Web Chat Application

Analysis and Design Document

Student: Pop Cristian Constantin

**Group: 30235**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <07/Apr/17> | <x.x> | Iteration 1.1 | Pop Cristian |
| <05/May/17> | <x.x> | Iteration 1.2 | Pop Cristian |
|  |  |  |  |
|  |  |  |  |

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

The purpose of this project is to design and implement a chat application that can be accessed by the users by using a web browser. The application should allow sending multiple types of messages including text messages, messages containing images and file attachments. The application will allow users to belong to multiple conversations, each conversation containing multiple users. When a user sends a message to a conversation every user associated with that conversation should see it.

The application should use a database that will be used for two purposes:

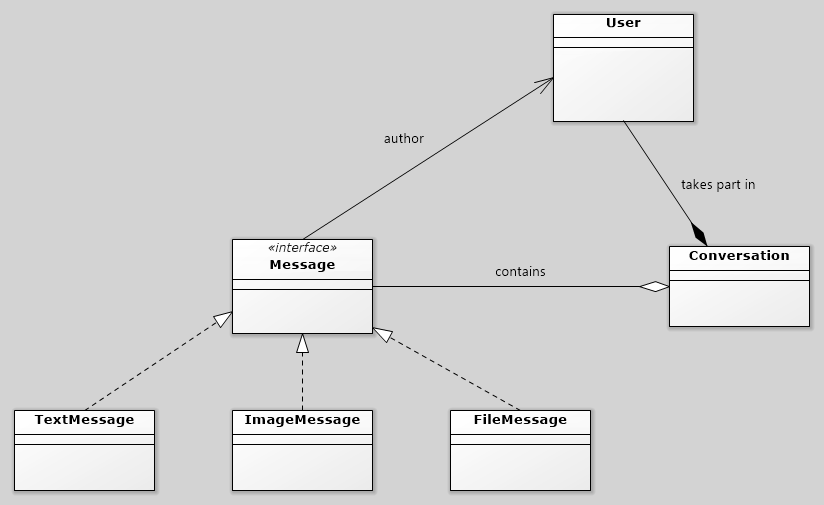
* Storing information about users and the credentials that the user should use for logging in
* Storing messages sent by the users

# 

# Elaboration – Iteration 1.1

# Domain Model

Below is a conceptual class diagram that describes the following:

* There will be multiple kind of messages and each type will be handled differently
* A conversation will contain multiple messages. These messages will be considered owned by that conversation
* A conversation has multiple users that take part in it. New users can be added or existing ones removed

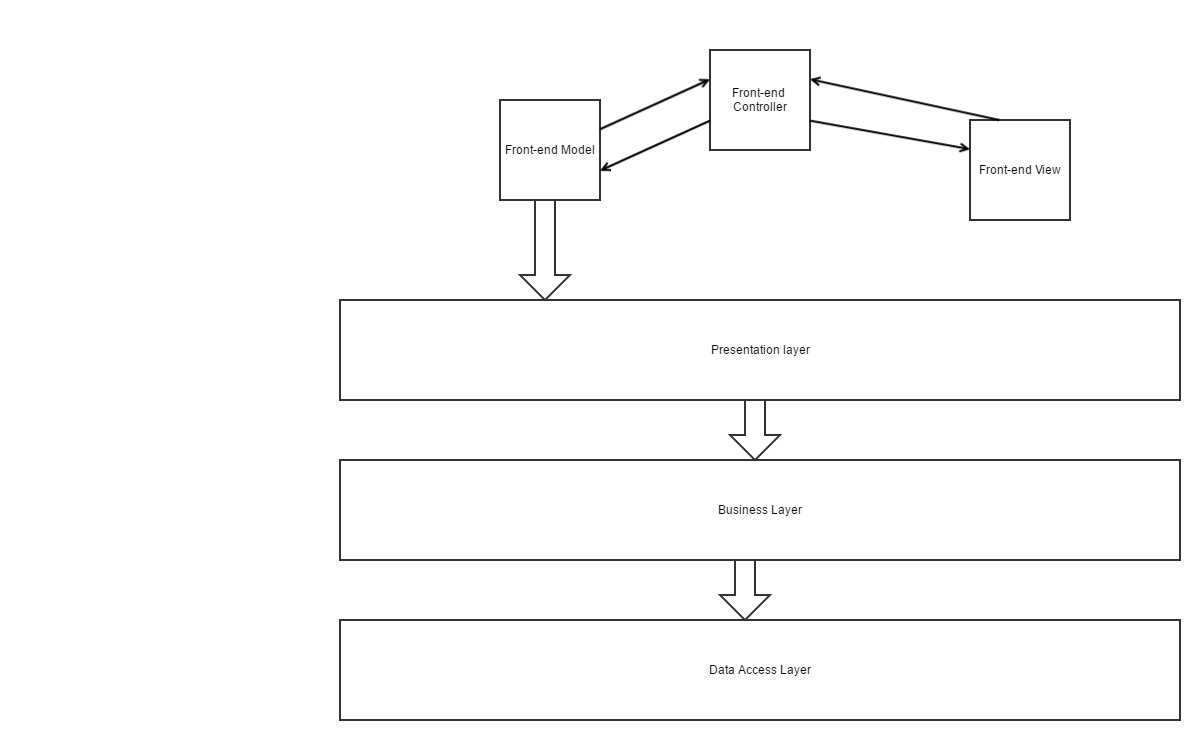
# Architectural Design

## Conceptual Architecture

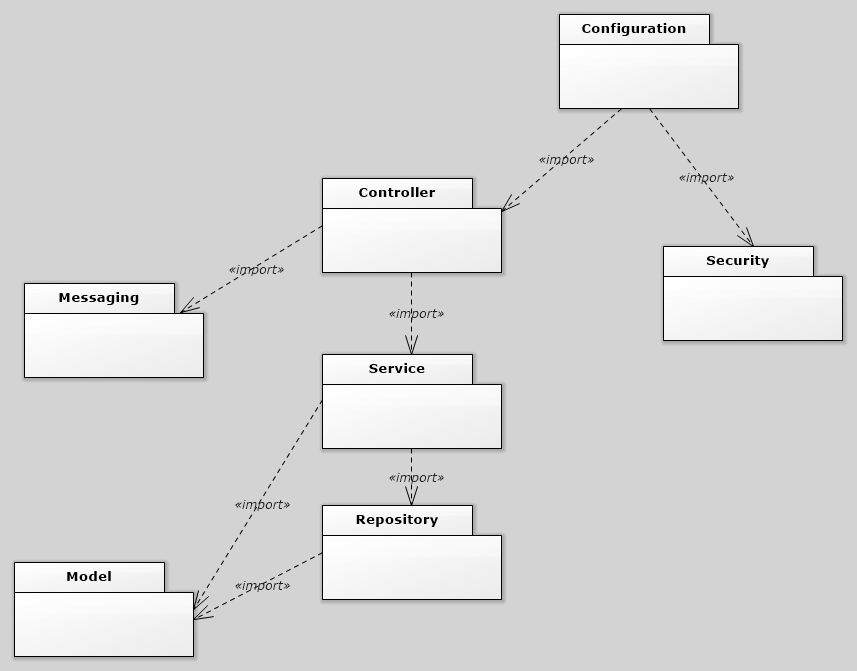
The system will use a client-server model with the Client running inside the user’s web browser and the server being on a separate machine responding to the client’s requests.

The architectural pattern used will be the Layers pattern. This will split the application into multiple isolated layers that contain different parts of the application. The main layers of this system will be:

* Data Access Layer – contains the logic used to access the database and provides data for other layers
* Business Layer – contains the logic of the application that describes how transactions should be performed and how objects should be handled
* Presentation Layer – contains the logic used to communicate with the client. It takes inputs from the user and processes them by calling services from the Business Layer and then returns a reply for the user with the result of their actions

The application will also have some components that might span across different layers. Examples of such components are the ones responsible for security, logging and exception handling.

## Package Design



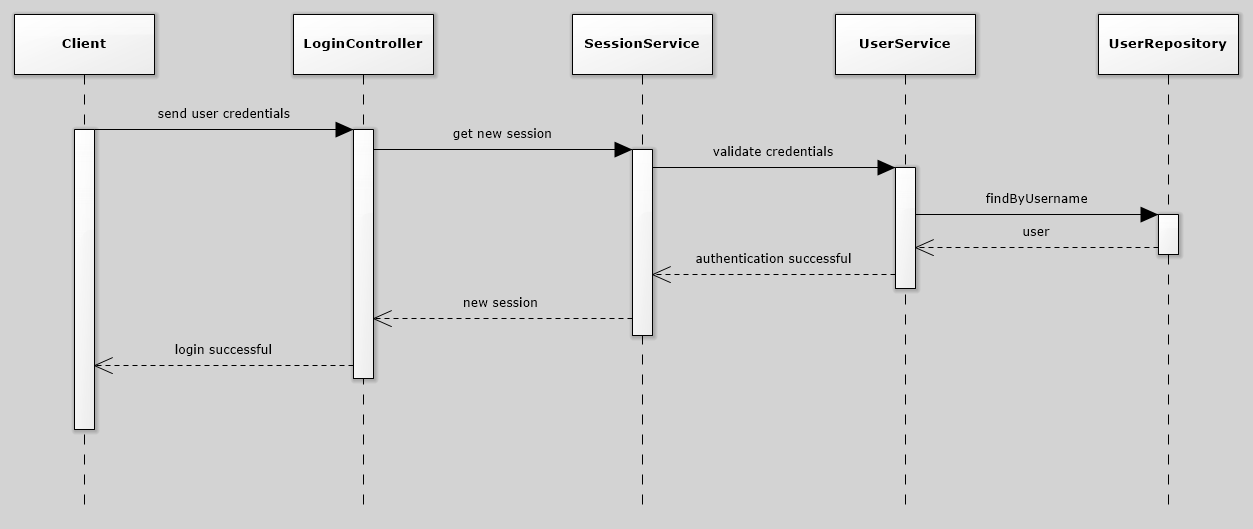
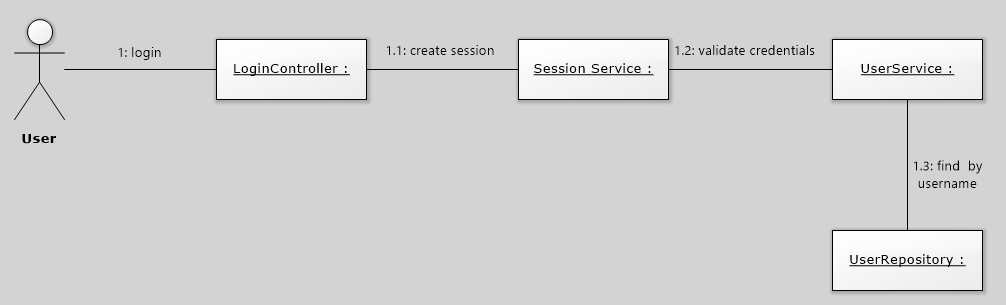
## Component and Deployment Diagrams

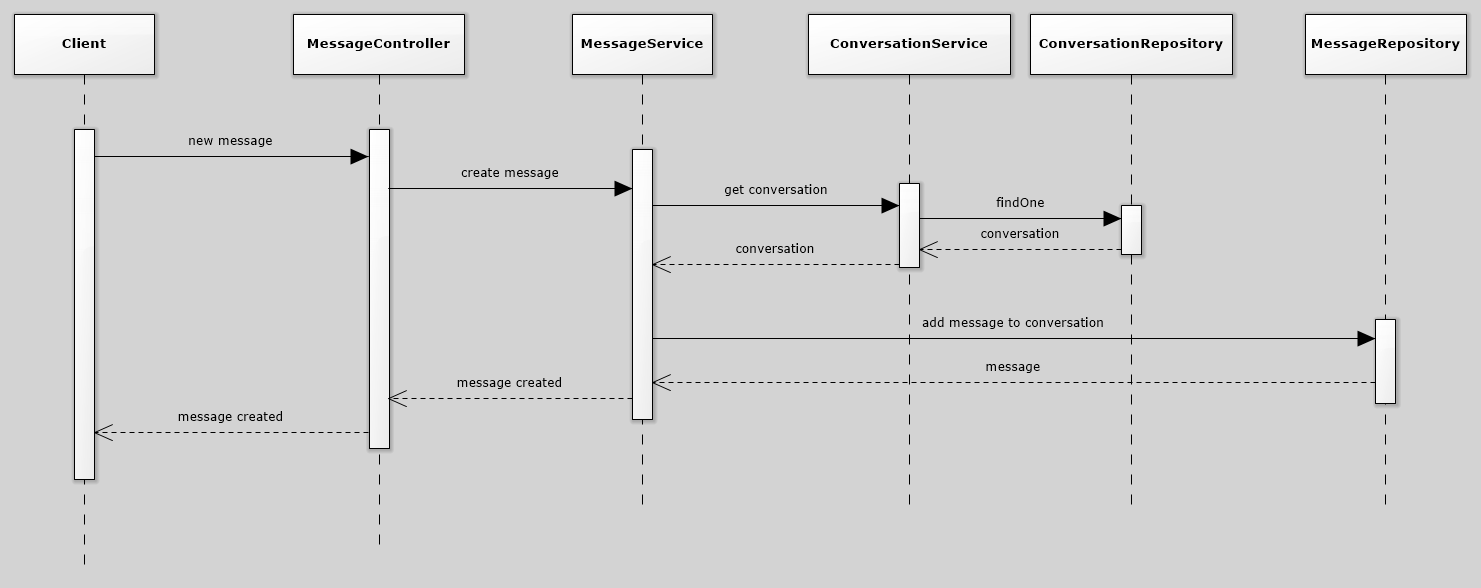
# 

# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

Scenario 1: User logging into the application

Scenario 2: User adding a new message

## Class Design

The GoF patterns that will be used:

* Factory Method – used to make the creation of message objects easier. The services would not have to know any details about the creation of these objects (for example how each concrete message is instantiated)
* Observer – the application will use a publisher/subscriber model to send out messages to its users
* Builder – used to make the creation of objects easier by not having to use constructors with a large number of parameters. All the model classes (except Message because there will be a factory method for this purpose for messages) will have a builder associated with them.

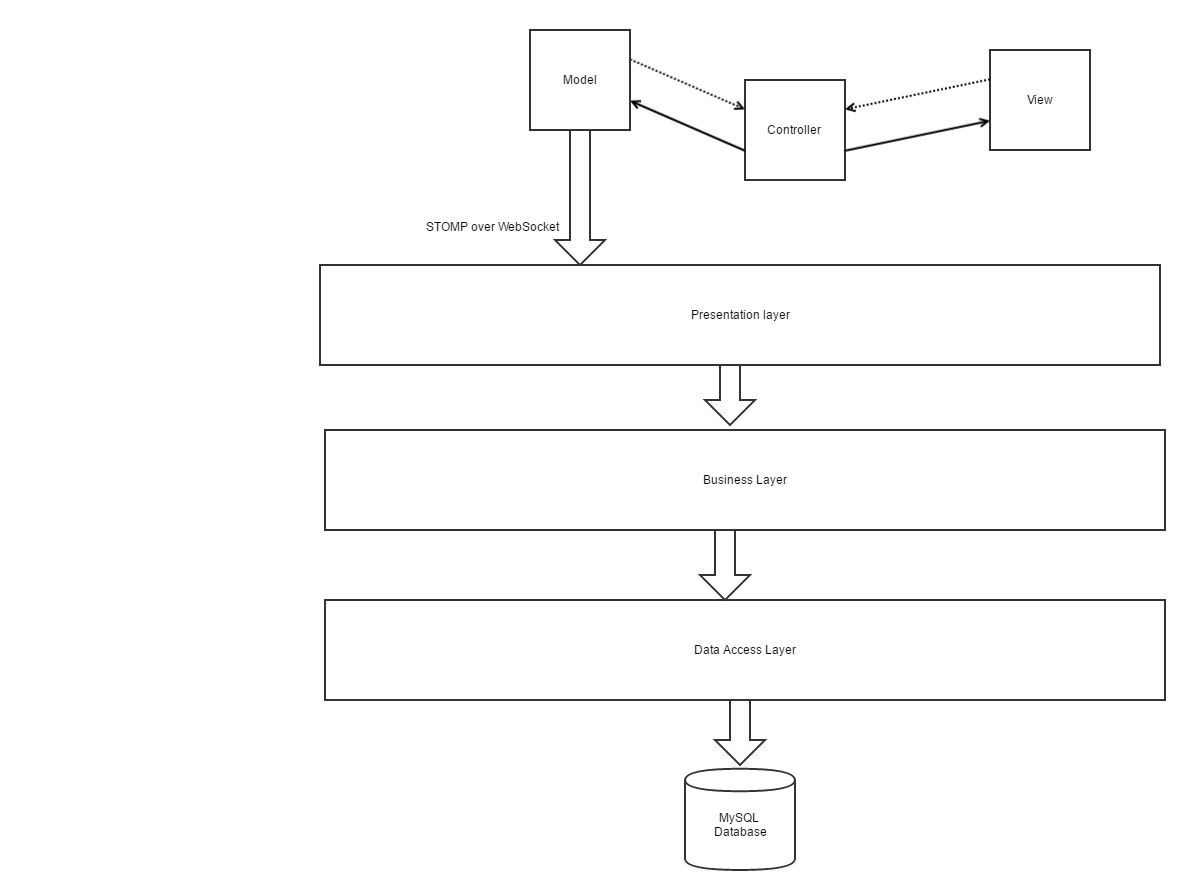
# Data Model

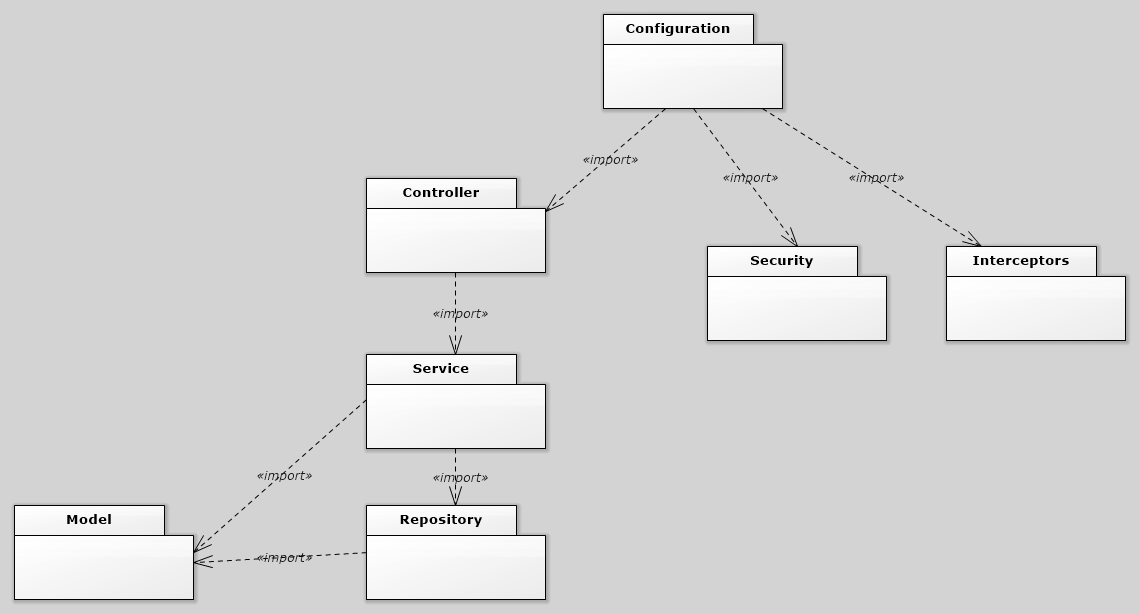
# Unit Testing

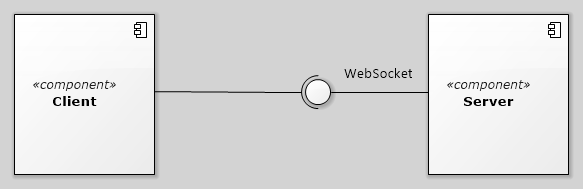
The important services from this project will be tested by creating unit tests with Junit4 and SpringBootTest. The unit tests will contain tests where it is expected that the operation will be performed successfully and tests where the operation should fail. In the last case testing will ensure that the system will not let abnormal data be inserted into the database or sent to other users and that it will respond with the expected exception or error message. Also, the tests should be ran on a different spring profile that uses different resources from the regular profile*.*

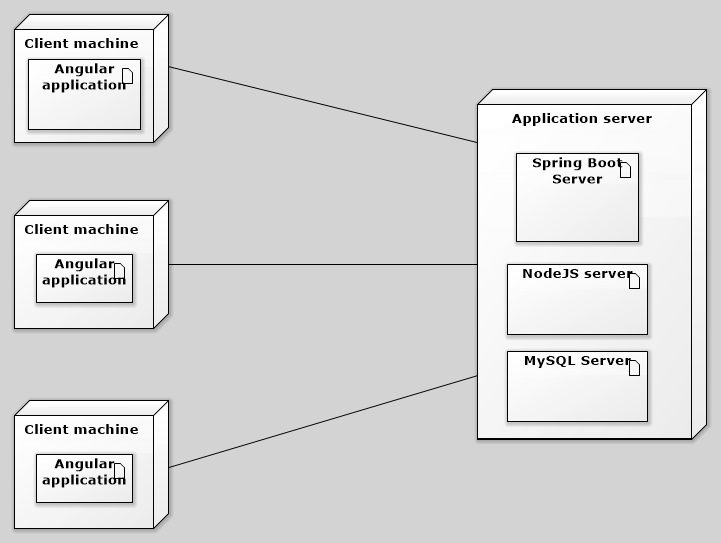
# Elaboration – Iteration 2

# Architectural Design Refinement

 The conceptual architecture diagram was changed to also include the database and the protocol used for communications between the client and the server (STOMP over Websockets)

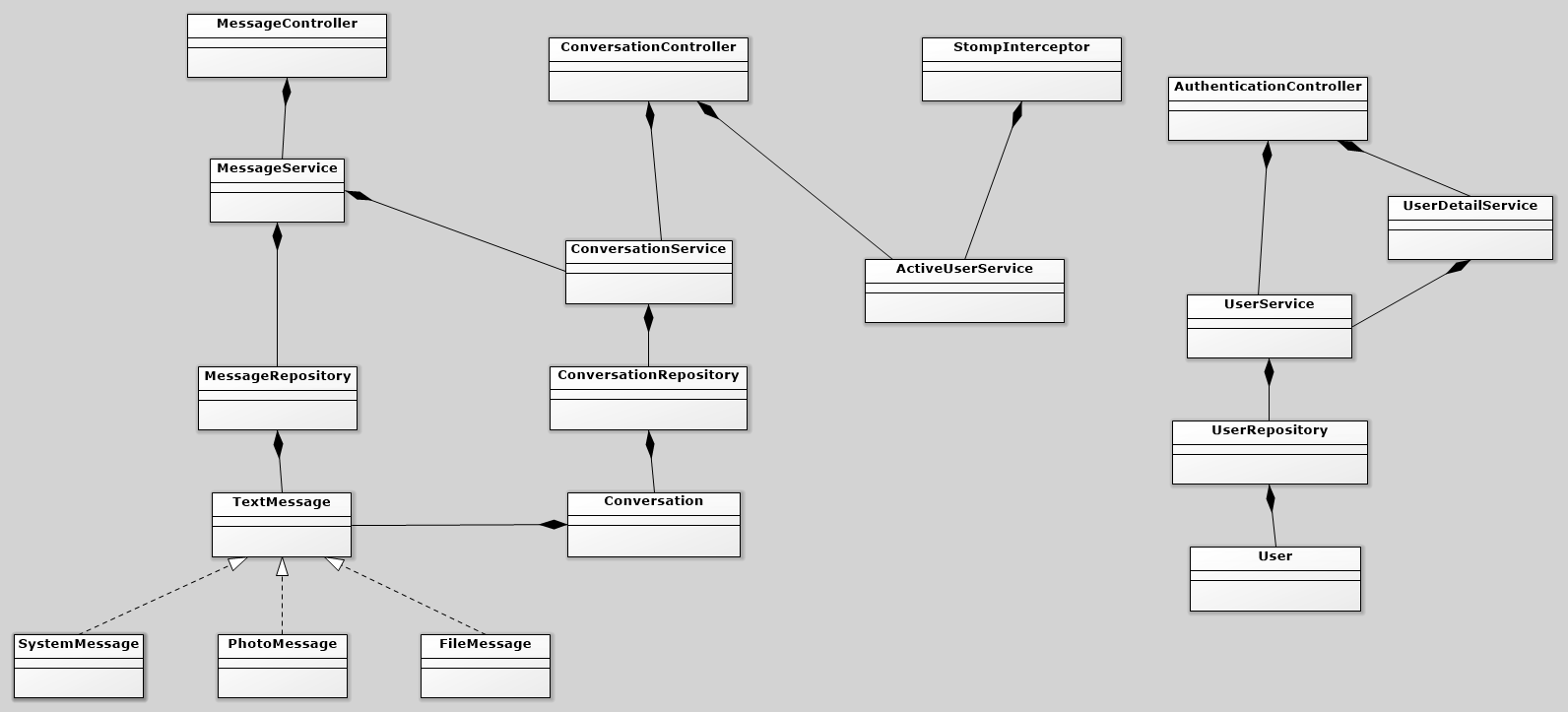
 The package diagram was updated to better reflect the actual application. The messaging package was removed since that feature is implemented by Spring and it was replaced by the Interceptors package which is used to attach callbacks to certain events like users connecting or disconnecting.

The component diagram is unchanged

The deployment diagram was updated to include the NodeJS server artifact on the server node.

# Design Model Refinement

## The updated class diagram

****

The factory method used for generating messages was removed since the Builder pattern was a better choice for this particular scenario.

Also, TextMessage is now a base class since the information stored in that class is general and applies to all its subclasses and it has 3 subclasses: ImageMessage, FileMessage and SystemMessage.

A new component is the ActiveUserService. That component keeps track of the logged in users and their sessions. When a user closes all its sessions they are eliminated from the list containing active users. The component is updated by two StompInterceptors that listen for Stomp Connect and Stomp Disconnect events.

# Construction and Transition

# System Testing

Unit tests were performed on the services of this system by using SpringBootTest and Junit4. The test were ran by using a different Spring profile (the “test” profile, as opposed to the “production” or “default” profile) which used a H2 in-memory database that was re-created every time the tests were ran. The “production” profile used a MySQL database where data was kept for persistence.

# Future improvements

Some improvements that could be made for this system:

* Allow users to have a customizable profile that could contain personal information, a profile picture, a list of friends, etc.
* Allow users to have different roles in conversations as right now every user from a conversation is equal
* Allow the restrictions on file size and file extensions to be set dynamically by an administrator
* Allow administrators to define profanity filters for messages sent on all conversations
* Add notifications for users. They would be notified when a message from a conversation (except the one they are currently seeing) arrives or when a user adds or removes them from a conversatsion

# Bibliography

<https://cli.angular.io/>

<https://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single/>

<https://auth0.com/blog/securing-spring-boot-with-jwts/>

<https://angular.io/docs/ts/latest/quickstart.html>

<https://getbootstrap.com/>

<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-testing.html>

<https://docs.spring.io/spring/docs/current/spring-framework-reference/html/websocket.html>

<https://spring.io/guides/gs/messaging-stomp-websocket/>

<https://github.com/stomp-js/ng2-stompjs>

<https://www.sitepoint.com/implementing-spring-websocket-server-and-client/>