SE 307 Concepts of Object Oriented Programming

Project Report

Student Name: Kadir Biberoğlu

Student ID: 20200601105

Date: 25.12.2025

1. Introduction

This project aims to develop a robust **Syllabus Management System** to address the inconsistencies and difficulties in managing course syllabi within the university. The current system relies on disparate solutions like "Firebird" and various Word document templates, leading to data duplication, loss of version history, and management inefficiencies.

The proposed solution is a C# based application that standardizes syllabus data using **JSON format**, enables **version control** to track changes, and integrates **notification services** to keep stakeholders informed of updates.

2. Requirements Analysis

The system was designed to meet the following key requirements:

- **Authentication:** Secure login mechanism simulating OAuth, with role-based access control (e.g., separating standard users from admins).
- **Syllabus Management (CRUD):** Capability to List, Create, Update, and Delete syllabi.
- **Data Standardization:** Storing all syllabus data in structured **JSON** format to separate content from presentation.
- **Version Control:** Tracking changes (Who, When, Why) and allowing reversion to previous states, mimicking Git-like functionality.
- **Notifications:** alerting interested parties (subscribers) via Email and SMS (simulated) when a syllabus is updated.
- **Platform:** Built using **C#**, adhering to Object-Oriented Programming principles and SOLID best practices.

3. System Design and Implementation

3.1. Architecture

The application follows a modular architecture, separating concerns into distinct layers:

- **Presentation Layer:** A Console-based user interface managed by a `CommandInvoker`.
- **Service Layer:** Contains business logic (`SyllabusService`, `NotificationService`, `AuthorizationService`).
- **Data Access Layer:** `JsonDataRepository` handles reading/writing data to JSON files.
- **Domain Layer:** Defines core models like `Syllabus`, `User`, and interfaces.

3.2. Design Patterns Applied

To ensure maintainability and scalability, several standard design patterns were implemented:

- **Command Pattern:** Used to encapsulate user actions (e.g., `ListSyllabiCommand`, `CreateSyllabusCommand`). This allows for decoupling the UI from the business logic and facilitates extending the menu with new commands easily.
- **Factory Pattern:** Implemented in `ServiceFactory` class. It centralizes the creation of services (like `GetSyllabusService`), ensuring that dependencies are injected correctly and single instances are managed effectively.
- **Adapter Pattern:** The `EmailAdapter` class adapts the external `ThirdPartyEmailService` to the internal `INotificationChannel` interface. This allows the system to switch or add new notification providers without changing the core logic.
- **Observer Pattern:** The `NotificationService` acts as a subject that notifies subscribed observers (channels like Email) when a significant event (e.g., valid update) occurs. Users can subscribe to specific courses.

4. Features Implemented

- **User Authentication:** Users log in with a password. The system verifies credentials against the stored user data.
- **JSON Data Storage:** Syllabi are saved as JSON files, ensuring a standardized, machine-readable format.
- **CRUD Operations:** Full Create, Read, Update, Delete functionality for course syllabi.
- **History & Versioning:**
    - Every update creates a commit log (Audit Log) with "Who", "When", and "Why".
    - Users can view the history of changes.

    - **Revert Capability:** Integrating a "git revert" style feature to restore a syllabus to a previous version.
- **Notifications:** When a syllabus is updated, subscribed users are notified via the configured channels (simulated Email/SMS).

5. Conclusion

The Syllabus Management System successfully addresses the problem of fragmented syllabus data. By leveraging C# and Object-Oriented best practices, the application provides a centralized, version-controlled, and standardized environment for managing course information. The use of design patterns ensures the codebase remains clean, testable, and easy to extend for future requirements.