CS 319 – Object Oriented Software Engineering

Deliverable 3

Spring 2025

Team 12

Ahmet Kenan Ataman <22203434>

Berfin Örtülü <21802704>

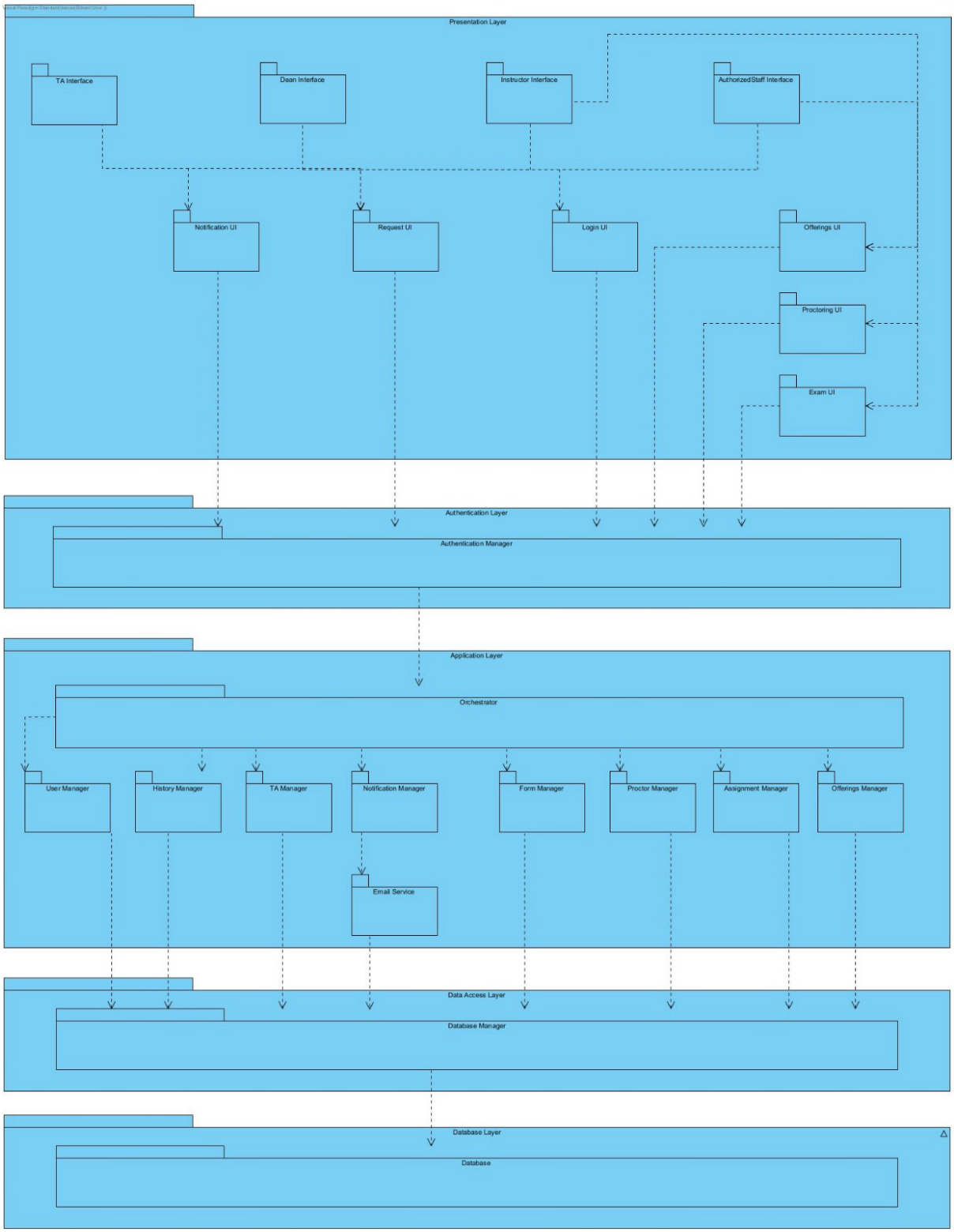
Erdem Uğurlu <22203391>

Gülferiz Bayar <21901442>

Mehmet Emre Şahin <22201765>

Table of Contents

1. Subsystem Decomposition Diagram



2. High-Level Software Architecture

2.1 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, 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.

2.1.1 Presentation Layer

This is the part of the system that 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. This layer works consistently with the Authentication and Application Layers.

TA Interface

- Allows and Navigates TAs to log workload, create leave requests.
- Displays upcoming proctoring assignments and other notifications.
- This package is associated with Login UI, Request UI, Notification UI packages.

Dean Interface

- Supports centralized exam coordination and cross-department collaboration.
- Allows and Navigates Deans to view and edit inter-department proctoring operations.
- Displays aggregated assignment data for multi-department exams.
- This package is associated with Login UI, Request UI, Notification UI packages.

Instructor Interface

- Used to assign proctoring tasks, approve leave requests, manage exams and manage course offerings.
- Displays upcoming exams and their status, whether proctoring operations have been made or not.
- This package is associated with Login UI, Request UI, Notification UI, Proctoring UI, Offerings UI, Exam UI packages.

Authorized Staff Interface

● Offers advanced settings, user management, and override controls for proctoring.

● Provides access to system-wide settings, TA list management, and override features.

● Used to assign proctoring tasks, approve leave requests, manage exams and manage course offerings as well as Instructors.

● This package is associated with Login UI, Request UI, Notification UI, Proctoring UI, Offerings UI, Exam UI packages.

Login UI

● Navigates Users to their pages.

● This package is associated with the Authentication Manager package.

Request UI

● Allows users to create requests, such as leave requests or swap requests.

● This package is associated with the Orchestrator package.

Notification UI

● Users see their notifications here.

● This package is associated with the Orchestrator package.

Proctoring UI

● Authorized Staff, Dean, and Instructors set the proctoring settings here either manually or automatically.

● They are able to filter and set restrictions here.

● This package is associated with the **Orchestrator** package.

Offerings UI

● Authorized Staff, Dean, and Instructors can add/edit/delete offerings such as course information, lab information, and classroom information.

● This package is associated with the **Orchestrator** package.

Exam UI

● Authorized Staff, Dean, and Instructors can add/edit/delete exams and their information here.

● This package is associated with the **Orchestrator** package.

2.1.2 Authentication Layer

This layer handles the login, logout, session, and credential validation operations. It ensures that only authenticated users access the system and routes their requests securely.

Authentication Manager

- Regulates the authentications of the users (login, session management, credential verification).
- This package connects the Login UI to the core system logic and ensures only valid users pass to the Application Layer.

2.1.3 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.

Orchestrator

- The package where the connection between the Presentation layer and other packages is made.
- Passes the information from the Presentation layer to the Application layer.
- This package is associated with **User Manager, History Manager, Authentication Manager, Notification Manager, Form Manager, Proctor Manager, Assignment Manager, Offerings Manager** packages.

User Manager

- Handles user profile updates, role changes, and account removal.
- Manages user registration, profile updates, role assignments, and account-related logic.
- This package is associated with the **Database Manager** package

Notification Manager

- Sends real-time system messages and alerts such as “Leave Approved” or “Proctor Assigned.”
- This package is associated with the **Email Service** package.

Offerings Manager

- Manages the creation and tracking of course offerings, including instructor assignments.
- This package is associated with the **Database Manager** package.

History Manager

- Stores historical data related to assignments, leave requests, and form submissions to ensure accountability and reporting.

● This package is associated with the **Database Manager** package.

Email Service

● Sends the notifications via email.

● This package is associated with the **Database Manager** package.

Form Manager

● Handles submission and validation of workload entries and leave forms. It ensures that each request complies with system rules before processing.

● This package is associated with the **Database Manager** package.

Assignment Manager

● Finalizes and stores assignments other than proctoring, such as labs, and logs every assignment event for traceability.

● This package is associated with the **Database Manager** package.

TA Manager

● Maintains individual TA workload records, availability, and assignment eligibility. It is central to workload fairness and proctoring decisions.

● This package is associated with the **Database Manager** package.

Proctor Manager

● Manages proctor assignment—both automated and manual—based on availability, leave status, and workload balance.

● This package is associated with the **Database Manager** package.

2.1.4 Data Access Layer

Rather than having our **Application Layer** 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.

● Handles data retrieval and storage operations.

● Connected to every package in the Application Layer for database operations.

2.1.5 Database Layer

We used MySQL for the database. 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.

● Manages the retrieval and storage of data from databases.

● Handles indexing, data integrity, and foreign key enforcement.

3. Design Goals

3.1 Usability

We designed the platform with clarity and role-specific simplicity in mind. Each role sees only the tools they need, allowing them to focus on the tasks most relevant to their responsibilities. 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." This clear separation ensures that users are never overwhelmed with irrelevant options, keeping the interface clean and intuitive.

The UI is responsive across devices, with a sidebar layout and clean page sections to avoid clutter. Most key actions can be completed within two or three clicks, which was a deliberate design choice to maximize efficiency and reduce frustration. But usability for us was not just about minimizing clicks it was also about making the pages self-explanatory and self-guiding. Simply looking at the dashboard or section pages gives the user a clear sense of what actions are available, what each part does, and where to go next, even without external instructions.

We paid careful attention to grouping related tasks under logical headings, using consistent icons and color cues to make important actions stand out, and providing inline helper text or tooltips when necessary. As a result, over 90% of essential workflows can be completed without leaving the main dashboard, and users can confidently carry out tasks like submitting a leave request or assigning a proctor without needing to navigate through deep or hidden menus.

To sum up, system usability was not an afterthought it was a primary design concern, shaping both the visual and functional aspects of the system to ensure quick adoption, reduce error rates, and minimize confusion across departments. We intentionally balanced efficiency and clarity, making sure that the system offers both simplicity for everyday use and enough flexibility to handle complex operations when needed.

3.2 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.

4. Design Trade-offs

4.1 Rapid Development vs. Functionality

Given the time constraints of this academic project, our team adopted a rapid development approach that required careful prioritization of functionality. We recognized early on that it would not be feasible to implement every possible feature at full depth within the limited project timeline. Therefore, we strategically organized the system's functional requirements by importance, ranking them according to their criticality, expected impact, and direct contribution to user value.

At the top of this priority list, we placed the essential and high-impact features; such as user authentication, workload logging, leave management, proctoring assignment (both manual and automated), and basic notification delivery; because they form the backbone of the system's purpose and ensure that the core workflows function smoothly. These are the non-negotiable components that deliver immediate value to the system's users and enable the platform to replace existing manual or fragmented processes.

Meanwhile, features that were lower on the priority scale were consciously scheduled for the later stages of development or left out of the initial delivery. By deferring these less critical elements, we ensured that the most valuable and frequently used functionalities were developed with greater attention to reliability and efficiency, without overextending our team's limited time and resources.

Importantly, during this prioritization process, we also proactively considered potential corner cases and failure scenarios, making sure that even the minimum version of each core feature would handle edge conditions gracefully (e.g., conflicting assignments, overlapping leaves, or incomplete approvals). This decision-making balance allowed us to deliver a robust and useful project while ensuring that the features left for future expansion would not critically undermine the system's core operations or user experience in the short term.

It is worth emphasizing that rapid development has been our main design goal from the beginning. As a result, we worked carefully to maintain a balance — focusing on delivering a complete and polished system within the project timeline, while ensuring that postponing certain lower-priority features would not compromise the system's reliability or its ability to deliver meaningful value to its users.

4.2 Usability vs. Security

In the design of the TA Management System, we were consistently aware that security and usability stand in natural tension with each other. On one hand, robust security measures — such as multi-layer authentication, advanced access controls, and strict compliance with security standards — are essential for protecting sensitive data, including personal TA records, workload histories, and proctoring assignments. However, we also recognized that adding too many security layers can significantly reduce usability, making the system more cumbersome for everyday users.

For example, while advanced security features like multi-factor authentication or strict password complexity requirements enhance system safety, they can also introduce friction that slows down routine actions like logging in or submitting leave requests. Similarly, adapting the system to comply with formal security standards or porting it to other platforms (such as desktop or mobile environments) would require reworking many of the underlying security mechanisms, which in turn could affect the simplicity and intuitiveness of the user experience.

We deliberately balanced these two goals in our design:

- ◊ On the usability side, we ensured that users (TAs, instructors, staff) could quickly and easily perform key actions, such as logging in, navigating their dashboards, submitting workload records, or managing proctoring assignments, without excessive barriers. For example, the system allows participants to access essential features like screen sharing or task swapping without requiring additional authentication at every step.
- ◊ On the security side, we implemented core protections, such as secure authentication, role-based access control, and proper data handling, while deferring more advanced compliance work to later development phases. We acknowledged that adding advanced features too early could harm usability, especially for non-technical users, and we prioritized a smooth, intuitive experience for the project's academic setting.

It's important to highlight both usability and security. Our team worked carefully to strike a balance, ensuring the system remained approachable and efficient for its intended users, while laying the groundwork for future expansions where deeper security enhancements (and possible platform adaptations) can be integrated without undermining the system's core usability.