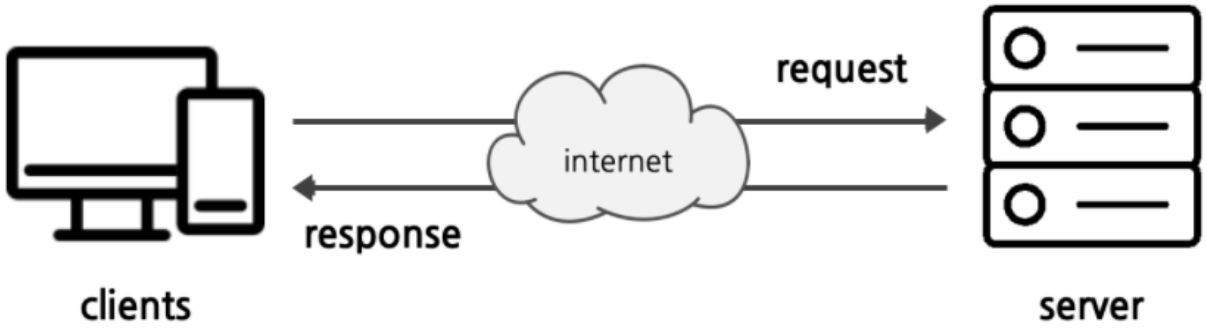
고급 소프트웨어 실습(CSE4152)

9주차
React – Chatting App(2)

목차

web client & server



- 웹은 웹 클라이언트(브라우저) 프로그램과 웹 서버 프로그램으로 구성
- 클라이언트와 서버는 인터넷으로 서로 연결
- 클라이언트는 사용자와 직접적으로 상호 작용을 하며 사용자가 필요로 하는 데이터를 서버로 요청(request)
- 서버는 클라이언트로부터 요청 받은 데이터를 클라이언트로 보내 줌 (response)

React & Express 연결

- Express로 간단한 서버를 구현할 수 있음
- Express 서버와 React 어플리케이션을 연결
- 실습 채팅 프로그램에서 사용되는 tool의 역할
  - React.js : 프론트엔드 라이브러리
  - Express.js : API, 어플리케이션 기능을 담당하는 서버 구현을 위한 프레임워크
  - Axios : 서버로 HTTP request 를 보내기 위해 사용하는 npm 라이브러리
  - CORS: 추가적인 HTTP 헤더를 사용하여 서로 다른 origin(url 구조)의 리소스에 접근하기 위해 사용하는 라이브러리

HTTP methods

- HTTP 메소드는 클라이언트가 웹 서버로 user request의 목적, 종류 등의 정보를 전달하는 수단이다.
- HTTP 메소드에는 GET, POST, HEAD, PUT, DELETE 방식 등이 있다.
- 주로 사용되는 메소드는 GET, POST 방식 메소드

Get method

- URL에 원하는 데이터를 포함하여 request를 보낼 수 있어 데이터 검색 기능 등에 적합하다.
- 바이너리 및 대용량 데이터 전송이 불가능하다.
- request line과 header의 최대 크기가 웹 서버에 따라 제한된다.
- 웹 주소창에 URL을 직접 입력하거나 링크를 클릭, 웹 페이지 코드의 method 속성 값이 get인 경우에 GET 메소드 방식의 request가 발생한다.
- 자료 검색, 게시글 조회, 특정 데이터 조회 등에 사용된다.
- url에 데이터가 그대로 노출되므로 보안 문제가 발생한다.

Post method

- GET 메소드 방식과 다르게 url에 데이터가 노출되지 않는다.
- Request message body에 데이터가 포함된다.
- 바이너리 및 대용량 데이터의 전송이 가능하다.
- 전송 데이터의 최대 크기가 제한되지 않는다.
- 캐시, 브라우저 히스토리 기록, 북마크가 되지 않는다.
- 보안이 요구되는 데이터, 혹은 길이가 매우 긴 데이터를 전송해야 하는 경우에 주로 사용된다.

Port / Port Number

- Port number는 네트워크 상에서 데이터를 주고 받는 프로세스를 식별하기 위해 각 프로세스가 할당 받는 host 내부의 고유한 번호이다.
- Port number를 통해 동일한 IP 내에서 각각의 프로세스를 구분할 수 있다.
- 16 bits로 이루어져 있어 총 65,536개까지 port number가 존재할 수 있다.
- “Well-known port”로 지정된 포트 번호들이 있다.
  - 21 : FTP (파일 전송)
  - 22 : SSH (원격 시스템 접속)
  - 80 : HTTP (웹)
  - 25: SMTP (전자우편)

React – useState() 함수

- state: 사용자와의 상호작용으로 인해 데이터 값이 바뀌는 컴포넌트의 동적인 값으로, 웹에서는 state를 관리해주어야 한다.
- React 패키지의 useState() 함수를 사용해서 state를 관리 할 수 있다.
- 선언 형태: const [‘state변수이름’, ‘state의set함수이름’] = useState(‘state초기화값’);
- state에는 숫자, 문자열, 배열, 객체, 객체 배열 등 다양한 값이 쓰일 수 있다.
- state의 set 함수로 state에 원하는 값을 저장하며, 다른 함수에서 사용 가능하다.
- useState() 함수 사용 예:

```jsx
import React, { useState } from 'react';

function Counter() {
  // number를 0으로 초기화
  const [number, setNumber] = useState(0);

  const onIncrease = () => {
    setNumber(num => num + 1);
  }

  return (
    <div>
      <h1>{number}</h1>
      {/* 버튼 클릭 시 number가 1씩 증가 */}
      <button onClick={onIncrease}>+1</button>
    </div>
  );
}

export default Counter;
```

React – useEffect() 함수

- React 패키지의 useEffect() 함수를 사용하여 컴포넌트가 처음 나타났을 때(mount), 사라질 때(unmount) 원하는 특정 작업을 처리할 수 있다.
- 처음 창을 킬 때, 새로 고침 시 등의 상황에서 실행됨
- useEffect() 함수 사용 예:

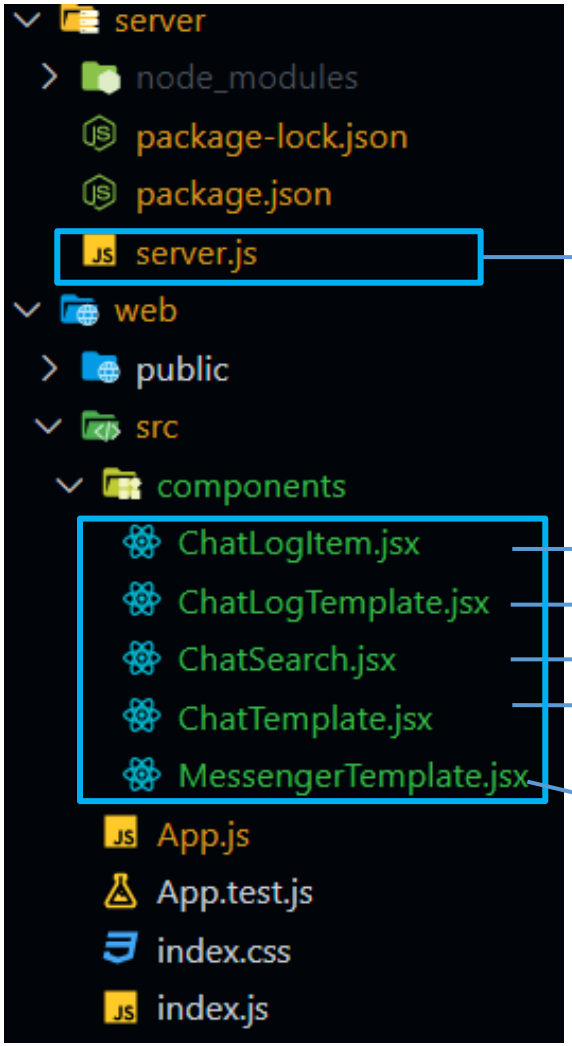
```
import React, { useEffect } from 'react';

function Comp() {
  useEffect(() => {
    console.log('컴포넌트가 화면에 나타남');
    return () => {
      console.log('컴포넌트가 화면에서 사라짐');
    };
  }, []);
  return (
    ...
```

실습

- 기존의 완성된 채팅 프로그램(8주차)에서 express server를 사용하여 채팅 기능을 구현

실습 파일 구조



9주차: 서버 사용하여 채팅 기능 구현

채팅 서버 기능이 구현된 파일

- 채팅 한 줄을 표현하는 컴포넌트
- 채팅 기록이 표현되는 컴포넌트
- 채팅 내용 검색창을 표현하는 컴포넌트 (empty)
- 채팅 프로그램 페이지의 하위 컴포넌트를 감싸는 최상위 템플릿 컴포넌트
- 사용자의 채팅 내용을 입력 받는 폼 컴포넌트

Server.js

- 실습 채팅 프로그램에서 server.js는 Express를 사용해 8001 포트 번호를 할당한 서버를 생성
- 기록된 채팅 데이터를 가지고 있음
- GET method request가 오면 Express 서버에서 채팅 데이터를 response로 처리하는 함수 구현
- POST method request가 오면 Express 서버에 메시지로 전달된 채팅 내용을 데이터에 새로 추가하는 함수 구현

Server.js

```
1   const express = require('express');
2   const http = require('http');
3   const app = express();
4   const cors = require('cors');
5   const bodyParser = require('body-parser');
6
7   app.use(bodyParser.urlencoded({ extended: true }));
8   app.use(bodyParser.json())
9   app.use(cors());
10  app.set('port', process.env.PORT || 8001);
11
12  //서버 생성
13  http.createServer(app).listen(app.get('port'), function(){
14      console.log('Express server listening on port ' + app.get('port'));
15  });
16
17  let data = [
18
19  ]
```

cors 사용하여 다른 origin의 리소스에도 접근 가능

8001번 포트를 사용하도록 포트 번호 설정

지정한 포트(8001번) 사용하는 서버를 생성

채팅 내용을 저장하는 data를 빈 배열로 초기화

Server.js

```
21  app.use('/get', (res) => {
22      res.send(data)
23  });
24
25  app.use('/post', (req) => {
26      const message = req.body.message
27      const name = req.body.name
28
29      data = [...data, {
30          'message': message,
31          'name': name
32      }]
33
34      console.log(data)
35  });
```

GET method request에 대한 response로 현재 채팅 데이터를 전송

POST method request message의 name과 message 정보 저장

data에 request message의 채팅 내용 저장

콘솔에 현재 data 내용 출력

채팅 앱 로컬에서 실행

- 현재 구현한 채팅 프로그램 서버에서는 8001번 포트, 로컬(클라이언트)에서는 3000번 포트를 사용
- docker 컨테이너에 접속할 때에 기존과 달리 8001번 포트도 추가적으로 포트 포워딩 설정을 해주어야 함
  - docker 실행 명령어에 -p 8001:8001 옵션 추가

단방향 통신의 한계

- Express 서버와 GET, POST 방식으로 어플리케이션을 구현할 시에 다른 사용자가 보낸 메시지를 수동 새로 고침을 해야만 확인이 가능하다.
- 현재 단방향 통신 방식, 즉 계속해서 GET request를 보내 서버의 데이터를 가져오는 방식은 비정상적인 프로그램 동작을 유발한다.
- 실시간 채팅을 위해서는 websocket, socket.io 방식으로 양방향 통신을 구현하는 것이 바람직하다.

양방향 통신 – Websocket

- Websocket은 웹 서버와 클라이언트의 실시간 양방향 통신이 가능한 환경을 제공해주는 통신 기술
- 원할 때 양방향으로 요청을 보낼 수 있음

양방향 통신 – Socket.io

- Socket.io 역시 웹 서버와 클라이언트의 실시간 양방향 통신이 가능한 환경을 제공해주는 통신 기술
- 통신 시작 시 각 브라우저에 대해 websocket, polling, streaming 방식들 중 최적의 방식을 찾아 메시지를 보내 줌

실습

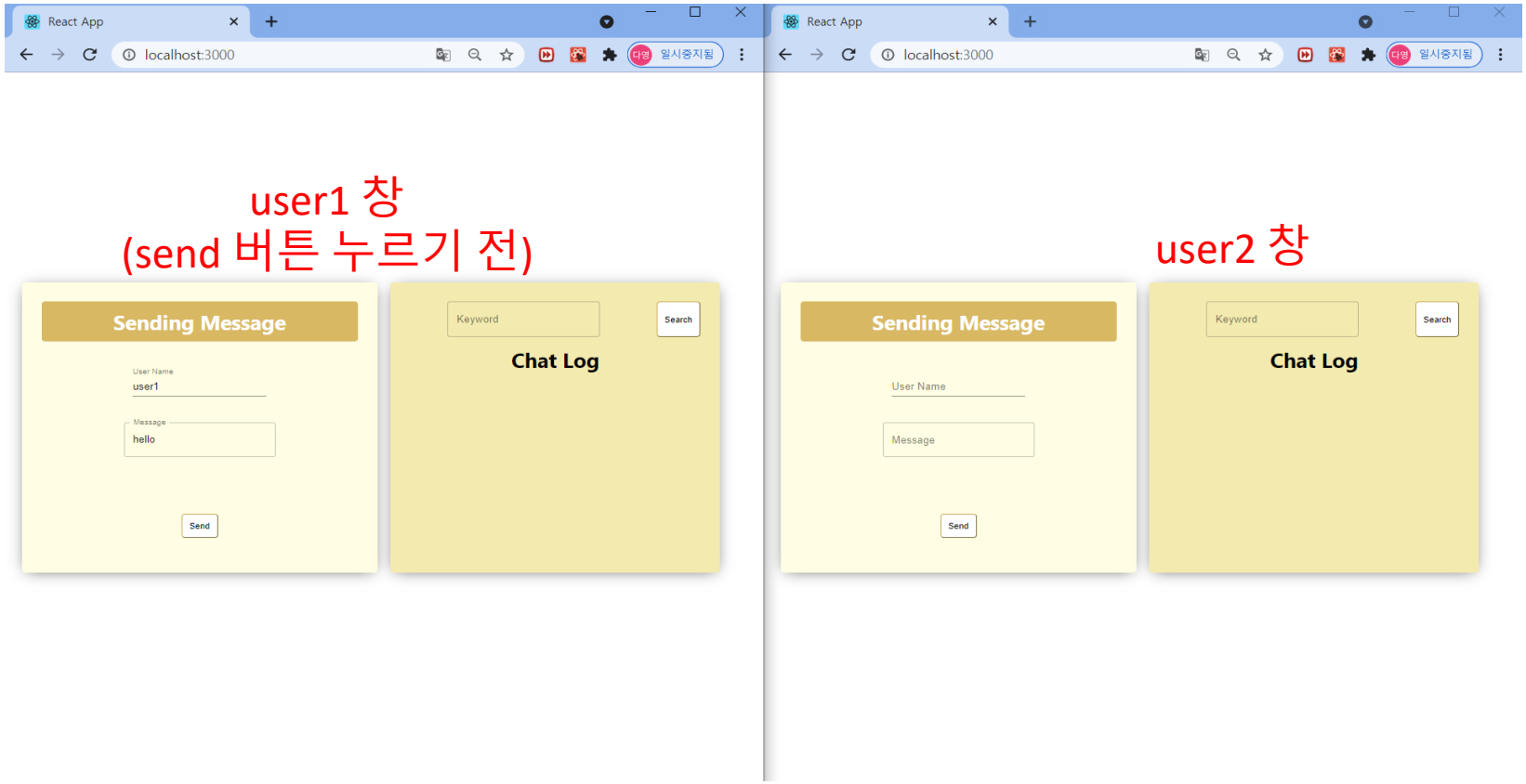
- 구현해야 할 함수
  - Web > src > components 안 파일들
  - ChatTemplate.jsx
    - getChatLog: axios.get 함수를 사용해서 채팅 데이터 가져오고 현재 chats state에 저장
    - useEffect: 처음 창을 키거나 새로 고침 시에 채팅 기록을 가져 옴(getChatLog 함수 이용)
  - MessengerTemplate.jsx
    - onMsgSubmit:
    - 메시지 입력 창에서 빈 경우에는 '메시지를 입력하세요'라는 내용의 경고 창 출력 후 리턴
    - 사용자의 이름 입력 창이 빈 경우에는 사용자 이름을 '이름없음'으로 저장
    - message와 username의 속성을 갖는 채팅 데이터를 axios.post 함수 사용하여 서버로 전송

실습

- 구현해야 할 함수
  - ChatSearch.jsx
    - 채팅 키워드 검색창 & 검색 버튼에 대한 컴포넌트 작성
    - 스타일: 완성 모습 이미지에 가깝게 구현
    - 기능: ChatTemplate 컴포넌트의 onSearchKeySubmit 함수를 사용해서 검색창에 키워드 검색 시 해당 내용을 포함하는 채팅만 빨간색으로 표시
  - ChatLogTemplate.jsx
    - ChatSearch 컴포넌트가 표시되도록 수정

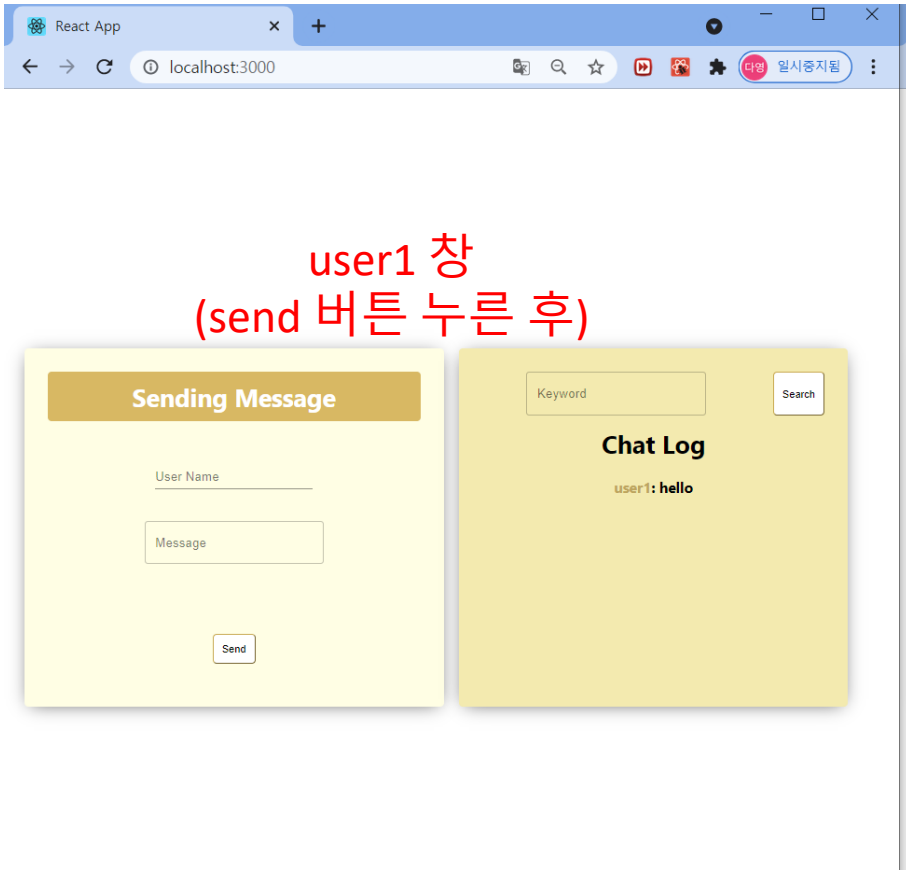
실습

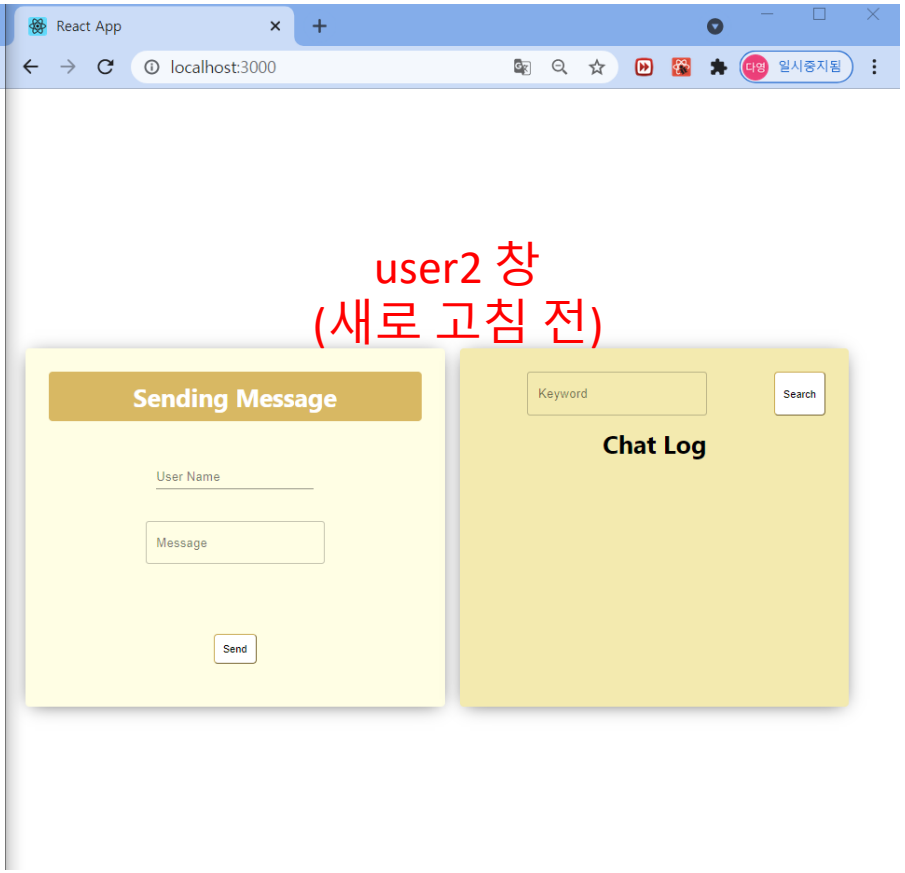
- 완성 모습 – 채팅


React App
localhost:3000
일시중지됨
user1 창
(send 버튼 누르기 전)
Sending Message
User Name
user1
Message
hello
Send
Keyword
Search
Chat Log
React App
localhost:3000
일시중지됨
user2 창
Sending Message
User Name
Message
Send
Keyword
Search
Chat Log

실습

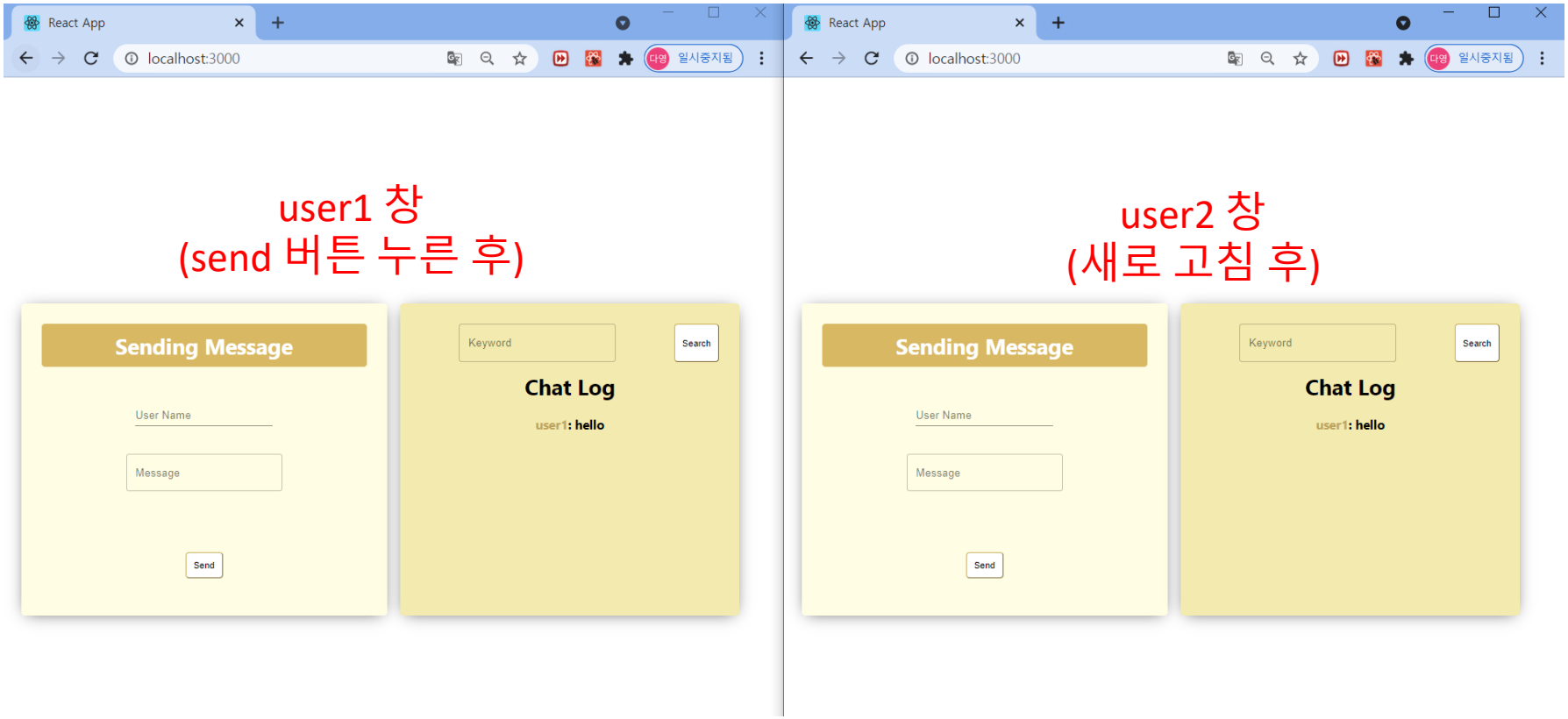
- 완성 모습 – 채팅


React App
localhost:3000
일시중지됨
user1 창
(send 버튼 누른 후)
Sending Message
User Name
Message
Send
Keyword
Search
Chat Log
user1: hello


React App
localhost:3000
일시중지됨
user2 창
(새로 고침 전)
Sending Message
User Name
Message
Send
Keyword
Search
Chat Log

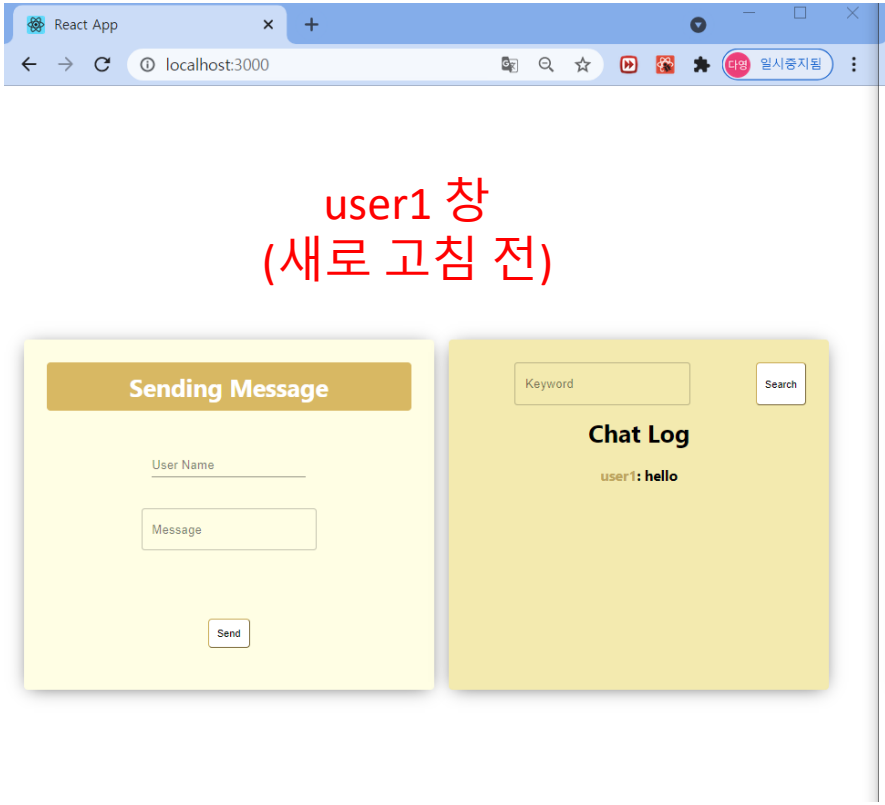
실습

- 완성 모습 – 채팅

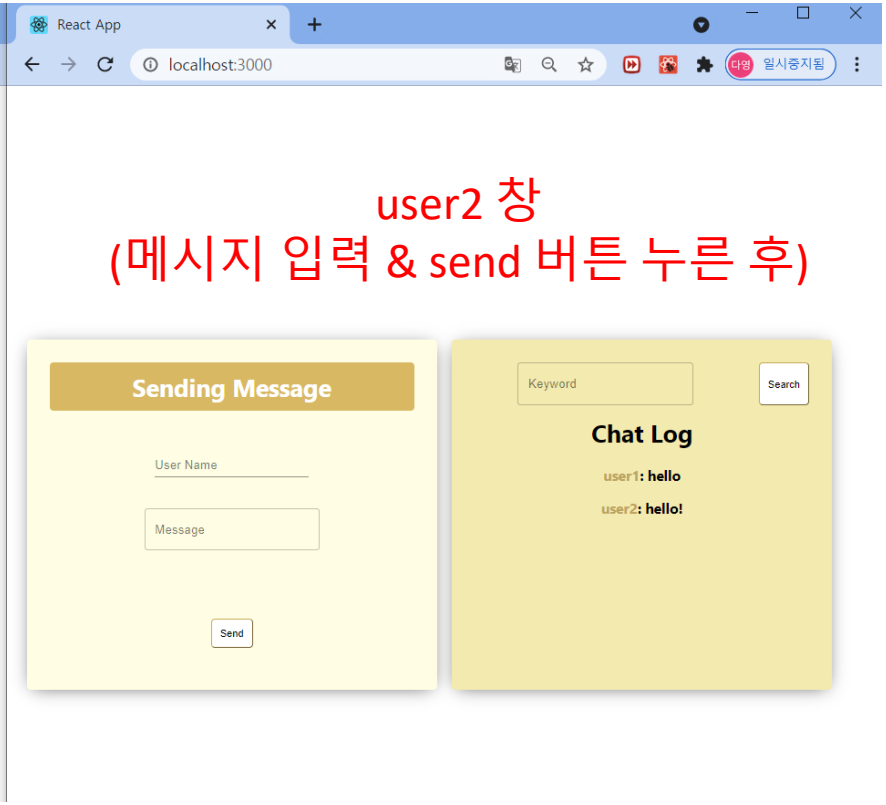

React App
localhost:3000
일시중지됨
user1 창
(send 버튼 누른 후)
Sending Message
User Name
Message
Send
Keyword
Search
Chat Log
user1: hello
React App
localhost:3000
일시중지됨
user2 창
(새로 고침 후)
Sending Message
User Name
Message
Send
Keyword
Search
Chat Log
user1: hello

실습

- 완성 모습 – 채팅



user1 창
(새로 고침 전)

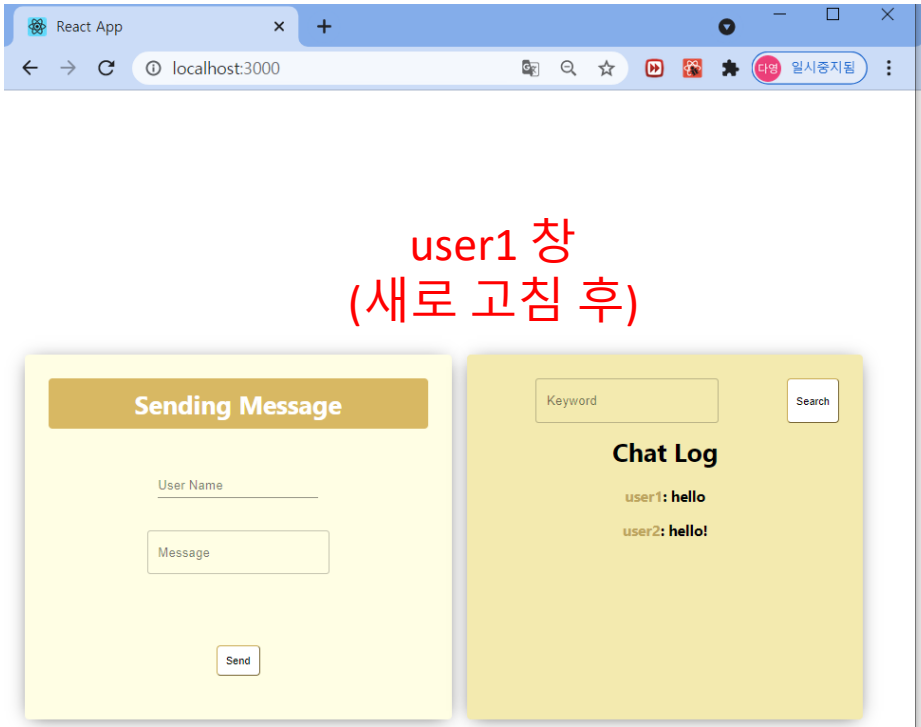


user2 창
(메시지 입력 & send 버튼 누른 후)

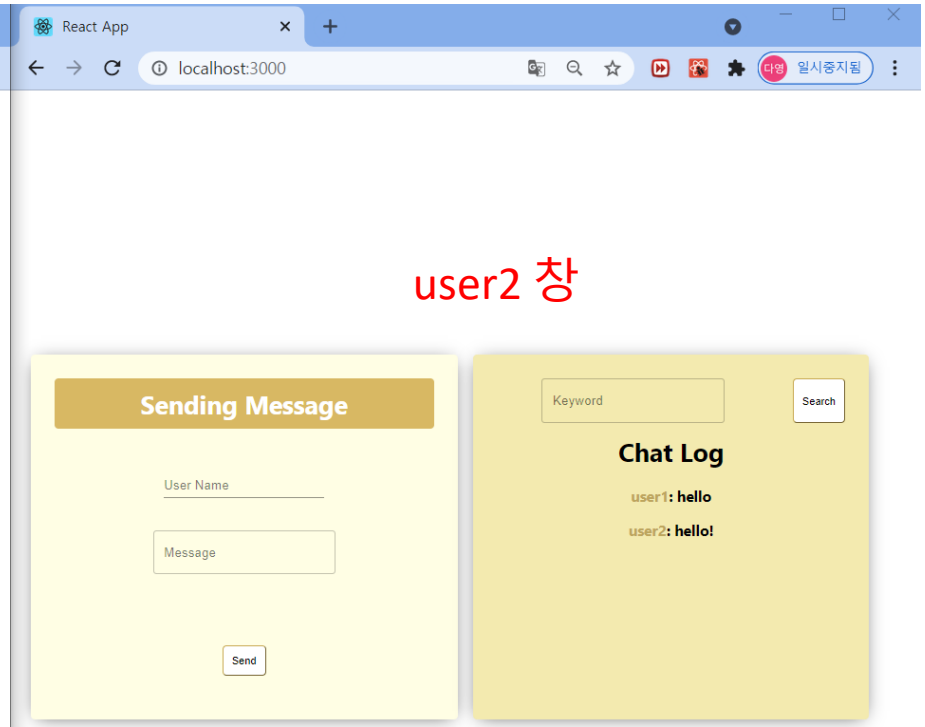
실습

- 완성 모습 – 채팅

user1 창
(새로 고침 후)

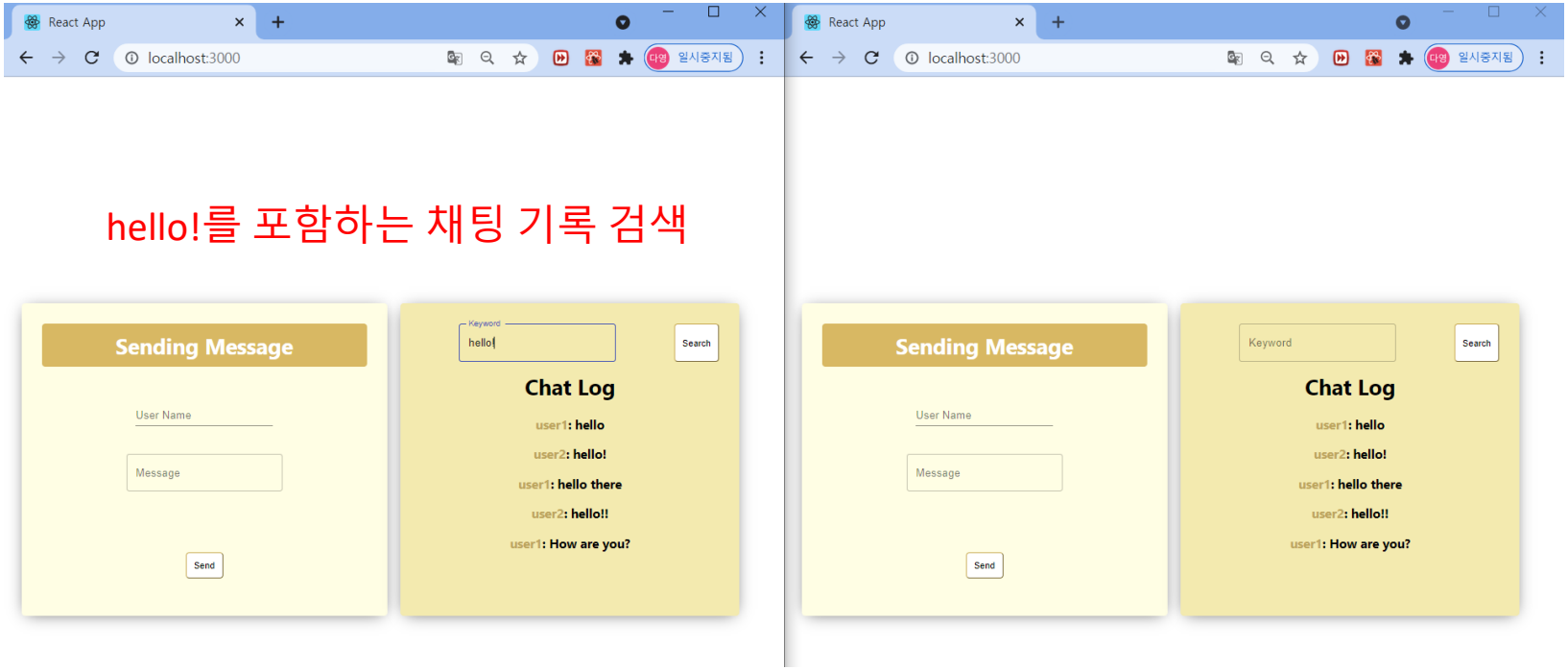


user2 창



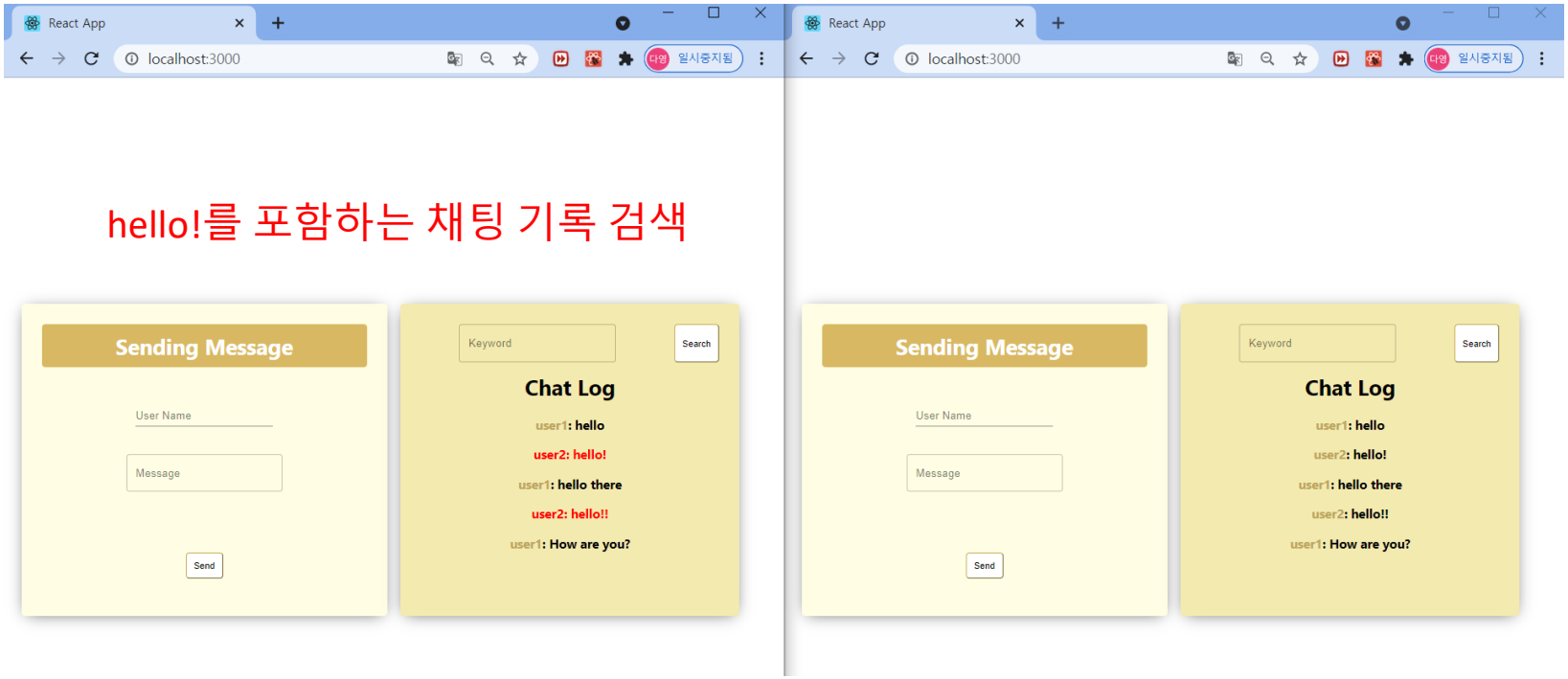
실습

- 완성 모습 – 키워드 검색



실습

- 완성 모습 – 키워드 검색



과제: 보고서 – websocket, socket.io 조사

- websocket과 socket.io 통신 방식에 대해 자세히 조사한 후 이에 대해 기술
- 실제로 websocket/socket.io 통신 방식을 사용해서 실제 채팅 프로그램을 구현하게 됐을 때의 전체적인 구현 방식 기술

참고

- JavaScript 레퍼런스 (MDN) : https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
- JavaScript 레퍼런스 (MSDN) : https://docs.microsoft.com/ko-kr/previous-versions/visualstudio/visual-studio-2010/z688wt03(v=vs.100)?redirectedfrom=MSDN
- Express 레퍼런스 (MDN) : https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs
- 웹 환경 관련 : https://opentutorials.org/course/3084/18890