
CS319
Object-Oriented Software Engineering

**Deliverable 3 & 4 Final Submission**

Section 1

Can Tücer (22203239)
Begüm Filiz Öz (22203470)
Orhun Ege Çelik (22202321)
Ruşen Ali Yılmaz (22203805)
Mehmet Emin Avşar (22202995)
Göktuğ Ozan Demirtaş (22202913)

# 1. Top 2 Design Goals with Possible Trade-offs

## a. Usability

### i. Description

The main purpose of TOYS is to facilitate the jobs of Bilkent Tanıtım Ofisi (BTO) staff and applicants from high schools. Hence, providing high usability is our top design goal. To achieve this, the following design choices will be made:

- A BTO employee should be able to volunteer for a tour with less than 3-page redirections and 5 mouse clicks.
- A high school advisor should be able to register for a tour with less than 5 mouse clicks.
- Procedures required to do basic operations (like tour application/personnel management) should be explained with short documentation.
- Pastel colors that do not exhaust the eye should be used to make components easier to discriminate.
- Page naming should be clear and distinct. Page names presented in the navigation bar and at the top of the page should fully describe the functionality of the page with as few words as possible.

### ii. Trade-offs

- **Usability vs Rapid Development**

One possible tradeoff for usability would be rapid development. Making the system as user-friendly as possible extends the development time. Designing new pages and fixing bugs becomes harder as design choices must be made with thorough consideration.

- **Usability vs Portability**

Another trade-off would be portability. Since the usage of specific user interface elements and design choices is needed to further increase usability, incompatibilities, especially with older browsers and smaller devices, will be introduced. This will make the system less portable in favor of increasing the usability for other devices.

## b. Functionality

### i. Description

TOYS should be the only platform BTO staff and prospective visitors use to meet all of their needs. Thus, it should provide more functionality than only allowing BTO staff to arrange tours. Any extra feature will make the app more desirable for the users as it removes the necessity of using multiple platforms concurrently. This may include data analysis, guide suggestions for advisors, or viewing the reviews of high school tour attendees. Every feature that may be required by any user role should be included within TOYS. Specifically:

- It should allow for tours and fair applications. Individual tour applications ask for major preferences to accommodate the future prospects of the students.
- It should allow guides to see available tours and volunteer to conduct them. In addition, it should inform the guides about the tours they are responsible for.
- It should involve personnel management for the director and coordinator. Guide payments, advisor offers, guide assignments, and trainee applications are all procedures that will be allowed via the platform.
- It should analyze necessary data for the director. For example, the director will be able to compare Bilkent with rival universities and gain more information about the students who eventually choose Bilkent.

### ii. Trade-offs

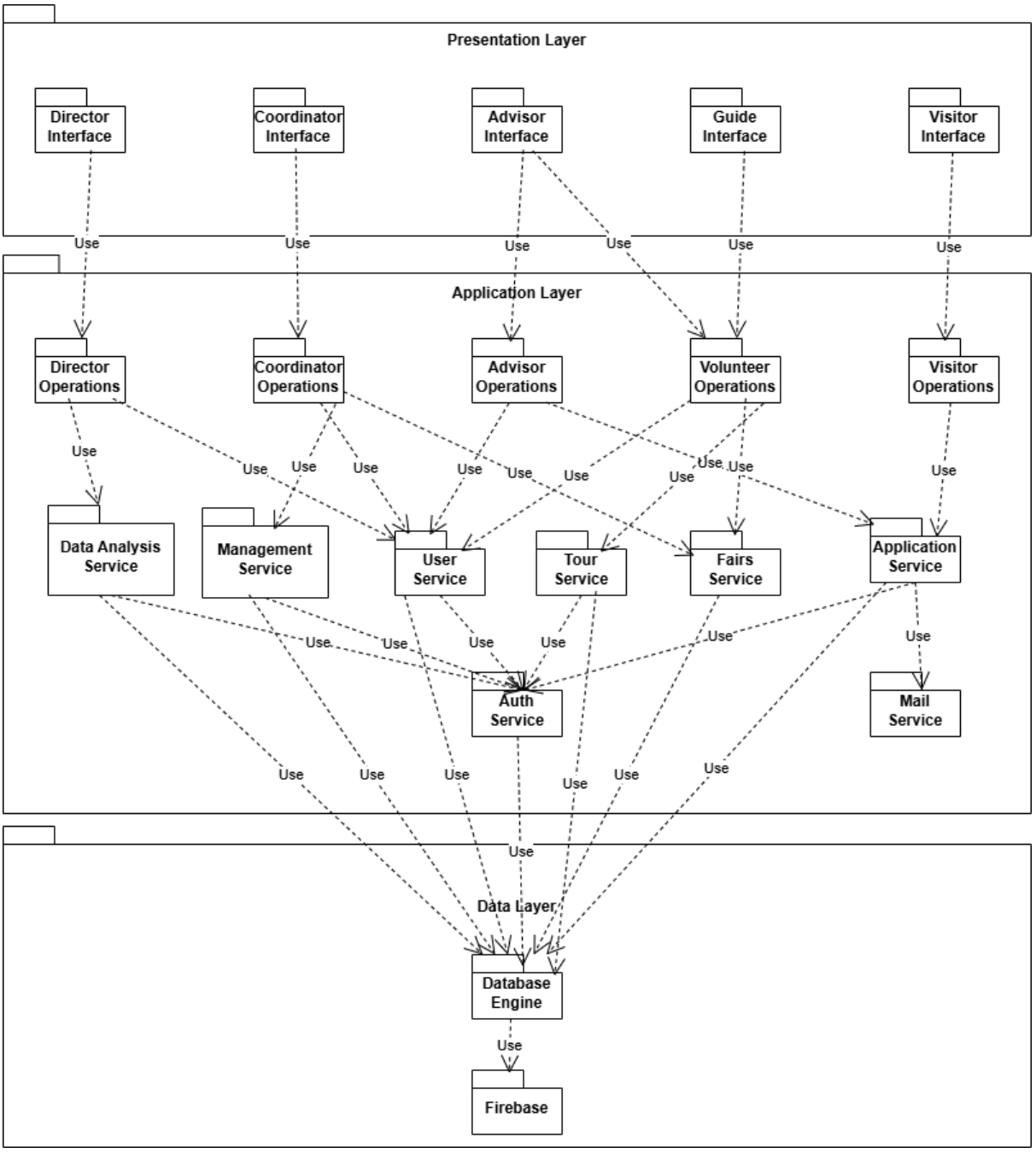
- **Functionality vs Reliability**

As the implementation of more functionality makes the system more complex, there would be a decrease in reliability. Since the system has more functionality, such as analytics and personnel management in addition to tour applications, the scope of the project widens. There will be more classes and more relations between them. Therefore, the code will be more prone to mistakes and potential breaches. These breaches could include security breaches or bugs that would exacerbate user frustration. As a result, the system would be less reliable.

- **Functionality vs Low Cost**

Similarly, the development and hosting of more functionality would require a longer development time together with better technology, increasing both the development and running costs of our project. More advanced technology involves an increased cost, and more storage space would be required as a result of increased functionality.

# 2. Subsystem Decomposition Diagram



# 3. Detailed Class Diagram

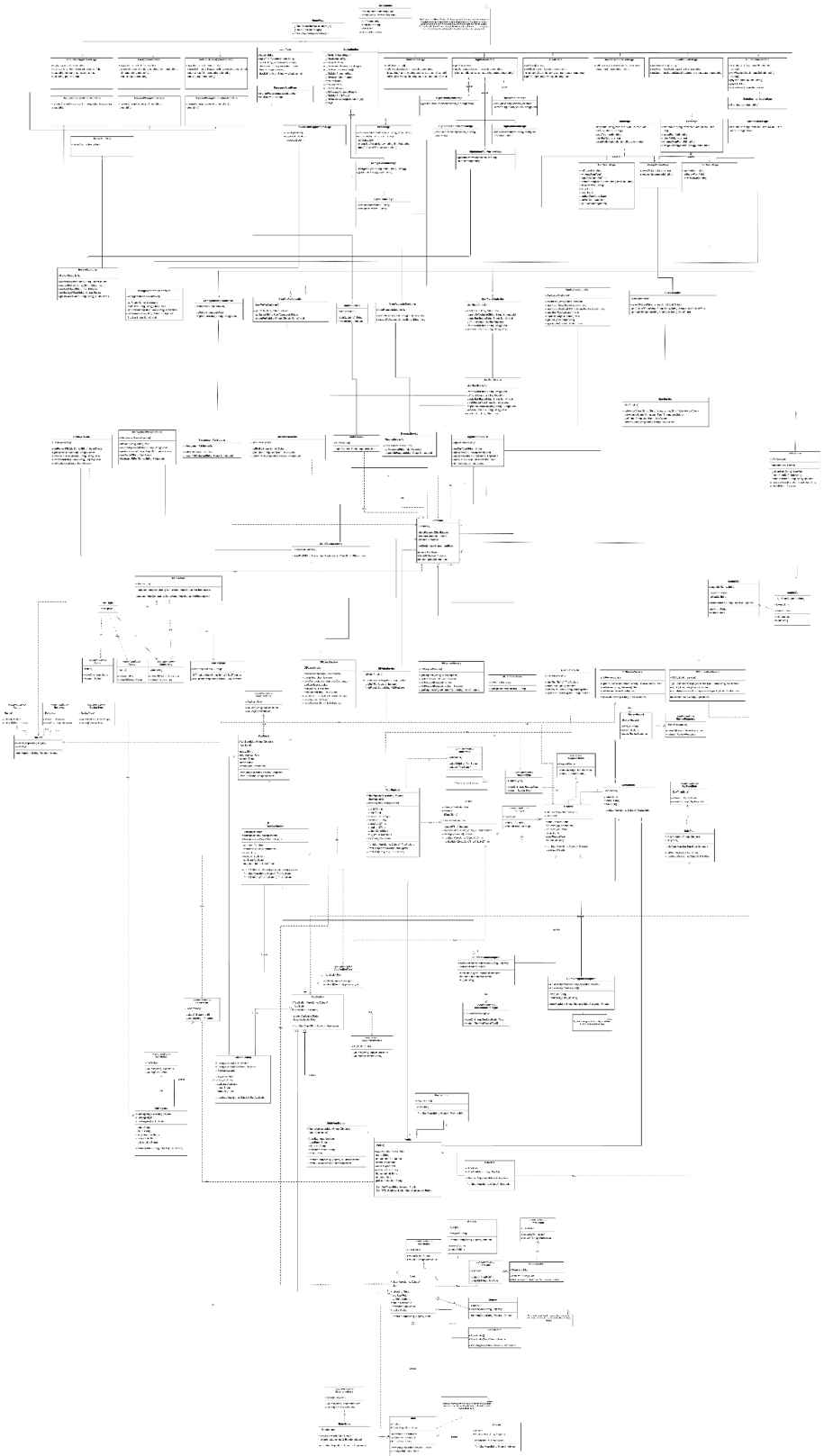


**Fig.1:** Class Diagram

The high-resolution picture is given in the link:
https://drive.google.com/file/d/1piuDCps5TE9owO0wUKLYu2MwIskp5oLJ/view?usp=sharing

# 4. Design Patterns

## a. Facade Pattern

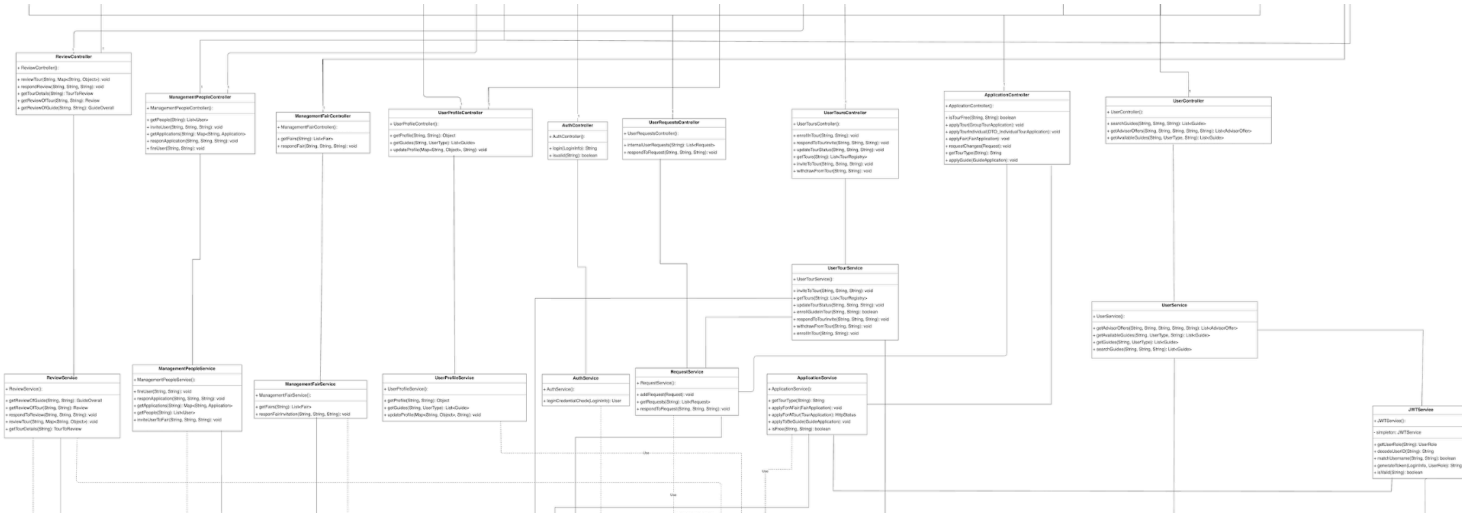


**Fig.2:** Facade Pattern in the Class Diagram

We delegate logic and responsibilities to the backend, instead of doing these in the frontend. This provides a level of security, so that no user has direct database access. This also allows for permission delegation, restricted and censored information access. We achieve this in the following way:

1. Spring Controllers handle network connections, and parameter parsing. They do not contain any business logic on high level response handling. The requests are forwarded to related services.
2. Spring Services handle the high level logic of the request. This includes authentication check, and other request specific logic. Non-special logic is delegated to other services. For authentication, the system uses Javascript Web Tokens or JWTs. There are medium length encoded strings with some information embedded in them. These are handled by JWTService.
3. For database interactions, we use specialized services. These are abstractions on top of the Firebase Firestore service.

This Facade Pattern applies to the entirety of the backend services. Endpoints are context-sensitively grouped into Spring Rest Controllers, which handle the network side of requests, logic is delegated to Spring Services, which can access other services. These abstractions allow for proper unit testing capability, along with easier debugging.

## b. Singleton Pattern

Most pages in our website need access to the current user to display the relevant information. These can be buttons, text or any other elements. To facilitate this, we make use of React Contexts, which in our project is effectively a singleton. We have a context called UserContext in our code base that has the following properties:

1. The UserContext provider wraps every rendered component in a page.

2. At any point in time, there is one UserContext and its contents are the same across all components.
3. Any change in a UserContext is reflected across every use of it.
4. The contents of the current UserContext can be accessed from any component currently on the page, regardless of what level it is in without depending on what was passed from a parent component.

The UserContext holds the information of the currently logged in user, or empty values when a user is not logged in.

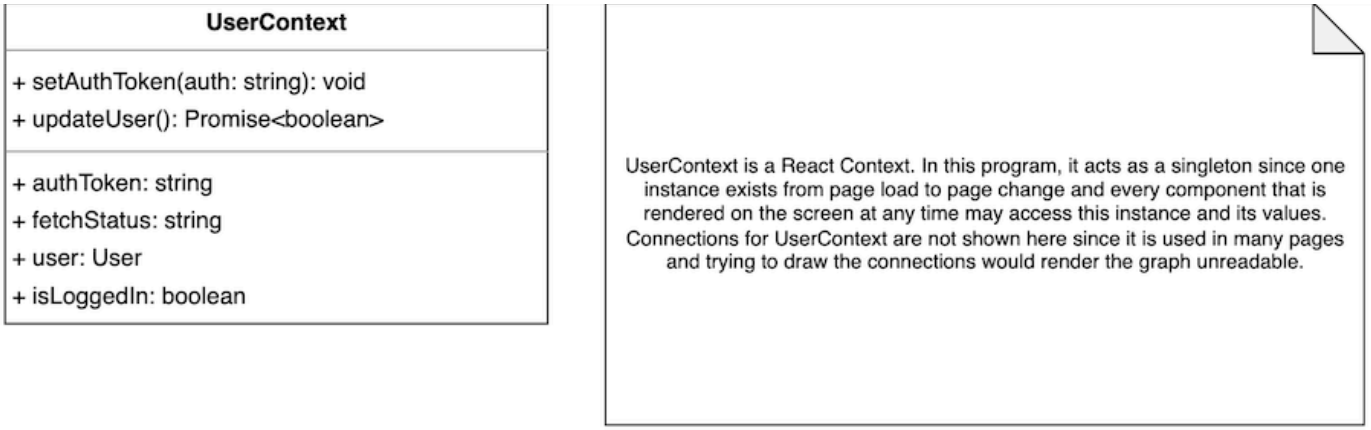



**Fig.3:** Singleton Pattern in the Class Diagram