SOFTWARE ENGINEERING PROJECT

Team ‘Télétravail, Famine, Pâtes-Riz’ :

ALMEIDA Mickael

BERNARD Hippolyte

DRAY Gabriel

TABLE DES MATIERES

ORGANIZATION OF THE TEAM

STUDENTS AND THEIR RESPONSIBILITIES:

- ALMEIDA Mickael (*Captn138* on GitHub): Lead Tech

  Threads : 1 receiver and 1 listener in order to keep receiving messages while sending others

  Networking : connection between client and server

  Security : The passwords are hashed with SHA512 before being inserted into the database. Thus, no clear password is stored in the database;

  UMLs : Drawing the architecture UML

- BERNARD Hippolyte (*Rxdsilver* on GitHub): Lead Graphist

  Graphical User Interface : what the user sees, in a window with buttons, text fields etc

  Basic classes : the classes to make the chat work.

  Uses Cases : the use cases of the chat.

- DRAY Gabriel (*Laartem* on GitHub): Lead Organization

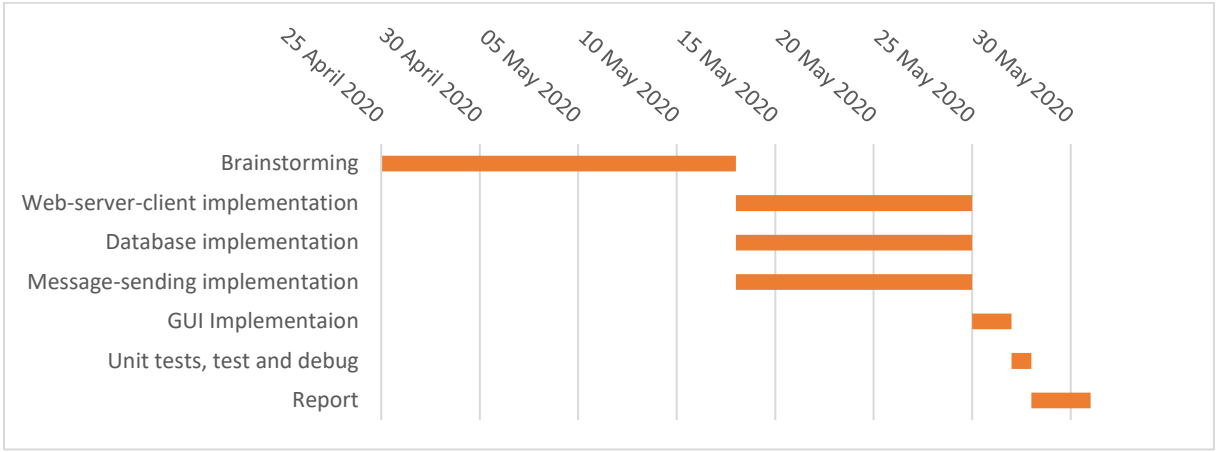
  Database : How to store messages and users/passwords (hashed) in a database.

  Organization and Managing of the group : Giving the tasks and the deadlines for each work.

  Unit Tests : Building the unit tests.

DEVELOPMENT

We have chosen to work with a V-model, as we are only three and we need to overlap the development of several component. Thus, we chose this model as it provides us a convenient way of working.



SPECIFICATIONS

SYSTEM BEHAVIOR DOCUMENT :

Here are listed the requirements of our project. They are classified as functional (F) and non-functionnal (NF).

Requirement ID	Name	Type	Description
REQ01	Responsivity	F	Server must be maximum 1 second responsive
REQ02	Multi-platforming	F	Application must run on Windows/Linux/Mac
REQ03	Stockage	NF	Messages must be stored in a database
REQ04	Reliability	NF	Messages must not be lost
REQ05	Security	NF	Passwords must never be sent nor stored in clear
REQ06	Non-duplication	NF	Usernames must be unique
REQ07	Account	F	Accounts are locked by the couple username/password
REQ08	Connectivity	F	User can connect on multiple instances
REQ09	Continuity	NF	Sending messages doesn't interrupt the reception of new messages
REQ10	Simplicity	F	Users can send messages easily

To produce the list of the requirements, we ask ourselves what was necessary and what wasn't. We started by starting out what were the simplest and needed functionalities, and then we tried to imagine how we could make the application and finally what would be in it. Also, we looked at all the chatting apps we already know to see what could be done.

USE CASES

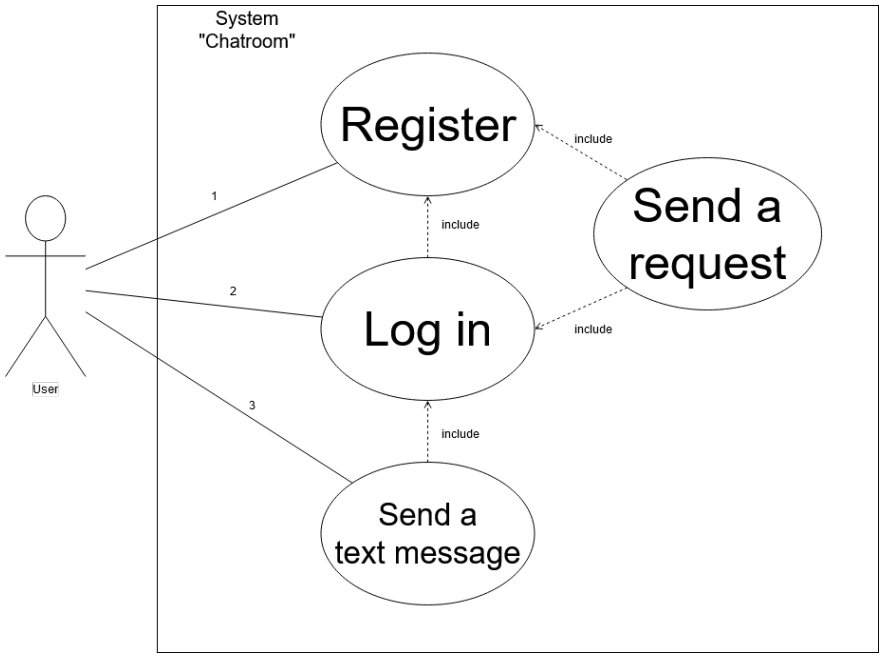
USE CASE 1	**Register**	
Goal in context	User issues a request to the server and expects registration to be done in second	
Preconditions	Application is running	
Success end condition	User sent a username which is not taken yet	
Failed end condition	User sent a username which is already taken	
Actors	User, local network, server	
Trigger	Registration request comes in	
Description	Step	Description
	1	User sends a registration request with a username and a password
	2	Server captures username and password and checks if the username is not taken
	3a	Server sends confirmation of the registration
	4a	User is registered and can connect
	3b	Server sends failed registration information
	4b	User must retype username and password
Performance	1 second or less	
Superordinate	None	
Subordinate	Send a message (use case 3), Log in (use case 2)	

USE CASE 2	**Log in**	
Goal in context	User issues a request to the server and expects connection to be done in second	
Preconditions	Application is running, user has a registered account	
Success end condition	User is registered, user send the right username and password	
Failed end condition	User is not registered or has sent the wrong username or password	
Actors	User, local network, server	
Trigger	Connection request comes in	
Description	Step	Description
	1	User sends a connexion request with a username and a password
	2	Server captures username and password and checks if the couple corresponds to a registered user
	3a	Server sends confirmation of the connexion
	4a	User is connected
	3b	Server sends failed connection information
	4b	User must retype username and password
Performance	1 second or less	

Superordinate	register (use case 1)
Subordinate	Send a message (use case 3)

USE CASE 3	**Send a message**	
Goal in context	User issues a request to the server and expects message to be sent in second	
Preconditions	Application is running, user is connected	
Success end condition	User is connected to the network	
Failed end condition	User is not connected to the network	
Actors	User, local network, server	
Trigger	Message comes in	
Description	Step	Description
	1	User sends a message with its username and a content
	2	Server captures messages
	3a	(If the message is a text message) Server saves the message in the database
	4a	Server sends the message to everyone
	3b	(If the message is a connexion request) see use case 2
	3c	(If the message is a registration request) see use case 3
Performance	1 second or less	
Superordinate	Register (use case 1), Log in (use case 2)	
Subordinate	None	

Here we have summarized the use cases in a diagram:



DESIGN

MODULES

Our project will be separated in 3 modules.

First one is ‘chatroom’. It contains all the base classes, such as Message, Sender and Reciever. These are the class that are used by everything else.

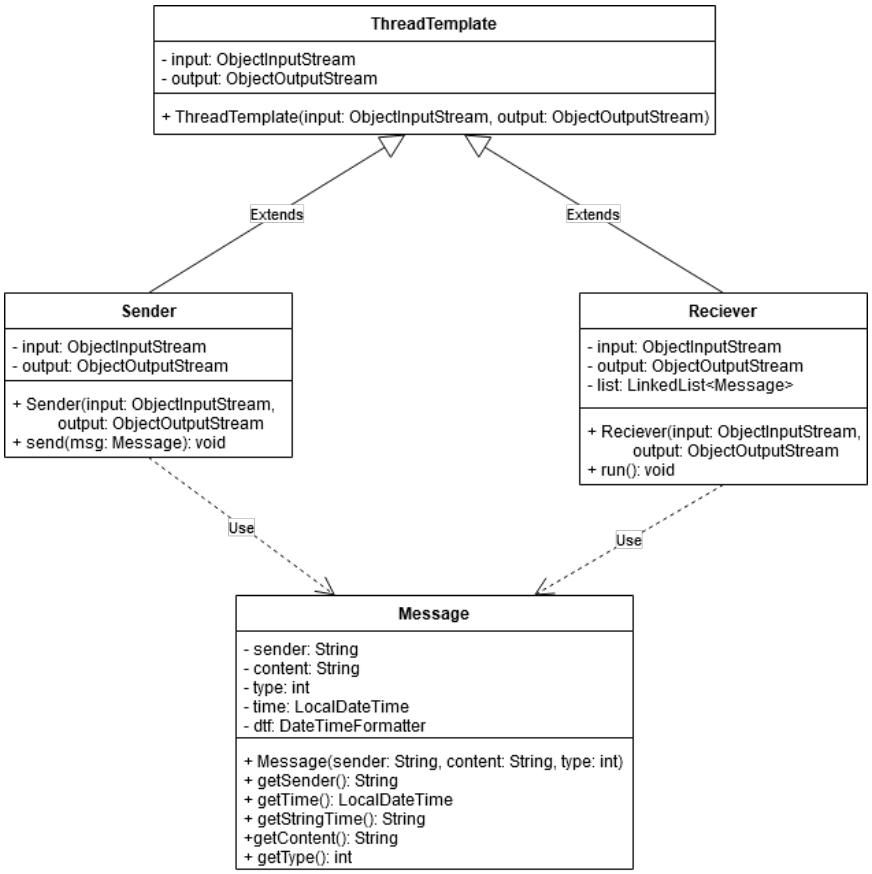
Second one is ‘client’. It contains a GUI class and a Client class. This one will handle interactions between the GUI and the rest of the classes.

Last one is ‘server’. It only contains a class Server because it is the only one that is needed on server-side.

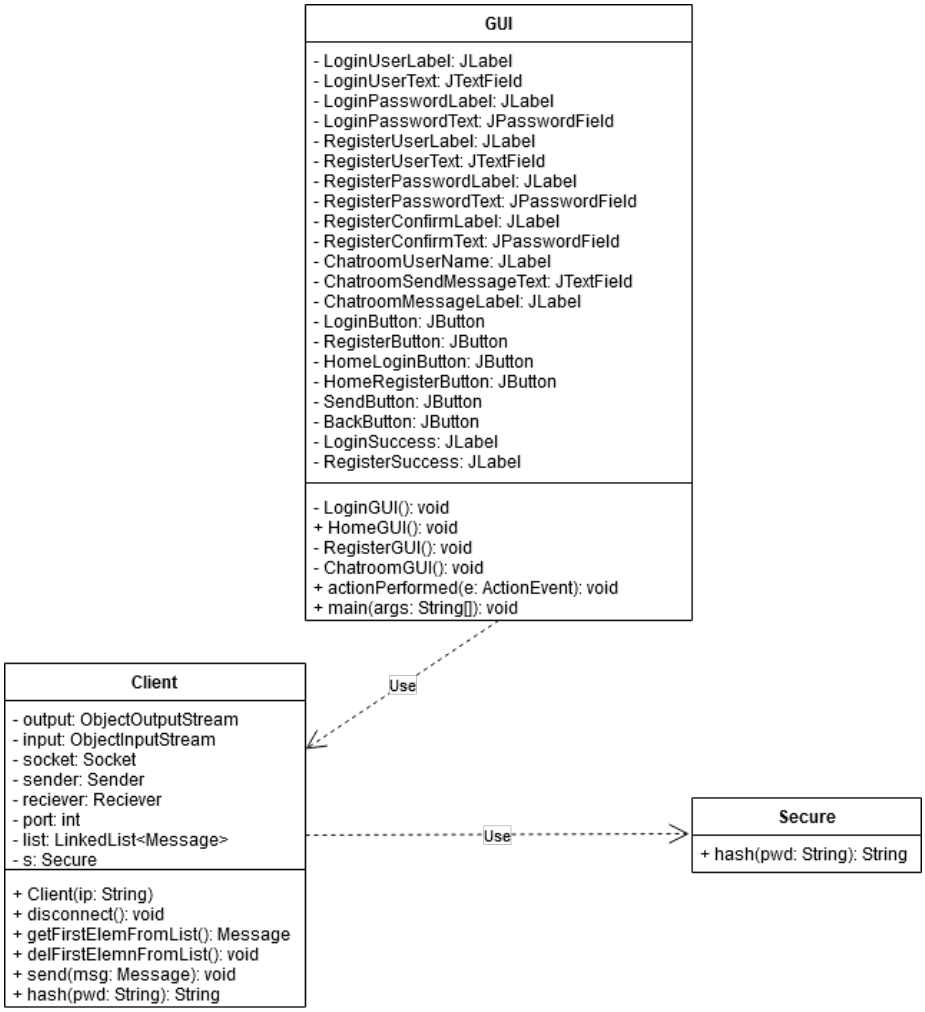
This means, on client-side, we have a package composed of the module client and the module chatroom, while on server-side, we will have a package composed of the module server and chatroom.

We modeled the different modules of our project in 3 UML diagrams.

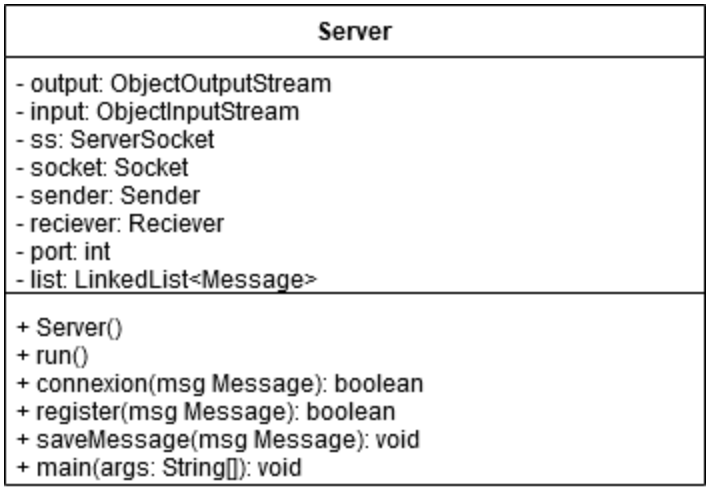
Package chatroom:



Package client:



Package server:



DESIGN CHOICES

We tried to get an application as clean and minimalist as possible. We didn't want too much information displayed that could lead to misunderstanding of the functioning of the interface, so we tried to automate some things and got rid of some others.

We have chosen the Facade pattern design, because not only it provides a unique control point, but it will also be easier to interact with, as there will only be interactions with one class.

SOLID PRINCILPLES

• Single responsibility: We have separated the code in several classes, each one corresponds to a precise task. For example, Reader only reads incoming messages while Sender only sends messages.

• Open/Closed: We use an abstract class to generate our threads. This means we could extend this class to create new types of threads without changing the code.

• Liskov's Substitution: This principle was actually not applied to the Threads.

• Interface Segregation: We did not create interfaces to implement in the classes.

• Dependency inversion: We did not use interfaces when calling methods.

GITHUB REPOSITORY

Our project can be found online on the following GitHub repository:

https://github.com/Captn138/Chatroom

We can find all the Javadoc in the 'Javadoc' folder, a copy of the report in the 'Report' folder, the modules in the 'Modules' folder, the Eclipse code in the 'SoftwareEngineeringProject2020' folder, the Unit Tests in the 'Tests' folder and some things that didn't work in the 'Unused' folder.

TESTS

TEST PLAN

- Check if the app runs on Windows, MacOS and Linux.
- Try to create an account.
- Try to connect on multiple instances.
- Try to send a message and check that it is received in less than 1s.
- Try to send a message on two instances and check that REQ09 is applied.

TEST REPORT

Test 1 :
Server launch : success
Application freezing when clicking on “Register” or “Login”
Forcekill the application.
End of Test 1.

After several tests, the result is basically the same : we can’t access to the chatroom.
We spent a lot of time looking at the code, but we couldn’t manage to find out where the error comes from.