# 클라우드컴퓨팅

# HW #1

- Web server with socket API


과목명 | 클라우드컴퓨팅

분반 | 1분반

담당 교수 | 유시환 교수님

학과 | SW융합경제경영

학번 | 32151906

이름 | 박준형


제출일 | 2022.04.11

# 개요

**과제 목표**

가) Socket API를 사용하여 웹 서버 구현

나) 구현된 웹 서버는 기본적인 GET 요청을 처리할 수 있어야 하고, 요청 받은 파일을 클라이언트에게 전달할 수 있어야함

다) 보너스 1 : 동시다발적 요청 수행 구현

라) 보너스 2 : 동적 웹사이트 구현

**과제 산출물**

가) 웹 서버 소스 코드

나) 레포트

**개요**

본 레포트는 소켓 API를 활용하여 파이썬으로 구현된 웹 서버 소스 코드의 사용 방법과 작동 과정을 설명한 후, 이에 대한 고찰을 제시한다.

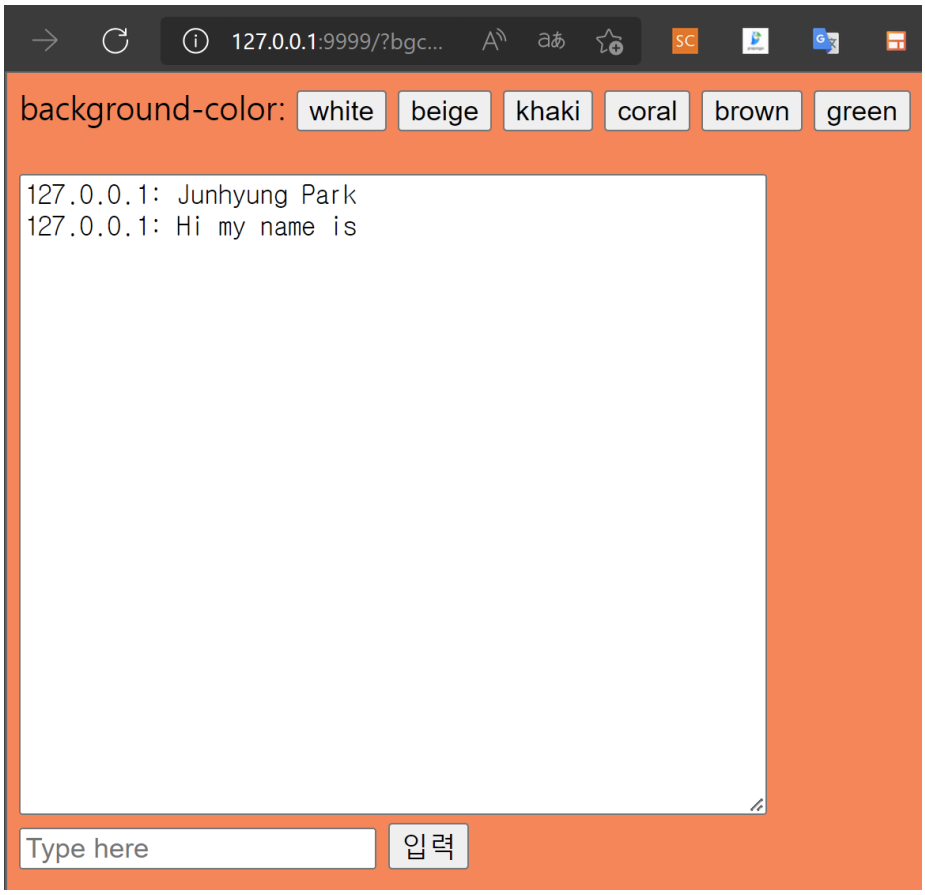
# 목차

# 웹 서버

## 프로그램 사용 방법

```
(base) C:\Users\Admin\Documents\클라우드 컴퓨팅\Webserver>python main.py
Start Linstening
```

(그림 1) 프로그램 실행 방법




(그림 2) 웹 사이트

가) 터미널을 통해 웹 서버 소스코드가 있는 디렉토리에서 'python main.py'를 입력하여 main.py 실행

나) 소스코드에 설정되어 있는 'ip:port'를 웹 브라우저 주소 입력창에 입력하여 접속 (기본 설정 = '127.0.0.1:9999')

다) background-color: 우측의 버튼을 통해 배경 색 변경 가능

라) 'Type here'이 적힌 곳에 텍스트를 작성 후 입력 버튼을 누르면 상단의 textarea에 텍스트 기록

**프로그램 작동 과정**

가) 소켓 API를 통해 설정된 호스트와 포트에 서버 소켓을 바인딩한 후 클라이언트로부터의 연결을 대기

나) 클라이언트와 연결되면, 해당 클라이언트와의 연결 소켓을 _Client 클래스 객체에 담고 해당 소켓과의 개별 통신을 담당하는 CommunicationManager 클래스의 _communicating 메서드 스레드를 생성 후 다시 다른 클라이언트와의 연결을 대기하는 과정을 While 반복문으로 수행

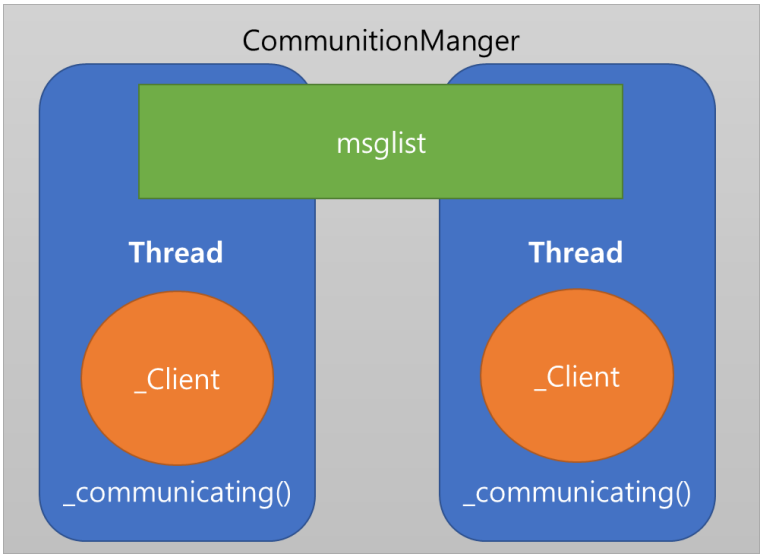
-_communicating 스레드의 클라이언트 소켓과의 통신 과정 :

A. 클라이언트가 background-color 우측의 버튼을 클릭 시, 서버에는 해당 버튼에 적힌 색깔 값이 담긴 GET 요청이 전송되며, 이는 서버의 _Client 클래스 객체의 color 속성에 저장

B. 클라이언트가 'Type-here'이 적힌 곳에 텍스트를 작성 후 입력 버튼을 클릭 시, 서버에는 해당 텍스트 값이 담긴 POST 요청이 전송되며, 해당 텍스트는 서버의 CommunicationManager 클래스 객체 내의 msglist 배열에 저장 (해당 msglist 배열은 모든 _communicating 스레드가 공유)

C. 클라이언트의 요청에 대한 응답으로, 기본 HTML 파일을 _Client 클래스 객체의 color 속성에 따라 수정하고 msglist 내의 텍스트들을 기본 HTML 파일의 textarea에 삽입 후 전송

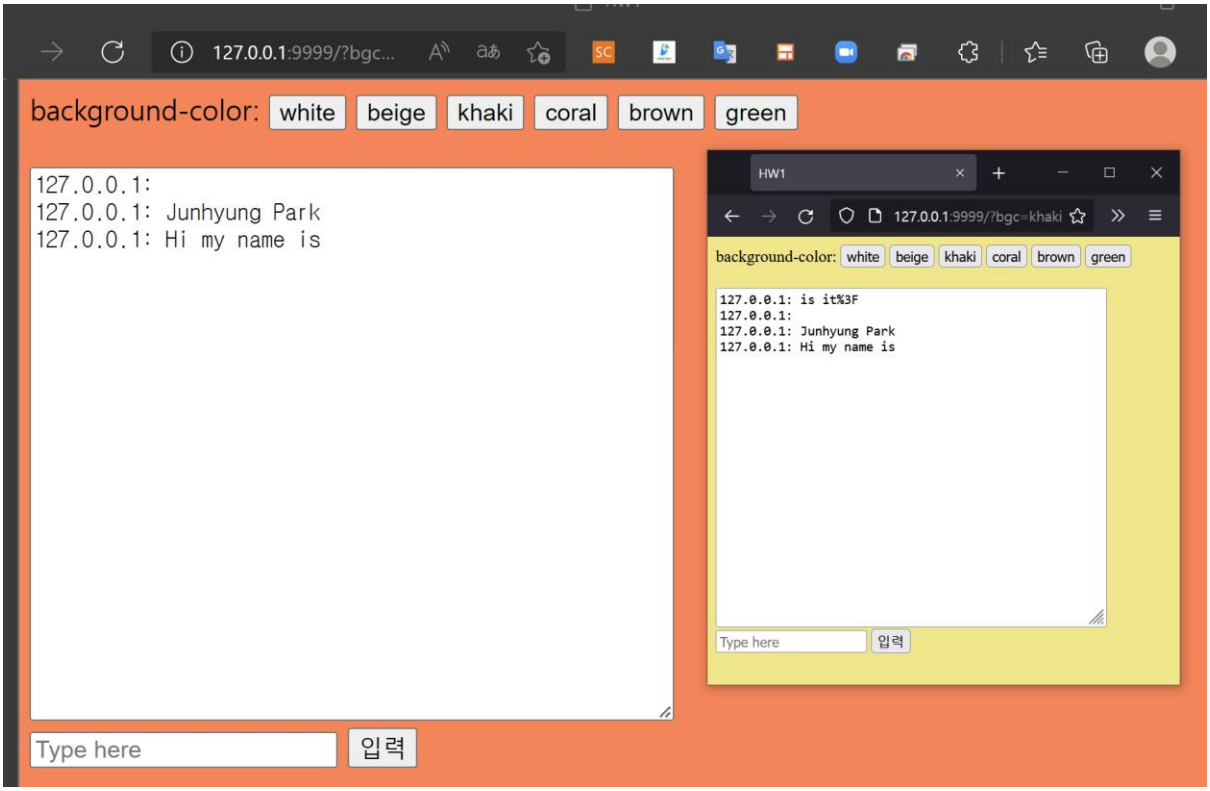
다) 상기의 과정에서 try-except 구문에 따라 반복문이 break되면 서버 소켓을 닫고 프로그램 종료




(그림 3) msglist를 공유하는 _communicating 스레드

# 고찰

## 실시간 채팅 서비스




(그림 4) 실패한 실시간 채팅 서비스

원래는 실시간 채팅 서비스를 구현하고 싶었는데, (그림 3)에서 좌측의 브라우저와 우측의 브라우저의 textarea에 서로 다른 내용이 담긴 것을 보면 알 수 있듯 구현하지 못했다. 실시간 채팅 서비스를 구현하려면 지금의 방식처럼 서버에서 정적 페이지를 수정하여 보낼 것이 아니라, 자바스크립트 등을 이용하여 클라이언트 브라우저가 통신을 받아 페이지를 실시간으로 렌더링하도록 했어야 했는데, 이걸 뒤늦게 알았다. 지금의 방식은 채팅이 아니라 게시판을 구현한 것이라고 볼 수 있다.

처음에는 웹 소켓을 통해 실시간 채팅 서비스를 구현하려 했는데, 이를 언어에서 자체적으로 제공하는 소켓 API만을 이용하여 구현하고 싶다는 생각이 들었다. 웹 소켓이 일종의 업그레이드된 프로토콜이라는 글을 읽어서 소켓 API만으로 쉽게 구현할 수 있을 줄 알았는데 결코 쉽지 않았고, 다음에는 롱 풀링 기법으로 구현하려 했는데, 롱 풀링 기법 구현의 난이도를 떠나서 결국 클라이언

트 브라우저에서 자바스크립트 등을 통해 실시간 렌더링해주지 않으면 브라우저 상에서의 실시간 채팅이 불가능하다는 것을 깨닫고 그냥 요청이 오면 응답을 보내는 기본적인 HTTP 통신 방식으로 구현했다.

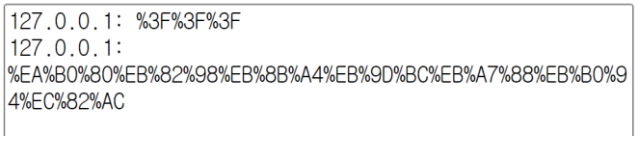
클라이언트 소켓별로 각각 receive 스레드를 만들고, 각각의 스레드에서 받은 요청들에서 메시지들을 모아 HTML을 수정하는 스레드를 만들고, HTML에 수정사항이 있을 때 마다 모든 클라이언트에게 한번에 응답을 보내는 스레드를 만들어서 구현한다던 지 등 다양한 방법들을 시도했었는데, 자바스크립트를 다룰 줄 알았더라면 기존의 방법들을 살릴 수 있었을 텐데 그러지 못해 결국 이도 저도 아닌 웹 사이트가 된 것 같아 아쉽다.

**favicon.ico**


(그림 5) 주소창 좌측의 favicon.ico

클라이언트에서의 브라우저를 통한 HTTP 요청은 한 번만 오지 않는다. 사용자가 브라우저를 통해 서버에 요청을 보내면 브라우저 자체적으로 favicon.ico에 대한 요청이 같이 오기 때문에 두 번의 요청이 오며, 이 favicon.ico는 주소창 좌측의 이모티콘이다. 처음에는 두 번의 요청이 오고 있다는 것조차 인지를 못해서 오류 처리 등에 애를 먹다가, favicon.ico에 대한 요청에는 특별한 처리를 하지 않도록 조건문을 작성하여 해결했다. 근데 실시간 채팅을 포기하게 되면서 굳이 따로 조건문을 둘 필요가 없어져서, 결국 해당 조건문과 관련한 코드들은 삭제됐다.

**브라우저에서의 서버로의 요청에 대한 인코딩**




(그림 6) '가나다라마바사'와 '???'를 'Type here'에 입력한 결과

(그림 6)은 해당 'Type here'에 한글과 ?를 입력했을 때 생기는 결과로, 이는 브라우저에서 정적 페이지를 렌더링하는 과정에서 깨진 것이 아니라 애초에 클

라이언트 브라우저에서 서버로 요청을 보낼 때 깨진 채로 전송이 되어 msglist에 저장된 것이다. 이러한 이유에 대해서 검색해보니 전송할 때 'EUC-KR'로 인코딩되어 서버로 전송이 되기 때문이며, xml이나 자바스크립트 등으로 'utf-8'로 인코딩되도록 일종의 필터를 만들어줘야 한다고 한다.

**HTTP 프로토콜**

알면 별것 아니지만 모르면 마냥 어렵고 싫고 무서운 것들이 있는데, HTTP 프로토콜이 그랬다. 사실 알고 보면 그냥 정해진 규격에 맞춰 헤더를 작성하여 보내고, 그 밑에 HTML 텍스트를 보내면 브라우저에서 알아서 렌더링하는 건데, 처음 봤을 때는 뭐가 너무 많다 보니 뭘 적어야 하고 뭘 수정해야 하는 지 난감했다.

뭘 적어야 하는 지를 알고 난 다음에는 일반적인 HTTP 통신의 요청-응답 방식에 대한 이해가 부족해서 문제가 생겼는데, 웹 소켓이 아닌 일반적인 HTTP 통신은 서버에서 아무리 응답해도 클라이언트에서 요청하지 않으면 클라이언트 브라우저에 영향을 미칠 수 없다. 어느 한 클라이언트에서 서버로 메시지가 들어와 서버에서 전체 클라이언트로 메시지를 보내려 해도, 클라이언트에서 요청이 오지 않으면 해당 메시지를 보낼 수가 없다. 이러한 문제를 롱 풀링 기법 등을 이용해 해결하더라도, 정적 페이지를 통해서 기록된 메시지들을 전달할 경우 어느 한 클라이언트 브라우저에서 메시지를 작성하던 중에 강제로 새로고침이 되어 작성 중이던 메시지가 날아간다던 지 하는 문제가 생긴다. 이러한 문제점들을 미리 알 지 못한 채 구현하는 과정에서 깨달았고, 덕분에 많이 배웠지만 다른 한편으로는 그만큼 시간이 걸렸다.