**Prepared by: Tangqi Feng and Eduards Vagals**

**For the attention of: Brian McGinley and Kevin O’Brien**

**Candidates No: G00329372 / G00315079**

**Program: Bachelor’s in Software Development Year 3**

**Module: Proffesional Skills in IT**

**Submission Date: 24th April 2017**

# Table of Contents

[Table of Contents 2](#_Toc480405778)

[Introduction 3](#_Toc480405779)

[Technologies and Architecture 4](#_Toc480405780)

[Service Side Technologies 4](#_Toc480405781)

[Client Side Technologies 5](#_Toc480405782)

[Service Side Architecture 5](#_Toc480405783)

[Client Side Architecture 6](#_Toc480405784)

[Limitations 8](#_Toc480405785)

[Known Bugs 9](#_Toc480405786)

[Recommendations for Future Development 9](#_Toc480405787)

[Conclusions 10](#_Toc480405788)

# Introduction

The idea of the project was inspired by Micro Services Architecture designs discussed in the following [blog post](http://microservices.io/patterns/microservices.html), where many different APIs, written in different languages, can exchange data.

In our opinion, the concept of Micro Services can save time, money and other important resources spent for development and maintenance. For example, if one Client fails, or there is a major bug issue, there is alternative for the user to use other client application to access Service. There are more advantages of Micro Systems we talk about in recommendations part of a document.

The purpose of this project is to present RESTful back-end architecture that can serve resources to a client applications using Uniform Resource Identifiers (URIs). Most programming languages support HTTP protocol for web communication that makes it possible to develop Client applications using variouse technologies, utilizing their features to fit-best business requirements.

This report presents the designs for two such clients, each of which use one common back-end to obtain data and use it as a means for multi-user communication. Flask Client allows users to access Service via browsers, while JavaFX gives option to use desktop application to do the same.

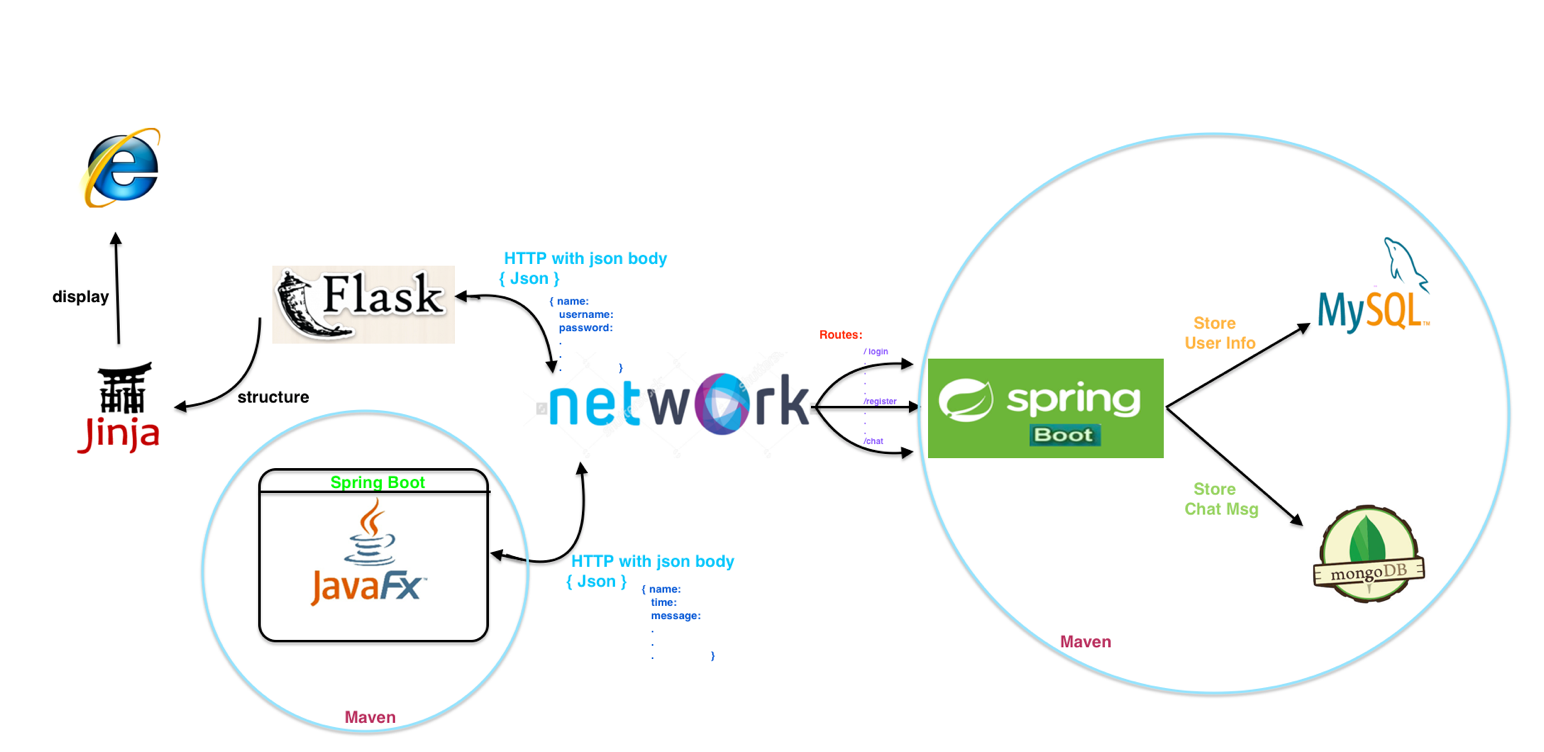
We’ve choosed to develop chatting system as a subject for trying out Micro Services Architecture.

Our projects show only a part of what architecture is capable of, i.e., having multiple clients integrated into system, written using different APIs.

Other major aspect of such architecture is in splitting back-end service into independent modules (micro services) that are loosely coupled with each other on network, but involves more complexity and technical background in developming it.

This design document is a guide on how to start developing your own Micro System and serve as a good foundation for future projects.

# Technologies and Architecture



### Service Side Technologies

#### *Spring Framework*

The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications – on any kind of deployment platform. A key element of Spring is infrastructure support at the application level: Spring focuses on the “plumbing” of enterprise applications so that programmers can focus on application-level business logic, without unnecessary ties to specific deployment envitonments.

**Features**

* Dependency Injection and control flow
* Spring MVC web application and RESTful web service framework
* Foundational support for JDBC, JPA and NoSql Databases
* And more…

All available features and modules are described in the [Modules section of the reference documentation](http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html#overview-modules) for Spring.

There are several other competing frameworks like Java EE(JSF) or even Flask (we used it for different purpose this time) that can provide Service similar way.

During a last semester, we have developed applications using Java EE and Flask frameworks, so we picked Spring framework to expand our knowledge in modern technologies.

#### *MySql Database*

We are using MySql database to store user details for it’s convenient way of storing data in tabular fashion, security features it provides and its capabilities to expand into more complex database, as it supports relationships between data tables.

Currently we have one table to store registered users.

#### *MongoDB*

MongoDB is a document based database that we are using to store a chat messages. It provides convenient data structure for our chat messages that can be easly reflected to a POJO or a Json document because of it’s common structure.

#### *Maven package builder*

To glue things up, we use Maven package builder to handle necessary dependencies required for the project and it also is an easy way to share the project among the team members, as it uses a common package structure and easly imported to a workspace of common IDEs.

All the dependencies are listed in a *pom.xml* file in a root folder of a project.

### Client Side Technologies

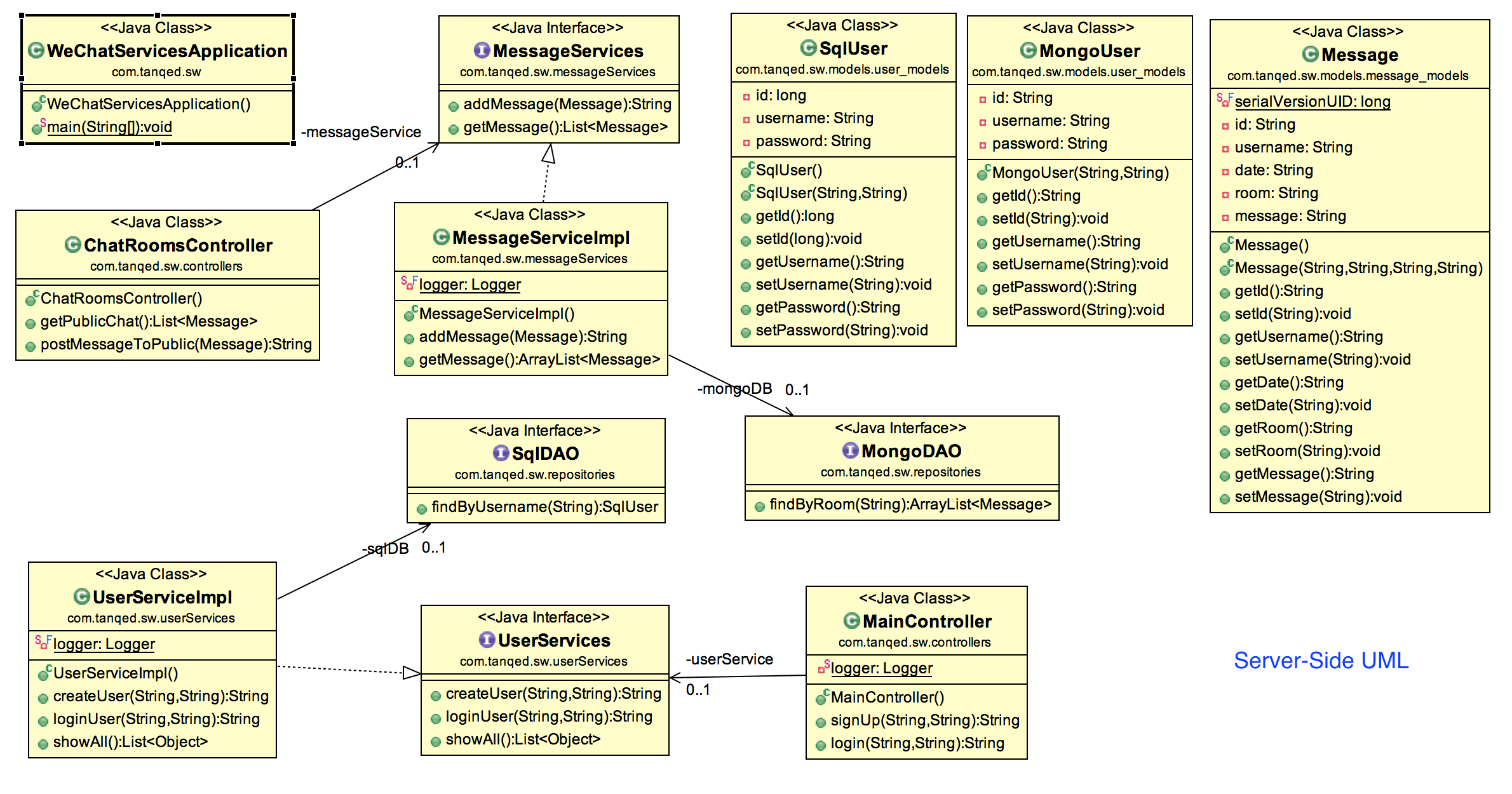
#### *Browser Client: Flask (python)*

Flask is a small and powerful web framework for Python. Flask  is called a micro framework because it does not require particular tools or libraries. It's easy to learn and simple to use, enabling you to build your web app in a short amount of time.

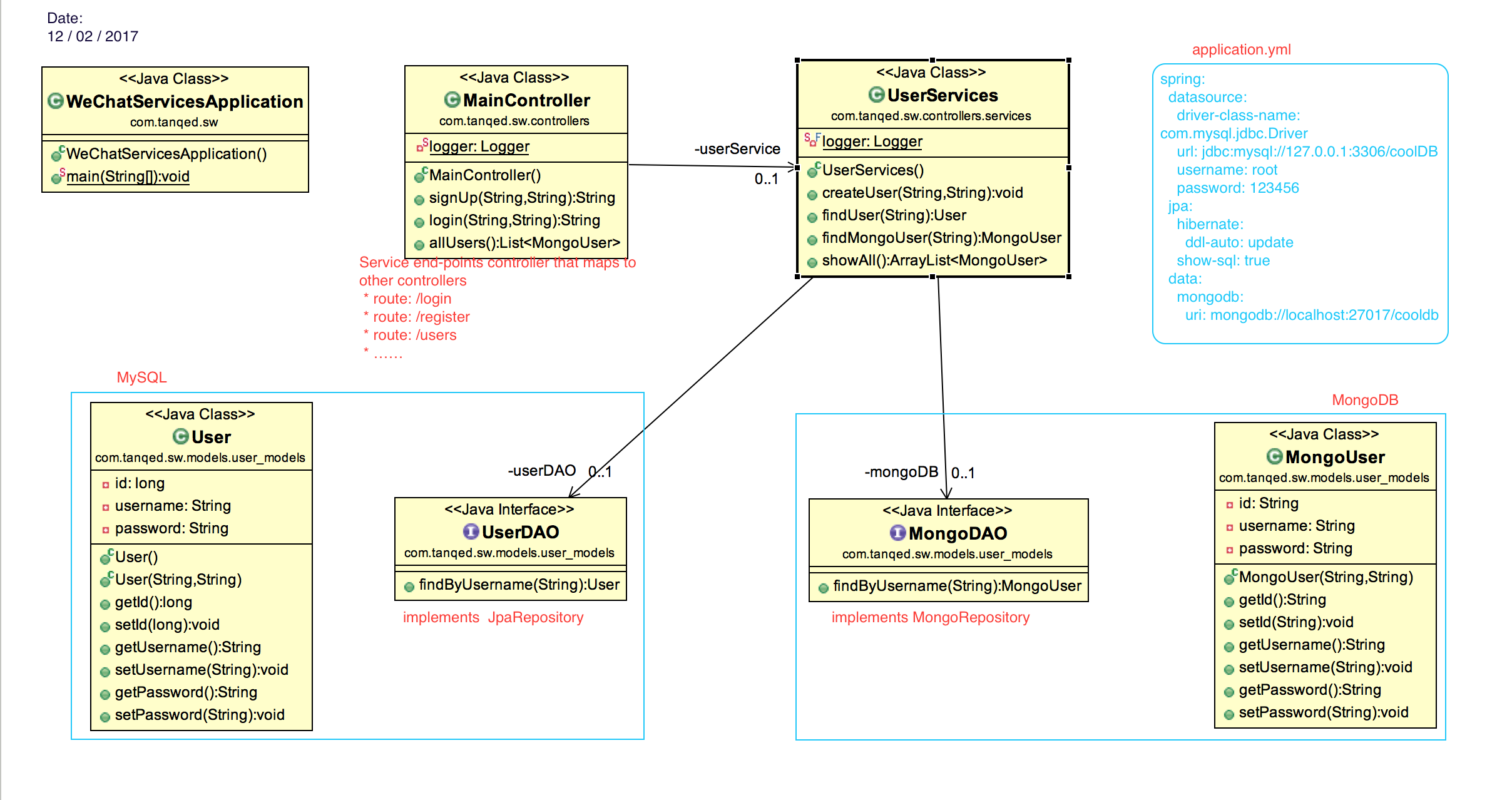
Jinja2 ( Flask’s default template engine ) is a [template engine](https://en.wikipedia.org/wiki/Template_engine_(web)) for the [Python programming language](https://en.wikipedia.org/wiki/Python_(programming_language)) that is designed to be flexible, fast and secure. The Jinja template engine allows customization of tags, also Jinja allows the template designer to call functions with arguments on objects.

#### *Desktop Client*

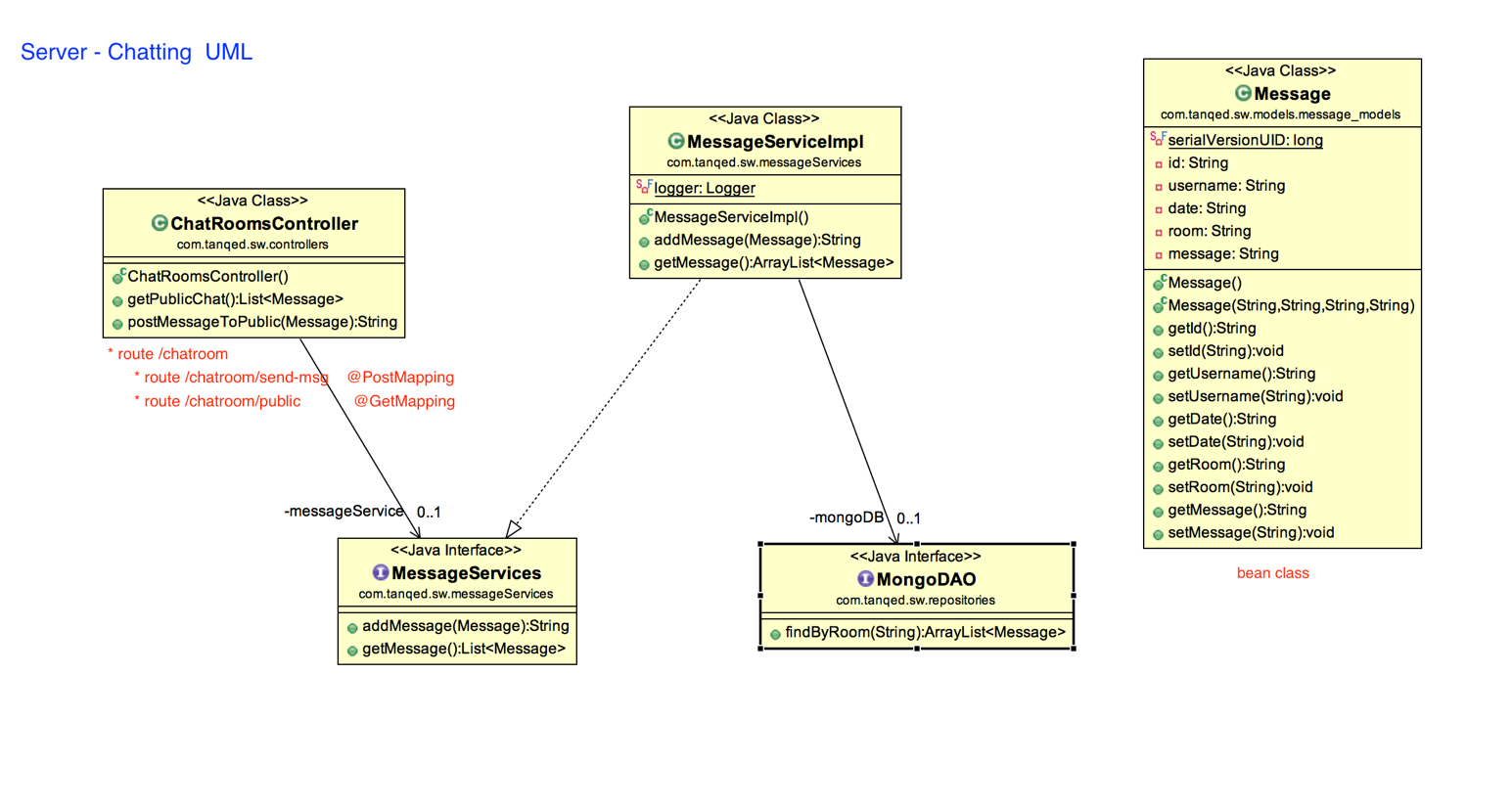
### Service Side Architecture



Log-in & register part:



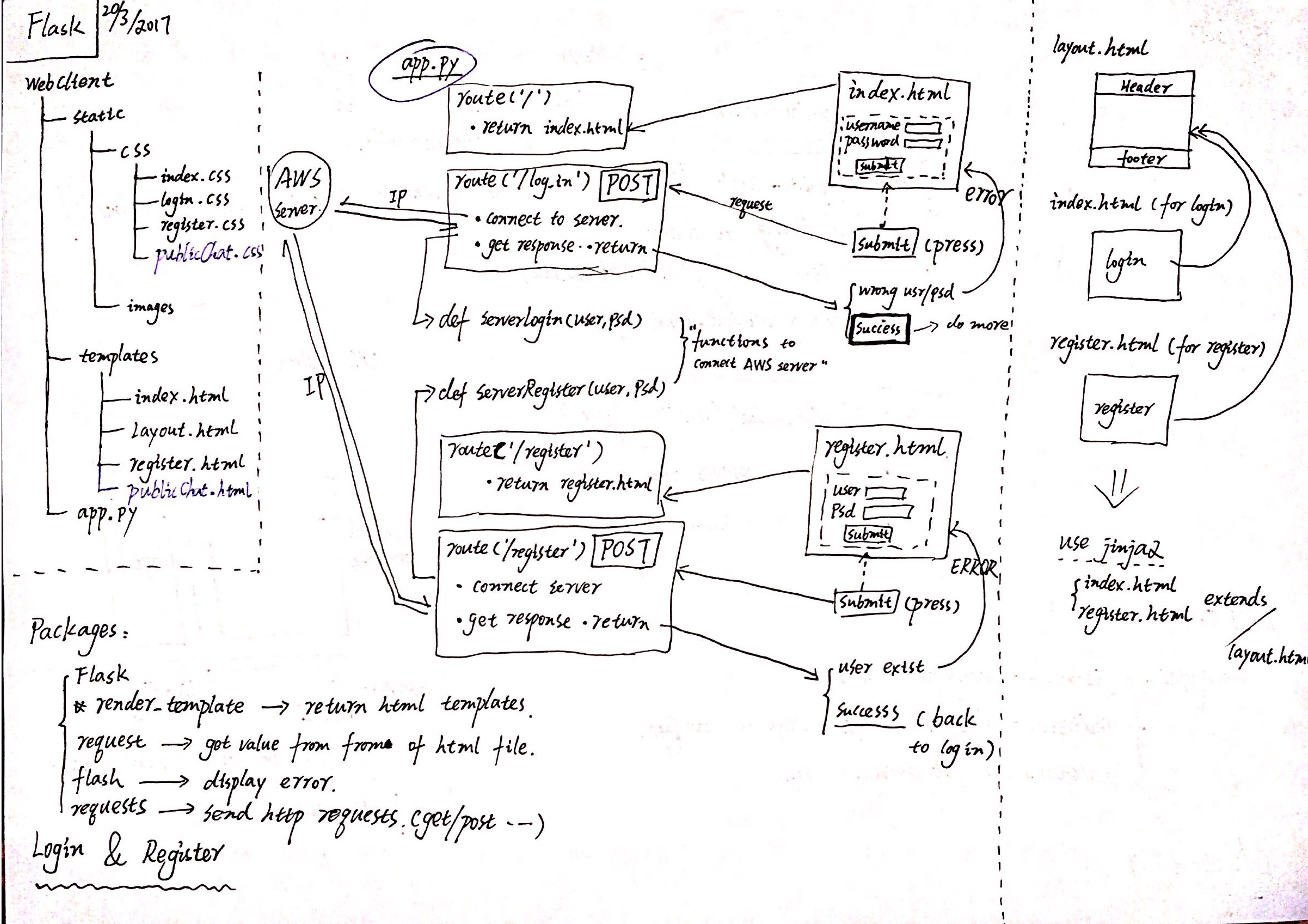
Chatting part:



### Client Side Architecture

#### *Browser Client*

This flask project contains *app.py* file, static folder ( css files and images ) and templates folder (html files)



Note: the Server is always running in the AWS (Amazon Web Services)

For this log-in & register part, some packages are be used:

* Render\_Template

To render a template, all you have to do is provide the name of the template and the variables you want to pass to the template engine as keyword arguments. Every html file will be returned correctly by this method.

* Request.Form

The Form collection retrieves the values of form elements posted to the HTTP request body, with a form using the **post** method. This method will transmit the data you typed like username, password to the back-end.

* Flush

Flask provides a really simple way to give feedback to a user with the flashing system. The flashing system basically makes it possible to record a message at the end of a request and access it next request and only next request. Here it is used to display the error messages like “wrong username/password”, “user existed” etc.

* Redirect

It returns a response object and redirects the user to another target location with specified status code. When you logined successful it will redirect you to public chat page.

* Requests

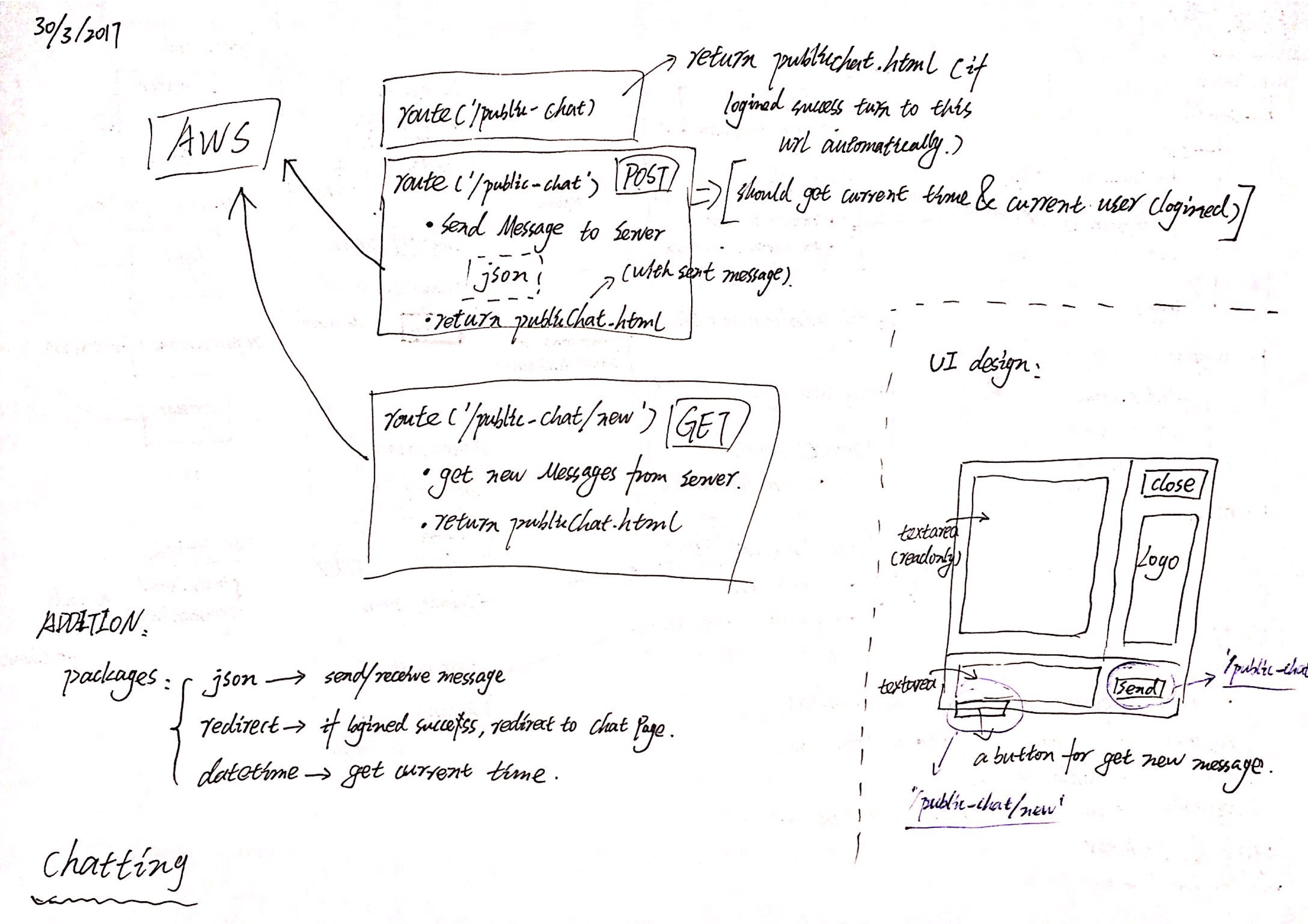
Handle different HTTP methods. For example, using *requests.get* method to send username/password to the server for log-in, and *requests.post* method for register communications.

The home page ( ‘/’ ) is the login page. From this page, users can go public chat page if logined successful, Or go to register page to create an account.

**Functions**:

ServerLogin(user,psd) & login\_get() ---- send get method to Server for checking valid account

Se**rve**rRegister(user,psd) & register() ---- send POST method to Server for creating account



For this chatting part, some extra packages are be used:

* Json.loads()

This method is used to parse the json object (chat messages) from Server.

* Datetime

The datetime module supplies classes for manipulating dates and times in both simple and complex ways. Here it is used to get current time when you send a message, then, add into message (json) objects.

**Functions**:

sendMessage() ---- send json object to Server. Json: {‘username’: ‘ ’, ‘date’: ‘ ’, ‘room’: ‘’, ‘message’: ‘’ }

getMessage() ---- use POST method get message objects, then use json.loads() to parse it.

#### *Desktop Client*

# Limitations

#### *Time frames and learning curve*

During the development cycle of our project, we observed how the requirements have changed from the initial idea to a final product. Biggest impact on these decisions were made by a time we’ve got to accomplish project to a presentable state and a steep learning curve of the new technologies we’ve used for development.

To accommodate this issue, we had to give up some ideas likes file sharing, multiple chat rooms and tokenazation of request/response which adds more complexity to a project. Instead we choosed to address a development of basic, but stable system that can be easly adoptable to all those new features that can be added.

#### *Service Resources*

It is obviouse that any system or service has its limits in how many requests it can handle, if they were to come in bulk in short period of time. Our project is not an acception. Because we are developing a chatting application via HTTP protocol, it means a client must poll (ask service) with new requests to find out if there is a change in chat. If you think of it, an instant messanger would have to make a request every second to catch a change immidiatly and that is only for one user. What about 10 users?

#### *Flask Client refresh new message*

The *flask* client should get new messages from Server automatically, we try some methods like using *python library* and using *ajax* call, but it does not work. Both the two methods can get new messages correctly when it is called, it can be checked easily on the back-end. But we do not find a way to display the new message to the web page, which means it can get new data but cannot display to users. In our opinion, we have use *Jinjia2* template engine to structure the html page, there should be a solution to combine *Jinjia2* and ajax/python requests to display dynamic data. But we did not finish this part at this moment because of limited time. Then, for refreach message function, we just use a botton to call the method instead.

* Creating FX Controller from overloaded constructor

# Known Bugs

#### *Spring Boot and Flask cannot work together*

*Spring Boot* (port 8080) and *flask* (port 5000) cannot communicate to each other if run on same localhost. The problem is : *flask* can communicate with browser and *flask* can communicate with *Spring Boot* partially, but they cannot work together.

For this problem, we hava tried many no-working ways from the internet. We desided to move the server to the AWS online virtual machine, and use ip address to communicate.

As is known, *Spring Boot* project combines tomcat server in it. We need convert it to *war* file and put them into local tomcat server to let it works online if we want to run the project with out the IDE like eclipse and intellij. Currently, we still do not find the way to get the right *war* file, the solution is using eclipse to run the Server on the AWS.

# Recommendations for Future Development

#### *Better Design Plan and Scheduling*

We would definatly recommend investing time into design planning of a project and well thought out scheduling plan for tasks to be accomplished.

Important aspect is gathering of business requirements, what program should be able to do and what client want it to do. Than, from our experience of developing this project, it is much easier to pick a stack of technologies and plan design, and architecture for the project.

Following a schedule improves performance, saves time and money, because developers know exactly what should be done and by when. Development cycle can be broken up into milestones that indicates the state of development cycle.

#### *Desirable functionalities*

* Implement more advanced security for an application to prevent hijacking of sensitive user data, as well as securing database storage on the back-end.
* Introduce tokens into request/response communication to track a state of a client application on the back-end.
* Give users ability to upload or choose an avatar
* Allow users to private message each other and ability to create a group chats

#### *What to improve in existing system*

* Over all, HTTP communication is not suitable for chat application and next time we would consider using web sockets instead. Recent discovery for us was the Spring Framework ability to give foundation for message based applications.

# Conclusions

* Spend more time in researching framework capabilities.
* Learning outcomes
* Would you develop applications in future using these technologies?
* Likes/Dislikes