Communication

Web Socket

- Web Socket
  - ✓ 순수 웹 환경에서 실시간 양방향 통신을 위한 Spec이 바로 '웹 소켓(Web Socket)'
  - ✓ 웹 소켓은 웹 서버와 웹 브라우저가 지속적으로 연결된 TCP 라인을 통해 실시간으로 데이터를 주고 받을 수 있도록 하는 HTML5의 새로운 사양으로 웹 소켓을 이용하면 일반적인 TCP소켓과 같이 연결지향 양방향 전이중 통신이 가능
  - ✓ 웹 소켓이 필요한 경우
    - 실시간 양방향 데이터 통신이 필요한 경우
    - 많은 수의 동시 접속자를 수용해야 하는 경우
    - 브라우저에서 TCP 기반의 통신으로 확장해야 하는 경우
    - 개발자에게 사용하기 쉬운 API가 필요할 경우
    - 클라우드 환경이나 웹을 넘어 SOA 로 확장해야 하는 경우

Web Socket

- Web Socket
  - ✓ 웹 소켓 역시 일반적인 TCP 소켓 통신처럼 웹 소켓 역시 서버와 클라이언트간 데이터 교환이 이루어지는 형태로 다른 HTML5 스펙과는 달리 클라이언트 코드만으로 실행 가능한 코드를 만들 수 없음
  - ✓ 웹 소켓 클라이언트
    - 웹 소켓이 제공하는 클라이언트 측 API는 매우 심플한데 기본적인 서버 연결, 데이터 송신, 데이터 수신만 정의하면 나머지는 일반적인 스크립트 로직
    - 웹 소켓이 동작하기 위해서 제일 처음 서버와 연결이 되어야 하는데 HTML5가 제공하는 WebSocket 객체를 통해 서버 연결을 수행

      let wSocket = new WebSocket("ws://yourdomain/demo");
    - 서버와 연결이 되면 이제부터 데이터를 주고 받을 수 있게 되는데 WebSocket 객체의 send 함수로 데이터를 서버로 송신할 수 있음

      wSocket.send(" 송신 메시지 ");
    - 서버에서 푸시(전송)하는 데이터를 받으려면 message 이벤트를 구현

      wSocket.onmessage = function(e){ }//매개변수 e를 통해 수신된 데이터를 조회
    - open 이벤트: 연결이 설정되면 발생
    - close 이벤트: 연결이 끊어지면 발생

Web Socket

- Spring Web Socket
  - ✓ websocket 의존성 설정
  - ✓ 백그라운드로 돌아가는 빈을 등록하기 위해 @Service 어노테이션과 WebSocket의 연결점을 알려주는 @ServerEndoint 어노테이션을 지정한 클래스를 생성하는데 @ServerEndpoint는 WebSocket을 활성화시키는 매핑 정보를 지정
  - ✓ 위에서 지정한 클래스는 클라이언트가 접속할 때마다 생성되어 클라이언트와 직접 통신하는 클래스로 새로운 클라이언트가 접속할 때마다 클라이언트의 세션 관련 정보를 정적형으로 저장하여 통신이 가능하도록 작성
  - ✓ 클라이언트의 접속, 메시지 수신, 접속 해제에 따른 이벤트 핸들러를 어노테이션과 함께 메서드를 작성

어노테이션	설 명
@OnOpen	클라이어트가 접속할 때 발생하는 이벤트
@OnClose	클라이언트가 브라우저를 끄거나 다른 경로로 이동할 때
@OnMessage	메시지가 수신되었을 때

  - ✓ 일반적으로 스프링에서 빈들은 싱글톤으로 관리되지만 @ServerEndpoint 어노테이션이 달린 클래스들은 WebSocket이 생성될 때마다 인스턴스가 생성되고 JWA에 의해 관리되기 때문에 스프링의 @Autowired가 설정된 멤버들이 정상적으로 초기화 되지 않기 때문에 이를 연결해 주고 초기화해 주는 클래스가 필요

Web Socket

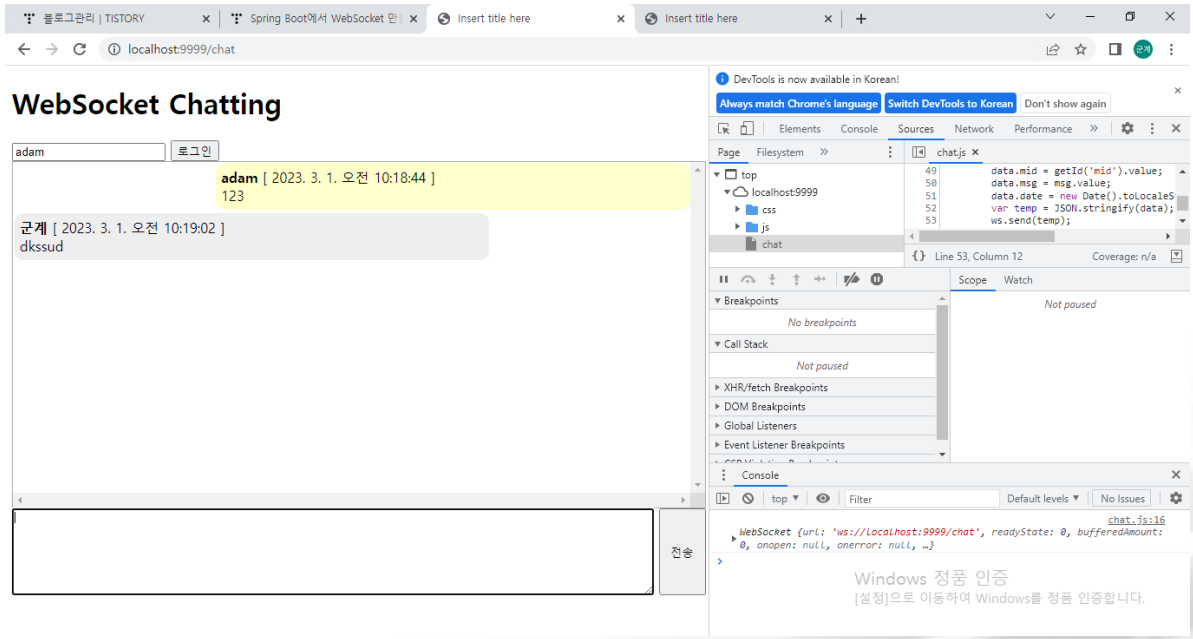
- Spring Web Socket
  - ✓ websocket 의존성 설정
  - ✓ 백그라운드로 돌아가는 빈을 등록하기 위해 @Service 어노테이션과 WebSocket의 연결점을 알려주는 @ServerEndoint 어노테이션을 지정한 클래스를 생성하는데 @ServerEndpoint는 WebSocket을 활성화시키는 매핑 정보를 지정
  - ✓ 위에서 지정한 클래스는 클라이언트가 접속할 때마다 생성되어 클라이언트와 직접 통신하는 클래스로 새로운 클라이언트가 접속할 때마다 클라이언트의 세션 관련 정보를 정적형으로 저장하여 통신이 가능하도록 작성
  - ✓ 클라이언트의 접속, 메시지 수신, 접속 해제에 따른 이벤트 핸들러를 어노테이션과 함께 메서드를 작성

어노테이션	설 명
@OnOpen	클라이어트가 접속할 때 발생하는 이벤트
@OnClose	클라이언트가 브라우저를 끄거나 다른 경로로 이동할 때
@OnMessage	메시지가 수신되었을 때

  - ✓ 일반적으로 스프링에서 빈들은 싱글톤으로 관리되지만 @ServerEndpoint 어노테이션이 달린 클래스들은 WebSocket이 생성될 때마다 인스턴스가 생성되고 JWA에 의해 관리되기 때문에 스프링의 @Autowired가 설정된 멤버들이 정상적으로 초기화 되지 않기 때문에 이를 연결해 주고 초기화해 주는 클래스가 필요

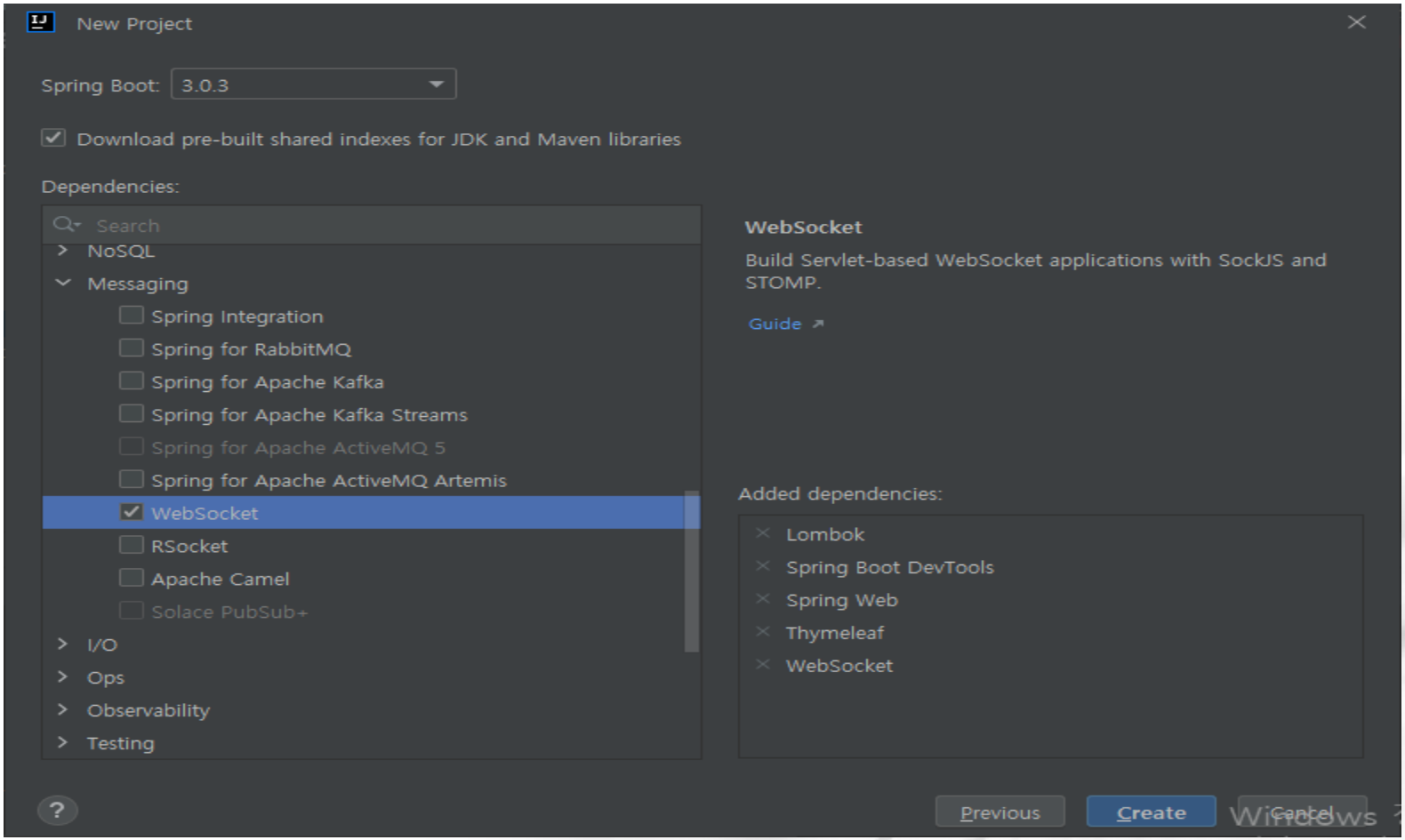
Web Socket

❖ 웹 소켓을 이용한 채팅



Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ Spring Boot Project 생성(communication)



Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ application.properties 파일을 제거하고 application.yml 파일을 생성하고 작성

```
server:
  port: 9999

spring:
  thymeleaf:
    cache: false

  #Live Reload 기능 활성화
  devtools:
    livereload:
      enabled: true
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 채팅을 위한 웹 소켓 클래스 생성 후 작성 – service.WebSocketChat

```
@Service
@ServerEndpoint(value="/chat")
public class WebSocketChat {
    private static Set<Session> clients =
            Collections.synchronizedSet(new HashSet<Session>());


    @OnOpen
    public void onOpen(Session s) {
        System.out.println("open session : " + s.toString());
        if(!clients.contains(s)) {
            clients.add(s);
            System.out.println("session open : " + s);
        }else {
            System.out.println("이미 연결된 session 임!!!");
        }
    }
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 채팅을 위한 웹 소켓 클래스 생성 후 작성 – service.WebSocketChat

```
    @OnMessage
    public void onMessage(String msg, Session session) throws Exception{
        System.out.println("receive message : " + msg);
        for(Session s : clients) {
            System.out.println("send data : " + msg);
            s.getBasicRemote().sendText(msg);

        }

    }

    @OnClose
    public void onClose(Session s) {
        System.out.println("session close : " + s);
        clients.remove(s);
    }
}
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 웹 소켓 클래스 설정 클래스 생성 후 작성 – config.WebSocketConfig

```
@Component
public class WebSocketConfig {

    @Bean
    public ServerEndpointExporter serverEndpointExporter() {
        return new ServerEndpointExporter();
    }
}
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ Controller 클래스 생성 후 작성 – controller.ChatController

```
@Controller
public class ChatController {

    @RequestMapping("/chat")
    public ModelAndView chatt() {
        ModelAndView mv = new ModelAndView();
        mv.setViewName("chatting");
        return mv;
    }
}
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 화면 출력을 위한 html 파일을 생성 – templates/chatting.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Insert title here</title>
    <link rel='stylesheet' type='text/css' href='./css/chat.css'>
</head>
<body>
<div id='chat'>
    <h1>WebSocket Chatting</h1>
    <input type='text' id='mid' value='noname'>
    <input type='button' value='로그인' id='btnLogin'>
    <br/>
    <div id='talk'></div>
    <div id='sendZone'>
        <textarea id='msg' value='hi...' ></textarea>
        <input type='button' value='전송' id='btnSend'>
    </div>
</div>
<script src='./js/chat.js'></script>
</body>
</html>
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 화면 디자인을 위한 css 파일을 생성 – static/css/chat.css

```
@charset "UTF-8";

*{
    box-sizing: border-box;
}

#chat{
    width: 800px;
    margin: 20px auto;
}

#chat #talk{
    width: 800px;
    height: 400px;
    overflow: scroll;
    border : 1px solid #aaa;
}
#chat #msg{
    width: 740px;
    height:100px;
    display: inline-block;
}
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 화면 디자인을 위한 css 파일을 생성 – static/css/chat.css

```
#chat #sendZone > *{
    vertical-align: top;

}
#chat #btnSend{
    width: 54px;
    height: 100px;
}
#chat #talk div{
    width: 70%;
    display: inline-block;
    padding: 6px;
    border-radius:10px;

}
#chat .me{
    background-color : #ffc;
    margin : 1px 0px 2px 30%;
}
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 화면 디자인을 위한 css 파일을 생성 – static/css/chat.css

```
#chat .other{
    background-color : #eee;
    margin : 2px;
}
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 웹 소켓 통신을 위한 js 파일을 생성 – static/css/chat.js

```
function getId(id){
    return document.getElementById(id);
}

let data = {};//전송 데이터(JSON)

let ws ;
let mid = getId('mid');
let btnLogin = getId('btnLogin');
let btnSend = getId('btnSend');
let talk = getId('talk');
let msg = getId('msg');
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 웹 소켓 통신을 위한 js 파일을 생성 – static/css/chat.js

```
btnLogin.onclick = function(){
    ws = new WebSocket("ws://" + "172.30.66.24:9999" + "/chat");
    console.log(ws)
    ws.onmessage = function(msg){
        let data = JSON.parse(msg.data);
        let css;

        if(data.mid == mid.value){
            css = 'class=me';
        }else{
            css = 'class=other';
        }

        let item = `<div ${css} >
                        <span><b>${data.mid}</b></span> [ ${data.date} ]<br/>
                    <span>${data.msg}</span>
                                                    </div>`;

        talk.innerHTML += item;
        talk.scrollTop=talk.scrollHeight;//스크롤바 하단으로 이동
    }
}
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 웹 소켓 통신을 위한 js 파일을 생성 – static/css/chat.js

```
msg.onkeyup = function(ev){
    if(ev.keyCode == 13){
        send();
    }
}

btnSend.onclick = function(){
    send();
}

function send(){
    if(msg.value.trim() != ''){
        data.mid = getId('mid').value;
        data.msg = msg.value;
        data.date = new Date().toLocaleString();
        let temp = JSON.stringify(data);
        ws.send(temp);
    }
    msg.value ='';

}
```

Web Socket

- ❖ 웹 소켓을 이용한 채팅
  - ✓ 리액트 프로젝트
    - App.js 수정

```
import './App.css';
import React, {useState, useEffect} from 'react';

function App() {

  const [mid, setMid] = useState('noname');
  const [msg, setMsg] = useState('');
  const [ws, setWs] = useState();

  useEffect(() => {
    if(ws !== undefined){
      ws.onmessage = function(msg){
        let data = JSON.parse(msg.data);
        let css;

        if(data.mid === mid){
            css = 'class=me';
        }else{
            css = 'class=other';
        }
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 리액트 프로젝트
    - App.js 수정

```
let item = `<div ${css} >
        <span><b>${data.mid}</b></span> [ ${data.date} ]<br/>
          <span>${data.msg}</span>
        </div>`;

        let talk = document.getElementById("talk");
        talk.innerHTML += item;
        talk.scrollTop=talk.scrollHeight;

    }
    }
  }, [ws, mid])
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 리액트 프로젝트
    - App.js 수정

```
const onMidChange = (e) => {
  setMid(e.target.value);
}

const onMsgChange = (e) => {
  setMsg(e.target.value);
}

const onClick =  () => {
  setWs(new WebSocket("ws://192.168.0.20:9999/chat"));
}

const onSendClick = () => {
  send();
}

const handleKeyPress = (e) => {
  if(e.key === 'Enter'){
    send()
  }
}
```

Web Socket

- ❖ 웹 소켓을 이용한 채팅
  - ✓ 리액트 프로젝트
    - App.js 수정

```
const send = () => {
  let data = {}
  data.mid = mid
  data.msg = msg;
  data.date = new Date().toLocaleString();
  let temp = JSON.stringify(data);
  ws.send(temp);
  setMsg('')
}
```

Web Socket

- ❖ 웹 소켓을 이용한 채팅
  - ✓ 리액트 프로젝트
    - App.js 수정

```
return (
  <div className="App">
    <div id='chat'>
    <h1>WebSocket Chatting</h1>
    <input type='text' value={mid} onChange={onMidChange}/>
    <input type='button' value='로그인' onClick={onClick}/>
    <br/>
    <div id='talk'></div>
    <div id='sendZone'>
      <textarea id='msg' value={msg} onChange={onMsgChange}
   onKeyUp={handleKeyPress}></textarea>
      <input type='button' value='전송' id='btnSend' onClick={onSendClick}
   />
    </div>
    </div>
  </div>
 );
}

export default App;
```

Web Socket

- ❖ 웹 소켓을 이용한 채팅
  - ✓ 리액트 프로젝트
    - App.css 수정

```
@charset "UTF-8";

*{
    box-sizing: border-box;
}

#chat{
    width: 800px;
    margin: 20px auto;
}

#chat #talk{
    width: 800px;
    height: 400px;
    overflow: scroll;
    border : 1px solid #aaa;
}
```

Web Socket

- ❖ 웹 소켓을 이용한 채팅
  - ✓ 리액트 프로젝트
    - App.css 수정

```
#chat #msg{
    width: 740px;
    height:100px;
    display: inline-block;
}

#chat #sendZone > *{
    vertical-align: top;

}
#chat #btnSend{
    width: 54px;
    height: 100px;
}
```

Web Socket

- 웹 소켓을 이용한 채팅
  - ✓ 리액트 프로젝트
    - App.css 수정

```
#chat #talk div{
    width: 70%;
    display: inline-block;
    padding: 6px;
    border-radius:10px;

}

#chat .me{
    background-color : #ffc;
    margin : 1px 0px 2px 30%;
}

#chat .other{
    background-color : #eee;
    margin : 2px;
}
```

Web Push

- Web Push
  - ✓ Server-Sent Events
    - HTML5가 등장하기 전까지는 HTML에 서버 푸시를 위한 표준화된 기술이 없었기 때문에 웹에서 실시간 정보를 받아와야 할 때 외부 플러그 인을 이용하거나 서버 푸시를 흉내 낸 Ajax 폴링(polling) 기법 등을 사용
    - 플러그인 종속적인 웹은 해당 플러그 인을 설치해야 하는 불편함이 있으며 폴링처럼 주기적인 요청을 통한 구현은 쓸모없는 요청의 발생으로 인한 대역폭의 낭비가 불가피
    - HTML5의 Server-Sent Events(이하 SSE)는 이러한 문제 없이 서버가 필요할 때마다 클라이언트에게 데이터를 줄 수 있게 해주는 서버 푸시 기술
  - ✓ SSE의 장점
    - 전통적인 HTTP를 통해 통신하므로 다른 프로토콜이 필요가 없음
    - 재접속 처리 같은 대부분의 저수준 처리가 자동
    - IE 하위 브라우저를 제외한 브라우저 대부분을 지원

Web Push

- Client
  - ✓ 클라이언트에서는 다음 코드처럼 EventSource 객체를 생성하는 것만으로 주기적인 서버 호출이 일어남

```
let eventSource = new EventSource("server.jsp");
```

  - ✓ 데이터 수신 이벤트: 서버로 부터 전달받는 데이터를 처리하기 위해 수신 이벤트를 정의
    - EventSource 객체의 message 이벤트를 통해 정의
    - 이벤트로 전달되는 data 속성으로 수신 데이터를 액세스

```
eventSource.addEventListener("message",
    function(e){
      alert(e.data);
    }
,false);
```

Web Push

- Server
  - ✓ 서버 측에서 클라이언트로 전달하는 데이터는 일반적인 텍스트 형태이지만 그 포맷이 정해져 있으며 몇 가지 규칙을 따라야 함
    - MIME 타입: 서버 데이터는 text/event-stream 라는 MIME 타입으로 제공
    - 문자 인코딩: 서버 데이터의 문자 인코딩은 UTF-8 형식
    - 빈 줄은 이벤트를 구분하는 역할
    - 주석은 :(콜론)
    - 데이터는 '필드 명: 필드 값' 형식
      - data : 서버가 전달할 실제 데이터를 정의
      - retry: 반복 주기를 설정(단위: millisecond)
      - event: 이벤트 이름을 지정(지정하지 않으면 기본값인 message 가 된다)
      - id: 이벤트 id를 지정

Web Push

- 구현
  - ✓ chat.html 파일에 요청 과 출력 영역 작성

```
<a href="#" id="push" >웹 푸시 시작</a><br/>
메시지:<span id="disp"></span><br/>
```

Web Push

- 구현
  - ✓ chat.html 스크립트 코드 작성

```
<script>
        window.addEventListener('load', function(e) {
            document.getElementById("push").addEventListener("click", function(e){
                let eventSource = new EventSource("push");
                eventSource.onmessage = function(event) {
                    document.getElementById('disp').innerHTML = event.data + "<br />";
                };
            })
        })
</script>
```

Web Push

❖ 구현

✓ Push 서비스를 위한 Service 클래스를 생성하고 작성

```
@Service
public class WebPushService {
    public void push(HttpServletRequest request, HttpServletResponse response) {
        PrintWriter writer = null;
        try {
            response.setContentType("text/event-stream");
            response.setCharacterEncoding("UTF-8");
            writer = response.getWriter();
            Random r = new Random();
            writer.write("data: " + (r.nextInt(46)+1) + "₩n₩n");
            Thread.sleep(5000);
        } catch (Exception e) {
            e.printStackTrace();
        }
        writer.close();
    }
}
```

Web Push

❖ 구현

✓ ChatController 수정

```
@Controller
@RequiredArgsConstructor
public class ChatController {
    private final WebPushService webPushService;

    @RequestMapping("push")
    public void push(HttpServletRequest request, HttpServletResponse response){
        webPushService.push(request, response);
    }

    @RequestMapping("/chat")
    public ModelAndView chatt() {
        ModelAndView mv = new ModelAndView();
        mv.setViewName("chatting");
        return mv;
    }
}
```

Web Push

- ❖ 구현
  - ✓ react
    - ● package.json 파일에 Proxy 설정을 추가

      "proxy": "http://172.30.30.93:9999",

Web Push

- ❖ 구현
  - ✓ react
    - ● App.js 파일에 스크립트 코드 추가

```
const pushClick = (e) =>{
    let eventSource = new EventSource("/push");
    eventSource.onmessage = function(event) {
      document.getElementById('disp').innerHTML = event.data + "<br />";
    };
}
```

Web Push

- ❖ 구현
  - ✓ react
    - ● App.js 파일에 랜더링 코드 추가

```
<a href="#" id="push" onClick={pushClick}>웹 푸시 시작</a><br/>
메시지:<span id="disp"></span><br/>
```

Proxy

- CORS(Cross-Origin Resource Sharing)
  - ✓ 웹 페이지의 제한된 자원을 외부 도메인에서 접근을 허용해주는 매커니즘
  - ✓ 서로 다른 도메인에서 리소스를 공유하는 방식
  - ✓ 기존의 있던 Same-origin policy(동일 출처 정책)와 반대되는 개념
  - ✓ Same-Origin Policy 정책
    - www.example.com 이라는 사이트에서 사용하는 api가 있는 경우 해당 api는 외부에 공개하려는 목적이 아닌 자신들의 사이트에서 사용하기 위해 만들어진 경우 이 api를 다른 사이트에서 알게 되었고 허락없이 무단으로 가져다 사용하게 된다면 이 사이트 입장에서는 상당히 곤란함
    - 프로토콜, 도메인, 포트가 모두 같을 때 데이터를 사용할 수 있도록 하는 정책을 동일 출처 정책이라고 함
    - api를 사용할 때 ajax 나 Fetch API 를 사용하는데 이 same-origin policy를 따르지 않을 경우 오류가 나면서 api 사용이 거절됨
    - 한 사이트에서 여러 도메인을 가지고 있을 경우에 문제가 생기게 되는데 단지 도메인만 다를 뿐 똑같은 서비스라고 하더라도 동일 출처 정책을 따르지 않기 때문에 그 자원을 사용할 수 없게 됨

Proxy

- CORS(Cross-origin resource sharing)
  - ✓ 클라이언트 측에서의 동일 출처 정책 해결책
    - JSONP(Json with padding) 방식
    - js, css등의 파일은 same-origin policy를 따르지 않아도 되기 때문에 서버에서 값을 돌려줄 때 마치 js 파일처럼 값을 돌려주고 클라이언트에서는 이 값을 콜백 함수로 재처리 하여 사용하는 방식
  - ✓ 서버 측에서의 해결책
    - 스프링 4.2 부터 지원되는 @CrossOrigin 어노테이션은 CORS를 스프링을 통해 설정할 수 있는데 @CrossOrigin 어노테이션을 붙여주면 기본적으로 모든 도메인, 모든 요청 방식 에 대해 허용 한다는 의미
    - 메서드 와 클래스 상단에 작성 가능
    - @CrossOrigin 어노테이션은 여러 Properties를 가지고 있는데 그 중 origins는 허용할 도메인을 나타내는 것으로 복수 개일 경우 콤마로 구분하여 넣어주면 됨

Proxy

- @CrossOrigin

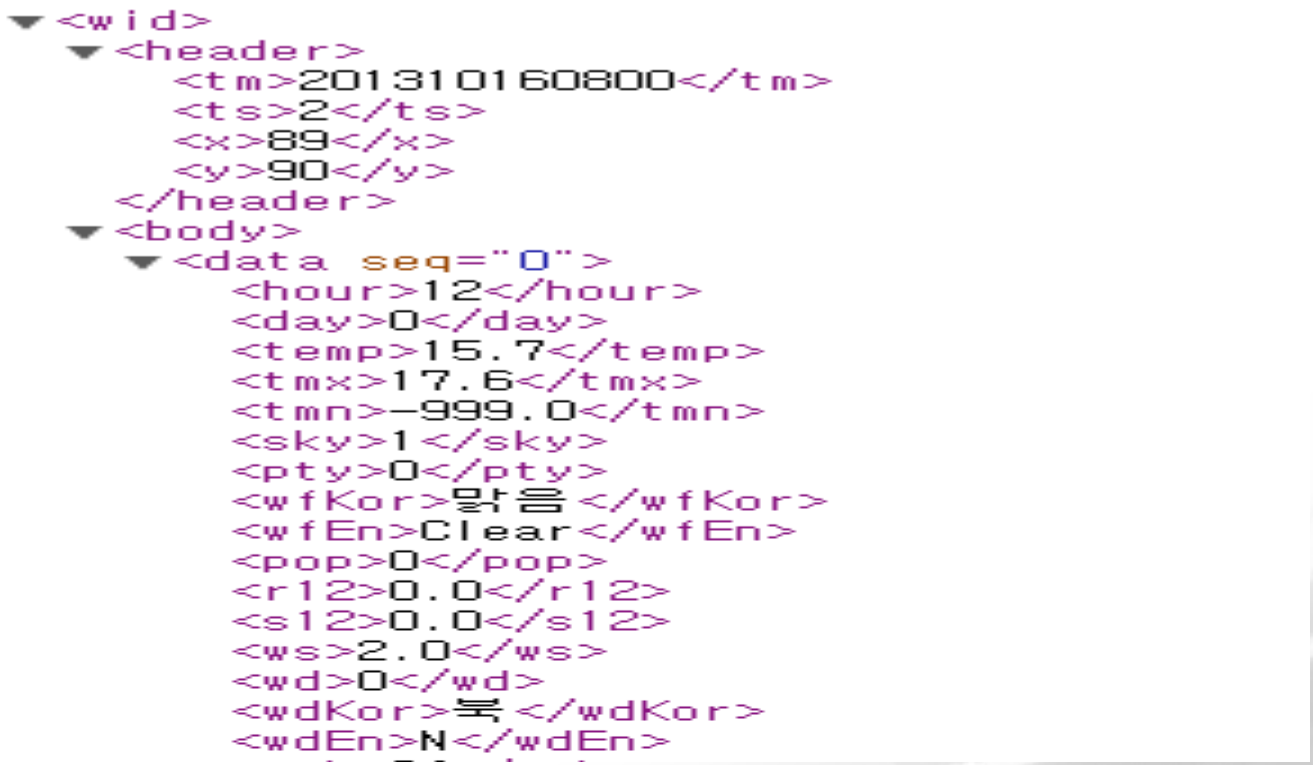
```
@CrossOrigin(origins = "http://domain1.com, http://domain2.com")
@RequestController
@RequestMapping("/account")
public class AccountController{

    @RequestMapping("/{id}")
    public Account retrieve(@PathVariable Long id){
    // ...
    }

    @RequestMapping(method = RequestMethod.DELETE, value = "/{id}")
    public void remove(@PathVariable Long id){
    // ...
    }
}
```

Proxy

- Proxy Server
  - ✓ ajax로는 CORS가 적용되지 않은 다른 도메인의 데이터를 가져올 수 없는데 Proxy Server를 구축해서 Proxy Server를 이용해서 외부 데이터를 가져온 후 서버에게 요청해서 데이터를 가져와야 함
  - ✓ Proxy Server를 구축하는 방법은 프로그래밍 언어 별로 다른데 Java 에서는 HttpURLConnection 클래스를 이용해서 URL 통신이 가능

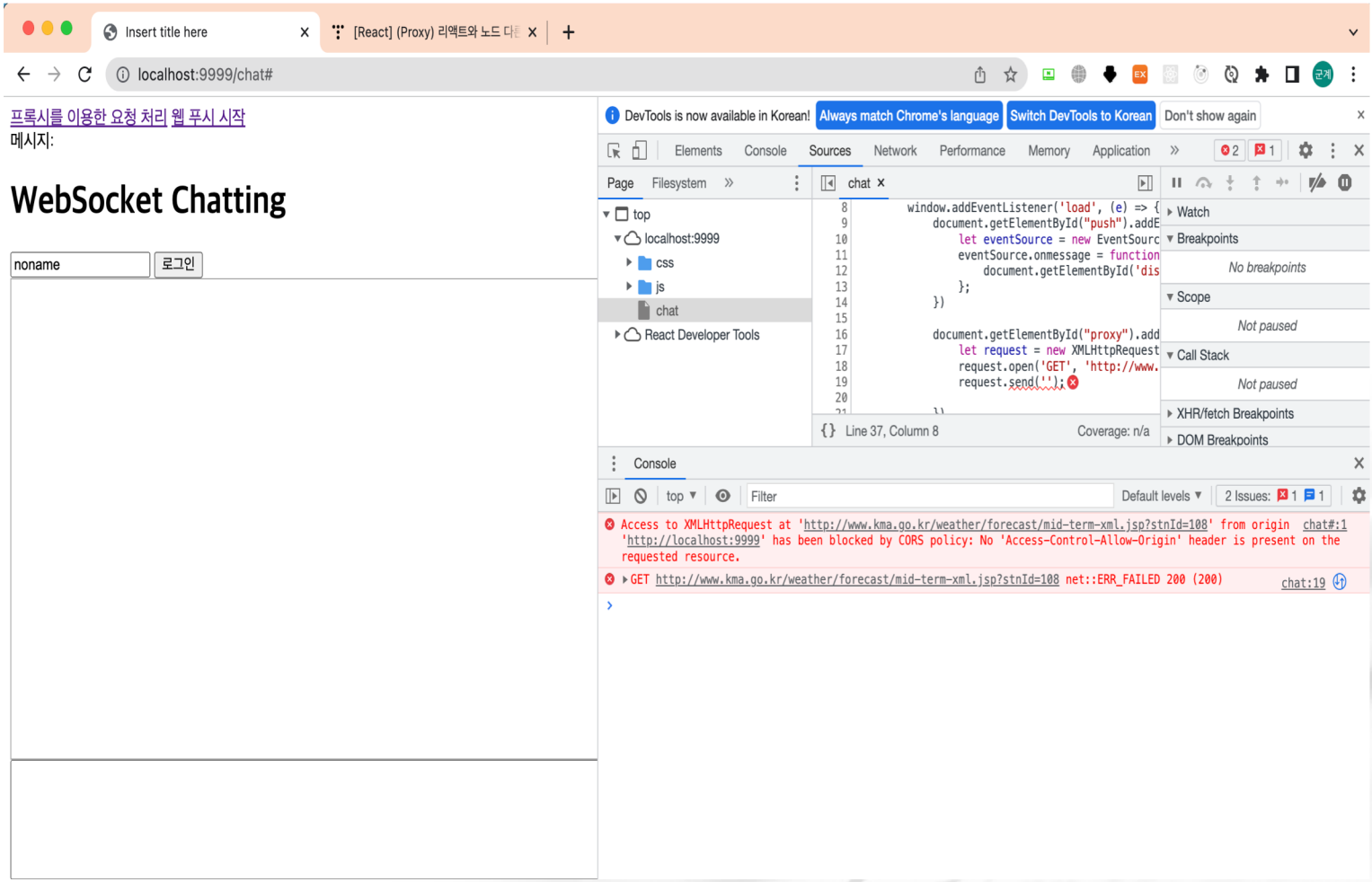
Proxy

- http://www.kma.go.kr/weather/forecast/mid-term-xml.jsp?stnId=109



Proxy

❖ Proxy 구현



Proxy

- Proxy 구현
  - ✓ chat.jsp 파일에 proxy 요청을 생성

```
<a href="#" id="proxy">프록시를 이용한 요청 처리</a>
```

Proxy

- Proxy 구현
  - ✓ chat.jsp 파일에 스크립트 코드 추가

```
document.getElementById("proxy").addEventListener("click", function(e){
        let request = new XMLHttpRequest();
        request.open('GET', 'http://www.kma.go.kr/weather/forecast/mid-term-
xml.jsp?stnId=108');
        request.send('');
})
```

Proxy

❖ Proxy 구현

메일보내기 proxy 가져오기

서울

2017-08-14:흐리고 비
2017-08-15:구름많고 비
2017-08-16:구름많음
2017-08-17:구름많고 비
2017-08-18:구름많음
2017-08-19:구름많음

인천

2017-08-14:흐리고 비
2017-08-15:구름많고 비
2017-08-16:구름많음
2017-08-17:구름많고 비
2017-08-18:구름많음
2017-08-19:구름많음

Proxy Server

- Proxy 구현
  - ✓ chat.jsp 파일에 스크립트 코드 수정

```
document.getElementById("proxy").addEventListener("click", function(e){
        let request = new XMLHttpRequest();
        request.open('GET',
'proxy?addr=http://www.kma.go.kr/weather/forecast/mid-term-xml.jsp?stnId=108');
        request.send('');
        request.onreadystatechange = function() {
            if (request.readyState == 4) {
                if (request.status >= 200 && request.status < 300) {
                    let parser = new DOMParser();
                    let resultXML = parser.parseFromString(request.responseText,
"text/html"); // String -> xml 만들어주는 과정
                    let cities = resultXML.getElementsByTagName('city');
                    let datas = resultXML.getElementsByTagName('data');
                    for (let i = 0; i < datas.length; i++) {
                        let data = datas[i];
                        let output = '';
                        if (i % 6 == 0) {
                            output += '<div>';
                            output += '<h2>' + cities[i/6].childNodes[0].nodeValue +
'</h2>';
                        }
```

Proxy Server

- Proxy 구현
  - ✓ chat.jsp 파일에 스크립트 코드 수정

```
                output += '<span>' +
data.getElementsByTagName('tmEf')[0].childNodes[0].nodeValue + '</span>';
                output += ':<span>' +
data.getElementsByTagName('wf')[0].childNodes[0].nodeValue + '</span><br/>';
                if (i % 6 == 0) {
                    output += '</div>';
                }
                // 출력
                document.getElementById('disp').innerHTML += output
            }
        }
        else if(request.status >= 400 && request.status < 500){
            alert(request.status);
        }
    }
}
})
```

Proxy Server

- Proxy 구현
  - ✓ ProxyService 클래스를 생성하고 데이터를 읽어오는 메서드를 작성

```
@Service
public class ProxyService {
    public String download(HttpServletRequest request, HttpServletResponse response) {
        StringBuilder sb = new StringBuilder();
        try {
            URL url = new URL(request.getParameter("addr"));
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            if (conn != null) {
                conn.setConnectTimeout(20000);
                conn.setUseCaches(false);
```

Proxy Server

- Proxy 구현
  - ✓ ProxyService 클래스를 생성하고 데이터를 읽어오는 메서드를 작성

```
                if (conn.getResponseCode() == HttpURLConnection.HTTP_OK) {
                    InputStreamReader isr = new InputStreamReader(conn.getInputStream());
                    BufferedReader br = new BufferedReader(isr);
                    while (true) {
                        String line = br.readLine();
                        if (line == null) {
                            break;
                        }
                        sb.append(line + "₩n");
                    }
                    br.close();
                    conn.disconnect();
                }
            }
        } catch (Exception e) {
            System.out.println("가져오기 실패:" + e.getMessage());
        }
        return sb.toString();
    }
}
```

Proxy Server

- Proxy 구현
  - ✓ ProxyController 클래스를 생성하고 클라이언트의 요청을 처리하는 메서드를 작성

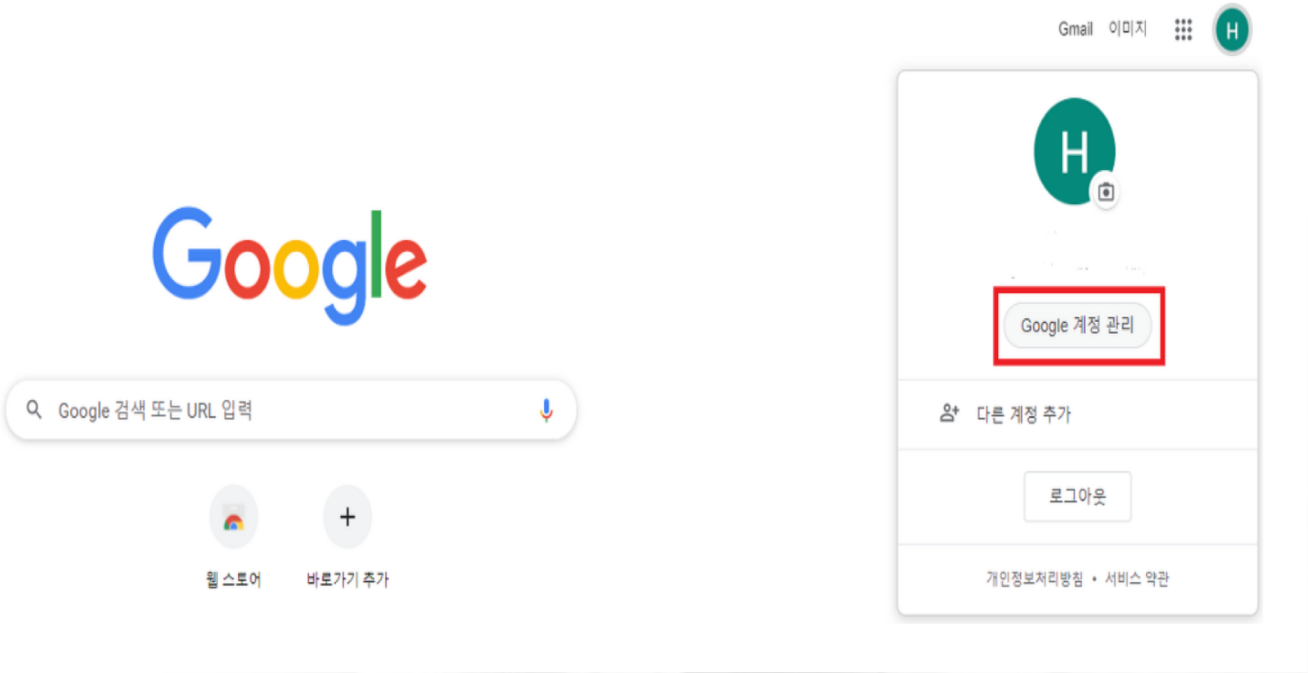
```
@RestController
@RequiredArgsConstructor
public class ProxyController {
    private final ProxyService proxyService;

    @RequestMapping("proxy")
    public String proxy(HttpServletRequest request, HttpServletResponse response) {
        String result = proxyService.download(request, response);
        System.out.println(result);
        return result;
    }
}
```

메일 전송

❖ 메일 전송

✓ gmail을 이용한 메일 서버 계정 설정

1. Google 홈페이지 → Google 계정 관리(우측상단)

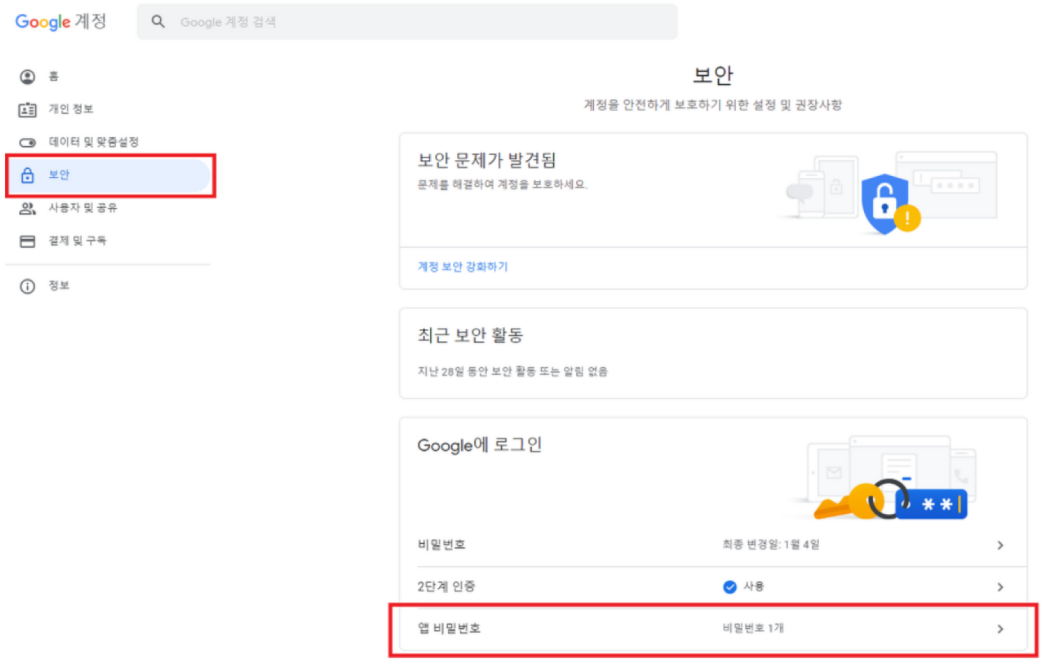


메일 전송

❖ 메일 전송

✓ gmail을 이용한 메일 서버 계정 설정

> 2. 보안 → 앱 비밀번호
>
> 앱 비밀번호는 위에 2단계 인증을 해야 생성됩니다.

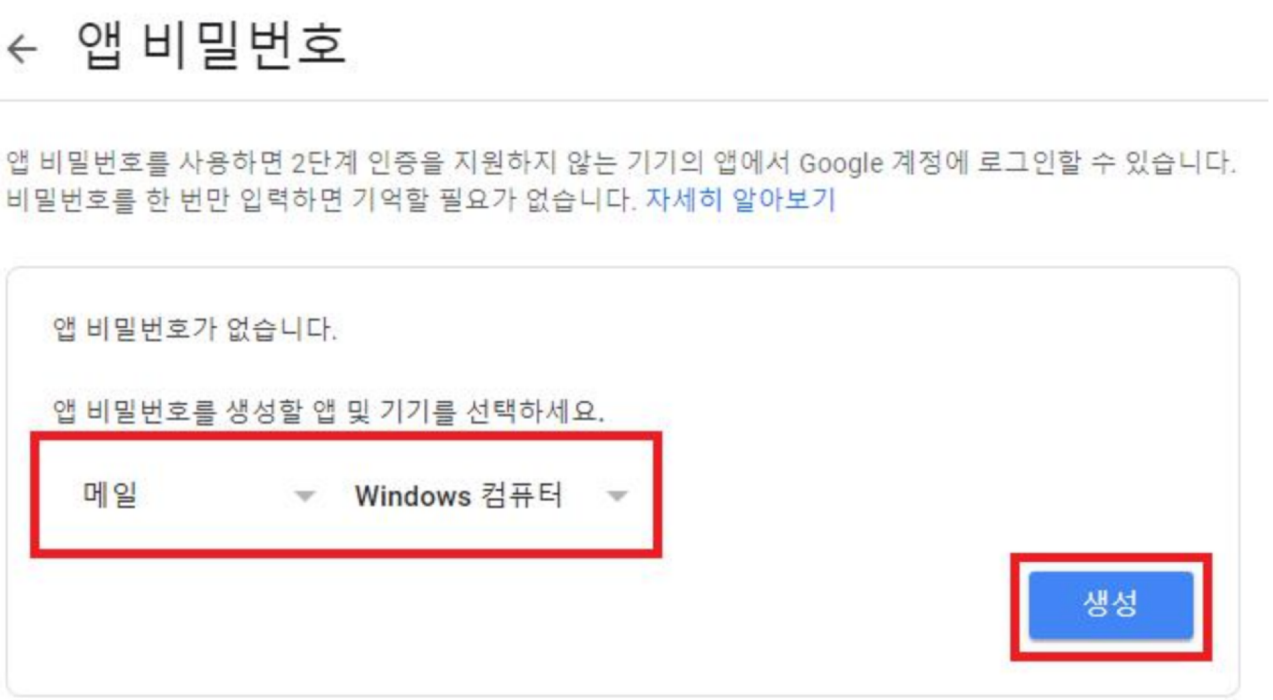


메일 전송

❖ 메일 전송

✓ gmail을 이용한 메일 서버 계정 설정

3. 메일 → Windows 컴퓨터 선택

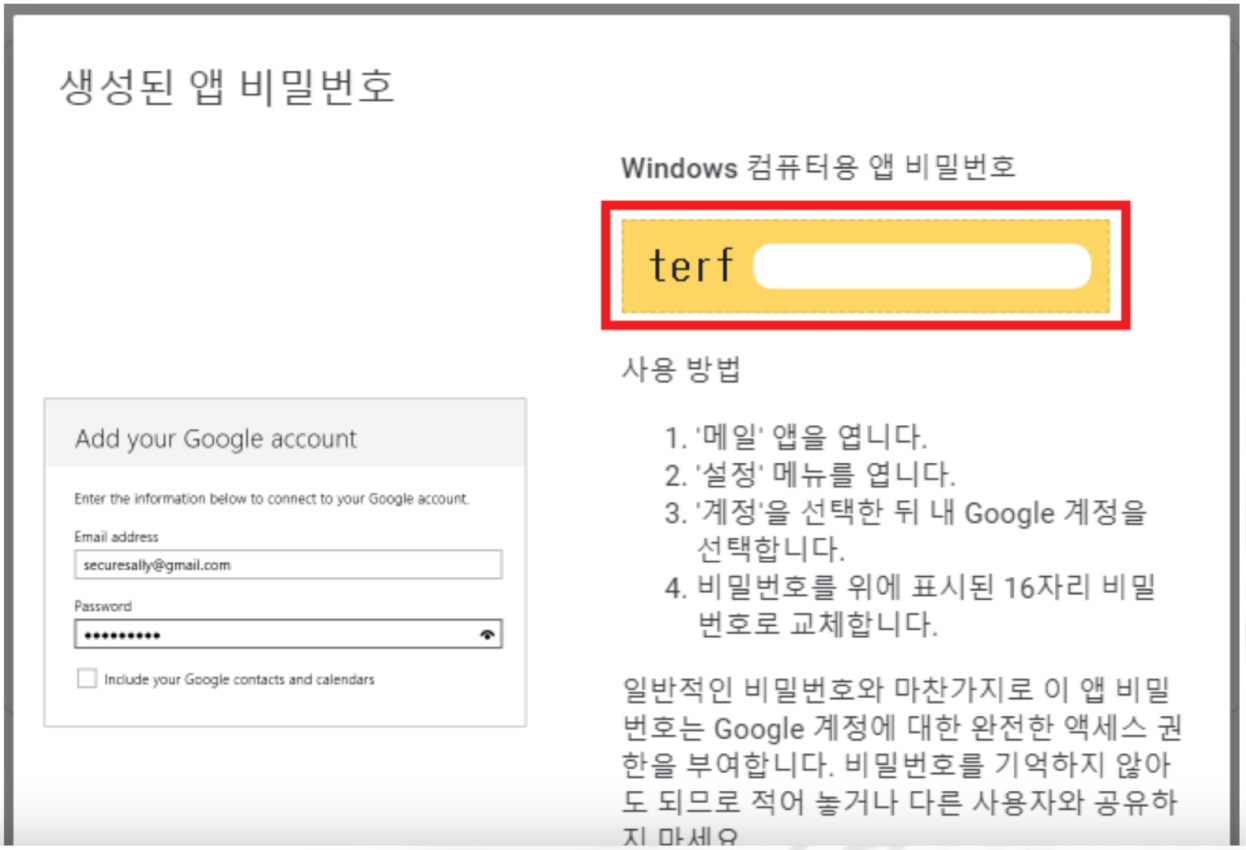


메일 전송

❖ 메일 전송

✓ gmail을 이용한 메일 서버 계정 설정

4. 생성된 앱 비밀번호를 메모해둔다. (smtp 설정에 사용될 예정)

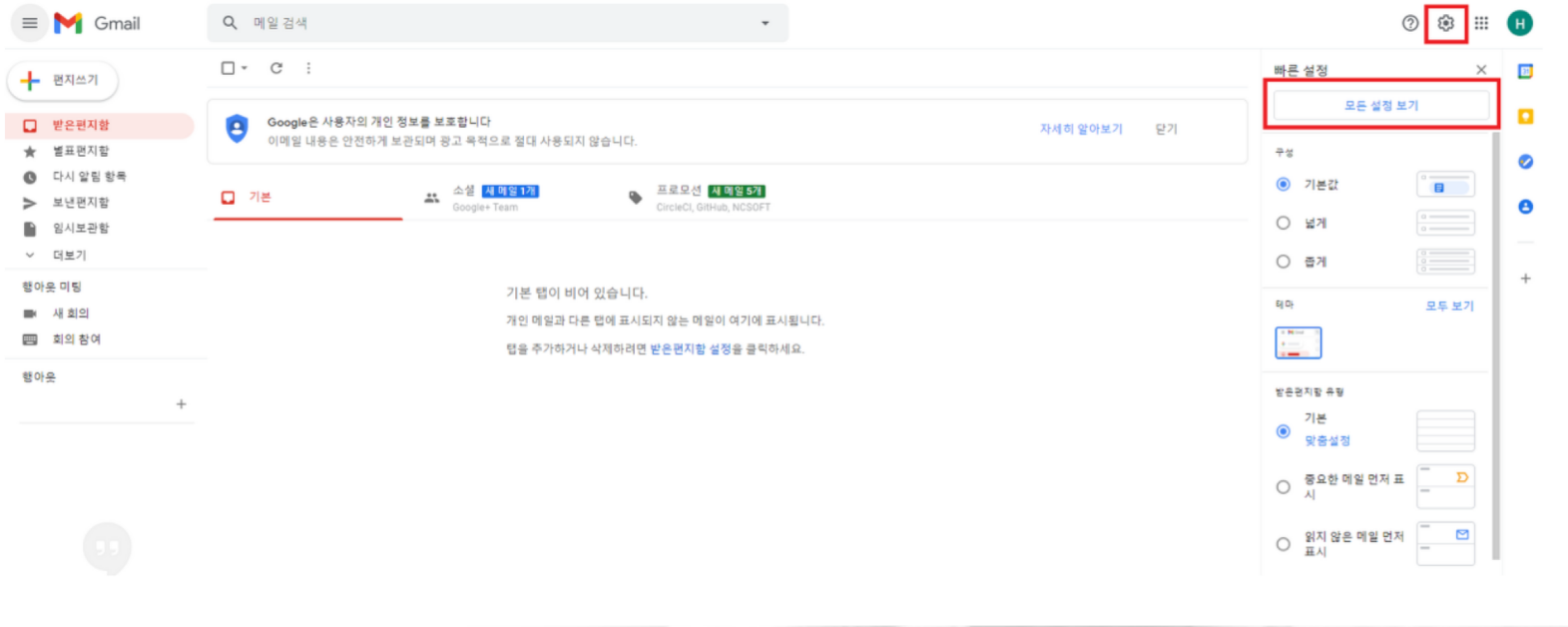


메일 전송

❖ 메일 전송

✓ gmail을 이용한 메일 서버 계정 설정

5. 빠른 설정(우측상단 톱니바퀴) → 모든 설정 보기



메일 전송

❖ 메일 전송

✓ gmail을 이용한 메일 서버 계정 설정

> 6. 전달 및 POP/IMAP(탭) → 모든 메일에 POP 사용하기 → IMAP 사용 → 변경사항 저장

메일 전송

- 메일 전송
  - ✓ 프로젝트에 의존성 라이브러리 추가

    implementation 'org.springframework.boot:spring-boot-starter-mail'

메일 전송

- 메일 전송
  - ✓ application.yml 파일에 메일 서버 설정 코드 추가

```
spring:
  mail:
    host: smtp.gmail.com
    port: 587
    username: 메일계정@gmail.com
    password: 앱비밀번호
    properties:
      mail.smtp.auth: true
      mail.smtp.starttls.enable: true
```

메일 전송

- 메일 전송
  - ✓ Spring에서는 MailSender 인터페이스와 JavaMailSender 클래스를 이용해서 메일을 보낼 수 있음
    - MailSender 인터페이스의 메서드
      - send(SimpleMailMessage simpleMessage)
      - send(SimpleMailMessage[] simpleMessage)

메일 전송

- 메일 전송
  - ✓ Spring에서는 MailSender 인터페이스와 JavaMailSender 클래스를 이용해서 메일을 보낼 수 있음
    - SimpleMailMessage는 메일 구성에 사용되는 클래스
      - setFrom(String): 발신자 설정
      - setReplyTo(String): 응답 주소 설정
      - setTo(String): 수신자 설정
      - setTo(String []): 수신자 목록 설정
      - setCc(String): 참조자 설정
      - setCc(String []): 참조자 목록 설정
      - setBcc(String): 숨은 참조자 설정
      - setBcc(String []): 숨은 참조자 목록 설정
      - setSentDate(Date): 메일 발송일 설정
      - setSubjectString): 메일 제목 설정
      - setText(String): 메일 내용 설정

메일 전송

- 텍스트 메일 전송
  - ✓ 메일 전송 링크 생성 – index.html 파일 생성

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>메일 전송</title>
</head>
<body>
  <a href="textmail">메일 보내기</a>
</body>
</html>
```

메일 전송

- 텍스트 메일 전송
  - ✓ ChatController 클래스에 시작 요청 과 textmail 요청을 처리하는 메서드를 생성

```
@RequestMapping("/")
public String index(){
        return "index";
}

@GetMapping("textmail")
public void textmail(){ }
```

메일 전송

- 텍스트 메일 전송
  - ✓ textmail.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>메일 보내기</title>
</head>
<body>
<div class="container">
  <h4>메일 보내기</h4>
  <form method="post">
    <div align="center">
      <input type="text" name="tomail" size="120" style="width:100%"
     placeholder="상대의 이메일"  >
     </div>
```

메일 전송

❖ 텍스트 메일 전송

✓ textmail.html 파일을 생성하고 작성

```
<div align="center"><!-- 제목 -->
      <input type="text" name="title" size="120" style="width:100%"
    placeholder="제목을 입력해주세요"  >
    </div>
    <p>
    <div align="center"><!-- 내용 -->
      <textarea name="content" cols="120" rows="12" style="width:100%; resize:none"
    placeholder="내용#"></textarea>
    </div>
    <p>
    <div align="center">
      <input type="submit" value="메일 보내기" class="btn btn-warning">
    </div>
  </form>
</div>
</body>
</html>
```

메일 전송

- 텍스트 메일 전송
  - ✓ MailService 클래스를 생성하고 작성

```
@Service
@RequiredArgsConstructor
public class MailService {
    private final JavaMailSender javaMailSender;
```

메일 전송

- 텍스트 메일 전송
  - ✓ MailService 클래스를 생성하고 작성

```
public void sendMail(HttpServletRequest request) {
    String setfrom = "ggangpae1@gmail.com";
    String tomail  = request.getParameter("tomail");     // 받는 사람 이메일
    String title   = request.getParameter("title");      // 보내는 사람 이메일
    String content = request.getParameter("content");    // 보내는 사람 이메일

    try {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom(setfrom);  // 보내는사람 생략하거나 하면 정상작동을 안함
        message.setTo(tomail);     // 받는사람 이메일
        message.setSubject(title); // 메일제목은 생략이 가능하다
        message.setText(content);  // 메일 내용
        javaMailSender.send(message);
    } catch(Exception e){
        System.out.println(e);
    }
  }
}
```

메일 전송

- 텍스트 메일 전송
  - ✓ ChatController 클래스에 메일을 보내는 요청을 처리하기 위한 코드를 작성

```
private final MailService mailService;

@PostMapping("textmail")
public String textmail(HttpServletRequest request){
        mailService.sendMail(request);
        return "redirect:/";
}
```

메일 전송

- HTML 메일 전송
  - ✓ HTML 메일을 보내기 위해서는 MimeMessage 클래스를 이용
  - ✓ JavaMailSender 클래스는 createMimeMessage()메서드를 제공해서 MimeMessage 객체를 리턴하는데 이 객체를 이용해서 메시지를 생성

메일 전송

❖ 첨부 파일 전송

✓ build.gradle에 의존성 추가

```
implementation group: 'commons-io', name: 'commons-io', version: '2.11.0'
```

메일 전송

- 첨부 파일 전송
  - ✓ index.html 파일에 추가

```
<br/>><a href="filemail">첨부파일 메일 보내기</a>
```

메일 전송

- 첨부 파일 전송
  - ✓ filemail.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<head>
  <meta charset="UTF-8">
  <title>Send Mail</title>
</head>
```

메일 전송

❖ 첨부 파일 전송

✓ filemail.html 파일을 생성하고 작성

```
<body>
<div class="container">
  <div style="float: left; width: 50%;">
    <h1>첨부파일 메일 보내기</h1>
    <form method="post" enctype="multipart/form-data">
      <table>
        <tr id="box" class="form-group">
          <td>받는 사람</td>
          <td>
            <input type="text" class="form-control" name="address" placeholder="이메일
주소를 입력하세요">
          </td>
          <td>
            <input type="button" class="form-control" value="추가"
onclick="add_textbox(this)">
          </td>
        </tr>
```

메일 전송

- 첨부 파일 전송
  - ✓ filemail.html 파일을 생성하고 작성

```
<tr id="box2" class="form-group">
        <td>참조 메일 주소</td>
        <td>
          <input type="text" class="form-control" name="ccAddress" placeholder="참조
수신인을 입력하세요">
        </td>
        <td>
          <input type="button" class="form-control" value="추가"
onclick="add_textbox2(this)">
        </td>
      </tr>
      <tr class="form-group">
        <td>제목</td>
        <td>
          <input type="text" class="form-control" name="title" placeholder="제목을
입력하세요">
        </td>
      </tr>
```

메일 전송

❖ 첨부 파일 전송

✓ filemail.html 파일을 생성하고 작성

```
<tr class="form-group">
        <td>내용</td>
        <td>
          <textarea class="form-control" name="content" placeholder="보낼 내용을
입력하세요"> </textarea>
        </td>
      </tr>
      <tr class="form-group">
        <td>첨부 파일 </td>
        <td>
          <input type="file" name="file" class="file-input" />
        </td>
      </tr>
    </table>
    <button class="btn btn-default">발송</button>
  </form>
 </div>
</div>
</body>
```

메일 전송

❖ 첨부 파일 전송

✓ filemail.html 파일을 생성하고 작성

```
<script>
  const add_textbox = (obj) => {
    const box = obj.parentElement.parentElement;
    const newP = document.createElement("tr");

    newP.innerHTML = "<tr class='form-group'><td>메일 주소</td><td><input
     type='text' class='form-control' name='address' ></td><td><input type='button'
     class='form-control' value='삭제' onclick='opt_remove(this)'></td></tr>";
    box.parentNode.insertBefore(newP, box.nextSibling);
  }

  const add_textbox2 = (obj) => {
    const box = obj.parentElement.parentElement;
    const newP = document.createElement("tr");

    newP.innerHTML = "<tr class='form-group'><td>참조 메일 주소</td><td><input
     type='text' class='form-control' name='ccAddress' ></td><td><input
     type='button' class='form-control' value='삭제'
     onclick='opt_remove2(this)'></td></tr>";
    box.parentNode.insertBefore(newP, box.nextSibling);
  }
```

메일 전송

- 첨부 파일 전송
  - ✓ ChatController 클래스에 get 요청을 위한 처리하기 위한 코드를 추가

```
@GetMapping("filemail")
public void filemail(){}
```

메일 전송

- 첨부 파일 전송
  - ✓ 첨부 파일을 위한 DTO 클래스 생성

```
@Getter
@AllArgsConstructor
@NoArgsConstructor
public class AttachFileDto{

    private String realFileNm;
    private String attachFileNm;

}
```

메일 전송

- 첨부 파일 전송
  - ✓ 요청에 대한 파라미터를 위한 DTO 클래스 생성

```
@Getter
@Setter
@NoArgsConstructor
public class MailDto {

    private String[] address;
    private String[] ccAddress;
    private String title;
    private String content;
    private List<AttachFileDto> attachFileList = new ArrayList<>();
}
```

메일 전송

❖ 첨부 파일 전송

✓ MailService 클래스에 첨부 파일을 전송하기 위한 서비스 메서드 작성

```
public void sendMultipleMessage(MailDto mailDto, MultipartFile file) throws
    MessagingException, IOException {
        MimeMessage message = javaMailSender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(message, true);
        //메일 제목 설정
        helper.setSubject(mailDto.getTitle());
        //참조자 설정
        helper.setCc(mailDto.getCcAddress());
        helper.setText(mailDto.getContent(), false);
        helper.setFrom("ggangpae1@gmail.com");
        //첨부 파일 설정
        String fileName = StringUtils.cleanPath(file.getOriginalFilename());
        helper.addAttachment(MimeUtility.encodeText(fileName, "UTF-8", "B"), new
    ByteArrayResource(IOUtils.toByteArray(file.getInputStream())));

        //수신자 한번에 전송
        helper.setTo(mailDto.getAddress());
        javaMailSender.send(message);
    }
```

메일 전송

❖ 첨부 파일 전송

✓ ChatController 클래스에 첨부 파일을 전송하기 요청 처리 메서드 작성

```
@PostMapping("filemail")
public String sendMail(MailDto mailDto, MultipartFile file) throws MessagingException,
    IOException {
        mailService.sendMultipleMessage(mailDto, file);
        System.out.println("메일 전송 완료");
        return "redirect:/";
}
```