**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](#_Toc480631041)

[Introduction 3](#_Toc480631042)

[Technologies and Architecture 4](#_Toc480631043)

[Service Side Technologies 4](#_Toc480631044)

[Client Side Technologies 5](#_Toc480631045)

[Service Side Architecture 6](#_Toc480631046)

[Client Side Architecture 10](#_Toc480631047)

[Limitations 14](#_Toc480631048)

[Known Bugs 15](#_Toc480631049)

[Recommendations for Future Development 15](#_Toc480631050)

[Conclusions 16](#_Toc480631051)

# 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.

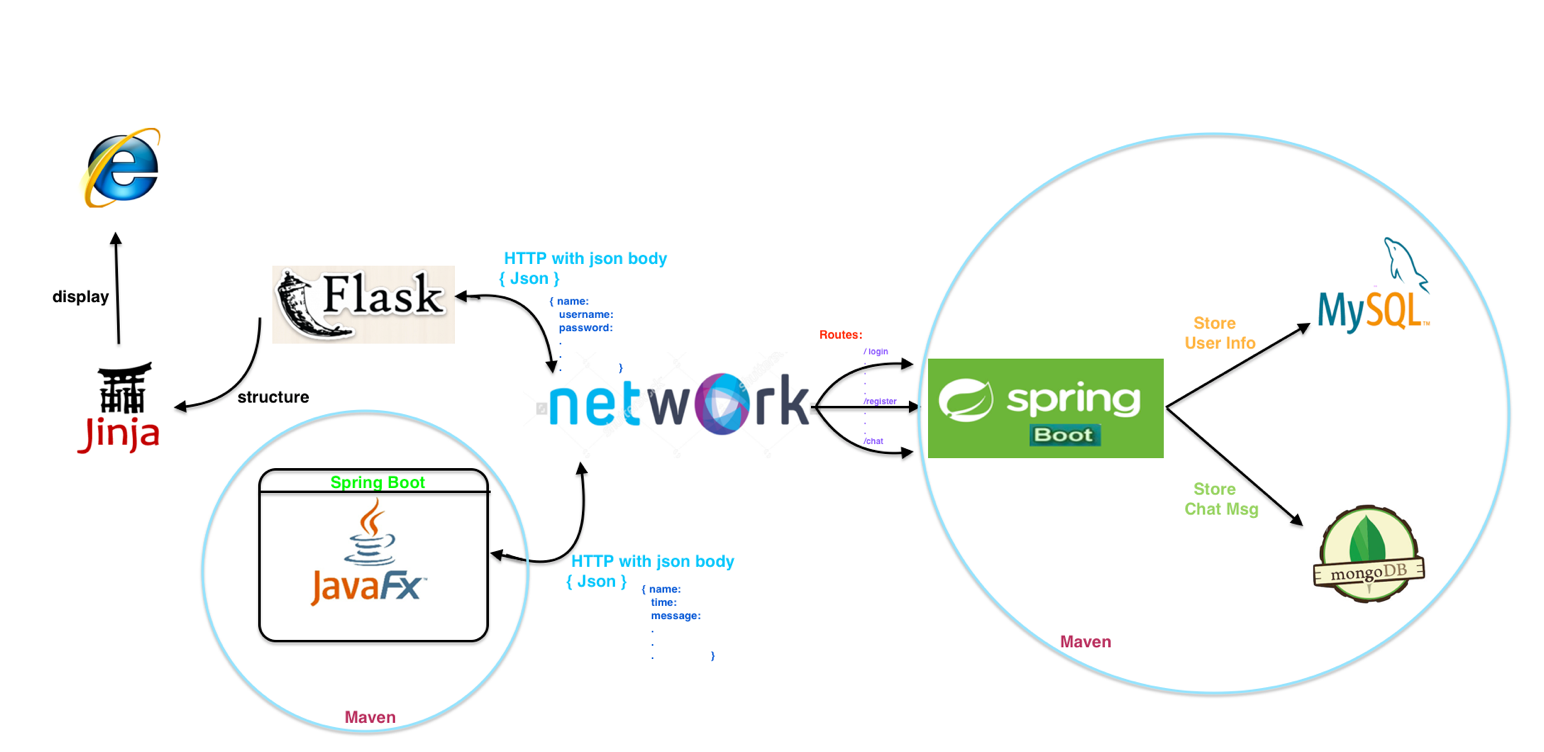
The final product of this project is a Chatting System via HTTP with basic accounts keeping and security, that is used as a subject for trying out Micro Services Architecture.

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 developing 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.

***Lombok Library***

Lombok is one of the tools that is used to avoid repetitive code, also know as boilerplate code.

For example, a Model class usually consists of many properties that require getter/setter methods. To make class tidier, with less lines of code, we can annotate parameters with @Getter @Setter annotations to ommit getter/setter method definition for a property.

For more cool stuff from Lombok, visit their project [website](https://projectlombok.org/features/).

### 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.

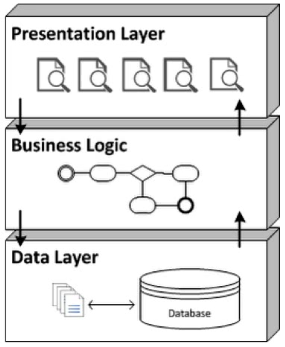
#### *Desktop Client: JavaFX + Spring Boot*

JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms. Written as a Java API, JavaFX application code can use Java API libraries to access native system capabilities and connect to server-based middleware applications.

To enchance JavaFX application with dependancy injection and to add Client application features, we decided to use Spring Boot framework. Spring provides libraries specific to a client web applications, for example, to make a REST request to our Service application on cloud.

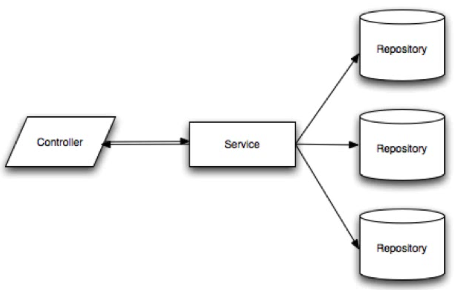
Maven is used here too to build a project structure and manage necessary dependencies.

### Service Side Architecture



N-Tier applications:

* Presentation Layer interacts with users
* Business Logic provides variouse services to accommodate presentation layer (feed with data, processing user queries and return results).
* Data Layer accessing database derictly, based on requests from Services in Business Logic and returns a valid data or exceptions if data is not available.



* Controllers – presentation layer
* Services – business logic
* Repositories – data layer

**Key annotations and definitions in Spring**

Dependency Injection – also called Wiring, helps in gluing java classes together and at the same time keeping those classes as independent as possible to increase reusability and ease of testing.

Spring Beans – Java Object that is instantiated, assembled, and otherwise managed by a Spring IoC container. To become a Spring Bean, java class has to be either defined in XML file (in Spring versions prior to Spring Boot), marked with one of the following steriotype annotation

* @Component
* @Controller
* @Service
* @Repository

or defined in configuration class (marked with @Configuration annotation) as @Bean method.

@Autowired – annotation that performs dependancy injections of Spring Beans.

Autowired annotation can be used on properties, setter methods or class constructors with parameters to instantiate an object of a certain type. By default, Spring autowire objects by its type.

@Qualifier – in java, it is possible to have many objects of a same type. To use a specific object out of this group, autowiring can be done by name of a bean in combination with @Qualifier(“beanName”) annotation.

To give bean a name, you have to add *name = “beanName”* to a steriotype annotation mentioned above, for example:

@Component(*name = “beanName”*)

public class MyClass {

// class body

}

and autowiring will look like this:

@Qualifier(“beanName”)

@Autowired MyClass myClass;

**Controllers –** provide an URI end points to a Client applications to request data. Controllers “presenting” data to a client based on requested URI.

To make a Rest Controller in Spring Boot application and make it respond to requests, variouse Spring annotations are used:

@RestController – A new Spring 4 annotation, which marks the class as a controller where every method returns a domain object instead of a view like html or jsp page.

A domain object is a resource represerntation class (POJO / Model) providing object fields, constructors, and accessors for the data like id, name, message, etc.

@RequestMapping – annotation that links a URIs to a class or method inside the controller.

It is possible to mark entire class to have a specific URI and methods to extend from this URI to form a pattern to a resources.

For example, we can mark a class with @RequestMapping(“/chatroom/”) and a method with @RequestMapping (“/public”). The full URI to a public chat room would look like this: *…/chatroom/public*

Alternative to @RequestMapping annotation, you can use annotations like:

@GetMapping – is itself annotated with @RequestMapping(method = “GET”)

@PostMapping – @RequestMapping(method = “POST”)

to narrow down HTTP methods that end-point can process (GET, POST, DELETE, PUT, etc)

@ResponseBody – annotation that process a Model object to a JSON or XML format, hence by the name of annotation, this json/xml object than is placed into a body of a request and dispatched to a Client as a part of a response.

By default, @ResponseBody is set to convert given Model to a JSON, but can be altered to XML via specifying MediaType in RequestMapping like this @RequestMapping(produces = MediaType.APPLICATION\_XML\_VALUE).

Spring’s *MappingJackson2HttpMessageConverter* class using Jackson 2 library for marshaling Models.

@RequestBody – is similar annotation to a @ResponseBody, but works other way around. @RequestBody and a Model for it is specified in parameter of method. Request body is passed through an *HttpMessageConverter* to resolve the method argument depending on the content type of the request (json/xml) and instantiate a Model that can be used in business logic of an application.

**Services –** indicates business logic.

@Service annotation makes the class autowireable. Now Spring treat Service class as Spring Bean and can inject it as a dependency into our controllers using @Autowired annotation.

In our Server, both *UserServices* (links to Sql Repository) and *MessageServices* (links MongoDB Repository) contain the methods to call operations on databases.

**Repositories** – interfaces for quering databases

In our project, we are using two types of repository interfaces from Spring framework:

* JpaRepository
* MongoRepository

To make a use out of them, we have to define our own interfaces that extends from two above:

* MongoDAO
* SqlDAO

This allows us to use implemented methods from Spring framework to query database, as well as defining our custom methods to query data by a particular parameter. There is no need for a programmer to write MySql or MongoDB statements, although you can define a custom methods with custom statement implementation for more complex quering.

**JPA – Java Persistence API**

It is a Java specification for accessing, persisting, and managing data between Java objects / classes and a relational databases like MySql. It can be performed in XML or Java Annotations.

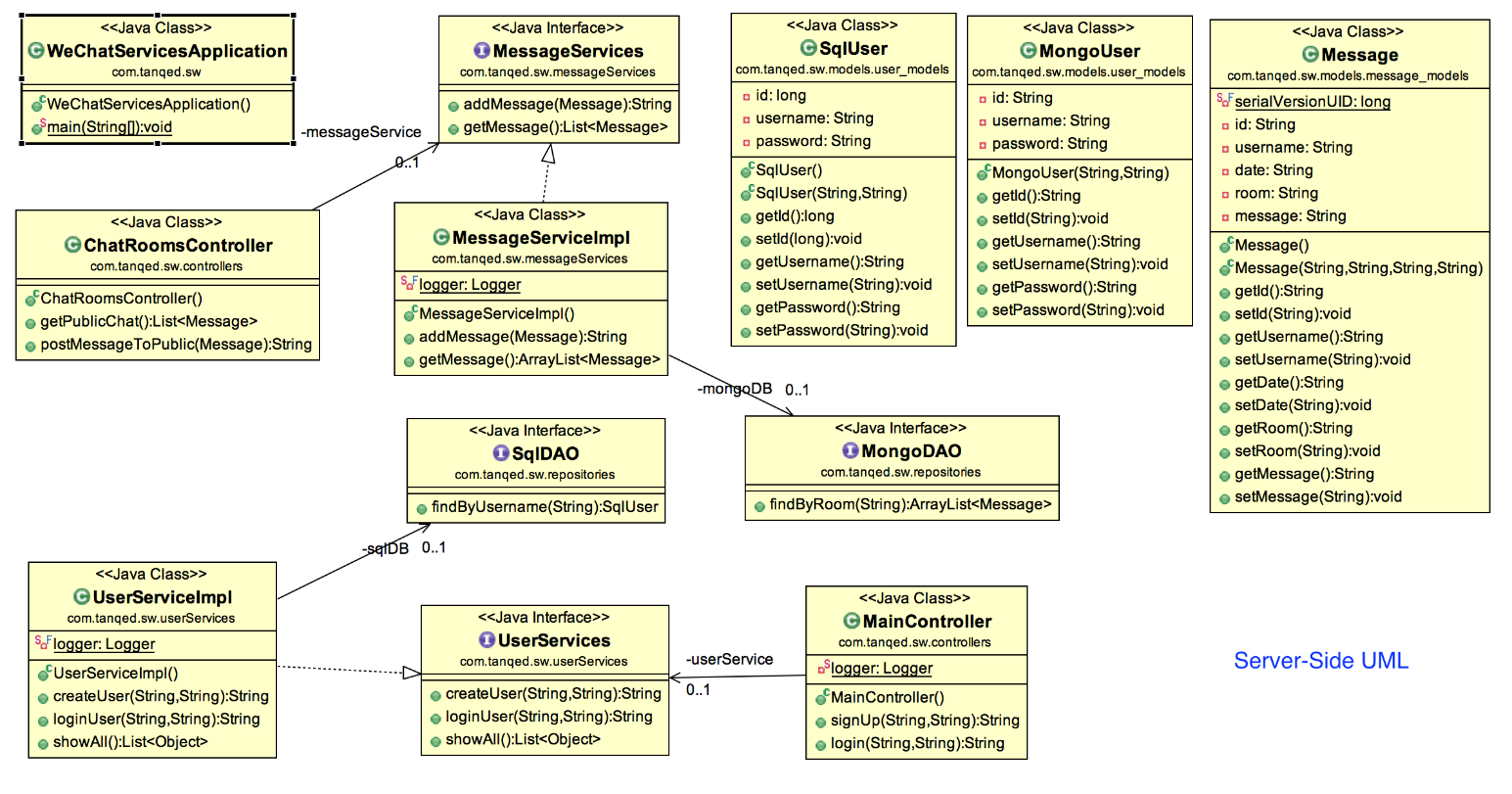
JPA Annotations for Models:

* @Entity - declares an object as an entity that the database should be aware of.
* @Table - describes more specific details about the entity e.g. table name.
* @Id – Identifier attribute for a Primary Key.
* @GeneratedValue – Used in conjunction with @Id to generate a unique value for a field.
* @Column – describes more specific details about the column e.g. column name.

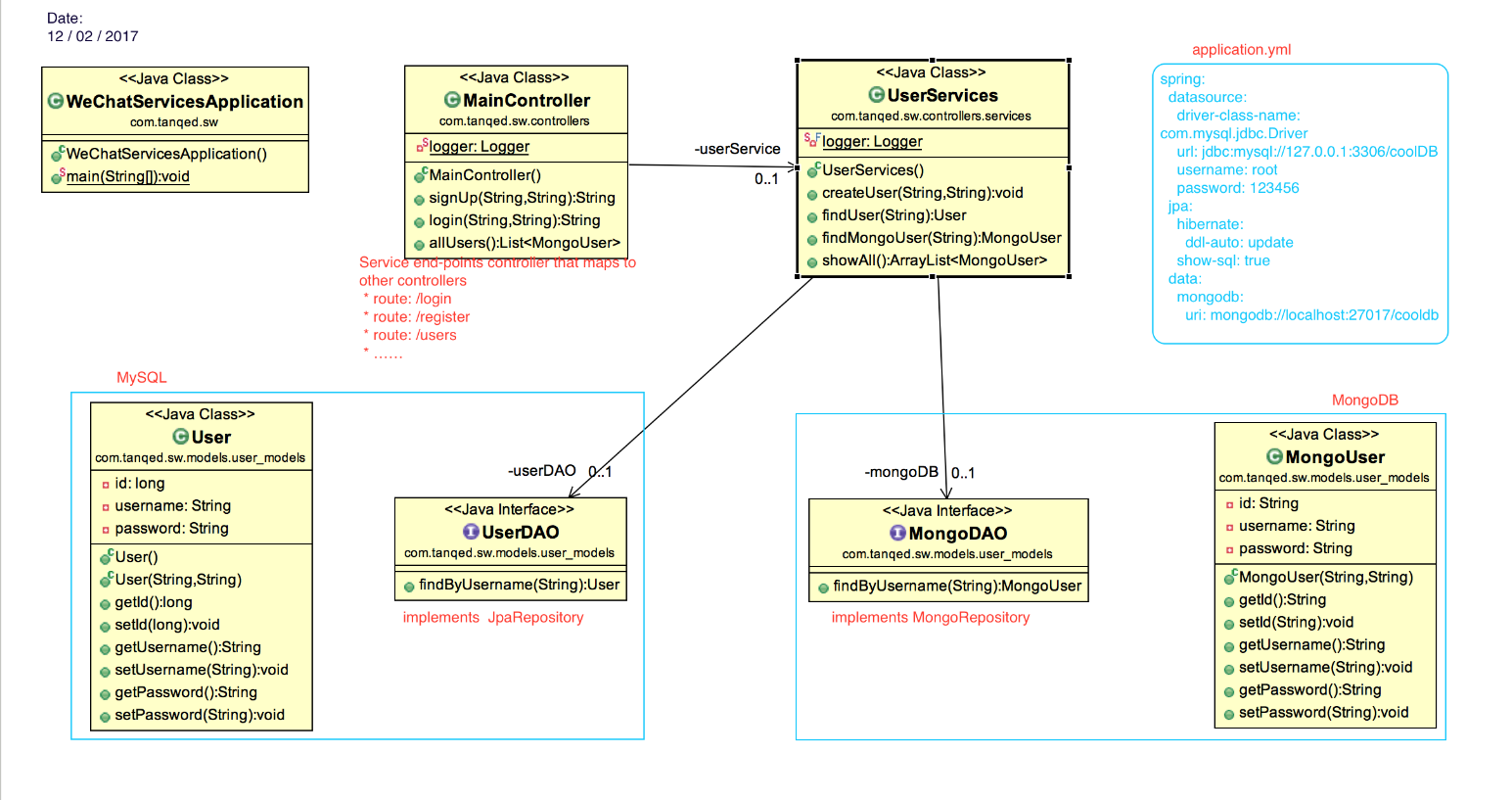
**NoSql databases**

For MongoDB, there is no need in JPA. Only annotation used is an @Id that define a property of a Model to be an id in Mongo database.

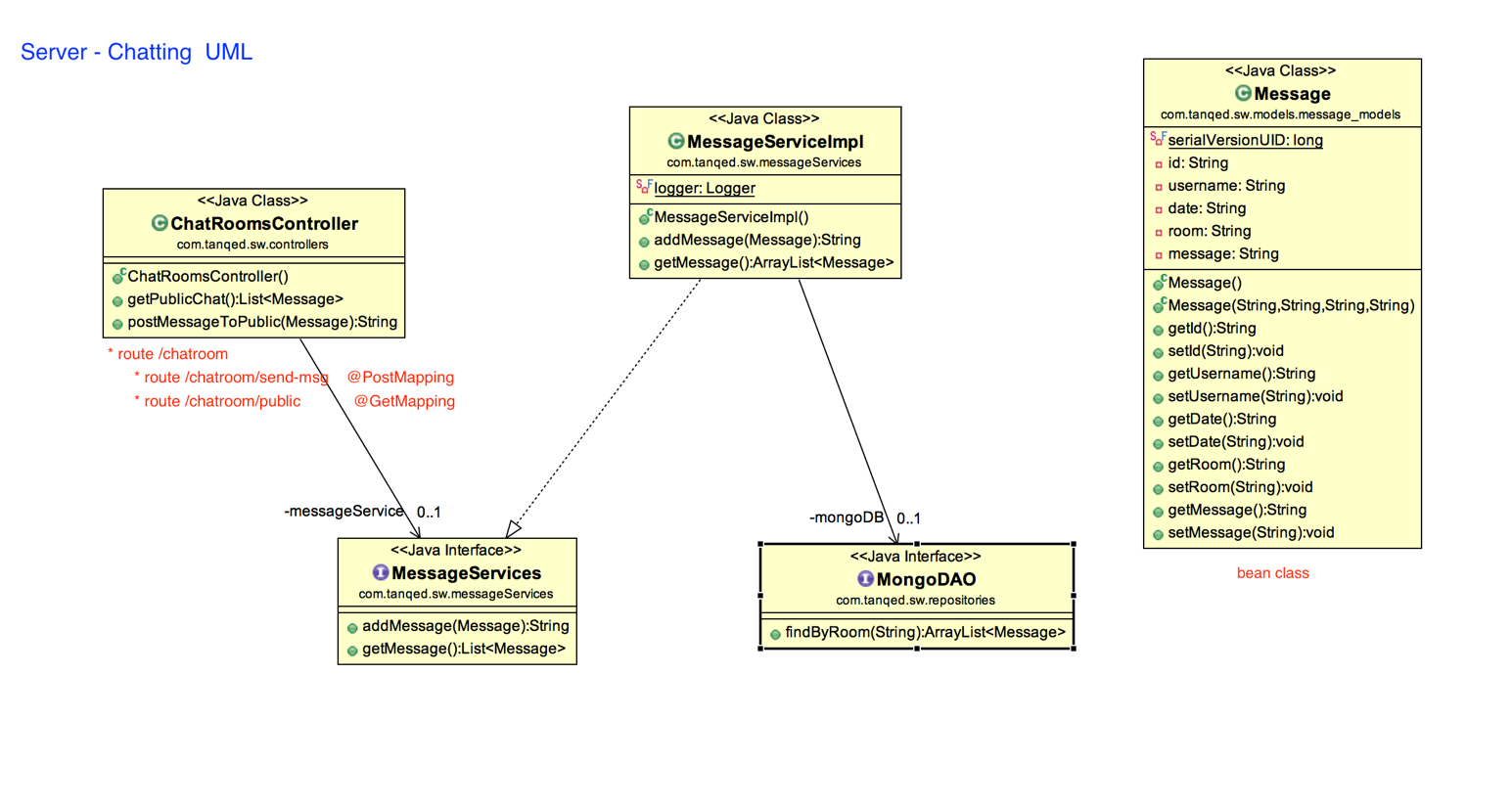
**Full Service Architecture UML Diagram**



**Log-in & register UML diagram**



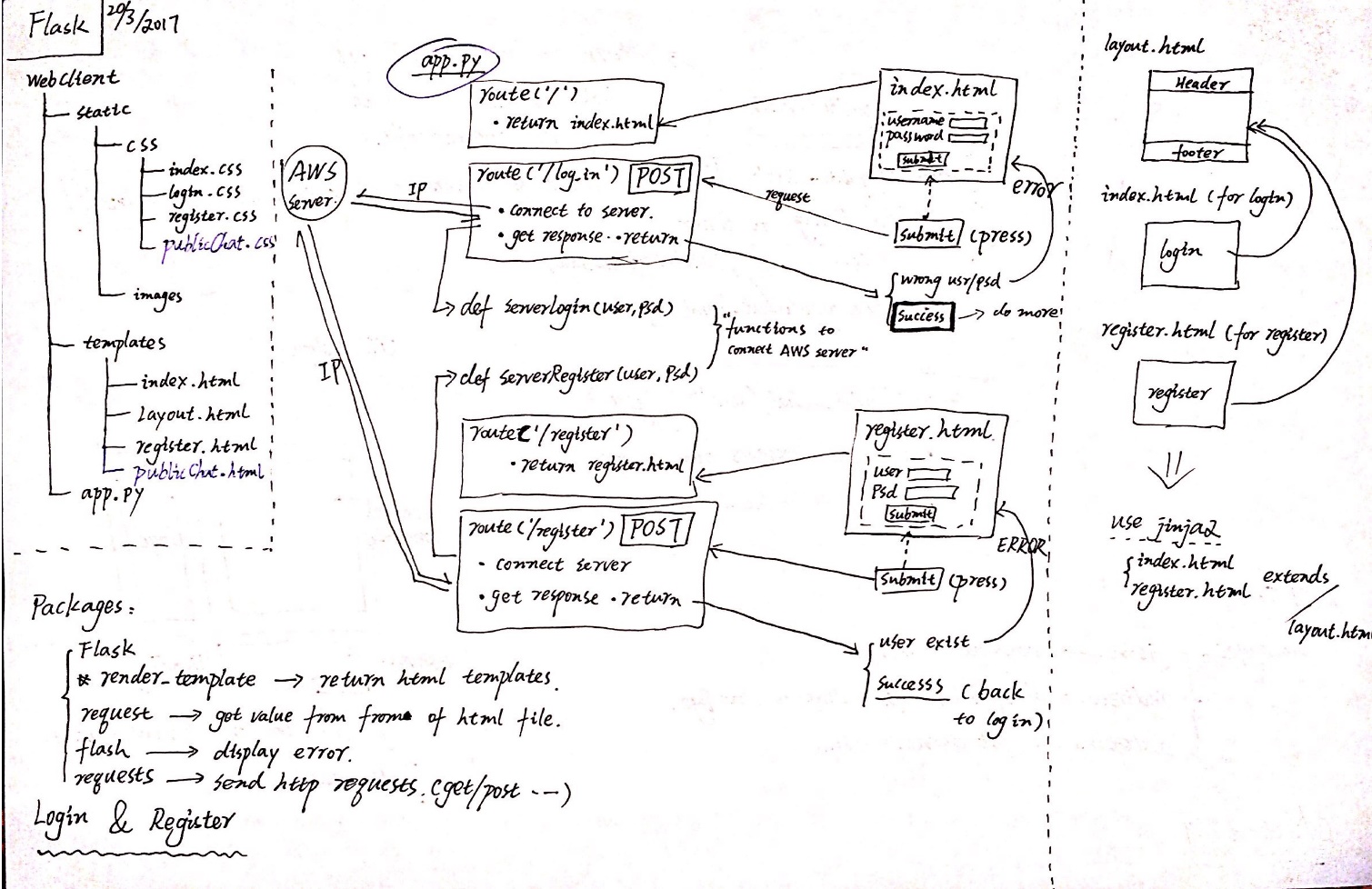
**Chatting UML diagram**



### 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

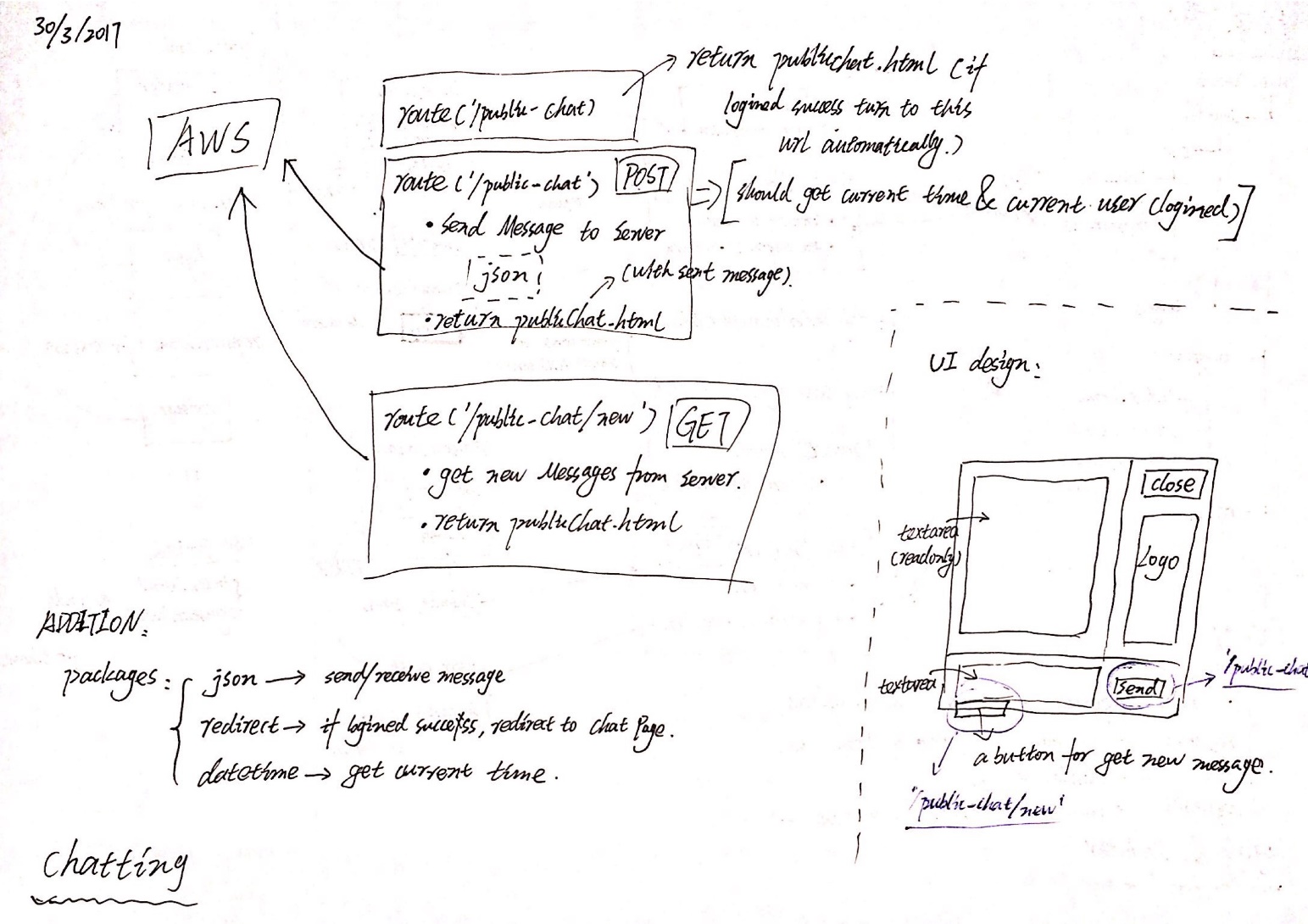
Handle different HTTP methods. For example, using *requests.get* method to send username/password to the server for log-in, and *requests.post* method for register communications.

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

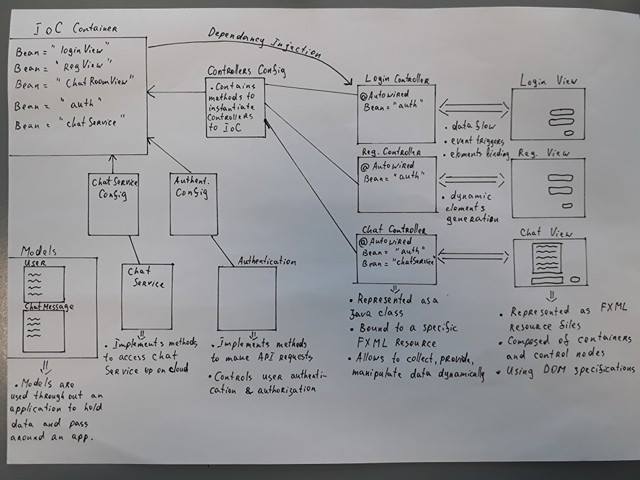
The datetime module supplies classes for manipulating dates and times in both simple and complex ways. Here it is used to get current time when you send a message, then, add into message (json) objects.

**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*



**A bit on IoC Container and Config files**

IoC stands for Inversion of Control. Container is at the core of the Spring Framework.

The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. Container uses DI (dependancy injection) to manage the components that make up an application.

The container gets its instructions on what object to instantiate, configure, and assemble by reading Java configuration files provided with an app.

In JavaFX application, we have 3 configuration classes with instructions for the container:

* ConfigurationControllers – loading FXML resources into application and registering its controllers as Spring beans to perform Spring DI. It also contains a inner class called View to wrap up controller and return an actuall view to a user.

Note: Spring cannot inject objects into FX Controllers if IoC doesn’t know about them. JavaFX API instantiating FX Controllers during a resource loading.

* AuthenticationConfigurations – set up authenticator and rest template for it
* ChatRoomsConfig – set up chatrooms and rest template for them

**FXML resources and Controllers**

JavaFX gives opportunity to separate code (controller) and visualization. Visualization is stored in XML format in file with \*.fxml extension.

To bind controller to a particular view, pay attention to an attribute of root node:

<AnchorPane id="RootNode"fx:controller="com.tanqed.sw.controllers.chatrooms.PublicChatRoomController">

To bind nodes to a properties in controller, an attribute fx:id is used.

<TextArea fx:id="messageTextArea">

Note that fx:id must match a name of a property inside controller.

public class PublicChatRoomController{

@FXML TextArea messageTextArea;

}

The @FXML annotation is tagging a nonpublic class property for use by FXML markup.

Similirally, we can use @FXML annotation to mark a methods:

|  |
| --- |
| @FXML |
| public void loginSubmit(ActionEvent event){  // login logic  } |
|  |

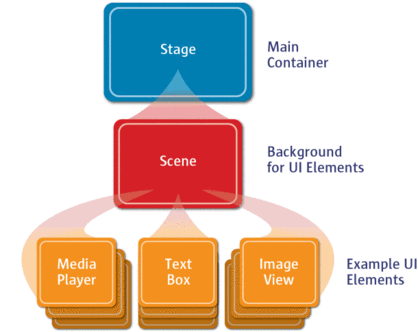
Attaching an event in FXML markup

<Button onAction="#loginSubmit" text="Login">

For a convenient designing of GUI in JavaFX application, use [Scene Builder](http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html) tool and [Netbeans IDE](https://netbeans.org/downloads/).

**Navigation between views**

The following graph illustrates the structure of every JavaFX application:



The Stage is a main container which is usually a Window with a border and the typical minimize, maxeimize and close buttons. Stage is received as a parameter to a start(….) method. Inside the stage you add a Scene, which can be switched out by another Scene. The Scene is composed of actual JavaFX nodes like AnchorPane, TextField, Button, etc.

The mechanism for navigation we used is switching scenes on Stage when certain events occur, for example user has logged in and is navigated to a chatroom.

To gain access to a Stage within FX Controllers, we added a public static reference to it in Application class, where start(…) is overriden, and setting it to a Stage given as a parameter of start(…).

public class Application extends AbstractJavaFxApplicationSupport {

public static Stage stage;

|  |
| --- |
| @Override |
| public void start(Stage stage) throws Exception { |
| this.stage = stage;  } |

}

Now we are able to get access to a Stage from any other class and set a new Scene like this:

Application.stage.setScene(new Scene(....));

One would notice that Stage property is exposed to a change by any other class and is not encapsulated, but for now this is the only way we found to get to the Stage without receiving NullPointerException.

What exactly do we place for the parameter of a Scene constructor?

Constructor is expecting a Parent Node of the FXML resource. That is where we use our Spring Bean defined in config file and getting its Parent.

We can choose what View we want by autowiring the bean by its name.

|  |
| --- |
| @Qualifier("chatRoomView") |
| @Autowired |
| private ConfigurationControllers.View chatView; |

Application.stage.setScene(new Scene(chatView.getParentView()));

We can than use this technique in FXML event method to change visualization on Stage.

**Requests to Cloud Service**

To make a REST Request to web Service, Spring Boot provide a RestTemplate class with implemented HTTP methods and a RestTemplateBuilder to return a simple RestTemplate for use.

All what’s left is to inject template into a the class we want to use it with.

|  |  |
| --- | --- |
| @Configuration | |
| public class AuthenticationConfigurations {  @Bean  RestTemplate restTemplate(RestTemplateBuilder builder) {  return builder.build();  }  @Bean(name = "authent")  Authenticator authenticationControl(RestTemplate restTemplate){  Authenticator authenticationControl = new Authenticator();  authenticationControl.setRestTemplate(restTemplate); // method from Authenticator class  return authenticationControl;  }  } | |
|  | |
| @Component |
|  |  |

public class Authenticator {

private RestTemplate restTemplate;

public void setRestTemplate(RestTemplate restTemplate) {

this.restTemplate = restTemplate;

}

}

RestTemplate provide methods like getForObject(…) or postForObject(…) that excepts necessary parameters to make a request and returns a result. Result type to return can be specified in one of the parameters to a method used.

# 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.

#### *Implementing asynchornious thread in JavaFX*

Not too long ago we figured out that implementing Runnable Tasks from JavaFX thread is

complicated task than we though. It involves a special Tasks from JavaFX concurrency library.

Unfortinutly we didn’t get enough time to look at it and implement it in JavaFX Client.

Material to study [JavaFX Multi Threading](http://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm).

# 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.

#### *On navigating in JavaFX Client*

Whenever we try to navigate to another page, exception of unknown kind appears.

From exception, we figured out it has something to do with a lost stylesheet.

The problem does not produce any functional bugs. No solution yet implemented.

# 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.
* We believe there is a better way to manage navigation in JavaFX Client, but not very different from what we have already.
* Adding stylesheets to JavaFX Client. Has additional learning involved in FX syntaxis for stylesheet.

# Conclusions

During this project development, we were able to find a use of our knowledge obtained from previous years modules, this semester modules, as well as learning new technologies not covered on a course.

From our experience, we’ve concluded that it is worth spending more time in researching Spring framework capabilities, as it can provide much more fundament for an application development.

Spring framework became our favorite Java Framework to develop with and we would definatly consider to develop other projects using it.

**Learning outcomes**

* Distributed Systems – where more than one device can communicate with each other on web using one meeting point, i.e., Web Service.
* Spring Framework – a powerful Java framework that allows to write web Services and help manage development of Client applications like JavaFX applications.
* Object Oriented Design Patterns – to design a robust, reusable, scalable, adaptable to a change applications.
* Development tools – Postman, Lombok, Robomongo all helped in development of this project
* Amazon Web Services – for hosting our Spring Boot Service appliation up on cloud. Requires some firewall configurations to open ports to TCP inbound rules.
* RESTful Services – using a stateless Service for processing HTTP communications across the web.
* We have improved our ability to collaborate in group project, solving complex problems together, while also assigning smaller tasks to each individual.

Thank You for attention,

Tangqi & Eduards.