WEB PROGRAMMING FINAL PROJECT

WEB BASED QUIZ GAME

NAME: 윤준하
STUDENT ID: 201210909
COURSE: WEB PROGRAMMING
PROFESSOR: 이효종 교수님

1. Introduction

기존 브라우저들에게 가장 큰 고민은 HTML의 DTD 참조로 인한 호환성의 문제였다. 역사상 최초로 브라우저간의 표준화가 이루어졌다. Global 5대 표준 웹 브라우저(IE,Chrome,Safari,Opera,Firfox)에서 공통 표준기술로 HTML5 버전을 채택함에 따라 웹 기술이 엄청나게 발전하였다. 또한 PC O/S와 모바일 O/S의 각종 브라우저에서도 모두 지원함에 따라 2014년 10월, W3C 차세대 웹 표준기술로 확정되었다.

이러한 웹 표준의 변화와 기술의 발전에 발맞추어 기존에 여러 형태로 존재했던 "그림 연상 퀴즈"를 HTML5의 주요 기능들과 세밀한 제어가 가능한 JSP의 성능을 통해 **"웹 브라우저만으로 실행 가능한 그림 맞추기 게임"**을 개발하고자 한다.

시작하기 앞서, HTML5의 주요 기술에 대한 호환성과 개발 가능성에 대한 여부를 확인하였다.

	IE	Chrome	Safari	Opera	Firefox
CSS 3	support	support	support	support	support
Web Sockets	10+	support	4.2+	12.1+	7+
Canvas API	support	support	support	support	support
Viewport definition	support	support	support	support	support
WebGL	support	30+	8.0b+	12+	support
AJAX 2.0	10+	support	5.0+	12+	10+

Source: mobilehtml5.org

Figure 1: HTML5 compatibility with testing on real devices (2015)

2015년 주요 브라우저들의 HTML5 호환성 통계(Figure 1)를 보면 Web Socket과 canvas를 사용하여 웹 프로젝트를 진행하여도 문제가 없을 것으로 판단하였다.

다음으로 시장현황과 전망을 분석해보았다. 스피드 퀴즈에 그림을 접목한 이러한 형태의 퍼즐 게임은 이미 여러 플랫폼과 O/S에서 개발하여 서비스되어왔다. 하지만 클라이언트에 특정 프로그램(또는 어플리케이션)을 설치해야한다는 단점을 갖고 있다. 그럼에도 불구하고 "캐치마인드"라는 DirectX 기반 윈도우 온라인 게임 서비스는 10년 이상 운영되고 있고, 실제 사용자도 꾸준히 유지되고 있는 상황이다. 비슷한 게임을 모바일 어플리케이션에서도 찾아볼 수 있다. "내가그린기린그림"의 경우는 소셜 네트워크를 접목하여 크게 성장했지만 3년을 넘지 못하고 서비스를 종료한 상태이다. 이에 반해 같은 모바일 어플리케이션 "스케치퀴즈"는 지속적으로 유지·보수되고 있다.

Figure 2는 각 게임의 간단한 스크린샷이며 왼쪽부터 캐치마인드(2001년~), 내가그린기린그림(2012~2015년), 스케치퀴즈(2011년~)이다.

Figure 2: games already had been serviced (or expired)

PC게임 시장과 모바일게임 시장 모두에서 성공적인 결과를 거두었지만, 국내에서는 웹으로 개발된 사례는 찾아볼 수 없는 상황이다. 한국콘텐츠진흥원에 따르면, [1]모바일과 태블릿 시장의 확장에 따라 HTML5 기반의 웹 활용이 늘어날 것으로 전망된다고 기술했다. 따라서 이미 시장에서 검증된 아이디어이며 웹으로 개발했을 때 두 시장 모두 공략할 수 있으므로 그 장점을 최대화할 수 있을 것으로 전망한다.

웹 기반으로 서비스 중인 게임이 해외에 존재하고 있었지만 (www.guessasketch.com), Adobe社의 Flash 기반으로 개발되었다. HTML5가 표준으로 제정됨에 따라 Flash는 비표준이고 기존의 Flash 기반 Mash-up들이 [2]HTML5의 Canvas로 대체하고 있음을 생각하면, 장기적인 측면에서 이 프로젝트의 발전 가능성은 우수하다고 볼 수 있다.

Figure 3: Adobe Flash blocked by default for all users in the latest version of browsers

2. Problem Description

이 프로젝트에서 웹 어플리케이션이 해야 할 일이 크게 서버와 클라이언트로 나뉠 수 있다.

I. 서버

1) 서버는 연결된 세션을 통해 접속한 유저들을 관리할 수 있다.
   1) 비회원으로도 접속할 수 있으며, 로그아웃시 모든 정보가 소멸된다.
   2) 회원은 로그인과 회원가입이 가능하며, 추후 접속 시 이전의 경험치나 별명이 유지된다.

1) 모바일 애플리케이션 비즈니스 현황과 전망 - 한국콘텐츠진흥원 (2011-20호)
2) Chrome의 Flash 차단 정책과 Flash에서 Canvas로 전환 사례 (http://d2.naver.com/helloworld/1899560)

2) 서버는 게임을 진행하기 위한 채팅방을 제공할 수 있다.
   1) 방에 접속하기 위한 대기실(Lobby)가 존재하며, 방 목록을 제공한다.
   2) 모든 유저가 방에서 퇴장하면 방을 자동으로 사라진다.
3) 서버는 같은 채팅방에 있는 유저들의 데이터 교환을 처리할 수 있다.
   1) 연결된 세션들을 방을 통해 알 수 있으며, 그 세션들에게 데이터를 송·수신할 수 있다.
   2) 데이터는 채팅 메시지나 방의 상태, 그림의 정보 그리고 정답 유무 등을 의미한다.
   3) 서버는 같은 방에 있는 유저들이 동일한 그림을 볼 수 있도록 한다.
4) 서버는 문제로 제시되는 정답을 관리할 수 있다.
5) 서버는 그리는 사람과 정답을 맞추는 사람의 권한을 구분할 수 있다.

II. 클라이언트

1) 클라이언트는 채팅방을 통해 게임을 시작할 수 있다.
   1) 채팅방은 대기실(Lobby)에서 입장할 수 있으며 퇴장할 수 있다.
   2) 게임을 시작하기 위해 준비할 수 있다.
2) 클라이언트는 게임 중 그림화면(캔버스)를 공유할 수 있다.
   1) 서버로부터 데이터를 수신하여 자신의 캔버스에 새로 그린다.
3) 클라이언트는 서버와 데이터를 송·수신할 수 있다.
   1) 특정 행동을 취하면 서버가 해석할 수 있도록 메시지를 송신할 수 있다.
   2) 서버로부터 받은 데이터(응답)을 해석하여 페이지를 갱신할 수 있다.
4) 클라이언트는 서버와 데이터를 송·수신할 수 있다.
   1) 특정 행동을 취하면 서버가 해석할 수 있도록 메시지를 송신할 수 있다.
5) 클라이언트에게는 그림을 그리는 권한이 존재한다.
   1) 그림을 그리는 사람은 채팅을 할 수(정답을 맞출 수) 없다.
   2) 그림을 그리지 않는 사람은 채팅을 할 수(정답을 맞출 수) 있다.

Canvas를 자주 사용하는 게임인 만큼 채팅을 제외한 많은 상호작용이 캔버스에서 이루어질 것으로 예상된다. 따라서 Canvas 기술을 기반하여 브라우저에서 풍부한 상호작용이 가능한 컨텐츠를 제공할 수 있는 프레임워크인 CreateJS(http://createjs.com/)을 선택했다. Adobe, MS, Mozilla 재단 등이 적극 후원하는 만큼 브라우저간의 호환성와 이식성에도 큰 문제가 없을 것으로 사료된다.

3. Algorithm / Implementation

1. Database Diagram

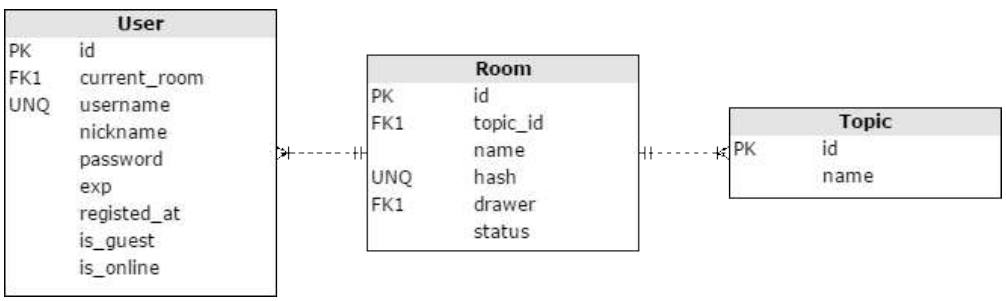

Figure 4: Diagram for Database

Topic 테이블은 게임의 진행에서 사용될 주제들을 담는다. 하나의 퀴즈에 대한 정답을 선정하기 위한 주제들을 저장하는 용도로 사용된다.

Room 테이블은 Topic 테이블의 id를 참조하는 topic_id를 가진다. 이는 방에서 게임이 진행 중일 때, 문제의 정답을 참조하기 위함이다. Room과 Topic이 긴밀히 연결된 것은 아니기 때문에 별도의 CASCADE를 지정하진 않았다.

User 테이블은 유저가 회원가입 후 로그인을 했을 때, 이전에 게임을 통해 얻은 경험치(exp)와 별명(nickname)을 저장하기 위한 용도로 사용된다. 실제 서비스를 이용하는 데에는 크게 불편함이 없다. username은 UNIQUE로 설정되어 아이디가 겹치지 않도록 했다. current_room은 Room 테이블의 hash를 참조한다. id를 참조하지 않는 이유는 특정 방에 접속하거나 User가 속한 방의 hash만 조회하려할 때 Room의 id를 참조하여 Room테이블을 매 번 검색하는 부하를 줄이기 위함이다. hash는 UNIQUE 속성을 가지고 있으므로 겹칠 경우가 없어 외래키로 지정해도 된다고 판단하였다.

2. Interface

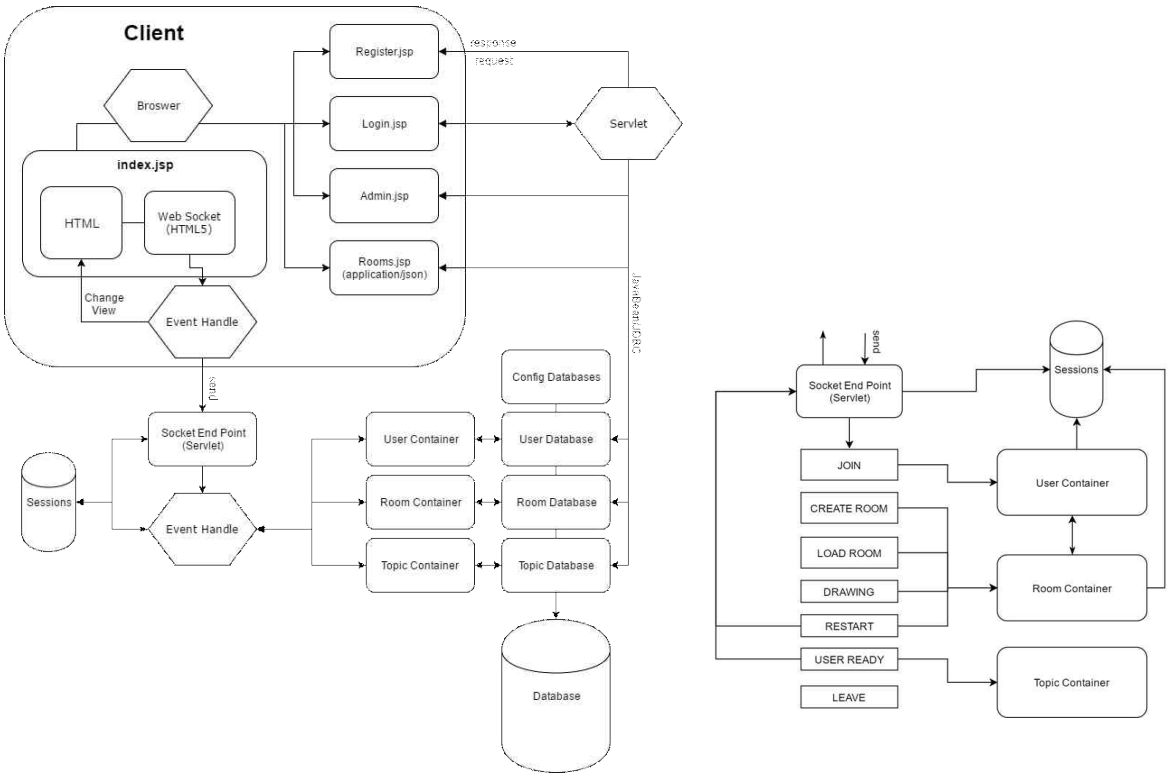

Figure 5: Flow Chart (left), and event handle on end-point (right)

전체적인 흐름은 Figure 6과 같다. 유저는 브라우저를 통해 웹 어플리케이션을 실행하게 된다. 로그인이나 회원가입, 관리자 페이지 등은 JSP를 사용하여 동적 페이지로 확인할 수 있다. 이는 JSP의 서블릿 컨테이너과 JavaBean 등을 사용하여 처리했다. JSP와 Servlet을 동시에 사용하는 MVC모델을 사용함으로써 프리젠테이션(View) 로직과 비즈니스(Model/Controller) 로직을 분리할 수 있었고, 유지보수가 용이한 구조로 구축하였다.

2-1. Server-side

실제 게임은 하나의 페이지에서 웹 소켓을 통해 비동기적으로 이루어진다. 웹 소켓은 서버에 있는 Socket End Point와 Handshaking하게 되고 클라이언트는 여기서 세션을 통해 관리된다.

서버에서 RoomContainer 등 Container 클래스를 중간에 두었는데, 웹 소켓으로 연결된 세션과 JDBC를 자연스럽게 연결하기 위해 작성했다.

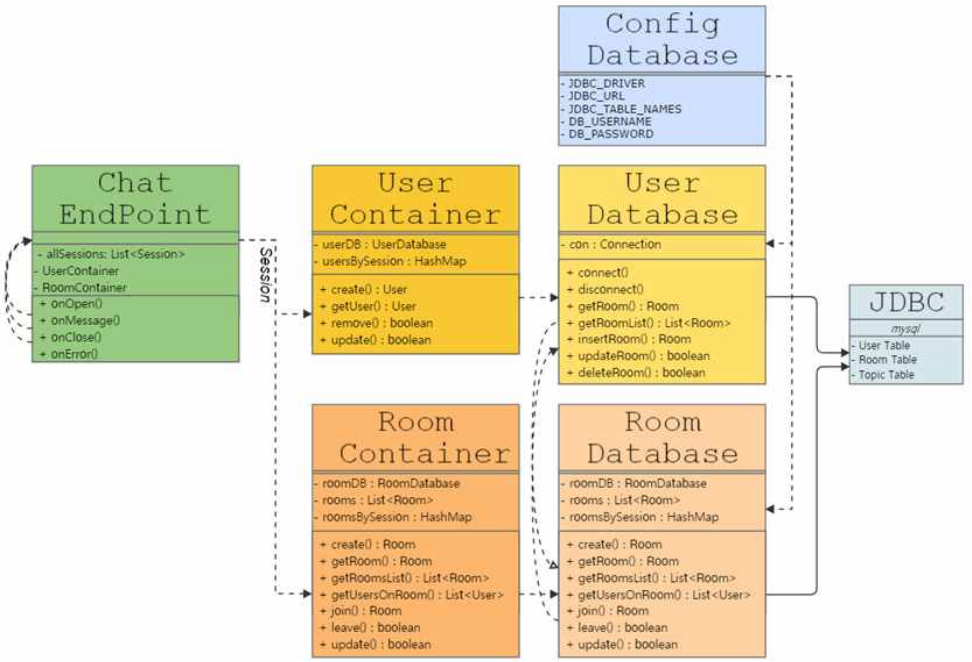

Figure 6: System Design with Container on server

SocketEndPoint는 웹 소켓을 통해 연결된 브라우저와 통신이 가능한 상태이며, 여러 세션들을 allSessions라는 리스트로 관리한다. 때문에 연결된 모든 세션과 통신이 가능하며, 가장 중요한 세션과 세션 사이의 데이터 교환에 있어서도 문제가 없다. onClose()와 onError() 메소드를 통해 연결이 끊겼을 경우, 퇴장이나 데이터베이스 관련 처리도 맡고 있다.

Container는 SocketEndPoint에서 인스턴스로 생성되지만, 서버가 실행되는 동안에 한 Container에 대해 하나의 인스턴스만 존재할 수 있도록 *싱글톤 패턴* 을 사용하였다. 웹에서 진행되는 실시간 게임이라는 특징상 세션을 통해 유저를 구분해야 하는 데, 그러기 위해선 세션의 정보를 구분하고 이를 게임과 관련된 정보와 연결되도록 해야한다. 이를 위해 접속한 세션이 로그인중이라면 UserContainer를 통해 사용자의 정보를 데이터베이스로부터 가져올 수 있게 하였고, 세션이 특정 방에 입장하는 등 방과 관련된 행동은 모두 RoomContainer에서 처리하도록 비즈니스 로직을 다시 분리하였다.

그렇기 때문에 SocketEndPoint는 온전히 클라이언트로부터 온 메시지들을 분리하여 알맞은 처리를 선택하기만 하고, 각 Container들이 그에 대한 처리를 함으로써 유지·보수가 편리할 수 있게 하였다. (Figure 5.right)

2-2. Client-side

실시간 게임에서 페이지의 이동이 잦으면 유저로 하여금 불편함을 호소하게 만든다. 또한 페이지의 불필요한 로드가 많아지므로 많은 요청과 느린 응답이 발생할 것이다. 따라서 하나의 페이지에서 진행되되,

UI와 Element들에 변화를 줌으로써 유저의 자연스러운 행동을 유발하고자 하였다. 그 첫 번째는 비동기통신을 사용하여 해결하려는 시도였다. 이는 웹 소켓에서 자체적으로 안정적인 비동기통신을 사용하고 있었다. 두 번째는 페이지의 변화를 주기위해 javascript, jQuery 그리고 적절한 프레임워크인 Semantic-ui로 페이지의 변화를 주는 것이었다. 이 과정에서 방을 만들기 위한 Dialog를 생성하는 등의 자연스러운 고급 효과를 추가하였다.

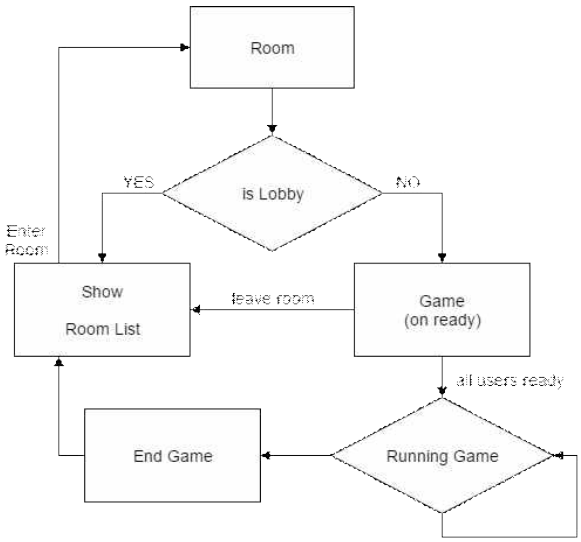

Figure 7: Work flow by user's perspective

3. Implements

먼저, 로그인과 회원가입 그리고 관리자 페이지 등의 JSP 코드부터 살펴보겠다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" import="chat.user.*" %>
<%
 User user_session = (User)session.getAttribute("user");
if( user_session != null ) response.sendRedirect("index.jsp");
```

Figure 8-1: Using JSP Implict Object (session)

JSP View 영역에서 내장 객체인 Session을 이용하여 로그인 정보를 유지하도록 하였다. 이미 로그인 중이라면 index.jsp로 돌아가게 하는 등으로 구현하였고, logout.jsp에서는 session 객체에서 user를 삭제하도록 하였다.

```
<jsp:useBean id="userDB" class="chat.user.UserDatabase" scope="page" />
<%
String username = request.getParameter("username");
String password = request.getParameter("password");
if( userDB.login(username, password) == false ){
%>
<script>alert("로그인에 실패했습니다!");location.href='login.jsp';</script>
<%
} else {
%>
<script>
sessionStorage.setItem('username', "<%= username %>");
</script>
<%
  session.setAttribute("user", userDB.getUser(username));
```

Figure 8-2: jsp to process login (login.exe.jsp)

Form 태그에 아이디와 비밀번호를 입력하면 login.exe.jsp 라는 처리를 위한 JSP 페이지로 연결되는데, JavaBean 액션 태그를 사용하여 DB에서 조회하고, 그 결과를 출력하도록 하였다. JSP 내장 객체인

session에도 받아온 User 객체를 저장하고, 클라이언트에서도 이를 알고있기 위해 HTML5에서 지원하는 브라우저 내부 세션 window.sessionStorage에도 저장하도록 하였다.

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
2  <%@ page import="java.sql.*,java.util.*,java.text.*,javax.servlet.http.*,chat.room.*,chat.user.*,game.topic.*" %>
6  <jsp:useBean id="topicDB" class="game.topic.TopicDatabase" scope="page" />
22 <%
23 ArrayList<Topic> Topics = topicDB.getTopicList();
24 %>
26   <jsp:include page="admin.header.html"></jsp:include>
35           <label>목록 보기 및 추가</label>
36           <select name="topic_names[]" multiple class="ui search multiple topics dropdown">
37             <option value="">Topics</option>
38             <% for(Topic t : Topics){ %>
39             <option value="<%= t.getTopic().toUpperCase() %>"><%= t.getTopic().toUpperCase() %></option>
40             <% } %>
```

Figure 8-3: get topic list by JDBC (admin.topic.jsp)

관리자 페이지 역시 JavaBean 액션 태그를 사용하여 JDBC를 통해 현재 서버에 등록되어있는 주제(정답)들의 목록을 조회하고, 추가할 수 있다.

Figure 8-4: input form can be multiple select (left) and processing multiple topics on admin.topic.exe.jsp (right)

Semantic UI에서 제공하는 자바스크립트를 사용하여 전체 주제 목록들이 압축되어 보일 수 있도록 하였다. 중복된 항목은 입력될 수 없도록 하였으며, 쉼표로 여러개의 항목을 추가할 수 있다. 이미 존재하는 항목에 대해서는 경고창이 뜨고 다시 돌아온다. 존재 여부와 항목 추가에 대한 결과는 TopicDatabase 클래스에서 insertTopics() 메소드를 호출한다.

- Web socket on client

index.jsp에서 HTML5의 웹 소켓을 사용하여 모든 이벤트를 처리할 수 있도록 구현하였다. 웹 소켓과 연결되는 부분부터 게임과 관련된 모든 부분을 자바스크립트로 구현하였고 이를 모듈화하였다.

```
113 var proxy = CreateProxy("ws://"+getCurrentUrlPath()+"/chat");
114
115 document.addEventListener("DOMContentLoaded", function(event) {
118     proxy.initiate({
119       chatBox: document.getElementById('chatBox'),
126       stage: "canvas"
127     });
128     proxy.run();
130     $(".card.room").on('click', function(e){
131       proxy.join($(this).data('href'));
132     });
133
134     $("#createRoom form").on('submit', function(e){
135       proxy.makeRoom(formSerialize(e.target));
136     });
```

Figure 9-1: modularize web socket processing

chatroom.js에 작성한 CreateProxy 생성자를 통해 웹 소켓과 게임을 연결하는 객체를 생성한다. 이 생성자는 클라이언트 브라우저에서 자바스크립트를 통해 사용자가 join(), makeRoom() 등의 함수를

호출하면 서버에 적절한 데이터를 전송하고, 클라이언트의 페이지에 변화를 주도록 구현하였다.

```
setInterval(function(){
  if( proxy.getSocket() == null ){
    // proxy.run();
    $("#dimmer").dimmer('show');
    $("#dimmer .text").html('접속이 끊겼습니다.<br>새로고침을 해주세요');
  }
}, 2000);
```

Figure 9-2: modularize web socket processing (2)

클라이언트는 2초 간격으로 서버의 상태를 확인하고, 연결이 끊어지면 화면을 어둡게 하는 레이어를 표시하여 연결이 끊어졌음을 알린다. 주석된 proxy.run()을 해제하면 연결이 끊어졌을 경우에는 다시 연결을 시도한다.

```
62  var CreateProxy = function(wsUri) {
63      var websocket = null;
64      var audio = null;
65      var elements = null;
66
67      var stage = null;
68      var holder = null;
428     return {
495         join: join,
496         sendMessage: function() {
497             elements.messageBox.focus();
499             if (websocket != null && websocket.readyState == 1) {
500                 var input = elements.messageBox.value;
501                 if (input == '') { return; }
503                 elements.messageBox.value = '';
505                 var message = { messageType: 'MESSAGE', message: input };
508                 websocket.send(JSON.stringify(message));
509             }
510         },
511         sendMessage_keyup: function(e) {
512             if (e.keyCode == 13) {
513                 this.sendMessage();
514             }
515         },
516         logout: function() {
517             if (websocket != null && websocket.readyState == 1) { websocket.close();}
```

Figure 10: CreateProxy Module

웹 소켓의 연결 주소를 받아 소켓을 열고 연결한다. logout 함수에서는 연결을 끊는 것을 확인할 수 있다. 그 외에 sendMessage 등 특정 동작을 취하는 경우 함수를 호출하여 서버에게 데이터를 전송하거나 요청하도록 함수를 작성하였고, sendMessage()의 경우 sendMessage_keyup을 키보드 이벤트와 바인딩하여 사용자가 채팅창에서 Enter키를 입력하는 경우마다 서버에 메시지를 전송하도록 하였다. 전송되는 데이터는 JSON형태로 전송하기 쉽도록 구현하였으며, 돌아오는 응답 역시 JSON의 형태로 구현됐다.

- HTML & Javascript for User Interface

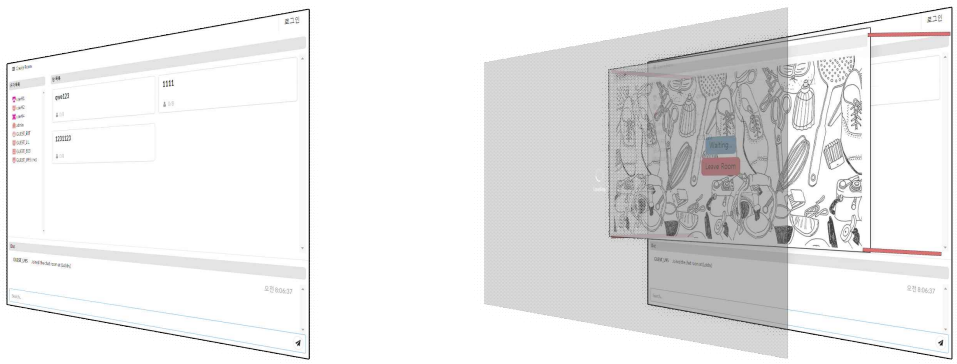

Figure 11: layered UI

하나의 페이지에서 엘리먼트들의 변화만으로 방의 이동과 게임 화면을 전환하기 위해 요소들의 위치를

변경하고 표시 여부를 조정하면서 구조를 유지하였다. 따라서 유저에게 마치 페이지가 이동된 것 같은 착각을 일으킬 수 있다. 방에 입장하게 되면 캔버스가 있는 박스를 채팅방 목록이 있던 자리에 보이게 하고, 유저 목록과 채팅창의 상태를 갱신하였다.

```
var loadRoomList = function() {
    elements.roomList.innerHTML = '';
    $.ajax({
        type: "POST",
        url: 'restful/rooms.jsp',
        dataType: 'json',
        timeout: 2000,
        beforeSend: function() {
            elements.roomList.innerHTML = '';
        },
        complete: function() {
            $(".card.room").on('click', function(e){
                proxy.join($(this).data('href'));
            });
        },
        success: function(list) {
            elements.roomList.innerHTML = '';
            for(var r in list){
                var room = list[r];
                if(room.hash.toLowerCase() == 'lobby') continue;
                var str = '<a class="ui card room" data-href="'+room.hash+'" style="float: left;"><div class="content">
                str += '<div class="header">'+room.name+'</div></div><div class="extra content"><div class="left floate
                str += (room.users.length || 0) +'/8</div></div></a>';
                elements.roomList.innerHTML += str;
            }
        },
```

Figure 11: using ajax to load list of rooms

loadRoomList는 어떠한 세션에서 RoomContainer를 통한 방 정보의 변동이 생기면 서버로부터 호출 요청을 받고 호출된다. 함수가 실행되면 ajax를 이용한 비동기 통신을 사용하여 JavaBean으로 작성된 rooms.jsp의 응답을 가져온다. rooms.jsp는 모든 방의 정보를 json으로 출력한다. 유저는 이러한 함수간의 대화를 통해 유저 목록이 자연스럽게 갱신되는 것을 확인할 수 있다.

- Image processing on Canvas

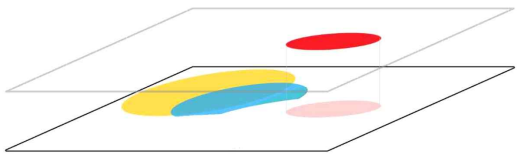

Figure 12: A stage is the root level Container for a display list – CreateJS

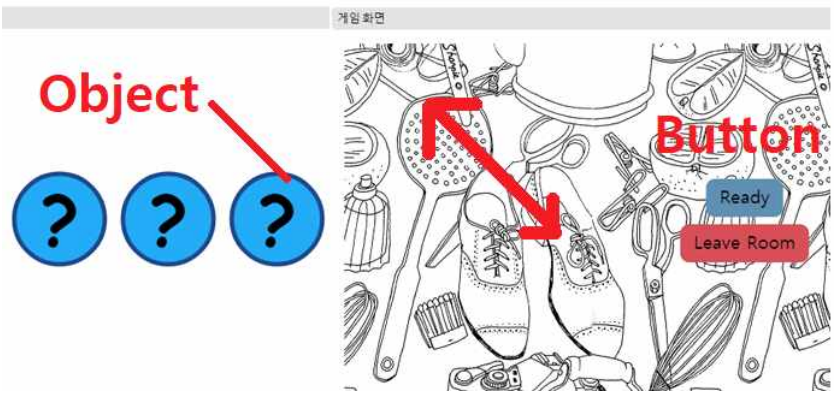

Figure 12-1: objects can be animated and buttons on canvas with createjs

기존의 캔버스에서는 레이어의 개념이 존재하지 않았다. 하지만, Create.js 프레임워크를 사용하면 캔버스 내에 여러 객체들이 존재하는 것처럼 다룰 수 있어서 이러한 복잡한 구현이 생략된다. 그 결과, 캔버스 위에 클릭 가능한 버튼이나, 텍스트의 이동 등으로 화려한 인터페이스를 제공할 수 있었다.

- Web Socket on server

```
package web.socket;
@ServerEndpoint("/chat")
public class ChatEndpoint {
    private Logger log = Logger.getLogger(ChatEndpoint.class.getSimpleName());
    private UserContainer userContainer = UserContainer.initialize();
    private RoomContainer roomContainer = RoomContainer.initialize();
    private static ArrayList<Session> allSessions = new ArrayList<Session>();

    public void open(final Session session, EndpointConfig config){
    public void onMessage(final Session session, final String messageJson) {
    public void onClose(Session session, CloseReason reason) {
    public void onError(Session session, Throwable ex) {
    public void broadcastReload(){
}
```

Figure 13: End-point on server to handle web socket

웹 소켓 프로토콜(ws://...)을 통해 접속한 클라이언트의 세션은 ChatEndPoint 클래스의 각 메소드들과 연결된다. 소켓이 연결되면 open메소드를 통해 allSessions에 연결된 세션이 저장된다. 저장된 세션은 이후 다른 세션들과 정보 전달 시에 사용되며 게임을 진행함에 있어 id가 되는 중요한 key이기도 하다. 실제로 클라이언트와 서버가 서로 대화하는 부분은 onMessage에서 모두 이루어진다. 여기선 클라이언트로부터 받은 JSON형태의 데이터를 파싱하여 분석한다. 받은 메시지의 타입을 분류하기 위해 chat.message 패키지아래에 ChageMessage 와 MessageType 이라는 클래스를 두어 구분하기 용이하도록 하였다.

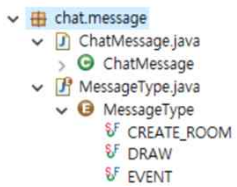

Figure 13-1: Message classes for distinguish message(or event) type

때문에 추후에 새로운 기능이 추가되는 경우 그 처리를 하기 쉽도록 확장이 용이한 구조로 구현했다. 이러한 MessageType을 통해 내부적으로 switch문으로 이벤트를 구분하여 처리한다. 아래는 그 예시로, 방에 입장하는 경우이다.

```
switch(chatMessage.getMessageType()){
case JOIN:
    roomhash = chatMessage.getMessage();
    if( roomhash == null ) roomhash = "lobby";
    roomContainer.leave(session);

    myRoom = roomContainer.join(session, roomhash);
    if( myRoom == null ){
        // no room
    } else {
        properties.put("roomhash", myRoom.getHash());
        currentUser.setCurrent_room(myRoom.getHash());
        myRoom.sendMessage("{\"eventType\":\"message\", \"user\":"+currentUser.toJson()
            +", \"room\":"+myRoom.toJson()+",\"message\":\"Joined the chat room at (" + myRoom.getName() + ")\"}");
    }
    userContainer.update(session, currentUser);
    broadcastReload();
    break;
```

Figure 13-2: handle event by using switch-case with MessageType class

방에 입장할 때에 기존의 방을 퇴장하고, 현재 입장을 요청한 방에 세션을 입장시킨다. 방이 존재하는 경우에는 세션의 정보에 연결된 방의 hash값을 저장하고 Room클래스의 sendMessage()를 통해 같은 방에 존재하는 모든 세션들에게 입장했음을 알리도록 소켓을 통해 메시지를 전송한다. 마지막으로 유저와 세션의 동기화를 위해 userContainer가 유저의 정보를 갱신한다. boardcastReload 함수는 모든 세션들에게 각자 자신이 속한 방의 상태를 업데이트하라는 신호를 broadcast한다.

이처럼 서버에서 실행되고 있는 Socket의 End-point 클래스는 메시지를 구분하여 상황에 맞는 처리를 한다. 다음으로, 각 Container들의 정의와 구현에 대해 설명하고자 한다.

- Containers for synchronizing

서버에서 메시지를 구분하여 각 컨테이너들에게 특정 행동을 요구한다. 주로 세션과 관련된 처리를 담당한다. 예를 들면 세션 정보와 함께 새로운 정보를 추가하거나, 동일한 세션에 연결된 어떤 정보를 변경하거나 삭제한다. 그리고 그 정보와 연결된 관계에서 데이터베이스의 갱신이 필요하다면 JDBC로 처리한다. 각 테이블에 맞게 정의하여 JDBC를 사용하는 Databases 클래스는 다음 장에서 다룬다.

각 컨테이너들은 역할에 맞는 적절한 메소드들이 정의되어있다. (Figure 6) 예를 들어, RoomContainer와 같은 경우에는 다음과 같은 메소드와 속성들(attributes)을 갖는다.

```
 1 package chat.room;
12 public class RoomContainer {
13
14     private static RoomContainer instance = null;
15     private HashMap<String, Room> rooms = new HashMap<String, Room>();
16     private HashMap<Session, Room> roomsBySession = new HashMap<Session, Room>();
17
18     RoomDatabase roomDB = new RoomDatabase();
19
20+    public void RoomConatiner(){
23+    public synchronized static RoomContainer initialize() {
27+    public synchronized Room getRoom(String room_hash){
36+    public synchronized Room getRoom(Session session){
39+    public synchronized Room join(Session session, String room_hash) {
49+    public synchronized Room create(Session session, String room_title){
61+    public synchronized void leave(Session session) {
79+    /** returns an array containing all ChatRoom objects
81+    public ArrayList<Room> getRoomListArray()
85+    public ArrayList<Session> getUsersOnRoom(String room_hash)
91 }
```

Figure 14: RoomContainer class for sync between session and informations on database

싱글톤 패턴[3]으로 구현되어 있으며, HashMap형태의 rooms와 roomsBySession라는 속성을 갖고 있다. rooms는 방의 목록을 hash값으로만 가지고 있으며, 부여한 hash값이 어떤 방(Room 객체)을 의미하는 지 연결하는 데 사용된다. 또한, 이미 존재하는 hash는 생성되지 않도록 HashMap의 형태를 띈다.

roomsBySession은 End-Point에서 연결된 소켓의 세션이 현재 어떤 방에 속해있는 지 세션을 key로 저장한다. 때문에 세션은 RoomContainer에게 어떤 작업을 요청하는 즉시, 연결된 Room 객체를 얻어 정보를 조회하거나 수정할 수 있다.

3) https://en.wikipedia.org/wiki/Singleton_pattern

하나의 예시로, 새로운 방을 만드는 경우에 세션을 두 HashMap에 갱신하고, 새로운 hash 값을 부여하여 서버의 메모리와 DB에 모두 저장한다. DB에는 JDBC를 사용하는 RoomDatabase를 통해 요청한다.

```
49    public synchronized Room create(Session session, String room_title){
50        instance.leave(session);
52        Room newRoom = new Room();
53        newRoom.setName(room_title);
54        newRoom.setHash(new BigInteger(130, new SecureRandom()).toString(32).toLowerCase().substring(0,10));
56        instance.rooms.put(newRoom.getHash(), newRoom);
57        instance.roomsBySession.put(session, newRoom);
58        roomDB.insertRoom(newRoom);
59        return instance.join(session, newRoom.getHash());
60    }
```

Figure 14-1: implementation `create` method on RoomContainer class

- Databases Class using JDBC

각 컨테이너 또는 JSP의 JavaBean의 요청을 처리하기 위한 클래스이다. 여러 데이터베이스가 공통된 연결 정보를 갖고 있으므로 설정 정보만을 갖는 ConfigDatabase를 참조한다. 이로 인해 ConfigDabases만 서버 설정에 따라 바꾸면 포팅과 마이그레이션이 쉽고 빠르게 가능해진다.

```
1   package chat.room;
2
15  public class RoomDatabase {
16
17      private static final String JDBC_DRIVER = ConfigDatabase.JDBC_DRIVER;
18      private static final String JDBC_URL = ConfigDatabase.JDBC_URL;
19      private static final String DB_USERNAME = ConfigDatabase.DB_USERNAME;
20      private static final String DB_PASSWORD = ConfigDatabase.DB_PASSWORD;
21
22      private static final String ROOM_TABLE = ConfigDatabase.ROOM_TABLE;
23      private static final String TOPIC_TABLE = ConfigDatabase.TOPIC_TABLE;

25      private Connection con = null;
28      private UserDatabase userDB = new UserDatabase();

30      public RoomDatabase() {
38      public void connect() {
46      public void disconnect() {
144     public Room insertRoom(Room newRoom){
145         connect();
146
147         String SQL = "INSERT INTO " + ROOM_TABLE + " (name, hash) values (?, ?)";
148         PreparedStatement pstmt;
149         try {
150             pstmt = con.prepareStatement(SQL);
151             pstmt.setString(1, newRoom.getName());
152             pstmt.setString(2, newRoom.getHash());
153             pstmt.executeUpdate();
154             pstmt.clearParameters();
155         } catch (SQLException e) {
156             e.printStackTrace();
157         } finally {
158             disconnect();
159         }
160
161         return getRoom(newRoom.getHash());
162     }
```

Figure 15: implementation `create` method on RoomContainer class

방을 추가하는 경우, 적당한 쿼리문을 만들어 처리한다. 각 요청에 맞게 쿼리문을 생성하도록 구현하였고, 사용자의 정보가 필요한 경우에는 UserDatabase의 인스턴스인 userDB를 통해 접근/조회할 수 있다. RoomDatabase의 getTopicByRoom 메소드에서는 Inner Join을 사용한다.

각 Container 클래스들과 Database 클래스들의 호출 관계에 대해서는 <Figure 5. Flow chart>를 통해 확인할 수 있다.

3. Results

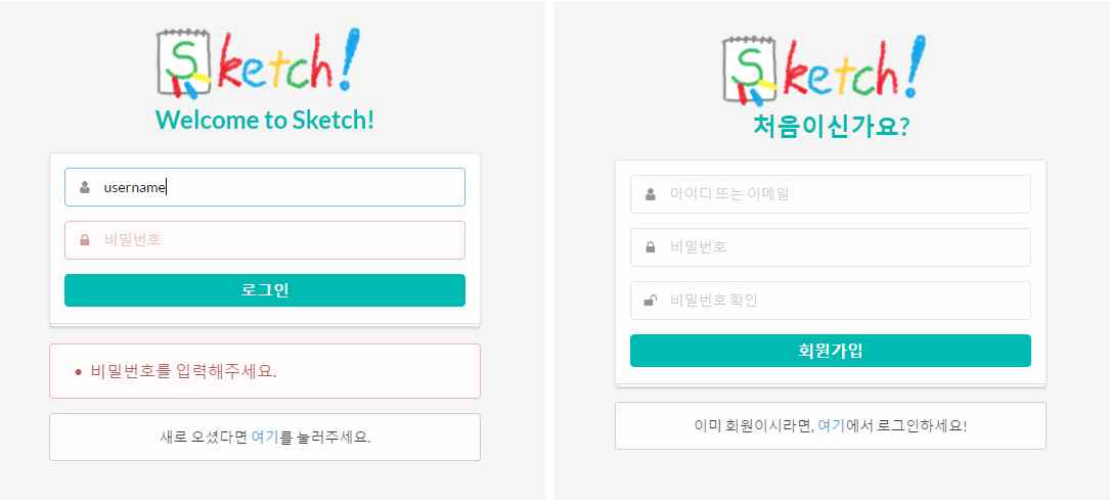

Figure 16: Sign-in Page (left) and Register Page (right)

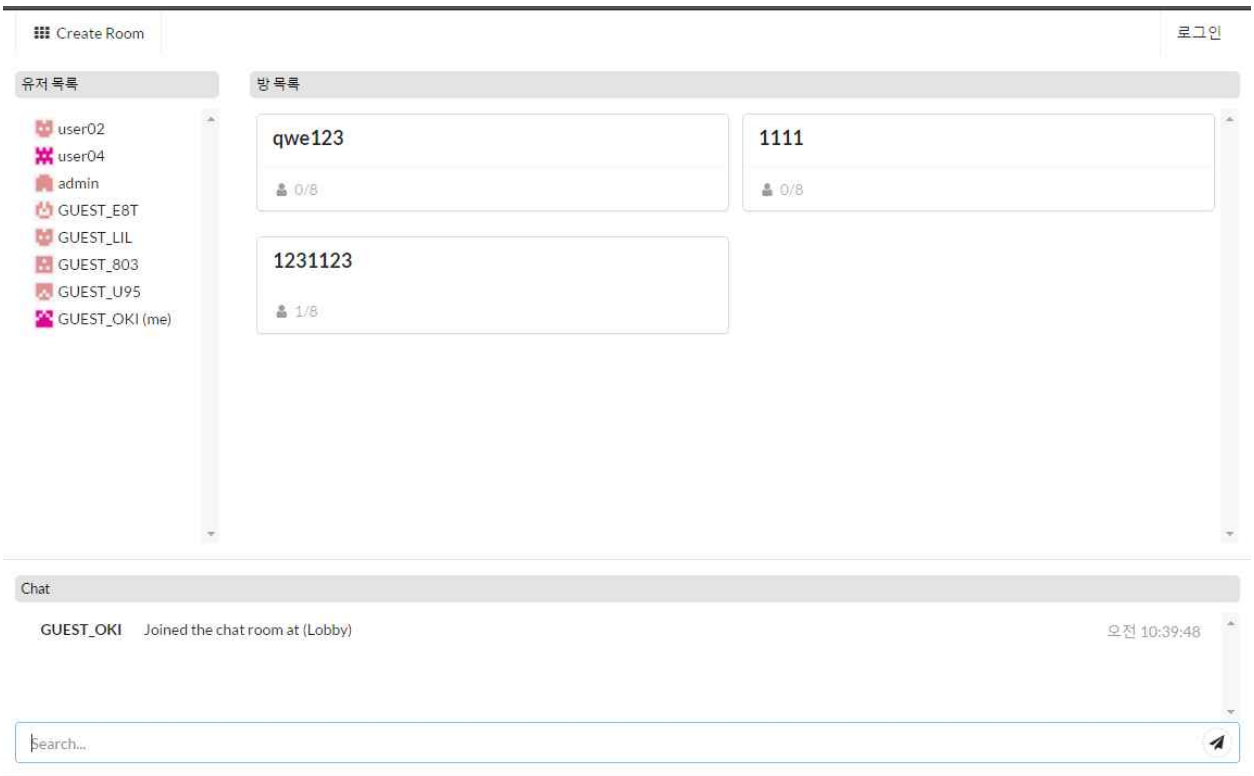

Figure 17: Default Layout on lobby

로비인 경우 좌측 상단에 'Create Room' 버튼이 있으며, 클릭 시 작은 Dialog로 방을 생성할 수 있다. 좌측의 유저 목록을 통해 현재 방에 접속 중인 유저들의 목록을 확인할 수 있으며, 우측의 방 목록으로 현재 생성된 방들을 확인할 수 있다. 아래에는 채팅창이 있어 대화하거나 정답을 맞출 수 있다.

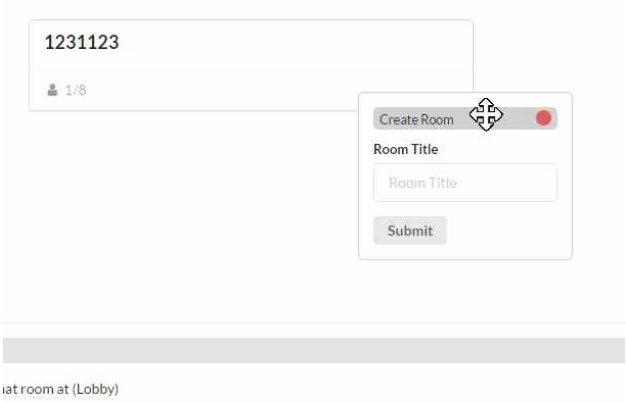

Figure 17-1: draggable dialog to create room

방을 생성하기 위한 작은 Dialog는 마우스 커서를 이용해 화면 내에서 움직일 수 있다.

Figure 18: Dimmer when socket disconnect

접속이 끊어진 경우에는 화면이 어두워지면서 안내 메시지가 출력된다.

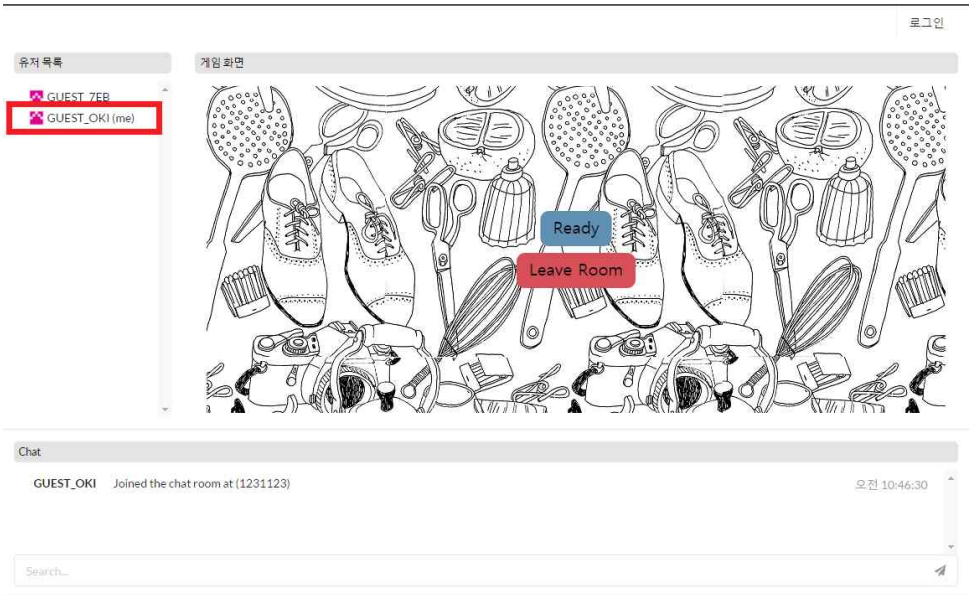

Figure 19: Room which waiting other users to game

방에 입장하면 '방 목록'이 '게임 화면'으로 바뀐 것을 확인할 수 있다. 유저 목록에서 (me) 표시로 자신을

구분할 수 있으며, 좌측 상단에 있었던 'Create Room' 버튼도 사라진 것을 알 수 있다.

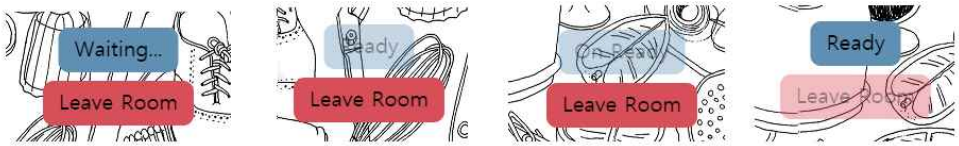

Figure 19-1: buttons on game canvas

캔버스에 그려진 버튼들에 마우스를 올리면 반투명으로 변한다. Ready를 클릭하면 On Ready로 메시지가 변하면서 버튼을 클릭할 수 없는 형태로 바뀐다. 또한 방에 혼자 있는 경우에는 Waiting... 이라며 게임 시작을 위해 준비도 할 수 없음을 알 수 있다.

Figure 20-1: client's screenshot when game started

모든 유저가 준비하면 게임이 시작되고, 현재 그림을 그리는 사람이 유저 목록에 빨간 배경으로 표시됨을 볼 수 있다. 게임의 첫 시작화면은 정답의 길이를 알려주면서 시작된다.

Figure 20-2: drawer's screen when game started

그림을 그리는 사람은 채팅이 비활성화가 되면서 자신이 그림을 그린 사람으로 바뀐 것을 확인할 수 있다. 그리고 canvas에는 정답이 출력되고 캔버스에 그림을 그릴 수 있는 상태가 된다.

캔버스 위에 찍힌 검은색 점이 그림을 그리기 위한 마우스 포인터(브러쉬)이다.

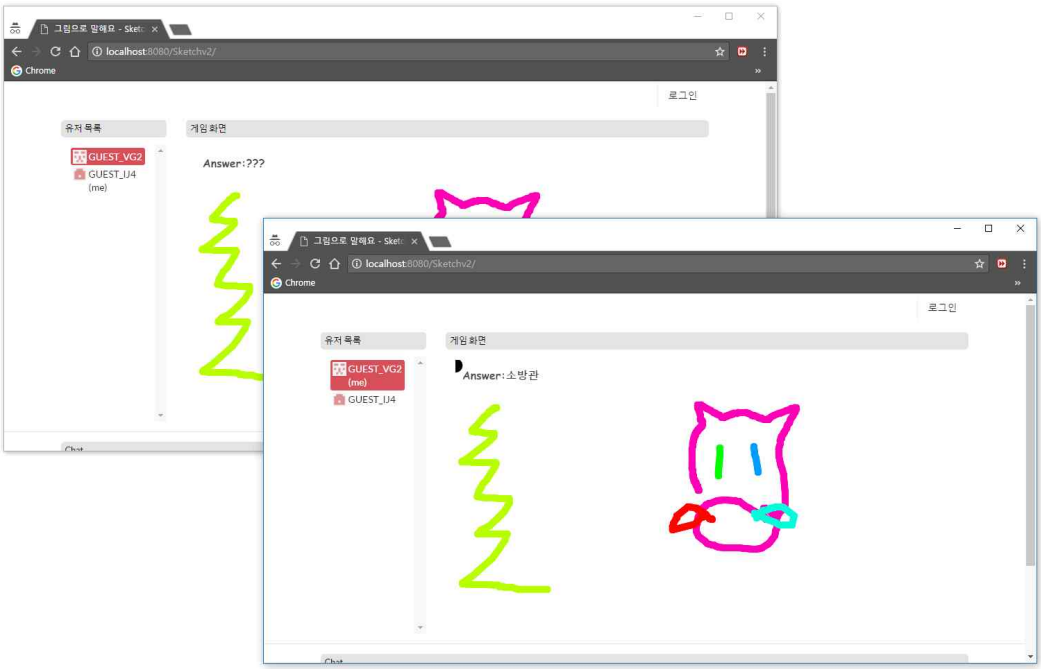

Figure 20-3: well synchronized

그림을 그리면 서버를 통해 해당 방에 있는 모든 유저들에게 그림이 잘 동기화되어 그려지는 것을 확인할 수 있다.

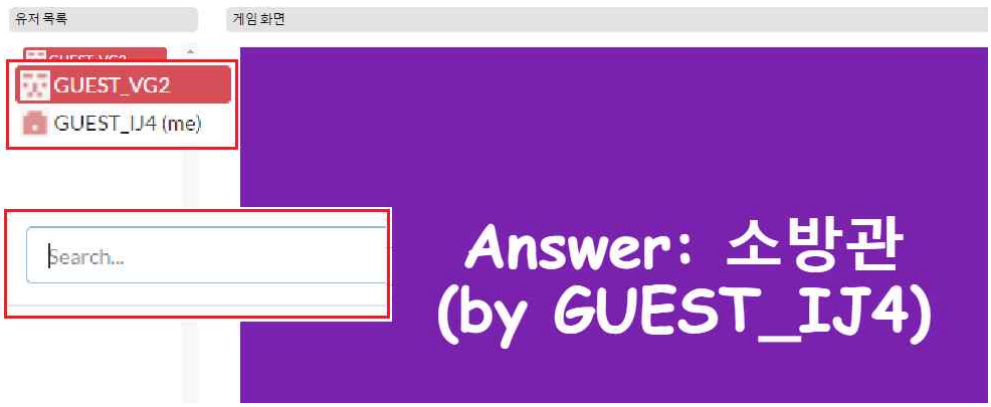

Figure 20-4: getting answer and take a turn

다른 유저가 채팅을 통해 정답을 맞추면 나머지 유저들을 위해 정답과 정답을 맞춘 유저가 출력되고 그림을 그리는 사람이 바뀌면서 채팅이 활성화되는 것을 볼 수 있다.

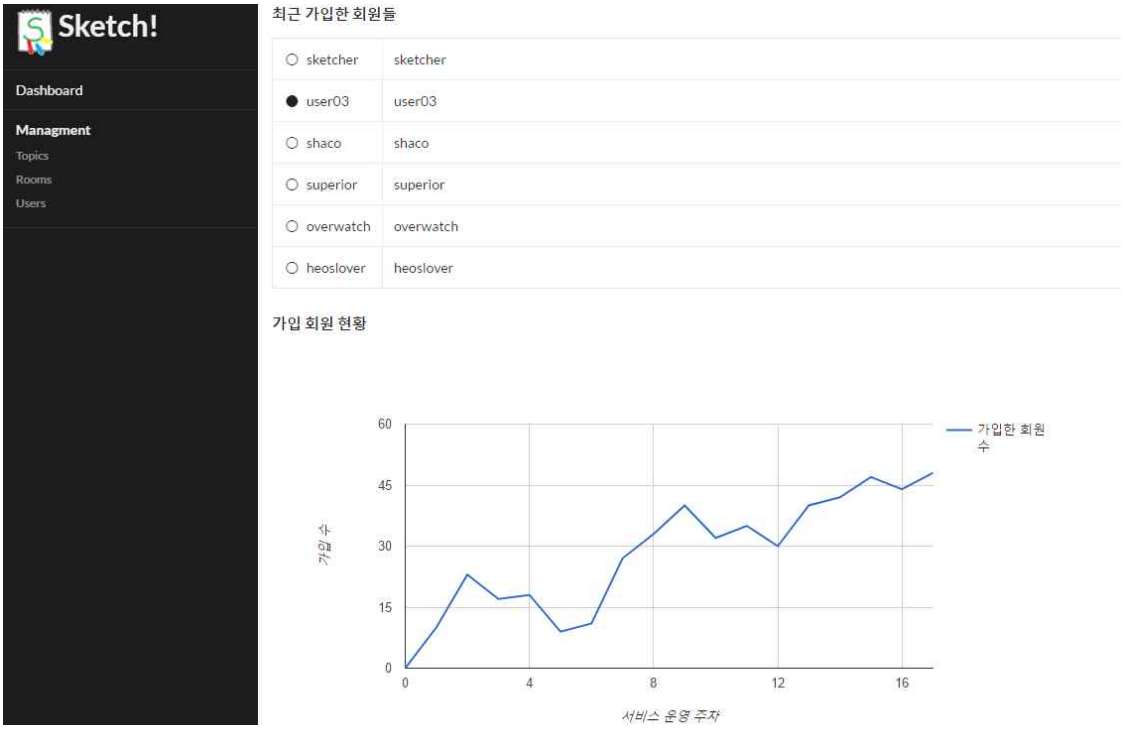

Figure 21: dashboard on admin page

관리자로 로그인하면 관리자페이지로 연결된다. 최근 가입한 회원들의 목록을 볼 수 있고, 검은색으로 표시된 회원은 “현재 접속중”이라는 의미이다. UserContainer에서 통계 테이블과 연결하기만 하면 곧장 서비스 운영 주차에 따른 가입 회원 수를 차트의 형태로 출력할 수 있지만, 현재는 틀만 남은 상태이다.

Figure 22-1: topic management page for admin

주제(정답)을 관리하는 페이지이다. 기존의 주제들의 사용 빈도를 차트로 확인할 수 있도록 표시하였으며, 이는 사용자 통계와 동일하게 동작하도록 설계를 예정하였다.

문제 관리에서 새로운 항목을 작성하여 ‘추가하기’ 버튼을 누르면 차트와 선택 상자에 추가된다.

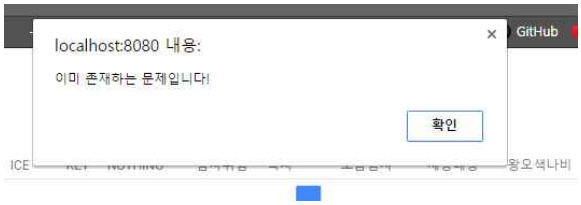

Figure 22-2: alert with topic already exists

이미 존재하는 문제에 대해서는 추가되지 않도록 한 후, 경고 메시지를 출력하는 것을 볼 수 있다.

4. Conclusion

기존 Java의 장점을 그대로 가져와 웹 서비스가 가능하도록 제공되는 JSP는 JavaBeans나 액션 태그들을 통해 강력한 구현의 용이성을 보였다. 특히 JavaBeans는 (정보를 사용자에게 보여주는)뷰와 (사용자들에게 보여주기 위해 사용되는)프로그램 구현 코드를 완벽하게 분리함으로써 개발의 편의성도 제공했다.

하지만 최신 웹 개발 시장에 비해 생산성은 조금 떨어지는 경향이 있었다. 이를 보안하기 위해 Spring MVC라는 강력한 풀스택 프레임워크가 개발되었지만 그 비율은 점차 줄어드는 추세이다. 또한, 메모리 누수가 발생할 수 있음에 주의해야 한다는 것 때문에 싱글톤 패턴을 사용하는 등의 노력을 기울였다.

프로젝트를 진행함에 있어서 부딪혔던 가장 큰 문제점은, HTML5 WebSocket이였다. HTML5의 websocket을 지원하지 못하는 브라우저가 조사한 결과 이상으로 많은 비중을 차지하고 있을뿐더러, 소켓의 연결이 지속적으로 불안정하였다. 많은 서비스에서 기존의 일반 http 스펙에 실시간 통신을 흉내내어 소켓과 동일한 형태로 안정적인 통신을 지원하는 솔루션들이 존재했다. (SockJS, Socket.io 등) JSP에서는 Spring MVC framework에서 SockJS를 통해 실시간 통신을 지원하고 있었다. 따라서 현재 개발된 순수 HTML5 websocket 기반 통신에서 안정적인 소켓 솔루션으로 수정해야 할 필요성이 보인다.

CreateJS라는 캔버스 기반 이미지 렌더링 라이브러리는 성능이 뛰어났다. 자체적인 캐시 관리와 이미지 처리를 통해 캔버스에 그림을 빠르게 그려내었고, 객체 단위로 조작할 수가 있어서 고수준의 개발이 가능하였다.

그 외에 Back-end에서 Front-end와 처리 형식을 맞추기 위해 JSON형태로 통일한 점은 검증이 필요해보인다. 객체의 변환이 정확히 이루어지지 않는 경우도 많았고, 불필요한 클래스가 생기기도 하였다. 하지만 이것은 Jackson JSON 라이브러리의 특징인 것으로 추측된다. 동일한 기능을 제공하는 Google의 Gson의 경우는 조금 더 직관적이고 고수준으로, 불필요한 코드가 줄어든 것을 보았다.

Databases의 설계면에서 아쉬움이 많이 남는다. 실시간 통신 기반 게임이라는 점에서 Database를 사용할 일이 적었다. 그리고 Database 과목을 수강하지 않아 부정확한 면모가 일부분에서 드러난다.

하지만 각 클래스들의 역할을 분리하고 그 관계를 정의함으로써 개발의 확장성을 충분히 확보하였다고 판단된다. 이 프로젝트를 개발하면서도 기능을 붙여나가는 식으로 개발했으므로 조금은 검증을 거쳤다고 생각한다. 혼자 진행한 프로젝트라 많은 시간을 투자하지 못함과 기획과 설계한 만큼 결과물을 만들어내지 못한 것에 대한 아쉬움이 많이 남지만, 프로젝트를 통해 여러 라이브러리와 기술 동향을 살폈고 그 문제점과 대체 방안을 알아보았다. 이를 거름삼아 이후에 더 나은 프로젝트를 개발하는 데에 도움이 될 것이라 생각한다.

5. Sources

- CreateJS (http://createjs.com/) : Rich interactive content with Canvas API
- Google Charts (https://developers.google.com/chart/) : Interactive charts on admin page
- Semantic UI (https://semantic-ui.com/) : for User Interface all around site
- jQuery UI (https://jqueryui.com/) : using draggable dialog with creating room
- identicon.js (https://github.com/stewartlord/identicon.js/) : identifier with image
- PNGlib.js (https://github.com/michelem09/pnglib.js) : dependency by identicon.js
- FasterXML/jackson (http://jackson.codehaus.org/) : parsing JSON on server-side