



CS319 Object-Oriented Software Engineering

Süleyman Yağız Başaran

22103782

Gülbera Tekin

2200335420

Onur Tanınmış

22003312

Serhat Yılmaz

22002537

Melisa İrem AkeI

22003923

Betül Doğrul

22003559

Table of Contents

CS319 Object-Oriented Software Engineering	1
1. Introduction	3
1.1. Purpose of the System	3
1.2. Design Goals	3
1.2.1. Safety	3
1.2.2. Usability	3
1.2.3. Performance	3
1.2.4. Maintainability	4
1.2.5. Environment Requirements	4
1.2.6. Contact Support	4
1.2.7 Independence	4
2. High-Level Software Architecture	5
2.1. Subsystem decomposition	5
2.1.1 Presentation Layer	6
2.1.2 Application Layer	7
2.1.3 Database Interface Layer	8
2.1.4 Data Layer	9
2.2. Hardware/Software Mapping	9
2.3. Persistent Data Management	10
2.4. Access Control and Security	10
2.4.1 Access Control Axis	11
2.5. Global Control Flow	15
2.6. Boundary Conditions	15
2.6.1. Initialization	15
2.6.2. Termination	16
2.6.3. Failure	16
3. Low-level design	16
3.1 Object design trade-offs	16
3.2 Final Object Design	16
3.3 Packages	18
3.3.1 External Packages	18
3.3.2 Internal Packages	19
3.5 Design Patterns	19
3.5.1 Façade Pattern	19
3.5.2 Strategy Pattern	20

1. Introduction

1.1. Purpose of the System

The primary purpose of developing CampusConnect as a web-based application is to establish a robust and secure communication platform tailored for the Bilkent University community. Centered on the core design goals of safety, usability, performance, maintainability, environment requirements, and contact support, the system aims to provide students with a seamless and trustworthy environment. Ensuring the safety and privacy of users, the platform facilitates accessible communication and maintains high-performance standards. With a user-centric focus, CampusConnect prioritizes usability for an intuitive experience, while its robust infrastructure ensures maintainability and adaptability. The system aligns with environmental considerations and promotes sustainable practices. Lastly, including a contact support feature reinforces the commitment to user assistance, enhancing the overall reliability and satisfaction of the CampusConnect platform for the Bilkent University community.

1.2. Design Goals

1.2.1. Safety

CampusConnect is a campus-only website requiring school ID and mail at registration to ensure platform safety from outer environments. CampusConnect ensures user privacy and safety. When a user loses their Bilkent ID card, instead of sharing their ID in the feed, it alerts the user that their ID is found. The process is not open to others to protect users' personal information and identities. The FreeZone feature is against hate speech, so that moderators will have a strict policy against possible harmful content, and it will provide a safe and positive environment for users. After users use features like second-hand sales or donations, they can rate each other, increasing their accountability. The user will be able to give feedback, and it will improve safety measurements for the future.

1.2.2. Usability

CampusConnect prioritizes an intuitive and seamless user experience to ensure accessibility and satisfaction among Bilkent University students. The user interface (UI) design is crafted with clarity and simplicity, allowing users to navigate effortlessly across the various pages, including Login/Create Account, Profile, Second-Hand, Donation, Borrowing, Lost & Found, FreeZone, and the LiveChat section.

1.2.3. Performance

CampusConnect is designed with a robust and efficient tech stack to ensure optimal speed, responsiveness, and scalability. The backend, powered by Spring Boot, leverages the strengths of the Java programming language, simplifying development with defaults for code and annotation configurations. This choice aims to enhance the overall performance of our application. On the front end, we have adopted React, known for its component-based architecture, fostering modularity and reusability. This approach facilitates more manageable

maintenance and supports scalability as our project evolves. Regarding data management, PostgreSQL has been selected as the database, offering capabilities to handle large datasets and concurrent transactions. Features such as partitioning and indexing contribute to improved performance, mainly as data volumes grow.

1.2.4. Maintainability

By employing Spring Boot for the backend, our code structure follows a layered approach, enhancing maintainability through a clear separation of concerns. The convention-over-configuration nature of Spring Boot simplifies development, ensuring a standardized codebase that is easy to manage and update. React's component-based architecture on the front end fosters modularity and reusability, promoting straightforward maintenance and scalability.

1.2.5. Environment Requirements

The system is designed for every device with an internet connection. The technologies chosen—Spring Boot, React, and PostgreSQL—ensure compatibility with standard local web servers and popular browsers, providing a consistent user experience. The application is developed with an emphasis on energy efficiency and minimal environmental impact, aligning with sustainable practices.

1.2.6. Contact Support

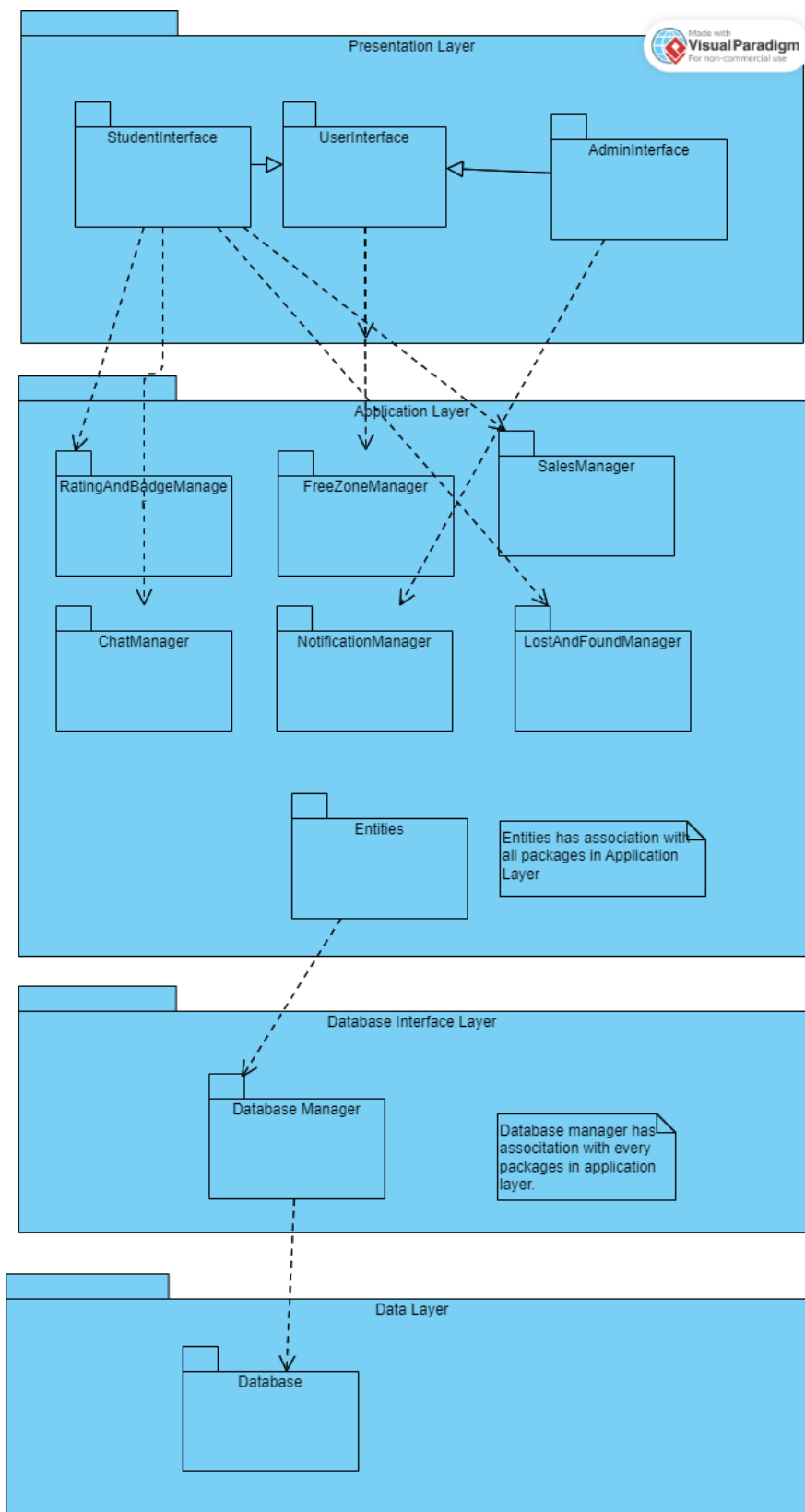
Users can contact support regarding their problems during LiveChat or FreeZone. Moreover, they will also be able to send feedback to support future enhancements to the application.

1.2.7 Independence

The system is designed so that the classes, modules, and subsystems are independent of each other to have a manageable and stable system in case of a modification.

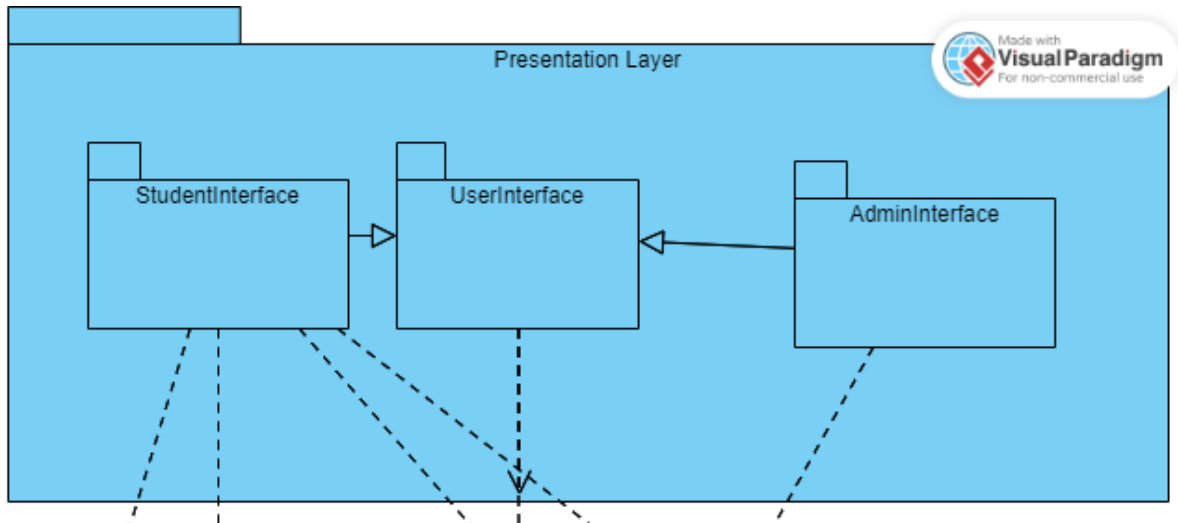
2. High-Level Software Architecture

2.1. Subsystem decomposition



2.1.1 Presentation Layer

The presentation layer serves as the UI(User Interface), where users interact with the functionalities of CampusConnect. The Admin Interface is responsible for managing the FreeZone feature of the application and disabling users from sharing harmful content in FreeZone. The student interface encompasses all elements that users can see or interact with.



Student Interface:

- This package interacts with the Application Layer components like SalesManager, LostAndFoundManager, ChatManager, and RatingAndReviewManager.
- This package specifies interactions with the users with the authentication.
- It extends the UserInterface package.

Admin Interface:

- This package includes screens and functionalities only available to admins to edit FreeZone harmful content and see the reports.
- It extends the UserInterface package.
- The admin interface is associated with NotificationManager so that they can send messages to users.

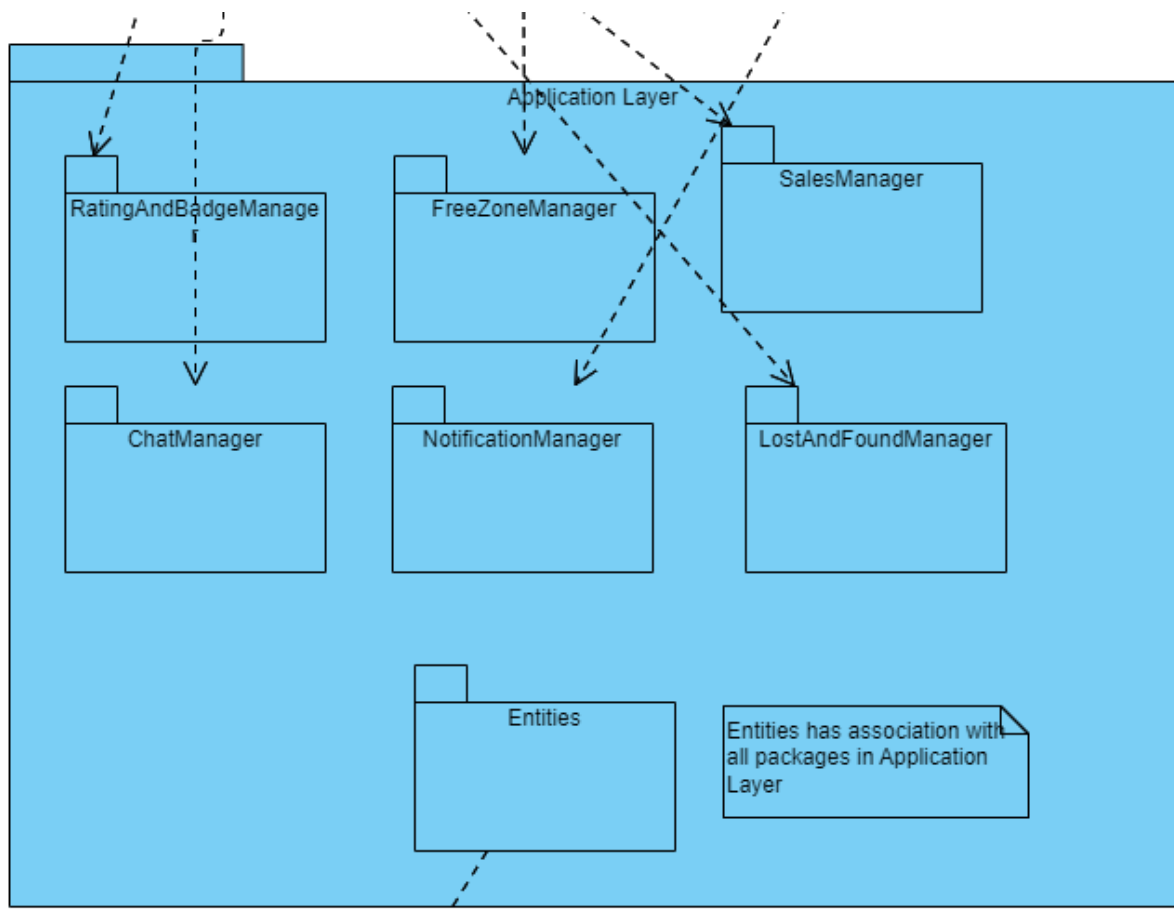
User Interface:

- This general package contains screens and interactions available to all authenticated users.

- The user interface is associated with FreeZone, a common platform for students and admins.

2.1.2 Application Layer

Control object packages that modify data based on retrieved data make up the Application Layer. In the Application Layer, this Presentation Layer data is processed to provide the referring information according to the Presentation Layer. These control object packages interact with the Database Manager within the Database Interface Layer.



RatingAndBadgeManager:

- This package consists of control classes related to the rating & badge system of the application.
- This package is associated with StudentInterface.

FreeZoneManager:

- This package consists of control classes related to the FreeZone API management.
- This package is associated with UserInterface.

SalesManager:

- This package consists of control classes related to the Second-Hand Sales API management.
- This package is associated with StudentInterface.

ChatManager:

- This package consists of control classes related to the LiveChat API management.
- This package is associated with StudentInterface and every other package in the Application Layer.

Notification Manager:

- This package consists of control classes related to sending and receiving notifications.
- This package is associated with AdminInterface, which is every other package in the Application Layer.

LostAndFoundManager:

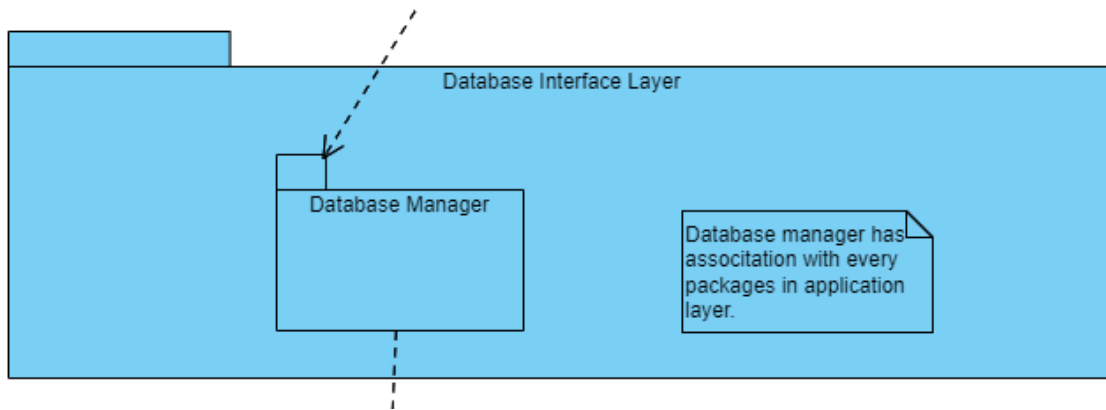
- This package consists of control classes related to the Lost & Found API management.
- This package is associated with StudentInterface.

Entities:

- This package consists of entities in the program.
- This package is associated with every other package in the Application Layer.

2.1.3 Database Interface Layer

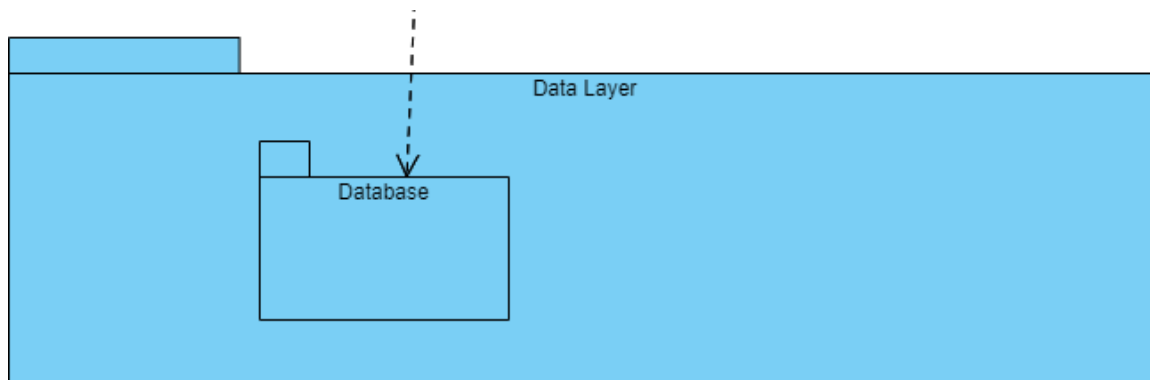
The Database Interface Layer contains one element: the Database Manager. This is a transactional managing layer between the Application and Data Layer.



The Database Manager package consists of classes that are related to database management. This package is associated with every other Application Layer and Database package.

2.1.4 Data Layer

The Data Layer includes only the database of the application. We chose PostgreSQL to handle tasks such as storing user profiles, managing second-hand item listings, recording transactions, and facilitating user communication. It plays a vital role in ensuring the application's reliability, security, and performance.



Database:

- This package refers to the external database (PostgreSQL).
- This package is associated with the Database Manager package.

2.2. Hardware/Software Mapping

Our app uses Spring Framework 3.1.5 with Java 17 for its backend. Because of its modular design, the Spring framework creates a flexible environment, making it easier to extend functionality in the program in the future. The management of components and management of the database is easier with the spring framework. Also, spring provides us with its security features, which let the user securely enter the program, post their items, and

chat. Also, the front-end and back-end interaction is easier with the Spring framework since we use React 18.2.0.

We chose React for the front end because it is easier to manage complex user interfaces in React. Also, the component-based architecture of React is advantageous for the code reusability, maintainability, and flexibility to add features of chat boxes, item posts, and user profiles. Our LiveChat feature needs a dynamic continuous update of the chat, and the virtual-DOM feature of React eases this process.

For the database, we chose PostgreSQL because it is an open-source relational database, which makes it easy to maintain structured data such as user information, posts, donations, sales, etc. Also, its scalability feature allows us to manage a growing number of user data.

The project requires Java 17, an appropriate Node.js version for React (npm version 9.8.1), a computer with a minimum of 8 GB RAM, a suitable IDE for Java 17, and an internet connection as its hardware components.

2.3. Persistent Data Management

For the CampusConnect database, PostgreSQL was chosen due to its widespread usage in computer engineering companies in Turkey and its robust way of storing data. CampusConnect is an OOP-based application, so it was also one of the factors that led to PostgreSQL being chosen.

To keep data persistent and reliable, we utilized SpringBoot with PostgreSQL. SpringBoot will manage application backend services. SpringBoot will automatically update the database in response to users' interactions. Using these technologies, we ensure that CampusConnect can grow, stay safe, and easily update. This way, it can adapt and serve the changing requirements of its users effectively.

2.4. Access Control and Security

For access control and security in our CampusConnect application, we will use Spring Security. This framework has a feature called "Spring Security Authenticate," which we plan to implement. It's a tool that helps us manage who can access our application, ensuring only authorized users get in.

Spring Boot is another tool we use for handling common online threats. This means it can protect our application from web attacks that might disrupt our service or access sensitive information.

A crucial part of keeping our application secure is how we manage login details. Spring Boot helps us here by securely storing these credentials. It does this by 'hashing' the passwords, turning the original password into a different string of characters. Even if someone gets access to our database, they won't be able to understand or use these hashed passwords.

Moreover, Spring Boot's security capabilities don't just stop at checking who can log in (authentication); they also manage what each user can do once logged in (authorization). This means we can set different levels of access and permissions for other users, depending on their role in the application. Essentially, we're creating a secure environment for CampusConnect using Spring Security and Spring Boot. This will help us ensure that our users' data is protected and that the application runs smoothly and securely.

2.4.1 Access Control Axis

	User							
	Receiver	Donator	Buyer	Seller	Lender	Founder	Admin	
Register (CreateAccount)	registerUser(string, string)	registerUser(string, string)	registerUser(string, string)	registerUser(string, string)	registerUser(string, string)	registerUser(string, string)		
Login	login(LoginRequest)	login(LoginRequest)	login(LoginRequest)	login(LoginRequest)	login(LoginRequest)	login(LoginRequest)	login(LoginRequest)	
Logout	logout(User)	logout(User)	logout(User)	logout(User)	logout(User)	logout(User)	logout(User)	
View Profile	viewProfile(long)	viewProfile(long)	viewProfile(long)	viewProfile(long)	viewProfile(long)	viewProfile(long)	viewProfile(long)	
Edit Profile	editProfile(long)	editProfile(long)	editProfile(long)	editProfile(long)	editProfile(long)	editProfile(long)		

<i>Delete Account</i>	deleteAccount(long)	deleteAccount(long)	deleteAccount(long)	deleteAccount(long)	deleteAccount(long)	deleteAccount(long)	deleteAccount(long)
<i>View Transactions At Profile</i>	viewTransactionsAtProfile(long)	viewTransactionsAtProfile(long)	viewTransactionsAtProfile(long)	viewTransactionsAtProfile(long)	viewTransactionsAtProfile(long)	viewTransactionsAtProfile(long)	
<i>View Personal Posts At Profile</i>	viewPersonalPostsAtProfile(long)	viewPersonalPostsAtProfile(long)	viewPersonalPostsAtProfile(long)	viewPersonalPostsAtProfile(long)	viewPersonalPostsAtProfile(long)	viewPersonalPostsAtProfile(long)	
<i>Give Rate At Profile</i>	giveRateAtProfile(long, long)	giveRateAtProfile(long, long)	giveRateAtProfile(long, long)	giveRateAtProfile(long, long)	giveRateAtProfile(long, long)	giveRateAtProfile(long, long)	
<i>Make Donation</i>		postItem(FeedPost)					
<i>Take Donation</i>	takeDonation(FeedPost)						
<i>Search Donation</i>	searchDonation(String)	searchDonation(String)					
<i>View Donation Feed</i>	viewDonationFeed()	viewDonationFeed()					
<i>Post Item For</i>					postItem(FeedPost)		







<i>Borrow</i>)		
<i>View Borrow Feed</i>					viewBorrowFeed()		

<i>Request Borrow Item</i>	requestBorrowItem(FeedPost)						
<i>Search Borrowing Feed</i>	searchFeed(Borrow)				searchFeed(Borrow)		
<i>Sell Item</i>				sellItem(FeedPost)			
<i>Buy Item</i>			buyItem(FeedPost)				
<i>Search Second-hand sales</i>			search(SecondHand, key)	search(SecondHand, key)			
<i>View Second-hand Feed</i>			viewFeed(SecondHand)	viewFeed(SecondHand)			
<i>Report lost ID</i>						reportID(long)	
<i>Search Lost and Found</i>	search(LostAndFound,					search(LostAndFound,	

	key)					key)	
<i>Post Found Item</i>						postItem(FeedPost)	
<i>View Lost and Found Feed</i>	viewFeed(LostAndFound)					viewFeed(LostAndFound)	
<i>Comment</i>	comment(FeedPost)	comment(FeedPost)	comment(FeedPost)	comment(FeedPost)	comment(FeedPost)	comment(FeedPost)	
<i>Like/Reaction</i>	likePost() giveReaction(Reaction)	likePost() giveReaction(Reaction)	likePost() giveReaction(Reaction)	likePost() giveReaction(Reaction)	likePost() giveReaction(Reaction)	likePost() giveReaction(Reaction)	
<i>View Freezezone Feed</i>	viewFeed(Freezezone)	viewFeed(Freezezone)	viewFeed(Freezezone)	viewFeed(Freezezone)	viewFeed(Freezezone)	viewFeed(Freezezone)	
<i>Post On Freezezone</i>	postItem(FeedPost)	postItem(FeedPost)	postItem(FeedPost)	postItem(FeedPost)	postItem(FeedPost)	postItem(FeedPost)	
<i>Delete Content</i>	deleteContent(FeedPost)	deleteContent(FeedPost)	deleteContent(FeedPost)	deleteContent(FeedPost)	deleteContent(FeedPost)	deleteContent(FeedPost)	deleteContent(FeedPost)
<i>Live Chat</i>	chatWith(User)	chatWith(User)	chatWith(User)	chatWith(User)	chatWith(User)	chatWith(User)	

<i>Report Content</i>	reportContent(FeedPost)	reportContent(FeedPost)	reportContent(FeedPost)	reportContent(FeedPost)	reportContent(FeedPost)	reportContent(FeedPost)	
<i>Contact Developers</i>	contactDev()	contactDev()	contactDev()	contactDev()	contactDev()	contactDev()	contactDev()

Table 1: Access Control Matrix

 = General
  = Donation Actions
  = Borrow Actions
  = Second-Hand Sales
 = Lost and Found item Actions
  = Freezone

2.5. Global Control Flow

The user actions control the application's control flow. The user's interaction with the user interface will warn the controller objects of the backend. According to the interaction type of the user, data will be fetched from the database, or new data will be added to the database. Therefore, we use event-driven control flow as our global control flow mechanism. The management process of the application is distributed among different objects in the backend to avoid having centralized management, which may cause more problems than implementing a decentralized design.

2.6. Boundary Conditions

2.6.1. Initialization

To set up the front end with React, users should enter “**npm install**” to install all necessary packages, followed by “**npm build**” to build the application.

On the backend side, our application will be ready to start after completing the required configurations and making any critical Maven updates. The first step is to run the Spring Boot backend. Once Spring Boot runs, the “**npm start**” command can be used in the React command prompt to launch the front end.

CampusConnect, a campus-only web application, restricts unlogged users from viewing only the landing page. New users can register, but they must provide valid credentials, such as a university ID and email, for authentication. Without these valid credentials, the system will not grant access.

2.6.2. Termination

The application can be terminated in a development environment, such as by using CTRL+C, the stop button in the SpringBoot IDE (like IntelliJ), or the React IDE (such as Visual Studio Code). Additionally, unexpected errors and crashes may lead to the application's termination. Furthermore, turning off the device running the application is another potential cause for the app to stop.

2.6.3. Failure

To create a user-friendly interface, we provide error messages with explanatory information about the reason for the invalid operation or the failure. User support and contact information are provided to handle unexpected and specific user-dependent errors. In case of a critical failure, the system will be shot down, and the user will need to reenter the system again. After login, the last updated database version will be used to fetch and add new data.

3. Low-level design

3.1 Object design trade-offs

- **Maintainability vs. Performance**

The maintainability and the performance of the CampusConnect system are the two most crucial design decisions. The code and the organization of the systems should be maintainable to have concise, understandable, and adaptable code. Also, the system's performance should be high to provide an excellent application to the user and a better execution speed and responsiveness. Therefore, the maintainability and performance of the system becomes a trade-off.

- **Functionality and Usability:**

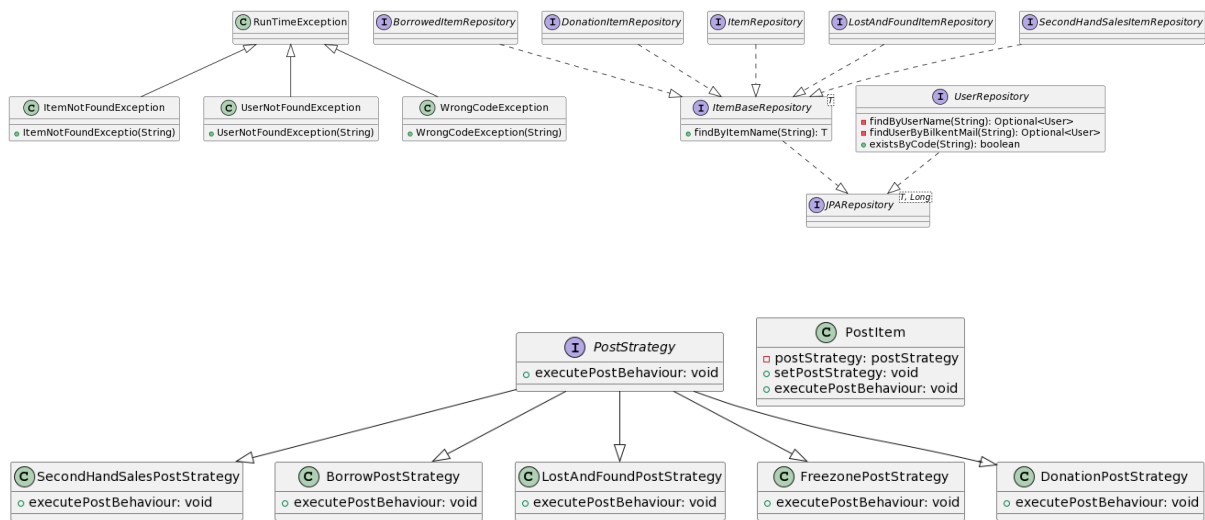
The more functionality there is, the less usability of the application. More functionality causes a more complex user interface and may confuse the users while using the application. The functionality provides a range of features from which the user can benefit. On the other hand, it complicates the interface and may decrease the application's user-friendliness.

3.2 Final Object Design

The class diagrams are provided on the following pages. For the sake of simplicity, the getter and setter methods are not included in the class diagrams.

The class diagram can be seen from the following link:
<https://online.visual-paradigm.com/community/share/untitled-1l5zbuv9jo>





3.3 Packages

3.3.1 External Packages

- **SpringBoot:** This package is a Java framework that eases the backend implementation in the application. This package interacts with both React and Database.
- **React:** This package develops the application's frontend and interacts with SpringBoot.

In the backend **SpringBoot**, the following packages are used:

- Spring Data Jpa
- Spring Hateoas
- Spring Web
- Lombok
- Spring Security
- PostgreSQL Driver

For the frontend, the following packages are used:

- "axios": "^1.6.1",
- "css": "^3.0.0",
- "react": "^18.2.0",
- "react-dom": "^18.2.0",
- "react-icons": "^4.12.0",
- "react-router-dom": "^6.18.0",
- "react-scripts": "5.0.1",
- "react-scroll": "^1.9.0",

- "tailwind": "^4.0.0",
- "tailwindcss": "^3.3.6",
- "web-vitals": "^2.1.4"

3.3.2 Internal Packages

- **Controller Package:** This package involves boundary classes of items and users.

ItemController, UserController

- **Entities Package:** This package involves all boundary classes of item and user entities.

BorrowedItem, DonationItem, Item, LostAndFoundItem, SecondHandSalesItem, User

- **Exception Package:** This package involves all boundary classes of exceptions.

ItemNotFoundException, UserNotFoundException, WrongCodeException

- **Repositories Package:** This package involves all boundary classes of repositories.

BorrowedItemRepository, DonationItemRepository, ItemBaseRepository, ItemRepository, LostAndFoundItemRepository, SecondHandSalesItemRepository, UserRepository

- **Service Package:** This package involves all boundary classes of services.

ItemService, UserService

- **Config Package:** This package is used to implement the Live-Chat feature of the application.

WebSocketConfig, Controller Package

3.5 Design Patterns

3.5.1 Façade Pattern

We implement the Façade Pattern while organizing our database connection classes. We abstracted the database layer using repositories. It encapsulates the complexities of the data access process in the backend and provides a simplified interface to interact with the database. Using repositories, we used APIs to access and modify data, providing a simple and manageable interface for us. Therefore, the modifications to the database are localized so that the rest of the application is not affected by the changes.

We also used the Façade Design Pattern in our user authentication. We created a UserRepository to interact with the database and then created the UserService class to

control the authentication-related events of the user. Therefore, the changes can be made inside one class and do not affect other entities of the system.

The item class is also implemented with the Façade Design Pattern. There is a repository to interact with the database and a service to control the posting, deletion, and modification. Therefore, it creates a Façade for database operations.

3.5.2 Strategy Pattern

We used the Strategy Pattern to organize the posts and their respective algorithms. There are five types of posts: SecondHandSalesPost, BorrowPost, LostAndFoundPost, DonationPost, and FreezonePost, and each has different attributes and behaviors in different scenarios. We chose to use strategy patterns for dynamic selection and interchangeability for algorithms. Also, we used the strategy pattern in the sorting strategy of feeds and posts.