Bilkent University

Department of Computer Engineering

Campuswide Connection System

OnlyBilkent (team12 - 01)

Design Report

Anıl Altuncu - 21901880
Zehra İyigün - 22002913
İpek Sönmez - 22103939
Bartu Albayrak - 22101640
Gizem Gökçe Işık - 21803541

Instructor: Eray Tüzün
Teaching Assistant(s): Yahya Elnouby, Selen Uysal

Iteration 2

Contents

Design Report

OnlyBilkent

1. Introduction

The purpose of this report is to define the software system by explaining its purpose and modified non-functional requirements of the software system in the first report. The second part explains the software architecture, decomposition of different subsystems inside the system, relation between hardware and software, and data management. Lastly, the last part is reserved for the low-level design, which covers the object design, packages, class interfaces, and design patterns.

1.1. The Purpose of the System

OnlyBilkent is designed to meet the diverse needs of the Bilkent University community under one web application. Exclusively designed for Bilkenters, this single application connects students, club boards, and graduates. OnlyBilkent empowers users to effortlessly buy and sell items, recover lost belongings, donate unused items, and simplify borrowing. Unlike other separate systems, OnlyBilkent stands out by streamlining and enhancing university life with accessibility, dedicated to fostering a more connected and harmonious campus experience for Bilkent University students, club boards, and graduates.

1.2. Design Goals

1.2.1. Usability

Our aim is to ease and fasten communication among the student network of Bilkent University. Therefore, the system should be easy to use and non-complicated. Consistent design elements, such as consistent symbols for post categories throughout every page of the application, will be maintained. A straightforward and simple left-side menu will display post categories, notifications, and announcements. Posting, messaging, and making an announcement will be done.

1.2.2. Maintainability

Given the possibility of future needs, such as incorporating new features, designing our application with maintainability in mind is significant. OnlyBilkent components will be organized into separate modules such as the post module, buyer-seller module, lender-borrower module, announcement module, etc.

1.2.3. Reliability

Users often seek to search for specific posts on OnlyBilkent, whether for educational purposes or to stay updated on campus events. Therefore, minimizing the data loss caused by system crashes is reasonable. Database backups will be performed every 12 hours. Using redundant database systems with replication to ensure data remains available even if one database server fails will minimize the data loss by 90%.

1.2.4. Safety

Authentication system where OnlyBilkent users need to log in with their own university e-mail addresses to ensure that they are from Bilkent University. This means that each user will have only one account associated with their university e-mail address. AES 128 Encryption will be used to encrypt the passwords of the users. After encryption, the password will be stored in the database. Admins will control the access authorization of the users.

2. High-level Software Architecture

2.1. Subsystem Decomposition

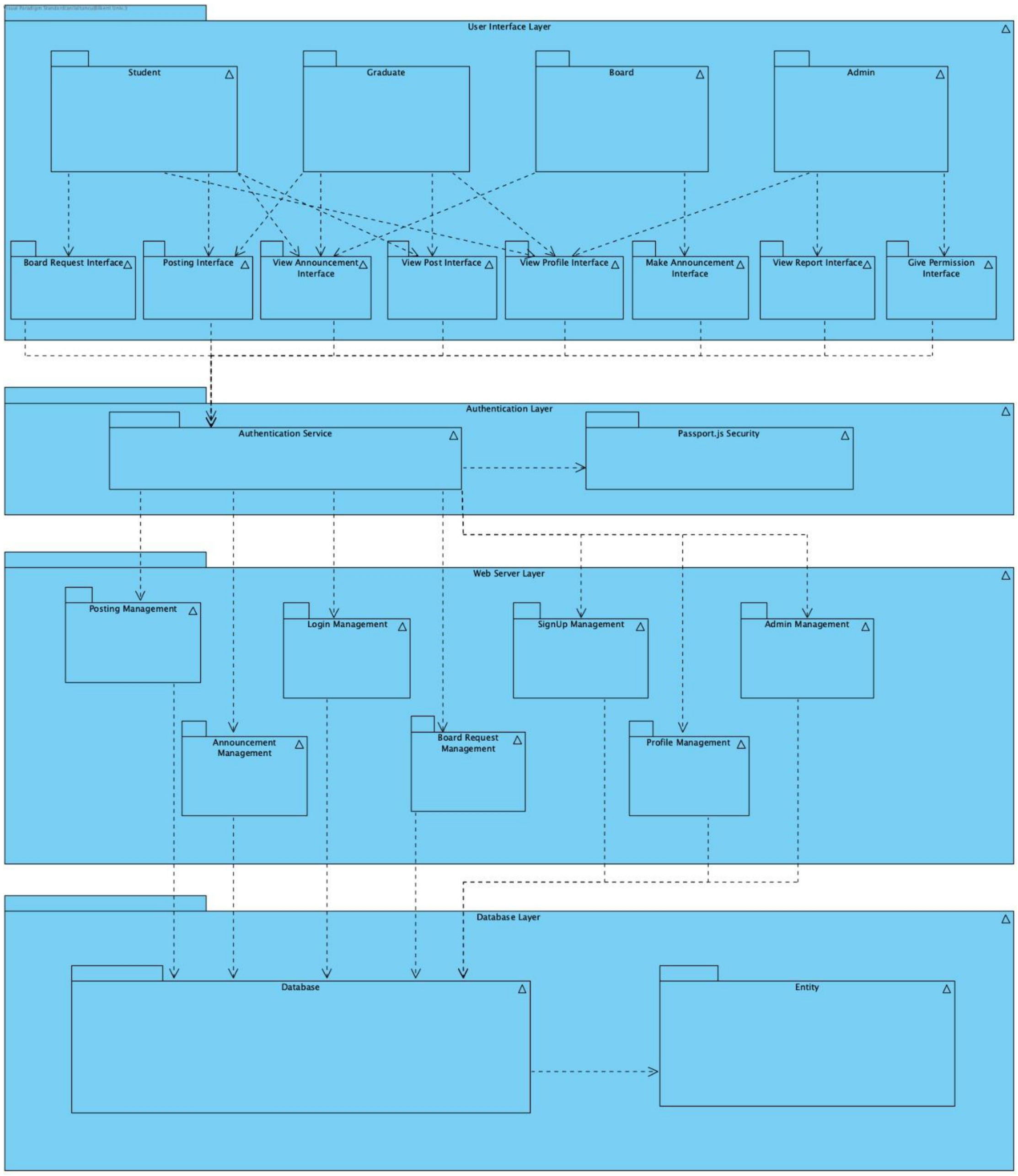


Fig. 1 Subsystem Decomposition Diagram

We utilized a four layered structure in our system, encompassing the User Interface Layer, Authentication Layer, Web Server Layer, and Database Layer. We aimed to decompose into subsystems regarding their distinct functions. Each subsystem holds the responsibility to call other systems ensuring the system remains maintainable while in action. This division also aims to enhance scalability by ensuring each layer operates independently. As a result, it complements our system to be easier to maintain, and functional.

2.1.1 User Interface Layer

The User Interface Layer is the interface between users, and the screens, providing interaction with our system. Through this layer, users can navigate between different screens by clicking on the designated areas or inputting information into designated places. The accessible screens are different for each user type being Student, Graduate, Board and Admin.

2.1.2 Authentication Layer

The Authentication Layer is responsible for ensuring secure access to the app, managing user logins, registrations, and verifying user identities. It implements Passport.js authentication protocols guaranteeing that user data remains accessible only to authorized individuals.

2.1.3 Web Server Layer

Web Server Layer acts as a management system of our system. This layer manages interactions between the User Interface, Authentication, and Database layers, ensuring functionality throughout the app.

2.1.4 Database Layer

The Database Layer is responsible for handling data storage, retrieval, and management. It stores user profiles, posts, direct messages, likes, and other relevant information securely. Implementing an optimized database structure ensures quick access to information, enabling swift responses to user queries such as filtering out posts regarding them being latest, oldest etc.

2.2. Hardware/software mapping

In our system, we've chosen Bootstrap for the frontend and Spring Boot for the backend. We're opting for React paired with JavaScript. Additionally our indirect use of HTML5 and CSS3 ensures support across all modern browsers. Our backend infrastructure relies on Java, Spring Boot, and JDBC, with deployment planned on AWS servers. Considering the yearly student population of Bilkent University being around 12000 people, the number of student clubs in Bilkent University in the

2023-2024 school year being 111 and the total graduate number being around 55000 we anticipate a maximum of 15000 users. We believe that AWS

2.3. Deployment Diagram

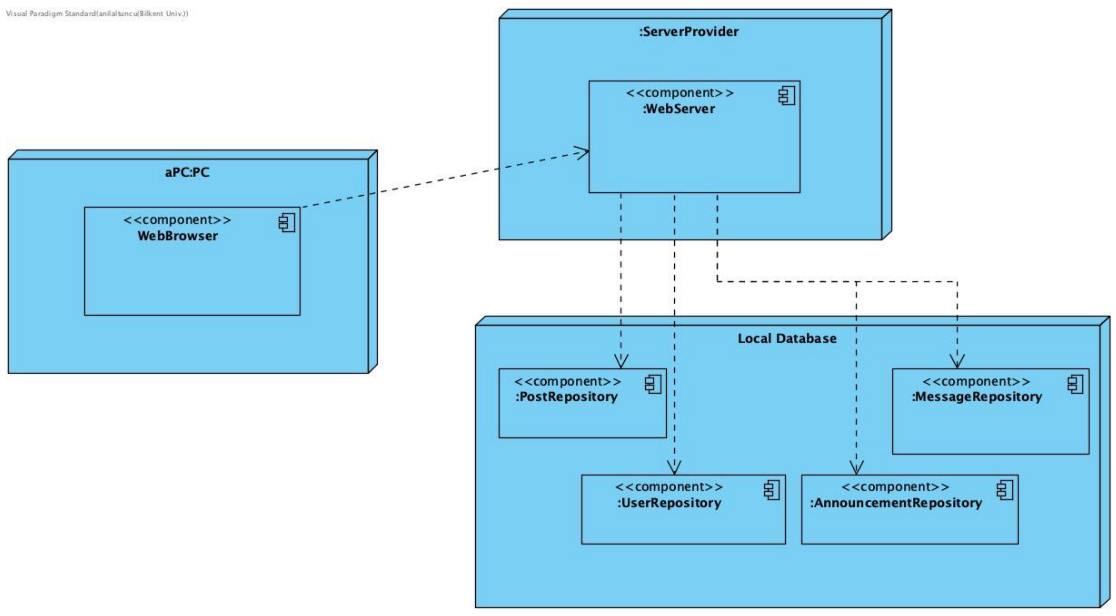


Fig. 2 Deployment Diagram

Fig. 2 displays the interaction between user and server. mainly the objects that inherit the Sendable classThe design depIts design is very similar to subsystem decomposition. Component groups classes that work together closely and they can be classified by their type whereas artifacts represent concrete elements in the physical world

2.4. Persistent data management

OnlyBilkent system significantly relies on complex data, with different types of sendable objects like posts, announcements, and messages and different types of users like students, board accounts, and admin. Therefore, we chose a NoSQL database, MongoDB, to manage this data. MongoDB is a popular open-source NoSQL database management system that falls under the category of document-oriented databases. Document-oriented database MongoDB offers an intuitive data model that is fast and easy to work with and a flexible schema that allows the data model to evolve as application needs change.[1] Unlike relational databases, document databases' schema is dynamic and self-describing, so we don’t need to pre-define it in the database first. Fields can vary from document to document. Structures can be modified at any time, avoiding disruptive schema migrations.[2] We will use MongoDB for user and sendable object data. Moreover, other data, such as images and documents, will be stored in the remote file system and will be accessed via file paths stored in the database.

2.5. Access control and security

OnlyBilkent is a web-based application, and it is exclusively for Bilkent University students and graduates. Therefore, our app requires enhanced security and authorization. We use Node.js for our back-end, and Passport.js is a widely used authentication middleware of Node.js that makes it easy to implement authentication and authorization. Passport.js cleanly encapsulates this functionality while delegating unrelated details, such as data access to the application.[3] The access control is managed by attaining roles for users. Each of the users has certain roles assigned to them. These roles are Student, Graduate, Board Account, and Admin. We use protected routes to restrict the access of different user types to certain user interfaces. This way, users can only see their own user interfaces.

2.5.1 Access Control Matrix

	Student	Board Member	Graduate	Admin
Login	X	X	X	X
Edit Profile	X	X	X	X
Renew Password	X	X	X	X
Create Post	X		X	
Edit Post	X		X	
Ban Profile				X
Add Announcement		X		
View Announcement	X	X	X	X
Remove Announcement		X		X
Edit Announcement		X		X
Report Profile	X		X	
View Reports				X
Search Post	X		X	
Direct Message	X		X	

Give Board Member Permission				X
View Board Member Requests				X
Request Board Member Account	X			
Notify		X		X

3. Low-level Design

3.1 Object design trade-offs

During the low-level design phase, various decisions regarding object design need to be made to ensure an efficient and effective implementation. Trade-offs in object design include deciding between different design options by considering performance, memory usage, and maintainability.

Efficiency and Portability

Efficiency is how well a system or product uses resources to achieve its goals. Since OnlyBilkent is a web application, optimizing resource usage is crucial to ensuring fast response times and optimum performance. This may include efficient algorithms, data structures, and code optimizations. Portability is the ease with which a system or product can be transferred or adapted to different environments or platforms. As a web application, OnlyBilkent may not need to be portable in the traditional sense since it is accessed via web browsers. Since OnlyBilkent is a web application, there is no need for the web application to be portable. Therefore, efficiency is the key design goal in this case.

Functionality and Usability

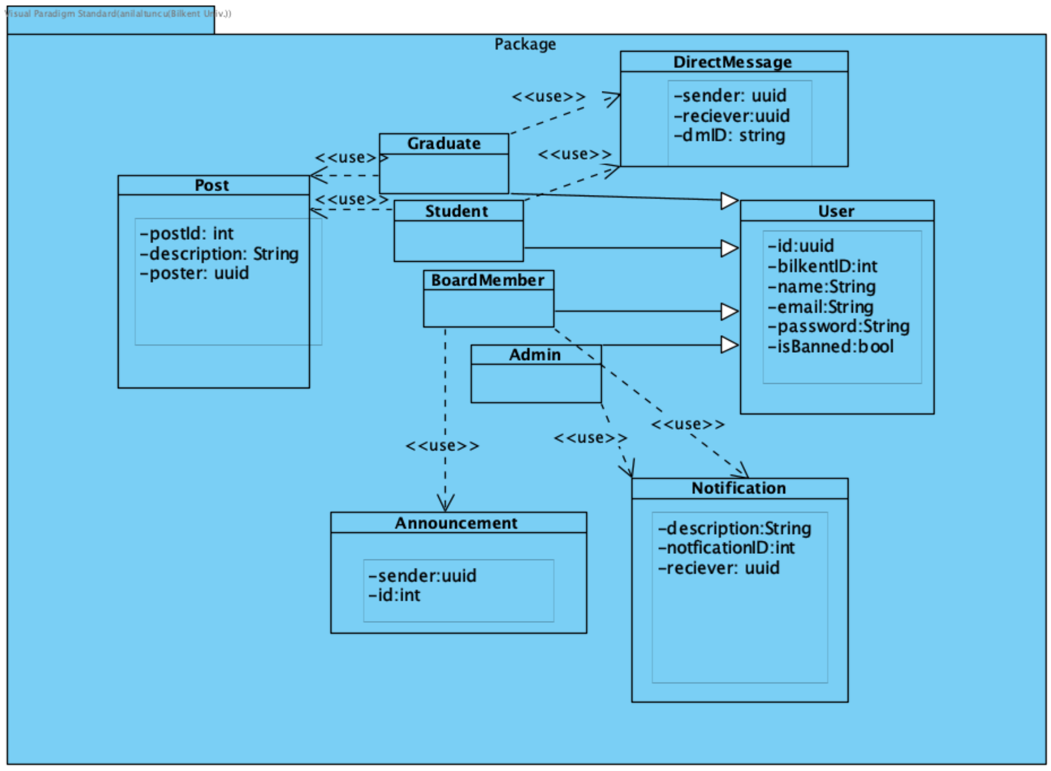
Functionality is the extent to which a system or product performs its intended tasks and features. As a web application, OnlyBilkent should offer features and functions that will meet the needs of users. This may include features related to the core purpose of the app. Usability is the ease with which users can interact with a system or product to achieve their goals. Considering that OnlyBilkent is a web application, providing an intuitive and user-friendly interface is essential. This includes straightforward navigation, simple workflows, and effective user feedback. Added features help users in their lives, so functionality stands out more compared to usability.

Cost and Robustness:

OnlyBilkent, developed by students, has budget constraints. This may include limitations on software licenses, infrastructure costs, and other expenses related to development and maintenance. There is nothing to think about in terms of cost. On the other hand, ensuring robustness is vital. The application must handle errors effectively, handle unexpected input gracefully, and withstand adverse conditions to provide a better user experience.

3.2 Layers

3.2.1 Data Management Layer



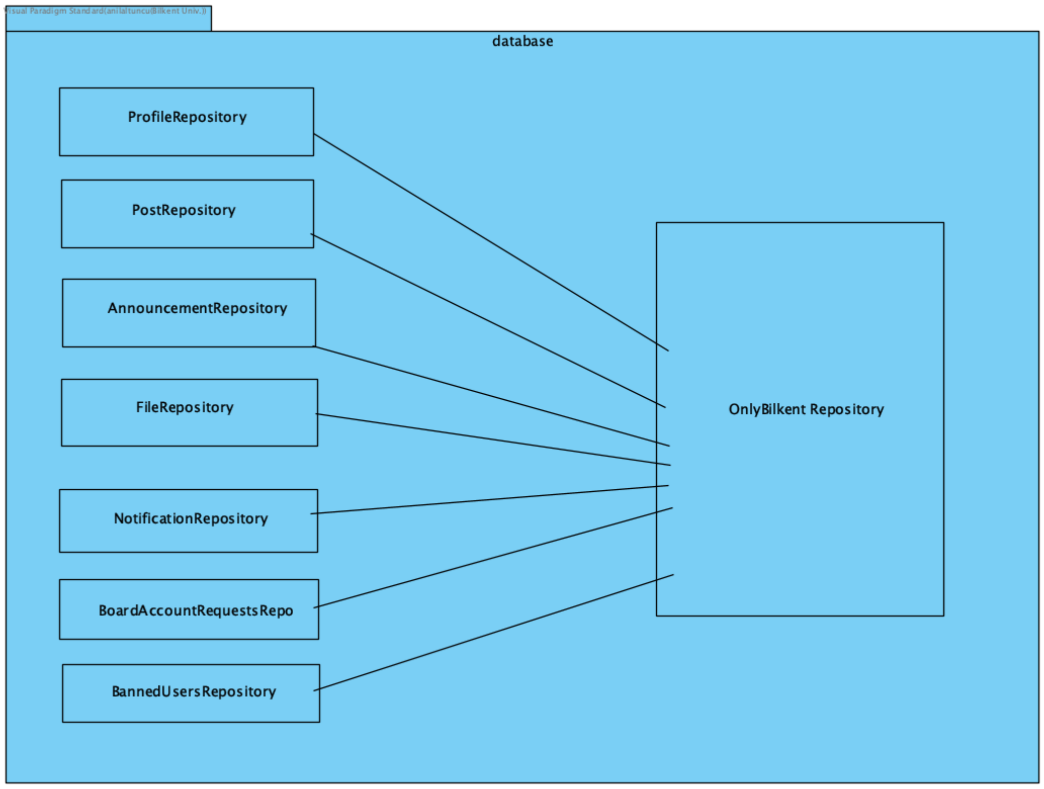


Fig 3 Data Management Layer

3.2.2 User Interface Management Layer

3.3 Class Interfaces

3.3.1 User Interface Layer Class Interfaces

3.3.1.1 Login Page

public loginRequest(String email, String password): On click, sends a request to the web server to verify user credentials.
public goToForgotPassword(): Goes to the forgot password page on click

3.3.1.2 Register Page

public addUser(String email, String password, String password2, String name): On click, registers the user that signs in to the system.
public register(User user): On click, registers the user provided by the admin

3.3.1.3 ForgotPassword Page

public forgotPasswordRequest(String email): On click, sends a verification email to the user's email.
public goToLoginPage(): On click, goes to the login page

3.3.1.4 Dashboard Page

public goToCategoriesPage(): On click, opens the categories page to view posts from different categories.
public goToAnnouncements(): On click, opens the announcements page.
public goToProfilePage(): On click, opens the profile page of the user.
public searchPost(): On click, opens the search bar and initializes searching along posts.
public goToMakePostPage(): On click, opens a making post page to make and send a post.
public goToBoardAccountRequestPage(): On click, opens a board account request page to make a request to the admin for a board account.
public goToAnnouncements(): On click, opens the announcements page.
public goToNotifications(): On click, opens the notifications page.

3.3.1.5 NotificationsPage

public Notification getNotifications(): Retrieves the notifications to display details.

3.3.1.6 Student Profile Page

public editProfile(): Allows the user to edit their profile
public goToMakePostPage(): On click, goes to make post page.
public User getUser(): Retrieves the user information to display details.
public logOut(): Logs out from the account and forwards to auth page

3.3.1.7 Board Account Profile Page

public editProfile(): Allows the user to edit their profile
public goToMakeAnnouncementPage(): On click, goes to make post page.
public User getUser(): Retrieves the user information to display details.
public logOut(): Logs out from the account and forwards to auth page

3.3.1.8 Admin Profile Page

public changePassword(): On click, changes the password of the user.
public changeBio(): On click, changes the bio information of the user.
public changeName(): On click, changes the name of the user on the profile page.
public viewStudent(User user): Gets the desired student from the user list.
public viewBoard(Board board): Gets the desired Board Account from the board list.
public Student [] showUserList(): On page load, fetches the list of students.
public Board [] showBoardList(): On page load, fetches the list of board accounts.
public User [] showBoardAccountRequestsList(): On page load, fetches the list of board account requests.
public User [] showBannedUserList(): On page load, fetches the list of banned users.
public User [] showReportedUserList(): On page load, fetches the list of reported users.
public banUser(User user): Bans the user of the system.

3.3.1.9 Board Account Request Page

public makeBoardAccount(String clubName, String password): Makes a board account request to the admin for club accounts.

3.3.1.10 Make Post Page

public makePost(String category, String title, String image): On click, makes a post and initializes category, title and image of the post. On clicking 'Send' sends the post.

3.3.1.11 Make Announcement Page

public makeAnnouncement(String title, String content): Makes an announcement about a club event.

3.3.1.12 Categories Page

public goToDormitoryCategory(): On click, goes to posts about dormitories.
public goToSecondHandMaterialPage(): On click, goes to posts about second hand school material selling.
public goToLessonRecommendPage(): On click, goes to posts about lesson recommendations.
public goToLostFoundPage(): On click, goes to posts about lost and found items around campus.

public messageUser(): Sends a message to the sender of the post.

3.3.2 Data Management Layer Class Interfaces

3.3.2.1 Profile Repository

findEmailById(id:UUID): Finds user email from their uuid.
findBioById(id:UUID): Finds user biography from their uuid.
findColorById(id:UUID, colorNo: int): Finds the color theme of a user's profile page using the theme color number and user uuid.
findContactById(id:UUID): Finds user contact from their uuid.
findNameById(id:UUID): Finds user name from their uuid.
findProfilePictureById(id:UUID): Finds user profile picture from their uuid

3.3.2.2 Post Repository
findPostByID(postId:int): Finds spesific post by given id.

3.3.2.3 Announcement Repository
findAnnouncement(id:int): Finds the announcement with the given announcement id.

3.3.2.4 File Repository
findProfilePhotoById(id:UUID): Finds the profile photo with the given user uuid.
findPhotoById(postId:int): Fİnds the photo if included in a specific post.

3.3.2.5 Account Repository

findAccountById(id:UUID): Finds the user account with the given uuid.
findAccountByEmail(email:String): Finds the account with the given email.
findAccountByBilkentId(bilkentId:long): Finds the user account with the given Bilkent Id.

3.3.2.6 Notification Repository
getNotificationsByID(id:UUID): retrieves the notifications of the specific account by given uuid.

3.3.2.7 Board Account Request Repository
getRequests(): Retrieves users which made requests.

3.3.2.8 Banned Users Repository
getBannedUsers(): Retrieves banned users.

3.3.2.9 Direct Message Reporsitory
findMessage(dmID:string): Finds the direct message with the given dmID.

3.4 Design Patterns

3.4.1 Strategy Pattern [4]

Implementing the Strategy Pattern within our campus connection system presents an advantageous approach to handle diverse strategies for user reports, Admin functionalities, and user bans based on varying criteria and severity levels. This pattern allows us to encapsulate these strategies as interchangeable components, enabling us to select the appropriate strategy dynamically at runtime. For instance, different handling approaches might be required for user reports—some may necessitate immediate action, while others may demand a more nuanced review process. By employing the Strategy Pattern, we can easily swap and apply different algorithms or approaches without altering the core system, thus ensuring scalability, flexibility, and maintainability. This pattern also fosters a clear separation between the report handling logic and the rest of the system, streamlining updates or changes to the handling strategies without disrupting the system's integrity. Ultimately, the Strategy Pattern empowers our system to adapt to varying scenarios and evolving requirements by offering a modular and adaptable strategy selection mechanism.

3.4.2 Factory Method Pattern [4]

The Factory Method Pattern proves advantageous for the dynamic creation of various post types within our system. When a user initiates the creation of a post—such as for lost and found items, dormitory mate finding, or selling second-hand school materials—the Factory Method determines the specific type of post object to generate based on the user's selection. This design pattern enhances flexibility by enabling seamless introduction of new post types without necessitating modifications to existing code. It effectively abstracts the object creation logic, reducing coupling between the creation process and the rest of the system. Centralizing the creation logic in this manner streamlines management and modifications to the instantiation process, providing a scalable and adaptable framework for expanding our application.

4. Glossary and References

[1]: https://www.mongodb.com/document-databases, 14 Nov, 2023
[2]:https://www.mongodb.com/document-databases#what-makes-document-databases-different-from-relational-databases 14 Nov, 2023
[3]: https://www.passportjs.org/concepts/authentication/password/ 14 Nov, 2023
[4]: Freeman, Eric, et al. *Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software*. O'Reilly, 2021. 3 Dec, 2023