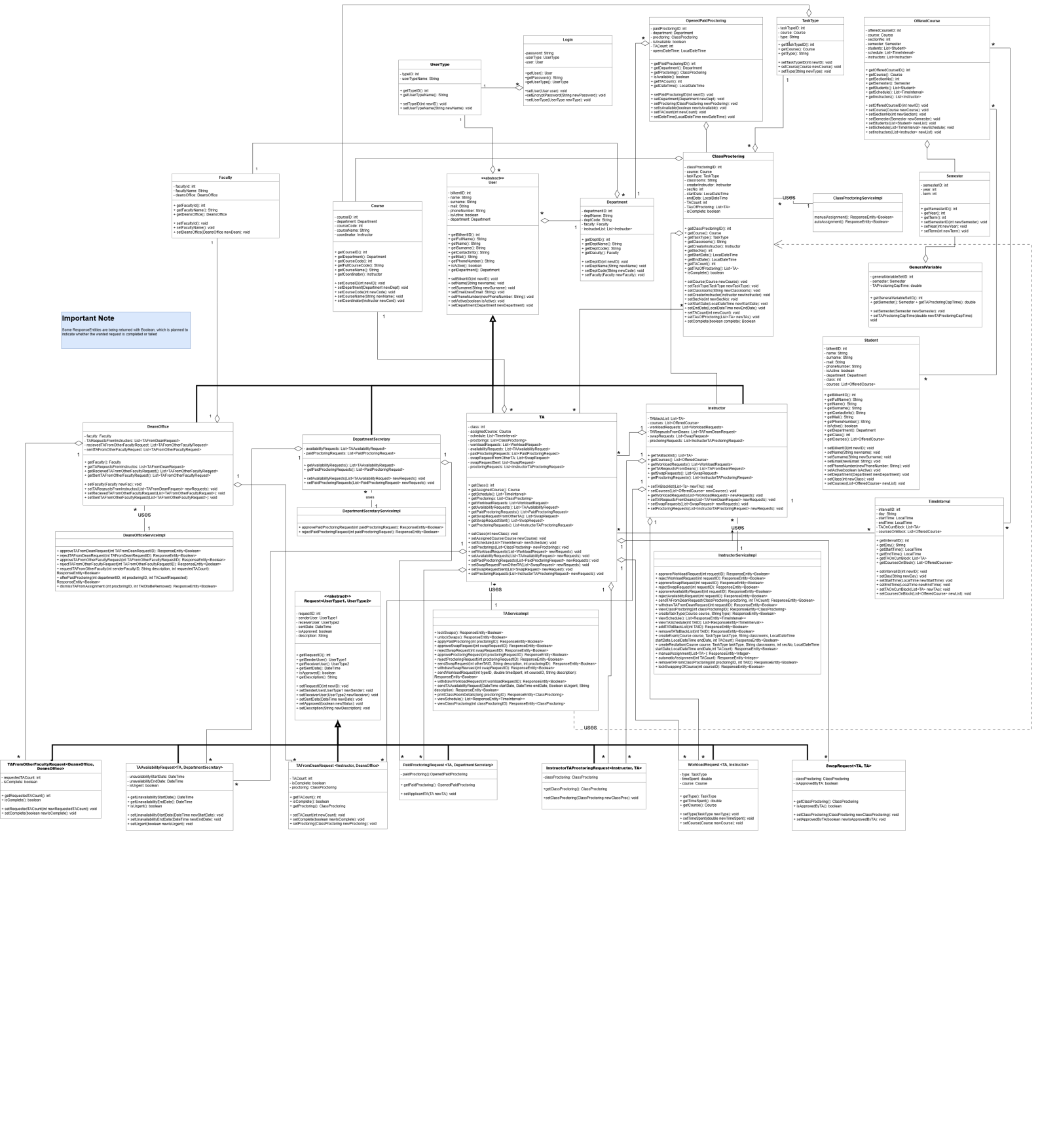
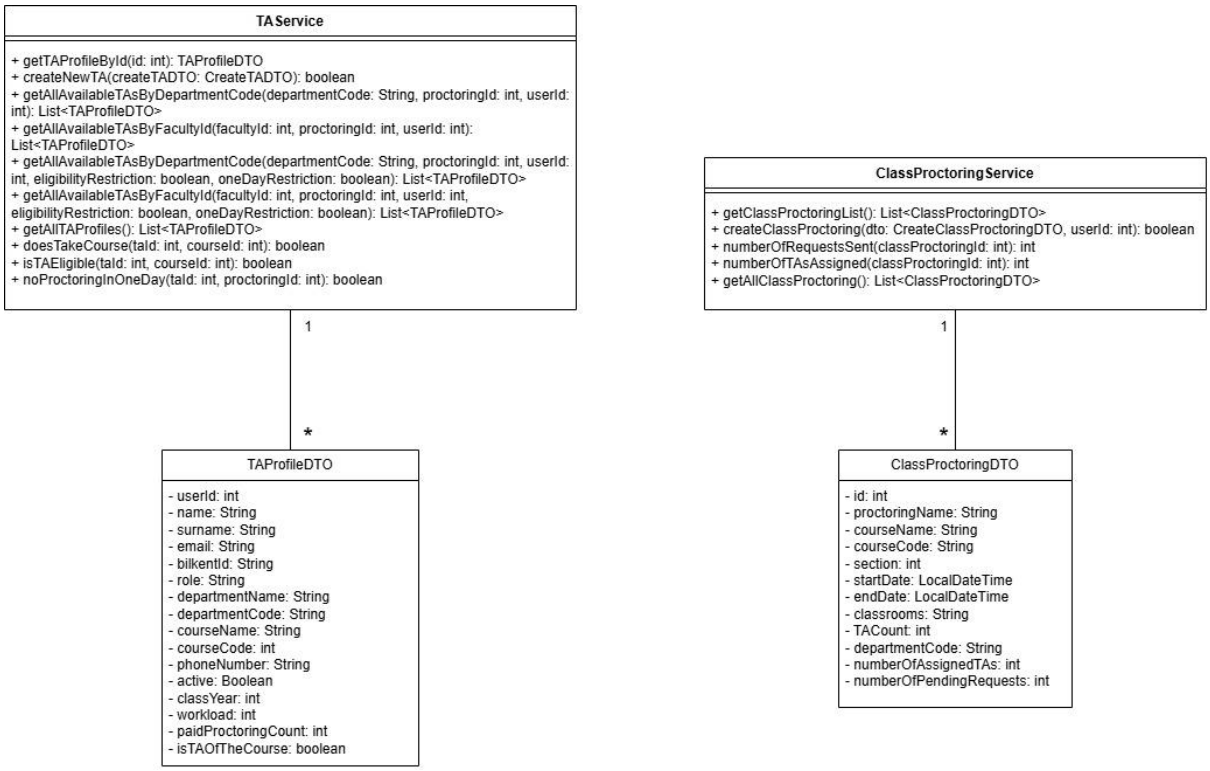

UserType
- typeID: int
- userTypeName: String
+ getTypeID(): int
+ getUserTypeName(): String
+ setTypeID(int newID): void
+ setUserTypeName(String newName): void
Login
-password: String
-userType: UserType
-user: User
+getUser(): User
+getPassword(): String
+getUserType(): UserType
+setUser(User user): void
+setEncryptPassword(String newPassword): void
+setUserType(UserType newType): void
OpenedPaidProctoring
- paidProctoringID: int
- department: Department
- proctoring: ClassProctoring
- isAvailable: boolean
- TACount: int
- openedDateTime: LocalDateTime
+ getPaidProctoringID(): int
+ getDepartment(): Department
+ getProctoring(): ClassProctoring
+ isAvailable(): boolean
+ getTACount(): int
+ getDateTime(): LocalDateTime
+ setPaidProctoringID(int newID): void
+ setDepartment(Department newDept): void
+ setProctoring(ClassProctoring newProctoring): void
+ setIsAvailable(boolean newIsAvailable): void
+ setTACount(int newCount): void
+ setDateTime(LocalDateTime newDateTime): void
TaskType
- taskTypeID: int
- course: Course
- type: String
+ getTaskTypeID(): int
+ getCourse(): Course
+ getType(): String
+ setTaskTypeID(int newID): void
+ setCourse(Course newCourse): void
+ setType(String newType): void
OfferedCourse
- offeredCourseID: int
- course: Course
- sectionNo: int
- semester: Semester
- students: List<Student>
- schedule: List<TimeInterval>
- instructors: List<Instructor>
+ getOfferedCourseID(): int
+ getCourse(): Course
+ getSectionNo(): int
+ getSemester(): Semester
+ getStudents(): List<Student>
+ getSchedule(): List<TimeInterval>
+ getInstructors(): List<Instructor>
+ setOfferedCourseID(int newID): void
+ setCourse(Course newCourse): void
+ setSectionNo(int newSection): void
+ setSemester(Semester newSemester): void
+ setStudents(List<Student> newList): void
+ setSchedule(List<TimeInterval> newSchedule): void
+ setInstructors(List<Instructor> newList): void
Faculty
- facultyId: int
- facultyName: String
- deansOffice: DeansOffice
+ getFacultyId(): int
+ getFacultyName(): String
+ getDeansOffice(): DeansOffice
+ setFacultyId(): void
+ setFacultyName(): void
+ setDeansOffice(DeansOffice newDean): void
Course
- courseID: int
- department: Department
- courseCode: int
- courseName: String
- coordinator: Instructor
+ getCourseID(): int
+ getDepartment(): Department
+ getCourseCode(): int
+ getFullCourseCode(): String
+ getCourseName(): String
+ getCoordinator(): Instructor
+ setCourseID(int newID): void
+ setDepartment(Department newDept): void
+ setCourseCode(int newCode): void
+ setCourseName(String newName): void
+ setCoordinator(Instructor newCord): void
<<abstract>>
User
- bilkentID: int
- name: String
- surname: String
- mail: String
- phoneNumber: String
- isActive: boolean
- department: Department
+ getBilkentID(): int
+ getFullName(): String
+ getName(): String
+ getSurname(): String
+ getContactInfo(): String
+ getMail(): String
+ getPhoneNumber(): String
+ isActive(): boolean
+ getDepartment(): Department
+ setBilkentID(int newID): void
+ setName(String newname): void
+ setSurname(String newSurname): void
+ setEmail(newEmail: String): void
+ setPhonenumber(newPhoneNumber: String): void
+ setIsActive(boolean isActive): void
+ setDepartment(Department newDepartment): void
Department
- departmentID: int
- deptName: String
- deptCode: String
- faculty: Faculty
- instructorList: List<Instructor>
+ getDeptID(): int
+ getDeptName(): String
+ getDeptCode(): String
+ getDaculty(): Faculty
+ setDeptID(int newID): void
+ setDeptName(String newName): void
+ setDeptCode(String newCode): void
+ setFaculty(Faculty newFaculty): void
ClassProctoring
- classProctoringID: int
- course: Course
- taskType: TaskType
- classrooms: String
- creatorInstructor: Instructor
- isicNo: int
- startDate: LocalDateTime
- endDate: LocalDateTime
- TACount: int
- TAsOfProctoring: List<TA>
- isComplete: boolean
+ getClassProctoringID(): int
+ getCourse(): Course
+ getTaskType(): TaskType
+ getClassrooms(): String
+ getCreatorInstructor(): Instructor
+ getIsicNo(): int
+ getStartDate(): LocalDateTime
+ getEndDate(): LocalDateTime
+ getTACount(): int
+ getTAsOfProctoring(): List<TA>
+ isComplete(): boolean
+ setCourse(Course newCourse): void
+ setTaskType(TaskType newTaskType): void
+ setClassrooms(String newClassrooms): void
+ setCreatorInstructor(Instructor newInstructor): void
+ setIsicNo(int newIsicNo): void
+ setStartDate(LocalDateTime newStartDate): void
+ setEndDate(LocalDateTime newEndDate): void
+ setTACount(int newCount): void
+ setTAsOfProctoring(List<TA> newTAs): void
+ setComplete(boolean complete): Boolean
ClassProctoringServiceImpl
manualAssignment(): ResponseEntity<Boolean>
autoAssignment(): ResponseEntity<Boolean>
Semester
- semesterID: int
- year: int
- term: int
+ getSemesterID(): int
+ getYear(): int
+ getTerm(): int
+ setSemesterID(int newSemester): void
+ setYear(int newYear): void
+ setTerm(int newTerm): void
GeneralVariable
- generalVariableSetID: int
- semester: Semester
- TAProctoringCapTime: double
+ getGeneralVariableSetID(): int
+ getSemester(): Semester + getTAProctoringCapTime(): double
+ setSemester(Semester newSemester): void
+ setTAProctoringCapTime(double newTAProctoringCapTime): void
Student
- bilkentID: int
- name: String
- surname: String
- mail: String
- phoneNumber: String
- isActive: boolean
- department: Department
- class: int
- courses: List<OfferedCourse>
+ getBilkentID(): int
+ getFullName(): String
+ getName(): String
+ getSurname(): String
+ getContactInfo(): String
+ getMail(): String
+ getPhoneNumber(): String
+ isActive(): boolean
+ getDepartment(): Department
+ getClass(): int
+ getCourses(): List<OfferedCourse>
+ setBilkentID(int newID): void
+ setName(String newname): void
+ setSurname(String newSurname): void
+ setEmail(newEmail: String): void
+ setPhoneNumber(newPhoneNumber: String): void
+ setIsActive(boolean isActive): void
+ setDepartment(Department newDepartment): void
+ setClass(int newClass): void
+ setCourses(List<OfferedCourse> newList): void
Instructor
- TABlackList: List<TA>
- courses: List<OfferedCourse>
- workloadRequests: List<WorkloadRequest>
- TARequestsFromDeans: List<TAFromDeanRequest>
- swapRequests: List<SwapRequest>
- proctoringRequests: List<InstructorTAProctoringRequest>
+ getTABlacklist(): List<TA>
+ getCourses(): List<OfferedCourse>
+ getWorkloadRequests(): List<WorkloadRequests>
+ getTARequestsFromDeans(): List<TAFromDeanRequest>
+ getSwapRequests(): List<SwapRequest>
+ getProctoringRequests(): List<InstructorTAProctoringRequest>
+ setTABlacklist(List<TA> newTAs): void
+ setCourses(List<OfferedCourse> newCourses): void
+ getWorkloadRequests(List<WorkloadRequest> newRequests): void
+ setTARequestsFromDeans(List<TAFromDeanRequest> newRequests): void
+ setSwapRequests(List<SwapRequest> newRequests): void
+ setProctoringRequests(List<InstructorTAProctoringRequest> newRequests): void
TA
- class: int
- assignedCourse: Course
- schedule: List<TimeInterval>
- proctorings: List<ClassProctoring>
- workloadRequests: List<WorkloadRequest>
- availabilityRequests: List<TAAvailabilityRequest>
- paidProctoringRequests: List<PaidProctoringRequest>
- swapRequestFromOtherTA: List<SwapRequest>
- swapRequestSent: List<SwapRequest>
- proctoringRequests: List<InstructorTAProctoringRequest>
+ getClass(): int
+ getAssignedCourse(): Course
+ getSchedule(): List<TimeInterval>
+ getProctorings: List<ClassProctoring>
+ getWorkloadRequests: List<WorkloadRequest>
+ getAvailabilityRequests(): List<TAAvailabilityRequest>
+ getPaidProctoringRequests(): List<PaidProctoringRequest>
+ getSwapRequestFromOtherTA(): List<SwapRequest>
+ getSwapRequestSent(): List<SwapRequest>
+ getProctoringRequests(): List<InstructorTAProctoringRequest>
+ setClass(int newClass): void
+ setAssignedCourse(Course newCourse): void
+ setSchedule(List<TimeInterval> newSchedule): void
+ setProctorings(List<ClassProctoring> newProctorings): void
+ setWorkloadRequests(List<WorkloadRequest> newRequests): void
+ setAvailabilityRequests(List<TAAvailabilityRequest> newRequests): void
+ setPaidProctoringRequests(List<PaidProctoringRequest> newRequests): void
+ setSwapRequestFromOtherTA(List<SwapRequest> newRequests): void
+ setSwapRequestSent(List<SwapRequest> newRequests): void
+ setProctoringRequests(List<InstructorTAProctoringRequest> newRequests): void
TimeInterval
- intervalID: int
- day: String
- startTime: LocalTime
- endTime: LocalTime
- TAOnCurrBlock: List<TA>
- coursesOnBlock: List<OfferedCourse>
+ getIntervalID(): int
+ getDay(): String
+ getStartTime(): LocalTime
+ getEndTime(): LocalTime
+ getTAsOnCurrBlock: List<TA>
+ getCoursesOnBlock(): List<OfferedCourse>
+ setIntervalID(int newID): void
+ setDay(String newDay): void
+ setStartTime(LocalTime newStartTime): void
+ setEndTime(LocalTime newEndTime): void
+ setTAsOnCurrBlock(List<TA> newTAs): void
+ setCoursesOnBlock(List<OfferedCourse> newList): void
DeansOffice
- faculty: Faculty
- TARequestsFromInstructors: List<TAFromDeanRequest>
- recieveTAFromOtherFacultyRequest: List<TAFromOtherFacultyRequest>
- sentTAFromOtherFacultyRequest: List<TAFromOtherFacultyRequest>
+ getFaculty(): Faculty
+ getTARequestsFromInstructors: List<TAFromDeanRequest>
+ getRecievedTAFromOtherFacultyRequest(): List<TAFromOtherFacultyRequest>
+ getSentTAFromOtherFacultyRequest(): List<TAFromOtherFacultyRequest>
+ setFaculty(Faculty newFac): void
+ setTARequestsFromInstructors(List<TAFromDeanRequest> newRequests): void
+ setRecievedTAFromOtherFacultyRequest(List<TAFromOtherFacultyRequest>): void
+ setSentTAFromOtherFacultyRequest(List<TAFromOtherFacultyRequest>): void
DeansOfficeServiceImpl
+ approveTAFromDeanRequest(int TAFromDeanRequestID): ResponseEntity<Boolean>
+ rejectTAFromDeanRequest(int TAFromDeanRequestID): ResponseEntity<Boolean>
+ approveTAFromOtherFacultyRequest(int TAFromOtherFacultyRequestID): ResponseEntity<Boolean>
+ rejectTAFromOtherFacultyRequest(int TAFromOtherFacultyRequestID): ResponseEntity<Boolean>
+ requestTAFromOtherFaculty(int senderFacultyID, String description, int requestedTACount): ResponseEntity<Boolean>
+ offerPaidProctoring(int departmentID, int proctoringID, int TACountRequested): ResponseEntity<Boolean>
+ dismissTAFromAssignment (int proctoringID, int TAIDtoBeRemoved): ResponseEntity<Boolean>
DepartmentSecretary
- availabilityRequests: List<TAAvailabilityRequest>
- paidProctoringRequests: List<PaidProctoringRequest>
+ getAvailabilityRequests(): List<TAAvailabilityRequest>
+ getPaidProctoringRequests(): List<PaidProctoringRequest>
+ setAvailabilityRequests(List<TAAvailabilityRequest> newRequests): void
+ setPaidProctoringRequests(List<PaidProctoringRequest> newRequests): void
uses
DepartmentSecretaryServiceImpl
+ approvePaidProctoringRequest(int paidProctoringRequest): ResponseEntity<Boolean>
+ rejectPaidProctoringRequest(int paidProctoringRequest): ResponseEntity<Boolean>
InstructorServiceImpl
+ approveWorkloadRequest(int requestID): ResponseEntity<Boolean>
+ rejectWorkloadRequest(int requestID): ResponseEntity<Boolean>
+ approveSwapRequest(int requestID): responseEntity<Boolean>
+ rejectSwapRequest(int requestID): ResponseEntity<Boolean>
+ approveAvailabilityRequest(int requestID): ResponseEntity<Boolean>
+ rejectAvailabilityRequest(int requestID): ResponseEntity<Boolean>
+ sendTAFromDeanRequest(ClassProctoring proctoring, int TACount): ResponseEntity<Boolean>
+ withdrawTAFromDeanRequest(int requestID): ResponseEntity<Boolean>
+ newClassProctoring(int classProctoringID): ResponseEntity<ClassProctoring>
+ createTaskType(Course course, String type): ResponseEntity<Boolean>
+ viewTASchedule(int TAID): List<ResponseEntity<TimeInterval>>
+ viewTASchedule(int TAID): List<ResponseEntity<TimeInterval>>
+ addTAToBlackList(int TAID): ResponseEntity<Boolean>
+ removeTAFromBlackList(int TAID): ResponseEntity<Boolean>
+ createExam(Course course, TaskType taskType, String classrooms, LocalDateTime startDate,LocalDateTime endDate, int TACount): ResponseEntity<Boolean>
+ createRecitation(Course course, TaskType taskType, String classrooms, int isicNo, LocalDateTime startDate,LocalDateTime endDate,int TACount): ResponseEntity<Boolean>
+ manualAssignment(List<TA>): ResponseEntity<Integer>
+ automaticAssignment(int TACount): ResponseEntity<Integer>
+ removeTAFromClassProctoring(int proctoringID, int TAID): ResponseEntity<Boolean>
+ isicSwappingOfCourse(int courseID): ResponseEntity<Boolean>
TAServiceImpl
+ isicISwaps(): ResponseEntity<Boolean>
+ unisicSwaps(): ResponseEntity<Boolean>
+ applyPaidProctoring(int proctoringID): ResponseEntity<Boolean>
+ approveSwapRequest(int swapRequestID): ResponseEntity<Boolean>
+ rejectSwapRequest(int swapRequestID): ResponseEntity<Boolean>
+ approveProctoringRequest(int proctoringRequestID): ResponseEntity<Boolean>
+ rejectProctoringRequest(int proctoringRequestID): ResponseEntity<Boolean>
+ sendSwapRequest(int otherTAID, String description, int proctoringID): ResponseEntity<Boolean>
+ withdrawSwapRequest(int swapRequestID): ResponseEntity<Boolean>
+ sendWorkloadRequest(int typeID, double timeSpent, int courseID, String description): ResponseEntity<Boolean>
+ withdrawWorkloadRequest(int workloadRequestID): ResponseEntity<Boolean>
+ sendTAAvailabilityRequest(DateTime startDate, DateTime endDate, Boolean isUrgent, String description): ResponseEntity<Boolean>
+ printClassRoomDetails(long proctoringID): ResponseEntity<ClassProctoring>
+ viewSchedule(): List<ResponseEntity<TimeInterval>>
+ viewClassProctoring(int classProctoringID): ResponseEntity<ClassProctoring>
<<abstract>>
Request<UserType1, UserType2>
- requestID: int
- senderUser: UserType1
- receiverUser: UserType2
- sentDate: DateTime
- isApproved: boolean
- description: String
+ getRequestID(): int
+ getSenderUser(): UserType1
+ getRecieverUser(): UserType2
+ getSentDate(): DateTime
+ isApproved(): boolean
+ getDescription(): String
+ setRequestID(int newID): void
+ setSenderUser(UserType1 newSender): void
+ setRecieverUser(UserType2 newReciever): void
+ setSentDate(DateTime newDate): void
+ setApproved(boolean newStatus): void
+ setDescription(String newDescription): void
TAFromDeanRequest <Instructor, DeansOffice>
- TACount: int
- isComplete: boolean
- proctoring: ClassProctoring
+ getTACount(): int
+ isComplete(): boolean
+ getProctoring(): ClassProctoring
+ setTACount(int newCount): void
+ setComplete(boolean newIsComplete): void
+ setProctoring(ClassProctoring newProctoring): void
PaidProctoringRequest <TA, DepartmentSecretary>
- paidProctoring:[] OpenedPaidProctoring
+ getPaidProctoring(): OpenedPaidProctoring
+ setApplicantTA(TA newTA): void
InstructorTAProctoringRequest<Instructor, TA>
-classProctoring: ClassProctoring
+getClassProctoring(): ClassProctoring
+setClassProctoring(ClassProctoring newClassProc): void
WorkloadRequest <TA, Instructor>
- type: TaskType
- timeSpent: double
- course: Course
+ getType(): TaskType
+ getTimeSpent(): double
+ getCourse(): Course
+ setType(TaskType newType): void
+ setTimeSpent(double newTimeSpent): void
+ setCourse(Course newCourse): void
SwapRequest<TA, TA>
- classProctoring: ClassProctoring
- isApprovedByTA: boolean
+ getClassProctoring(): ClassProctoring
+ isApprovedByTA(): boolean
+ setClassProctoring(ClassProctoring newClassProctoring): void
+ setApprovedByTA(boolean newIsApprovedByTA): void
TAFromOtherFacultyRequest<DeansOffice, DeansOffice>
- requestedTACount: int
- isComplete: boolean
+ getRequestedTACount(): int
+ isComplete(): boolean
+ setRequestedTACount(int newRequestedTACount): void
+ setComplete(boolean newIsComplete): void
TAAvailabilityRequest<TA, DepartmentSecretary>
- unavailabilityStartDate: DateTime
- unavailabilityEndDate: DateTime
- isUrgent: boolean
+ getUnavailabilityStartDate(): DateTime
+ getUnavailabilityEndDate(): DateTime
+ isUrgent(): boolean
+ setUnavailabilityStartDate(DateTime newStartDate): void
+ setUnavailabilityEndDate(DateTime newEndDate): void
+ setUrgent(boolean newIsUrgent): void
Important Note
Some ResponseEntities are being returned with Boolean, which is planned to indicate whether the wanted request is completed or failed
uses

B) Software Design Patterns Used



Data Transfer Object (DTO) Design Pattern

The Data Transfer Object (DTO) design pattern is a straightforward and highly effective architectural strategy that significantly enhances the structure and maintainability of software applications. By acting as an intermediary data carrier between layers, DTOs help enforce clear separation of concepts. This pattern is especially valuable in applications that demand strong decoupling between layers, optimized performance through minimized data exchange, robust input validation, and enhanced security by exposing only necessary information. Its simplicity, combined with its ability to support scalable and clean application architecture, makes it an essential tool in modern software development.

Purpose and Application in Our Project

DTOs play a critical role in managing the structured and secure exchange of data involving complex workflows between TAs, Instructors, Deans Office and Department Secretary. These workflows include processes like availability updates, proctoring management, workload assignments and more in terms of managing TAs.

Using DTOs reduce complexity since it allows us to simplify big chunks of data into smaller, easier to work DTOs. This works in favor for both backend and front end. It makes the data and the inputs easier to manage and maintain while fetching or posting.

Furthermore, it increases performance by sending only the required data.

DTOs enable us to:
- Encapsulate and validate input/output data clearly and consistently.
- Shield internal data models from overexposure
- Define explicit API contracts to ensure well-documented and predictable interfaces.
- Support scalability and flexibility without impacting core business logic.
- Promote modularity by minimizing tight coupling between architectural layers.

Benefits of the DTO Design Pattern:
1. Security – Protects sensitive internal data from exposure.
2. Validation – Supports pre-processing and filtering of input data.
3. Flexibility – Enables different representations of the same domain model
4. Efficiency – Ensures lightweight data transmission by carrying only essential fields.
5. Decoupling – Enhances maintainability and scalability by isolating domain logic from user interfaces.

DTOs are indispensable in our system, which requires high data integrity, robust API design, and efficient communication across diverse roles and services. They contribute to the system's clarity, testability, and long-term scalability and maintenance. It is used to encapsulate data and transfer it efficiently between the different layers of the application, such as the controller, service, and repository layers. DTOs are simple, serializable objects that carry data without any business logic, making them ideal for decoupling internal representations from external interfaces.

In the context of the Bilkent TA & Proctor Management System, we manage a variety of data interactions involving different user roles (TAs, Instructors, Admin, Department Secretary, Deans Office) and operational data (e.g., Availability, Courses, Proctor Assignments). That is because in our project we have lots of more of DTOs other than the ones showed in the example.

Singleton Design Pattern

The Singleton Pattern is a creational design pattern that ensures a class has only one instance while providing a global point of access to it. It is commonly used in scenarios where a single object must coordinate actions across the system, such as configuration managers, logging utilities, or shared resource controllers. It is commonly known that the general principles of Spring Boot supports that. For example, classes annotated by @Service, @Repository, and @Component in Spring are singletons by default.

Purpose and Application in Our Project

In our backend application the Singleton pattern serves several critical purposes:

Resource Management: Our application appears to follow a layered architecture with Controllers, Services, Repositories, and DTOs. By implementing services and repositories as singletons, we ensure efficient use of resources across the application.

State Coordination: For components that need to maintain application state or coordinate activities, singleton instances prevent inconsistencies that could arise from multiple instances.

Implementation in Our Project:

Our Services directory contains business logic components that are implemented as singletons through Spring's @Service annotation

The Repositories directory contains data access objects that are singleton instances created through @Repository annotation

Components in the Components directory are likely Spring-managed singletons via @Component annotation

The BackendApplication class serves as the main entry point and contains the singleton application context.

Configuration Management: Classes in the Configurations directory such as WebConfiguration, SecurityConfiguration that are annotated with @Configuration, creating singleton beans that define application settings consistently across the system.

<<interface>>
TAService

+ TAProfileDTO getTAProfileById(int id)
+ boolean createNewTA(CreateTADTO dto)
+ List<TAProfileDTO> getAllAvailableTAsByDepartmentCode(String departmentCode, int classProctoringId, int userId)
+ List<TAProfileDTO> getAllAvailableTAsByDepartmentCode(String departmentCode, int classProctoringId, int userId, boolean eligibilityRestriction, boolean oneDayRestriction)
+ List<TAProfileDTO> getAllAvailableTAsByFacultyId(int facultyId, int classProctoringId, int userId)
+ List<TAProfileDTO> getAllAvailableTAsByFacultyId(int facultyId, int classProctoringId, int userId, boolean eligibilityRestriction, boolean oneDayRestriction)
+ List<TAProfileDTO> getAllTAProfiles()
+ boolean doesTakeCourse(int taId, int courseId)
+ boolean isTAEligible(int taId, int courseId)
+ boolean noProctoringInOneDay(int taId, int proctoringId)

<<@Service>>
TAServiceImpl

- TARepo taRepo
- UserRepo userRepo
- LoginRepo loginRepo
- ClassProctoringTARelationRepo classProctoringTARelationRepo
- TAWorkloadRequestService taWorkloadRequestService
- CurrentUserUtil currentUserUtil
- TAProfileMapper taProfileMapper
- LoginMapper loginMapper
- ClassProctoringRepo classProctoringRepo
- TAAvailabilityService taAvailabilityService
- AuthStaffProctoringRequestService authStaffProctoringRequestService
- CourseRepo courseRepo
- StudentRepo studentRepo
- TAAvailabilityService taaAvailabilityService
- DepartmentSecretaryRepo departmentSecretaryRepo

+ TAProfileDTO getTAProfileById(int id)
+ boolean createNewTA(CreateTADTO dto)
+ List<TAProfileDTO> getAllAvailableTAsByDepartmentCode(String departmentCode, int classProctoringId, int userId)
+ List<TAProfileDTO> getAllAvailableTAsByDepartmentCode(String departmentCode, int classProctoringId, int userId, boolean eligibilityRestriction, boolean oneDayRestriction)
+ List<TAProfileDTO> getAllAvailableTAsByFacultyId(int facultyId, int classProctoringId, int userId)
+ List<TAProfileDTO> getAllAvailableTAsByFacultyId(int facultyId, int classProctoringId, int userId, boolean eligibilityRestriction, boolean oneDayRestriction)
+ List<TAProfileDTO> getAllTAProfiles()
+ boolean doesTakeCourse(int taId, int courseId)
+ boolean isTAEligible(int taId, int courseId)
+ boolean noProctoringInOneDay(int taId, int proctoringId)

1 1 1 1

uses * uses * uses * uses *

<<Entity>>
TA

<<Entity>>
ClassProctoring

<<Entity>>
Course

<<Entity>>
Login

Benefits of the Singleton Pattern

The Singleton pattern provides several advantages to our Spring Boot application:

Memory Efficiency: By maintaining only one instance of service classes, repositories, and components, our application conserves memory resources that would otherwise be consumed by duplicate objects.

Consistent State: Singleton objects ensure that all parts of the application access the same instance, maintaining consistent state and preventing data synchronization issues.

Lazy Initialization: Spring's singleton beans are lazily initialized by default, meaning they are only created when needed, further optimizing resource usage.

Thread Safety: Spring-managed singletons are thread-safe by default, handling concurrent access without developers needing to implement complex synchronization logic.

Simplified Testing: Singleton components in Spring can be easily mocked or replaced in testing scenarios using Spring's dependency injection capabilities.

Reduced Coupling: By accessing services and repositories through dependency injection rather than direct instantiation, our code remains loosely coupled and more maintainable.