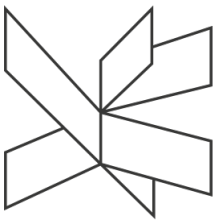
Employee Management System



VIA University College

By

Philip Philev

293735

Levente Nagy

293115

Audrius Sauciunas

293156

Gustaw Longfors

293107

Ioana Grigore

293088

Supervisors

Steffen Vissing Andresen

Ole Hougaard

Software Technology Engineering

2st Semester

Horsens, Denmark

June 2020

This report is dedicated to *Everyone reading*

Number of Pages: 46

Abstract

The purpose of this report is to document the development of the communications system. The system was developed as an idea to enhance a company's efficiency in the field of communication. The focus was directed towards creating a way for employees to communicate with each other digitally and assist coherent workflow.

The documentation presents all stages of system development. The functionality of the system was decided after analysing the main problems a company can face in the field of communication, followed by design and implementation of the system.

The software was developed by using Java and JavaFX for UI, following the SCRUM framework and the database was created using a relational database management system PostgreSQL.

The result of the project is a communications system that is made from three parts: server, client, and database, following SOLID principles. The most important classes have been tested using JUnit.

Contents

List of Figures

Chapter 1

Introduction

Software is an integral part of every large modern company, be it, for communication, storage of employee records or certain functionality depending on the field of operation. It's cost varies depending on the number of features present, and it increases significantly if developed for a specific company.

Nevertheless, many small and medium businesses, to this day, utilize hand-written records and use phone calls or third-party chatting applications to manage and maintain relationships with employees. This can hinder productivity and growth, as efficient and clear communication alongside digitalized employee records allows teams to focus on work and not on how to work.

Furthermore, with software including communication functionality, employees can maintain productivity for longer, since applications used for communication such as Facebook have an inherent risk of unnecessary distractions and time consumption. Additionally, an integrated chat could allow employees to exchange only work related information and not search through lengthy personal conversations.

Having an integrated application with functionality that can easily be extended, can lead to accelerated growth and focus on work, rather than process, in starting and developing companies. It can save time by integrating internal communications, while also offering the ability to easily manage employees with temporary or starting status. Finally, if the company specializes in certain area of expertise, functionality reflecting the area can be included in order to stimulate efficiency and reduce external distractions.

Chapter 2

Analysis

The analysis part contains the requirements, both functional and non-functional, use case diagrams and use case descriptions. It also includes the domain model, elaborated after the case.

2.1 Functional requirements

Critical priority:

1. As an admin I would like to access and edit all Employees' data
2. As an admin I would like to access and edit all Departments' data
3. As a user I would like to access a chat room so that I can communicate with other employees in real-time

High priority:

4. As an employee I would like to have my own account, so that I could avoid identity theft
5. As a user I can send private messages to other employees so that we could communicate privately

Medium Priority:

6. As a user I would like to get notifications when other employees/departments request for help so that I know plan my work ahead of time
7. As an employee I would like to be able to create an urgent job listing so that other department employees would immediately be notified

8. As an employee I would like to have a local to do list to keep track of my progress

Low Priority:

9. As an employee I would like to be able to create an urgent job listing so that other department employees would immediately be notified.
10. As a manager I would like to send system-wide notifications to either a department or all departments
11. As a user I would like to get notifications when other employees/departments request for help so that I know plan my work ahead of time. .

2.2 Non-Functional Requirements

1. The system must support Windows 10
2. The system must use PostgreSQL database management system.
3. The system must have an authorization element, to make sure employees can only access data which they are entitled to.

2.3 Use Case Diagram

The use case diagram is based on the requirements mentioned above. The communication system is multi-user, and it has three actors: employee, manager and admin. The admin has access to most of the functionalities. The diagram represents what functionalities the actors can use.

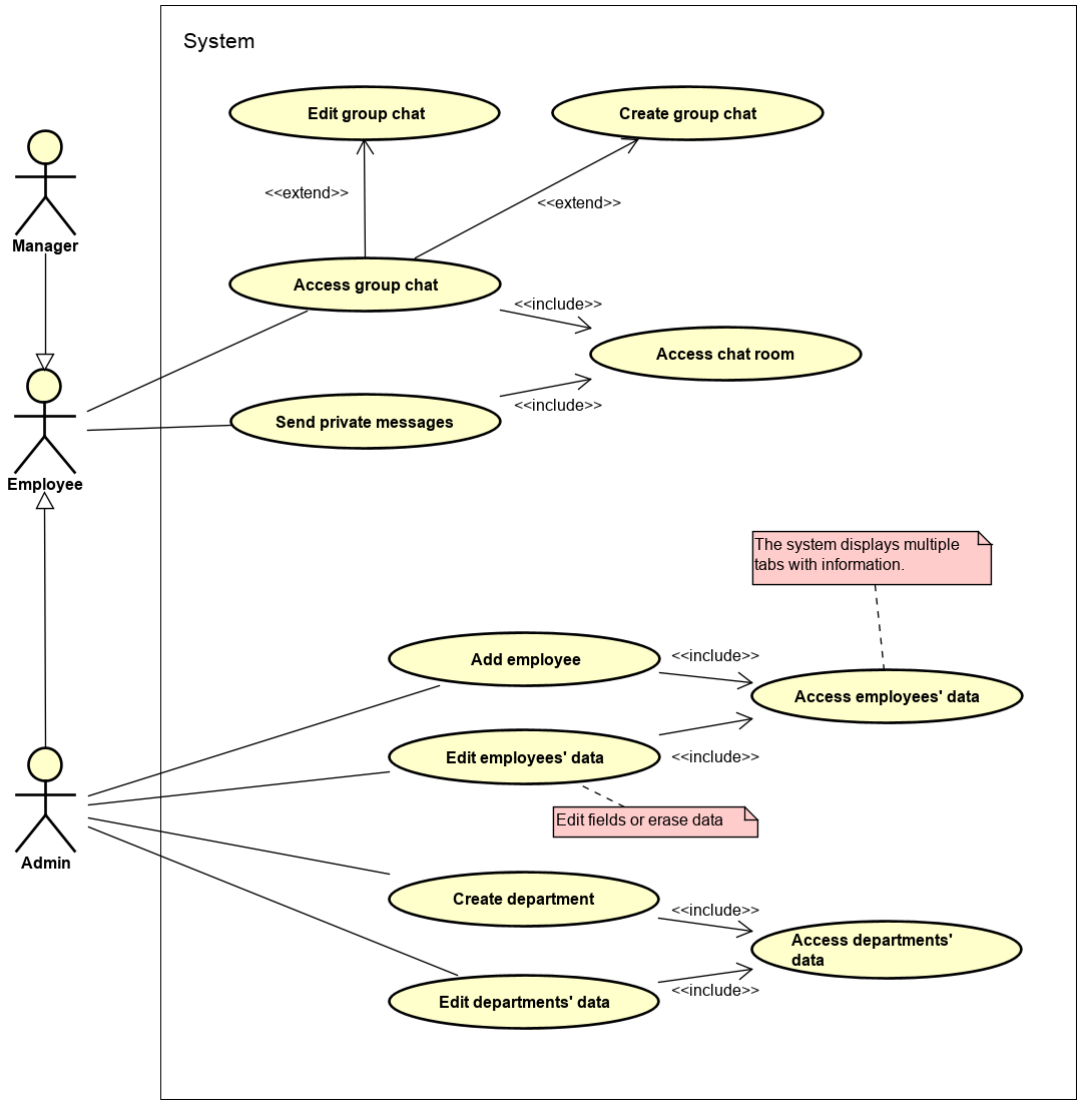


Figure 2.1: Use Case Diagram

There are use case description corresponding to each use case. The use case description portrays each functionality and indicates the steps the user goes through while using the system. Every use case description also has an activity diagram which represents the flow of activities during the process.

Each functionality is shown in a use case description and presents an explanation of the process of information flow as well as the logical steps a user must take to accomplish the given use case result, or an alternative. In Figures 2.2, 2.3 and 2.4 are the more important Use Case descriptions.

ITEM	VALUE
UseCase	Edit employees' data
Summary	The administrator edit's an employee's data
Actor	Admin
Precondition	The Admin must be logged.
Postcondition	The employee's information is changed.
Base Sequence	1) The Admin clicks on the "Administrative" button. 2) The Admin clicks the "Employees" button. 3) The Admin selects an employee. 4) The Admin clicks the "Edit" button. 5) The Admin fills in the fields. 6) The Admin clicks "Save". 7) The employee's information is edited.
Branch Sequence	
Exception Sequence	*While in step 5) the actor can use the "Cancel" button. 1) The actor clicks "Cancel". 2) The actor goes back to step 3).
Sub UseCase	
Note	User Story 19, 15

Figure 2.2: Employee Use Case description

ITEM	VALUE
UseCase	Edit departments' data
Summary	An administrator edits a department.
Actor	Admin
Precondition	The administrator is logged in
Postcondition	The department is edited.
Base Sequence	1) The Admin clicks on the "Administrative" button. 2) The Admin clicks the "Departments" button. 3) The Admin selects a department. 4) The Admin clicks the "Edit" button. 5) The Admin fills in the fields. 6) The Admin clicks "Save". 7) The deparment is edited.
Branch Sequence	*While in step 5) the actor can use the "Cancel" button. 1) The actor clicks "Cancel". 2) The actor goes back to step 3).
Exception Sequence	*While in step 5) the actor can use the "Cancel" button. 1) The actor clicks "Cancel". 2) The actor goes back to step 3).
Sub UseCase	
Note	User Story 20

Figure 2.3: Department Use Case description

ITEM	VALUE
UseCase	Edit group chat
Summary	An Employee edits a groupchat
Actor	
Precondition	The actor must be logged in.
Postcondition	The group chat is edited.
Base Sequence	1) The employee clicks on the "Chat" button. 2) The employee clicks the "Group" button. 3) The employee selects a group chat. 4) The employee clicks the "Edit" button. 5) The employeer fills in the fields. 6) The employee clicks "Save". 7) The group chat is edited.
Branch Sequence	
Exception Sequence	*While in step 5) the employee can use the "Cancel" button. 1) The employee clicks "Cancel". 2) The employee goes back to step 3).
Sub UseCase	
Note	User Story 2

Figure 2.4: Group Chat Use Case description

Additionally, each use case has an activity diagram. They are a graphical explanation of our use case descriptions, and as such show the flow of the task as well as all possible decisions the actor can take. They are Figures 2.5, 2.6, 2.7.

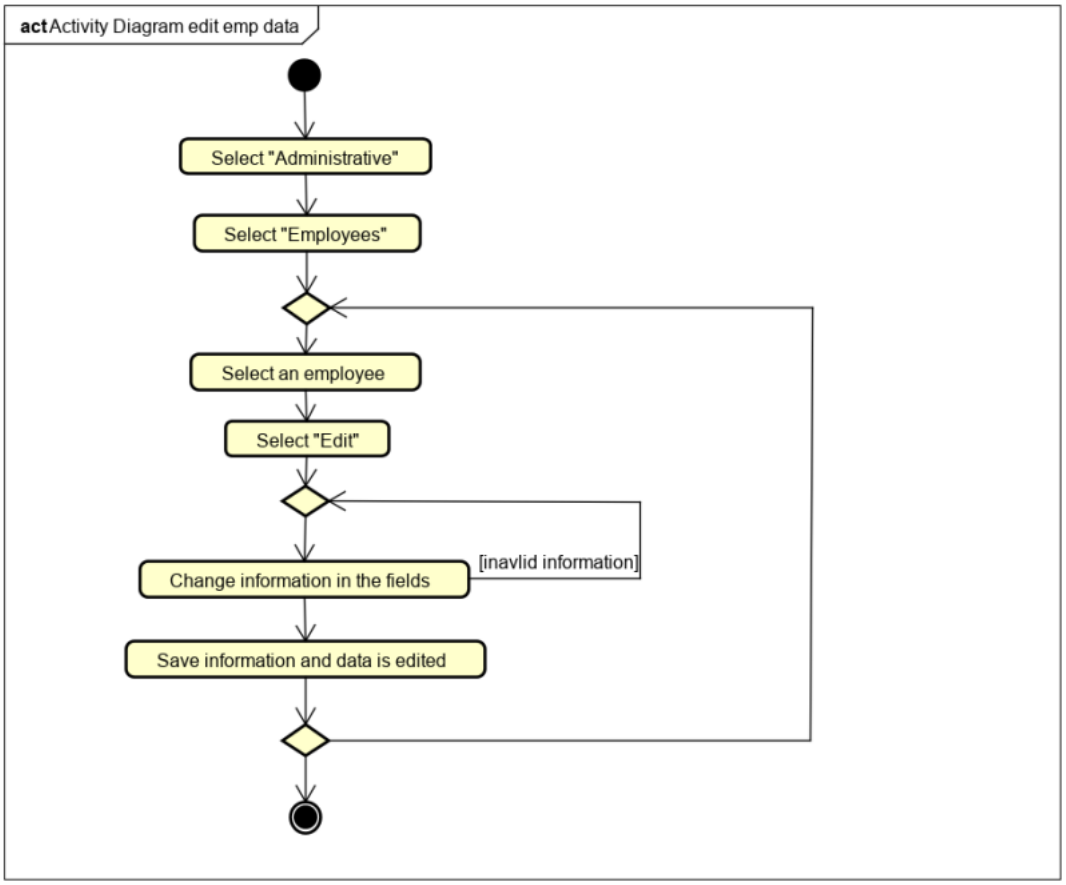


Figure 2.5: Activity Diagram for Edit Employee Data

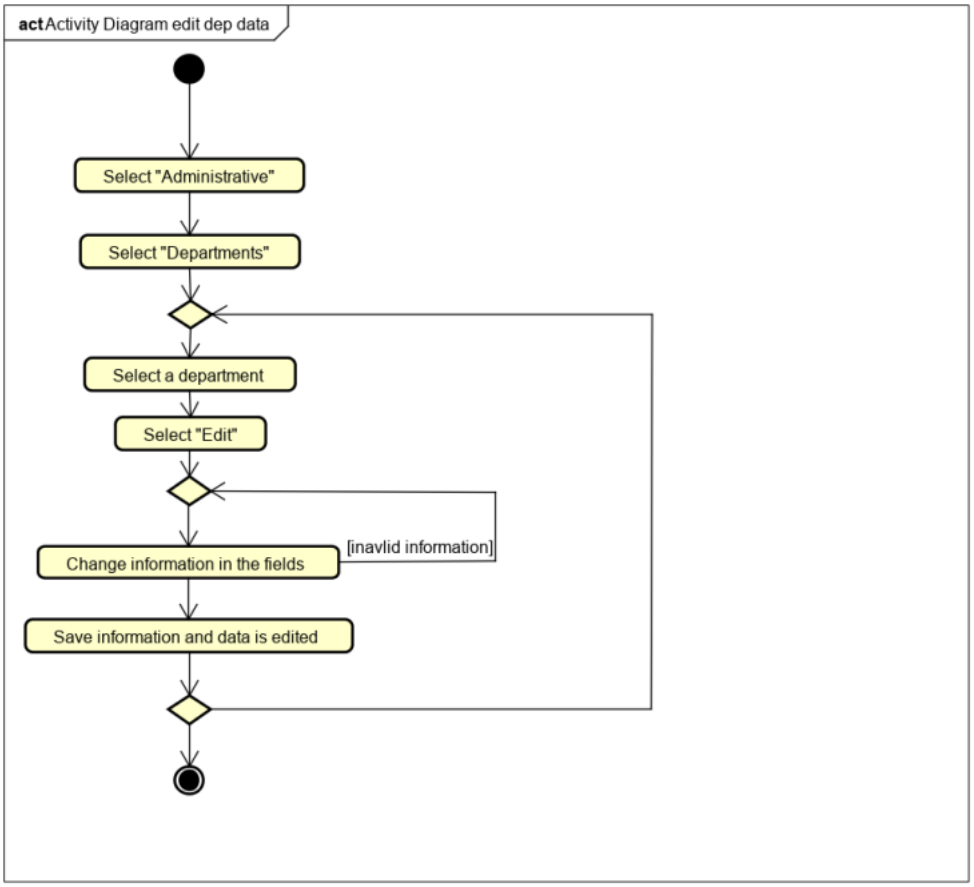


Figure 2.6: Activity Diagram for Edit Department Data

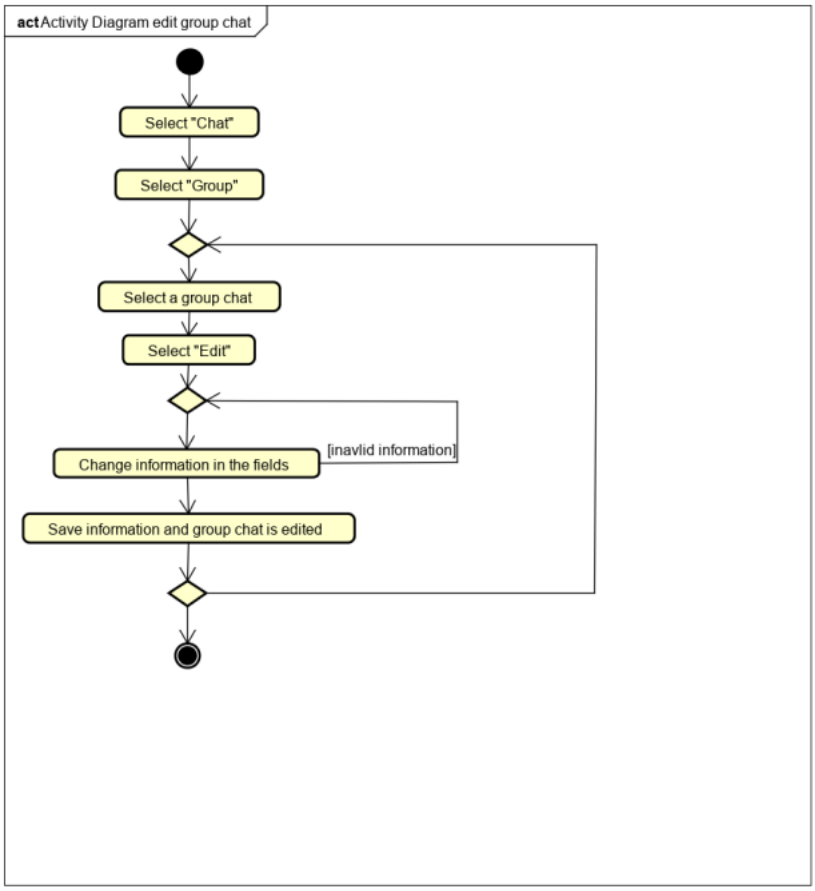


Figure 2.7: Activity Diagram for Edit Group Chat

2.4 Domain Model

The last part of the analysis concerns the domain model based on the requirements, use case descriptions and activity diagrams. It provides an overview of the classes and the relationships between them. The domain model can be seen in Figure 2.8.

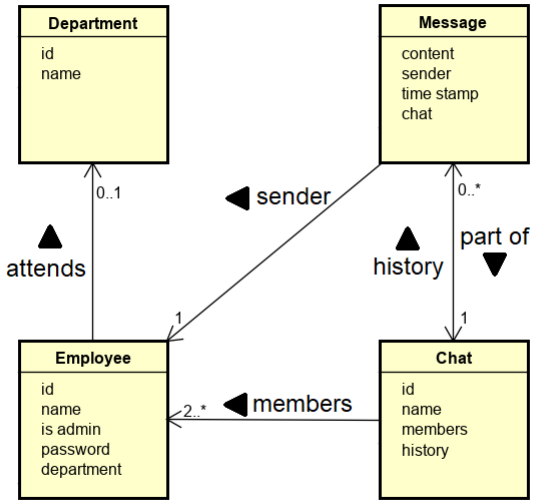


Figure 2.8: Domain Model

It is clearly visible that the employee and department objects aim to satisfy the product backlog item 18 and 19. The relation between them is quite simple, an employee has a department which he belongs to, but employees can also be a separate entity.

The chat and message objects aim to satisfy items 7 and 2. These two items have no logical separation because a private chat (item 7) can be perceived as a group chat of 2 people (item 2). The relation between these objects are more complex, the chat has 2 or more than 2 employees as its members, and it has messages as part of the chat's history. A message has one employee who is the sender of that message and a chat which it is a part of.

The id attribute of the Employee, Department and Chat are required because 2 of each of these instances can have the same name or other attributes, but a differentiate between these instances is necessary, thus the unique id is introduced.

Chapter 3

Design

3.1 Server

The system design started with focus on items 18 and 19 from the Product backlog. A strong base had to be designed, as the above-mentioned items are the foundation and backbone for most of the system. For example, login, chats and messages, all rely on employees (such as their name etc.).

3.1.1 Foundation

Designing a strong foundation was crucial. The main points of the initial design were:

1. What data must be stored
2. How can the data be made easily and safely editable?
    (a) Integrity of the data
    (b) Data loss caused by simultaneous editing
3. Security
    (a) Who can read certain data
    (b) Who can write certain data

3.1.2 Stored Data

In the first iteration, a sketch was made of the mandatory data based on the domain model and logical design choices.

For employees:

- Unique employee ID
- Boolean value describing an employee's admin privileges
- Name of the employee
- Password of the employee
- A department ID which the employee is part of

For departments:

- Unique department ID
- Name of the department

3.1.3 Data Integrity

It was determined that removing an employee can cause problems with the data's integrity, such as, null pointers for the employee's attributes. This could occur when an employee object is part of other elements like chats. To solve the problem a boolean attribute to the employee class must be introduced in order to distinguish whether the employee is archived or not, allowing for a reference to the archived employee.

Removed departments can cause similar problems. Nevertheless, it could be possible for departments to exist without employees, thus setting the reference to null was a certain design choice.

Data loss

Since the system is network-based, and all admins are going to have editing permission concerning employees and departments a problem can arise with simultaneous editing of data. This can results in unintentional data loss and code anomalies. An example could be when a user with admin authorization attempts to edit an employee's details, while a second user is accessing the same employee. The users clients will be storing cached values of respective attributes previously provided from the server, however when both submit the changes, it might be hard to determine which submission is valid or has priority, thus possible data corruption can occur.

A possible solution is the introduction of two transaction layers, which are as follows:

1. Editing employee records

2. Editing department records

The networking is Sockets based and the main backbone is the Server class. It handles the clients and opens separate sessions on new threads for each incoming connection, resulting in unique session identifiers to be use for the transaction layers.

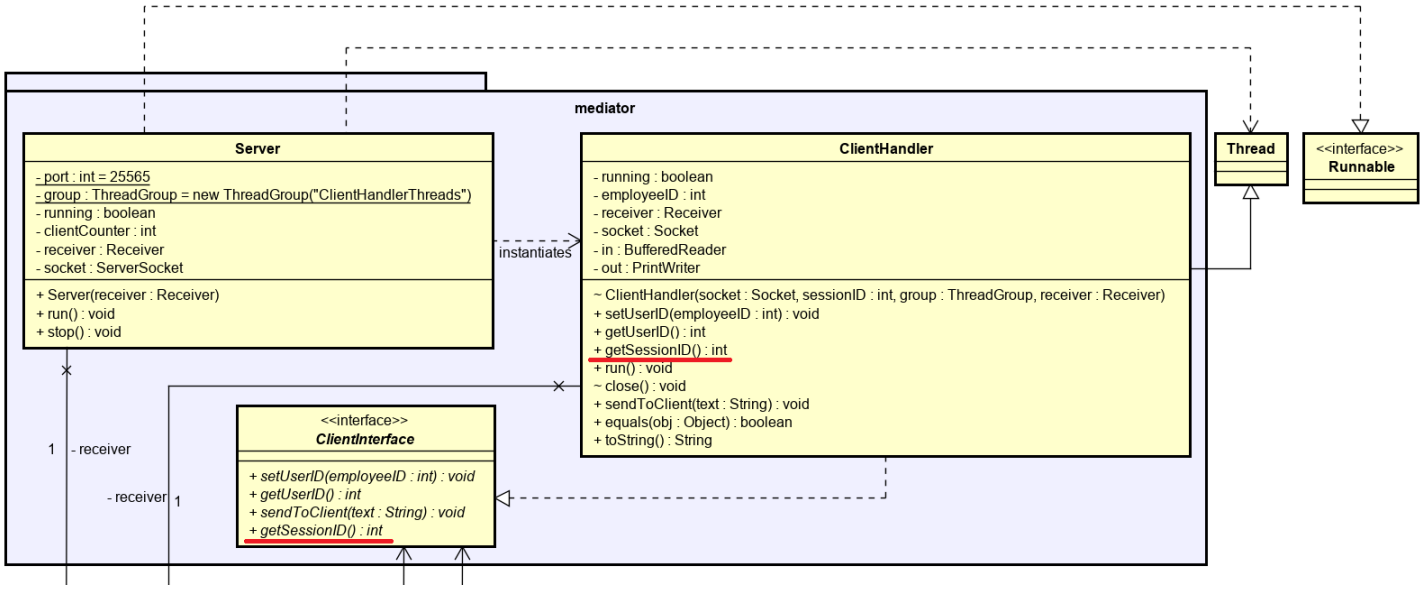


Figure 3.1: Server Mediator

To expand on the above definition, both of the transaction layers will have a record of their specific locking and editing queue system, based on a modified reader-writer lock pattern. Firstly, each lock instance uses the unique session identifier in contrast to an employee or department identifiers, thus making it possible for admin users to open multiple sessions simultaneously on their account without overloading the locking mechanism. This design choice, avoiding separate lock identifiers has certain advantages and disadvantages.

The main disadvantage is the lack of sub-locks on certain elements(avoiding overengineering). This is addressed by eliminating the need for sub-locks, achieved by discarding methods that would otherwise allow for the editing of department's employees while simultaneously having the ability to edit an individual employee's department. This would require sub-locking for each employee in a department when editing the department itself or sub-locking the department when editing an employee that belongs to that department.

The main advantage of this design approach is the offered simplicity. It allows for easier architecture design and later implementation, while also reducing the necessary computational power. Furthermore,

the waiting time for locks only includes the sessions in queue for editing concerning that specific instance, thus not locking all employees that belong to a department while changes are being made to the department itself.

Part of the class diagram which contains the department's readers-writers lock pattern is shown in **Figure 3.2**.

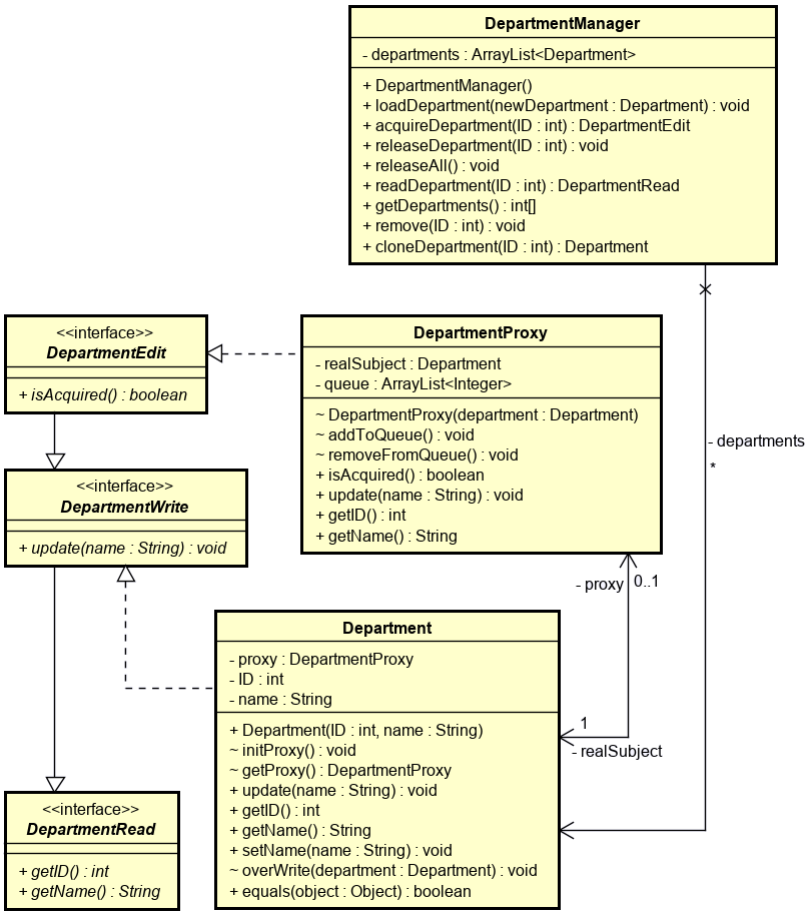


Figure 3.2: Departments modules

A further modification of the pattern is removing the need to lock for readers, achieved by the mediator's design. Since Java has a garbage collector instead of a manually defined destructor, every instance of an object is preserved and stored in memory until a pointer to that object exists. Because of that, if f.x an instance of a department object is read by a user, and an admin deletes that department while the user has an active pointer to it, he or she will still be able to view the deleted department.

Under those circumstances, data consistency across the Clients and the Server are achieved by the business logic only being present in the server, which lowers the possibility for errors. This results in each client only storing a cached version of the data (cached on login). Every time a client makes a change,

a request will be send to the server and it's business layer carries out the request. If the modification is approved and made, it notifies every client (including the sender) of the changes, preserving the data consistency across all clients. In theory this makes the reader locking obsolete, as there are no scenarios where someone could access an invalid instance of an object.

3.1.4 Reading right

The security features of the system and the access to the data is fully handled by the mediator on the server side, rather than having a data request-reply solution. On login the server sends all the current information to the issued client. While logged, as previously described, every approved change is broadcast to all involved clients.

Writing Rights

The writing right in a client are disabled depending on whether the user has admin privileges. This however, does not make admin requests impossible, f.x a tampered or a bugged client after bad deployment. Therefore, every editing request is evaluated in the server's respective request processing part, and if not approved changes would not be made.

3.1.5 Session timeout

Addressing another important feature regarding questions such as "How long can a lock be held?" is done by introducing a session timeout. If a client has not had interaction with the server for a certain time, the session timout transpires thus solving problems such as forgetting to log out and exposing company's data to manipulation.

3.1.6 Chat components

The chat system is a feature allowing for internal company communication. Firstly, the differentiation regarding private/group chats, as discussed in analysis, can be perceived similarly since a private chats are technically group chats with two members.

Nevertheless, boolean attributes need to be introduced to separate between both, since a group chat can have 2 members as well, making it impossible to distinguish between a chats identity without this attribute. Regarding the interface segregation and the single responsibility principle, this solution does not violate either, as there is no practical difference between the two, therefore it does not implement

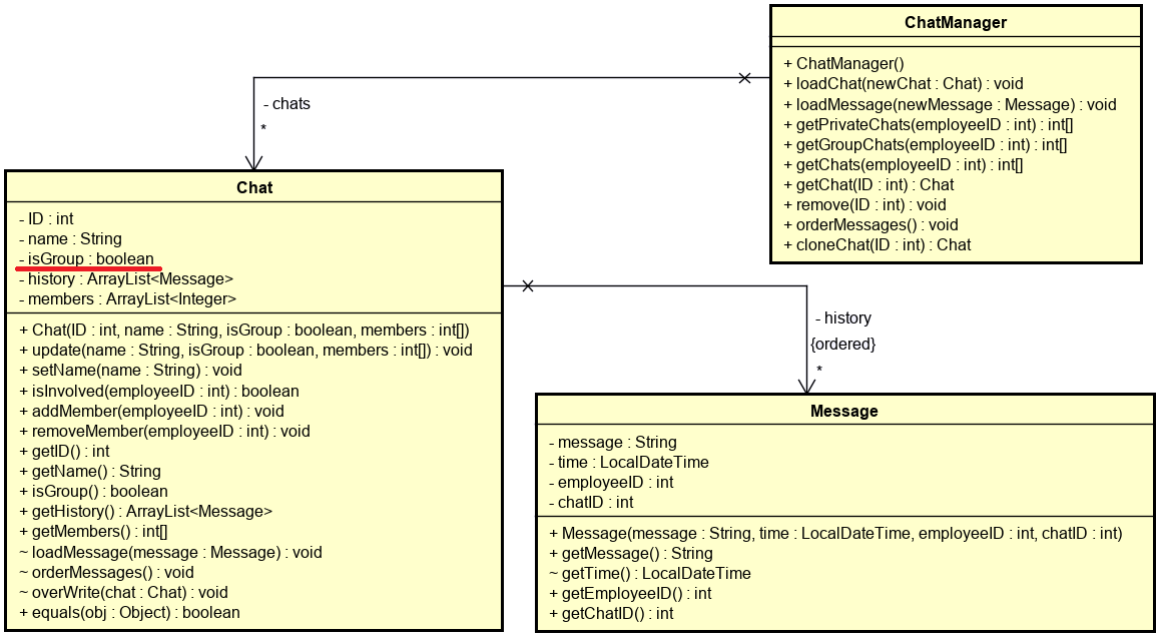


Figure 3.3: Chat module

methods not in use.

3.1.7 Database

The last subtopic, before discussing problems regarding the entity of the server module, is the persistence layer. PostgreSQL is used for the persistence and the EER diagram in **Figure** 3.4 was established based on the previous parts.

The non-obvious part from the diagram concerns the **EmpConnection** and the **MessagingGroup** table. Since the database needs to be normalized, the members of a chat cannot be stored in the **MessagingGroup** table since a chat has an undefined number of members. This led to the realization that no estimated number of fields matching the number of members can be achieved, thus more than one member per field would violate the 1NF. An N to M connection is thus required, leading to the creation of the **EmpConnection** table.

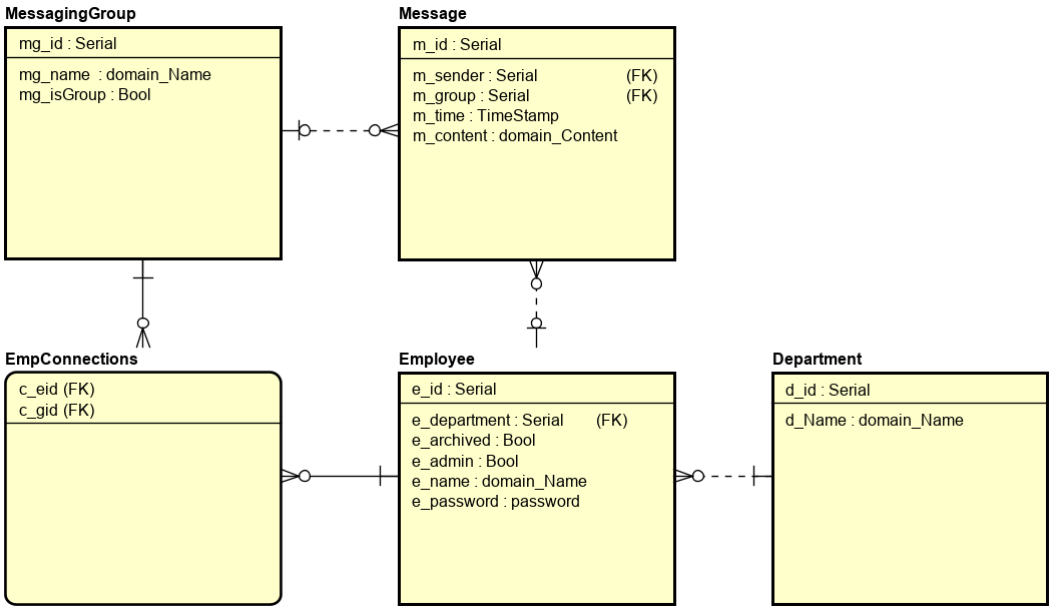
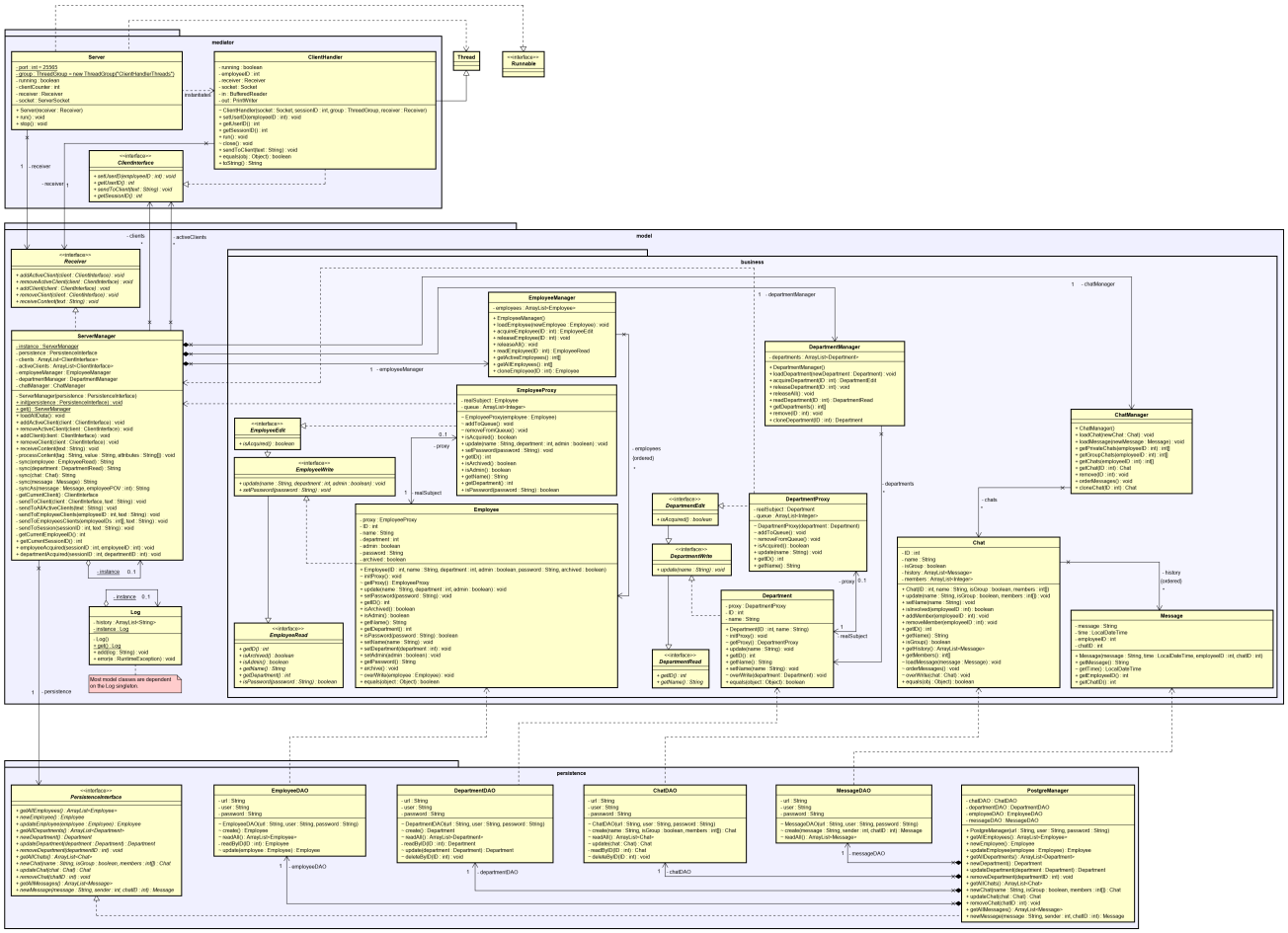


Figure 3.4: EER Diagram

3.1.8 Conclusion and continuation



The given server module is representative of the system. This allows for the inspection of some of the used **design patterns** and examples of the **S.O.L.I.D.** principles.

The **Layered Architecture Pattern** is the first relevant pattern, as the separations are clearly visible in the packages.

Between the layers an example of the **Dependency Inversion Principle** can be seen. The mediator layer only depends on the Receiver interface and the model only depends on the Client Interface. Thus, the mediator layer can be substituted easily if the Sockets need to be replaced.

This is also true for the persistence layer since the model is only dependent on the PersistenceInterface. It easily allows for a change in the persistence, not limiting the choice to PostgreSQL.

A well-marked example of the **Interface Segregation Principle** is the separation between the DepartmentEdit and DepartmentWrite interfaces (same for EmployeeEdit and EmployeeWrite). These pairs weren't combined as the **isAcquired()** method is only used by the proxy.

Many examples of the **Open-Closed Principle** can be found. If any building block is taken from the domain model and its representative objects examined, like in Employee, it can be seen that certain attributes can easily be extended without the need of modifying already existing code's. F.X if a social security number attribute is needed in the Employee class, it can be done effectively without logical rearrangement and refractoring(although some code (such as the constructor) has to be modified)

The **Single Responsibility Principle** is also present. The most evident representations are inside the persistence layer, regarding the PostgreManager and the DAOs, since the DAO classes are composites of the PostgreManager and only responsible for data manipulations and querying the database. They could have been easily included into the PostgreManager, however this would have removed the logical separation, thus being simultaneously responsible for managing Employee, Department, Chat and other message related database orations.

The patterns utilized are: **Singleton** for the Log and ServerManager, **Readers-Writers** Locking pattern using a **Smart-Protection Proxy** for the Employee and Department handling, and **Layered Architecture** pattern for the overall design.

3.2 Client

3.2.1 Foundation

The client's role is straightforward due to the design of the server. A design choice was made for the client to not handle business logic, thus, simplifying its role to making appropriately formatted requests towards the server, based on the client's UI interactions. It receives instructions from the server and acts accordingly, therefore, the received information and relations between each domain object are similar to the server's.

Stored data compared to the server

- The client does not (also true for backend) fetch the passwords of the employees.
- The client does not store archived employees.
- The client only stores messages as a Strings in their chronological order, premade by the server to match it with the issued client's point of view. (meaning messages sent by you appear as "YOU: hi" and messages sent by others "Name: hello".

3.2.2 Client navigation

After establishing the state diagram in **Figure** 3.5 regarding how the client reaches certain functions, the individual views were a logical continuation.

The light and dark blue states are representing one view, while the light blue is a sub-state of the other one. It logically follows that certain actions can only be reached from these sub-states, for example, if an explicit employee has not been selected from the existing employees list, the edit and archive functions are unreachable.

3.2.3 Problems and challenges

Not many problems were encountered while designing the client. One intriguing design choice was the **DontDestroyOnLoad** singleton used in the viewmodel. Its origin comes from the issue that information needs to be transferred between different viewmodel instances when switching views. A possible solution is to make the ViewModelFactory-ViewModels association navigable both ways and store information in the factory. This, however, would have violated the Single Responsibility Principle, thus the introduction of a new singleton class with a single purpose of storing currently selected information.

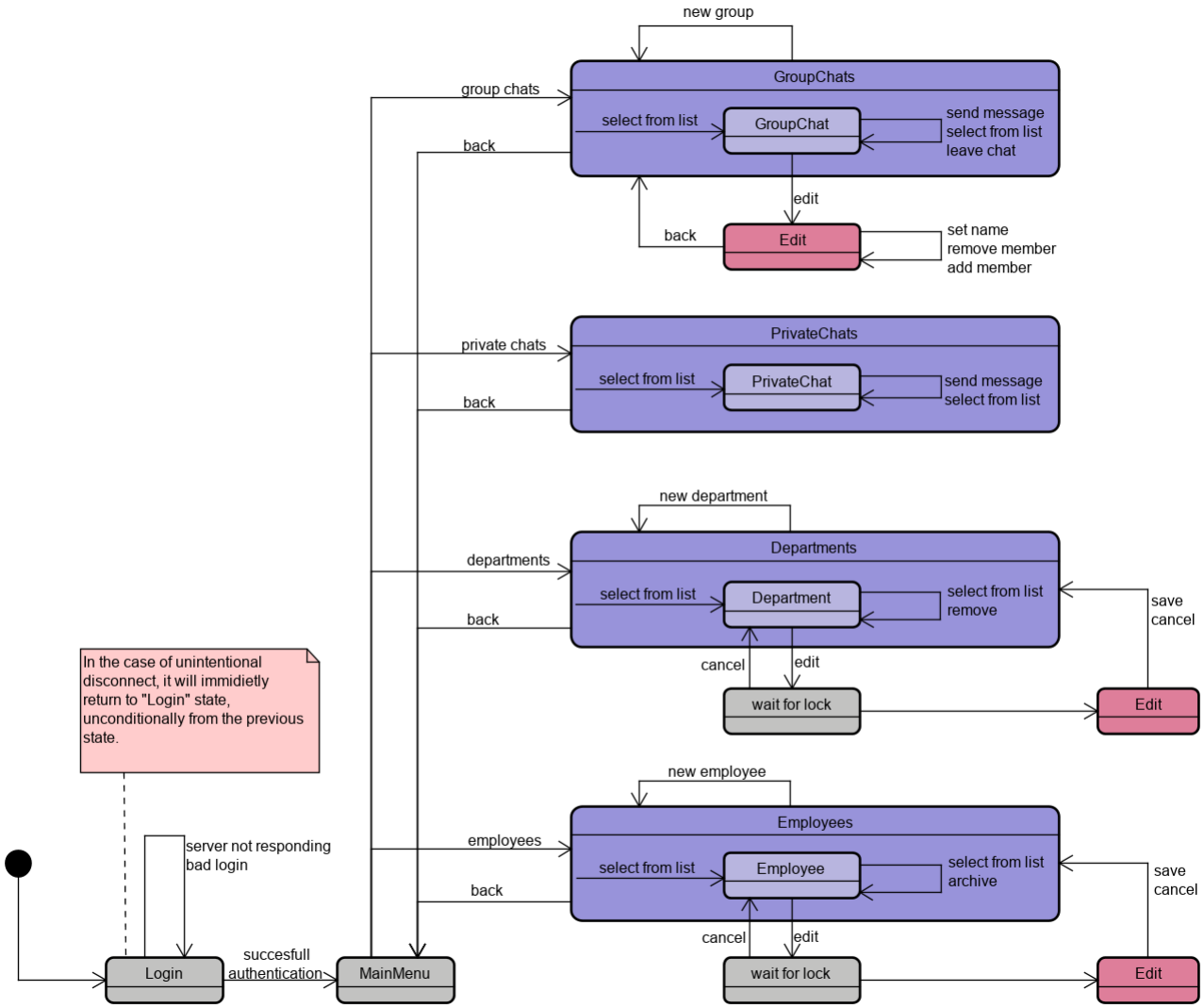


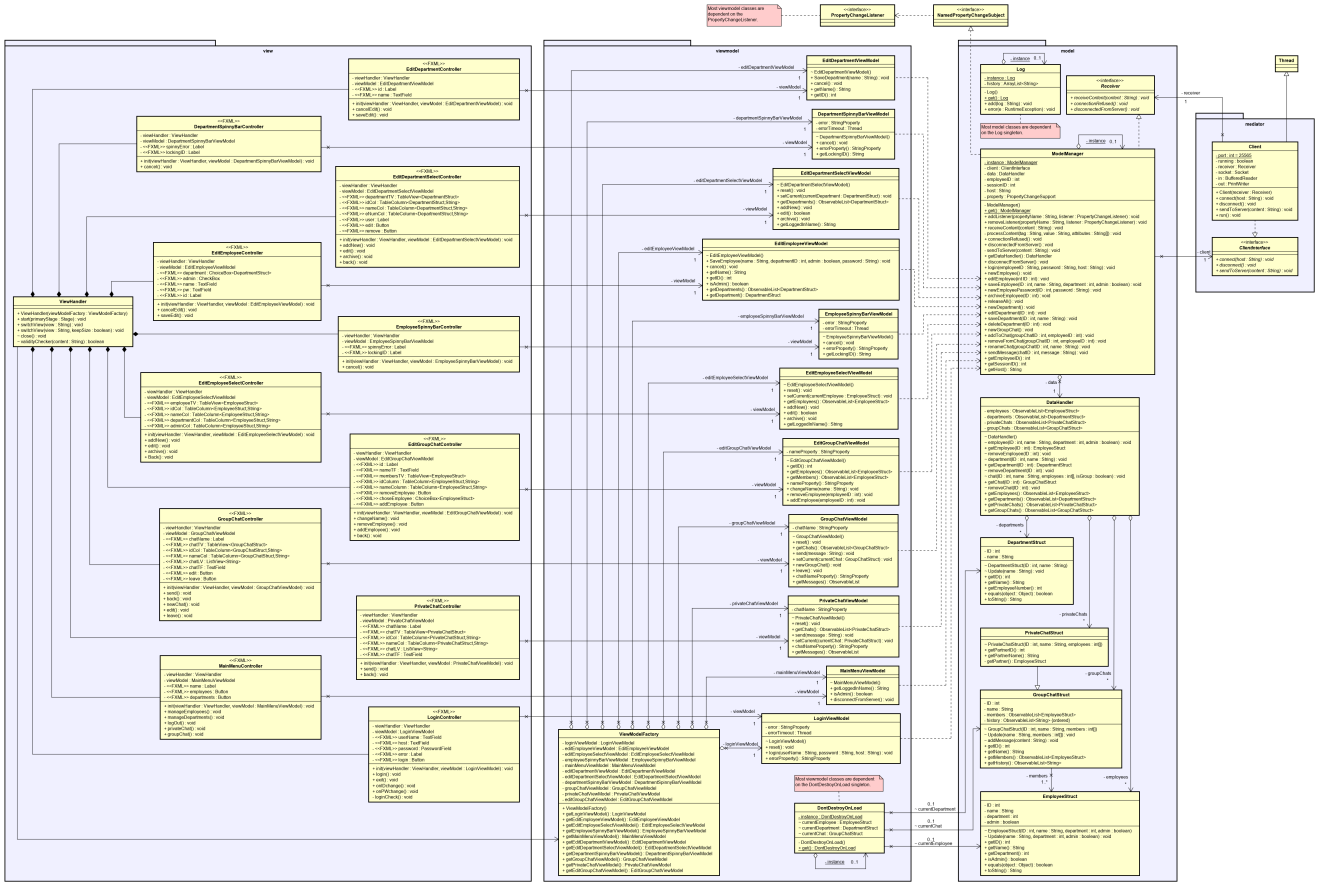
Figure 3.5: State diagram for user navigation

3.2.4 Conclusion and continuation

The subject of used design patterns in the client is evident by the previous mentions of viewmodel. Thus, the **Model-View-ViewModel Pattern** in combination with the **Observer Pattern**, comprises the design of the client.

For examples of the S.O.L.I.D. principles in the client, the **Dependency Inversion Principle** can be seen between the mediator and model, while **Liskov Substitution Principle** and **Interface Segregation Principle** can both be seen between the PrivateChatStruct and GroupChatStruct. **Open-Closed Principle** is certified in a similar way to the server.

Last but not least, the **Single Responsibility Principle** is evident, as there are many examples such as the Controllers and ViewModels being logically separated.



3.3 Networking and communication

3.3.1 Class responsibility

To summarize some of the previously mentioned features, the networking is Sockets based, but due to closely adhering to the dependency inversion principle, this can be changed without the need for modification of layers different from the mediator. Although the mediator is responsible for the networking itself, the ServerManager and ModelManager classes handle the logic of the communication. Both of these classes have their inherited method from the Receiver interface receiveContent() which handles the raw incoming content.The distinctive communication formula used offers greater control compared to sending serialized objects. It resembles an xml text node as seen bellow:

<tag//attr1=example//attr2=example>value</tag>

It consists of a tag, possible attributes and a value. The receiveContent() method processes the raw String and breaks it down to the specified elements. These elements are then passed to the processContent() method which recognizes the appropriately formatted combination of parts and acts accordingly.

3.3.2 Key sequences

The login sequence is the first to occur after opening a session towards the server. The diagram in **Figure**3.6 shows the **ClientHandler** forwarding the <login>ID//pwd</login> request to the **ServerManager**. It gets processed by the **processContent()** and receives the required Employee instance from **EmployeeManager**, based on the ID received. What follows is a check for password match with the Employee's password and if accepted a message with successful login is send. Access denied is sent otherwise.

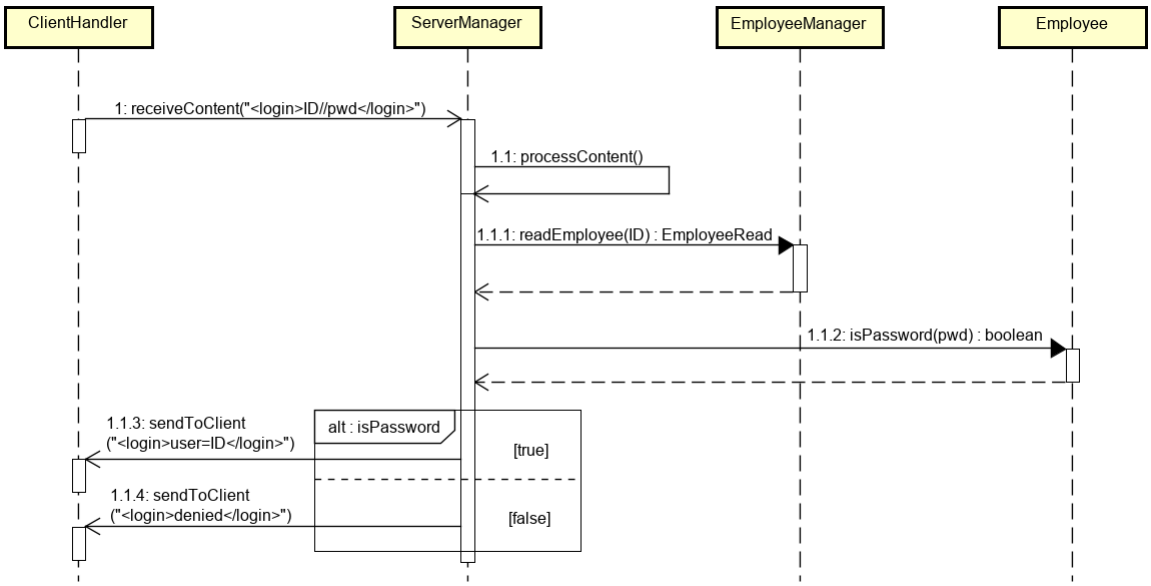


Figure 3.6: Login sequence diagram

The lock sequence diagram shown in **Figure** 3.7 occurs each time a client tries to edit an employee or a department. It has a similar preamble as the login, the **ClientHandler** forwards the <lock>employee=ID</lock> request to the **ServerManager** and it gets processed by the previously mentioned method. The **ServerManager** checks for admin privileges of the Employee forwarding the request and if it returns false an error for unauthorized request will be logged. If authorized, it puts the current session into the requested Employee's protection Proxy's queue, and if the session is first in queue the Proxy notifies(using the ServerManager) that it has the lock.

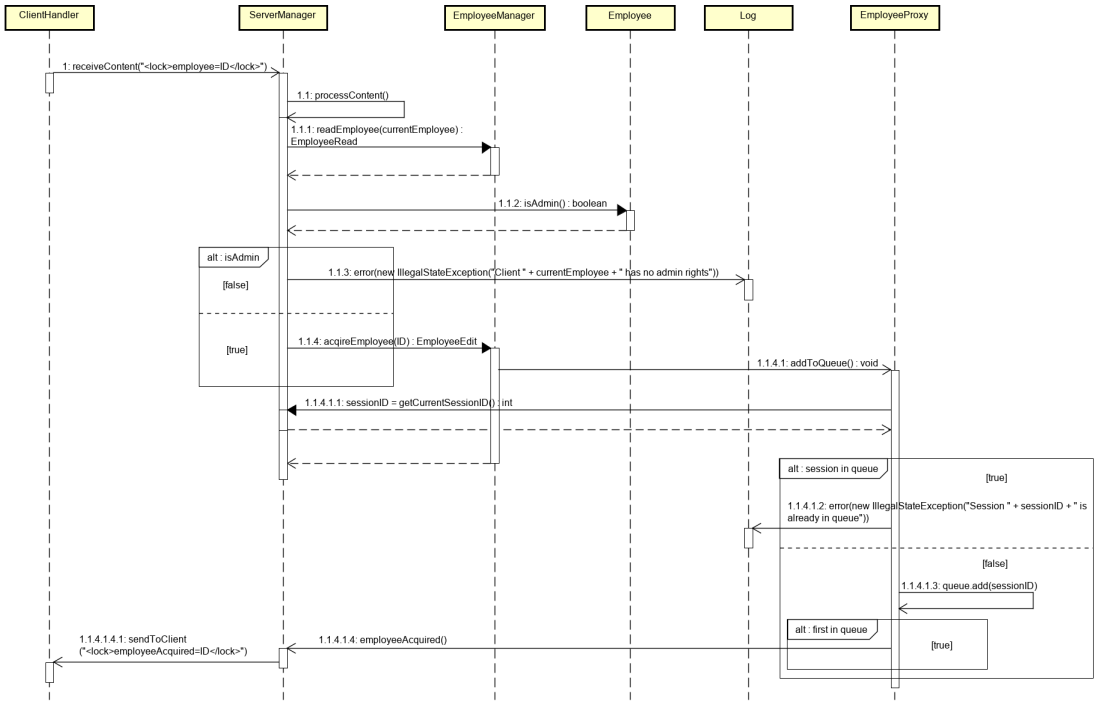


Figure 3.7: Lock sequence diagram

The edit sequence as shown in **Figure** 3.8 follows the locking sequence (if everything works nominally). It begins in a similar way to the locking, up to the point of the checking of admin privileges. It obtains the EmployeeProxy instance (in the form of EmployeeEdit interface) and tries to update the Employee (through the Proxy)and the Proxy checks if the request is coming from the session first in the queue. If not, it logs an error that the session has no lock on the employee (thus it has no write privileges). If its confirmed, it delegates the update method to the real subject. Next the **ServerManager** grabs a copy of the Employee and updates the database using it. What follows is the actual information, which arrives in the database, getting re-loaded into the **EmployeeManager** (overwriting the existing one). This is necessary to ensure the business layer has matching data to the persistence. It concludes with broadcasting the modifications made to all active clients.

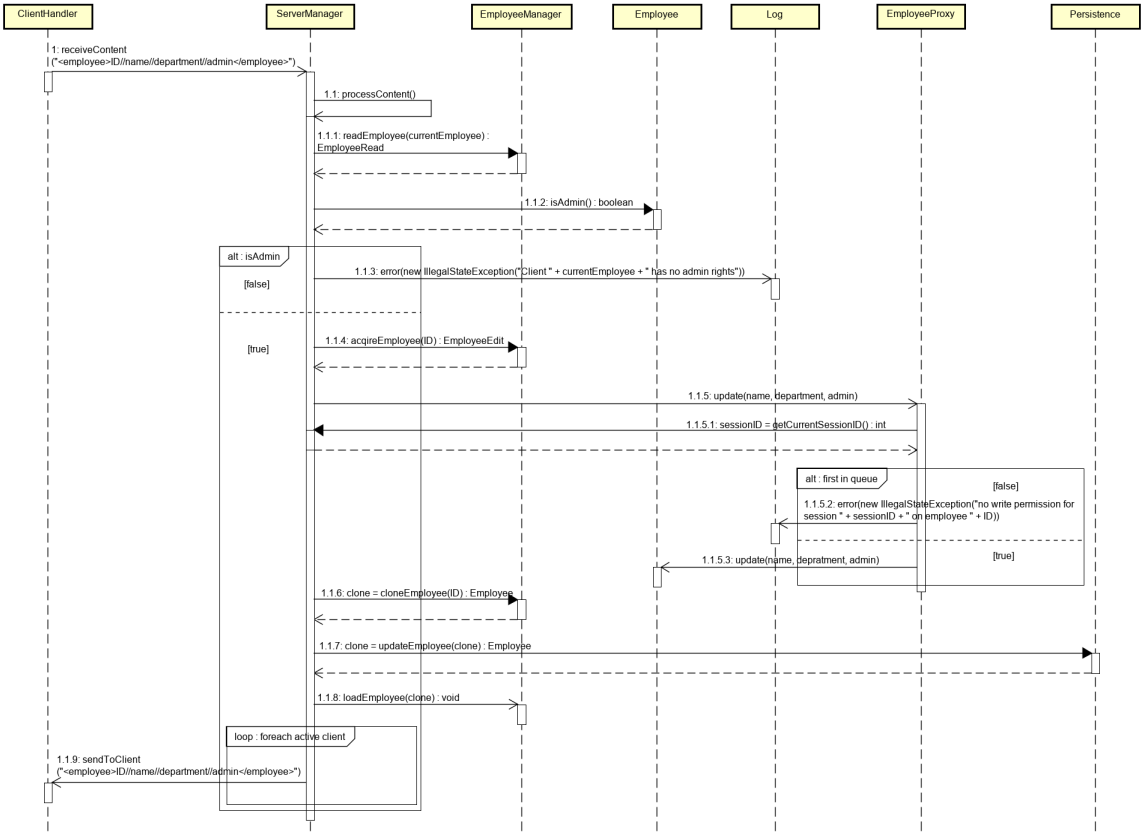


Figure 3.8: Edit sequence diagram

The final covered sequence in this document concerns the lock release shown in **Figure** 3.9. The initial receiving and processing parts match the ones explained in previous sequence diagrams. Following is the releaseAll() method getting called by the **EmployeeManager**, which calls the removeFromQueue() method on all existing non archived Employees Proxies. The proxy removes the caller session from the queue (if it is a part of it), while also notifying the next session in the queue about acquiring the lock, if the removed session was the first.

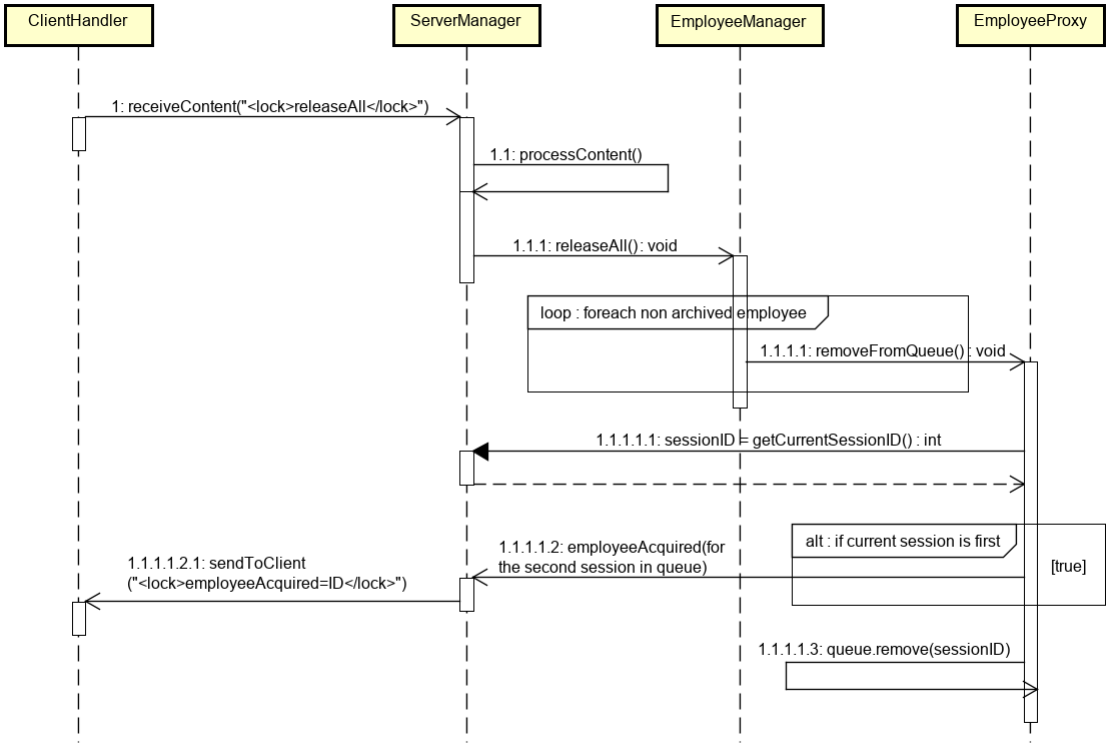


Figure 3.9: Release sequence diagram

Chapter 4

Implementation

The implementation process closely follows what has been discussed in the Design chapter. The diagrams are translated into code, taking into account the design choices previously presented. The client GUI corresponding to the appropriate views was developed by using JavaFX and it's documentation. [1]

4.1 JavaDOC

JavaDOC for both the server and client modules have been made where every attribute and method have their respective detailed description. This in addition to the source code provided, leaves a few methods to describe in depth.

The only part which requires more detailed description is the ProcessContent() method in both the client's ModelManager and the server's ServerManager classes.

4.2 Processing instructions and requests

This method has its own internal arrangement, as it is quite large in terms of actual code.

```java
private void processContent(String tag, String value, String[] attributes)
{
  RuntimeException exception = null;
  boolean recognized = false;
  switch (tag)
  {
    case "command":
      try {
        //_____________________________________________________________________________________
        if (value.substring(0,"archiveEmployee=".length()).equals("archiveEmployee="))
        {
          //do operations
          recognized = true;
        }
        //-------------------------------------------------------------------------------------
      } catch (StringIndexOutOfBoundsException | ArrayIndexOutOfBoundsException e) {exception = e;}
      try {
        //_____________________________________________________________________________________
        if (value.substring(0,"removeDepartment=".length()).equals("removeDepartment="))
        {
          //do operations
          recognized = true;
        }
        //-------------------------------------------------------------------------------------
      } catch (StringIndexOutOfBoundsException | ArrayIndexOutOfBoundsException e) {exception = e;}
      break;
  }
  if (!recognized)
    if (exception == null)
      Log.get().error(new UnsupportedOperationException("instruction not recognized"));
    else
      Log.get().error(new UnsupportedOperationException("instruction not recognized", exception));
}
```

Figure 4.1: Process content method

This is a simplified version of the actual method, in order to present its working principle.For more detailed analysis refer to the JavaDoc or source code.

Firstly, a **RuntimeException** ”exception” and a boolean ”recognized” variables are initialized as null and false respectively, followed by a switch statement responsible for recognizing the tag.

After recognition of the tag, the value also has to be determined; performed in a try catch block since certain .substrings() or .split() methods can produce **StringIndexOutOfBoundsException** and **ArrayIndexOutOfBoundsException** respectively. These exceptions have to be logged, however not

thrown (since it only means the content was not properly formatted), thus a catch block catches these RuntimeExceptions and puts them in the previously declared exception variable.

Inside the try catch the value is processed and the system act according to them. At the very end of the try block, the recognized variable is set to true.

After the switch, the content is examined for successful recognition and if not, the error is logged A possible cause is usually ”new UnsupportedOperationException(”instruction not recognized”)” found in line 30 and 32.

Extending this method is straightforward, if a new tag is introduced a new case has to be introduced for it to be recognized. In the newly specified case, a try catch block for each possible scenario must be built in the previously described way.

Finally, a visual aid is provided by the comments provided bellow to separate between each recognized operation.

//_____ and //----------

Chapter 5

Testing

The testing chapter provides inside on the methods used and results obtained from testing the software. To verify correct functionality, the user stories which compose the product backlog had to be tested and achieve satisfactory results, as well as identify and remove bugs present in the code. This was achieved by having test cases and JUnit testing.

By definition the test cases were derived from the use cases and follow different scenarios. Verification is obtained when the expected outcome matches the achieved one, thus achieving integrity.

The tables bellow provide examples of test cases concerning some of the more important use cases.

Precondition: be Logged as admin into the system				
Add employee				
Step	Step Name	Expected	Actual	Status
1	Select the administration panel	Opens the administration panel	In order	Passed
2	Select the Employee button	Opens Employee panel	In order	Passed
3	Click on the New button	Creates a blank employee at the end of list	In order	Passed

The use case test for creating new employee is straightforward and demands loggin as administrator. (similar to most editing structures)

Precondition: be Logged as admin into the system				
Edit employee record				
Step	Step Name	Expected	Actual	Status
1	Select the administration panel	Opens the administration panel	In order	Passed
2	Select the Employee button	Opens Employee panel	In order	Passed
3	Select an existing employee	Allows to select Edit and Archive	In order	Passed
4	Click on Edit button	An queue screen proceeds	In order	Passed
5	Queue screen puts user in line	Opens edit panel if first in queue	In order	Passed
6	Modify fields of choice and save	Edits to employees specific fields are done	In order	Passed
7	Successful save	Return to Edit panel with changes applied	In order	Passed

The use case test "Edit employee records" verifies the use cases with the same name. It is one of the more important test cases since it interacts with multiple layers of the system.

Precondition: be Logged into the system				
Edit group chat				
Step	Step Name	Expected	Actual	Status
1	Select the chat panel	Opens the chat panel	In order	Passed
2	Select the Group chat button	Opens Group chat panel	In order	Passed
3	Select an existing group chat	Allows to select Edit and Leave	In order	Passed
4	Click on Edit button	Opens edit panel for group chat	In order	Passed
5	Fill in desired fields and Save	Modifies selected fields or add members	In order	Passed
6	Return to group chat panel	Changes are applied to selected group chat	In order	Passed

The group chat edit test case does not require admin privileges, however it offers the ability to modify name of chat, add members or leave the selected chat.

5.1 JUnit testing

For the more complex parts of the code white box testing has been conducted. For simpler parts JUnit tests were performed, although the axiomatic parts like setters and getter haven not been Junit tested.

ZOMBIES guidelines were followed, however, for in-depth detail please refer to the unit tests themselves. Some summarizing results beginning with server side:

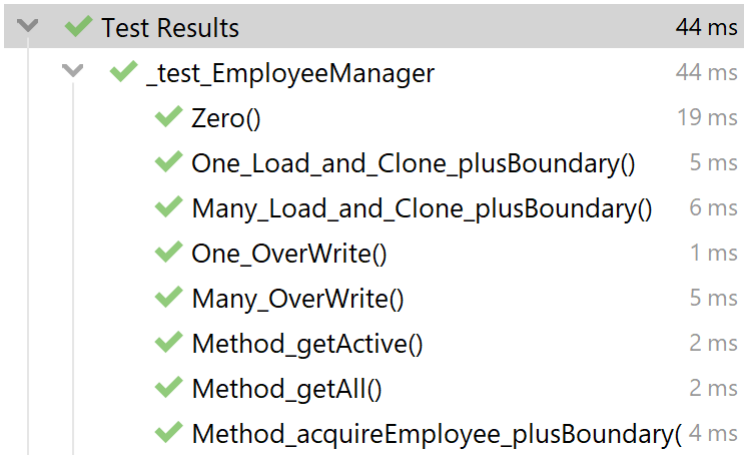


Figure 5.1: JUnit testing for Employee Manager

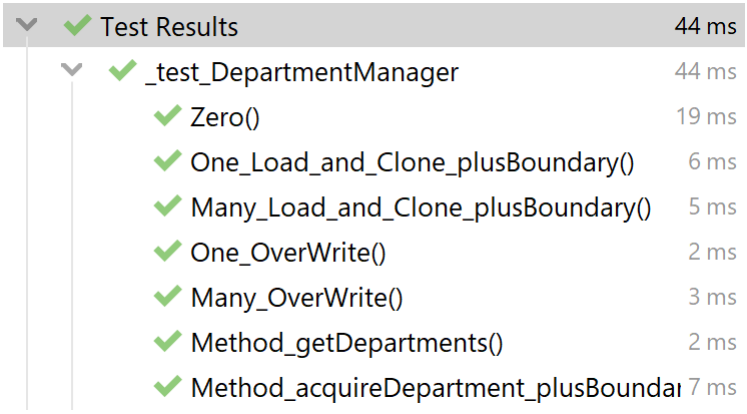


Figure 5.2: JUnit testing for Department Manager

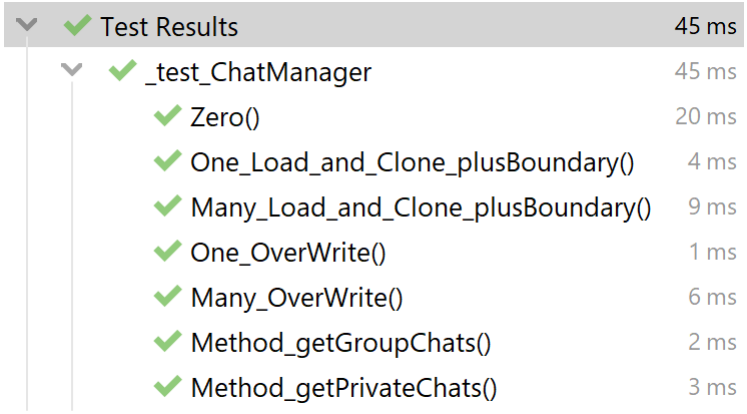


Figure 5.3: JUnit testing for Chat Manager

For the client side:

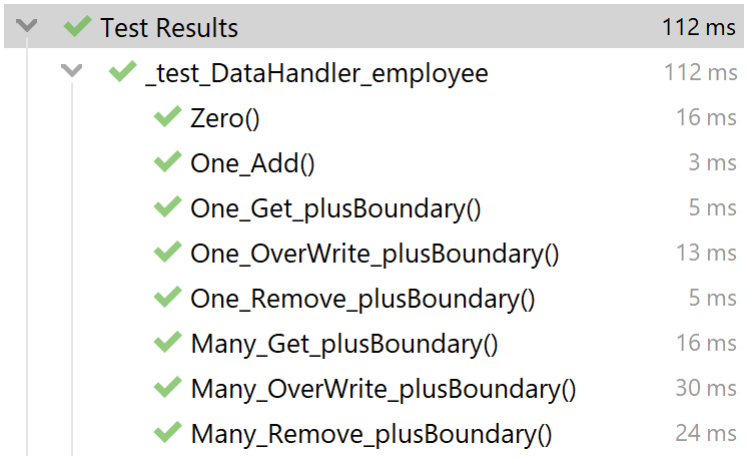


Figure 5.4: JUnit testing Employee DataHandler

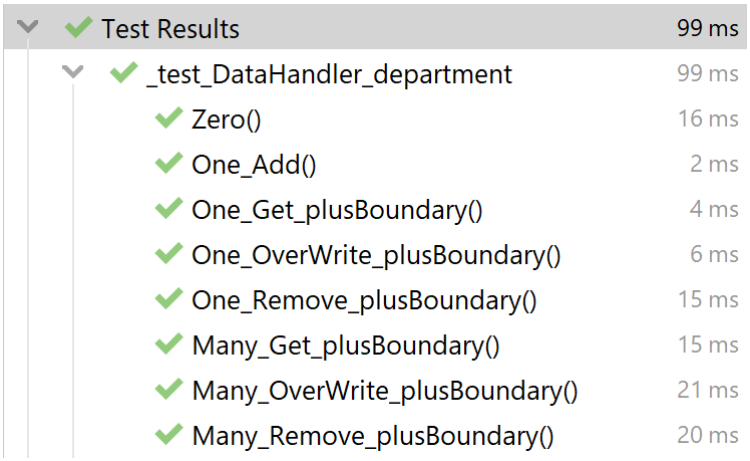


Figure 5.5: JUnit testing for Department DataHandler

Chapter 6

Results and Discussion

Currently, the accounts for employees are made directly through the database, which means employees don't have the ability to create their own passwords as they are made by the admin. This could be a future update for the system where employees can change their passwords, save log in credentials and the host.

The chatting system in both of its forms (Groups and Private) work quite well, for their intended purpose, albeit their performance speed and optimization could nonetheless be improved. Creating group and private messages is as intended simple and fast. Users have the possibility to both change chat groups data and delete it with a click of a button. The edit menu of group chats contains some irregularities - the menu contains different types GUI elements used for editing the group chat, and thus some standardization could be used here, although not necessary for the menu to function.

The manage data part of the system works as intended, as an admin you can interact with both employees and departments data which will instantaneously be updated on the database. To prevent the lost update problem, a solution was implemented so that only a single user can edit an employee at a time, while the others must wait for the current user to finish.

The UI is functional, but most definitely lacks any polishing present in systems with a longer development cycle. Its simplicity and clarity lead to an easier time for users engaging with the program for the first time. While the UI could be polished up, it was deemed unnecessary by both the product owner, as the UI in its current state covered all the functional requirements, and further time investments into it weren't deemed necessary.

Although the system has the main tools required for a communication system, as a future addition the system could include a matrix management system where managers can assign tasks to individual or multiple employees. Though the system and the database would have to be updated, since currently

an employee can have only one manager. It could also include postboards where employees could make posts, be it a help listing or a happy birthday post.

Chapter 7

Conclusion

After analyzing and concluding the customers' requirements, a translation was made into design, implementation and thorough testing. Solutions arose after creating proper use cases, activity diagrams and a domain model which created the first image of the system. With that in mind further steps of system development could be made.

With the creation of the class diagram we were sure that our system is closely following SOLID principles and various design patterns like proxy and singleton.

The GUI of the system was made with JavaFX which simplified to process of making a simple and effective UI as the need of hard-coding was replaced with a simple drag and drop visual editing function.

Although not all the requirements were implemented, the communication system still has basic functionality and the overall outlook is simple and convenient to use.

The database was made to follow the normalization process, and to ensure that there is no data redundancy, and all the relationships work the way they are intended. It also comes with the functionality of being able to review all the sent messages in the system, which adds a level of transparency.

Finally, the final version of the system was tested using JUnit framework, which ensured that the system is functioning and covers the most important use cases.

Chapter 8

References

[1] Openjfx.io. (2019). JavaFX. [online] Available at: https://openjfx.io/ [Accessed 23 May. 2020]

Appendix A

Process Report

A.0.1 Introduction

The aim of this report is to present a detailed summary of our group over the semester in a remote working environment, alongside both our failures and accomplishments. Within this report, will be contained information regarding group formation, general workflow as well as the strides we have made in our abilities and knowledge. Given the nature of a Central and Southern European group, cultural barriers were few and far between, as the cultural basis of our nationalities is naturally similar. As such, the few obstacles that were encountered were overcome with relative ease, partly due to the prior familiarity between the group members.

The goal of the second semester project is to develop client/server-based system, for that we chose to create a communications system, which would boost a company's efficiency by facilitating communication between employees of a company. The main requirement that was derived for our project was to implement a simple means of communication for employees, which resulted in a chat system.

In this project we utilized the concepts and knowledge that we have gained throughout the three different semester courses: Database Systems (DBS), Software Development with UML and Java 2 (SDJ2) as well as Software Engineering (SWE).

Our group agreed that we will work together on the project every Wednesday from 10:00 to 16:00 during the Inception phase and following the Elaboration phase and until the end of the designated time we worked every workday.

This paper further explains the different parts of the process including a group description, project description, initiation, execution and lastly personal reflections

A.0.2 Group Description

During group formation, we had two pairs of people, that had a similar vision of what we wanted to achieve during SEP2.

That's why our team was formed at the very first day of the semester, after talking about our experience from first semester, and the outlook on what we all wanted to achieve we formed a group of four, as we were all oriented towards getting as much knowledge as possible from the semester project. Later, our supervisor suggested an additional member, after consideration we agreed as our goals were aligned. It also was a great advantage that our cultures were similar, which resulted in the group meshing quite well, and as such avoiding a great many potential conflicts. We also had very similar personalities, and as such our work environment was casual, and less stressful due to the fact we could talk out most issues without worrying about how others will react. This led to a more coherent group than overwise would be possible.

A.0.3 Project Initiation

The Project was initiated from the moment we formed the 5-member group, and settled on the contents of the group contract, agreeing on how we are going to work and communicate with each other. The contract was then promptly signed, but due to unexpected world pandemic, we were forced to reconsider the question of how we planned to work as a team. The conception of our project came to be after long discussions regarding the scope of the project. Our first iteration of the idea was to create a game engine, however, due to time constraints we decided a more sensible route would be the construction of a communication system.

At first, we proceeded to brainstorm on the problems and issues of companies and businesses, and how we could solve said problems and issues with a software approach. The issue we focused on was the lack of an efficient communication system in companies around Denmark, which would result in a disorganized workflow, inefficient employees as well as confusion surrounding all communication. With the conclusion of our brainstorm we decided that the best course of action in order to tackle the said problem would be a communications system.

Once planning had started, we begun to decide on what tools would be used to aid us in the creation of the system. We had to find a solution regarding storing all the relevant documents in a way so that it could easily be accessed by group members. The best solution that we found was to use Microsoft Teams. Due to an unexpected world situation, we also had to rethink of how we were going to communicate with each in a remote environment. For that, a discord server was set up, through which we would be enabled

to communicate, discuss ideas, plan meetings, as well as execute sprints. In addition, for version control of the system, GIT was utilized, which allowed us to work together on the code in a remote fashion.

Throughout the project studienet and itslearning material was frequently referred, so as to make sure that the group is following the right path, by following the guidelines in the referenced material as well as utilizing the templates that had been provided. The allocated time for the project had been divided into 7 SCRUM sprints, each of which was 3 days long.

At first, it wasn't easy to switch from the already tested waterfall, to an agile process approach. In the very beginning albeit trying our best to follow the principles of SCRUM, we were informed by the supervisor regarding our methodology, with our approach being close to waterfall rather than SCRUM. Despite our difficulties in the transition from waterfall to UP and SCRUM, we began the project period on the right track.

A.0.4 Project description

Throughout the semester our team was dedicated to creating a system that would cover all the knowledge that we gained which includes various design patterns, like MVVM and singleton, client/server model, relational databases and the SOLID principle. Our system covered all the requirements, which helped us in deepening our knowledge.

We made sure that our project stays true as much as possible to the project description that we have created at the beginning of the semester, it ensured us that we are on the right path and that we are tackling the problem that we have raised in the first place.

The main difficulty of this semester was that because of the outbreak all of us had to work from home, which resulted in team members losing motivation, despite this, with the help of our supervisor, we slowly regained our motivation and gave ourselves all in to the project. Unfortunately, the situation caused time and workflow problems, as remotely we couldn't achieve as much as we did in person

A.0.5 Project Execution

The project execution was a relatively mixed experience, primarily due to the Covid-19 pandemic, and the subsequent lockdown that followed. The software development process framework that had been provided for the current semester was Unified Process (UP), with an iterative approach. UP differs significantly from the software development model which we used in the previous semester, namely Waterfall, in the fact that while Waterfall takes a linear approach to software development, UP is iterative and incremental. While Waterfall lacks in flexibility, making it a very rigid development model,

UP is, at least comparatively, flexible. At the start of the semester we made our grand plans for the project, without realizing what was to come. As such, we set our expectations far beyond a reasonable level, considering the Covid-19 lockdown was to come. Once the lockdown started, it could be argued that Waterfall would have served us better than UP in these trying times, as it took us quite a while to properly settle into the UP dynamic, and the remote working environment we were forced into only deepened our woes. As such, we had to take some counter measures to make sure we had a project ready for delivery at the deadline.

Firstly, we decided not to rotate roles, so as to not strain the team with constant role swaps, having a clear, established hierarchy. This decision proved to be a great boon for our project team, as this led to a continuity of ideas, development and priorities, removing the friction in communication that would be caused by a role swap.

Secondly, we divided the project into sprints. Each sprint was the same as the last, following the exact same formula from start to finish. Every sprint began with a SCRUM planning meeting at the start of the Sprint. During said meetings, the team created, under the direction of the Scrum Master, a sprint backlog for the upcoming sprint. At the start of each day, we held a Daily Scrum meeting, during which the team discussed what they had done the previous day, what still needed to be done and the like. Concluding the sprint, we ended our sprints with a simple as-seen-on-tv Sprint Review, during which the product owner comment on the state of the project âĂŞ regarding the general satisfaction with the progress and current state of the system, and then he product manager would conclude the sprint. Finally, the team held a Sprint Retrospective, during which the group members could discuss the sprint from their point of view.

Developing a proper product backlog was the key in creating a communications system, as it gives a direction that the team must take. The current product backlog had many user stories, which proved to be quite cumbersome and hard to finish in a timely manner, that's why only Critical and High priority tasks were completed. Throughout the project period, we had to shrink the product backlog to not include several features and shift priorities downwards, due to time needed to finish relatively complex tasks, of which there too many, as we hadn't broken the tasks down enough.

At the beginning of our sprints, we were not quite sure how to properly follow UP, which resulted in our supervisor pointed out that it appeared like we had followed waterfall once more and not unified process, after which we had to do a major reorganization of how we were working. From there on out, we began focusing more on the methodology itself, which resulted in us advancing the progress on our system exponentially. There was an issue at hand though, for the tasks took longer than was expected, which resulted in them being incomplete by the end of the sprint, or not being started at all, due to

previous miscalculations. Therefore, our efficiency was not at its peak, but instead of getting worse, following Unified process we managed to slowly build our efficiency back up. In the end we could finally present a properly working system with basic functionality. This was a major upgrade from the previous semesters waterfall method, where one had a system only at the end of the implementation, whereas following UP and SCRUM we already had a working system in the beginning of implementation, albeit of a smaller nature.

A.0.6 Personal Reflections

Audrius

Personally, I feel like this semester had a lot of potential to be great, but sadly I must admit that it was a terrible experience. While I do agree that I did learn more about remote work, due to the Covid-19 pandemic, it didn't make the semester good in any way. I had a lot of motivation to learn and improve my skills, but that chance has been taken away. Before the pandemic, working on a project meant that you would learn a lot from both your group members and teachers, but with the remote work that was just not possible. We couldn't exactly write code together and ask which part does what, and explain yourself as you code, we couldn't ask questions as easily to the teachers, as by the time you got a reply youâĂŹve already started working on a different part of the system, or you just never got the reply at all in the case of DBS, which made a huge impact to the productivity. Because of this we had to direct our questions to a different teacher. It was the most stressful semester yet, as not only the situation made things worse, but also the loss of motivation from other group members made it impossible to have hope, trying my best to put everyone back together time after time proved to be fruitless. Only after a certain time had passed the team members started to regain their sense of responsibility and motivation. From then on, we started to fully follow SCRUM, and progress heavily on the system. Though we made progress, I personally do not believe that I progressed as much as I should during this semester, mainly due to the pandemic. While our group did not have a lot of conflict, there were a lot of misunderstandings, which were caused by the uncertainty of the situation and desperation. Despite all of this, I felt like that most of the time I could depend on them, as everything would be done on time, although frequently at the last minute. I believe that the best thing about our group was the very casual atmosphere, which helped not to stress about deadlines too much. All in all, I believe that this semester was not a good one, as I do not feel like it helped me progress much further into my profession, and due to the situation made it hard to keep up with everything going on

Gustaw

This semester was atrocious. Not due to the teaching program, but rather due to external forces. The coronavirus pandemic and the subsequent lockdown, which lead to a remote working and learning environment, impacted my satisfaction with the course overall. While SEP2 itself was relatively okay, considering the requirements and learning material, the remote working and learning severely hampered both my potential contribution to the SEP group, as well as learning. Personally, I learn extremely poorly at home, and best in class, and thus was unable to bring out as much as I wanted to from this semester. While motivation during the early days of the semester was flying high, by the time the lockdown rolled around, it began dipping, and once we were in full lockdown, I lost all motivation to work, at least for a while. For the most part, due to the remote environment, a lot of learning opportunities from SEP2 where lost. For the most part we worked in pairs, and I focused on the DBS section a lot more. With everything in consideration, these were probably the better moments of SEP2, doing DBS. From the Java side, due to our rapid progress once work started, I felt that I could not learn as much, due to either a lack of communication in terms of code explanations, or lack of interaction with said part of the code. While stress was never a problem for me, as I handle stress very well, it seemed like stress was a problem for some of the other members, and seemingly they had to resort to shouting and anger with problems that, from my point of view, were banal and relatively simple to fix. On the note of stress and motivation, some members of the group couldn't at all motivate themselves, which hampered our progress. It is worth noting that the organization and structure provided by SCRUM proved, in some ways, to be the saving grace of this semester, as once the group got into the rhythm of working, the work went smoothly, and as such average motivation across the group was on the rise, week by week. Despite all the difficulties and setbacks, id still say, that all things considered, I am proud of what we had done.

Ioana

In my opinion this semester was the challenge that no one saw coming. This corona virus situation affected me to a big extent, but I managed to get back on track in the perfect moment. The main problem in my opinion was the fact that we had to do group work while not physically being in a group. I believe the beginning of the corona virus was a blur for all of us because we had to adapt to a new way of working. Before the pandemic I was excited and eager to do my best for the project because in SEP1 I did not quite get the chance to use my abilities. After the blur period that I went through, I realized that my team was extremely united. We functioned as a team even during hard times and I must admit I wouldn't have made it through this semester without them helping me understand. I happy that I managed to help my group with everything I could, even though I am fully aware of the

fact that they worked more than I could. I mostly worked on the analysis part and some parts in the project and process report. I learned a lot of new things semester, but I think that because of all the stress with the corona situation I missed some information. At the end of the day, I am proud of what we achieved in this SEP and how we managed to get through a hard situation together.

Levente

We started this semester with high expectations from ourselves, regarding the what we have planned to achieve. To be honest we could have met our expectations, but when the COVID-19 pandemic stared, and we were forced to work remotely, the already high expectation suddenly became unrealistically high. Nevertheless, we still learnt how to use the tools which we had to utilize this semester, like the scrum framework, unified process, S.O.L.I.D. principles and many others. In conclusion we had a useful semester learning a lot, but due to the previously mention COVID-19 scenario, the actual process was a mixed experience. My worst periods were when, some of my teammates understandably lost all their motivation, and as I somehow managed to keep up mine, I had to push them from time to time. This wasn't a pleasure for them, but neither for me, another solution for to this motivation problem was to put in some extra hours myself to compensate for some of my teammates. Personally, I'm more than satisfied with the knowledge and experience that I managed to gain this semester, although I had to put in twice as many hours into learning database systems because I couldn't follow my lecturer during classes.

Philip

The semester was a strange but necessary experience to say the least. Before i proceed with description of the work process and structure, I would like to mention why it was a necessary experience. In many IT companies it is not unusual to have "Home offices" from time to time. That being said, it is a different experience when you are starting from scratch compared to maintaining code or bug fixing remotely. Nevertheless, I believe it was invaluable to learn to better organize remotely and strictly follow a process that is designed around specific working hours and strict documentation. The working process was rather shaky in the beginning, while trying to obtain some inertia for greater productivity. It was in April when group members started realizing "This is where we stand, Let's do something about it." What follows is by no way perfect, but given the circumstances it was acceptable. Slowly but steadily we picked up pace and from the end of April and throughout May we were able to maintain some organization. The project itself was challenging at times, as creativity and consensus are hindered by remote work. Ideas were not easily conveyed, which led to some divisions and overestimations on how

to proceed. The solution seemed to be initiating work and estimating how much we can handle based on our previous performances. SCRUM also helped a great deal in keeping us grounded. On the topic of SCRUM and the agile framework, I had the opportunity to be Scrum master throughout this project. Obviously expectations were for documenting and reviewing live sessions and estimating our abilities face to face. Since this didn't happen, I tried to stay optimistic and perform the same duties remotely. In the beginning it was rather hazy how to perform my services responsibly, but throughout the increments I improved my communication with the product owner in order to ensure clear and realistic scope and goals. Same goes for the development team, I tried being in touch with everyone but it sometimes proved difficult to ensure attendance and concise reflection. I must say at the end I learnt much better how to perform the role, but would like to learn under a more experienced person in order to reflect on my own shortcomings. Finally I would like to reflect on the project as a person and not a member of the group. Motivation was not always there, it was like tides and i think it was similar with everyone. Discipline was a saving grace, as it allowed for stricter work ethics. Nevertheless not everyone was on board for that which may have lead to some disputes in the group, which were later resolved. I feel some of the courses were not as helpful as they could have been if thought live, and there were some technical issues with the software in use, which were not addressed till the end. (Skype for Business) To summarise it was a challenging period, but it allowed for some unique experiences which could hopefully help me in the later on.

A.0.7 Supervision

Throughout the project, our group was supervised by Steffen Vissing Andersen (SVA), who helped us stay on track and motivated us when times proved to be harsh. Most of our meetings were directed towards SCRUM and documentation. Our second teacher, Ole Ildsgaard Hougaard (OIH) helped us a lot in design and with the database.

Both teachers helped us a lot throughout the semester, Steffen motivated us a lot at the beginning of the project period and put us on track. At the beginning we thought that we were following SCRUM, though after a meeting with SVA, it turned out that we were too used to the waterfall approach that we started unconsciously using it again. After this, we made sure that we followed scrum and had all the necessary sprint artefacts. Throughout the period, we had multiple meetings to ensure we are on track as every now and then Steffen would point out various mistakes.

Ole gave us many insights on Java and the database. We frequently encountered problems while creating and designing the database, which would stall our progress. Ole was always quick to reply to our questions and was very clear. We learned a lot of useful information regarding the database part of the system, as

well as the persistence layer of the server. We had never had any experience regarding the connection between java and a database, so Oles help was instrumental in the completion of our system.

In conclusion, both teachers had a significant role, throughout the project period, contributing invaluable experience to our project.

A.0.8 Conclusion

This semester, the transition from the waterfall method to a combined UP and SCRUM methodology was a major improvement, as it allowed us to increase our flexibility and efficiency. Each member of the group had a part of the project that they were responsible for, which enabled all of us to gain knowledge in various fields. Through trial and error, our group learned how to manage our workflow more efficiently, by following the unified process and SCRUM, though it proved to be difficult at the beginning. During second semester, it became easier to focus on the main functionalities and to see clear results, as we had to implement a working system from onset, while continuously adding upon it, in contrast to last semester, where there weren't as many concrete directions, from which we had to begin and where we had to finish. As for the future, we believe that it is crucial to quickly adapt to a remote environment as well as to not postpone certain methodology rituals. As it is difficult to proceed onwards without a proper conclusion of previously done. While the group work done throughout the semester was not perfect as it was hard to adjust to a remote environment, we are satisfied with the outcome.