users, courses, enrollments) in centrally managed files. Services interact with the FileManager instead of directly accessing storage, matching the repository pattern from the lecture.

9.4 Justification

- The client–server model supports multiple user roles and remote access.
- Layering enforces separation of concerns, making the system easier to extend.
- A data-centered repository ensures consistency of shared data across services.
- This hybrid architecture is simple, maintainable, and appropriate for educational project scale.

9.5 Design Rationale

The chosen architecture was selected based on the following reasons:

- **Modularity:** Services are separated cleanly, reducing complexity.
- **Maintainability:** Layered design makes it easier to update individual modules.
- **Extensibility:** New roles, constraints or endpoints can be added without restructuring the system.
- **Simplicity:** File-based persistence and clear service boundaries reduce implementation overhead for an academic project.
- **Risk reduction:** Clear separation between layers reduces bugs and avoids tight coupling, as discussed in the lecture.

Chapter 10

Conformance to IEEE 1471 Architectural Description

The architectural description in this document follows the IEEE 1471 recommendations by presenting the system using multiple views:

- **Component View:** Shows structural decomposition into services and repositories.
- **Deployment View:** Shows execution environment and allocation of components.
- **Behavioral View:** Includes sequence, activity, and state diagrams.
- **Data View:** Includes ER diagram and file-based storage model.

These views address the concerns of different stakeholders such as developers, testers, administrators, and users.

Chapter 11

Architectural Impact on Non-Functional Requirements

The selected architecture has been designed to satisfy several critical non-functional requirements of the Course Registration Portal. The architectural choices directly support quality attributes such as modifiability, performance, reliability, scalability, security, and usability.

Modifiability

The layered structure separates responsibilities across the presentation, application, and data layers. Each service (e.g., AuthenticationService, CourseService, EnrollmentService) is modular and can be modified or extended without impacting unrelated components.

Extensibility

New features such as additional roles, course constraints, notifications, or alternative storage mechanisms can be added at the service layer without altering the core architecture. The repository pattern further simplifies integration of new persistence models.

Performance

The client–server architecture keeps the client lightweight while delegating processing to the server. Local file-based storage provides fast read/write operations for the scale of this system. Caching opportunities exist at the service level for frequently accessed data.

Reliability

Critical operations—including enrollment and seat updates—are protected using file-based locking to prevent race conditions. The architecture isolates faults to specific modules, reducing the chance of system-wide failures.

Fault Tolerance

File locks, controlled access through services, and structured error handling prevent data inconsistencies and mitigate the effects of partial failures or concurrent access.

Security

Security is enforced centrally in the AuthenticationService and validation logic. Separation of user roles (Student, Faculty, Administrator) prevents unauthorized operations, and the repository layer ensures protected access to sensitive data.

Scalability

The system can scale horizontally by deploying the server component on multiple nodes behind a load balancer. The architecture can also migrate from file storage to a relational database without major structural changes, thanks to the repository abstraction.

Maintainability

Clear separation of concerns, well-defined service boundaries, and minimal coupling between layers make the system straightforward to maintain, debug, and update.

Usability

By keeping the client-side simple and responsive and centralizing business logic on the server, the architecture ensures a consistent and predictable user experience across roles.

Chapter 12

API Specification (Summary)

12.1 Auth

- **POST /api/v1/login**
  Request: { `email, password` }
  Response: { `userId, role, token` }
- **POST /api/v1/logout**
  Invalidates the current session.

12.2 Courses

- **GET /api/v1/courses** — Fetch all courses
- **GET /api/v1/courses/{id}** — Fetch specific course
- **POST /api/v1/courses** (Faculty Only) — Create course
- **PUT /api/v1/courses/{id}** — Update course

12.3 Enrollments

- **POST /api/v1/enroll**
  Request body: { `studentId, courseId` }
- **DELETE /api/v1/enroll/{enrollmentId}**

12.4 Users (Admin Only)

- **POST /api/v1/users** — Create user

- **PUT /api/v1/users/{id}** — Modify user
- **POST /api/v1/users/{id}/activate**
- **POST /api/v1/users/{id}/block**

Chapter 13

Concurrency and Persistence

The system uses file-based persistence and must ensure consistent read/write access.

13.1 Locking Guidelines

A strict lock ordering is followed to avoid deadlocks:

1. **courses.dat**
2. **enrollments.dat**
3. **users.dat**

13.2 Persistence Notes

- Use `fsync()` after each write to ensure durability.
- Maintain periodic backups of all data files.
- Write operations must acquire locks before modifying files.

13.3 Example Pseudocode

“‘text acquireLock(”courses”) acquireLock(”enrollments”)
read courses.dat read enrollments.dat
update records
write courses.dat write enrollments.dat fsync()
releaseLock(”enrollments”) releaseLock(”courses”)

Chapter 14

Test Plan and Traceability

14.1 Test Plan

- **Unit Tests:** CourseService, EnrollmentService, Authentication.
- **Integration Tests:** login → enroll → unenroll workflow.
- **Concurrency Tests:** simultaneous enroll attempts.

14.2 Traceability Matrix

Maps each SRS requirement to:

- Design components
- UML diagrams
- Test cases

Chapter 15

Appendices

15.1 PUML Files

All `.puml` file sources have been included in earlier chapters. Separate single-file versions are available upon request.