<Chat Application >

Analysis and Design Document

Student:Ciontu Mihail & Pantea Paul

**Group: 30233**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <24/04/19> | <1.0> | <details> | <Ciontu Mihail & Pantea Paul> |
| <23/05/19> | <2.0> | <details> | <Ciontu Mihail & Pantea Paul> |
| <05/06/19> | <3.0> | <details> | <Ciontu Mihail & Pantea Paul> |
|  |  |  |  |

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

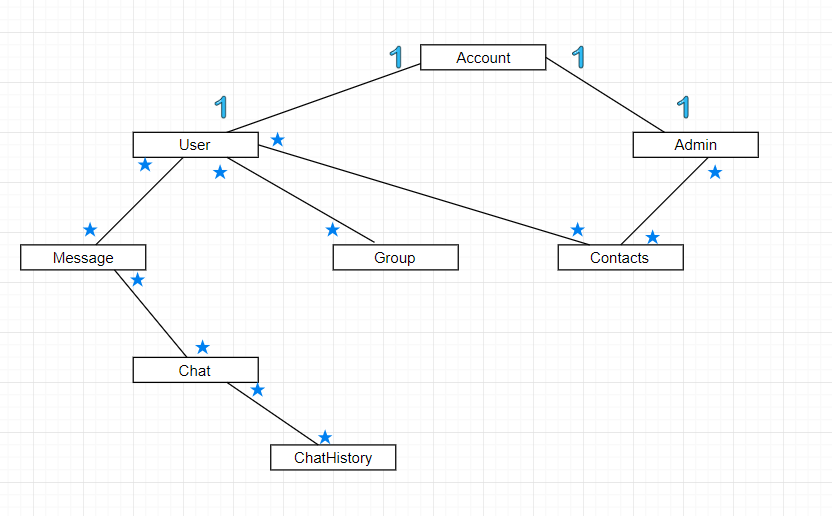
VI. Bibliography 5

# Project Specification

Software Design Project is a java application named "Chat Application". This will be a application which can be used by everyone who wants to chat with one or more people .A user can register in the system, log in the system, add a new friend, view all the friends profile and delete friends.This application is implemented in Java 8 using SpringBoot for back-end and React from front-end application.

# Elaboration – Iteration 1.1

# Domain Model



# Architectural Design

## Conceptual Architecture

The pattern that I used to implement the application is Layer Pattern Architecture, it is used to separate the classes into 3 main categories, presentation, business and database access.

Equipped with the knowledge of the layers to create, the relationships between them and the essence of the architecture, we are ready to implement it. As most of you probably expect, we will slice the system into layers by creating a separate package for each of them. When it comes to applying the dependency and separation rules, things are not so obvious. One could try putting each layer in a separate Maven module, but then capturing the weird relationship between domain and persistence would not be easy. I usually stick with packages and use common sense along with code reviews to make sure that none of the rules are broken.

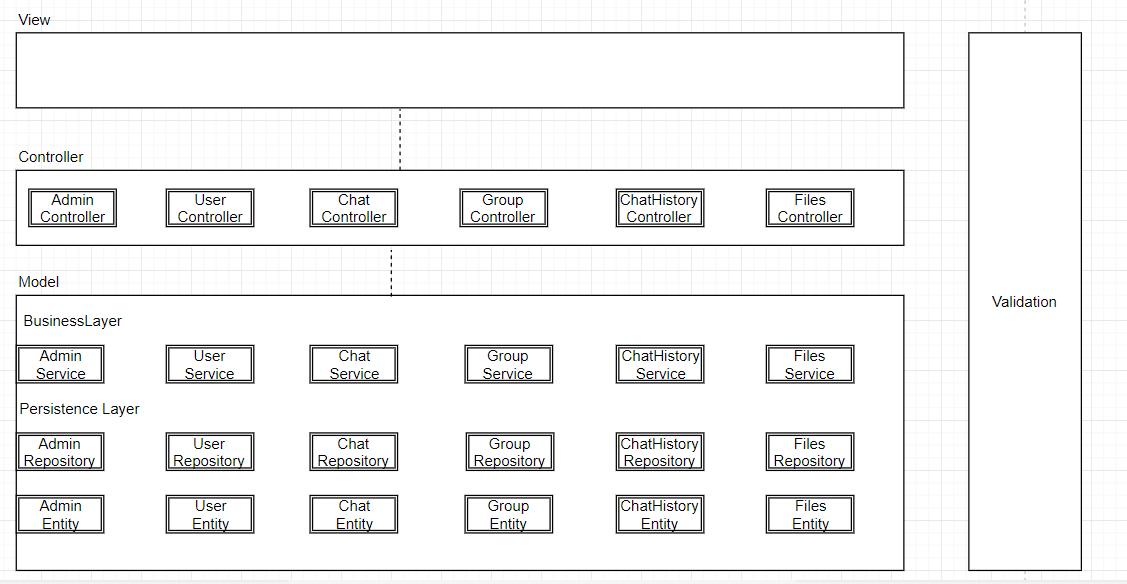
MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's

concerns.

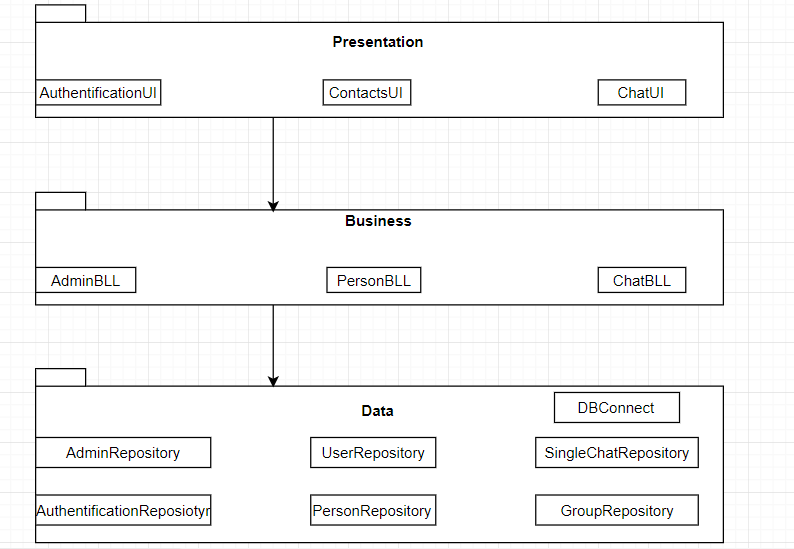
**Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

**View** - View represents the visualization of the data that model contains.

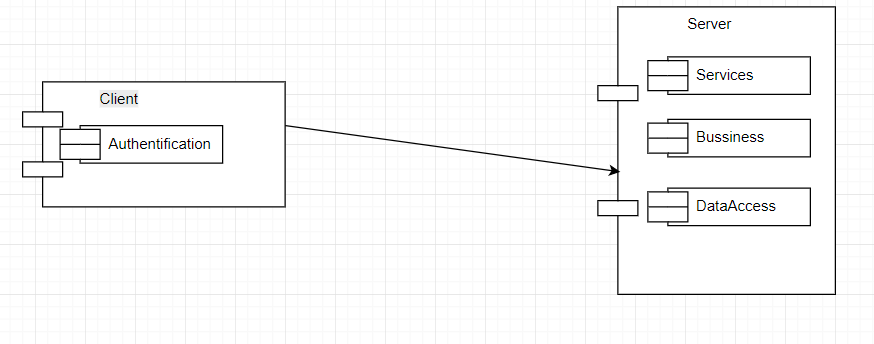
**Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.



## Package Design

**

## Component and Deployment Diagrams



# 

# 

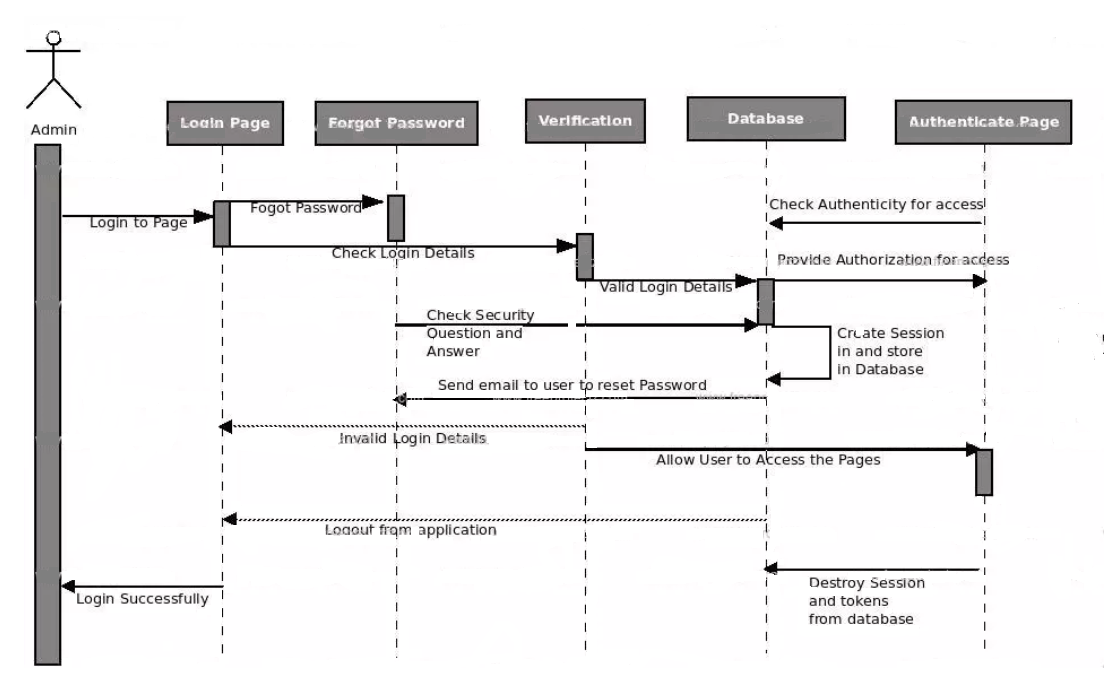
# Elaboration – Iteration 1.2

# Design Model

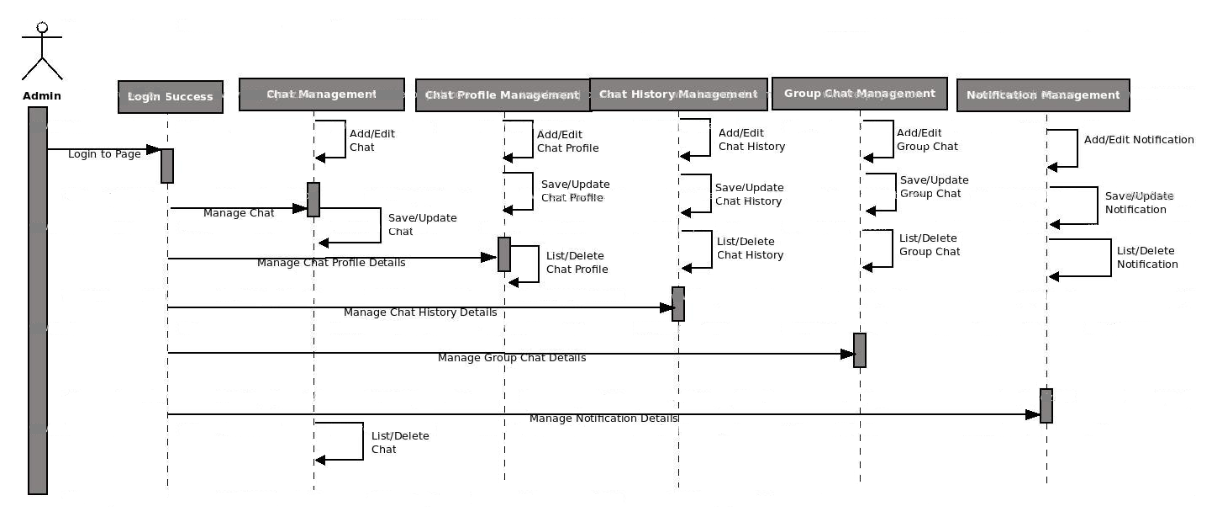
## Dynamic Behavior

This is the UML sequence diagram of Chat Application System which shows the interaction between the objects of Chat Profile, Notification, Delete Chat. The instance of class objects involved in this UML Sequence Diagram of Chat Application System are as follows: Chat Profile Object , Chat Object, Notification Object, Delete Chat Object.

This is the Login Sequence Diagram of Chat Application System, where admin will be able to login in their account using their credentials. After Login user can manage all the operation on Notification, Chat Profile, Chat, Delete Chat etc. The diagram below helps demonstrate how the login page works in a Chat Application System. The various objects in the all the operation on the chat – interact over the course of the sequence, and user will not be able to access this page without verifying their identity.



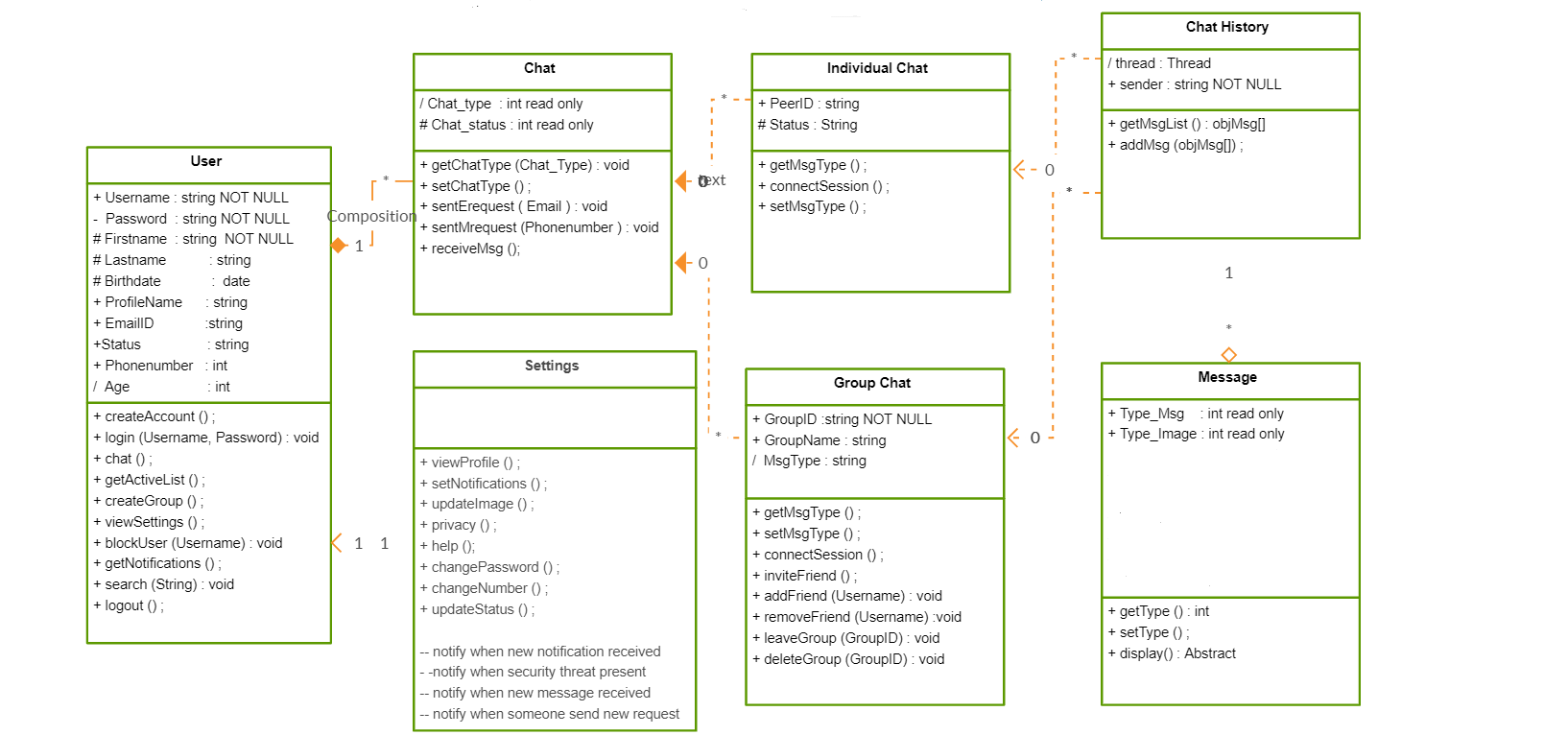
This is the UML sequence diagram of Chat Application System which shows the interaction between the object of Operations Chat:



## Class Design

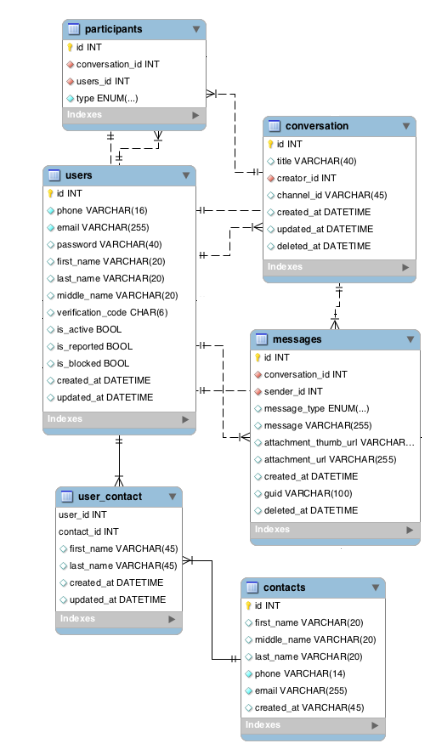
Design of this project is structured using Layers and Model-View-Controller architectural patterns. Considering this, it is needed a separation on layers of classes and also their relations.

In my project builder is used in model package in order to build all the objects like Users, Chats,Groups etc. The implementation of this pattern is easy. You need to create a new object using a constructor default, and implements all the setters for every single attribute in the class (setName, setPassword, etc ) and the last step is to implement a method build() witch return your student.



# Data Model

# The data model is a representation of the project and entities ad all relations between themselves.



# Unit Testing

JUNIT is a Regression Testing Framework used by developers to implement unit testing in Java, and accelerate programming speed and increase the quality of code. JUNIT test framework provides the following important features: Fixtures, Test suites, Test Runners, Junit classes. JUnit classes are important classes, used in writing and testing JUnits.

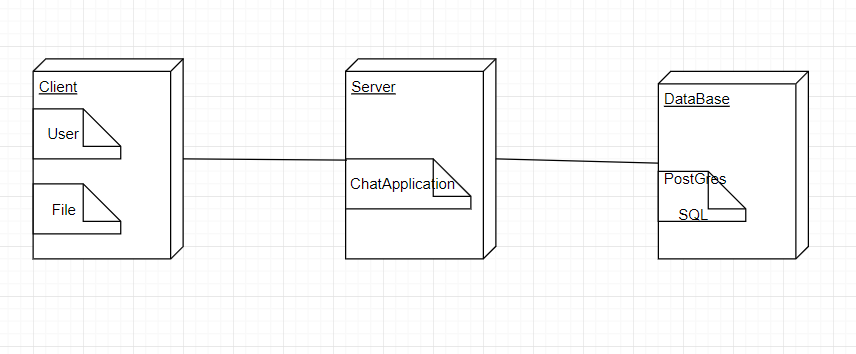
* 1. On Admin side:
* Delete User
* Update User
* Create User
* Block User
  1. On User side:
* Create Profile
* Edit Profile
* Update Profile
* Delete Profile
  1. On Chat side:
* Create Chat
* Leave Chat
* Add person on Chat=>Make Group
  1. On Login side:
* Login/Logout

# Elaboration – Iteration 2

# Architectural Design Refinement

In the conceptual architecture I modified the diagram by adding a send file option, controller and a package for the file classes: file, typefile. Also, in the deployment diagram I added the service for file.

Other than that, the architecture of the project remains the same.



# Design Model Refinement

## The class diagram was extended by adding the Files classes. The domain model was completely modified by adding relationships (1:1,1:n) instead of aggregation, as it was on the beginning.

# Construction and Transition

# System Testing

Unit tests were written in order to test the application and to increase the confidence in the correct functioning of the system

# Future improvements

Multiple features could be added to the product, such as a new type of article, allowing users to send files.

Each of these possible improvements have been accounted for in the design phase and the application should be easily extendable.

More tests could be written to make sure that the application is properly functioning.

# Bibliography

<https://spring.io/guides/gs/spring-boot/>

<https://reactjs.org/tutorial/tutorial.html>

<https://javalin.io/tutorials/websocket-example>

<https://www.callicoder.com/spring-boot-websocket-chat-example/>