**CS 319**
**Object-Oriented Software Engineering**

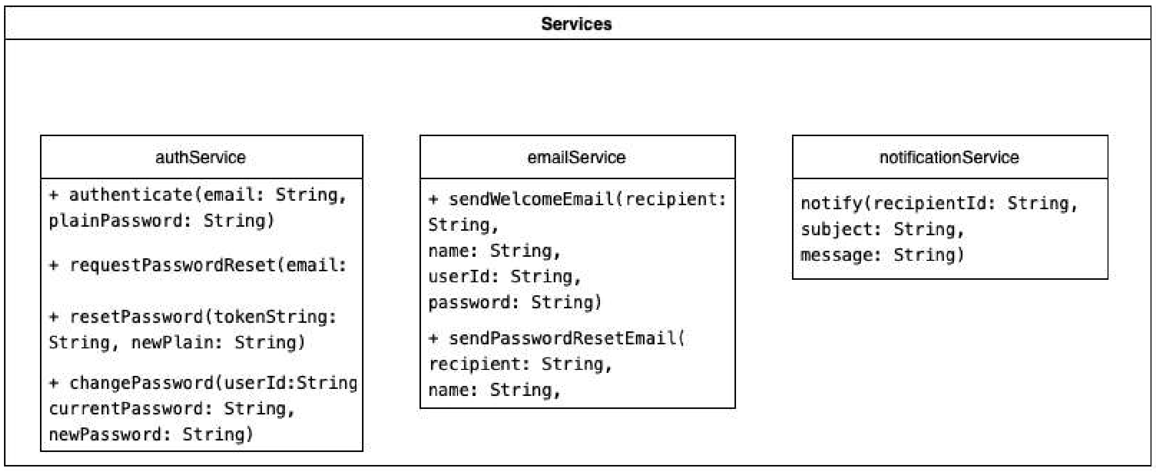
# ProctorHub
# Deliverable 4

Section-1
10.05.2025

Emine İrem Esendemir - 22202103
Halil Arda Özongun - 22202709
Yunus Emre Erkan - 22203670
Elif Lara Oğuzhan - 22203138
Sude Ergün - 22203822

# 1. DESIGN PATTERNS

## Singleton Pattern


Services
authService
+ authenticate(email: String, plainPassword: String)
+ requestPasswordReset(email:
+ resetPassword(tokenString: String, newPlain: String)
+ changePassword(userId:String currentPassword: String, newPassword: String)
emailService
+ sendWelcomeEmail(recipient: String, name: String, userId: String, password: String)
+ sendPasswordResetEmail( recipient: String, name: String,
notificationService
notify(recipientId: String, subject: String, message: String)

The Singleton Design Pattern is used to ensure that only one instance of a class or service exists throughout the entire application. In this project, we applied this pattern to the services such as *authService*, *notifyService*, and *emailService*.

The *authService* is responsible for handling authentication tasks such as login operations and token management. By implementing it as a singleton, the configuration for token generation, secret keys, and expiration settings is centralized.

The *notifyService* and *emailService* manage internal notifications and external email communication. These services are also structured as singletons so that shared configurations—such as notification templates, and logging preferences—are maintained in a single instance. This approach makes message handling more efficient.

Overall, the use of the Singleton pattern in these services contributes to a more reliable, and maintainable architecture. It ensures that common utilities are reused rather than repeatedly created, promoting consistency and reducing the chance of errors.

## Strategy Pattern

The Strategy Pattern is a behavioral design pattern that defines a family of algorithms, encapsulates each one, and makes them interchangeable. It allows the application to choose the correct algorithm based on user selection, without modifying the structure of the surrounding logic.

In this project, the Strategy Pattern is used in the exam creation flow, specifically for assigning TAs to proctoring duties. When adding a new exam, users are given the option to choose number between automatic and manual proctor assignment. This decision determines which algorithm is applied.

First, separate strategies were defined for different proctoring behaviors. The automatic assignment strategy takes into account workload, department, and availability, with an optional feature to prioritize TAs associated with the selected course. In the manual assignment, user can choose specific TA's and can assign them to proctorings.

Second, these strategies are encapsulated in different implementations. Each one represents a different algorithm for selecting TAs.

As a result, the system can switch between different proctoring assignment algorithms at runtime, based entirely on form selections made by the user. This design provides flexibility and keeps the code modular and easy to maintain.