NShare chat and file sharing program

Analysis and Design Document

Student:Bánhidi Zoltán

**Group:30234**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <31/5/17> | <1.0> | Documentation | Bánhidi Zoltán |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

NShare is a chat and file sharing program, which can be useful for realizing communication via internet. It tends to remove the implication of Flash drives or CD/DVD’s for sharing small files. These files include mainly images, but the format can be arbitrary. The program also has a chat feature which can be exploited.

The program has two types of users: administrator and regular user. The administrator can add, modify and remove user accounts from the application. The regular users are the main target of this application. They can chat with each other and share files by the network. The shared files must be smaller or equal than 8 MB (size limitation). The shared files, as the rest of the data maintained by the application are stored in a MySQL database.

The application is client-server based. The clients communicate with the server using HTTP requests, because the application is RESTful. The server manages the MySQL server locally. The implementation language is Java 8, using IntelliJ Idea IDE and SQL for the database part using MySQL. The user interface was created using the JavaFX library.

# Elaboration – Iteration 1.1

# Domain Model

# C:\Users\banhidi\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Main.pngArchitectural Design

## Conceptual Architecture

The system is designed using Layers Architectural Pattern. Layered architecture focuses on the grouping of related functionality within an application into distinct layers that are stacked vertically on top of each other. Functionality within each layer is related by a common role or responsibility. Upper layers such as the user interface layer send commands to lower layers, such as the business and data layers, and may react to events in these layers, allowing data to flow both up and down between the layers.

Another architectural design pattern used is *Client-Server Design Pattern.* Client-server architecture has two types of participants: one or more clients and a server. The server has to process the commands that come from the clients and send back a response. In this project, the server will be a Java application hosted in localhost. The clients should connect to the server using a specific host address and port (*http://localhost:8080/api*). After that, the clients could ask the server various tasks using HTTP requests. In case of this project, the client is Java desktop application.

## C:\Users\banhidi\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Main.pngPackage Design

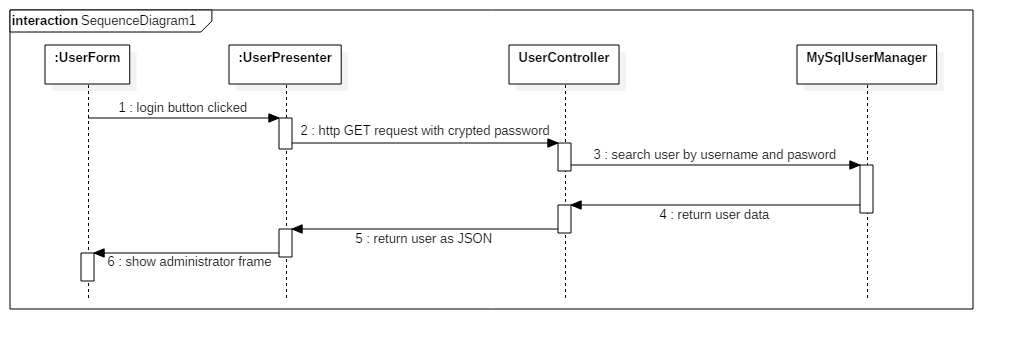
## C:\Users\banhidi\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ComponentDiagram1.pngComponent and Deployment Diagrams

# Elaboration – Iteration 1.2

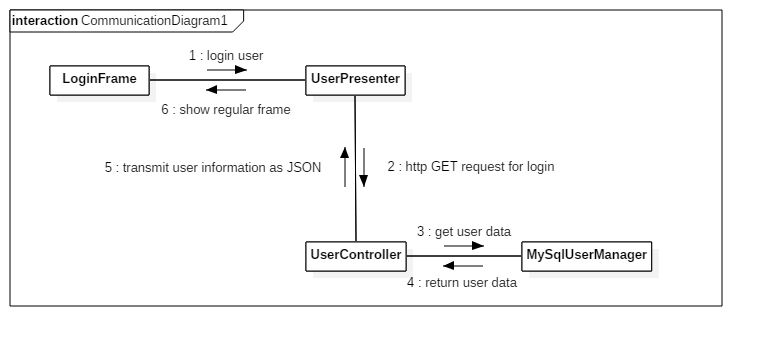
# Design Model

## Dynamic Behavior

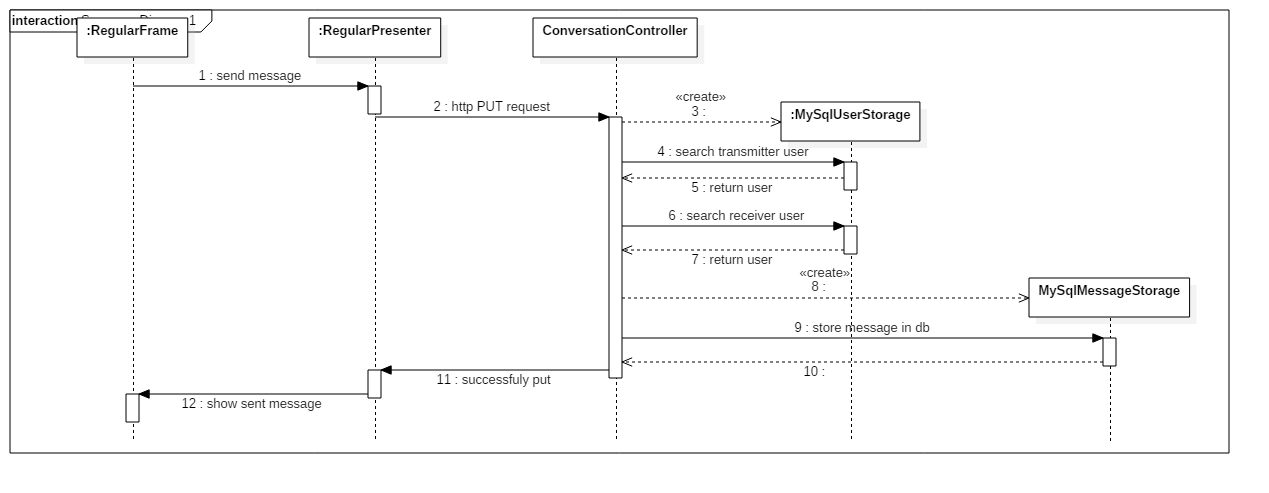
Sequence diagram for logging in an administrator user:



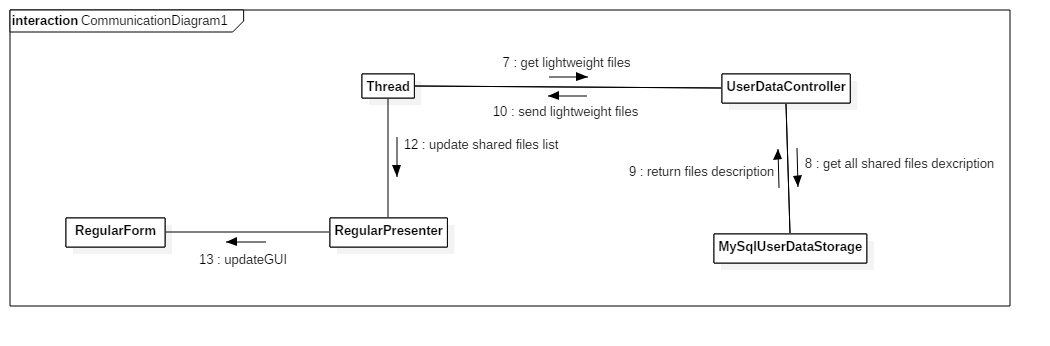
Communication diagram for logging in a regular user:



Sequence diagram for sending a message from a regular user for another one:

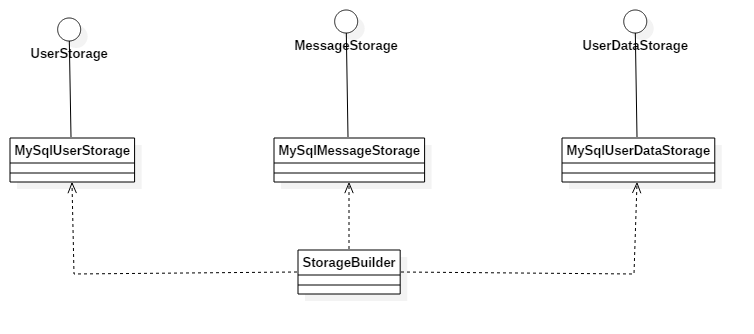


Communication diagram for getting new shared files (used a thread which regularily checks for new shared files):

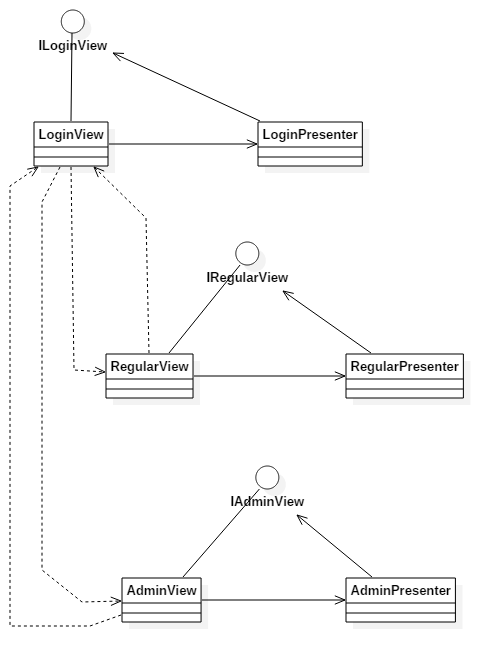


## Class Design

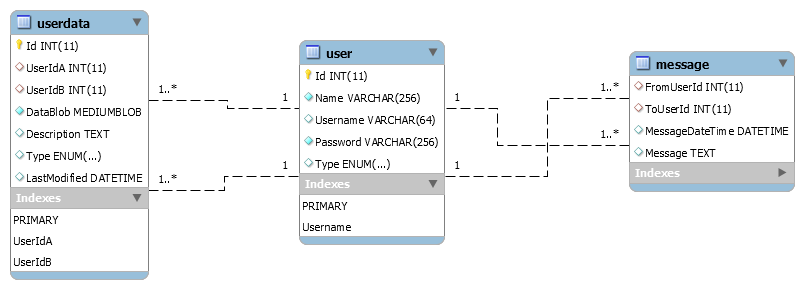
This class diagram illustrates the Abstract Factory Method design pattern used for building objects which manages the dates persistently.



This class diagram is a portion of the system which illustrates the Model-View-Presenter (MVP) design pattern:



# Data Model



# Unit Testing

Unit test were written for the *dataModel* package, which is the most important part of the application.

# Elaboration – Iteration 2

# Architectural Design Refinement

To implement this project, I used the Layers, Client-Server and Model-View-Presenter architectural designs. The Server was designed as a Java application running on localhost, port 8080 and serving HTTP Requests. The client is a Java desktop application which uses JavaFX and call methods from the server via HTTP requests. If the request is successful, it will receive a response with code 200 (OK) and the results of the method called. The results are transmitted in JSON format. For serialization and deserialization was used the Jackson library. If the request is not correct, the client will receive an error code and an description of the error. For implementing the RESTful architectural style I used the Spring framework.

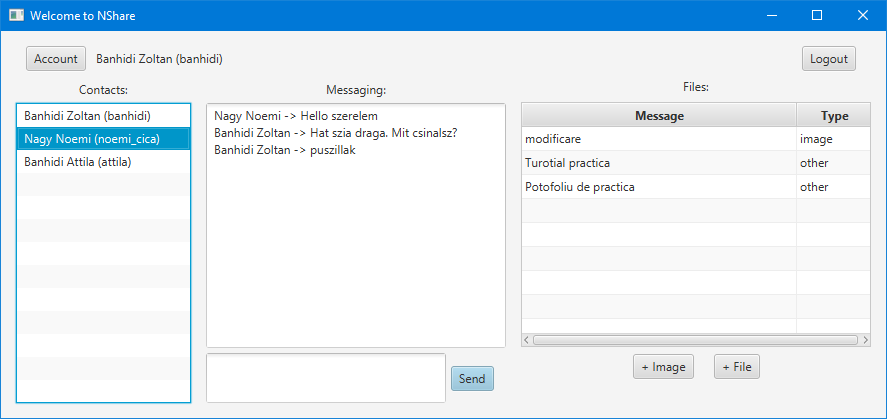
# Construction and Transition

# System Testing

For system testing were used the following methods:

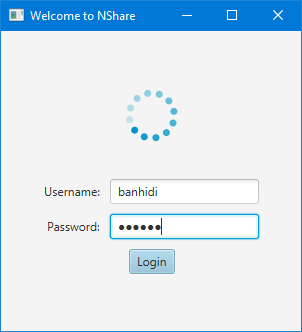
* Unit testing (Junit) for esting the “core” of the application
* Postman was used for testing the HTTP requests
* Manual testing for testing the system in real time and the GUI

In this picture is visible the LoginForm of the application. Introducing a valid username and password and after pressing the Login button or holding down the Enter key results in logging in the application. The type of the user determines which form (RegularForm or AdminForm) is showed.

The RegularForm permits the users to chat with the other users and also to

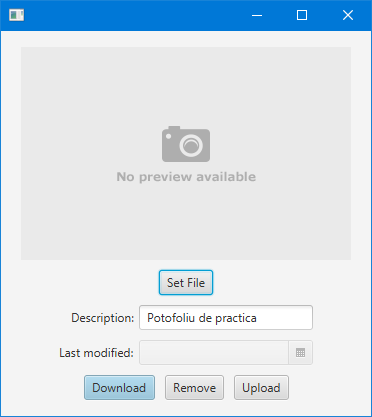
manage (send, remove or modify) shared files between each pair of users.

The files are divided in two classes: images and other files.

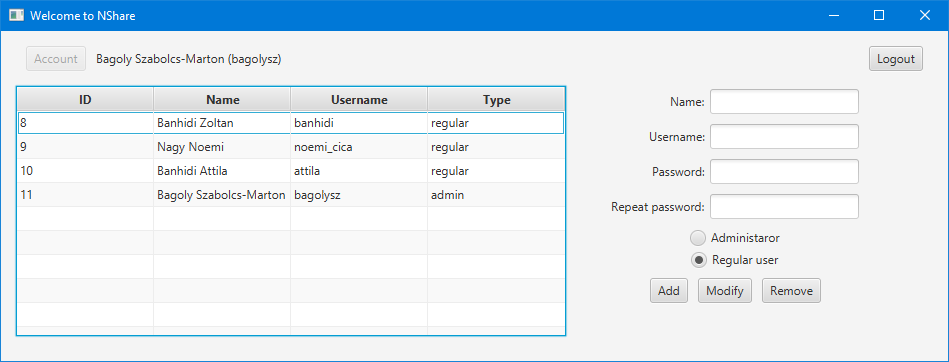


# 

The images can be previewed in an ImageView container. Pressing *Set File* button will appear a file choose dialog and the user can select the image to be shared. Pressing the *Upload* button the picture is uploaded to the server and the window is closed. The *Remove* deletes the current image from the server, and the *Downlaod* button enables to store locally the image.



The window for managing other files has no option for previewing pictures. It functions like the previous one.



The AdminForm has an interface for managing the current accounts in the application.

The administrator can view, edit, delete and add new users to the database.

# Future improvements

The system can be inproved by adding some new functionalities, like sharing files or chat in groups, increasing the maximum file size. Also should be improved the security of the application (the HTTP methods).

# Bibliography