TA Management System

Deliverable-3 2nd Iteration

05.05.2025

Course: CS 319

Section: 01

Group No.: 7

Instructor: Eray Tüzün

Teaching Assistant: Utku Boran Torun

Group Members: Bilge İdil Öziş [22102365]

Burak Kağan Gür [22203365]

Daib Malik [22201013]

Yunus Günay [22203758]

1. Top Two Design Goals:

a. **Usability:** Usability is the most critical design goal for the TA Management System. The application must serve a diverse user base that includes instructors, teaching assistants, dean, and secretary – each with different responsibilities and expectations. To address this, the system is designed to be intuitive, task-oriented, and accessible for all users, regardless of technical proficiency. Instructors should be able to assign TAs, review workload distributions, and configure preferences in no more than 7 clicks, without needing training. TAs must find it easy to view their assigned courses, submit leave or duty requests, and track feedback without navigating complex menus, and again within 7 clicks once they are authenticated by the system. Some instructors need a clear overview of the entire assignment matrix, including over-/under-loaded TAs, and must-have preferences. Similarly, the dean and department secretary may need to access reports, see charts, and accept/reject requests within 8 clicks. All of the above mentioned actions take no more than 4 seconds to take effect.

To achieve the best user experience, our dashboard and sidebar are tailored to each role and present only relevant options. Icons and color-coded badges help visually distinguish TA categories (e.g., must-have, avoided, preferred), while tabs and searchable tables help users find and act on information in under 2 seconds. All buttons that save information are blue and ones that risk potential deletion are colored red, so the user can make decisions

without hesitation. All delete options also ask for confirmation before deleting anything. Dialog boxes and inline warnings guide users during sensitive actions (like overriding must-have TA constraints), reducing error rates. A red notification dot with a number in it for unread notifications and alert system ensures that urgent updates (e.g., a TA's leave approval or a reassignment) are never missed. We also avoid clutter by using collapsible sections and categorized filters, enabling users to focus on their tasks without distraction. The sidebar has proper, intuitive names like Reports, TA Assignment, Proctoring Assignments, and more that won't require a user to guess which functionality lies where. This makes it easy for a new user to settle in quickly and navigate their way smoothly. We made sure to follow a design layout that is common with most applications these days that users are familiar with. For example, the Logout button always appears in the sidebar, so there is no need to go to a specific page (like settings or profile) to exit. Contact button shows immediately on the dashboard, so there is no need to go through a list of names/emails on a different page to contact a particular user. Additionally, paths to all critical operations are just 1 or 2 clicks away from the main dashboard, minimizing user effort. Lastly, real-time feedback, such as success toasts and validation warnings, help guide users to correct outcomes.

b. **Reliability:** Reliability is the second key design goal of the TA Management System. The platform must consistently perform critical functions such as assigning TAs, validating workloads, and

tracking preferences without failure. For example, when an instructor assigns a TA to a course, the system must accurately calculate the resulting workload and prevent violations of minimum or maximum thresholds. If a TA is removed or unavailable, the system must not crash or return corrupted data. We need to keep track of the correct workload distribution while other actions like swapping. Even a load differing by 1 or 2 units can burden the TA substantially. Reliability is ensured by addressing edge cases like avoiding duplicate TA assignments to the same course or preventing grader over-assignment beyond the instructor-specified count. At the same time, we ensure proper load allotment to activities to make sure an action like proctoring and overseeing a lab is not the same. This ensures the TA's workload is assigned fairly.

We rely on Django's ORM to enforce database integrity and transactional safety, so all assignment operations are atomic – either they succeed completely or not at all. Backend logic includes validation to avoid common pitfalls like assigning non-existent users or missing "must-have" TAs without proper acknowledgment. The system also provides fallback behaviors, if the API fails during development or testing. Role-based access control (RBAC) ensures that only authorized users can make sensitive changes, and all assignment actions are logged for later traceability. Updates to allocations, preferences, or leave requests are reflected in real-time, so users always view the latest state of the system, preventing outdated information from influencing decisions.

2. Trade-offs:

a. **Reliability vs. Performance:** To ensure accurate and up-to-date allocation data, we re-fetch and recalculate TA loads and preferences upon every page refresh or assignment change. So even when reports are being generated, they fetch data in real-time on every click. While this improves reliability by preventing stale views, it can introduce slight delays or increase server load during peak times. Instructors accessing many courses simultaneously may experience slower response times due to the backend's thorough validation routines. Caching or partial fetches could improve performance but at the cost of up-to-date accuracy, hence we prioritize reliability.

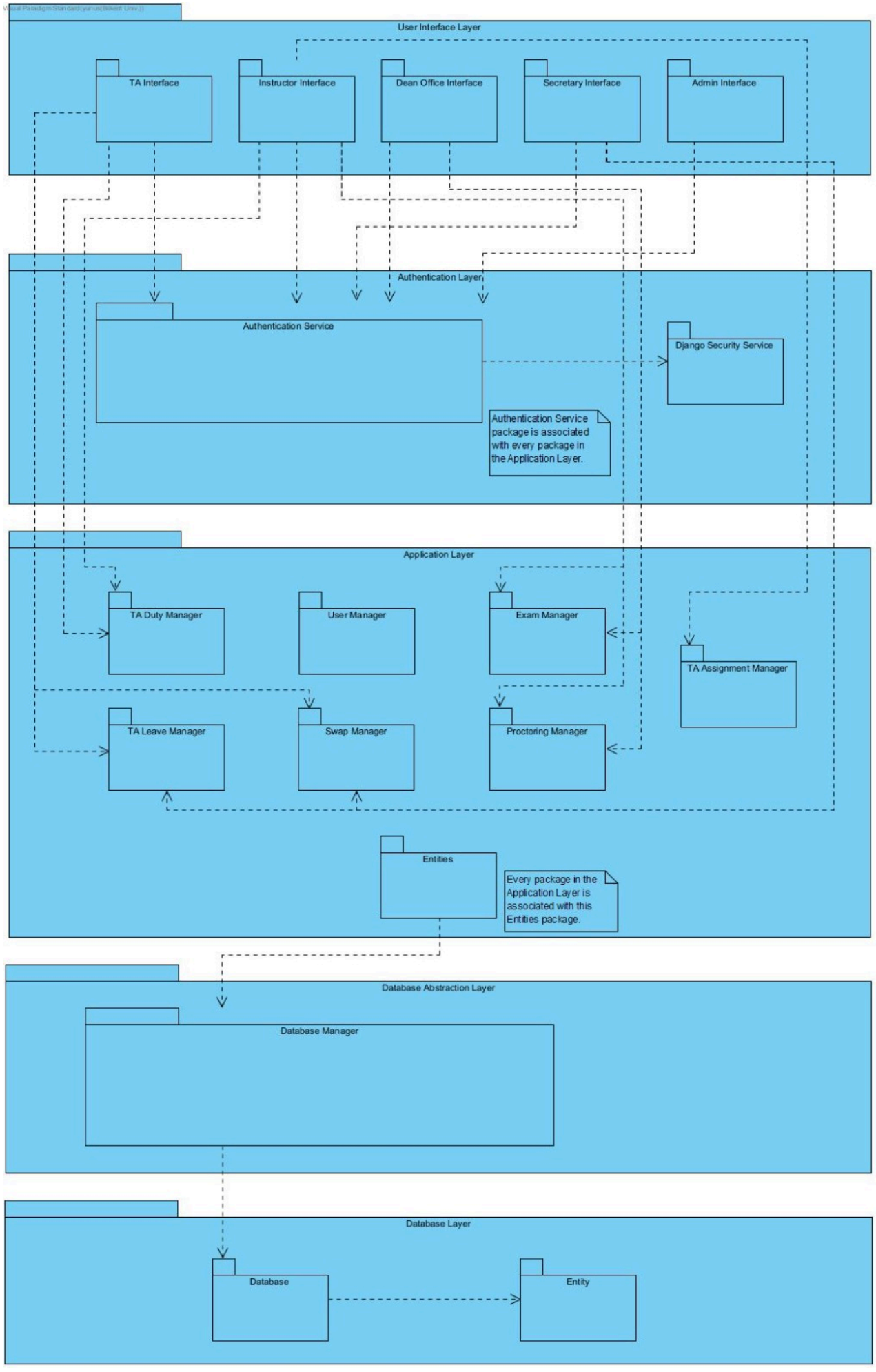
b. **Usability vs. Security:** Usability and security often exist in a state of tension, and this is particularly true in multi-role educational systems like our TA Management platform. To maximize usability, we aim to minimize friction – avoiding unnecessary login prompts, reducing the number of confirmation dialogs, and allowing features like auto-saving or pre-filled forms. For instance, instructors are able to assign TAs within 5 clicks (from dashboard), and TAs can submit requests without navigating deep forms. However, prioritizing usability

can open up potential vulnerabilities. A highly usable system might allow TAs to remain logged in for extended sessions or expose user-related data (like workload stats or assignments) to roles that don't strictly require it. To address this, we implement role-based access control (RBAC), session timeout limits, and conditional visibility at the UI level – but these protections sometimes result in extra clicks, access denial popups, or context switches, which slightly hurt the seamless experience.

c. **Usability vs. Auditability:** Since usability is our primary goal, the system emphasizes simplicity and ease for TAs. For example, any eligible TA can click "Swap" and send a request without needing to understand internal constraints. This low-friction design minimizes user frustration and encourages fast swaps during exam season. However, this usability focus comes at the cost of full auditability. Auditability refers to the system's ability to track, reconstruct, and explain what happened – and why. Currently, the app does not log blocked swap attempts or reasons behind rejected requests. This makes it harder to investigate bugs, or disputes/conflicts. While enforcing rules like "only one swap request per exam slot" helps avoid conflicts, it

adds complexity. If every blocked or invalid action were logged (e.g., "TA X tried to request TA Y, but Y already has a pending request"), the system would be more transparent – but at the expense of storage and minor performance overhead. That said, the system does support auditability to a reasonable extent, by recording who requested whom, when, and for which exam – enough for basic traceability.

3. Subsystem Decomposition Diagram



We implemented a 5-layer subsystem decomposition to improve the system's structure and functionality. The system is divided into five distinct layers: the User Interface, Authentication, Application, Database Abstraction, and Database layers, each responsible for a specific set of tasks.

a. **User Interface Layer:** This layer contains multiple role-specific interfaces such as TA Interface, Instructor Interface, Dean Office Interface, Secretary Interface, and Admin Interface. These interfaces handle user interactions and forward requests to the Authentication Layer.

TA Interface: This package provides screens for Teaching Assistants to view duties, request leaves, and initiate swaps. This package is associated with the Authentication Service, TA Duty Manager, TA Leave Manager, and Swap Manager.

Instructor Interface: This package allows Instructors to create exams, assign TAs, and monitor proctoring. This package is associated with the Authentication Service, Exam Manager, TA Assignment Manager, TA Duty Manager, and Proctoring Manager.

Dean Office Interface: This package offers high-level staffing and exam logistics oversight for the Dean’s office. This

package is associated with the Authentication Service, Proctoring Manager, and Exam Manager.

Secretary Interface: This package supports Secretaries in maintaining course data. This package is associated with the Authentication Service, TA Leave Manager.

Admin Interface: This package provides system-wide configuration and role management. This package is associated with the Authentication Service.

b. **Authentication Layer:** This layer sits between the User Interface and Application Layers to handle login, session/token management, and permission checks.

Authentication Service: This package functions as a central gateway for login, and per-request role enforcement. This package is associated with all UI packages and every package in the Application Layer.

Django Security Service: This package wraps Django's built-in security utilities (password hashing, CSRF middleware, permission decorators). This package is associated with the Authentication Service.

c. **Application Layer:** This layer handles all the backend functionalities. The controller classes are included in this layer.

TA Duty Manager: This package handles weekly duty rosters and schedules, detects and resolves conflicts depending on the assignment of duties. This package is associated with the Entities package, Authentication Service, and related user interface packages.

TA Leave Manager: This package processes TA leave requests and triggers automatic duty reallocation. It is associated with the Entities package, Authentication Service, and related user interface packages.

Swap Manager: This package swaps the TAs in their assigned proctoring duty with another TA based on availability and qualifications. This package is associated with the Entities package, Authentication Service, and related user interface packages.

Exam Manager: This package keeps track of exams and integrates proctoring needs. It is associated with the Entities package, Authentication Service, and related user interface packages.

Proctoring Manager: This package allocates proctors and handles last-minute substitutions. This package is associated with the Entities package, Authentication Service, and related user interface packages.

User Manager: This package handles CRUD operations for all user roles and bulk imports. It is associated with the Entities package, Authentication Service, and related user interface packages.

Entities: This package consists of the classes that have setters and getters, such as Exam, Proctoring, etc. This package is associated with every package in the Application Layer.

d. **Database Abstraction Layer:** This layer acts as a facade that hides the complexity of low-level data access, allowing the application to work with the database in a consistent and technology-independent way. So it includes the wrapper class of the database.

Database Manager: This package provides a clean and consistent API for all database operations. It handles transactions, coordinates data access across repositories, and shields the Application Layer from direct interaction with

ORM or SQL logic. This package is associated with all Application Layer packages and the Database package.

e. **Database Layer:** This layer holds and handles the physical data of the application.

Database: This package contains all tables, indexes, and constraints. It is associated with the Database Manager.

Entity: This package contains physical schema artefacts and it is associated with the Database package.