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

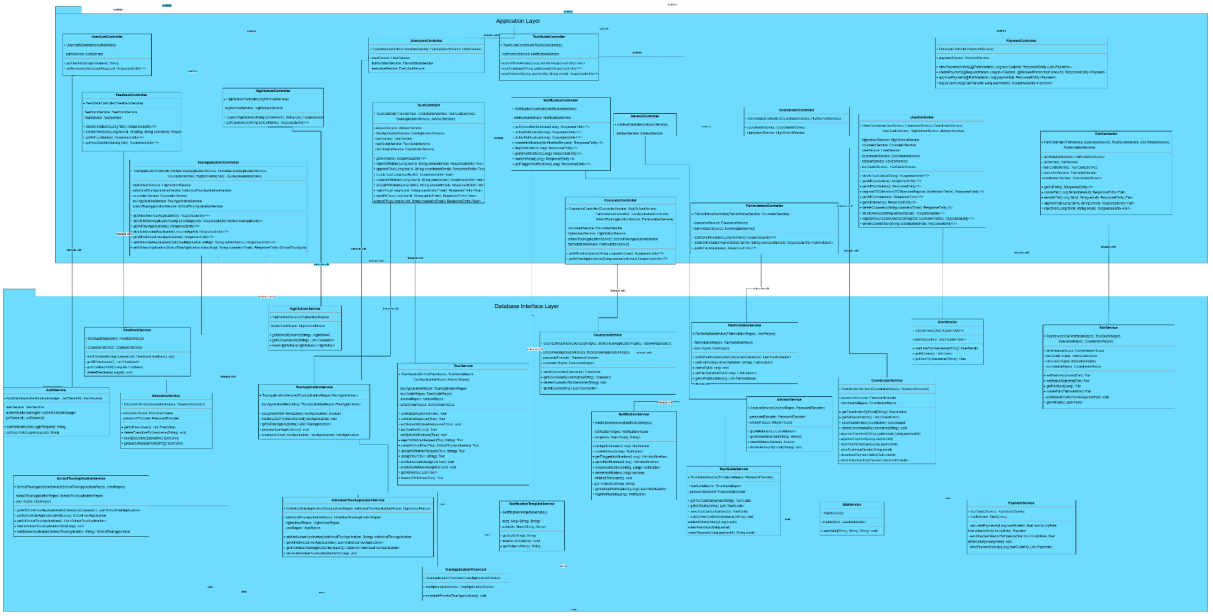
İbrahim Çaycı

İrem Damla Karagöz

Yiğit Özhan

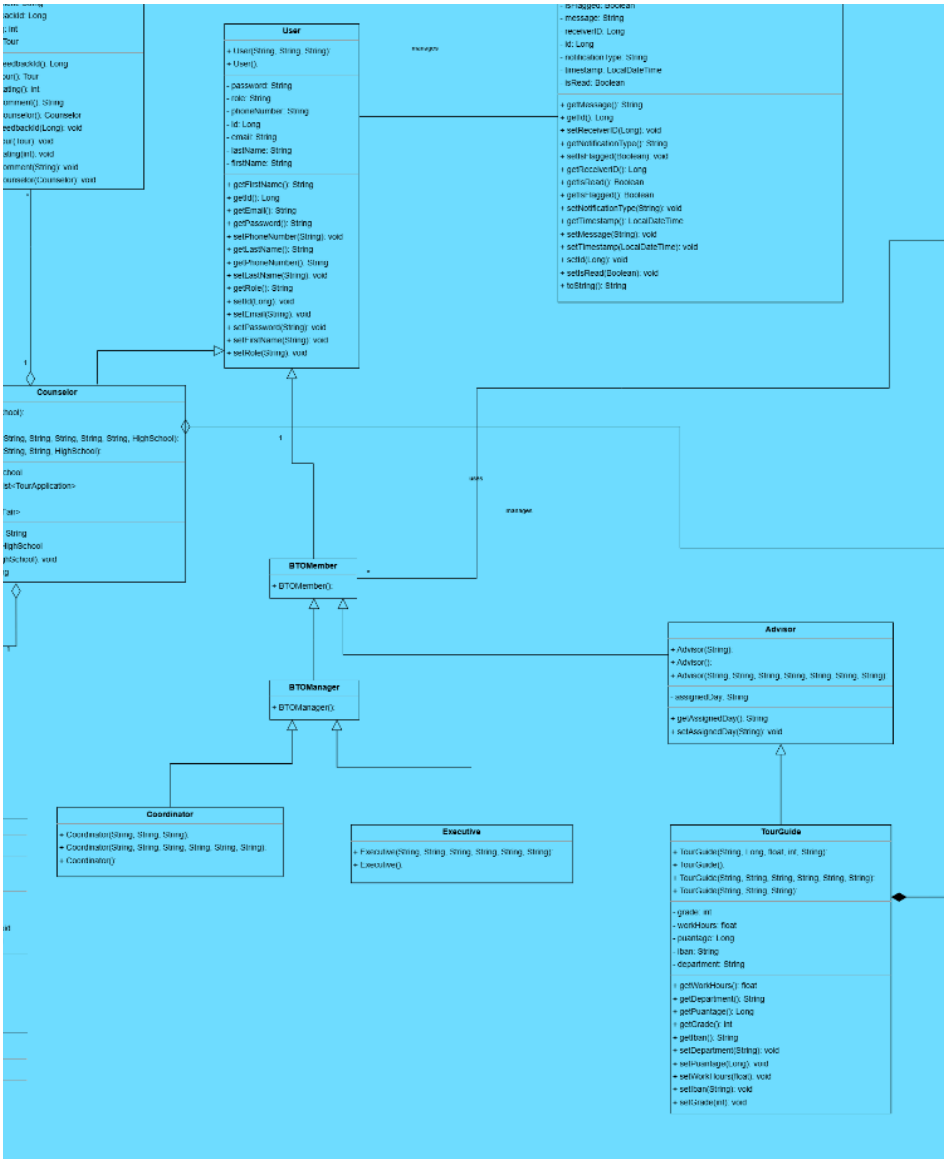
1. Design Patterns

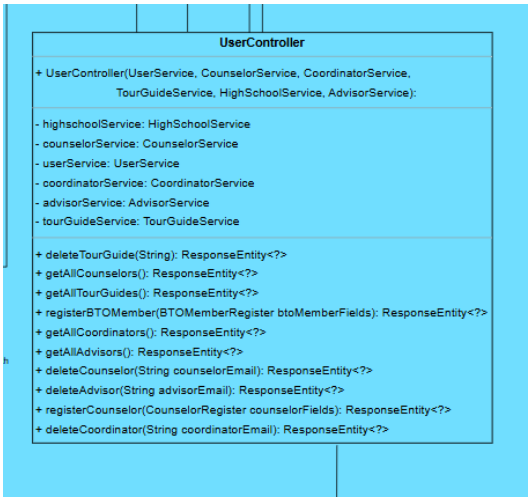
1.1 Facade Pattern



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



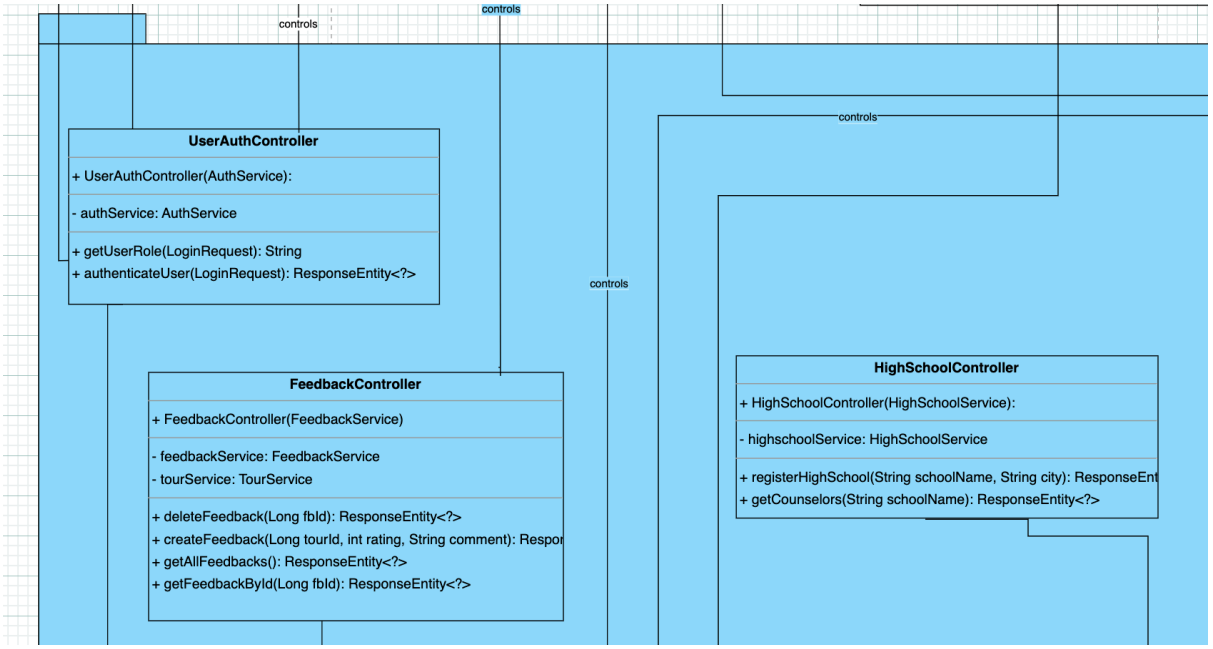


The Factory pattern serves as a centralized mechanism for creating objects, encapsulating the instantiation logic in one place. It provides a clear way to create objects dynamically at runtime without requiring the client to know the exact class being instantiated. This design promotes code maintainability,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

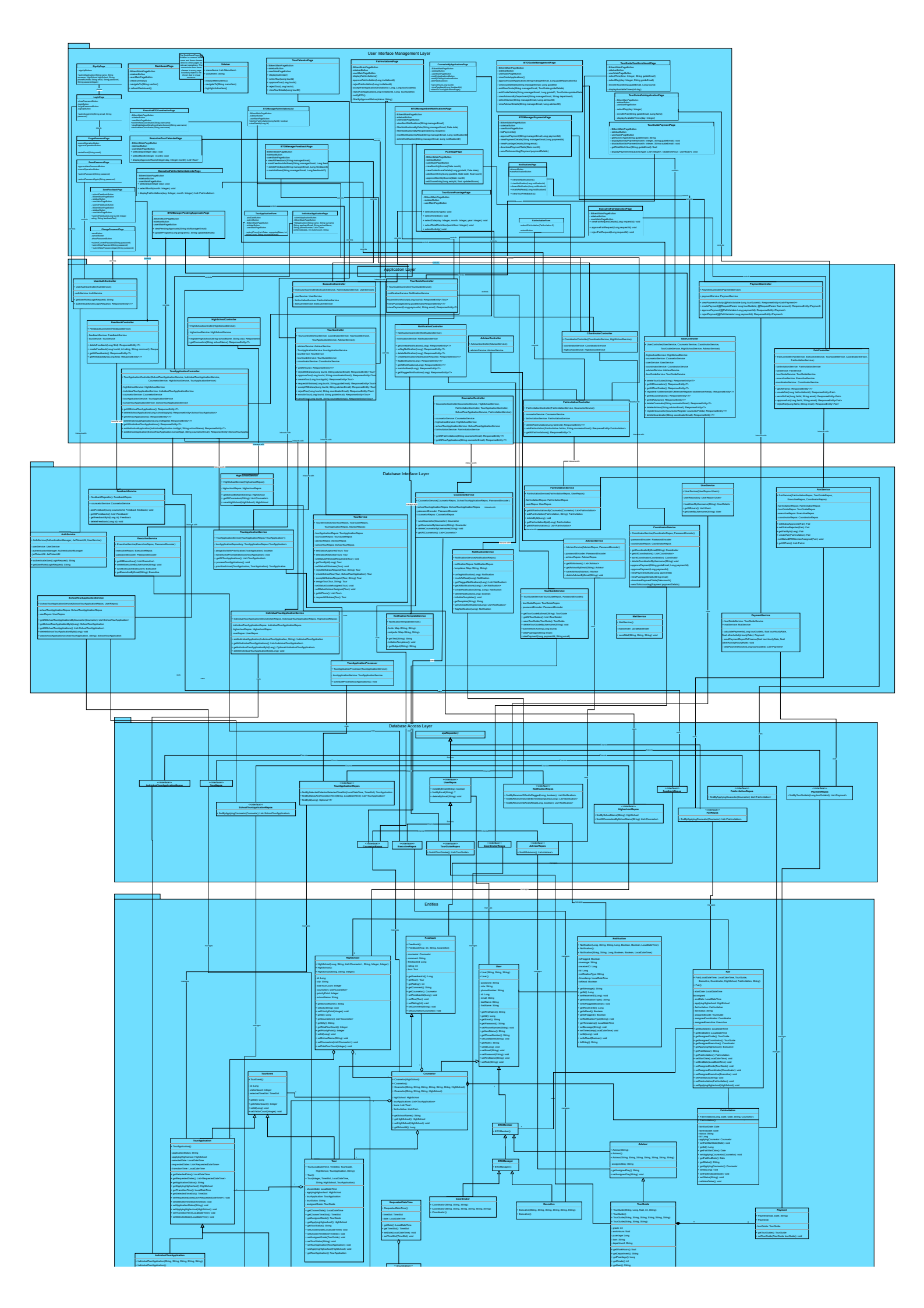
For example, the UserController includes methods like registerBTOMember and registerCounselor, which interact with the underlying factory or service to create user instances. If new roles, such as Advisor or Coordinator, are introduced, the factory can be updated without modifying the controller logic.

1.3 Singleton Pattern



The Singleton Design Pattern ensures that a class has only one instance and provides a global point of access to it. In our web app, we chose to use the singleton pattern for some of our classes, i.e., UserAuthController, FeedbackController, and HighSchoolController. This decision ensures that each of these classes maintains a single instance throughout the application's lifecycle, allowing us to avoid unnecessary object creation and enhance control over resource management. This approach promotes better communication between different components of the application, improves overall performance, and contributes to a robust, scalable, and maintainable architecture. The singleton pattern was instrumental in achieving consistency and efficiency, ensuring a well-organized system capable of handling concurrent requests seamlessly.

2. Detailed Design Diagram


User Interface Management Layer
Application Layer
Database Interface Layer
Database Access Layer
Entities