KB금융그룹 | 국민의 평생 금융파트너

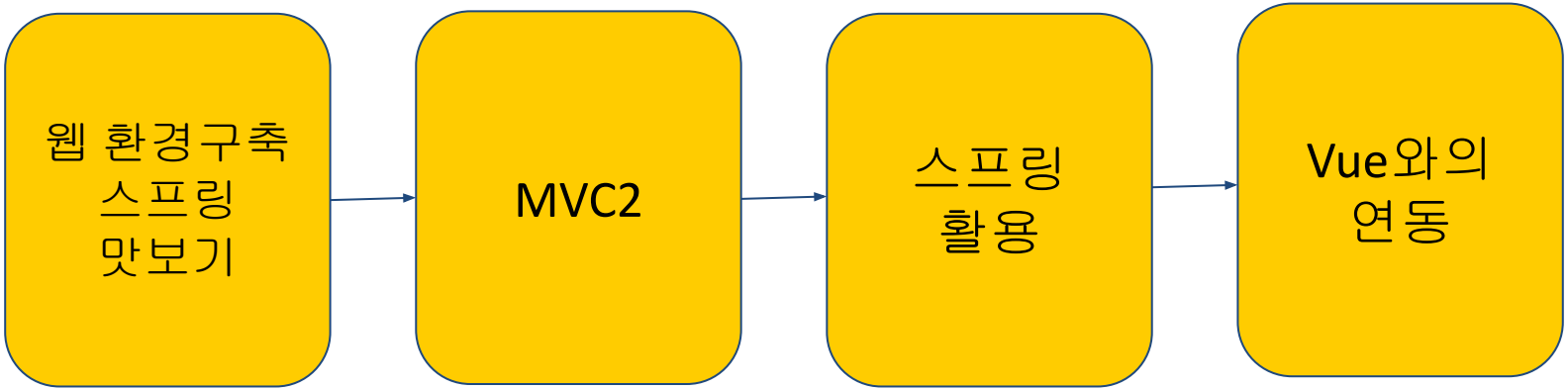
2024년 상반기 K-디지털 트레이닝

SPRING06 - WebSocket

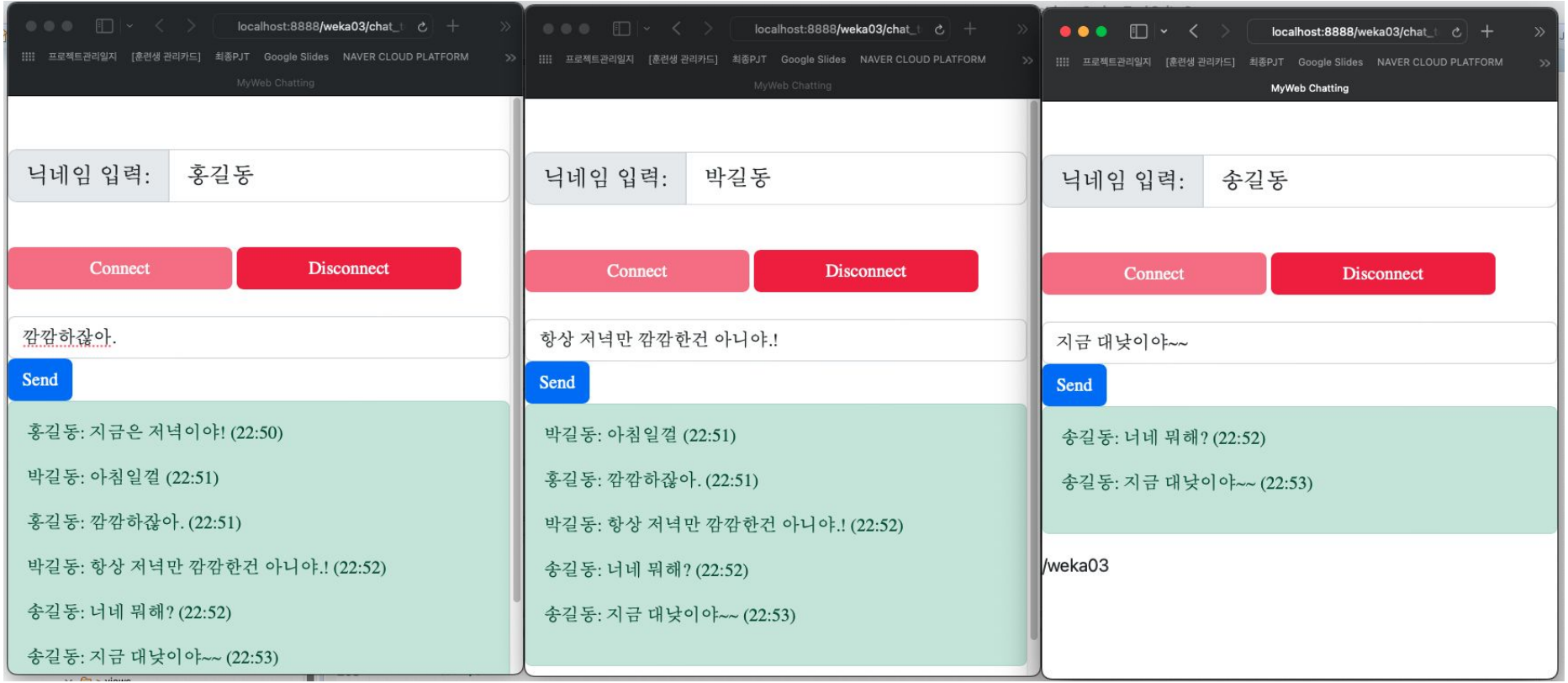
[KB] IT's Your Life

KB 국민은행

커리큘럼

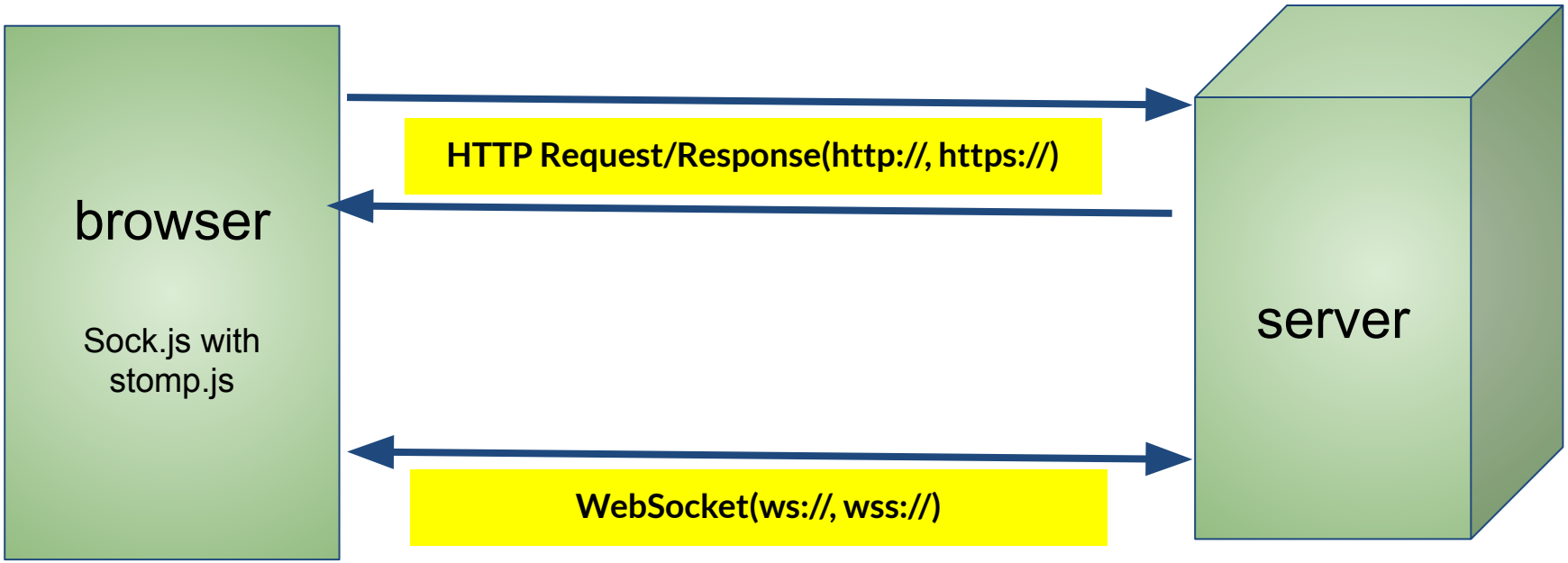


오늘의 목표

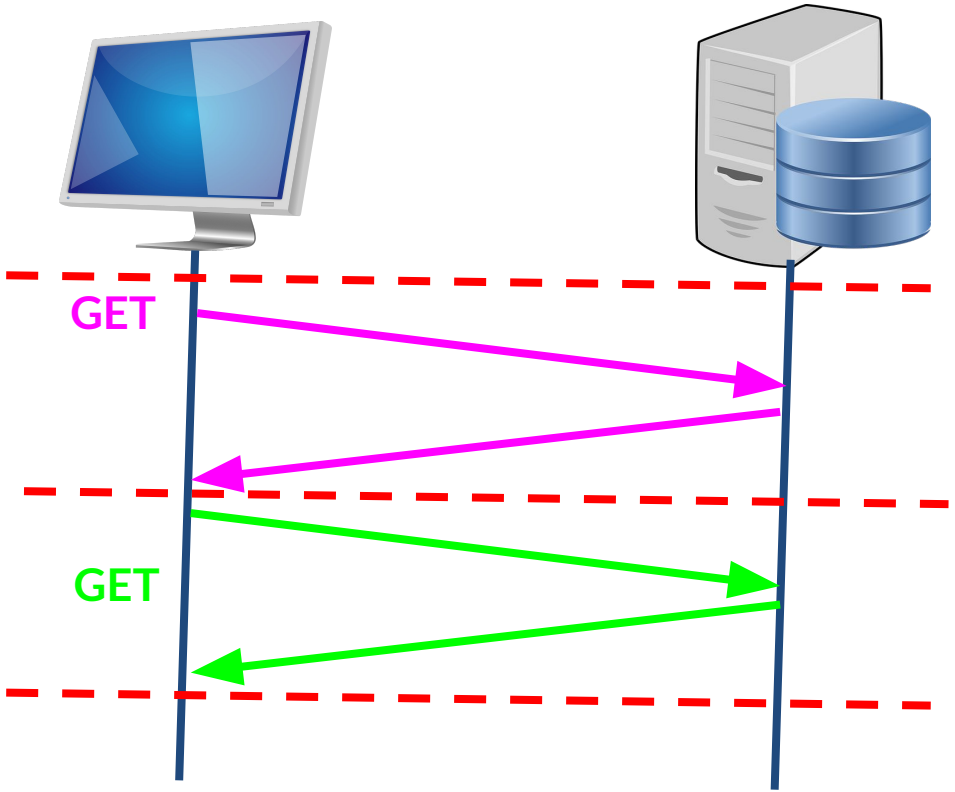

localhost:8888/weka03/chat_t
프로젝트관리일지 [훈련생 관리카드] 최종PJT Google Slides NAVER CLOUD PLATFORM
MyWeb Chatting
닉네임 입력: 홍길동
Connect
Disconnect
깜깜하잖아.
Send
홍길동: 지금은 저녁이야! (22:50)
박길동: 아침일껄 (22:51)
홍길동: 깜깜하잖아. (22:51)
박길동: 항상 저녁만 깜깜한건 아니야.! (22:52)
송길동: 너네 뭐해? (22:52)
송길동: 지금 대낮이야~~ (22:53)
localhost:8888/weka03/chat_t
프로젝트관리일지 [훈련생 관리카드] 최종PJT Google Slides NAVER CLOUD PLATFORM
MyWeb Chatting
닉네임 입력: 박길동
Connect
Disconnect
항상 저녁만 깜깜한건 아니야.!
Send
박길동: 아침일껄 (22:51)
홍길동: 깜깜하잖아. (22:51)
박길동: 항상 저녁만 깜깜한건 아니야.! (22:52)
송길동: 너네 뭐해? (22:52)
송길동: 지금 대낮이야~~ (22:53)
localhost:8888/weka03/chat_t
프로젝트관리일지 [훈련생 관리카드] 최종PJT Google Slides NAVER CLOUD PLATFORM
MyWeb Chatting
닉네임 입력: 송길동
Connect
Disconnect
지금 대낮이야~~
Send
송길동: 너네 뭐해? (22:52)
송길동: 지금 대낮이야~~ (22:53)
/weka03

WebSocket

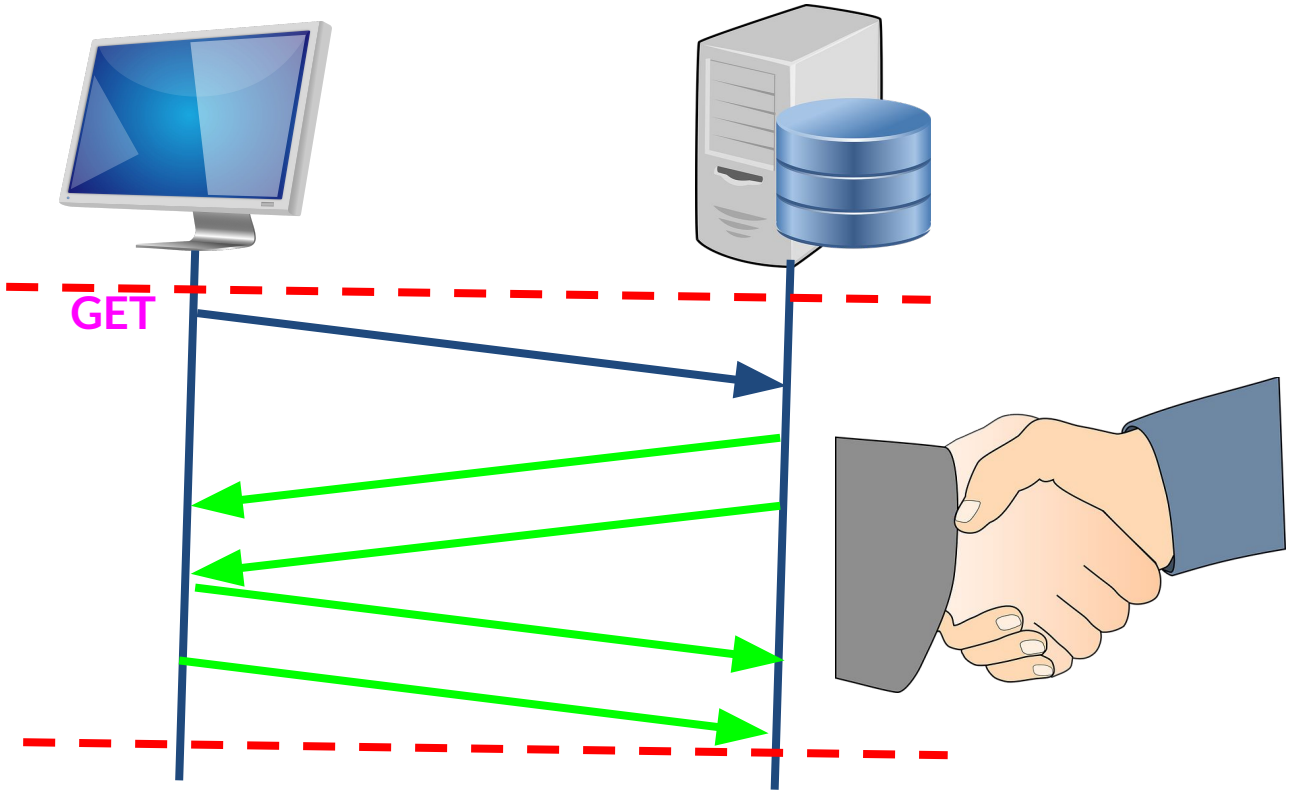
http vs. websocket



Http polling vs http long polling



Http polling vs http long polling



스프링 WebSocket

- **Spring WebSocket**은 스프링 프레임워크에서 웹 소켓(WebSocket) 프로토콜을 지원하는 모듈
- 웹 소켓은 TCP 기반의 프로토콜로, 클라이언트와 서버 간의 양방향 통신을 지원
- **주요 특징:**
  - **양방향 통신**: 클라이언트와 서버가 서로 메시지를 주고받을 수 있음.
    - 실시간 애플리케이션에서 매우 유용
  - **STOMP 지원**: Spring WebSocket 모듈은 STOMP 프로토콜을 통해 메시지 브로커와 통합할 수 있는 기능을 제공
  - **SockJS 지원**: WebSocket을 지원하지 않는 브라우저나 네트워크 환경에서 대체할 수 있는 폴백 메커니즘으로 SockJS를 제공
  - **STOMP를 통한 메시지 라우팅**: Spring WebSocket은 STOMP를 사용하여 특정 주제(Topic)나 큐(Queue)로 메시지를 라우팅

스프링 WebSocket

- **Spring WebSocket 사용 시나리오:**
  - **실시간 웹 애플리케이션**: 채팅, 알림, 실시간 데이터 스트리밍 등 실시간 통신이 필요한 웹 애플리케이션에 적합
  - **멀티플렉싱**: STOMP를 사용하여 단일 WebSocket 연결에서 여러 주제를 구독하고 메시지를 수신
  - **브로커 기반 메시징**: 메시지 브로커(예: RabbitMQ, ActiveMQ)와 통합하여 더욱 강력한 메시징 시스템을 구축

STOMP

- **STOMP**(Simple (or Streaming) Text Oriented Messaging Protocol)는 텍스트 기반의 메시징 프로토콜
- 웹 소켓(WebSocket) 프로토콜 위에서 사용
- 클라이언트와 메시지 브로커 간의 통신을 위한 규약을 정의하며, 메시징에 특화된 프로토콜
- 주요 특징
  - **프레임 구조**: STOMP 메시지는 프레임이라는 단위로 전송
    - 주요 프레임은 `CONNECT`, `SEND`, `SUBSCRIBE`, `UNSUBSCRIBE`, `BEGIN`, `COMMIT`, `ABORT`, `ACK`, `NACK`, `DISCONNECT` 등
  - **텍스트 기반**: 모든 메시지가 텍스트 형태로 전송되며, 간단하고 읽기 쉬운 형식
  - **브로커 지원**: STOMP는 메시지 브로커(예: ActiveMQ, RabbitMQ)와의 통신을 지원
    - 클라이언트는 브로커에 메시지를 보내고, 브로커는 이를 적절한 구독자에게 전달

STOMP

- **CONNECT**
  - **설명**: 클라이언트가 서버(브로커)와의 연결을 시작할 때 사용. 이 프레임을 통해 클라이언트는 서버에 연결을 요청하며, 서버는 CONNECTED 프레임을 응답으로 보냄
  - **주요 헤더**:
    - `accept-version`: 클라이언트가 지원하는 STOMP 버전(예: `1.0`, `1.1`, `1.2`).
    - `host`: 클라이언트가 연결하고자 하는 브로커의 호스트 이름.
    - `login`: (선택적) 브로커에 접속하기 위한 사용자 이름.
    - `passcode`: (선택적) 브로커에 접속하기 위한 비밀번호.

```
CONNECT
accept-version:1.2
host:stompbroker.com
login:user
passcode:password

\0
```

STOMP

- **SEND**
  - **설명**: 클라이언트가 브로커를 통해 특정 대기열(queue)나 주제(topic)로 메시지를 보낼 때 사용. 메시지는 해당 목적지로 라우팅되어, 구독자(subscriber)에게 전달
  - **주요 헤더**:
    - `destination`: 메시지가 전송될 목적지(큐 또는 토픽).
    - `content-type`: (선택적) 메시지의 MIME 타입.
    - `content-length`: (선택적) 메시지의 길이.

```
SEND
destination:/queue/chat
content-type:text/plain

Hello, World!
\0
```

STOMP

- **SUBSCRIBE**
  - **설명**: 클라이언트가 특정 주제나 큐를 구독하여, 해당 목적지로 전송된 메시지를 수신하고자 할 때 사용. 이 프레임을 통해 클라이언트는 특정 목적지를 지정하고, 해당 목적지로 전송된 모든 메시지를 수신
  - **주요 헤더**:
    - `destination`: 구독하려는 주제나 큐의 이름.
    - `id`: 구독을 식별하기 위한 고유 ID. 동일한 연결에서 여러 구독을 구분하는 데 사용됩니다.
    - `ack`: (선택적) 메시지 확인 모드(예: `auto`, `client`, `client-individual`).

```
SUBSCRIBE
id:sub-0
destination:/topic/news

\0
```

- **UNSUBSCRIBE**
  - **설명**: 클라이언트가 특정 목적지에 대한 구독을 취소할 때 사용. 더 이상 특정 주제나 큐에서 메시지를 수신하고 싶지 않은 경우, 이 프레임을 사용해 구독을 종료
  - **주요 헤더**:
    - `id`: 구독 시 사용한 고유 ID. 이 ID를 통해 어떤 구독을 취소할 것인지를 지정

```
UNSUBSCRIBE
id:sub-0

\0
```

- **DISCONNECT**
  - **설명**: 클라이언트가 서버와의 연결을 종료할 때 사용. 이 프레임을 전송하면 서버는 클라이언트와의 연결을 끊고, 연결을 정리
  - **주요 헤더**:
    - `receipt`: (선택적) 서버로부터 연결 종료 확인을 받기 위한 ID.

```
DISCONNECT
receipt:77

\0
```

STOMP

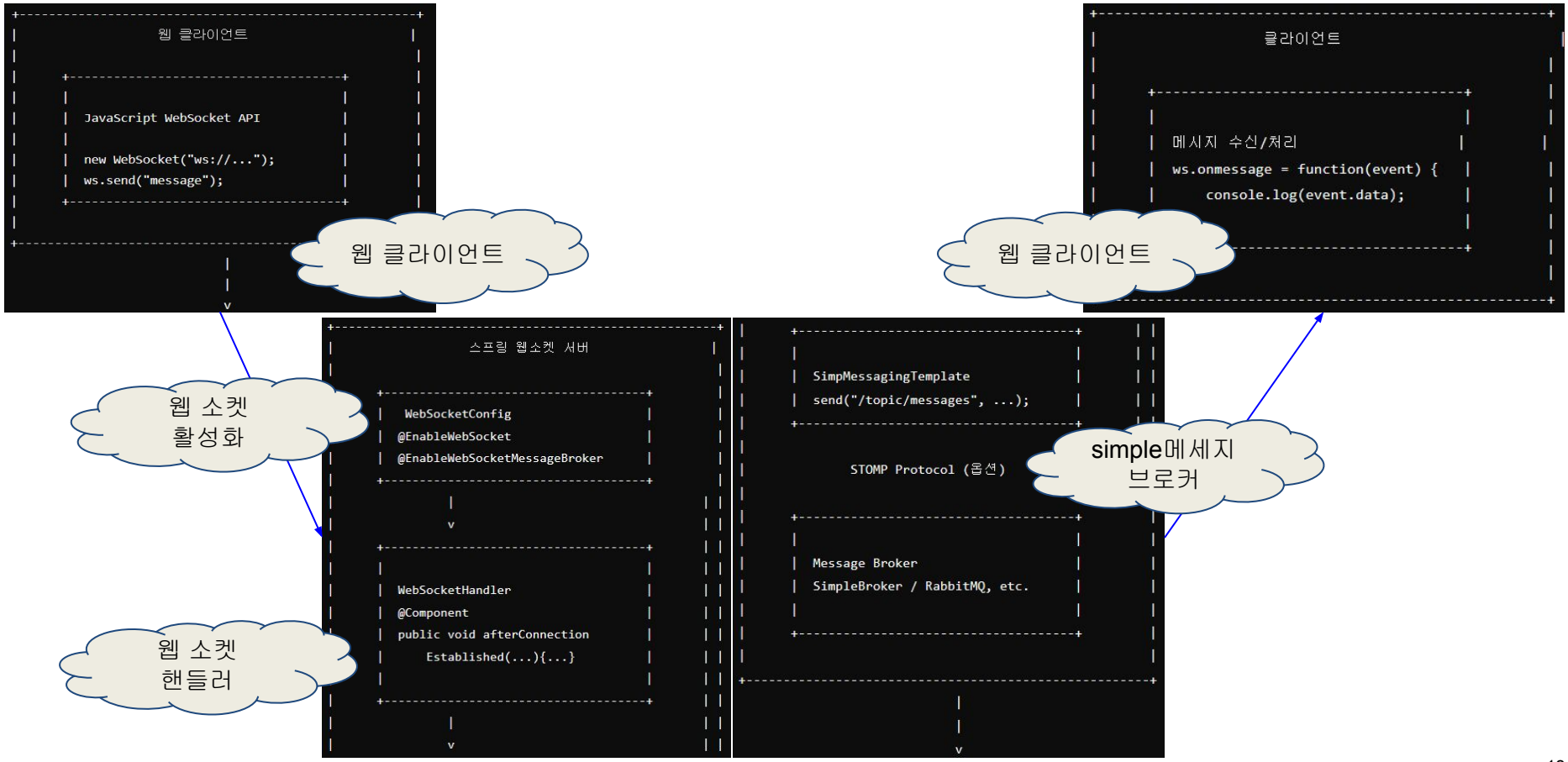
- **STOMP 사용 시나리오:**
  - **메시징 시스템**: STOMP는 채팅 애플리케이션, 실시간 피드, 알림 시스템 등 메시징 시스템에서 많이 사용
  - **브로커와의 통합**: 메시지 브로커를 사용해 복잡한 메시지 라우팅 및 처리 로직을 구현하는 경우 유용

STOMP와 Spring WebSocket의 통합

- 스프링에서 WebSocket과 STOMP를 통합하여 사용하는 경우

1. **WebSocket 설정**
   - Spring에서 WebSocket을 설정하고, `/ws-stomp` 같은 엔드포인트를 통해 클라이언트가 서버에 연결
2. **STOMP 메시지 처리**
   - 클라이언트는 특정 경로(예: `/app/hello`)로 메시지를 보내고, 서버는 이를 처리하여 다른 클라이언트로 브로드캐스트
3. **메시지 브로커 사용**
   - Spring WebSocket은 내부적으로 간단한 메시지 브로커(Simple Broker)를 제공하며, ActiveMQ나 RabbitMQ 같은 외부 브로커와 통합
4. **SockJS 사용**
   - WebSocket을 지원하지 않는 환경에서도 SockJS를 사용하여 WebSocket과 비슷한 경험을 제공

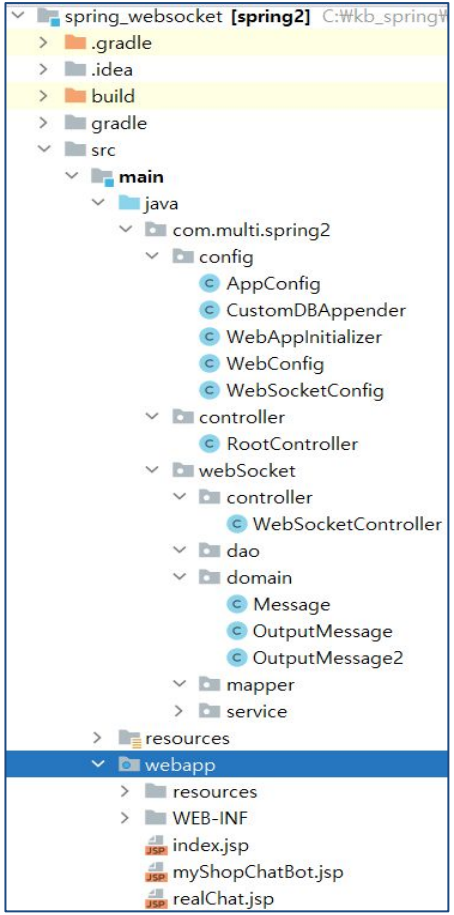
STOMP와 Spring WebSocket의 통합



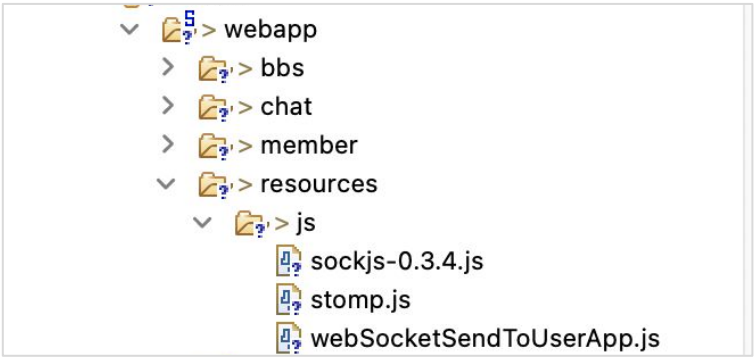
STOMP와 Spring WebSocket의 통합

- **웹 클라이언트**:
  - 클라이언트는 브라우저에서 WebSocket 객체를 생성하고, 서버에 연결을 시도.
  - 연결이 성공하면 클라이언트는 서버로 메시지를 보내거나 서버로부터 메시지를 받을 수 있음.
- **스프링 웹소켓 서버**:
  - 스프링은 `@EnableWebSocket` 또는 `@EnableWebSocketMessageBroker` 어노테이션을 사용하여 웹소켓을 활성화
  - 웹소켓 핸들러(`WebSocketHandler`)가 실제로 메시지를 처리하는 역할을 하며, 연결의 수립 및 메시지 송수신을 관리
- **STOMP 프로토콜(선택 사항)**:
  - STOMP 프로토콜을 사용하면 메시지 브로커를 통해 메시지를 관리.
  - 스프링에서는 기본적으로 SimpleBroker를 사용하거나 외부 메시지 브로커(예: RabbitMQ)를 사용할 수 있음.
- **메시지 브로커**:
  - 브로커는 클라이언트 간의 메시지를 중계하며, 이를 통해 복잡한 메시징 시나리오를 간편하게 구현할 수 있음.
- **클라이언트 측 메시지 처리**:
  - 클라이언트는 서버로부터 수신한 메시지를 처리하고, 이 데이터를 활용하여 필요한 작업을 수행

WebSocket(프로젝트파일 다운로드)




```
// 스프링
implementation 'org.springframework:spring-core:5.3.37'
implementation "org.springframework:spring-context:5.3.37"
implementation "org.springframework:spring-webmvc:5.3.37"
implementation 'javax.inject:javax.inject:1'
implementation 'org.springframework:spring-web:5.3.37'
implementation 'org.springframework:spring-jdbc:5.3.37'
implementation
'com.fasterxml.jackson.core:jackson-databind:2.15.2' //
Jackson 라이브러리 추가
implementation 'com.googlecode.json-simple:json-simple:1.1.1'

//webSocket
implementation 'org.springframework:spring-websocket:5.3.29'
implementation 'org.springframework:spring-messaging:5.3.29'
```




```
<script src="resources/js/sockjs-0.3.4.js"></script>
<script src="resources/js/stomp.js"></script>
<script src="resources/js/webSocketSendToUserApp.js"></script>
```

WebSocket

```
public class WebAppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    public WebAppInitializer(){
        System.out.println("WebAppInitializer created");
    }

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {
                AppConfig.class,
                WebSocketConfig.class
        };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { WebConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

    @Override
    protected Filter[] getServletFilters() {
        CharacterEncodingFilter characterEncodingFilter = new CharacterEncodingFilter();
        characterEncodingFilter.setEncoding("UTF-8");
        characterEncodingFilter.setForceEncoding(true);
        return new Filter[] { characterEncodingFilter };
    }
}
```

WebSocket

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig  extends AbstractWebSocketMessageBrokerConfigurer {

    //채팅방 이름 설정
    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker( "/topic");
        config.setApplicationDestinationPrefixes( "/app");
    }

    //채팅 내용을 보낼 주소(endPoint == url)
    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        //chat: 실시간 채팅용
        registry.addEndpoint( "/chat"); //자바 소켓 통신 가능
        registry.addEndpoint( "/chat").withSockJS();  //자바 스크립트 소켓 통신

        //chat2: 챗봇용
        registry.addEndpoint( "/chat2"); //자바 소켓 통신 가능
        registry.addEndpoint( "/chat2").withSockJS();  //자바 스크립트 소켓 통신
    }

}
```

WebSocket

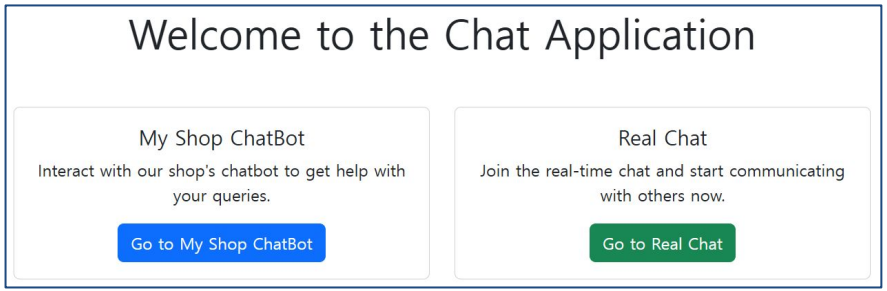
```
@Controller
public class WebSocketController {

    @MessageMapping ("/chat2") //채팅 내용 받을 때 사용하는 주소
    @SendTo ("/topic/messages2")
    public OutputMessage2 send2(Message message) {   //from: guest, text:1
        System.out.println("받은 데이터>>" + message);
        OutputMessage2 out =  new OutputMessage2();
        String menu =  "";
        switch (message.getText()) {
            case "1":
                menu =  "챗 봇>>  10) 운\n동화        11) 모자        12) 가방";
                break;
            case "2":
                menu =  "챗 봇>>  20) 배송조회 21) 주문완료상품조회";
                break;
            case "20":
                menu =  "챗 봇>> DB에서 가지고 온 배송 상황 목록 <br>" +
                        "상품명: 르꼬뿌       배송상황: 물건 준비중";
                break;
            case "10":
                menu =  "챗 봇>>  100) 나이커      200) 르꼬뿌          300) 라코스또";
                break;
            case "100":
                menu =  "챗 봇>>  1000) 나이커 운동화 세부 메뉴  1) 다시 처음 메뉴";
                break;
            case "1000":
                menu =  "챗 봇>>  1001)나이커빨강(1만원)  1002)나이커파랑(2만원)    1003)나이커보라(3만원)    100)이전 메뉴로";
                break;
            default:
                menu =  "챗 봇>>선택한 번호는 없는 메뉴입니다. 다시 입력해주세요.";
                break;
        }
        out.setMenu(menu);
        return out;
    }
```

```
@MessageMapping("/chat") //채팅 내용 받을 때 사용하는 주소
@SendTo("/topic/messages") //가입주소한 브라우저에 return message를 json으로 변환해서 보내줌.
public OutputMessage send(Message message) {
    System.out.println("받은 데이터>>" + message);
    OutputMessage out = new OutputMessage();
    out.setFrom(message.getFrom());
    out.setText(message.getText());
    Date date = new Date();
    out.setTime(date.getHours() + ":" + date.getMinutes());
    return out;
}
```

```java
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Message {
    private String from;
    private String text;
}
```

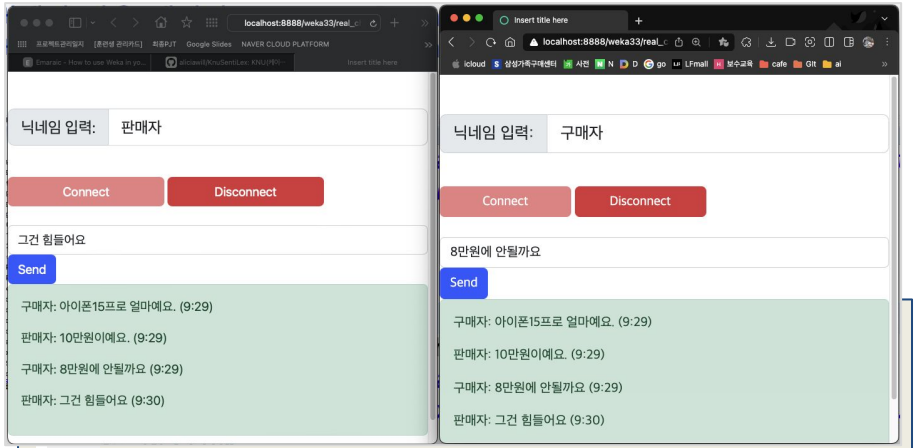
```java
@Data
@AllArgsConstructor
@NoArgsConstructor
public class OutputMessage {
        private String from;
        private String text;
        private String time;

}


//서버에서 받은 데이터를 채팅창에 들어와 있는 멤버
브라우저에게
//보낼 데이터를 넣을 vo ==>messageBroker가 json으로
만들어서 브라우저에 보낸다.
//@sendTo("topic/messages")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class OutputMessage2 {

        private String from;
        private String text;
        private String menu;
}
```

WebSocket - index.jsp




```jsp
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <title>Index Page</title>
 <!-- Bootstrap CSS -->
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container mt-5">
 <h1 class="text-center mb-5">Welcome to the Chat Application</h1>

 <div class="row justify-content-center">
   <div class="col-md-4">
     <div class="card">
       <div class="card-body text-center">
         <h5 class="card-title">My Shop ChatBot</h5>
         <p class="card-text">Interact with our shop's chatbot to get help with your queries.</p>
         <a href="myShopChatBot.jsp" class="btn btn-primary">Go to My Shop ChatBot</a>
       </div>
     </div>
   </div>

   <div class="col-md-4">
     <div class="card">
       <div class="card-body text-center">
         <h5 class="card-title">Real Chat</h5>
         <p class="card-text">Join the real-time chat and start communicating with others now.</p>
         <a href="realChat.jsp" class="btn btn-success">Go to Real Chat</a>
       </div>
     </div>
   </div>
 </div>
</div>

<!-- Bootstrap JS and dependencies -->
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.7/dist/umd/popper.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.min.js"></script>
</body>
</html>
```

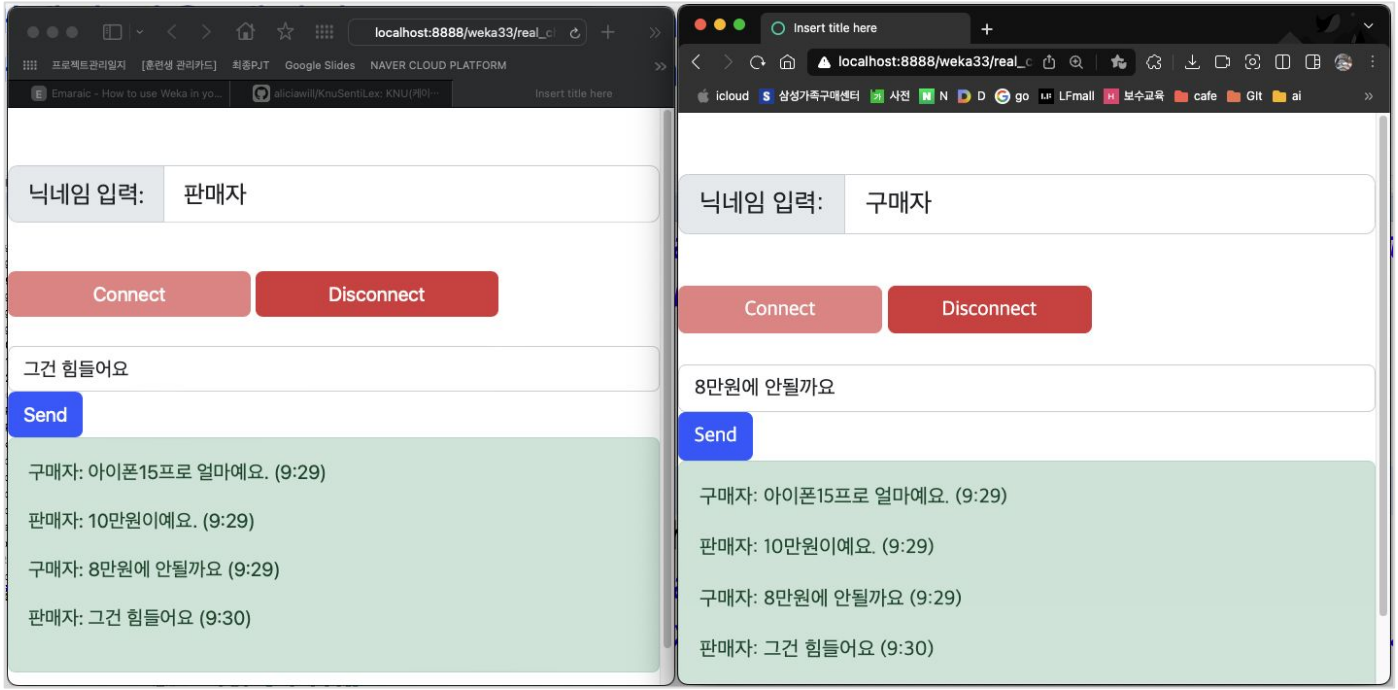
WebSocket-chatting




```
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
        rel="stylesheet">
 <script

src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"
></script>
 <script type="text/javascript" src="resources/js/sockjs-0.3.4.js"></script>
 <script type="text/javascript" src="resources/js/stomp.js"></script>
 <script type="text/javascript"
src="resources/js/webSocketSendToUserApp.js"></script>
 <script type="text/javascript">
   var stompClient = null;

   function setConnected(connected) {  //연결 여부에 따라 설정
     document.getElementById('connect').disabled = connected;
     document.getElementById('disconnect').disabled = !connected;
     document.getElementById('conversationDiv').style.visibility = connected ?
'visible'
         : 'hidden';
     //$('#response').html('')와 동일한 코드
     document.getElementById('response').innerHTML = '';
   }
```

```
function connect() {
  //1. 소켓객체 생성
  var socket = new SockJS('${pageContext.request.contextPath }/chat')
  //2. 데이터를 전송하고, 받을 수 있는 stompClent객체 생성
  stompClient = Stomp.over(socket);

  //3. 채팅방 지정하여 가입하자.
  stompClient.connect({}, function(frame) {
    setConnected(true) //css설정
    alert('연결됨. '+ frame)
    stompClient.subscribe('/topic/messages', function(messageOutput) {
      //서버에서 받은 메시지 출력
      showMessageOutput(JSON.parse(messageOutput.body));
    })
  })
  //4. 서버에서 보냈을 때 어떻게 처리할지 지정
}

//서버로 메세지 보냄
function sendMessage() {
  //입력한 정보를 가지고 와서
  var from = document.getElementById('from').value;
  var text = document.getElementById('text').value;
  //url을 /app/cht을 호출하고,data를 json형태의 sring으로 만들어서 보내라.
  stompClient.send("/app/chat", {}, JSON.stringify({
    'from' : from,
    'text' : text
  }));
}

//받은 데이터를 원하는 위치에 넣음. ==> 받은 데이터를 채팅목록으로 쌓는다.
function showMessageOutput(messageOutput) {
  //<p id="response">
  //  <p> 홍길동: 잘지내지?(13:00)</p>
  //</p>
  var response = document.getElementById('response');
  var p = document.createElement('p'); //p태그를 만들어라.
  p.style.wordWrap = 'break-word';
  p.appendChild(document.createTextNode(messageOutput.from + ": "
          + messageOutput.text + " (" + messageOutput.time + ")"));
  response.appendChild(p);
}
```

WebSocket-chatting



WebSocket-chatting


```
<head>
  <meta charset="UTF-8">
  <title>Insert title here</title>
  <script type="text/javascript"
          src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
  <link
          href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
          rel="stylesheet">
  <script
          src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"></script>
  <script type="text/javascript" src="resources/js/sockjs-0.3.4.js"></script>
  <script type="text/javascript" src="resources/js/stomp.js"></script>
  <script type="text/javascript" src="resources/js/webSocketSendToUserApp.js"></script>
  <script type="text/javascript">
    var stompClient = null;

    function setConnected(connected) {  //연결 여부에 따라 설정
      document.getElementById('connect').disabled = connected;
      document.getElementById('disconnect').disabled = !connected;
      document.getElementById('conversationDiv').style.visibility = connected ?  'visible'
              : 'hidden';
      //$('#response').html('')와 동일한 코드
      document.getElementById('response').innerHTML = '';
    }
```

WebSocket-chatting

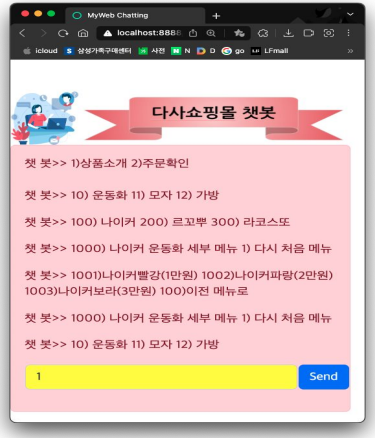
```
function connect() {
  //1. 소켓객체 생성
  var socket = new SockJS('${pageContext.request.contextPath }/chat')
  //2. 데이터를 전송하고, 받을 수 있는 stompClent객체 생성
  stompClient = Stomp.over(socket);

  //3. 채팅방 지정하여 가입하자.
  stompClient.connect({}, function(frame) {
    setConnected(true) //css설정
    alert('연결됨. '+ frame)
    stompClient.subscribe('/topic/messages', function(messageOutput) {
      //서버에서 받은 메시지 출력
      showMessageOutput(JSON.parse(messageOutput.body));
    })
  })
  //4. 서버에서 보냈을 때 어떻게 처리할지 지정
}

//서버로 메세지 보냄
function sendMessage() {
  //입력한 정보를 가지고 와서
  var from = document.getElementById('from').value;
  var text = document.getElementById('text').value;
  //url을 /app/cht을 호출하고,data를 json형태의 sring으로 만들어서 보내라.
  stompClient.send("/app/chat", {}, JSON.stringify({
    'from' : from,
    'text' : text
  }));
}
```

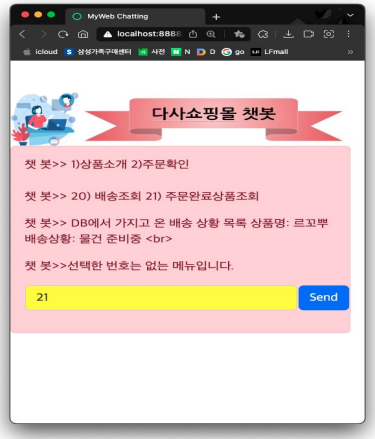
WebSocket-chatting

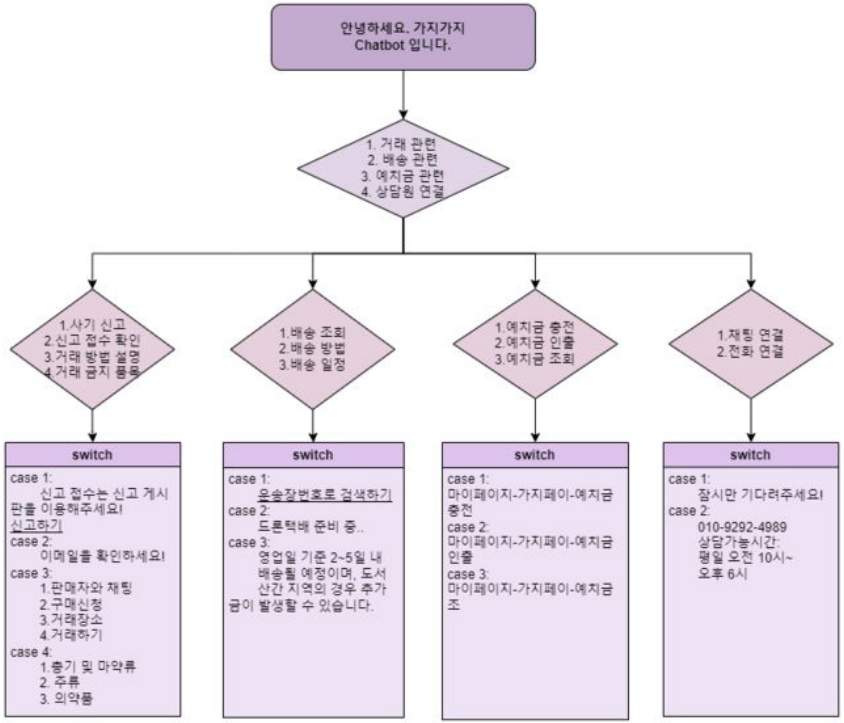
```
//받은 데이터를 원하는 위치에 넣음. ==> 받은 데이터를 채팅목록으로 쌓는다.
function showMessageOutput(messageOutput) {
  //<p id="response">
  //  <p> 홍길동: 잘지내지?(13:00)</p>
  //</p>
  var response = document.getElementById('response');
  var p = document.createElement('p'); //p태그를 만들어라.
  p.style.wordWrap = 'break-word';
  p.appendChild(document.createTextNode(messageOutput.from + ": "
          + messageOutput.text + " (" + messageOutput.time + ")"));
  response.appendChild(p);
}

//서버로 연결 끊음.
  function disconnect() {
    if (stompClient != null) {
      stompClient.disconnect();
    }
    setConnected(false); //연결끊어졌을 때 설정 변경
    console.log("Disconnected");
  }

</script>
```

```
</head>
<body onload="disconnect()">
<!-- $(function({
   disconnect()
})) -->
<br><br>
<div>
 <div class="input-group mb-3 input-group-lg">
   <span class="input-group-text">닉네임 입력:</span>
   <input type="text" class="form-control" id="from">
 </div>
 <br />
 <div>
   <button id="connect" onclick="connect();" class="btn btn-danger" style="width:200px">Connect</button>
   <button id="disconnect" disabled="disabled" onclick="disconnect();" style="width:200px" class="btn
btn-danger">Disconnect</button>
 </div>
 <br />
 <div id="conversationDiv">
   <input type="text" id="text" placeholder="Write a message..." class="form-control" />
   <button id="sendMessage" onclick="sendMessage();"
           class="btn btn-primary">Send</button>

   <p id="response" class="alert alert-success">
 </div>
</div>
${pageContext.request.contextPath }
</body>
```

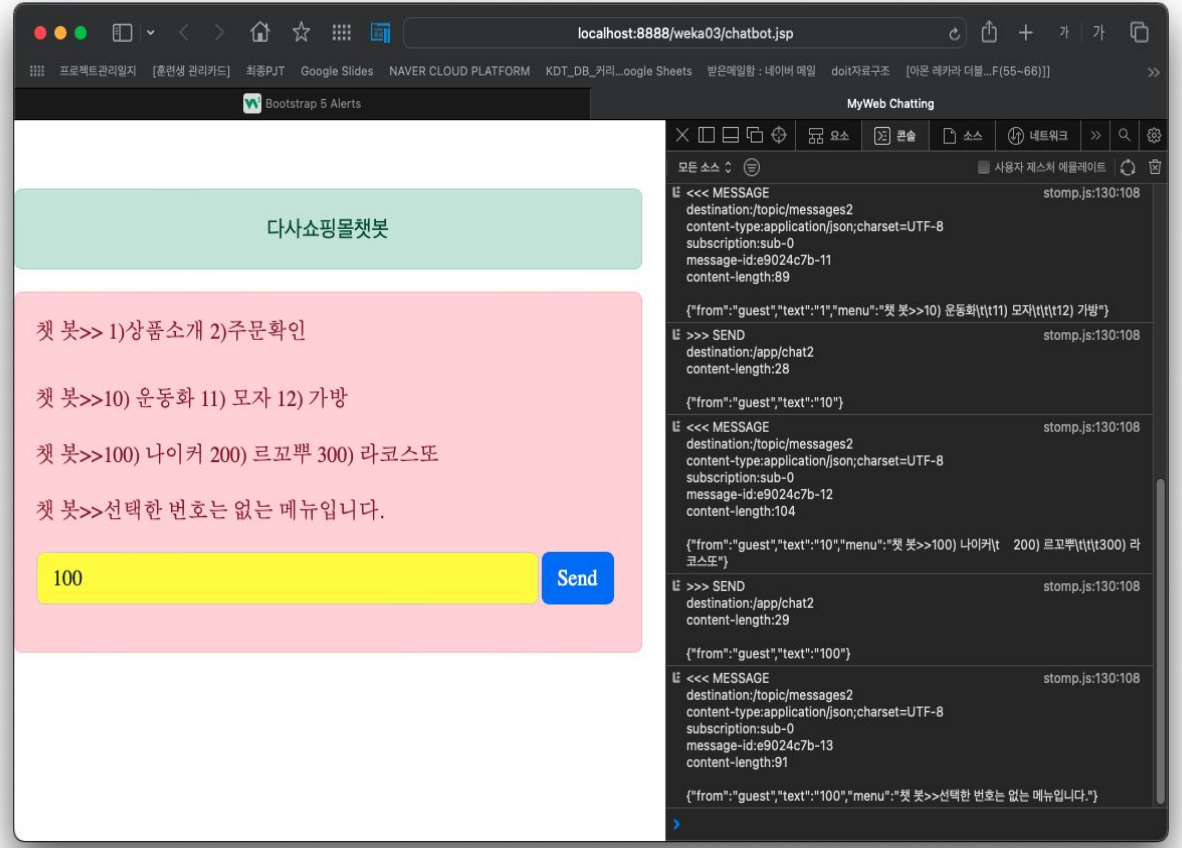
챗봇 시나리오 작성


MyWeb Chatting
localhost:888
다사쇼핑몰 챗봇
챗 봇>> 1)상품소개 2)주문확인
챗 봇>> 10) 운동화 11) 모자 12) 가방
챗 봇>> 100) 나이커 200) 르꼬뿌 300) 라코스또
챗 봇>> 1000) 나이커 운동화 세부 메뉴 1) 다시 처음 메뉴
챗 봇>> 1001)나이커빨강(1만원) 1002)나이커파랑(2만원) 1003)나이커보라(3만원) 100)이전 메뉴로
챗 봇>> 1000) 나이커 운동화 세부 메뉴 1) 다시 처음 메뉴
챗 봇>> 10) 운동화 11) 모자 12) 가방
1
Send


MyWeb Chatting
localhost:888
다사쇼핑몰 챗봇
챗 봇>> 1)상품소개 2)주문확인
챗 봇>> 20) 배송조회 21) 주문완료상품조회
챗 봇>> DB에서 가지고 온 배송 상황 목록 상품명: 르꼬뿌 배송상황: 물건 준비중
챗 봇>>선택한 번호는 없는 메뉴입니다.
21
Send


안녕하세요. 가지가지 Chatbot 입니다.
1. 거래 관련
2. 배송 관련
3. 예치금 관련
4. 상담원 연결
1.사기 신고
2.신고 접수 확인
3.거래 방법 설명
4.거래 금지 품목
1.배송 조회
2.배송 방법
3.배송 일정
1.예치금 충전
2.예치금 인출
3.예치금 조회
1.채팅 연결
2.전화 연결
switch
case 1:
신고 접수는 신고 게시판을 이용해주세요!
신고하기
case 2:
이메일을 확인하세요!
case 3:
1.판매자와 채팅
2.구매신청
3.거래장소
4.거래하기
case 4:
1.흉기 및 마약류
2. 주류
3. 의약품
switch
case 1:
운송장번호로 검색하기
case 2:
드론택배 준비 중..
case 3:
영업일 기준 2~5일 내 배송될 예정이며, 도서 산간 지역의 경우 추가 금이 발생할 수 있습니다.
switch
case 1:
마이페이지-가지페이-예치금 충전
case 2:
마이페이지-가지페이-예치금 인출
case 3:
마이페이지-가지페이-예치금 조
switch
case 1:
잠시만 기다려주세요!
case 2:
010-9292-4989
상담가능시간:
평일 오전 10시~
오후 6시

WebSocket


localhost:8888/weka03/chatbot.jsp
Bootstrap 5 Alerts
MyWeb Chatting
다사쇼핑몰챗봇
챗 봇>> 1)상품소개 2)주문확인
챗 봇>>10) 운동화 11) 모자 12) 가방
챗 봇>>100) 나이커 200) 르꼬뿌 300) 라코스또
챗 봇>>선택한 번호는 없는 메뉴입니다.
100
Send
<<< MESSAGE
destination:/topic/messages2
content-type:application/json;charset=UTF-8
subscription:sub-0
message-id:e9024c7b-11
content-length:89
{"from":"guest","text":"1","menu":"챗 봇>>10) 운동화\t\t11) 모자\t\t\t12) 가방"}
>>> SEND
destination:/app/chat2
content-length:28
{"from":"guest","text":"10"}
<<< MESSAGE
destination:/topic/messages2
content-type:application/json;charset=UTF-8
subscription:sub-0
message-id:e9024c7b-12
content-length:104
{"from":"guest","text":"10","menu":"챗 봇>>100) 나이커\t 200) 르꼬뿌\t\t\t300) 라코스또"}
>>> SEND
destination:/app/chat2
content-length:29
{"from":"guest","text":"100"}
<<< MESSAGE
destination:/topic/messages2
content-type:application/json;charset=UTF-8
subscription:sub-0
message-id:e9024c7b-13
content-length:91
{"from":"guest","text":"100","menu":"챗 봇>>선택한 번호는 없는 메뉴입니다."}

WebSocket

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Insert title here</title>
    <script type="text/javascript"
            src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
    <link
            href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
            rel="stylesheet">
    <script
            src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"></script>
    <script type="text/javascript" src="resources/js/sockjs-0.3.4.js"></script>
    <script type="text/javascript" src="resources/js/stomp.js"></script>
    <script type="text/javascript" src="resources/js/webSocketSendToUserApp.js"></script>
    <script type="text/javascript">
        var stompClient = null;

        function setConnected(connected) { //연결 여부에 따라 설정
            //document.getElementById('connect').disabled = connected;
            //document.getElementById('disconnect').disabled = !connected;
            document.getElementById('conversationDiv').style.visibility = connected ? 'visible'
                : 'hidden';
            //$('#response').html('')와 동일한 코드
            document.getElementById('response').innerHTML = '';
        }
```

WebSocket

```
function connect() {
    //1. 소켓객체 생성
    var socket = new SockJS('${pageContext.request.contextPath}/chat2')
    //2. 데이터를 전송하고, 받을 수 있는 stompClent객체 생성
    stompClient = Stomp.over(socket);

    //3. 채팅방 지정하여 가입하자.
    stompClient.connect({}, function(frame) {
        setConnected(true) //css설정
        alert('연결됨. '+ frame)
        stompClient.subscribe('/topic/messages2', function(messageOutput) {
            //서버에서 받은 메시지 출력
            showMessageOutput(JSON.parse(messageOutput.body));
        })
    })
    //4. 서버에서 보냈을 때 어떻게 처리할지 지정
}

//서버로 메세지 보냄
function sendMessage() {
    //입력한 정보를 가지고 와서
    var from = "guest";
    var text = document.getElementById('text').value;
    //url을 /app/cht을 호출하고,data를 json형태의 sring으로 만들어서 보내라.
    stompClient.send("/app/chat2", {}, JSON.stringify({
        'from' : from,
        'text' : text
    }));
}
```

```
//받은 데이터를 원하는 위치에 넣음. ==> 받은 데이터를 채팅목록으로 쌓는다.
    function showMessageOutput(messageOutput) {
        //<p id="response">
        //  <p> 홍길동: 잘지내지?(13:00)</p>
        //</p>
        var response = document.getElementById('response');
        var p = document.createElement('p');
        p.style.wordWrap = 'break-word';
        p.appendChild(document.createTextNode(messageOutput.menu));
        /*aTag = document.createElement('a');
        aTag.setAttribute('href', 'http://www.naver.com')
        aTag.innerHTML = '나는 aTag text'
        p.appendChild(aTag);
        */
        response.appendChild(p);
    }

    //서버로 연결 끊음.
    function disconnect() {
        if (stompClient != null) {
            stompClient.disconnect();
        }
        setConnected(false); //연결끊어졌을 때 설정 변경
        console.log("Disconnected");
    }

</script></head>
```

```
<body onload="connect();">
<br>
<br>
<img src=resources/image/chat1.png width=100 height=100><img src=resources/image/chat3.png width=400 height=100>
<div class="alert alert-danger" style="width: 500px;">
    <div>챗 봇>> 1)상품소개    2)주문확인</div>
    <br>
    <div id="response">

    </div>
    <div class="form-floating mb-3 mt-3" id="conversationDiv">
        <table>
            <tr>
                <td><input type="text" class="form-control" id="text" style="width: 400px; background: yellow"></td>
                <td><button id="sendMessage" onclick="sendMessage();"class="btn btn-primary">Send</button></td>
            </tr>
        </table>
    </div>
</div>
</body>
</html>
```

웹서버와 연결할 주소와 통신할 수 있는 소켓 생성

stompClient객체 생성-메시지의 send/receive담당

/topic/messages채팅방에 가입

서버와 연결하여 핸드쉐이킹

서버에서 가입한 클라이언트에게 메시지를 보냈을 때 받아서 처리하는 함수, 서버로 부터 메시지를 받을 때, JSON형태의 String으로 온 것을 JSON객체로 변환함.

서버로 메시지를 보낼 때, JSON형태의 string으로 전송

```
var socket = new SockJS('${pageContext.request.contextPath}/chat2');
//2. 데이터를 전송하고, 받을 수 있는 stompClient객체 생성
stompClient = Stomp.over(socket);

//3. 채팅방 지정하여 가입
stompClient.connect({},
    setConnected(true) //css설정
    alert('연결됨. '+ frame)
    stompClient.subscribe('/topic/messages2', function(messageOutput){
        //서버에서 받은 메시지를 출력
        showMessageOutput(JSON.parse(messageOutput.body));
    })
})
//4. 서버에서
}

//서버로 메세지
function sendMessage() {
    //입력한 정보를 가지고 와서
    var from = "guest";
    var text = document.getElementById('text').value;
    //url을 /app/cht을 호출하고,data를 json형태의 string으로 만들어서 보내라
    stompClient.send("/app/chat2", {}, JSON.stringify({
        'from' : from,
        'text' : text
    }))

에 넣음. ==> 받은 데이터를 채팅목록으로 쌓는다.
(messageOutput) {
    //<p id="response">
    //  <p> 홍길동: 잘지내지?(13:00)</p>
    //</p>
    var response = document.getElementById('response');
```

```
    }

    @MessageMapping("/chat2")
    @SendTo("/topic/messages2")
    public OutputMessage2 send2(Message message) throws Exception {
        String menu = "";
        switch (message.getText()) {
        case "1":
            menu = "챗 봇>>  10) 운동화     11) 모자          12) 가방";
            break;
        case "2":
            menu = "챗 봇>>  20) 배송조회   21) 주문완료상품조회";
            break;
        case "20":
            menu = "챗 봇>> DB에서 가지고 온 배송 상황 목록 \n" +
                    "상품명: 르꼬뿌     배송상황: 물건 준비중";
            break;
        case "10":
            menu = "챗 봇>>  100) 나이커        200) 르꼬뿌        300) 라코스또";
            break;
        case "100":
            menu = "챗 봇>>  1000) 나이커 운동화 세부 메뉴  1) 다시 처음 메뉴";
            break;
        case "1000":
            menu = "챗 봇>>  1001)나이커빨강(1만원) 1002)나이커파랑(2만원)  1003)나이
            break;
        default:
            menu = "챗 봇>>선택한 번호는 없는 메뉴입니다. 다시 입력해주세요.";
            break;
        }
        return new OutputMessage2(message.getFrom(), message.getText(), menu);
    }

}
```

```
//브라우저(클라이언트)가 입력해서 서버로 전송한 내용을 받는 역할
public class Message extends Object {

    private String from;
    private String text;
```

```
    public OutputMessage2(String from, String text, String menu) {
        super();
        this.from = from;
        this.text = text;
        this.menu = menu;
    }
```

핵심 요약

1. **WebSocket**
2. **STOMP**
3. **spring WebSocket**
4. 브로커 기반 메시징
5. 양방향 통신