SE 307 Concepts of Object Oriented Programming

Project Report

Student Name: Kadir Biberoğlu

Student ID: 20200601105

Date: 25.12.2025

1. Introduction

This project develops a robust **Distributed Syllabus Management System** to address inconsistencies in course management. Moving away from manual processes (e.g., Word documents), this solution offers a centralized, version-controlled repository utilizing **JSON** for data standardization. It integrates advanced features like simplified **OAuth-like authentication** simulation, **Role-Based Access Control (RBAC)**, and a **Multi-channel Notification System**.

2. Requirements Analysis

The system is designed to meet the following critical requirements:

- **Role-Based Access Control:** Distinct privileges for Instructors (Department-limited), Head of Departments (Full Read/Write), and Admins (System Management).
- **User Management:** Administrators have the capability to Create and Delete users dynamically.
- **Syllabus CRUD & Flexible Content:** Capability to manage syllabi with a schema-less design (Key-Value pairs) for flexibility.
- **Version Control System:** A Git-like tracking system that records every change with Commit IDs, Messages, Diffs, and Authors, allowing for rollback (Revert).
- **Automated Testing:** The system ensures reliability through a comprehensive Unit Test suite (xUnit & Moq) integration.
- **Containerization:** The entire application usage cycle (Build -> Test -> Run) is encapsulated in a robust **Docker** environment.

3. System Design and Implementation

3.1. Architecture

The application follows a modular Clean Architecture approach:

- **Presentation Layer:** Console-based UI driven by the Command Pattern (`CommandInvoker`).
- **Service Layer:** Encapsulates business rules (`SyllabusService`, `UserManagementService`, `PermissionService`).
- **Data Layer:** `JsonDataRepository` provides persistent storage using JSON files.
- **Test Layer:** Independent test project validating core logic and commands.

3.2. Design Patterns Applied

The system strictly adheres to SOLID principles and utilizes multiple GoF Design Patterns:

- **Command Pattern:** Decouples the UI from business logic. Operations like `CreateUserCommand` or `RevertCommand` are encapsulated objects.
- **Factory Pattern:** `ServiceFactory` centralizes dependency injection, ensuring loosely coupled components.
- **Observer Pattern:** `NotificationService` (Subject) notifies registered `INotificationChannel` (Observers) when a syllabus changes, enabling real-time alerts.
- **Adapter Pattern:** `EmailAdapter` wraps external service interfaces to match the system's `INotificationChannel`, promoting interoperability.
- **Strategy Pattern (Implicit):** The notification system allows runtime selection of different strategies (Email vs SMS).

4. Features Implemented

- **Dynamic User Management:** Admins can add or remove users from the system without system restart.
- **Advanced Version Control:**
  - **Commit History:** Detailed audit logs of all changes.
  - **Diff Generation:** Automatic calculation of "what changed" between versions.
  - **Revert:** Ability to rollback a syllabus to any previous commit state.
- **Docker & CI/CD Simulation:** A single `Dockerfile` that orchestrates the Build, Test, and Run lifecycle, ensuring code quality before deployment.
- **Subscription System:** Users can subscribe to specific course codes (e.g., "CE*") to receive automatic updates.

5. Conclusion

The Syllabus Management System successfully demonstrates the application of complex software engineering concepts. By integrating **Design Patterns**, **Unit Testing**, and **Containerization**, the project serves as a scalable and maintainable solution for academic course management. The addition of Admin tools and robust testing infrastructure ensures both flexibility and system stability.