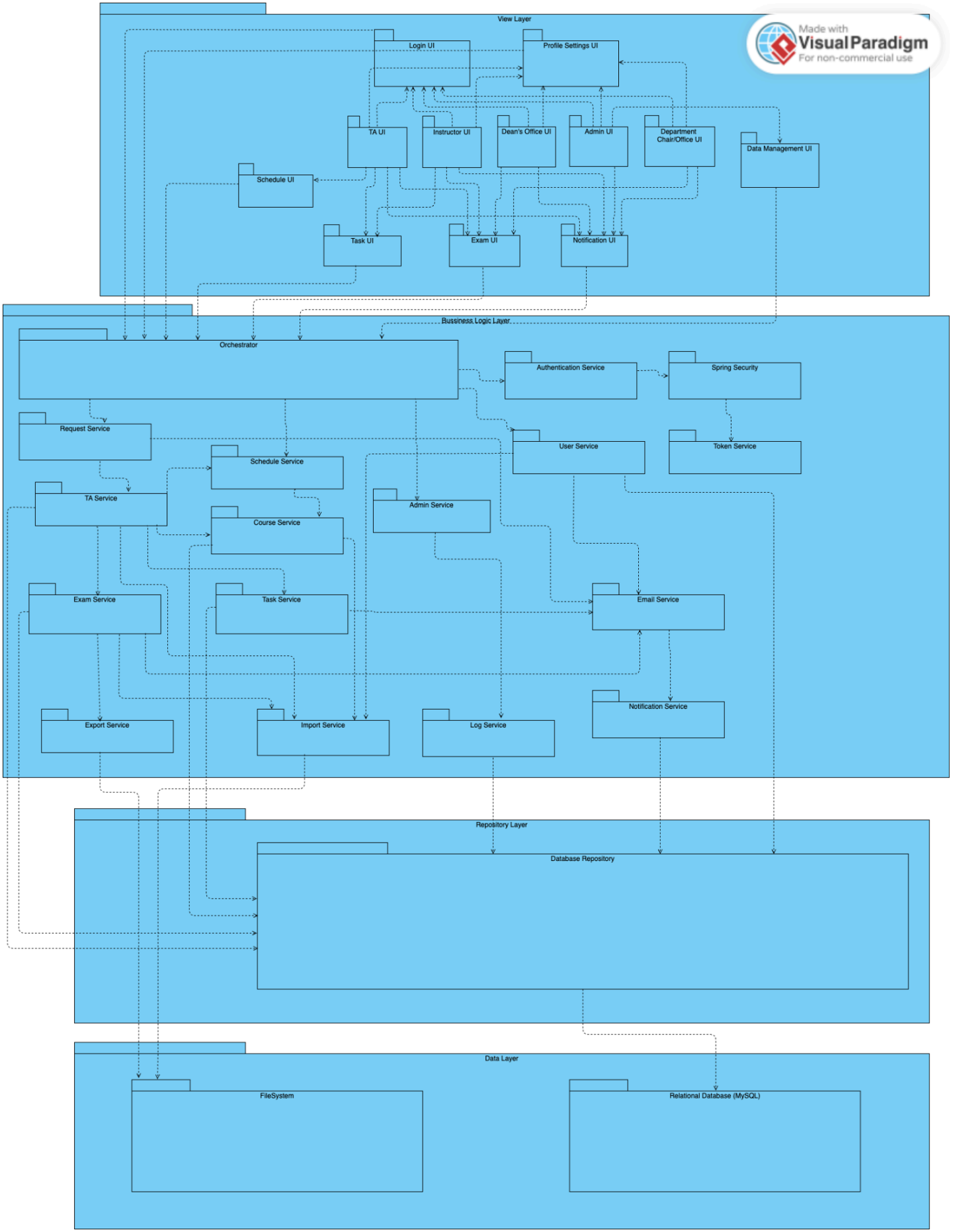
BILKENT UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
CS 319 OBJECT-ORIENTED SOFTWARE ENGINEERING
DELIVERABLE 3 ITERATION 1
SPRING 2025
GROUP 2 SECTION 1
24.04.2025

Full Name	ID
Perhat Amanlyyev	22201007
Emiralp İlgen	22203114
Anıl Keskin	22201915
İlmay Taş	22201715
Simay Uygur	22203328

Table of Contents



The link is: [diagramLink]

1.0 Design Goals

1.1 Modifiability

Modifiability is a core design goal of our system to provide ease for new features or rule changes, as we anticipate potential changes in business logic, academic policies, or user requirements over time particularly because the system is intended to support different departments (e.g., Computer Engineering and Industrial Engineering), each of which may have unique workflows, rules, and task structures. By following the Model-View-Controller (MVC) pattern and leveraging Spring Boot's layered architecture (controllers, services, repositories), we ensure that each component can be updated independently with minimal impact on the rest of the system. This modular design makes it easier to extend functionality, such as adding new task types or modifying TA assignment rules, by adding one class and updating a config without requiring major rewrites.

1.2 Reliability

Reliability is a critical design goal for our TA Management System, as it must consistently perform essential operations such as TA assignments, task approvals, and user management without unexpected failures. Since the academic staff and teaching assistants will use the system to manage real tasks, even small disruptions could lead to confusion or data inconsistencies, so that our system can operate correctly and continuously under expected and unexpected fault conditions. For example, the system is explicitly designed to prevent assigning a task to multiple TAs when it is meant for only one, avoiding conflicts or overlapping responsibilities. We ensure reliability through input validation, structured error handling, and transactional consistency in operations such as task updates and user role changes.

2.0 Design Trade-Offs

2.1 Modifiability vs. Performance

The modular structure that supports modifiability, such as the use of controllers, services, and repositories, introduces extra abstraction layers and well-defined interfaces. While this makes the system easier to change and extend, it also results in additional method calls and indirection, meaning components do not communicate directly but instead go through intermediary layers, for instance, a controller calling a service, which then calls a repository. This added flexibility can slightly degrade performance, especially in frequently accessed or time-sensitive parts of the system, due to the overhead of these layered interactions.

2.2 Reliability vs. Cost

Ensuring reliability requires robust validation, structured error handling, and conflict prevention mechanisms, such as enforcing that a task intended for a single TA is not accidentally assigned to multiple users. Achieving this level of trustworthiness often demands defensive coding, edge-case handling, and comprehensive testing, all of which increase development and maintenance effort. Additionally, if the system were to evolve toward high-availability architectures (e.g., redundant clusters, real-time replication, or long-term log retention), these safeguards would significantly raise hosting, storage, and operational costs. The overall trade-off is that while reliability enhances system robustness, it often comes at the expense of increased complexity, slower development cycles, and higher long-term costs.

3.0 System Decomposition Diagram

We follow a 4-layer architecture consisting of:

- View Layer: Handles all user-facing interfaces. It receives user input and displays data, delegating actions to the Business Logic Layer without containing any core business rules or data access code.

- Business Logic Layer: Implements the core application workflows and rules. It orchestrates services (e.g., authentication, TA assignments, scheduling) and enforces transactional integrity before passing data to repositories.

- Repository Layer: Abstracts persistence operations. It exposes data-access interfaces (e.g. Spring Data JPA repositories) for CRUD (create- read- update- delete) and custom queries, shielding the Business Logic Layer from database details.

- Data Layer: Manages physical storage of information. It comprises the relational database (MySQL) for structured entities and the filesystem for logs, import/export files, and other non-relational assets.

Each layer is responsible for a specific concern and communicates only with adjacent layers to ensure separation of responsibilities.

3.1 Subsystems

3.1.1 View Layer

This layer handles user interaction. Each interface is designed for a specific user role:

- **Login UI:** The authentication gateway includes username/password entry when enabled. It enforces password policies client-side (length, character rules) and displays clear error messages for invalid credentials or expired sessions.
- **Profile Settings UI** – Allows users to manage their personal information.
- **Instructor UI:** Instructors use this dashboard to submit preferred TA lists for their courses for any task, view assigned workloads, and approve or reject workloads.
- **Dean's Office UI:** Responsible for handling proctoring and override requests within the faculty. It provides a unified dashboard for quick decision-making, especially for urgent approvals, without exposing the underlying scheduling complexity.
- **Department Chair/Office UI:** It is to give department chairs fine-grained control over TA assignments: manual override of the restrictions, leave request handling, and weekly scheduling views. It presents conflict alerts in real-time and links directly to the scheduling workflow, so chairs can adjust assignments without navigating through unrelated menus.
- **Admin UI:** The global administration console lets system administrators manage user roles, configure system-wide settings (e.g., notification thresholds), and review audit logs. All data-management screens here enforce strict role-based access control (RBAC) checks, so only admins see sensitive operational metrics or can perform destructive actions.
- **TA UI:** Designed for teaching assistants, this view shows personal schedules, pending leave or swap requests, and proctoring assignments. It blends calendar widgets with inline forms so TAs can submit requests, such as swap, transfer requests, in just a few clicks, without having to navigate away from their dashboard.
- **Exam UI:** Focused on exam-centric workflows, this module lets authorized users create exam time slots, assign proctors, and enforce non-overlap constraints automatically. It visualizes room capacities and time conflicts, preventing double bookings before they occur.
- **Schedule UI:** A unified calendar view that overlays the TA's lessons, task assignments, and other duties, such as proctoring and office hours.
- **Task UI:** This panel handles task assignments, swap requests, and overrides. If a TA wants to request a swap for a task that is currently locked (not swappable), they can submit a request to unlock it. Once approved by an authorized role, the task becomes eligible for swapping and is listed accordingly. The UI also supports bulk task imports and shows all entries in a sortable table with direct links to their workflows.
- **Data Management UI:** Provides Excel import-export screens with schema validation feedback and rollback options for bulk TA uploads or schedule updates. It displays success/failure counts and highlights invalid records in-line, reducing manual cleanup after large data operations.
- **Notification UI:** This panel displays all important alerts and updates for the user, such as task assignments, request approvals, and system messages. Notifications are shown in real-time and can be clicked to view the related task or request. Unread notifications are highlighted, and users can filter them by type to easily find what they need.

3.1.2 Business Logic Layer

This layer implements all core application logic, orchestrates workflows, and enforces business rules. It exposes a clean API to the View Layer and relies on the Repository Layer for persistence.

- **Orchestrator**
  Acts as a high-level façade for complex, cross-cutting workflows (e.g. assignAllTAsAutomatically()). It sequences calls to multiple services, ensures transactional integrity, and centralizes multi-step processes so that individual services remain focused on their single responsibilities.

- **Authentication Service + Token Service + Spring Security**
  Together, these components manage the entire login and authorization lifecycle:

  - **Authentication Service** validates user credentials and applies password policies.

  - **Spring Security** hooks enforce role-based access control (e.g. TA vs. Instructor vs. Admin) across all endpoints.

  - **Token Service** issues, refreshes, and revokes JWTs, securing all subsequent requests.

- **User Service**
  Handles CRUD operations for user accounts and profiles. It validates input data, manages role assignments, and triggers notifications on key events (e.g. account creation, password change).

- **TA Service**
  Manages all TA-specific logic:

  - **Availability & Leave Tracking:** TAs set their working hours and submit leave requests, which are validated and routed through the Request Service.

  - **Eligibility Checks:** Ensures each TA meets department criteria (e.g. minimum GPA, prerequisite trainings) before assignment.

  - **Notification Triggers:** Sends alerts when schedules or assignments change.

- **Instructor Service**
  Empowers instructors to:

  - Submit and revise preferred TA lists for their courses.

  - Approve or reject assigned workloads.

    - Monitor TA performance metrics.
      It integrates with **Course Service** for metadata and **Notification Service** to keep instructors informed of relevant updates.

- **Admin Service**
  Provides system-wide administrative capabilities:

    - Manage user roles and permissions via RBAC policies.

    - Configure global settings (e.g. notification thresholds, exam time-window rules).

    - Access audit logs and perform data-cleanup operations.

- **Task Service**
  Oversees task-assignment workflows:

    - **Assignment & Swap Handling:** Validates time-slots, prevents double-bookings, and coordinates swap requests. If a task is locked against swaps, the service logs TA requests to unlock it and makes it available once approved.

    - **Override Approvals:** Applies business rules for special cases (e.g. emergency reassignment).

    - Throws domain-specific exceptions on rule violations (e.g. invalid time range).

- **Schedule Service**
  Manages scheduling logic and conflict detection:

    - Checks room capacities and time overlaps.

    - Supports batch schedule adjustments and rollback on inconsistency.

    - Publishes events for audit logging and notification.

- **Exam Service**
  Handles exam-related workflows:

    - Creates and updates exam time slots.

    - Assigns proctors while enforcing non-overlap constraints.

    - Integrates with Schedule Service to visualize room and time conflicts.

- **Course Service**
  Maintains course metadata and instructor preferences:

    - Stores TA-preference lists.
    - Publishes domain events when course parameters change.

- **Request Service**
  Routes and tracks all multi-level requests (swap, override, leave):
    - Implements approval chains (TA → Dept. Chair → Dean).
    - Applies business rules at each stage and publishes notifications to involved parties.

- **Email Service**
  Constructs and sends transactional emails (e.g. approval notifications, system alerts).
    - Uses a shared template engine.
    - Implements retry logic for transient failures.

- **Notification Service**
  Drives in-app, real-time updates via WebSockets or Server-Sent Events:
    - Delivers alerts for task changes, approvals, and system announcements.
    - Provides filtering by category and marks unread items.

- **Log Service**
  Captures every significant user or system action (e.g. task approvals, bulk imports, request submissions):
    - Writes immutable records to a centralized audit store.
    - Supports compliance reporting and forensic analysis.

- **Import Service & Export Service**
  Manage bulk data operations with robust feedback:
    - **Parsing & Validation:** Reads CSV (Comma-Separated Values) or Excel files, validates against schema rules, and highlights inline errors.
    - **Transformation & Persistence:** Applies data mappings, writes valid records in transactions, and rolls back on partial failures.
    - **Progress Events:** Emits real-time status updates so the UI can display success/failure counts and detailed error messages.

3.1.3 Repository Layer

- **Database Repository:**Contains Spring Data JPA interfaces (e.g. TARepository, ScheduleRepository, TaskRepository) with support for paging, sorting, and custom queries.

3.1.4 Data Layer

- **Filesystem**
  Stores non-relational data such as logs, import/export files, and backups.

- **Relational Database (MySQL)**
  The primary data store for all structured entities including users, schedules, tasks, exams, and requests.