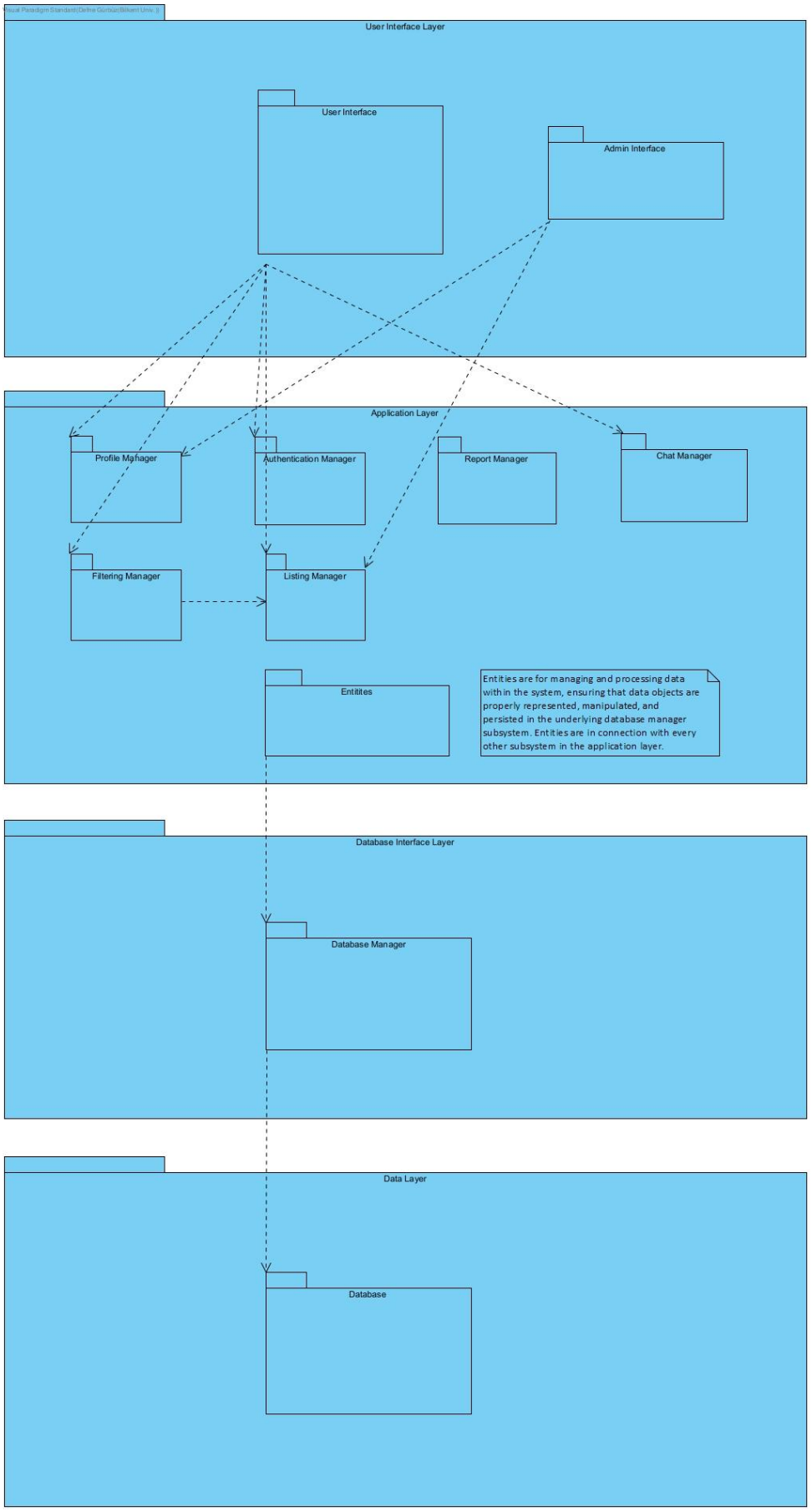
CS319 - Object Oriented Software Engineering
Project Design Report
Section 001
Team 1

Student Stash

Alp Batu Aksan - 22103246
Çağan Tuncer - 22102018
Defne Gürbüz - 22103295
İrem Hafızoğlu - 22101848
Ümmügülsüm Sümer - 22103772
Ghazal Amirrashed - 22001446

Contents

1. Subsystem Decomposition



User Interface Layer:

The user interface layer represents the interaction users have with screens. Users can enter information that would be passed into the application layer.

User Interface:

- This package consists of screens that users can view.
- This package is associated with the Profile Manager, Authentication Manager, Report Manager, Chat Manager, Filtering Manager, Listing Manager, and Admin Manager.

Admin Interface:

- This package consists of screens that users can view.
- This package is associated with the Listing Manager and Profile Manager**.**

Application Layer:

The application layer has control object packages that receive the pieces of information that were entered in the user interface layer and can process the retrieved data in this layer.

Profile Manager:

- This package handles the profile-related functionalities.
- This package is associated with User Interface and Admin Interface.

Authentication Manager:

- This package handles the authentication process.
- This package is associated with the User Interface.

Report Manager:

- This package handles the reporting process.
- This package is associated with the User Interface.

Chat Manager:

- This package handles the chatting process.
- This package is associated with the User Interface.

Filtering Manager:

- This package handles the filtering process.
- This package is associated with the User Interface.

Listing Manager:

- This package handles the Listing related functionalities.
- This package is associated with the User Interface and Admin Manager.

Entities:

- This package consists of all the objects in the app.

- This package is associated with every other package in Application Manager and Database Manager.

Database Interface Layer:

Application Layer and Database Layer have Database Interface Layer as a transactional managing layer between them.

Database Manager:

- This package handles the Database Management related functionalities.
- This package is associated with the Entities and Database.

Database Layer:

The Database Layer has the Database, which serves as the central repository responsible for storing data and preserving essential information for the system. Within this layer, the Database takes on the critical roles of reading and writing data. It functions as the primary custodian of the system's information, ensuring data integrity and accessibility as required by the application.

Database:

- This package represents the applications Database.
- This package is associated with the Database Manager.

2. Design Goals

2.1. Usability

The usability of Student Stash has been enhanced by providing a smooth user interface and transitions from one page to another. Users are automatically directed through every action they intend to perform. This automated navigation eliminates any potential confusion, providing a more intuitive experience. For instance, in notice creation action, Student Stash uses a step-by-step approach to the action, without any need for explicit instructions.

The web application will be cultivated by media elements of CSS to be compatible with all kinds of browsers, both in mobile and desktop environments. It will be enhanced not to collapse when switched to mobile mode. Most web applications are not responsive in mobile mode, making quick access nearly impossible or very difficult. Student Stash will try to solve this issue by implementing media-responsive elements in the stylesheet.

The utmost goal of Student Stash is to provide a safe and secure environment for the users to run transactions. In an environment where the user's possessions and money are the means of communication, the security of the system must be a high priority. This issue has been settled by admin intervention, aiding the users with facilities such as report and block

options. In the presence of any potential threat or discomfort, the users can report the issue to the admins. Including whenever the users feel their belongings and money are at stake, etc. Moreover, the chat mechanism, which is the mediator between the provider and the beneficiary, has been secured to the point that no third party has access to it.

2.2. Maintainability

StudentStash aims to provide a service that is going to solve a big problem for Bilkent students. The app should remain up and running for every new year of education. Because of this, proper maintenance of the app is critical to ensure that new students are included in this service. Also, as semesters pass, the curriculums of lessons are changing and with it new materials for lessons are demanded from the students. Maintaining the app according to these changes by adding new tags is critical to create the best user experience. Therefore, the tags are planned to be made in a way that it can be easily changed and revised. Also, with user feedback taken into consideration, regular patches with bug fixes and demanded new features will help StudentStash stay in relevancy. To achieve maintainability we used inheritance. When modifications or updates are necessary, changes made in a parent or base class automatically propagate to its derived classes, ensuring consistency throughout the system. This inherent relationship between classes through inheritance fosters a more manageable codebase, reducing redundancy and enabling efficient updates across interconnected components, thus bolstering the system's overall maintainability.

2.3. Trade-offs

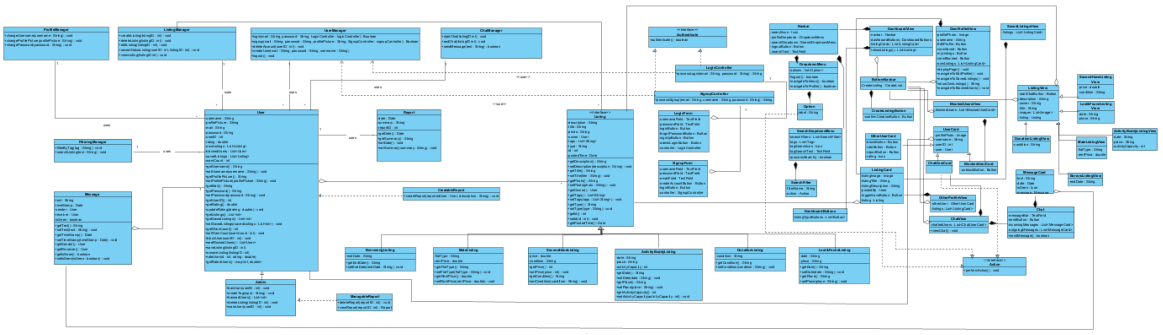
- **Usability vs Functionality**

StudentStash prioritizes usability over functionality. This approach aims to ensure a seamless and intuitive user experience, emphasizing smooth transitions and user-guided interactions within the platform. Consequently, this emphasis on usability might entail a slightly reduced emphasis on incorporating an exhaustive array of functionalities. However, by placing usability at the forefront, StudentStash aims to create a platform that is easily navigable and user-friendly, streamlining transactions and interactions within a secure environment.

- **Maintainability vs Performance**

StudentStash prioritizes maintainability over performance gains. This choice leans towards a design and structure that might sacrifice some immediate performance optimizations in exchange for a system that's easily maintainable and adaptable.

3. Class Diagram

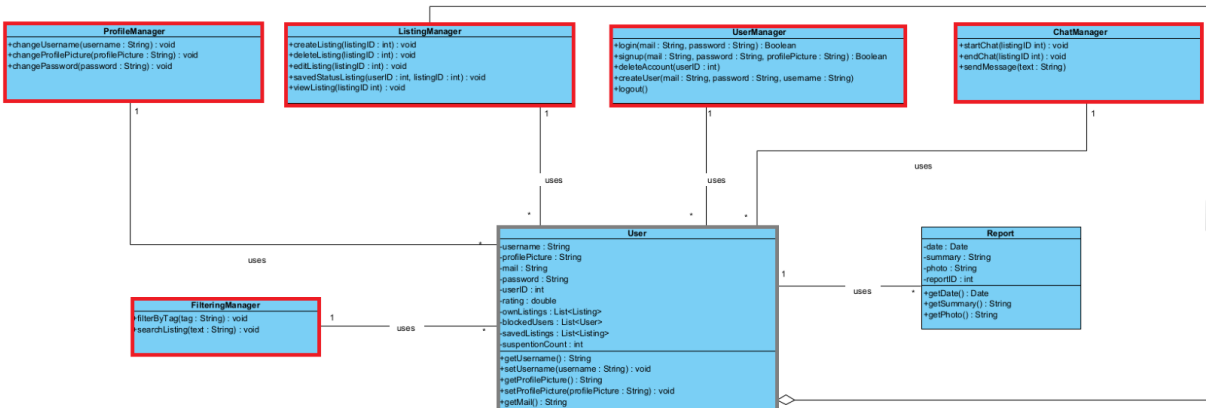


The .vpp file of the Class Diagram is on the GitHub repository, named class_diagram_final_submission.vpp.

4. Design Patterns

4.1. Singleton Pattern

While developing our web app, we chose to use singleton design pattern for our manager classes which are ProfileManager, ListingManager, FilteringManager, UserManager, and ChatManager. We made this decision to ensure that each of these manager classes has only one instance in throughout the application's lifecycle. Our aim was to have more control and management in our application and avoid unwanted creation of new objects. This decision also serves the purpose of having more united and consistent state in between the components of Student Stash. We will have better communication between different compounds of the application along with better overall performance. The singleton pattern, in this context, played a crucial role in a well-organized and scalable architecture, contributing to the overall robustness and maintainability of the application.



4.2. Strategy Pattern

In our web-application, we employed the strategy pattern within the listing interface, which encompasses the following listings; borrowing listing, mate listing, secondhand listing, activity-buddy listing, donation listing, and lost & found listing. So, when the users want to create different types of listings, the specific strategy class will be used for creation. With this design pattern, each listing type is defined independently, having a flexible and modular design. By encapsulating each listing type as a separate strategy class, we ensured that any changes or additions to the behavior of one type would not affect the others. It also supports code reuse, as common functionalities can be implemented in a base strategy interface shared among all listing types. This design choice makes the system extensible without requiring modification to existing code making it scalable architecture for the application.