PROJECT REPORT : Syllabus Management System

Course: SE 307 - Concepts of Object-Oriented Programming

Instructor: Assoc. Prof. Dr. Kaya Oğuz

Student: Sinan Mert Sener

Student ID: 20190602037

1. Executive Summary

The **Syllabus Management System** is a specialized software solution designed to address the challenges of curriculum management in academic institutions. The primary objective of this project was to transition from decentralized, inconsistent file formats (Word/PDF) to a structured, **JSON-based data architecture**.

The resulting application provides a robust platform for Instructors to manage course content with version control capabilities, while offering Students a simplified, read-only view of the finalized curriculum. The project was developed using **C#** and **ASP.NET Core MVC**, strictly adhering to Object-Oriented Programming principles and **SOLID** design patterns.

2. Functional Requirements & Features

The system was designed to meet specific functional requirements outlined in the project description:

Role-Based Access Control (RBAC): The system distinguishes between Instructor and Student roles. Only Instructors are authorized to create, update, or delete syllabi.

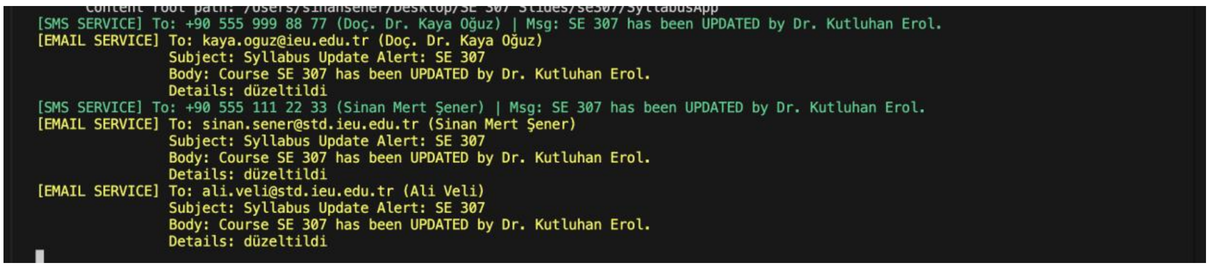
Custom Version Control: Unlike traditional CRUD operations that overwrite data, this system implements a commit-based history mechanism. Every update creates a snapshot, allowing full traceability of changes.

Automated Notifications: Critical actions (updates/deletions) trigger an automated notification simulation. Based on the course code prefix (e.g., "SE" or "CE"), the system intelligently routes SMS and Email alerts to the relevant Head of Department.

Academic Archiving: Students are presented with a filtered view of the history, showing only the finalized version of the syllabus for each academic year, preventing confusion from intermediate edits.

OAuth/EKOID Login Simulation: The system simulates the university's authentication system using a FakeUserDatabase, demonstrating how real OAuth integration would function.

Subscription System: Users can subscribe to specific courses or course patterns (e.g., "SE*") for all Software Engineering courses) to receive notifications when changes occur.



```
      Content root path: /Users/sinansener/Desktop/SE 307 Slides/se307/SyllabusApp
[SMS SERVICE] To: +90 555 999 88 77 (Doç. Dr. Kaya Oğuz) | Msg: SE 307 has been UPDATED by Dr. Kutluhan Erol.
[EMAIL SERVICE] To: kaya.oguz@ieu.edu.tr (Doç. Dr. Kaya Oğuz)
                Subject: Syllabus Update Alert: SE 307
                Body: Course SE 307 has been UPDATED by Dr. Kutluhan Erol.
                Details: düzeltildi
[SMS SERVICE] To: +90 555 111 22 33 (Sinan Mert Şener) | Msg: SE 307 has been UPDATED by Dr. Kutluhan Erol.
[EMAIL SERVICE] To: sinan.sener@std.ieu.edu.tr (Sinan Mert Şener)
                Subject: Syllabus Update Alert: SE 307
                Body: Course SE 307 has been UPDATED by Dr. Kutluhan Erol.
                Details: düzeltildi
[EMAIL SERVICE] To: ali.veli@std.ieu.edu.tr (Ali Veli)
                Subject: Syllabus Update Alert: SE 307
                Body: Course SE 307 has been UPDATED by Dr. Kutluhan Erol.
                Details: düzeltildi
```

(Figure 1: Automated Notification System providing immediate user feedback)

3. System Architecture

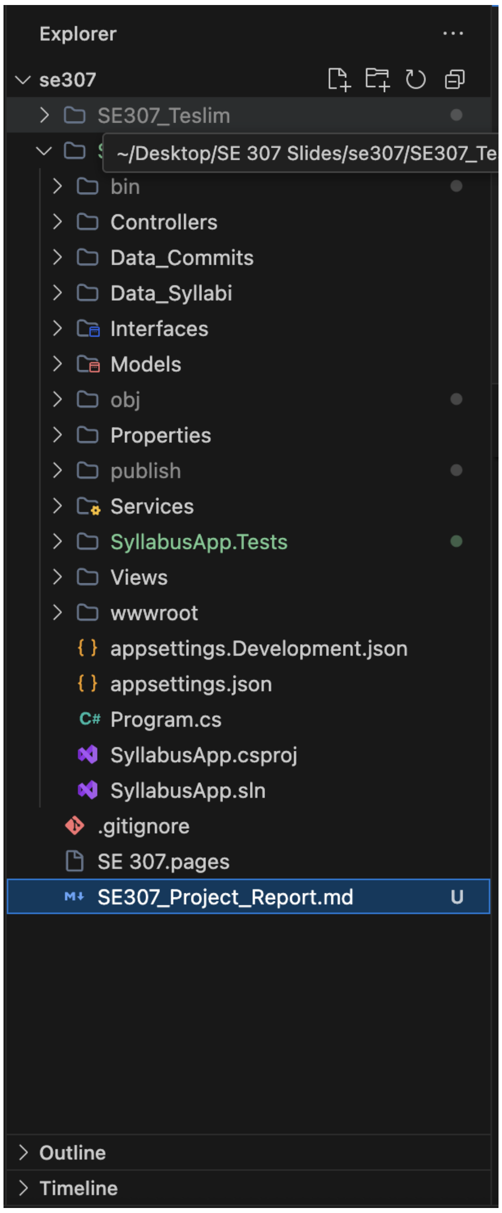
The project is built on the **ASP.NET Core MVC** framework, ensuring a clean separation between the user interface, business logic, and data handling.

3.1. Architectural Pattern (MVC)

Controllers: Act as the entry point for HTTP requests. They orchestrate the flow but delegate all business logic to the Service layer.

Views: Responsible for rendering the User Interface (UI) dynamically using Razor syntax.

Models: POCO (Plain Old CLR Object) classes that define the data structure without any dependency on the storage mechanism.



(Figure 2: Project Structure adhering to Separation of Concerns)

4. Technical Implementation & Design Methodology

This section details the engineering decisions and design principles applied during development. The system architecture is heavily influenced by the **SOLID principles** and **Object-Oriented concepts** covered in the course.

4.1. Design Principles (SOLID)

Single Responsibility Principle (SRP): The codebase is modular. For example, SyllabusManager is solely responsible for Data I/O, while NotificationService is dedicated to dispatching alerts. This ensures that a change in notification logic does not impact data storage.

Dependency Inversion Principle (DIP): The high-level SyllabusController does not depend on the low-level SyllabusManager class. Instead, it depends on the ISyllabusService interface. This decoupling is achieved through **Constructor Injection**.

Open/Closed Principle (OCP): The system is designed for extension. New notification types or different storage mechanisms can be added by implementing the existing interfaces without modifying the core controller logic.

Interface Segregation Principle (ISP): Instead of one large interface, the system uses three focused interfaces: ISyllabusService, INotificationService, and ISubscriptionService.

Liskov Substitution Principle (LSP): Both Instructor and Student classes can be substituted for their parent User class without affecting program correctness.

4.2. Object-Oriented Implementation

Inheritance & Polymorphism: To manage users efficiently, an abstract User base class was created. Student and Instructor classes inherit from this base. The system treats the current user polymorphically during the session, checking the specific role only when necessary (e.g., authorization).

Interfaces: Contracts such as ISyllabusService and INotificationService were defined to standardize the behavior of the components.

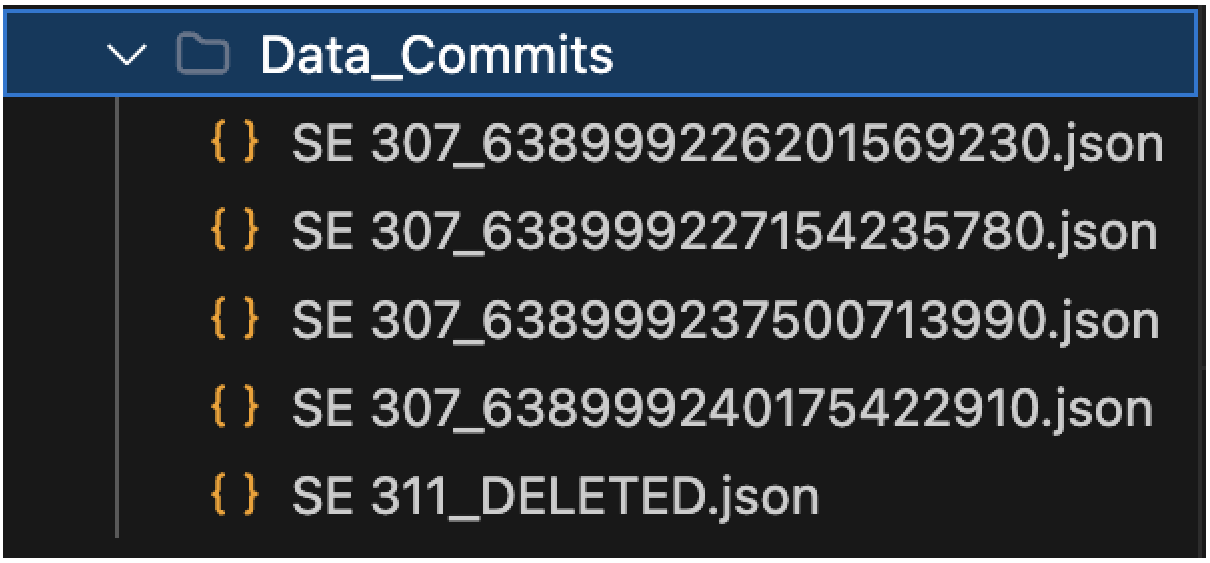
Generics: The system leverages C# Generics (List<T>, IEnumerable<T>) to handle collections of syllabi and commit history in a type-safe manner.

5. Data Persistence & Versioning

A core requirement of the project was to utilize JSON as the storage format instead of a relational database.

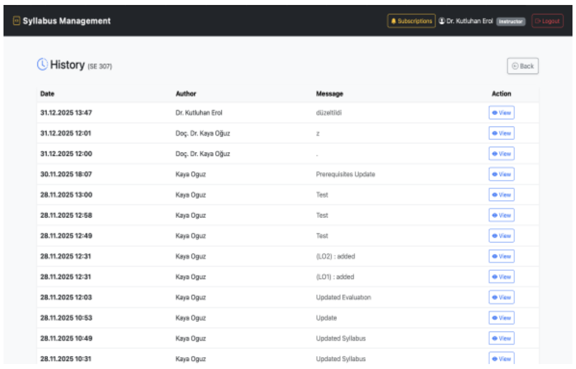
Storage Strategy: The system utilizes a file-based database approach. Each syllabus and each commit is serialized into a separate JSON file stored in the Data_Syllabi and Data_Commits directories.

Commit Logic: When an update occurs, the SyllabusManager captures the current state of the object, timestamps it, and saves it as a new commit file. This allows the system to reconstruct the history of any course at any point in time.



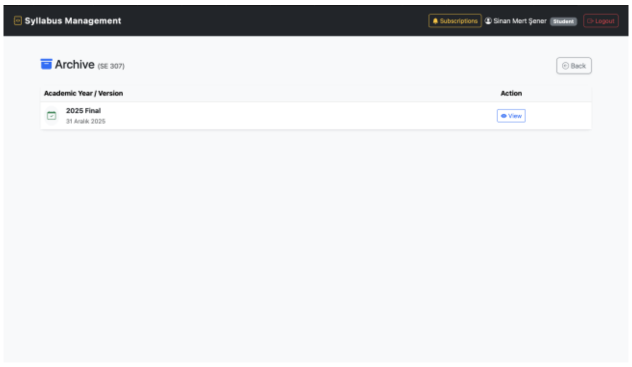
```
Data_Commits
    SE 307_638999226201569230.json
    SE 307_638999227154235780.json
    SE 307_638999237500713990.json
    SE 307_638999240175422910.json
    SE 311_DELETED.json
```

(Figure 3: JSON-based commit history storage)



History (SE 307)

Date	Author	Message	Action
31.12.2025 13:47	Dr. Kutluhan Erol	düzeltildi	View
31.12.2025 12:01	Doç. Dr. Kaya Oğuz	z	View
31.12.2025 12:00	Doç. Dr. Kaya Oğuz	.	View
30.11.2025 18:07	Kaya Oguz	Prerequisites Update	View
28.11.2025 13:00	Kaya Oguz	Test	View
28.11.2025 12:58	Kaya Oguz	Test	View
28.11.2025 12:49	Kaya Oguz	Test	View
28.11.2025 12:31	Kaya Oguz	(LO2) : added	View
28.11.2025 12:31	Kaya Oguz	(LO1) : added	View
28.11.2025 12:03	Kaya Oguz	Updated Evaluation	View
28.11.2025 10:53	Kaya Oguz	Update	View
28.11.2025 10:49	Kaya Oguz	Updated Syllabus	View
28.11.2025 10:31	Kaya Oguz	Updated Syllabus	View



Archive (SE 307)

Academic Year / Version	Action
2025 Final 31 Aralık 2025	View

(Figure 4: The implementation of the versioning logic as seen by different users)

6. Conclusion

The **Syllabus Management System** successfully delivers a functional, maintainable, and professionally architected solution. By moving away from rigid file formats to a structured JSON architecture and implementing a custom version control system, the project solves the defined problem of data inconsistency. Furthermore, the rigorous application of SOLID principles and MVC architecture ensures that the software is robust and ready for future extensions.