MongoDB + WebSocket[SockJS] 접목

소프트웨어공학과 / 201332001 강인성 / hogu9401@gmail.com

0. WebSocket 배경 지식

A. WebSocket 정의와 특성

WebSocket은 컴퓨터 네트워크 전용 통신 규약 중 하나인데 대부분 도메인 앞에 ws:// 를 붙어서 이용을 하기 마련이다. WebSocket는 W3C, IETF 단체에서(이는 인터넷 서비스와 네트워크 간의 규정을 정하는 기관. 데이터통신 시간에 데이터와 서로 통신을 하기 위한 규율을 정하는 단체들이 존재하듯이 서버를 통해 받는 인터넷 서비스와 네트워크 사이에 규율을 정하는 단체로 생각하면 된다.) Web Socket은 Web Server와 Web Browser 간의 통신을 위한 규정을 정의한 쌍방향 통신(Duplex Telecommunication)용 기술 규약으로 볼 수 있다. API는 W3C가 책정, WebSocket 프로토콜은 IETF가 책정하였다.

더욱 쉽게 생각을 해 본다면 과거에 어렸을 때 야후 꾸러기나 주니어 네이버에서 간단하게 즐겼던 플래시 게임 등의 기술을 이용 한 사례와 매년 마다 국세청에서 연말 정산을 뗄 때 이용하는 ActiveX를 매일 설치 하면서 이 프로그램을 진행하는 경우에 플러그 인 형태로 브라우저에 직접 설치를 하여 이 때문에 웹 페이지의 비중량이 생각보다 무거워지게 되어 단점을 보였지만, 이러한 응용프로그램을 거치지 않고 순수 웹 환경에서 실시간으로 양방향 통신을 위한 스펙을 고려를 해야 할 때로 다가왔을 때 이를 방지하는 프로토콜이 바로 WebSocket이 되시겠다. 그래서 WebSocket을 이용이 필요로 한 프로젝트의 조건에 대해서 설명을 한다면 아래와 같다.

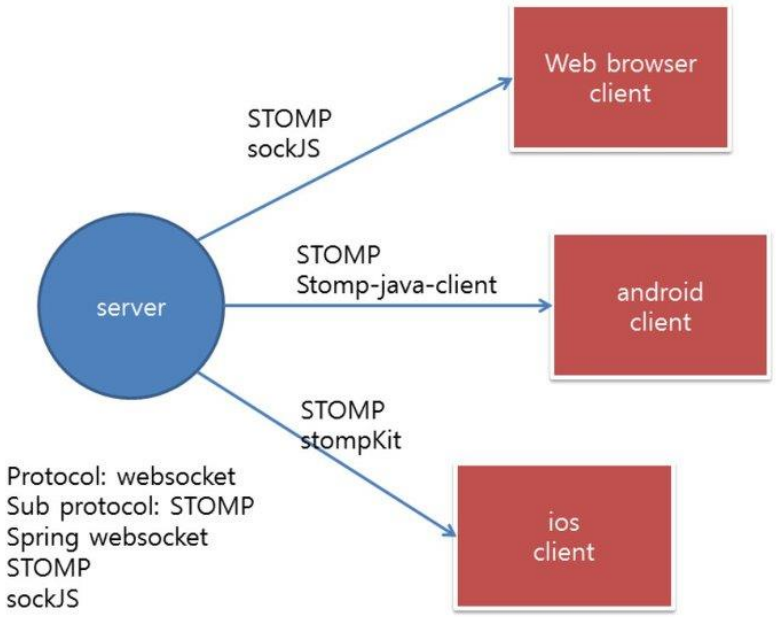
- 실시간 양방향 데이터 통신이 필요한 경우. 이는 이메일이나 쪽지, 메신저 등이 될 수 있는데 데이터의 양이 적은 경우에 통신을 할 때 필살기로 내세울 수도 있다.
- 많은 수의 동시 접속자를 수용해야 하는 경우. 예를 들어 유튜브 실시간 방송을 통해서 많은 사람들이 동영상을 보면서 후원을 하거나 댓글을 남기는 경우에 유튜버가 시청자가 올린 댓글들과 후원 확인을 손쉽게 확인하면서 유튜브 스트리밍에 지장이 없도록 하기 위하여 WebSocket을 이용을 하지만 아직까지 유튜브에서 동시 접속자 수용 면에서는 유튜버와의 속도를 맞추는 면에서는 무리수가 나오기는 한다.
- 브라우저에서 TCP 기반의 통신으로 확장해야 하는 경우. 우리가 여태껏 데이터들을 서버를 통해 받아왔지만 또 다른 통신(이를 Handler를 통해 작성한 Controller 클래스)을 확장을 해서 실시간으로 받아오면서 속도를 향상을 해야 할 경우에 필요로 하다.
- 개발자에게 사용하기 쉬운 API가 필요한 경우.
- 클라우드 환경이나 웹을 넘어서 SOA(Service Oriented Architecture)로 확장을 하게 된다면 WebSocket는 빠질 수가 없게 된다.

B. SockJS 소개

우리가 Spring에서 WebSocket을 이용하기 위해서는 2가지 라이브러리를 선택해서 이용을 할 수 있는데 하나는 Socket.io, 또 하나는 SockJS 2가지가 존재를 한다. 둘 다 JavaScript 언어를 기반으로 한 양방향 실시간 통신 프로토콜에 대해서 적용을 할 수 있도록 도와주는데 둘의 차이점은 SocketIO는 오로지 JavaScript 문장만을 활용해서 구성을 하고, SockJS는 JQuery 라이브러리를 기반으로도 제공을 하여 일반 JavaScript 문장 2가지를 이용해서 활용을 하는 사례를 볼 수 있다. 그렇지만 Angular, ReactJS, VueJS 등 JavaScript 라이브러리로 클라이언트 UI를 구현을 하는 경우에는 SocketIO를 대체적으로 쓰고 있지만, 다행이 SockJS에 맞춘 라이브러리도 제공은 하고 있는 추세이다. 이번 시간에는 SockJS에 대해서 간단한 개념들을 이해를 하면서 어떤 원리로 구성이 되어 있는지에 대해 공부를 해 보도록 하겠다.

SockJS 라이브러리는 SocketIO와 달리 STOMP 규격 프로토콜을 이용해서 접근을 할 수 있는 방법을 제공하여 기존 http 서버에서 못 하는 역할을 추가적으로 접목을 시켜 데이터를 저장을 할 수 있는 기능까지 제공을 하면서 또한 SocketIO를 이용하는 점에서는 브라우저 별 WebSocket 제공을 하는 관점에서는 어렵사리 제공을 하기 때문에 이에 맞춘 문제를 고려를 해야 하지만 SockJS는 이를 최소화를 하여 제공을 하고 있기 때문에 오히려 브라우저 문제를 해결하는 면에서는 이를 이용을 하는 것을 권장을 하고 있다.

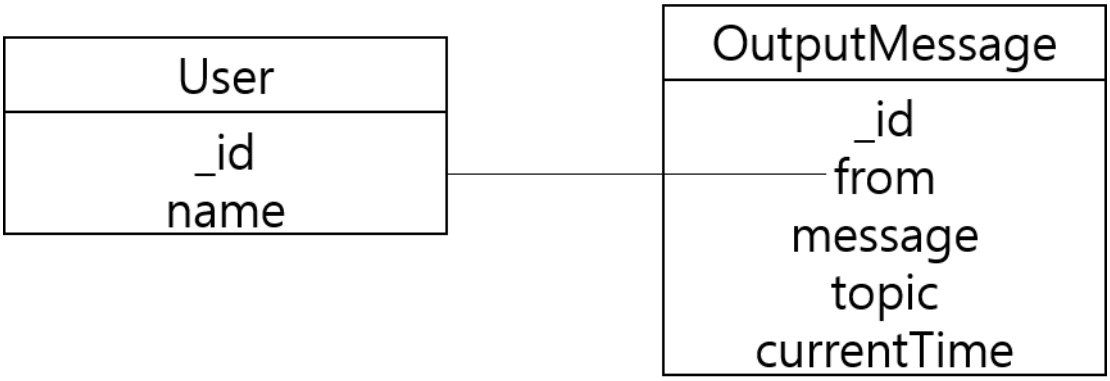
SockJS에서는 STOMP 규격 프로토콜을 이용해서 정리를 하고 Spring에서는 이 프로토콜의 Message URL를 통해 데이터를 전송하거나 정리를 하는 것을 중점으로 두고 있다고 보면 된다. 일단 웹 브라우저 버전이 어떻든 간에 SockJS 라이브러리를 이용한 JavaScript 문서만 있다면 이를 알아서 커버해서 적용을 시키고 또한 SockJS WebSocket과 STOMP 규격 프로토콜을 서로 격리 시켜서 이용을 하는 방법도 적용하는 방법도 있다.



[그림] SockJS WebSocket과 STOMP 프로토콜을 이용한 각 클라이언트에 접목을 하는 방법

1. MongoDB 관련 데이터베이스 생성

이전에 MongoConfig.java는 그대로 적용을 하면 된다. 그리고 본인은 아래와 같은 데이터베이스를 새로 생성하여 적용을 하는데 사용자에 대해서는 Security에 대해서 접목을 하게 된다면 설명이 길어 지기 때문에 일단은 WebSocket에서 사용자들과 실시간 전체 채팅을 하는 장면을 통해서 접목을 하는 것을 위주로 설명을 하겠다.



[그림] messenger_example 데이터베이스 구성. 간단하게 User는 이름만 쓰고 OutputMessage를 이용해 사용자가 작성한 문장들에 대해 실시간으로 보내는 것으로 구상을 하였다.

A. Domain 클래스 생성

```
package net.kang.domain;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import lombok.Data;

@Data
@Document(collection="user")
public class User {
	@Id
	String id;
	String name;
}
```

[mongoDB_JPA_Start03 > src > main > java > net > kang > domain > User.java]

```
package net.kang.domain;

import java.util.Date;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.format.annotation.DateTimeFormat.ISO;
```

```
import lombok.Data;

@Data
@Document(collection="outputmessage")
public class OutputMessage {
	@Id
	String id;
	@DBRef(db="messenger_example", lazy=false)
	User from;
	String message;
	String topic;
	@DateTimeFormat(iso = ISO.DATE_TIME)
	Date currentTime;
}
```

[mongoDB_JPA_Start03 > src > main > java > net > kang > domain > OutputMessage.java]

참고로 MongoDB에서 collection에서 주목을 해야 하는 것은 테이블 이름에서 대/소문자를 제대로 구분해야 한다. MySQL에서는 대/소문자에 대해서는 구분을 안 하였지만, MongoDB에서는 특별한 설정을 해 두지 않는 경우에 문제가 발생할 수 있기 때문에 이를 주의해야 한다. 또한 MongoDB에서 저장되는 시간은 Java Date 타입과 다르기 때문에 @DateTimeFormat 어노테이션을 이용해서 이를 극복하는 방법 이외에는 없다.

B. Repository 인터페이스 생성

```
package net.kang.repository;

import java.util.Optional;

import org.springframework.data.mongodb.repository.MongoRepository;

import net.kang.domain.User;

public interface UserRepository extends MongoRepository<User, String>{
	Optional<User> findByName(String name);
}
```

[mongoDB_JPA_Start03 > src > main > java > net > kang > repository > UserRepository.java]

참고로 본인은 MongoDB에서 User의 이름에 대해서는 유일함을 보장하는 Index를 따로 만들고 작업을 하였다. 이를 진행하지 않았다면 방금 앞에 User.Java에 @CompoundIndexes 어노테이션을 이용해서 적용을 시키길 바란다.

```
package net.kang.repository;

import org.springframework.data.mongodb.repository.MongoRepository;

import net.kang.domain.OutputMessage;

public interface OutputMessageRepository extends MongoRepository<OutputMessage, String>{
	OutputMessage findTopByOrderByCurrentTimeDesc();
}
```

[mongoDB_JPA_Start03 > src > main > java > net > kang > repository > OutputMessageRepository.java]

새로 쓴 함수는 사용자가 글을 추가하고 난 이후에 최신 글을 불러올 때 쓰는 문장을 Query Creation으로 작성하였다.

C. Service 클래스 생성

```
package net.kang.service;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import net.kang.domain.User;
import net.kang.repository.UserRepository;

@Service
public class UserService {
	@Autowired UserRepository userRepository;
	public User findByUser(String name) {
		Optional<User> user=userRepository.findByName(name);
		User realUser=user.orElse(new User());
		return realUser;
	}
}
```

[mongoDB_JPA_Start03 > src > main > java > net > kang > service > UserService.java]

본인은 User에 대해서 Security를 적용하지 않고 넘어갔지만, 실제로 Security를 이용하는 경우에 로그인, 회원 등록 등의 기능을 작성할 때에는 여기에 작성을 해서 실제로 로그인 서비스를 접목할 때 이를 참고하도록 하는 방면이 좋다.

```
package net.kang.service;

import java.util.Date;
import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import net.kang.domain.OutputMessage;
import net.kang.domain.User;
import net.kang.model.Message;
import net.kang.repository.OutputMessageRepository;
import net.kang.repository.UserRepository;
@Service
public class OutputMessageService {
	@Autowired UserRepository userRepository;
	@Autowired OutputMessageRepository outputMessageRepository;
	public List<OutputMessage> findAll(){
		return outputMessageRepository.findAll();
	}
	public boolean insert(Message message, String topic) {
		OutputMessage newMessage=new OutputMessage();
		Optional<User> user=userRepository.findByName(message.getFrom());
		User realUser=user.orElse(new User());
		if(user.equals(new User())) return false;
		newMessage.setFrom(realUser);
		newMessage.setCurrentTime(new Date());
		newMessage.setTopic(topic);
		newMessage.setMessage(message.getText());
```

```
                outputMessageRepository.insert(newMessage);
                return true;
        }
        public OutputMessage findTopByOrderByCurrentTimeDesc() {
                return outputMessageRepository.findTopByOrderByCurrentTimeDesc();
        }
}
```

[mongoDB_JPA_Start03 > src > main > java > net > kang > service > OutputMessageService.java]

Message 라는 객체를 받아 와서 각 사용자 정보와 메시지 내용들을 받아와서 채팅 글을 새로 추가를 하고, 아래 문장은 글을 추가하고 난 후에 그 사용자가 방금 작성한 글에 대해 가져오도록 하는 문장으로 보면 된다.

2. Spring WebSocket 적용하기

A. WebSocketConfig.java

우리가 WebSocket을 이용하기 위해서는 Configuration 클래스를 이용해서 설정을 해 두면 간편하게 이용을 할 수 있는데 과거에는 XML 파일을 이용해서 Bean으로 설정을 하는 방법이 있었는데 본인은 Java Configuration을 이용한 방법으로 작성을 하였다.

```
package net.kang.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import
org.springframework.web.socket.config.annotation.AbstractWebSocketMessageBrokerConfigurer;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;

@Configuration
@EnableWebSocketMessageBroker // Message를 주고 받는 WebSocket을 이용 가능하도록 어노테이션을 추가
public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer{
    @Override
    public void configureMessageBroker(MessageBrokerRegistry config){
        config.enableSimpleBroker("/topic");
        config.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry){
        registry.addEndpoint("/chat").setAllowedOrigins("*").withSockJS();
    }
}
```

[mongoDB_JPA_Start03 > src > main > java > net > kang > config > WebSocketConfig.java]

여기서 configureMessageBroker 함수는 MessageMapping 어노테이션을 통해 간단한 Message를 보낼 URL에 대한 설정으로 볼 수 있다. 이를 이용을 한 이후에 send 함수나 subscribe 함수를 통해 각 앞 자리에 각각 /app, /topic이란 Prefix를 포함해야 작동을 할 수 있도록 하는 문장으로 이해를 할 수 있고 이 Prefix에 대해서 send 함

수와 subscribe 함수를 통해 데이터를 실시간으로 통신할 수 있도록 해 준다. 그리고 registerStompEndpoints 함수는 send 함수에서 쓰인 URL에서 이를 포함시킨 Endpoint를 통해서 메시지 URL에 포함을 시켜야만 연결을 할 수 있는 API를 설정하는 것으로 이해를 하면 되겠다. 그리고 SockJS를 이용을 할 수 있도록 하기 위해서는 마지막에 withSockJS() 함수를 추가를 해 주는 것도 잊지 말아야 한다.

B. Controller 클래스(Handler 클래스로 칭함.) 생성

```
package net.kang.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.handler.annotation.DestinationVariable;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Controller;

import net.kang.domain.OutputMessage;
import net.kang.model.Message;
import net.kang.service.OutputMessageService;

@Controller
public class ChatController {
	@Autowired OutputMessageService outputMessageService;

	@MessageMapping("/chat/{topic}")
	@SendTo("/topic/messages")
	public OutputMessage send(@DestinationVariable("topic") String topic, Message
message) throws Exception{
		outputMessageService.insert(message, topic);
		return outputMessageService.findTopByOrderByCurrentTimeDesc();
	}

	@MessageMapping("/chat/list")
	@SendTo("/topic/message/refresh")
	public List<OutputMessage> findAll(){
		return outputMessageService.findAll();
	}
}
```

[mongoDB_JPA_Start03 > src > main > java > net > kang > controller > ChatController.java]

이 클래스는 Controller 클래스로 작성을 하였지만 실제로는 STOMP 통신에 대해 핸들링을 할 수 있는 Handler 클래스와 같은 맥락으로 볼 수 있다. 여기서 send 함수를 통해서 /chat/{topic} 의 URL를 요청하게 된다면 이에 맞춰서 채팅 내용을 추가하고 난 뒤에 새로 올린 채팅 내용들을 가져오도록 하는 함수로 이해를 할 수 있다. 여기서 SendTo 어노테이션을 통해 구독(Subscribe)에 대해서 subscribe 함수를 이용해서 설정을 해 둔다면 이를 읽어볼 수 있도록 설정을 하는 어노테이션으로 이해를 할 수 있는데 이에 대해서는 chatting.js에서 설정한 소스 코드를 통해서 읽어 본다면 이해하는데 어려움이 없을 것이다. 각각 쓰인 함수에 대해 설명한다면 send 함수는 새로운 메시지 내용을 보내서 MongoDB에 실제로 저장을 하고 난 후에 맨 마지막에 쓴 글을 반환하는 것과 아래

에는 채팅 방에 새로 입장한 사람들이 새로 고침 버튼을 클릭한다면 현재 채팅 인원들이 작성한 글들에 대해서 모두 불러와서 동기화를 시키는 함수로 볼 수 있다. 참고로 STOMP 프로토콜이 아닌 일반 WebSocket을 이용하는 경우에는 EchoHandler를 새로 만들어서 접목을 시켜야 하는데 이에 대해서는 다음에 언급을 해 보도록 하겠다.

C. chatting.js

```
$(function() {
    'use strict';

    var client;

    var messageIdList=[];

    function showMessage(mesg){
        if(!messageIdList.includes(mesg.id)){
                $('#messages').append('<tr>' +
                            '<td>' + mesg.from.name + '</td>' +
                            '<td>' + mesg.topic + '</td>' +
                            '<td>' + mesg.message + '</td>' +
                            '<td>' + new Date(mesg.currentTime).toLocaleDateString() +
'<br/>' + new Date(mesg.currentTime).toLocaleTimeString() + '</td>' +
                            '</tr>');
                messageIdList.push(mesg.id);
        }
    } // 1

    function setConnected(connected) {
                $("#connect").prop("disabled", connected);
                $("#refresh").prop("disabled", !connected);
                $("#disconnect").prop("disabled", !connected);
                $('#from').prop('disabled', connected);
                $('#text').prop('disabled', !connected);
                if (connected) {
                    $('#conversation').show();
                    $('#text').focus();
                }
                else $("#conversation").hide();
                $("#messages").html("");
    } // 2

    $("form").on('submit', function (e) {
        e.preventDefault();
    }); // 3

    $('#from').on('blur change keyup', function(ev) {
        $('#connect').prop('disabled', $(this).val().length == 0 );
    });

    $('#connect,#disconnect,#text,#refresh').prop('disabled', true); // 4

    $('#connect').click(function() {
                client = Stomp.over(new SockJS("/example03_1/chat"));
                client.connect({}, function (frame) {
                    setConnected(true);
                    client.subscribe('/topic/messages', function (message) {
                        showMessage(JSON.parse(message.body));
```

```
                });
            });
    }); // 5

    $('#disconnect').click(function() {
            if (client != null) {
                client.disconnect();
                setConnected(false);
            }
            messageIdList = [];
            client = null;
    }); // 6

    $('#send').click(function() {
            var topic = $('#topic').val();
            client.send("/app/chat/" + topic, {}, JSON.stringify({
                from: $("#from").val(),
                text: $('#text').val(),
            }));
            $('#text').val("");
    }); // 7

    $('#refresh').click(function() {
            client.send("/app/chat/list", {}, {});
            client.subscribe('/topic/message/refresh', function (message) {
                    var array=JSON.parse(message.body);
                    array.forEach(function(m){
                            showMessage(m);
                    });
            });
            $('#text').val("");
    }); // 8
});
```

[mongoDB_JPA_Start03 > src > main > webapp > js > chatting.js]

chatting.js은 JavaScript과 JQuery 라이브러리를 짬뽕해서 작성을 하였는데 이에 대해서 쉽게 인지를 하기 위해 각 함수 별로 번호를 부여해서 내용을 설명하겠다.

1 : showMessage에서는 MongoDB에 현재 저장된 채팅 글들에 대해서 각각 보여주도록 하되 이미 보여진 내용들에 대해서는 messageIdList 배열을 통해 저장을 하여 이를 방지하도록 추가를 하였다. Spring에서는 JSON으로 작성한 객체들에 대해 반환이 되긴 하지만 String으로 구성된 JSON(예를 들어 { "name" : "사람" } 등)이 반환이 되어 이를 Parsing을 하고 난 뒤에 적용을 해야 한다.

2 : 이는 JQuery 함수로 구성이 되어 있는데 접속을 완료한 이후에 버튼을 사용할 수 있게끔 하기 위한 함수들로 이해를 할 수 있다. 처음에는 이름을 입력하는 경우에만 접속 버튼만 작동을 하게 되어 있는데 접속이 완료 된다면 각각 퇴장, 새로 고침, 전송, 채팅 내용 등을 작동 가능하게 하는 것으로 이해를 하면 된다.

3 : 이는 우리가 Form을 이용해서 글을 작성하는 경우에 새로 고침이 되어 리다이렉트를 하는 것이 관례였지만 이를 방지하고 보내게끔 하는 설정으로 보면 된다. 이는 ReactJS, VueJS, Angular 등의 Singled Page Application을 다룬다면 주구장창 봤을 함수이다.

4 : 초기에는 접속, 퇴장, 새로 고침, 전송 버튼들을 사용하지 못 하게끔 막아 둔 함수이다.

5 : 연결 버튼을 클릭한다면 클릭한 이후에 접속자 정보들이 나오게끔 하는 함수로 보면 된다. 여기서 subscribe는 초기에 새로 고침 버튼을 안 누른 경우에 글을 먼저 올리고 난 뒤에 보도록 설정을 하였다. 이에 대해서 해결을 하기 위해 messageIdList 별로 정렬을 하고 난 뒤에 정리를 해서 보이게끔 하는 방법이 있지만 성능이 안 좋아지기 때문에 우선은 배제하였다.

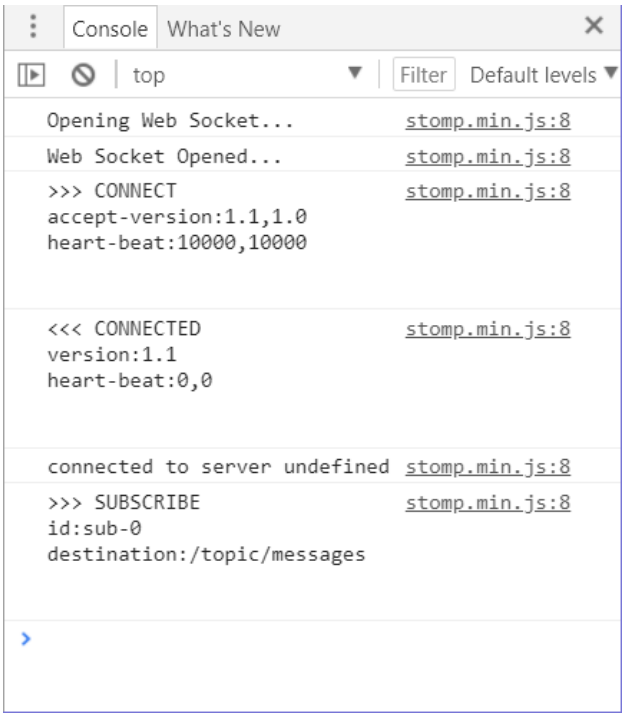
6 : 이는 클라이언트와 WebSocket 간의 연결을 해제하는 함수로 볼 수 있다.

7 : 이는 접속한 사용자 한정으로 채팅 내용들을 보내는 함수로 볼 수 있는데 topic 내용에서 select 태그를 이용해 선택을 하고 글 내용을 입력한다면 보내게 설정을 하였다.

8 : 이는 접속을 아직 못 한 사용자가 채팅 내용들에 대해 읽어볼 수 있도록 하기 위해서 새로 고침 버튼을 이용해 동기화를 할 수 있도록 적용한 함수로 볼 수 있다. 여기서 subscribe 함수는 현재 페이지에서 어느 WebSocket에 접속을 할지에 대해 설정을 하고 난 후에 이에 맞춰서 데이터를 가져오게끔 하기 위한 웹 페이지 내부의 또 다른 Router로 이해를 할 수 있다. 새로 고침 버튼을 클릭하지 않더라도 동기화를 하게끔 하는 작업에 대해서는 어떻게 접목을 시켜야 할지에 대해 구상을 하는 방법에 대해서는 ReactJS를 이용한 채팅 프로그램에서 다시 반영을 해 보이겠다.

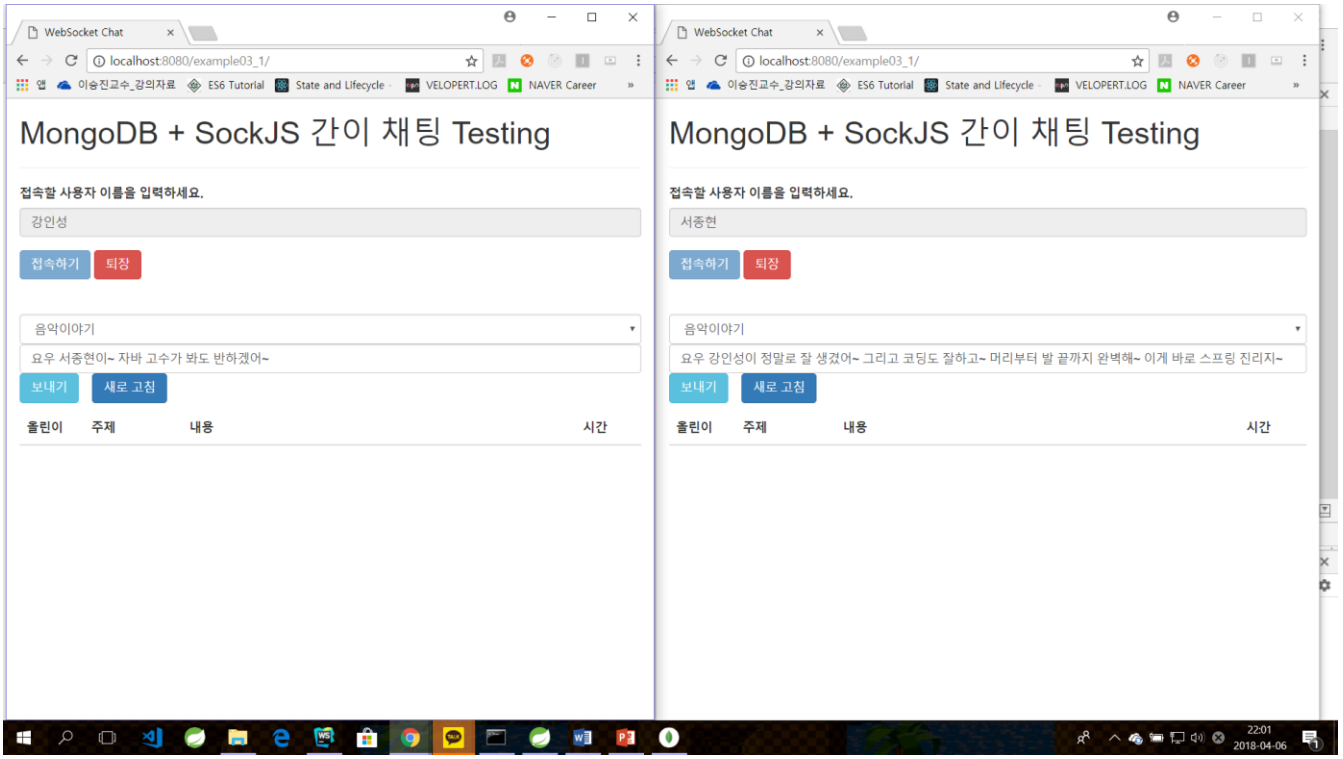
3. 실행 결과와 GitHub 주소 참조

우선적으로 사용자 이름 하나를 입력하고 난 뒤에 실행을 하고 난 후에 F12 키를 누르고 난 후에 콘솔 창에는 아래와 같이 나오게 된다.

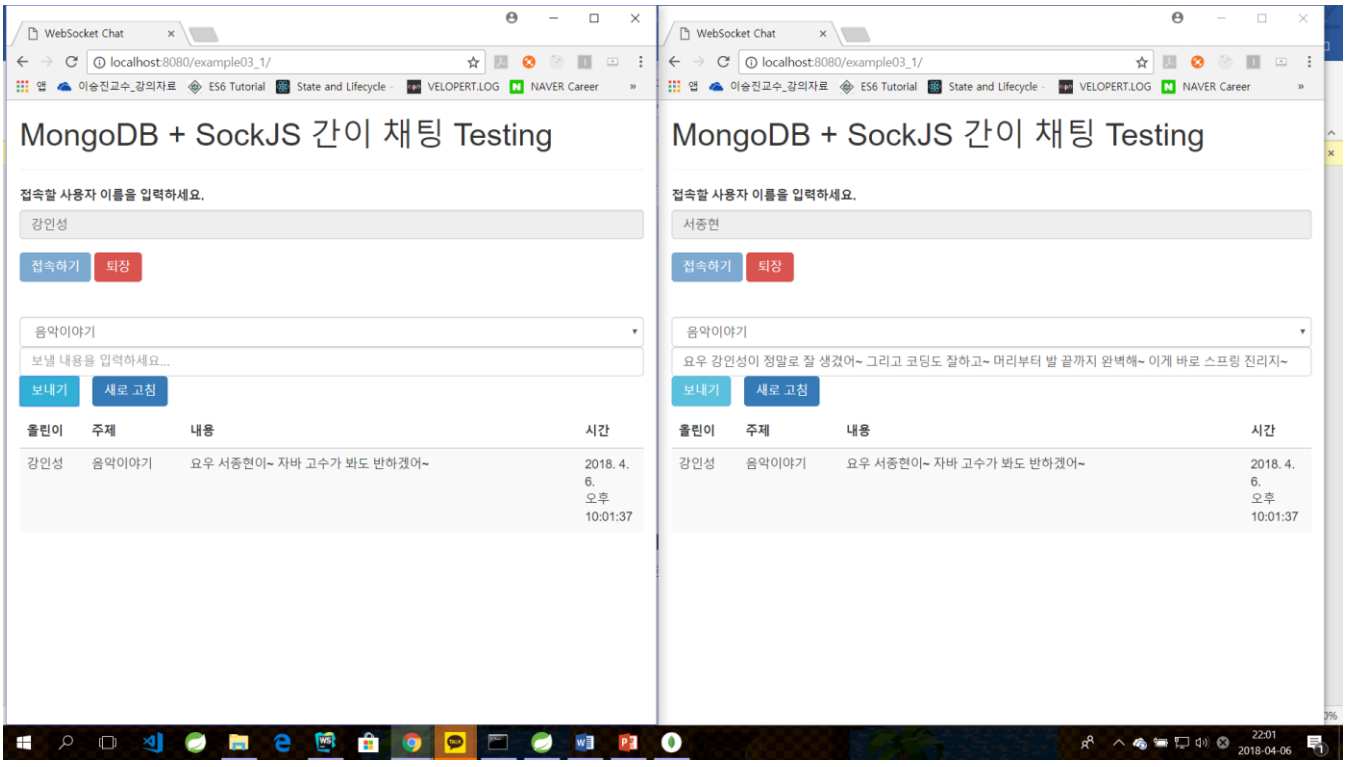


[그림] WebSocket과 연결을 한 결과. 실제로 WebSocket에 대해서 접근을 하는 경우에는 Server를 설정을 하는 점에서 Redis를 이용해 JWT 토큰을 저장을 하는 방안을 이용해서 설정을 하면 좋겠다.

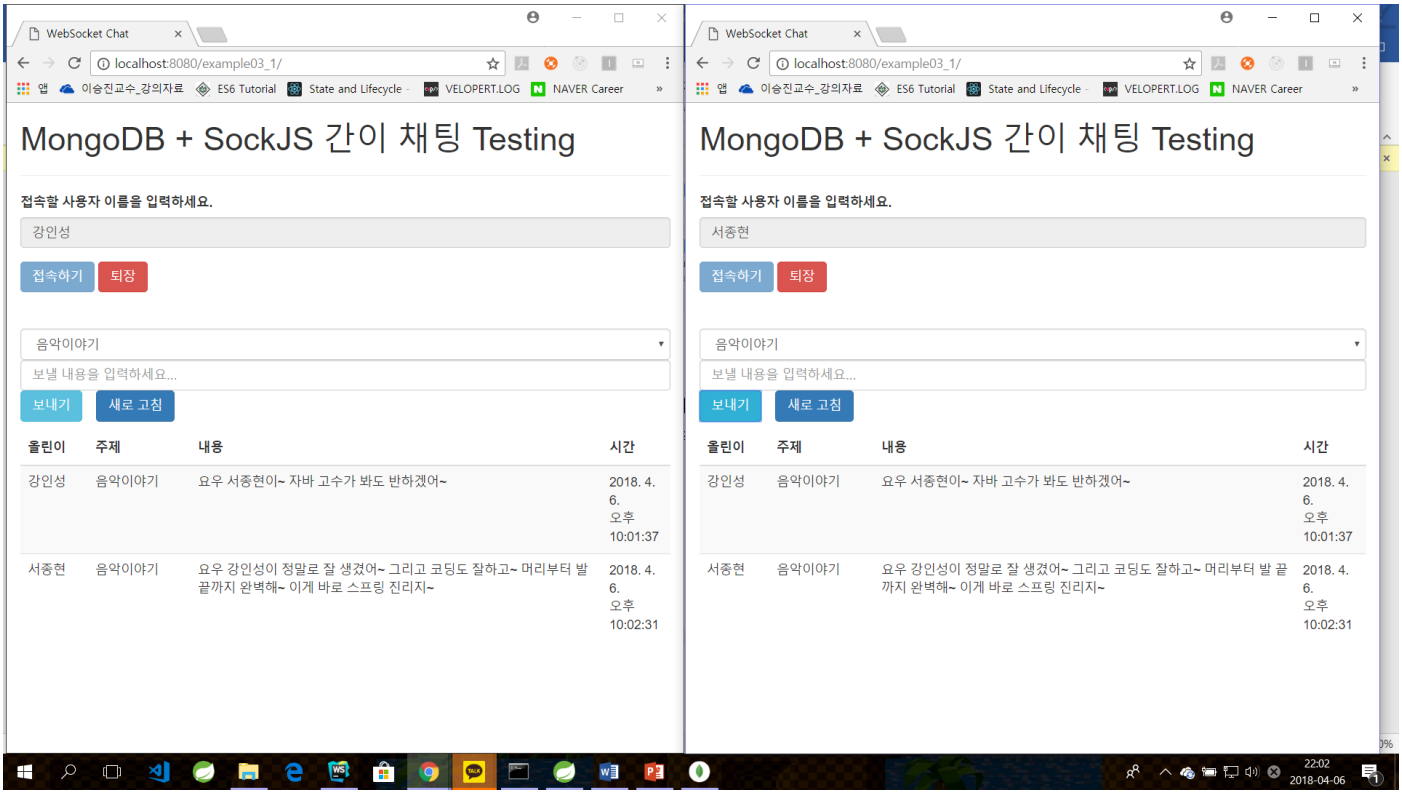
접속을 하고 난 후에 각 개인 별 채팅을 한다면 아래와 같은 결과가 나온다.



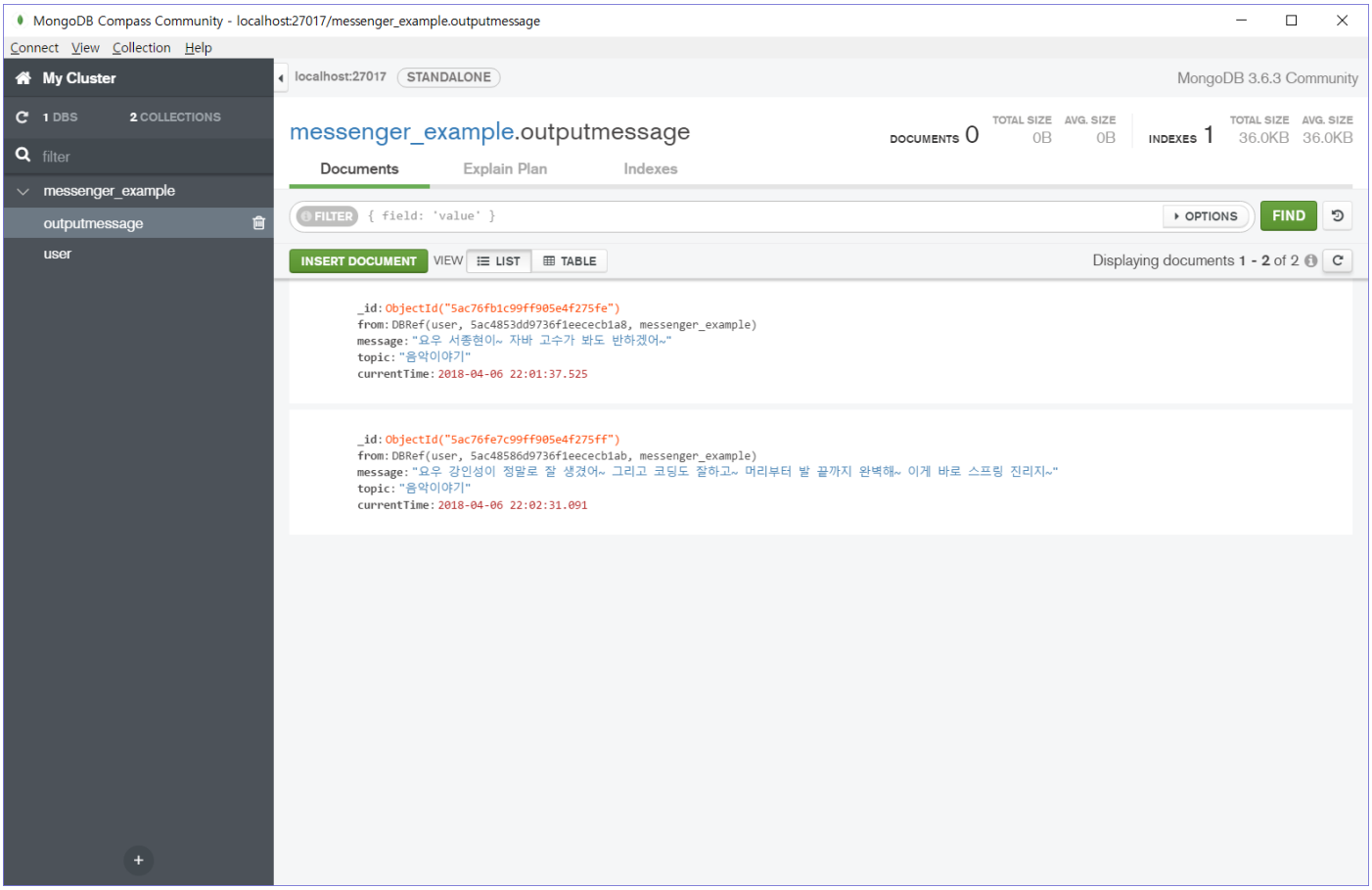
우선은 강인성 이라는 사람이 채팅을 보내면 아래와 같은 결과로 나온다.



그리고 서종현 이라는 사람이 채팅을 보내면 또한 아래와 같은 결과가 나오게 된다.



그리고 이 내용들은 실제로 도청을 하는 느낌이 들지만(... OO ㄱ) MongoDB에서 OutputMessage Collection에서는 위와 같은 글들에 대해서 저장이 되어 있다.



GitHub 주소 참조

https://github.com/tails5555/mongoDB_JPA_Start03

그리고 이 문서는 이 홈페이지에 들어가서 src > doc 파일에 pdf 파일로 추가를 해 뒀으니 소스 코드와 같이 참고하면 좋겠다.

추가로 우리는 SockJS를 이용을 했지만 반대로 SocketIO를 이용을 원하는 사람들은 아래와 같은 사이트를 추가로 기재를 하겠다.

http://d2.naver.com/helloworld/1336 - D2 네이버 SocketIO 개념 설명

https://socket.io/get-started/chat/ - SocketIO Tutorial 페이지

출처

http://asfirstalways.tistory.com/359 - WebSocket 내용에 대해 참고함

http://jusungpark.tistory.com/40 - WebSocket의 기본 개념

https://medium.com/@2xel/websocket-%EA%B8%B0%EB%B3%B8%EA%B0%9C%EB%85%90-%ED%95%99%EC%8A%B5-9de4154f308a – WebSocket 지원 브라우저 확인하기

http://clearpal7.blogspot.kr/2016/07/18-websocekt-stomp_25.html - WebSocket + STOMP 활용해서 채팅 페이지 구현하기