Flask 특강

4일차: 실시간 기능 (WebSocket)

목 차

- 3일차 복습
- HTTP vs WebSocket
- Flask WebSocket 기본
- 실시간 프로그램 실습: 실시간 알림, 타자기 효과, 숫자 카운터
- (5일차 프로젝트 공지)

3일차 복습

데이터베이스의 중요성

- 데이터 저장:
  - 사용자 정보, 게시글, 주문 내역 등 다양한 데이터를 파일 대신 체계적으로 저장
  - 데이터를 **영구적으로 보존**하며, 필요 시 언제든지 접근 가능

데이터베이스의 중요성

- 데이터 관리:
  - 구조화된 형태: DB는 데이터를 테이블 형태로 저장하여 관리
  - 검색 및 정렬: SQL 쿼리를 통해 데이터를 검색하거나 정렬 가능
  - 효율성: 인덱스(Index) 및 쿼리 최적화를 통해 대량 데이터도 빠르게 검색 가능

데이터베이스의 중요성

- 데이터 무결성:
  - 정확성 보장: 제약조건(Constraints)을 통해 잘못된 데이터 입력 방지
    - UNIQUE: 이메일 필드가 중복되지 않도록 설정
    - NOT NULL: 특정 필드는 비워둘 수 없도록 설정
    - FOREIGN KEY: 사용자와 게시글의 관계 유지
  - 트랜잭션(Transaction)
    - 데이터 일관성을 보장하기 위해 작업 단위를 묶어서 처리
    - ex) 주문이 완료되었을 때만 재고를 줄이고 & 결제 정보 저장

SQL과 ORM (Object-Relational Mapping)

SQL과 ORM 비교

SQL	ORM
SQL 쿼리 직접 작성 필요	Python 객체로 데이터 조작
데이터베이스에 최적화된 성능	유지보수 및 확장성이 높음
데이터베이스별 SQL 문법 차이 고려 필요	데이터베이스 독립적 코드 작성 가능

SQLAlchemy 소개

- Python에서 가장 널리 사용되는 ORM 및 데이터베이스 툴킷
- 관계형 데이터베이스와 상호작용하기 위한 강력하고 유연한 도구 제공
- 단순 데이터베이스 연동, 복잡한 쿼리 작성 등 다양한 요구를 충족할 수 있도록 설계됨

모듈화 소개

- Blueprint: 라우트 묶음

```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route("/todos")
def get_todos():
    return jsonify({"message": "Get Todos"})

@app.route("/users")
def get_users():
    return jsonify({"message": "Get Users"})

if __name__ == "__main__":
    app.run(debug=True)
```

```python
from flask import Flask, Blueprint, jsonify

# 1. Blueprint 객체 생성
todo_bp = Blueprint("todo", __name__)
user_bp = Blueprint("user", __name__)

# 2. Blueprint에 라우트 등록
@todo_bp.route("/todos")
def get_todos():
    return jsonify({"message": "Get Todos"})

@user_bp.route("/users")
def get_users():
    return jsonify({"message": "Get Users"})

# 3. Flask 앱에 등록
app = Flask(__name__)
app.register_blueprint(todo_bp)   # /todos
app.register_blueprint(user_bp)   # /users

if __name__ == "__main__":
    app.run(debug=True)
```

모듈화 소개

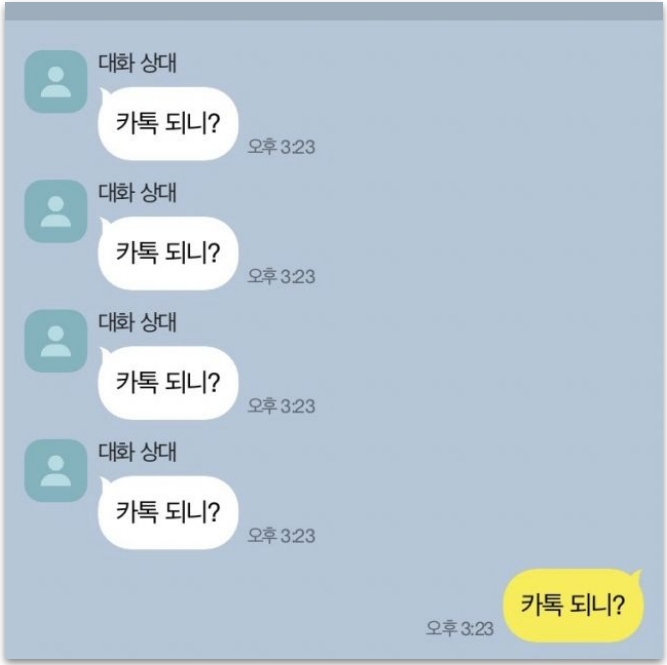
- **프로젝트 구조 예시**
  - **app/ 폴더**: 실제 애플리케이션 코드
    - routes, models
  - **instance/ 폴더**: DB, 설정 파일 등 실행환경별로 다른 것
  - (**templates/ 폴더**: HTML 템플릿)

```
project/
| app.py
|
├── app/
|   ├── __init__.py
|   ├── models.py
|   └── routes.py
|
└── instance/
    └── todos.db
```

HTTP vs WebSocket

WebSocket 소개

- 클라이언트 <-> 서버 **양방향 통신 가능**
- 한 번 연결하면 유지
- 실시간 서비스에 최적화
- ws:// 또는 wss:// (보안) 방식 활용



WebSocket이 필요한 이유

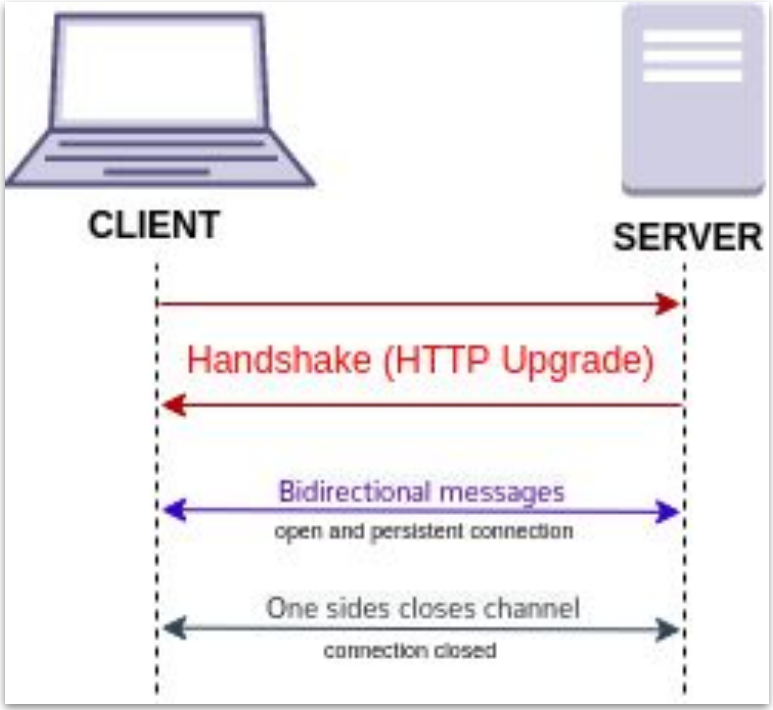
= 기존 HTTP 방식의 문제점

- 실시간성 부족
  - HTTP 요청-응답 방식은 클라이언트가 요청해야만 서버가 응답 가능
  - 실시간 알림, 채팅 등에는 비효율적
- 풀링(Pulling)과 긴 연결(Long Polling)의 한계
  - 풀링: 클라이언트가 일정 시간마다 서버에 요청 (과부하 문제)
  - 긴 연결: 클라이언트가 연결을 유지하며 서버 응답을 기다림 (비효율적)

HTTP vs WebSocket 비교

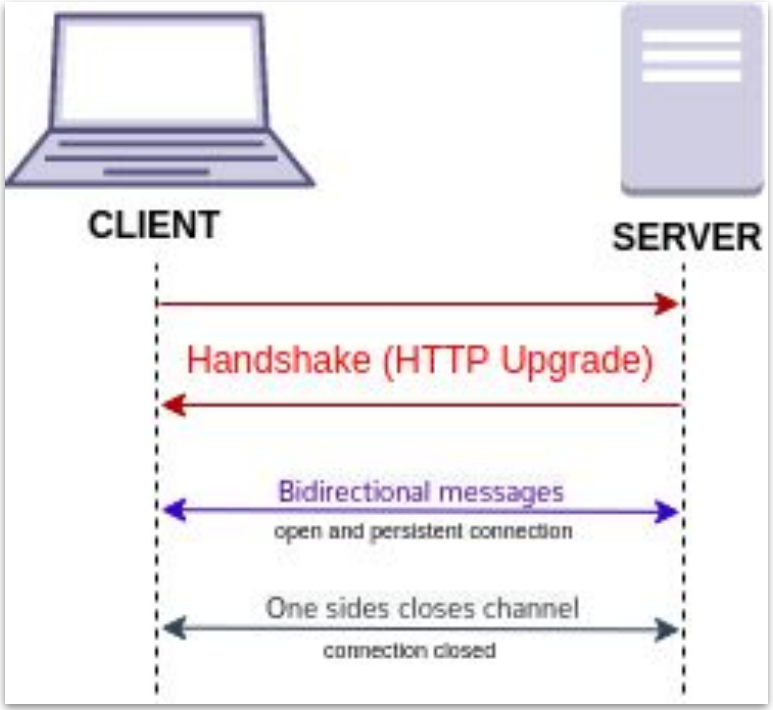
항목	HTTP	WebSocket
연결	요청마다 새로	한 번 연결 유지
통신	단방향 (Client -> Server)	양방향
사용 예시	게시판, 블로그	채팅, 알림, 실시간 주식

WebSocket 프로토콜 구조



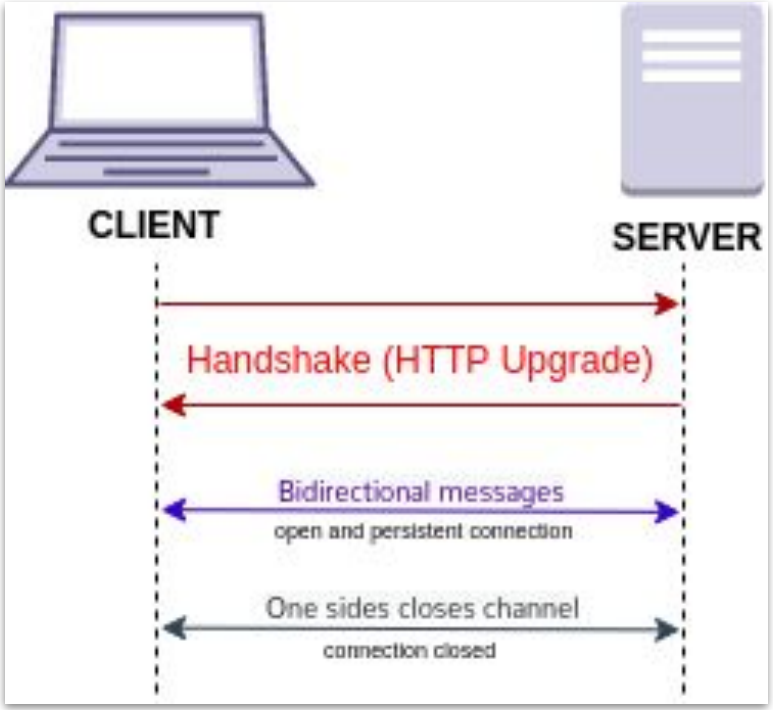
WebSocket 프로토콜 구조

- Handshake
  - HTTP 요청을 통해 WebSocket 연결 수립
    (이 때 Upgrade: websocket 헤더 사용)



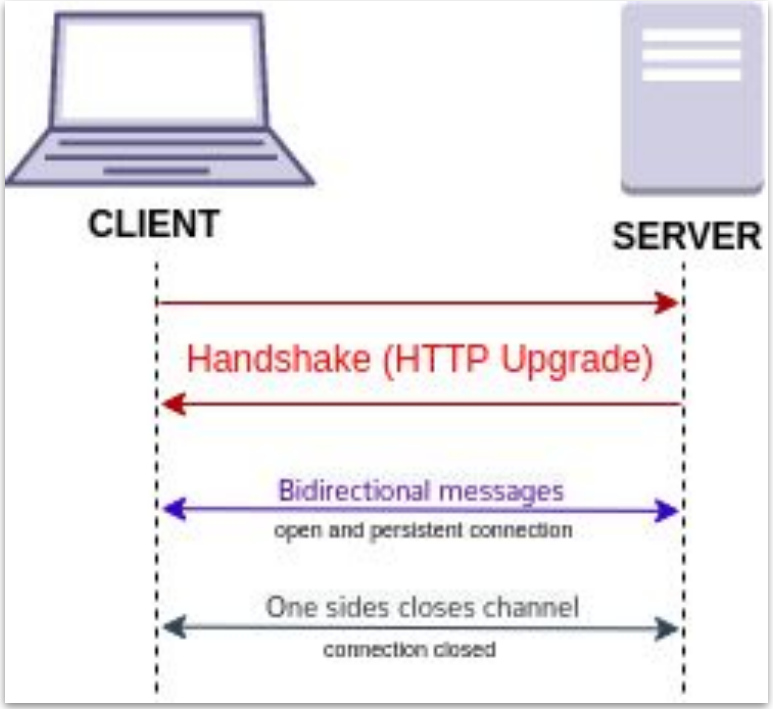
WebSocket 프로토콜 구조

- 메시지 송수신 (Message Exchange)
  - 텍스트 또는 바이너리 메시지 교환



WebSocket 프로토콜 구조

- 연결 종료 (Connection Close)
  - 클라이언트 또는 서버가 연결 종료



Flask-WebSocket 기본

[실습1] Flask에서 WebSocket 서버 구현하기

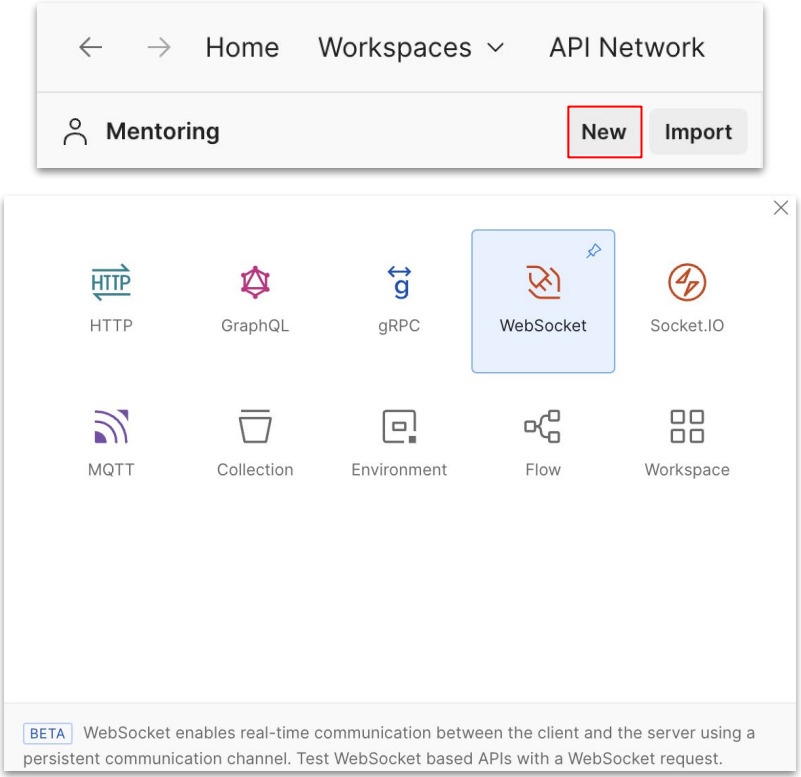
- pip install flask-sock

[실습 1] Flask에서 WebSocket 서버 구현하기

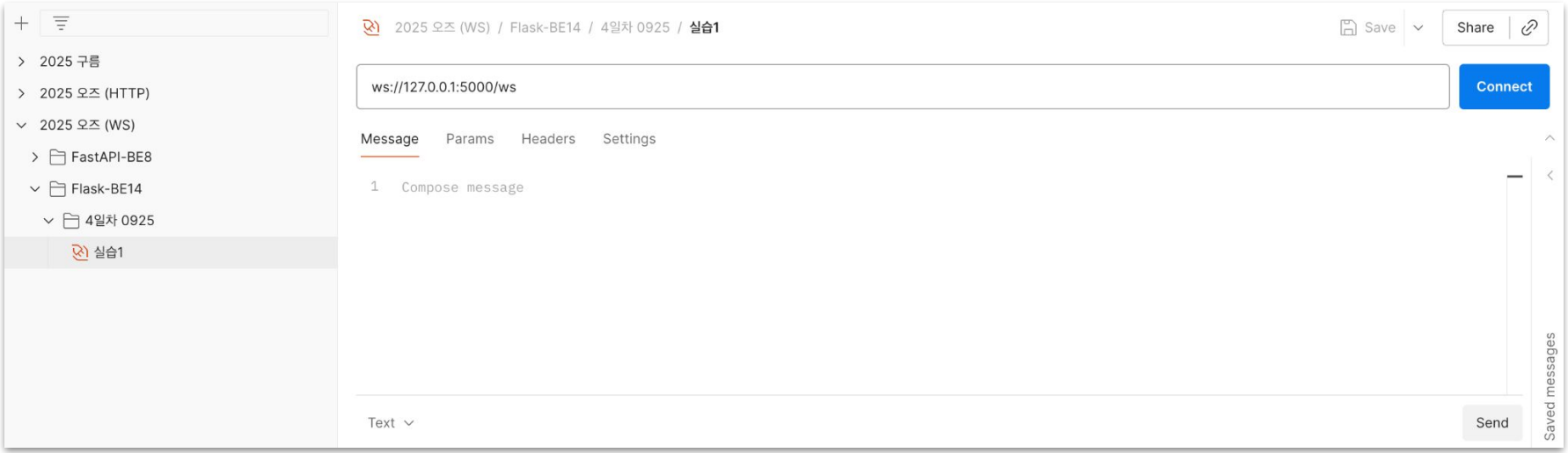
- URL: ws://127.0.0.1:5000/ws

```python
from flask import Flask
from flask_sock import Sock

app = Flask(__name__)
sock = Sock(app)

@sock.route('/ws')
def websocket(ws):
    while True:
        data = ws.receive()    # 클라이언트에서 메시지 받기
        if data is None:       # 연결 끊기 대비
            break
        print(f"받은 메시지: {data}")    # 서버 콘솔에 출력
        ws.send(f"Echo: {data}")         # 다시 클라이언트로 전송

if __name__ == "__main__":
    app.run(debug=True)
```

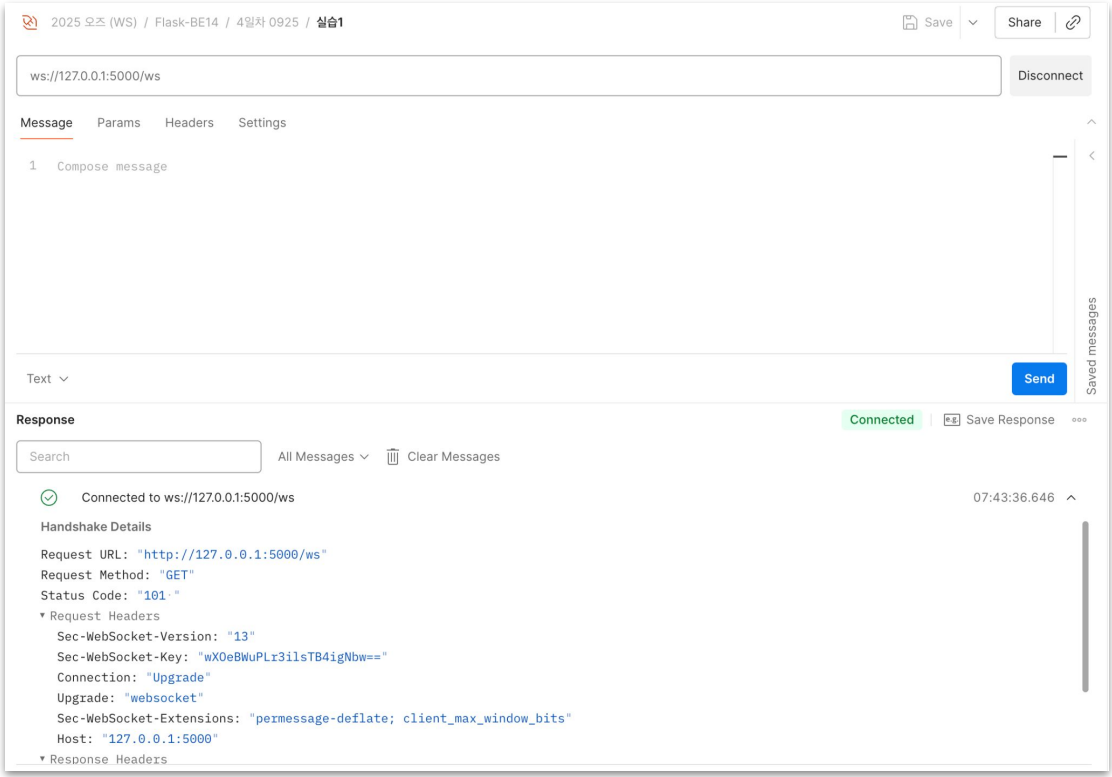
WebSocket 서버 구현 - Postman 설정



WebSocket 서버 구현 - Postman 설정



WebSocket 서버 구현 - Postman 설정



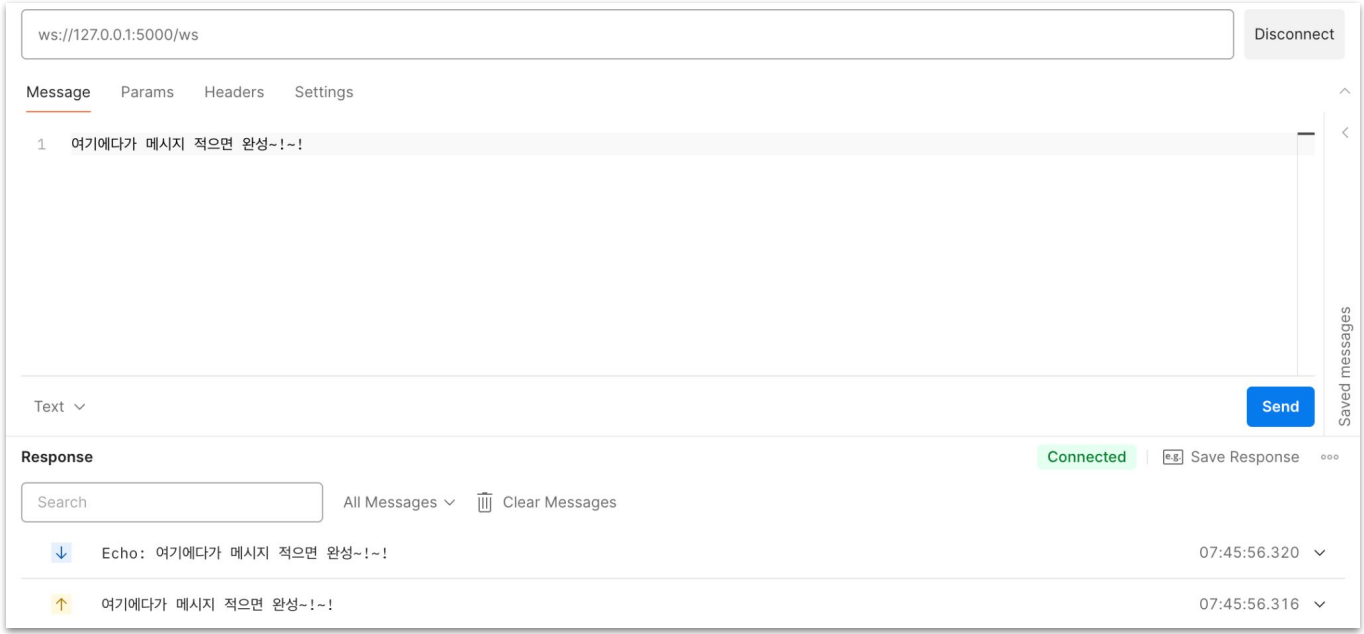
WebSocket 서버 구현 - Postman 설정

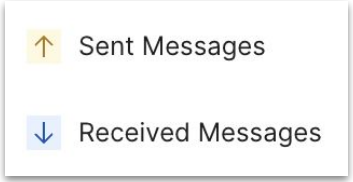
Handshake Details

```
Request URL: "http://127.0.0.1:5000/ws"
Request Method: "GET"
Status Code: "101 "
▾ Request Headers
  Sec-WebSocket-Version: "13"
  Sec-WebSocket-Key: "wXOeBWuPLr3ilsTB4igNbw=="
  Connection: "Upgrade"
  Upgrade: "websocket"
  Sec-WebSocket-Extensions: "permessage-deflate; client_max_window_bits"
  Host: "127.0.0.1:5000"
```

```
▾ Response Headers
  Upgrade: "WebSocket"
  Connection: "Upgrade"
  Sec-WebSocket-Accept: "5wxXCmO0PiXbQBORmkAERg27/Xs="
  Sec-WebSocket-Extensions: "permessage-deflate; client_max_window_bits=15"
```

WebSocket 서버 구현 - Postman 설정



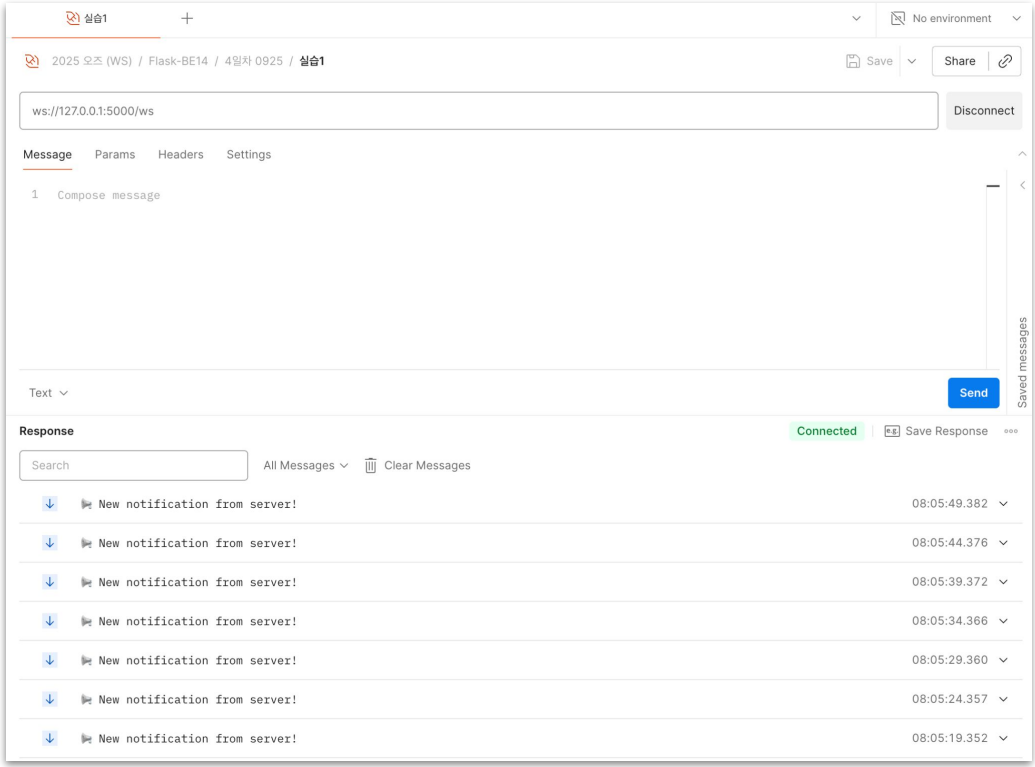


WebSocket 서버 구현 - 서버

```
rubykim@Rubyui-MacBookPro ~/Desktop/personal/OZ-coding-BE14-Flask/250925 (Day 4)/실습1
main* $ python app.py                                                    [7:36:14]
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 138-101-135
받은 메시지: 여기에다가 메시지 적으면 완성~!~!
```

실시간 프로그램 실습

[실습 2] 실시간 알림 서비스



[실습 2] 실시간 알림 서비스

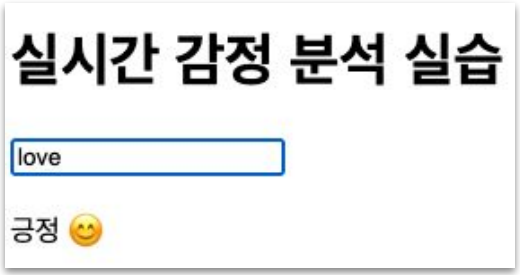
- 서버가 주기적으로 알림을 Push
- **클라이언트가 요청하지 않아도** 메시지를 주고받기
- 핵심
  - **백그라운드 작업 (Background Tasks)**: 클라이언트가 요청하지 않아도 자동으로 작업 진행
  - **스레드 (Thread)**: 서버가 동시에 여러 일을 할 수 있게 하는 단위

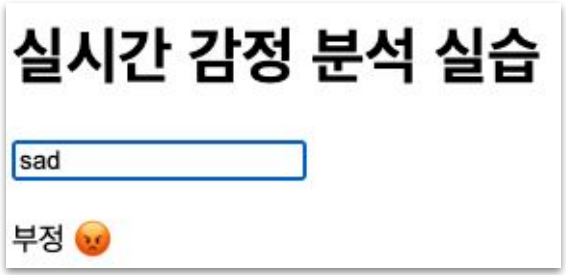
[실습 3] 타자기 효과

- 키보드를 입력 시 곧바로 “💬 누군가 입력 중…” 표시가 나옴
- 개념
  - 입력 시 -> typing 전송 -> 서버가 “💬 누군가 입력 중…” 응답
  - 입력 멈추고 1초 지나면 -> stop 전송 -> 표시 사라짐

[실습 4] 실시간 감정 분석

- 입력하면 서버가 텍스트 내용을 분석해 감정 판별
- 개념
  - 클라이언트: 입력 이벤트 발생 시 메시지 전송
  - 서버: 특정 키워드 기반 감정 분석 후 결과 전송
  - 클라이언트: 결과를 화면에 표시





QnA