CS 319 Object-Oriented Software Engineering

Instructor: Eray Tüzün, Teaching Assistant: Yahya Elnouby

AdayBilgi
Deliverable 4 - 1st Iteration

Group 1

Emine Noor

Eray İşçi

Hatice Kübra Çağlar

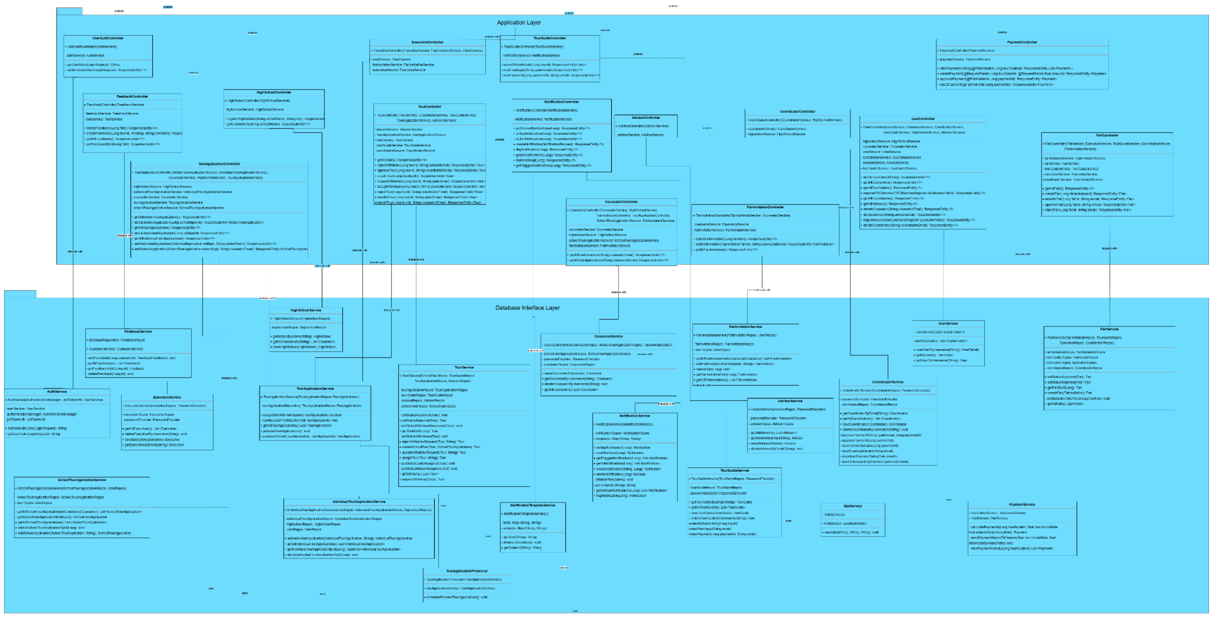
İbrahim Çaycı

İrem Damla Karagöz

Yiğit Özhan

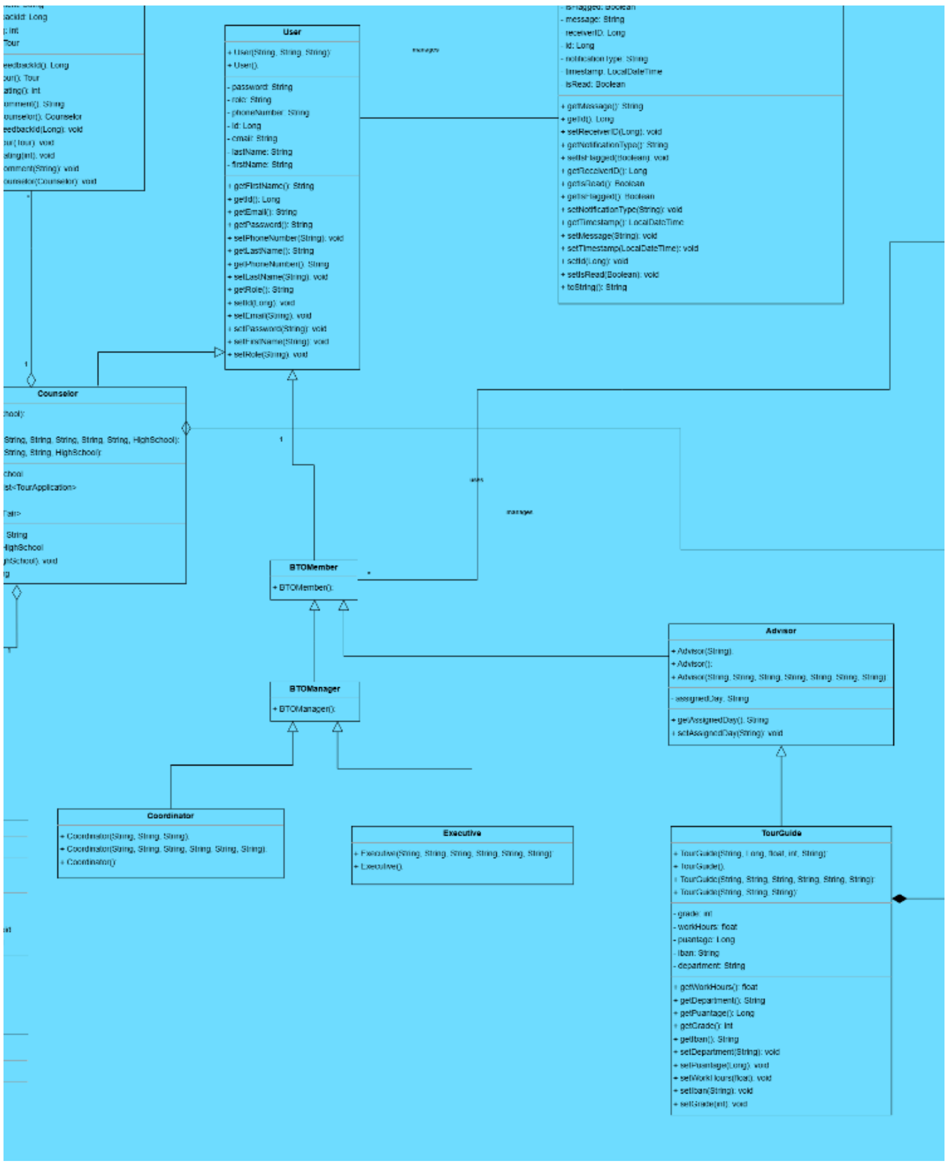
1. Design Patterns

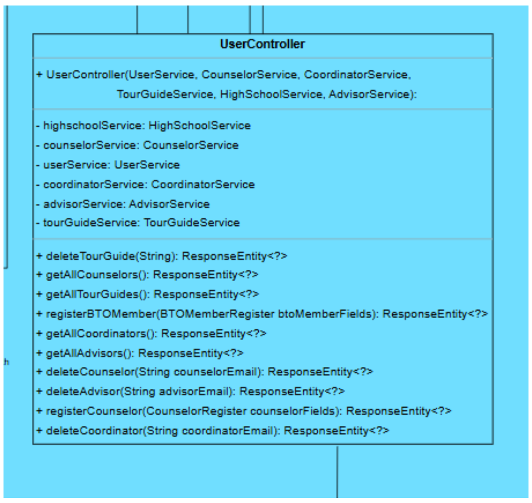
1.1 Facade Pattern



Our system uses the Facade design pattern to make it easier for controllers to communicate with the database and underlying service classes. Service classes (such as UserService, TourService, and FeedbackService) that contain intricate logic and database activities are interacted with by controllers like UserController, TourController, and FeedbackController. The Facade approach offers a consistent and straightforward interface for handling user requests by separating the controllers from the more complex aspects of the service layer. The system's maintainability and scalability are enhanced by this design, which guarantees that any modifications to the database or service layer require only minor alterations to the controllers. As a result, the controllers stay small and only manage HTTP requests and answers, leaving business logic to the service layer and the database crud operations are done in our repository layer.

1.2 Factory Pattern



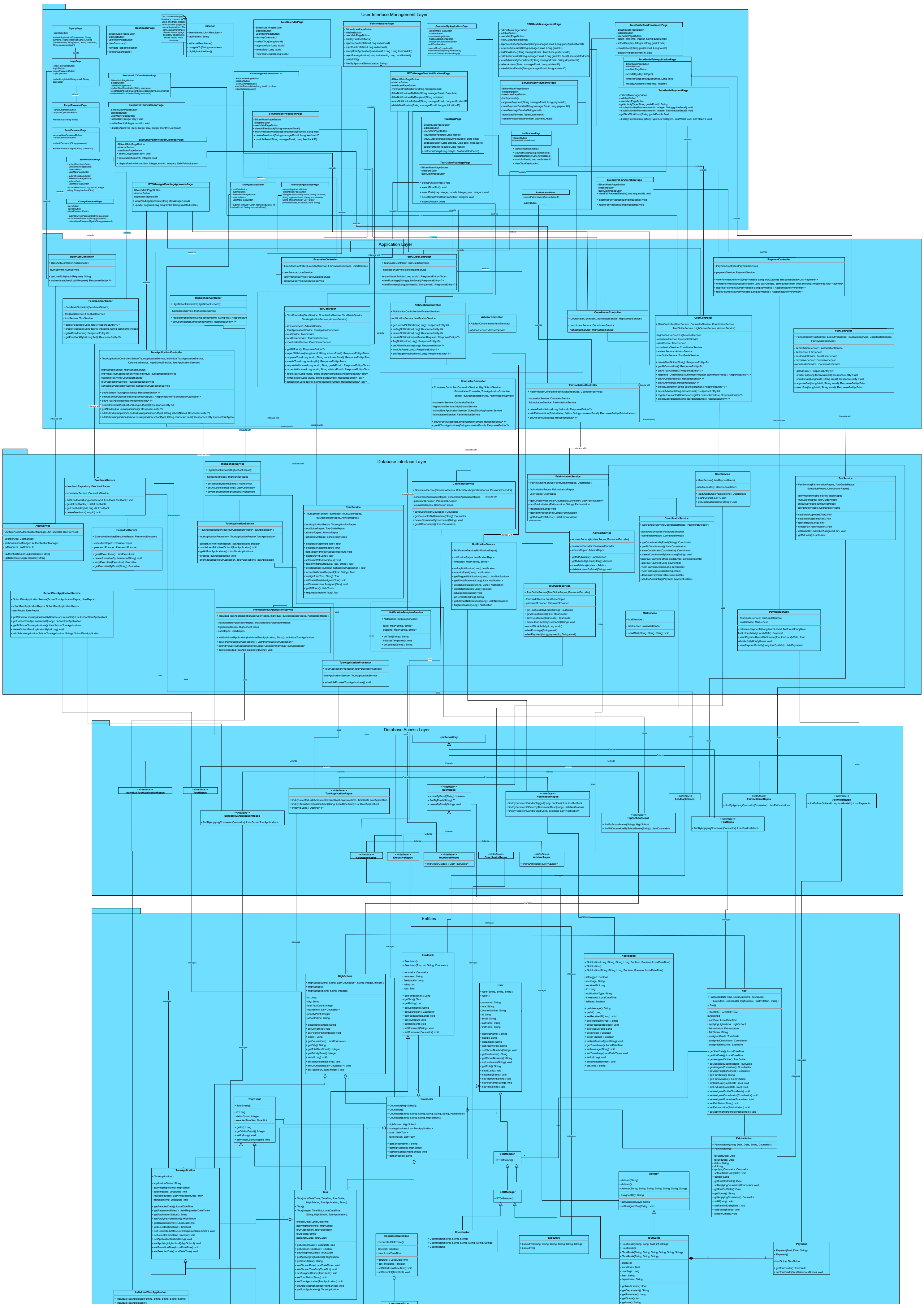

UserController
+ UserController(UserService, CounselorService, CoordinatorService, TourGuideService, HighSchoolService, AdvisorService):
- highschoolService: HighSchoolService
- counselorService: CounselorService
- userService: UserService
- coordinatorService: CoordinatorService
- advisorService: AdvisorService
- tourGuideService: TourGuideService
+ deleteTourGuide(String): ResponseEntity<?>
+ getAllCounselors(): ResponseEntity<?>
+ getAllTourGuides(): ResponseEntity<?>
+ registerBTOMember(BTOMemberRegister btoMemberFields): ResponseEntity<?>
+ getAllCoordinators(): ResponseEntity<?>
+ getAllAdvisors(): ResponseEntity<?>
+ deleteCounselor(String counselorEmail): ResponseEntity<?>
+ deleteAdvisor(String advisorEmail): ResponseEntity<?>
+ registerCounselor(CounselorRegister counselorFields): ResponseEntity<?>
+ deleteCoordinator(String coordinatorEmail): ResponseEntity<?>

The Factory pattern serves as a centralized mechanism for creating objects, encapsulating the instantiation logic in one place. It provides a clear way to create objects dynamically at runtime without requiring the client to know the exact class being instantiated. This design promotes code maintainability, scalability, and flexibility.

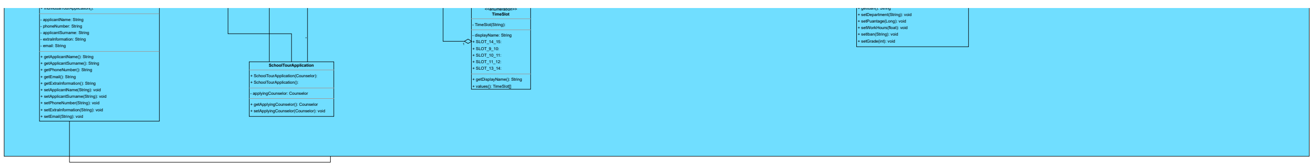
In this system, the UserController employs the Factory pattern to handle the creation of various user roles, such as BTOMember, BTOManager, TourGuide, and Counselor. The UserController interacts with services that utilize a factory to dynamically determine and instantiate the correct user type based on provided input. This approach ensures that object creation logic is encapsulated and isolated from the controller.

For example, the UserController includes methods like registerBTOMember and registerCounselor, which interact with the underlying factory or service to create user instances. If new roles, such as Advisor or Coordinator, are introduced, the factory can be updated without modifying the controller logic.

2. Detailed Design Diagram



User Interface Management Layer
SignUpPage
DashboardPage
Sidebar
TourCalendarPage
FairInvitationsPage
CounselorMyApplicationsPage
BTOGuideManagementPage
TourGuideTourEnrollmentPage
LoginPage
ExecutiveBTOCoordinationPage
BTOManagerFairInvitationsList
BTOManagerSentNotificationsPage
TourGuideFairApplicationPage
ForgotPasswordPage
ExecutiveTourCalendarPage
BTOManagerPaymentsPage
TourGuidePaymentPage
ResetPasswordPage
BTOManagerFeedbackPage
PuantajPage
NotificationsPage
SendFeedbackPage
ExecutiveFairInvitationCalendarPage
TourGuidePuantajePage
ChangePasswordPage
BTOManagerPendingApprovalsPage
TourApplicationForm
IndividualApplicationPage
FairInvitationForm
ExecutiveFairOperationPage
Application Layer
UserAuthController
ExecutiveController
TourGuideController
PaymentController
FeedbackController
HighSchoolController
TourController
NotificationController
AdvisorController
CoordinatorController
UserController
FairController
TourApplicationController
CounselorController
FairInvitationController
Database Interface Layer
HighSchoolService
FeedbackService
CounselorService
FairInvitationService
UserService
FairService
AuthService
ExecutiveService
TourApplicationService
TourService
CoordinatorService
AdvisorService
NotificationService
SchoolTourApplicationService
IndividualTourApplicationService
NotificationTemplateService
TourGuideService
MailService
PaymentService
TourApplicationProcessor
Database Access Layer
JpaRepository
IndividualTourApplicationRepos
TourRepos
TourApplicationRepos
UserRepos
NotificationRepos
FeedbackRepos
FairInvitationRepos
PaymentRepos
SchoolTourApplicationRepos
HighschoolRepos
FairRepos
CounselorRepos
ExecutiveRepos
TourGuideRepos
CoordinatorRepos
AdvisorRepos
Entities
Feedback
Notification
HighSchool
User
Fair
TourEvent
Counselor
BTOMember
FairInvitation
BTOManager
TourApplication
Tour
Advisor
RequestedDateTime
Coordinator
Executive
TourGuide
Payment
IndividualTourApplication



- applicantName: String
- phoneNumber: String
- applicantSurname: String
- extraInformation: String
- email: String
+ getApplicantName(): String
+ getApplicantSurname(): String
+ getPhoneNumber(): String
+ getEmail(): String
+ getExtraInformation(): String
+ setApplicantName(String): void
+ setApplicantSurname(String): void
+ setPhoneNumber(String): void
+ setExtraInformation(String): void
+ setEmail(String): void
SchoolTourApplication
+ SchoolTourApplication(Counselor):
+ SchoolTourApplication():
- applyingCounselor: Counselor
+ getApplyingCounselor(): Counselor
+ setApplyingCounselor(Counselor): void
TimeSlot
- TimeSlot(String):
- displayName: String
+ SLOT_14_15:
+ SLOT_9_10:
+ SLOT_10_11:
+ SLOT_11_12:
+ SLOT_13_14:
+ getDisplayName(): String
+ values(): TimeSlot[]
1
+ setDepartment(String): void
+ setPuantage(Long): void
+ setWorkHours(float): void
+ setIban(String): void
+ setGrade(int): void