CS 319 – Object Oriented Software Engineering

Deliverable 3

Spring 2025

Team 12

Ahmet Kenan Ataman <22203434>

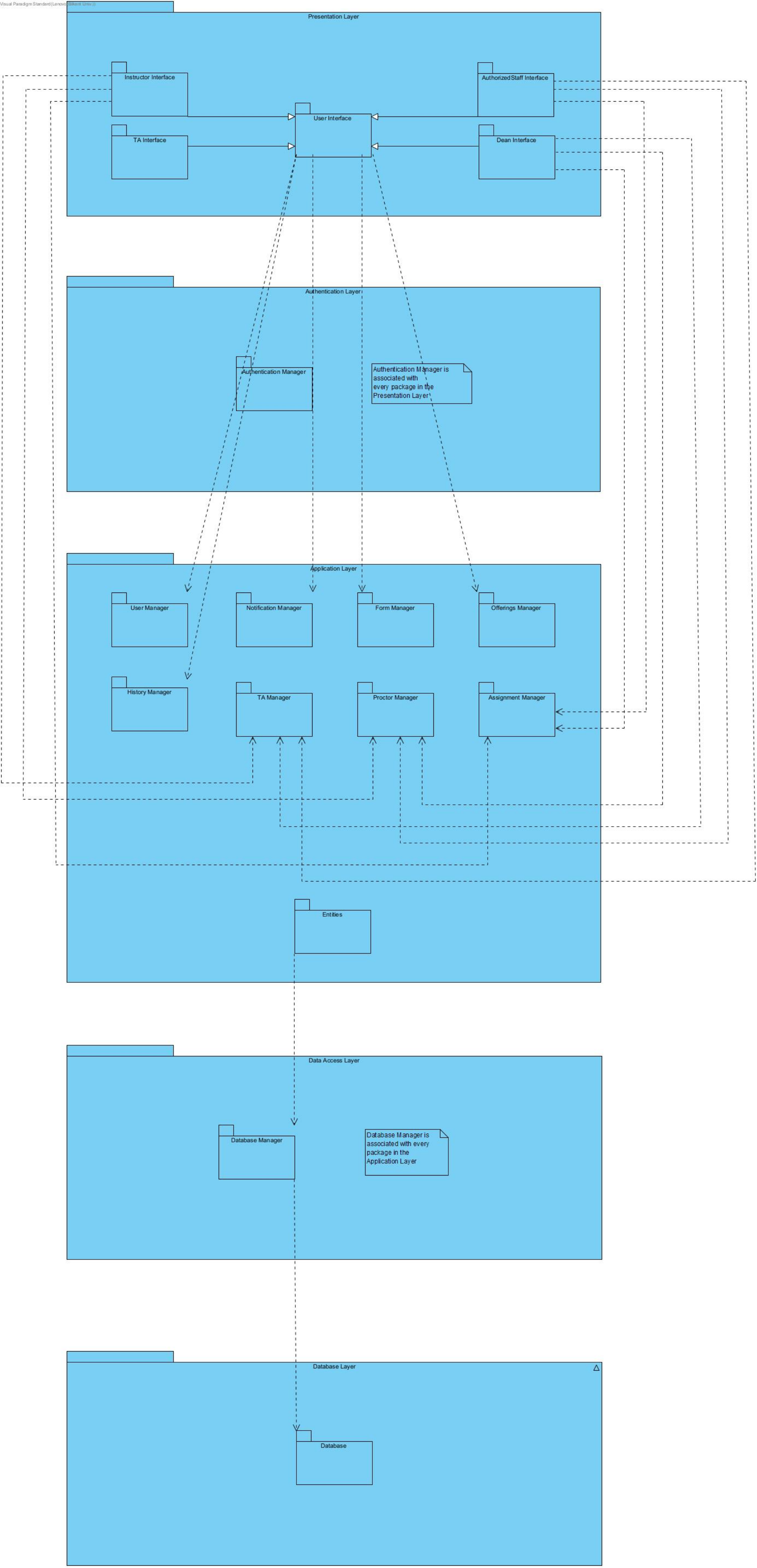
Berfin Örtülü <21802704>

Erdem Uğurlu <22203391>

Gülferiz Bayar <21901442>

Mehmet Emre Şahin <22201765>

TABLE OF CONTENTS


Presentation Layer
Instructor Interface
AuthorizedStaff Interface
User Interface
TA Interface
Dean Interface
Authentication Layer
Authentication Manager
Authentication Manager is associated with every package in the Presentation Layer
Application Layer
User Manager
Notification Manager
Form Manager
Offerings Manager
History Manager
TA Manager
Proctor Manager
Assignment Manager
Entities
Data Access Layer
Database Manager
Database Manager is associated with every package in the Application Layer
Database Layer
Database

1. High-Level Software Architecture

1.1 Subsystem Decomposition

For our TA Management System, we structured our architecture around a **layered design** to reflect both the complexity and the modularity we needed. Since the project involves multiple user roles—each with different responsibilities and privileges—we wanted each part of the system to have a clear purpose and responsibility. This not only makes our system more maintainable but also helps us divide the work efficiently as a team.

We ended up with five major layers: **Presentation**, **Authentication**, **Application**, **Data Access**, and **Database**. Each one plays a specific role in turning user actions into real-time backend operations.

1.1.1 Presentation Layer (Frontend)

This is the part of the system users actually see and interact with. We built it using React.js, and every role—TAs, Instructors, Admins, and Deans—has their own personalized dashboard. We wanted to keep this layer clean and intuitive, making sure that even users unfamiliar with technical systems could quickly get used to the interface.

User Interface

- Acts as a hub for routing to role-specific views.
- Connected with TA Interface, Instructor Interface, Dean Interface, and Authorized Staff Interface.
- **Responsibilities**:
    - Acts as the central navigation hub, directing users to their role-specific views upon authentication.
    - Ensures consistent layout and routing logic across all pages.
- **Associated With**:

    - TA Interface, Instructor Interface, Dean Interface, Authorized Staff Interface (via conditional rendering)
    - Authentication Manager (for role-based access control)
    - Notification Manager (for displaying system alerts)

TA Interface

- Allows TAs to log workload, request leave, view proctor assignments.
- Interacts with the User Interface and backend via workload and leave-related endpoints.
- **Responsibilities**:
    - Allows TAs to log lab sessions, grading, and other workload entries.
    - Enables leave request submission and displays approved or pending leaves.
    - Displays upcoming proctoring assignments and their statuses.
- **Associated With**:
    - Form Manager (for workload and leave form submissions)
    - TA Manager (to view and update workload summaries)
    - Proctor Manager (to display upcoming assignments)
    - Notification Manager (to receive leave/proctoring notifications)

Instructor Interface

- Used to assign proctoring tasks, approve leave requests, and manage course offerings.
- **Responsibilities**:
    - Enables instructors to assign proctoring tasks (manually or automatically).
    - Facilitates approval or rejection of TA leave requests.
    - Supports the creation and editing of course offerings.

- **Associated With**:
    - Proctor Manager (for viewing and managing proctoring logic)
    - Form Manager (for leave approval workflows)
    - Offerings Manager (for managing course-section relations)
    - Assignment Manager (to finalize and review assignments)

Dean Interface

- Enables viewing and editing inter-department proctoring operations.
- Supports centralized exam coordination and cross-department collaboration.
- **Responsibilities**:
    - Supports high-level proctor coordination across departments.
    - Displays aggregated assignment data for multi-department exams.
    - Provides override features in case of TA shortages.
- **Associated With**:
    - Assignment Manager (for viewing finalized schedules)
    - Proctor Manager (for override logic and manual assignment)
    - Notification Manager (to inform involved departments)

Authorized Staff Interface

- Offers advanced settings, user management, and override controls for proctoring.
- **Responsibilities**:
    - Provides access to system-wide settings, TA list management, and override features.
    - Supports emergency adjustments and system configuration.
- **Associated With**:
    - User Manager (for editing user roles and information)

        - TA Manager (for tracking TA performance and workload)
        - Assignment Manager (to review and override proctor decisions)

1.1.2 Authentication Layer

Authentication is at the core of our access control. We didn't want to build a system where anyone could access any page, so we implemented secure, **token-based authentication** using JWT.

Authentication Manager

- Validates user credentials during login.
- Attaches roles to issued tokens for controlling access to all frontend packages.
- Associated with every package in the Presentation Layer.

1.1.3 Application Layer

This layer handles the business logic and data validation for core application features. Each manager below is a modular Django component responsible for specific domain logic.

User Manager

- Handles user profile updates, role changes, and account removal.
- **Responsibility**: Manages user registration, profile updates, role assignments, and account-related logic.
- **Associated With**:
    - TA Interface, Instructor Interface, Admin Interface (via User Interface)
    - Authentication Manager (for user-role verification)
    - Database Manager

Notification Manager

- Sends system messages and alerts such as “Leave Approved” or “Proctor Assigned.”
- **Responsibility**: Sends real-time system notifications (e.g., assignment confirmations, leave approval updates) to appropriate users based on triggered events.
- **Associated With**:
    - TA Manager, Assignment Manager, Form Manager (to listen to system events)
    - User Interface (to display alerts)
    - Database Manager

Form Manager

- Handles validation and storage of user-submitted forms (leave, workload).
- **Responsibility**: Handles submission and validation of workload entries and leave forms. It ensures that each request complies with system rules before processing.
- **Associated With**:
    - TA Interface, Instructor Interface
    - TA Manager, Leave Approval Logic
    - Database Manager

Offerings Manager

- Manages course and section offerings, tracks assigned TAs and instructors.
- **Responsibility**: Manages the creation and tracking of course offerings, including instructor assignments and classroom capacity links.
- **Associated With**:
    - Instructor Interface (for course setup)
    - TA Manager, Assignment Manager (to match TAs to courses)
    - Database Manager

TA Manager

- Maintains and calculates individual TA workload.
- Considers approved leave when computing availability.
- **Responsibility**: Maintains individual TA workload records, availability, and assignment eligibility. It is central to workload fairness and proctoring decisions.
- **Associated With**:
  - Proctor Manager, Assignment Manager
  - Form Manager (for workload input)
  - TA Interface (for TA feedback)
  - Database Manager

Proctor Manager

- Automates proctoring assignment based on availability and load.
- Includes logic for PHD/MS priority and swap eligibility.
- **Responsibility**: Manages proctor assignment—both automated and manual—based on availability, leave status, and workload balance.
- **Associated With**:
  - Assignment Manager (for finalizing assignments)
  - TA Manager (for load info)
  - Instructor and Dean Interfaces
  - Database Manager

Assignment Manager

- Finalizes schedules and maintains assignment history logs.
- Interacts with both Proctor and TA Managers.
- **Responsibility**: Finalizes and stores proctoring assignments, tracks proctor swaps, and logs every assignment event for traceability.

- **Associated With**:
    - TA Manager, Proctor Manager
    - Notification Manager (to send assignment confirmations)
    - Instructor Interface, Dean Interface
    - Database Manager

History Manager

- Archives completed actions (e.g., past leaves, fulfilled assignments).
- Supports audit and transparency features.
- **Responsibility**: Stores historical data related to assignments, leave requests, and form submissions to ensure accountability and reporting.
- **Associated With**:
    - Assignment Manager, Form Manager
    - User Interface (for report viewing)
    - Database Manager

All managers rely on the central **Entities** module, which contains the domain models.

1.1.4 Data Access Layer

Rather than having our business logic interact directly with the database, we built this layer to act as a **bridge**. It made the code cleaner, more reusable, and easier to debug.

Database Manager

- Exposes generic and reusable CRUD operations for all entity types.
- Connected to every package in the Application Layer for database operations.

1.1.5 Database Layer

Finally, the database. We used MySQL because it’s structured, reliable, and works well with Django’s ORM. All our persistent data lives here—workloads, users, courses, classrooms, assignments, logs, and more.

Database

- Contains relational tables for TAs, instructors, assignments, workloads, classrooms, and leave forms.
- Handles indexing, data integrity, and foreign key enforcement.

All in all, from the early stages of development, our team prioritized architectural clarity and maintainability. Rather than adopting a tightly coupled structure in which components interact indiscriminately, we deliberately chose a layered and modular design. This approach allowed team members to take ownership of specific layers—such as the presentation, business logic, or data access—while ensuring consistency and integration across the system. Furthermore, this modular decomposition enhances the system’s scalability and extensibility. Future enhancements, such as the addition of TA feedback mechanisms, mobile interface support, or analytical dashboards, can be seamlessly integrated by extending or adding to the appropriate layer without necessitating major refactoring of the existing architecture.

2. Design Goals

Since our project aims to manage the responsibilities of Teaching Assistants (TAs) in a fair, scalable, and user-friendly way, our design goals are driven by what different user types (TAs, Instructors, Admins, and the Dean’s Office) need the most. These needs became clear as we analyzed the current manual workload assignment and proctoring systems at Bilkent University. The system is still under development, so the design goals reflect both our final vision and the practical constraints of a term-long project.

2.1 Functionality

Our main goal is to provide **complete and meaningful functionality** across all user roles. TAs can log lab work, submit leave, and monitor their semester workload. Instructors and department staff can assign proctoring duties both manually and automatically, manage classroom info, and monitor TA availability. Meanwhile, the Dean's Office can handle centralized exam scheduling across departments.

This wide range of features was essential because existing processes (spreadsheets, email chains, manual tracking) were fragmented and inefficient. Our system combines them under one roof and ensures that proctoring, leaves, and workloads are all interlinked for accurate conflict management. The system even updates workload automatically when proctoring is assigned or swapped.

> These functionalities are our top priority because they directly solve the pain points of current academic staff and TA workflows.

2.2 Usability

We designed the platform with **clarity and role-specific simplicity** in mind. Each role sees only the tools they need. For example, a TA's dashboard includes "Submit Workload," "Request Leave," and "View Assignments," while an instructor has access to "Assign Proctor," "Approve Leave," and "Classroom Management."

The UI is responsive across devices, with a sidebar layout and clean page sections to avoid clutter. Key actions can be completed within 2–3 clicks, and pop-ups help guide the user instead of redirecting them through deep menus.

We wanted even users unfamiliar with technical systems (like department secretaries or newly hired TAs) to comfortably navigate without external support.

> In short, system usability was not an afterthought—it was a primary design concern to ensure adoption and minimize confusion across departments.

2.3 Rapid Development

Since this is a term-long team project, we emphasized **rapid development and iteration** using an agile workflow. We broke the project into functional blocks—leave management, workload tracking, user roles, and proctoring—and assigned ownership within the team. By prioritizing reusable Django components, React modular components, and REST APIs, we managed to keep the codebase clean and adaptable.

Reusable forms and components (e.g., for user filtering, workload entry, or pop-up windows) helped us save time while maintaining consistency. We also used mockups early on to reduce guesswork during frontend development.

Although we had many features in mind, focusing on fast iteration helped us deliver the core MVP early and polish it through weekly milestones.

3. Design Trade-offs

3.1 Functionality vs. Usability

We realized early that squeezing every feature into one page would overwhelm users. Instead of mixing all controls together, we created separate pages for leave management, workload submission, and proctoring assignments. For quick interactions (like changing TA workload or confirming a leave request), we use modal windows instead of full navigation.

This balance helps us maintain both advanced functionality and ease of use without overwhelming users.

3.2 Rapid Development vs. Functionality

Given the timeline, we had to limit the scope of some features, especially those that required heavy logic (e.g., real-time conflict resolution across departments). Instead, we implemented fallback options like override warnings and simplified rules. We chose to focus on the **most valuable and frequently used features first**, leaving room for future expansion.

To manage this trade-off, we kept a feature backlog and made clear milestone goals for every 1–2 week sprint cycle.