CS 319 Object-Oriented Software Engineering

Deliverable 4

Design Goals / High Level Architecture

S2T8

Bilginer Oral	Ercan Bahri Nazlıoğlu	Alper Biçer	Mohamed Amine Boukattaya	Eren Karakaş
22103163	21903151	22003097	22301131	22002722

Table of Contents

1. Purpose of the System

CampusConnect was born out of the desire to enhance the university experience for Bilkent University students. Currently, when a Bilkent student loses an item, they have to check various Instagram accounts and navigate through their posts, which are not exclusive to reporting found items, to have a chance of finding their item. Similarly, if they want to save costs, they have to look at largely deserted Facebook groups to find second-hand items. CampusConnect is designed to solve this fragmentation and become a one-stop solution for students' various academic and social needs. It streamlines and simplifies operations such as reporting lost or found items, facilitating second-hand sales, enabling borrowing items, and hosting donation campaigns.

2. Design Goals

2.1 Maintainability

CampusConnect is structured to allow ease of maintenance. We've implemented a layered architecture that separates concerns, making the codebase more straightforward to navigate and update. Additionally, we used Swagger UI to maintain up-to-date API documentation. Therefore, as an example, a front-end engineer does not need extensive back-end knowledge to contribute to CampusConnect; looking at the existing API documentation is usually enough. Additionally, we used the singleton design pattern for our manager classes. As a result, maintainers have to deal with only one instance of such classes, avoiding problems such as inconsistency and obscure side effects.

2.2 Usability

The Presentation Layer of CampusConnect is designed with ease of user interaction in mind. The React.js framework enables us to create a dynamic and responsive user interface that caters to both administrators and ordinary users, ensuring a seamless experience across various devices. All of CampusConnect's functionality is accessible by going at most three pages deep, and the main functionalities a user might want to utilize a lot are grouped in an app-wide persistent sidebar.

2.3 Reliability

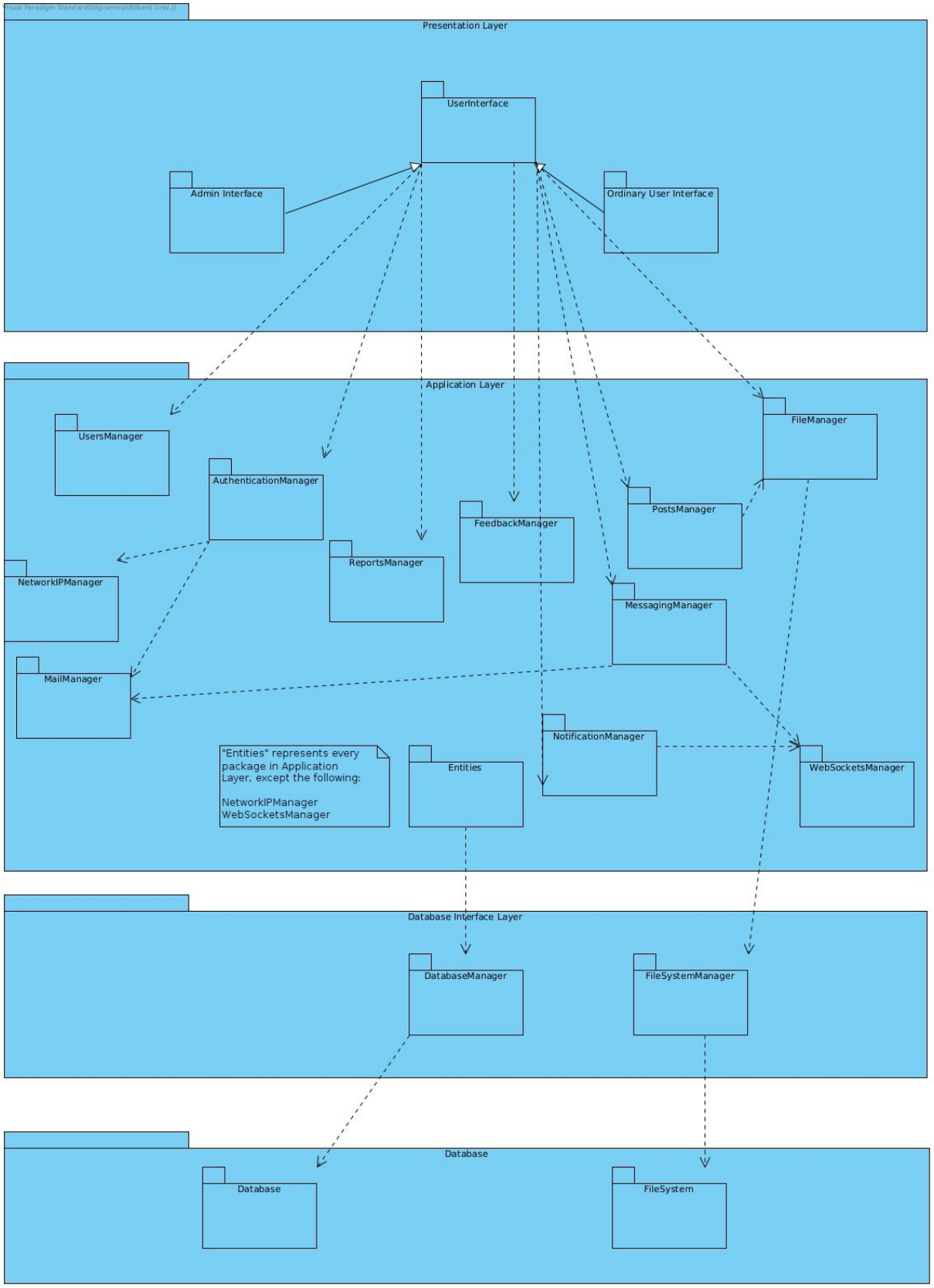
CampusConnect's back-end design uses industry standard and battle-tested technologies such as FastAPI and PostgreSQL in order to achieve robustness and reliability. Its database is hosted on AWS, with multiple local copies also existing to use if a problem occurs.

2.4 Independency

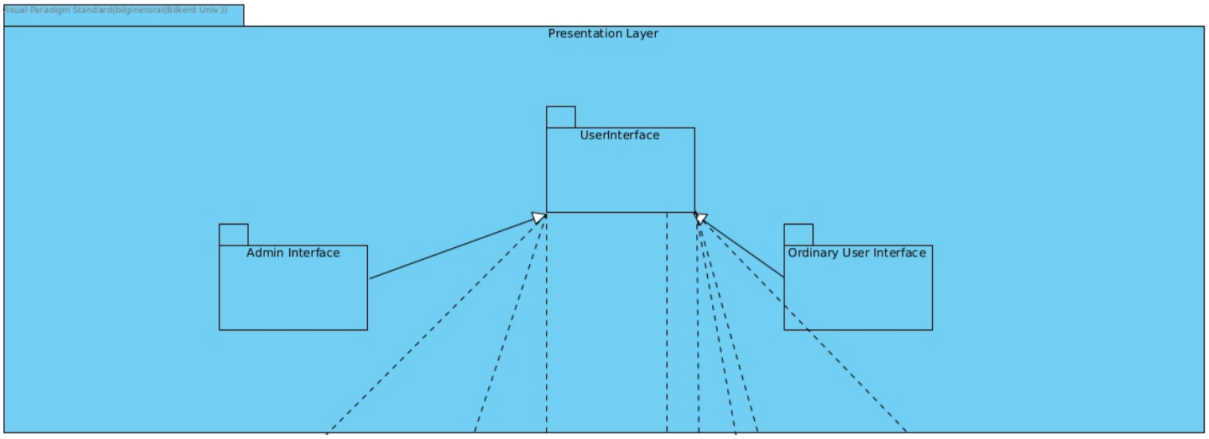
The layered architecture used allows CampusConnect's various subsystems to remain independent from each other. These layers of isolation mean changes in a layer of the architecture generally do not impact other components in different layers. Effectively, this design also allows the decoupling of layers, where a layer does not need to know the inner workings of another layer to function. Therefore, an engineer working on a layer also does not need to know the inner workings of another layer, enabling the maintainability advantages mentioned above. Similarly, we used the repository design pattern to abstract away the complexities of data access. This data access abstraction allows us to decouple our subsystems further, making even changing technologies possible and relatively painless.

3. Subsystem Decomposition

The layered structure of the proposed software is as follows:



3.1 Presentation Layer



Presentation layer, as its name suggests, is the layer that presents the proposed software to the end user. In the case of our system, the term “end user” refers to administrators of the application and the ordinary users. This layer prompts the users of the application with appropriate pages, input prompts, etc. Information that users write in this layer are directed to the application layer for further processing.

Admin Interface:

- This package consists of pages that the administrators of the system can interact with. These include the “Admin search” screen, “Reports” screen, and “Homepage”. In these screens, administrators can take actions to remove a content, or ban a user.
- This package is generalized to UserInterface package.

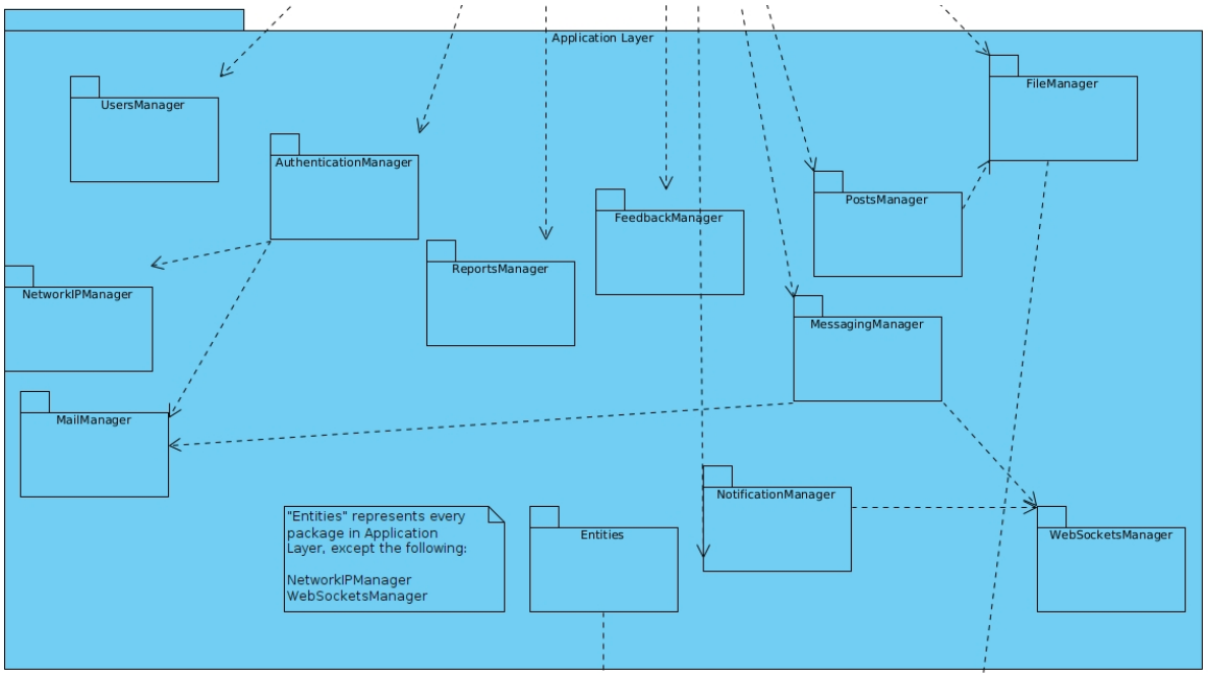
Ordinary User Interface:

- This package, similar to Admin Interface, consists of static and dynamic pages that ordinary users of the system can interact with. These include but are not limited to the “Homepage”, in which users can upload, view, edit and delete posts, the “Search” screen, the “Profile” screen, the Notifications” screen, and the “Messaging” screen.
- This package is generalized to the User Interface package.

User Interface:

- This package consists of every screen that any type of user (be it administrators or ordinary users) can interact with.
- This package is associated to the following packages in the Application layer (which implies that the user input/output is processed in the Application layer and then presented in the Presentation layer):
    - Authentication Manager
    - Reports Manager
    - Feedback Manager
    - Notification Manager
    - Messaging Manager
    - Posts Manager
    - File Manager
    - Users Manager

3.2 Application Layer



The application layer is the backbone of our proposed software. In this layer, requests made by the presentation layer are thoroughly processed within interconnected packages. Furthermore, the decision of whether to read/write to or from the database is taken in this layer, therefore the packages in this layer are associated to the Database Interface Layer.

UsersManager:

- This package handles operations such as blocking/unblocking other users and searching.

NetworkIPManager:

- This package is used to check the IP addresses of the connected clients.
- This package is called by the Authentication Manager, since the client IP address is not a Bilkent IP Address the connection will be refused by the system.

MailManager:

- This package is handles the email notifications that are to be send to the users of the system.
- This package is called by the Authentication Manager for two factor authentication.
- This package is also called by the Messaging Manager in case there was a messaging request to a offline user.

AuthenticationManager:

- This package handles the login/registration operations.
- In order to operate properly, this package is associated to Network IP Manager and Mail Manager.

ReportsManager:

- This package handles the reporting requests that are made by users.
- Ordinary User requests include the reporting of a comment, post or a user.
- Administrator requests include removing a reported comment, post or a user.

FeedbackManager:

- This package receives the user feedback from the “Feedback” screen in the presentation layer and takes the necessary actions.

NotificationManager:

- Main purpose of this package is to manage user notifications. When an action done on the presentation layer requires a user to be notified, the Notification Manager handles the necessary operations to notify users in time.
- This package calls WebSocketsManager in order to establish a real-time notification ability for the system.

PostsManager:

- While serving as one of the most important packages in our software, this package handles every requests that are related to creation, deletion, viewing and editing of posts made by users.
- This package manages the requests made for the posts types “Lost and Found”, “Second Hand Sales”, “Donations” and “Borrowing”.
- This packages calls the File Manager package, in order to manage image uploads/downloads for posts.

FileManager:

- This package is responsible of handling image uploads/downloads.
- The User Interface in the presentation layer is associated to this package for handling profile image uploads/downloads.
- The Posts Manager in the application layer is associated to this package for handling posts image uploads/downloads.

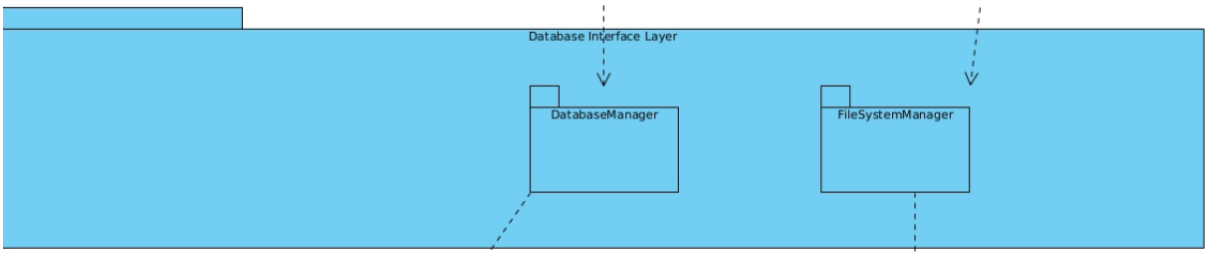
MessagingManager:

- The Messaging Manager is utilized to handle every issue related to the real-time messages between users.
- To operate properly, this package calls the Web Sockets Manager.

WebSocketsManager:

- The importance of this package is crucial in establishing a proper connection with the web sockets.
- This package contributes to sustaining robust functionality in operations that operate in real-time.
- Messaging Manager and Notification Manager are associated to this package in order to satisfy the real-time operating time.

3.3 Database Interface Layer



The database interface layer offers a simple interface to the packages in the application layer in order to read/write from or to the database and the filesystem.

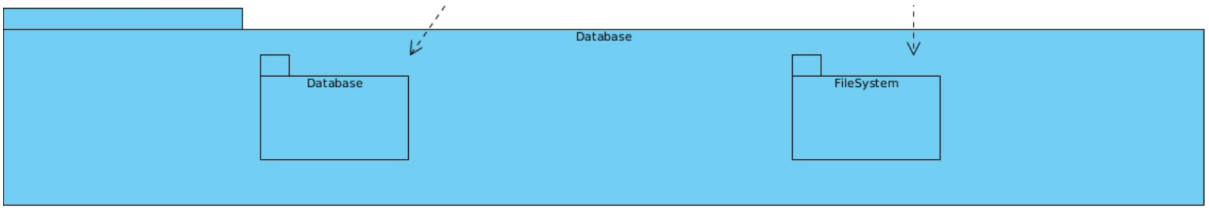
DatabaseManager:

- This package consists of the classes that handles fast and correct database connections.
- Every package except NetworkIPManager, WebSocketsManager and the FileManager in the application layer are associated to this package in order to access the database.

FileSystemManager:

- This package offers an interface for accessing the file system of the host environment in order to store/retrieve images of posts and users.
- The package FileManager in the application layer is associated to this package to handle file-related requests made by the presentation layer.

3.4 Database



The Data Layer houses the Database and the Filesystem, which is responsible for storing and managing the necessary information for the system. Reading and writing operations on the data occur within the Database and the Filesystem.

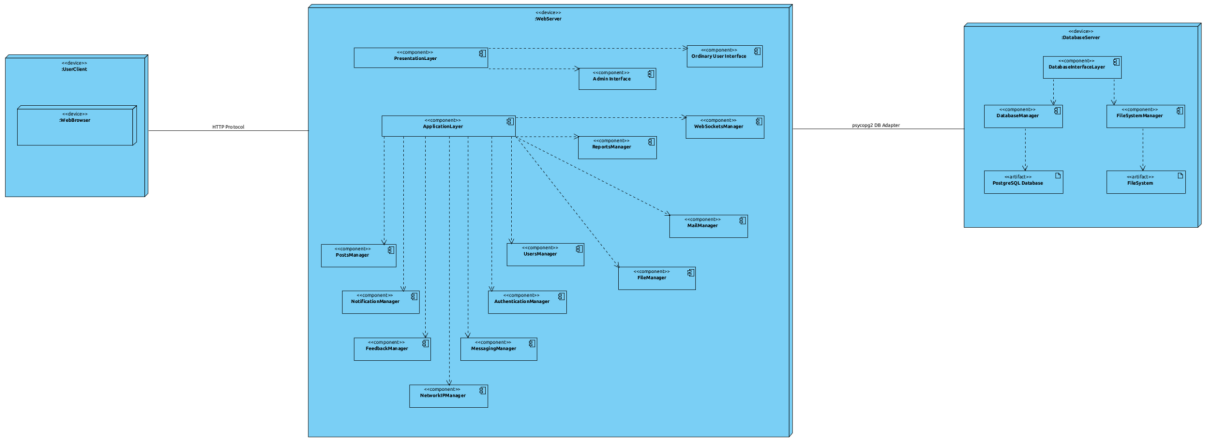
Database:

- This package helps the entities in the application layer read/write system data to the PostgreSQL database using the established connection in the Data Interface Layer.
- Thus the DatabaseManager package in the Data Interface Layer is associated to this class.
- In the existing version of our software, this package communicates with a local database. Yet, with the expansion of the application, there is a potential for this package to interface with a hosted database (e.g. Amazon RDS).

Filesystem:

- This package is responsible mainly for storing and accessing the files that are uploaded to the system by the users.
- Thus the FileSystemManager in the Data Interface Layer is associated to this class.
- In the present version of our software, this package engages with the local filesystem of the host environment. Nevertheless, as the application expands in scale, there is a possibility for this package to interface with a mounted Network Filesystem (NFS) to accommodate more extensive operations.

4. Deployment Diagram



This deployment diagram shows the relationship between the client, the web server, and the database server. Device nodes represent real or virtualized hardware, while the lines between them show how these device nodes are connected. The web server hosts the presentation and the application layers and is connected to the client node through the HTTP protocol. Similarly, the database server hosts the database interface layer and is connected to the web server node through the psycopg2 database adapter.

5. Hardware/Software Mapping

The front-end of the Campus Connect application is built using React.js framework. This decision is motivated by React's well-established efficiency in creating dynamic and responsive user interfaces. Its component-based architecture not only enhances code reusability but also contributes to a seamless and interactive user experience.

On the back-end, Campus Connect utilizes the capabilities of FastAPI. This modern and high-performance web framework is specifically designed for building APIs with Python 3.9+ and is widely recognized for its speed and user-friendly nature. It enables rapid development and deployment of complex applications, making it well-suited for handling the intricate interactions and data processing requirements of Campus Connect.

For managing the database operations of Campus Connect, PostgreSQL is employed. We chose PostgreSQL for its reputation for reliability, robustness, and performance, especially

when dealing with complex queries and large datasets. Its innovative features, such as full-text search and indexing, provide an efficient and secure means of managing the extensive data needs of Campus Connect.

Campus Connect is designed to be compatible with a variety of hardware, operating smoothly on any device that supports a modern web browser. This ensures accessibility for users across different platforms, including desktops, laptops, and mobile devices. While desktop users require standard peripherals such as a monitor and keyboard, mobile users can easily access the application on their devices on their web browser. The application is compatible with popular modern browsers such as Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, and Opera, guaranteeing a wide reach and user-friendly experience across different platforms and devices.

The development and local testing of the Campus Connect application are supported by a laptop with specific hardware configurations to ensure optimal performance. This laptop is equipped with 16 GB of RAM and an AMD Ryzen 7 processor, boasting 16 threads, which provides substantial processing power and efficient multitasking capabilities, crucial for the seamless operation of both front-end and back-end components. Additionally, the laptop features a 512 GB SSD, offering speedy and ample storage necessary for the app's development needs. These hardware specifications make the laptop an ideal environment for developing and running the Campus Connect app on a localhost, ensuring a smooth and efficient workflow.

6. Persistent Data Management

Our team at Campus Connect is not only focused on providing robust PostgreSQL database management and advanced session handling, but also on developing a sophisticated file system. This file system has been designed to efficiently manage and store various types of content generated on our platform, including user-uploaded images and multimedia files associated with different post types such as borrowing, donation, second-hand sales, and lost and found items.

To ensure quick access and secure storage of these files, the file system is being structured in a way that maintains data integrity and privacy. We are implementing strategies to categorize and index files in a manner that aligns with our database structure, enabling seamless integration between the file system and the database. This integration is crucial for ensuring fast and reliable file uploads and retrievals, thereby enhancing the overall user experience.

Our development and testing on a localhost environment also play a vital role in the development of the file system. This controlled setting allows us to rigorously test the file management processes, ensuring that the system is robust and capable of handling the expected load and variety of file types.

Considering the dynamic content of our platform and the need for real-time data reflection, the file system is being optimized for performance and scalability. It will be able to support the growing needs of the Bilkent community, accommodating an increasing volume of data as more students and staff engage with the platform.

Ultimately, our objective with this file system, in conjunction with our PostgreSQL database and tailored session management techniques, is to create a seamless, efficient, and secure environment for Campus Connect users. This will foster an engaging and interactive community experience for all Bilkent students and staff.

7. Access Control and Security

CampusConnect is a web-based program that involves trading between its users, therefore the security of personal information is important. Authorization is provided by the mechanisms called OAuth2 and OpenID. These are authentication and authorization protocols widely used for securing web APIs. OAuth2 allows a client (e.g., a third-party application) to access resources on behalf of a resource owner (e.g., a user) after authentication. OpenID Connect extends OAuth to provide user authentication and obtain information about the authenticated user.

Access Control is differentiated between general users and admins. On the client side, there are different user interfaces for each type of user. This ensures that the user will only use relevant functionality to its role. A user cannot request anything from the server which is not enabled for its role. As an example, an ordinary user cannot ban anybody from the program. This functionality is only accessible to the admins.

For the sake of security, login credentials are kept securely within the database after they have been hashed. Also, the layered architecture of the system ensures modifications on the database will only be made by our backend server.

Below is the access control matrix of the application. On this table, which actors have access to which methods is provided by the subsystems of the system.

Access Control Matrix

	Ordinary User	Admin
UsersManager	getBlockedUsers() blockUser(...) searchUser(...)	searchUser(...)
AuthenticationManager	login() register() authenticate(...) setTwoFactor(...) setCurrentUser(...) registerScreen() loginScreen()	login() register() authenticate(...) setTwoFactor(...) setCurrentUser(...) registerScreen() loginScreen()
MailManager	sendEmailTo(...)	sendEmailTo(...)
MessagingManager	getSrcUser(...) setSrcUser(...) getContent(...) setContent(...)	—
ReportsManager	createReport(...) setReason(...) setReported(...) report(...) getReportingUser(…) setReportingUser(…)	getReports() removeReport() getReason() getReportedID() getReportingUser(…) setReportingUser(…)
FeedbackManager	submitFeedback() getContent(...) setContent(...)	getSrcUser(...) getContent(...)
NotificationManager	getSrcUser(...) setSrcUser(...) getContent(...) setContent(...) setDestContent(…) getDestContent(...)	getSrcUser(...) setSrcUser(...) getContent(...) setContent(...) setDestContent(…) getDestContent(...)

FileManager	uploadImage(...) removeImage(...) listUploadedImages() updateImageMetadata() getUploadStatus()	uploadImage(...) removeImage(...) listUploadedImages() updateImageMetadata() getUploadStatus()
NetworkIPManager	checkIPAddress(...)	checkIPAddress(...)
WebSocketsManager	establishConnection(...) sendMessage(...) queueMessage(...) retryMessage(...) retrieveMessage(...) updateNotifications(...)	establishConnection(...) updateNotifications(...)
PostsManager	getPosts() addPost(...) deletePost(...) like(…) resolve(...) getContent(...) setContent(...) addComment(...) getComments(...) setDonationAim(...) getDonationAim(...) getMinDonation(...) setMinDonation(...) isResolved(...) getLFDate(...) setLFDate(...) getBasePrice() setBasePrice() setBidsEnabled(...) getAuctionDeadline(...) setAuctionDeadline(...) getAllBids(...) removeBid(...) getHighestBid() getBidPrice() setBidPrice(...)	getPosts() deletePost(…) getContent(...) setContent(...) setComment(…) getComments(...) getDonationAim(...) isResolved(..) getLFDate(...) getMinDonation(...) getBasePrice() getAuctionDeadline(...) getHighestBid() getAllBids(...) getBidPrice()

8. Boundary Conditions

8.1 Initialization

In order for users of the system to access the web application, initialized should be the web server that hosts the system and the PostgreSQL database. Since, in the current version of the system, the application is hosted on a localhost, there is no need for initializing an AWS or an Azure server. However, as the application scales up, we might choose to deploy our web application on a remote host.

Because our software is a web application, the users do not have specific requirements to initialize the application. The only exceptions to this are the following:

1. The users must have an account that they can log into.
2. The users must be connected to a Bilkent network.

In the case that a user does not have an account, they can choose the register with the system using their credentials, then login. However, there is no workaround to access the application if a user is not connected to a Bilkent network (except using Bilkent VPN service).

While the login operation is taking place, a unique access token is created for the user trying to log in. Once the user logs in, the relevant data is immediately fetched from the database in order to initialize the web application. Using the access token, the user can make requests to the web server through the presentation layer.

8.2 Termination

Termination of our software could either be intentional or unintentional. Intentional termination of the application can take place in a few scenarios. The first scenario is that when a user decides to logout of the system. As soon as the user logs out of the system, the access token is deleted. The second scenario is when the web server is shut down for maintenance. This is in the responsibility of the maintainers of the system, as the maintainers can choose to shut down the web server for short durations for maintenance. Unintentional terminations of the application stems from internal failures. In case of a failure-caused termination, the system attempts to save the last logged user data.

8.3 Failure

The root causes of the failure of the application include external and internal factors. While internal factors mostly refer to bugs and deficiencies in code, external factors encompasses a wide range of scenarios such as internet connection loss, web browser crashes, etc. In any case, the system attempts to save the last logged user data in data database before termination. The failure can be recovered by restarting the application afterwards.