"아조씨가 만든 GeoAI"

SQL을 모르는 '아조씨'도 만든, AI 통역사 기반 FOSS4G

이동훈 / geodata357@gmail.com

💬 "시장님, SQL을 배우셔야 할까요?"

강력한 FOSS4G, 하지만...

PostGIS 안에는 도시를 구할 수많은 데이터가 잠자고 있습니다.

하지만 이 힘은 `ST_DWithin`, `ST_Buffer` 같은 복잡한 SQL을 아는 전문가의 전유물이었습니다.

의사결정자의 딜레마

"당장 녹번동의 30년 넘은 건물이 궁금한데...
전문가의 보고서가 오려면 3일이 걸립니다."

우리의 질문

이 장벽을 허물고, 누구나 '말'로 이용할 수는 없을까?

💡 "Voice-to-Map" AI 대시보드 소개

개요

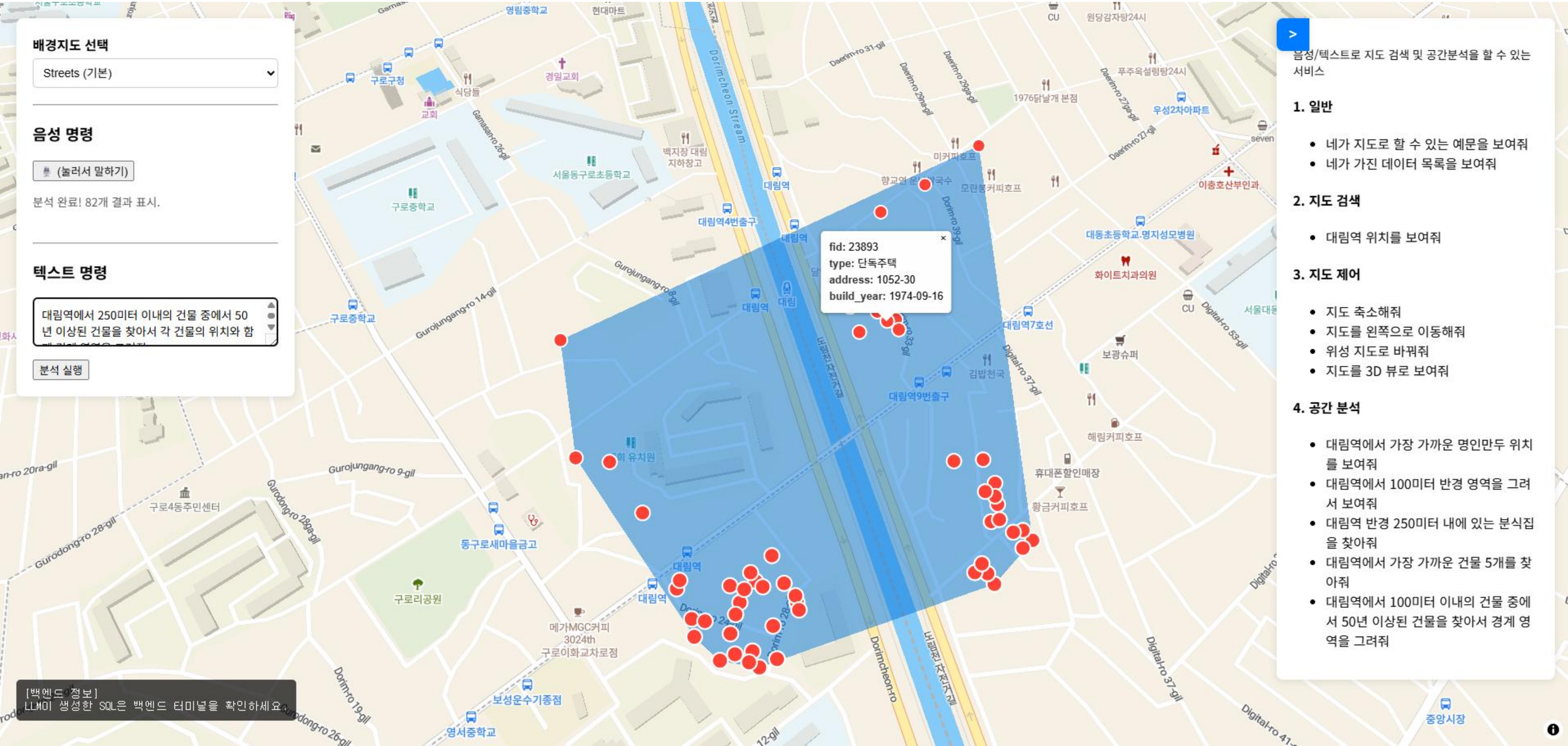
사용자의 자연어 명령을 LLM(AI)이 실시간으로 '의도'를 번역하고, 적절한 FOSS4G API를 호출하여 결과를 즉시 시각화하는 시스템

AI가 통역 가능한 5가지 명령

- ✓ **일반 질의:** "네가 가진 데이터 목록 보여줘"
- ✓ **지도 검색:** "녹번역 30년 넘은 건물 찾아줘"
- ✓ **공간 분석:** "녹번역 100미터 반경 영역 그려줘"
- ✓ **주제도 생성:** 지도 검색 및 공간 분석 결과를 표시
- ✓ **지도 제어:** "지도 축소해줘", "3D 뷰로 보여줘"

대림역에서 250미터 이내의 건물 중에서 50년 이상된 건물을 찾아서 각 건물의 위치와 함께 경계 영역을 그려줘

https://thlee33.github.io/ai_sql_map/



🏗️ 아키텍처: AI를 '통역사'이자 '라우터'로 활용하기

사용자 (Client)
"녹번역 500m 이내 건물은?"
"지도 축소해줘"

→

백엔드 (Backend)
FastAPI 서버 (Render.com)

→

AI 통역사 (Gemini)
사용자의 자연어를 3가지 JSON '계획'으로 번역

↓

Route 1: 일반 답변
AI가 직접 텍스트 생성
{"type": "GENERAL_ANSWER"}

Route 2: 지도 제어
MapLibre API 호출 명령
{"type": "CLIENT_COMMAND"}

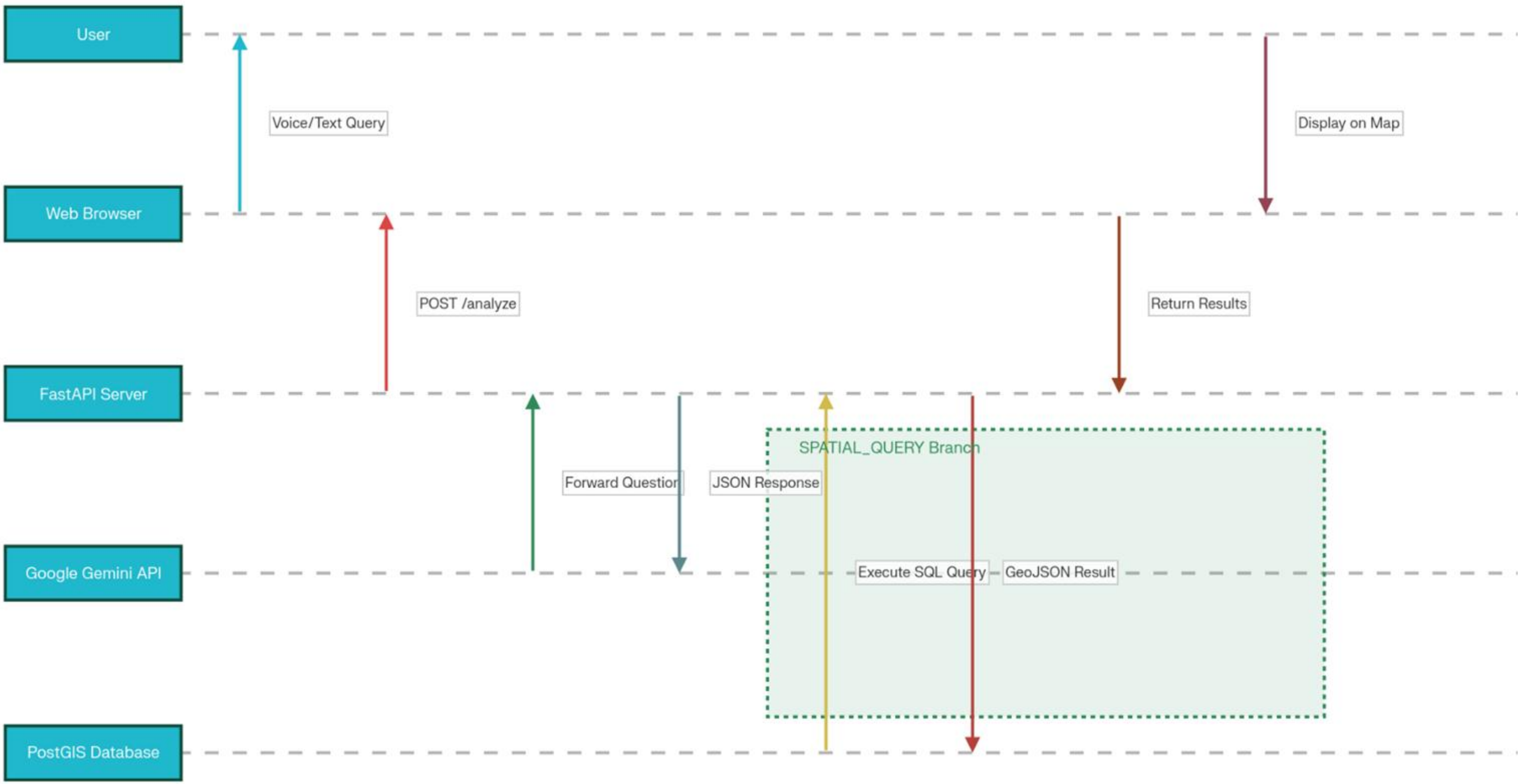
Route 3: 공간 분석
PostGIS SQL 생성
{"type": "SPATIAL_QUERY"}

↑ ↑ ↓

Client (MapLibre / UI)
AI 답변 또는 지도 제어 명령을 실행

Data Engine (PostGIS)
공간 SQL을 실행하고 GeoJSON 반환

시퀀스 : "500m 이내 건물" (공간 분석)



🧠 아키텍처의 "뇌": `main.py` 핵심 코드 (1)

LLM이 프롬프트를 분류

```
# main.py (system_instruction 부분)
system_instruction=f"""
당신은 최고의 GIS 전문가이자 범용 AI 비서입니다.
...질문의 의도를 3가지로 분류합니다.

[규칙]
1. **공간 분석/지도 표시 질문 (SPATIAL_QUERY)**:
  - "건물 찾아줘", "500미터 이내" 등
  - 반드시 PostGIS SQL 쿼리를 생성해야 합니다.
  - (예: `SELECT *, 'building' as data_type ...`)
  - 응답 형식: {{"type": "SPATIAL_QUERY", "content": "SELECT ..."}}

2. **클라이언트 제어 명령 (CLIENT_COMMAND)**:
  - "지도 확대/축소", "3D 뷰", "위성 지도로" 등
  - `content` 필드에 표준화된 명령어를 반환합니다.
  - (예: "3D로", "버드뷰" -> {{"type": "CLIENT_COMMAND", "content": "SET_PITCH_3D"}})
  - 응답 형식: {{"type": "CLIENT_COMMAND", "content": "..."}}

3. **일반/메타데이터 질문 (GENERAL_ANSWER)**:
  - "네가 가진 데이터 목록 보여줘" 등
  - SQL을 생성하면 안 됩니다.
  - 응답 형식: {{"type": "GENERAL_ANSWER", "content": "..."}}
"""
```

LLM이 SQL을 작성할 수 있도록 DB 스키마를 정의

```
# --- 3. DATABASE_SCHEMA  ---
DATABASE_SCHEMA = """
[데이터베이스 스키마]
1.  buildings (서울시 건물)
    - "fid" (INT): 고유 ID
    - "address" (TEXT): 주소 (예: '녹번동')
    - "build_year" (TEXT): 건축 연도 (예: '2022-01-19')
    - "name" (TEXT): 건물명
    - "type" (TEXT): 건물 주용도 (예: '단독주택', '공동주택')
    - geom (GEOMETRY(Point, 4326)): 위치 (EPSG:4326)

2.  subway_stations (서울시 지하철역)
    - "station_name" (TEXT): 역이름 (예: '녹번역')
    - geom (GEOMETRY(Point, 4326)): 위치 (EPSG:4326)

3.  restaurant (서울시 음식점)
    - "name" (TEXT): 가게 이름 (예: '부어치킨')
    - "type" (TEXT): 업종 (예: '한식', '중식', '분식', '일반음식점')
    - "address" (TEXT): 주소
    - "tel" (TEXT): 전화
    - geom (GEOMETRY(Point, 4326)): 위치 (EPSG:4326)
```

🧠 아키텍처의 "뇌": `main.py` 핵심 코드 (2)

FastAPI에서 분류별 결과를 프론트엔드에 제공

```python
@app.post("/analyze")
async def analyze_voice_query(query: VoiceQuery):

    llm_response = get_llm_response(query.text)

    response_type = llm_response.get("type")
    response_content = llm_response.get("content")

    if response_type == "SPATIAL_QUERY":
        if not response_content:
            return {"error": "LLM이 SQL을 생성하지

        cleaned_sql = response_content.strip().rstrip(';')
        geojson_result = execute_postgis_query(cleaned_sql)
        return geojson_result

    elif response_type == "CLIENT_COMMAND":
        return {"type": "CLIENT_COMMAND", "content": response_content}

    elif response_type == "GENERAL_ANSWER":
        return {"answer_text": response_content}

    else:
        error_message = llm_response.get("content", "알 수 없는 오류")
        return {"answer_text": f"오류가 발생했습니다: {error_message}"}
```

LLM이 생성한 SQL을 PostGIS에 쿼리하고, 그 결과인 GeoJSON을 전송

LLM이 생성한 쿼리를 GeoJSON으로 생성하는 함수

```python
def execute_postgis_query(sql_query: str):
    conn_info = f"host={DB_HOST} port={DB_PORT} dbname={DB_NAME} user={DB_USER} password={DB_PASS}"
    try:
        with psycopg.connect(conn_info) as conn:
            with conn.cursor() as cur:
                geojson_query = f"""
                WITH analysis_result AS (
                    {sql_query}
                )
                SELECT json_build_object(
                    'type', 'FeatureCollection',
                    'features', json_agg(
                        json_build_object(
                            'type', 'Feature',
                            'geometry', ST_AsGeoJSON(geom)::json,
                            'properties', row_to_json(analysis_result)::jsonb - 'geom'
                        )
                    )
                )
                FROM analysis_result
                WHERE geom IS NOT NULL;
                """
                print("--- 실행될 GeoJSON 쿼리 ---")
                print(geojson_query)
                cur.execute(geojson_query)
                result = cur.fetchone()
                if result and result[0]:
                    return result[0]
                else:
                    return {"type": "FeatureCollection", "features": []}
```

💪 아키텍처의 "손발": `index.html` 핵심 코드

```
// index.html (sendToBackend 함수 일부)
async function sendToBackend(text) {
...
    const result = await response.json();

 // 1. AI의 "지도 제어" 명령 라우팅
if (result.type === "CLIENT_COMMAND") {
    handleClientCommand(result.content);

// 2. AI의 "공간 분석" 결과 라우팅
    } else if (result.type === "FeatureCollection") {
        updateMap(result);

// 3. AI의 "일반 답변" 라우팅
    } else if (result.answer_text) {
        answerText.textContent = result.answer_text;
    ...
}

// AI 명령을 실제 MapLibre API로 실행
function handleClientCommand(command) {
    switch (command) {
        case 'ZOOM_OUT':
```

🚀 FOSS4G의 '자유도'를 해방시키다

기존 방식: 고정된 UI

UI가 허용하는 **정해진 질문**만 가능했습니다. (예: 정해진 반경 검색 버튼)

"3D 뷰로 바꿔줘"
→ (기능 버튼이 없으면 불가능)

AI 기반 방식: 대화형 UI

사용자의 **의도(Context)**에 맞는 **동적 질문**이 가능합니다
.

"3D 뷰로 바꿔줘"
→ (AI가 `SET_PITCH_3D` 명령을 즉시 생성)

AI는 '기능'을 호출하는 것이 아니라, PostGIS와 MapLibre의 '언어'를 구사합니다.

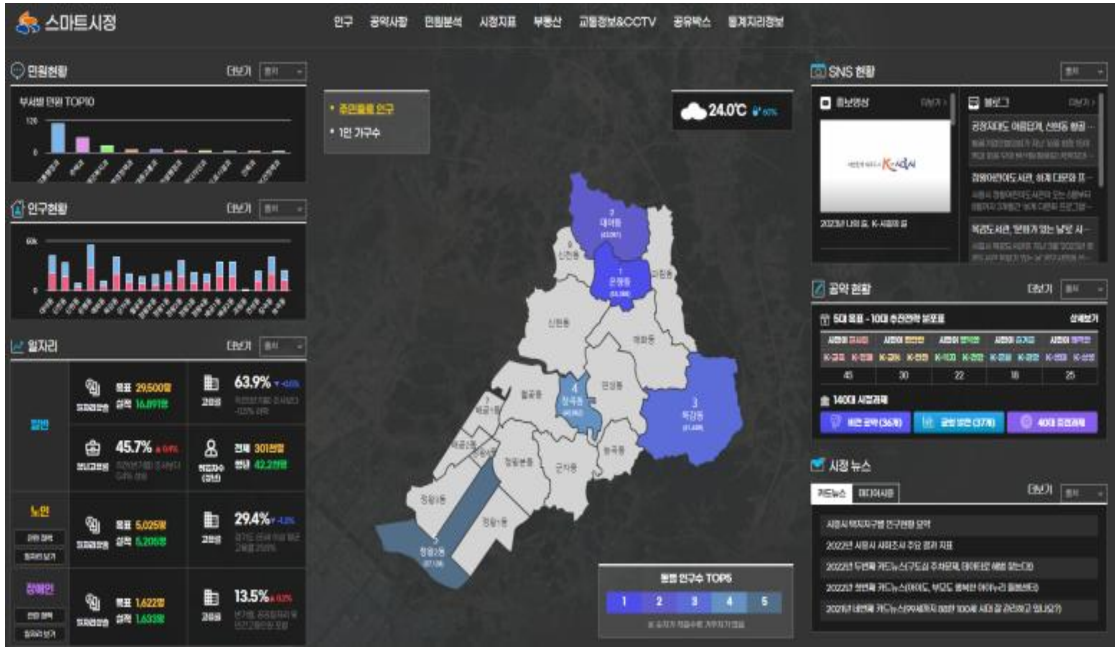
활용 분야 1: 정책 결정자 (시장님)

"SQL 없이, 보고서 없이, 즉각적인 인사이트"

복잡한 보고서나 전문가의 분석을 기다릴 필요 없이, '**골든 타임**' 내에 데이터를 기반으로 한 **즉각적인 정책 결정**이 가능합니다.

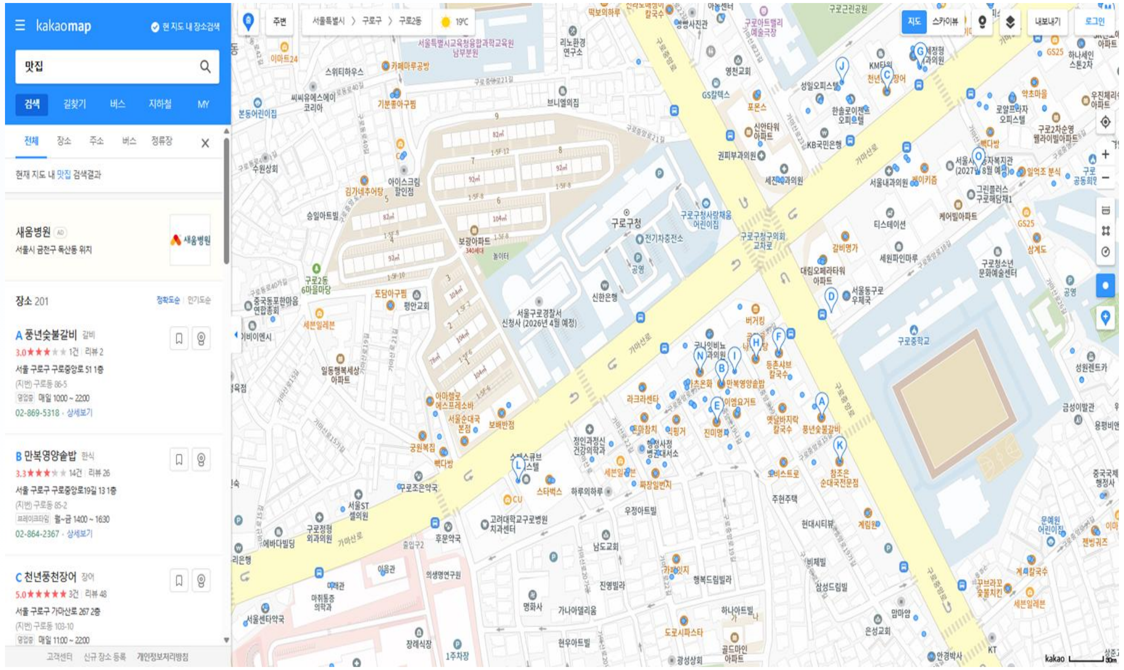
시나리오 예시

- ✓ "은평구에서 30년 넘은 노후 건물 현황을 보여줘."
- ✓ "범죄 발생 250m 반경 영역의 CCTV 설치 지점을 표시해."
- ✓ "새로 생길 공원 부지 1km 반경 내의 유동 인구는?"



시흥시 스마트 시정 지원 시스템

활용 분야 2: 일반 시민 (직장인, 학생)



"GIS를 모르는 모두를 위한 일상 속 공간 비서"

GIS 데이터를 전문가의 영역에서 **모두의 '일상 도구'**로 만듭니다. 데이터베이스 스키마(main.py)만 정의하면 **무한한 서비스로 확장**이 가능합니다.

시나리오 예시

- ✓ **(직장인)** "회사 근처 300m 이내, 8천 원 이하 점심 맛집 찾아줘."
- ✓ **(부동산)** "녹번역 500m 이내, 30평 이상, 10억 미만 아파트만 보여줘."
- ✓ **(학생)** "우리 동네에서 가장 가까운 공공 도서관은?"

🧑‍🔬 활용 분야 3: GIS 전문가 (개발자, 분석가)

"반복 노동은 AI에게, 전문가는 '진짜 분석'에 집중"

AI는 **"가장 똑똑한 GIS 인턴"**이자 **"가장 효율적인 API"** 역할을 수행하여, 전문가의 생산성을 극대화합니다.

시나리오 예시

- ✓ **(분석가)** "녹번동 건물 10년 단위 단계구분도 시각화해줘"
- ✓ **(개발자)** "녹번역에서 도보 15분 초과 영역 내의 30년 이상된 단독주택 중 서로 100M 내에 5개 이상 군집된 건물들만 찾는 SQL을 작성해줘
- ✓ **(모두)** "반복작업 또는 복잡한 SQL 작성은 AI에게 맡기고, 전문가는 결과가 맞는지 검토하고 활용하는 데 시간을 씁니다."

```sql
WITH facility_buffer AS (
  SELECT
    f.id,
    f.type,
    f.name,
    ST_Buffer(f.geom, 500) AS geom
  FROM facilities f
  WHERE f.type IN ('학교', '병원', '공장', '대피소')
),
pop_by_district AS (
  SELECT
    d.code AS district_code,
    SUM(CASE WHEN p.age BETWEEN 0 AND 14 THEN 1 ELSE 0 END) AS
pop_child,
    SUM(CASE WHEN p.age BETWEEN 15 AND 64 THEN 1 ELSE 0 END) AS
pop_adult,
    SUM(CASE WHEN p.age >= 65 THEN 1 ELSE 0 END) AS pop_senior,
    SUM(CASE WHEN p.sex = 'M' THEN 1 ELSE 0 END) AS pop_male,
    SUM(CASE WHEN p.sex = 'F' THEN 1 ELSE 0 END) AS pop_female
  FROM population p
  JOIN districts d ON ST_Contains(d.geom, p.geom)
  GROUP BY d.code
),
traffic_density AS (
  SELECT
    r.district_code,
    AVG(t.density) AS avg_density,
    AVG(t.avg_travel_time) AS avg_travel_time
  FROM traffic_data t
  JOIN roads r ON ST_Intersects(r.geom, t.geom)
  WHERE t.timestamp >= CURRENT_DATE - INTERVAL '1 DAY'
  GROUP BY r.district_code
),
risk_area AS (
  SELECT
    d.code,
    SUM(r.risk_score) AS total_risk,
    MAX(r.risk_score) AS max_risk
  FROM risk_points r
  JOIN districts d ON ST_Contains(d.geom, r.geom)
  GROUP BY d.code
  ORDER BY total_risk DESC
  LIMIT 10
),
land_use_summary AS (
  SELECT
    d.code AS district_code,
    COUNT(*) FILTER (WHERE l.type = '공업') AS industrial_count,
    COUNT(*) FILTER (WHERE l.type = '주거') AS residential_count,
    COUNT(*) FILTER (WHERE l.type = '상업') AS commercial_count,
    COUNT(*) FILTER (WHERE b.registered IS FALSE) AS
unregistered_building_count
  FROM land_parcels l
  JOIN buildings b ON ST_Intersects(l.geom, b.geom)
  JOIN districts d ON ST_Contains(d.geom, l.geom)
  GROUP BY d.code
),
facility_buffer_pop AS (
  SELECT
    fb.id AS facility_id,
    fb.type AS facility_type,
    fb.name AS facility_name,
    SUM(1) AS surrounding_population
  FROM facility_buffer fb
  JOIN population p ON ST_Within(p.geom, fb.geom)
  GROUP BY fb.id, fb.type, fb.name
)
SELECT
  d.code AS district_code,
  d.name AS district_name,
  p.pop_child,
  p.pop_adult,
  p.pop_senior,
  p.pop_male,
  p.pop_female,
  t.avg_density,
  t.avg_travel_time,
  l.industrial_count,
  l.residential_count,
  l.commercial_count,
  l.unregistered_building_count,
  r.total_risk,
  r.max_risk
FROM districts d
LEFT JOIN pop_by_district p ON d.code = p.district_code
LEFT JOIN traffic_density t ON d.code = t.district_code
LEFT JOIN land_use_summary l ON d.code = l.district_code
LEFT JOIN risk_area r ON d.code = r.code
ORDER BY r.total_risk DESC, t.avg_density DESC
LIMIT 20;
```

🌍 우리가 사랑하는 FOSS4G와 오픈소스

PostGIS

모든 공간 데이터를 저장하고 `ST_DWithin`, `ST_Buffer` 등 강력한 공간 함수를 실행하는 심장입니다.

MapLibre GL JS

GeoJSON 분석 결과와 지도 제어(`setPitch`)를 담당하는 오픈소스 렌더러입니다.

Python + FastAPI

AI와 DB를 연결하는 가볍고 빠른 Python 오픈소스 프레임워크입니다.

GitHub Pages & (Render)

프론트엔드와 백엔드/DB를 호스팅하는 인프라입니다.

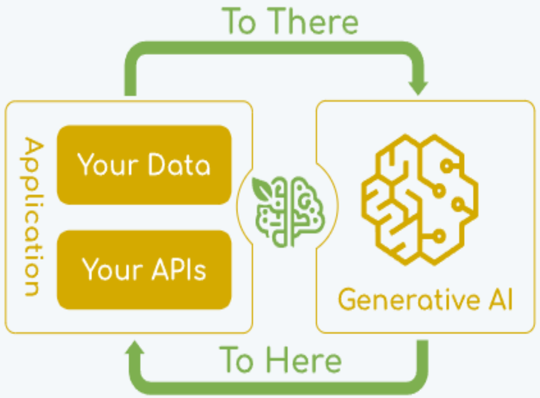
🌍 이미 이용하고 있다구요?!

스프링 AI https://spring.io/projects/spring-ai#overview

스프링 AI 1.0.3

개요 배우다 지원

Spring AI는 AI 엔지니어링을 위한 애플리케이션 프레임워크입니다. 그 목표는 이식성 및 모듈식 설계와 같은 AI 도메인 Spring 생태계 설계 원칙을 적용하고 POJO를 애플리케이션의 빌딩 블록으로 사용하여 AI 도메인에 사용하도록 홍보하는 것입니다.



Spring AI의 핵심은 AI 통합의 근본적인 과제인 엔터프라이즈 **데이터 및 API**를 **AI 모델**과 연결하는 것입니다.

Features

Spring AI provides the following features:

- Support for all major AI Model providers such as Anthropic, OpenAI, Microsoft, Amazon, Google, and Ollama. Supported model types include:
  - Chat Completion
  - Embedding
  - Text to Image
  - Audio Transcription
  - Text to Speech
  - Moderation
- Portable API support across AI providers for both synchronous and streaming API options are supported. Access to model-specific features is also available.
- Structured Outputs - Mapping of AI Model output to POJOs.
- Support for all major Vector Database providers such as *Apache Cassandra, Azure Vector Search, Chroma, Milvus, MongoDB Atlas, Neo4j, Oracle, PostgreSQL/PGVector, PineCone, Qdrant, Redis, and Weaviate.*
- Portable API across Vector Store providers, including a novel SQL-like metadata filter API.
- Tools/Function Calling - permits the model to request the execution of client-side tools and functions, thereby accessing necessary real-time information as required.
- Observability - Provides insights into AI-related operations.
- Document injection ETL framework for Data Engineering.
- AI Model Evaluation - Utilities to help evaluate generated content and protect against hallucinated response.
- ChatClient API - Fluent API for communicating with AI Chat Models, idiomatically similar to the WebClient and RestClient APIs.
- Advisors API - Encapsulates recurring Generative AI patterns, transforms data sent to and from Language Models (LLMs), and provides portability across various models and use cases.
- Support for Chat Conversation Memory and Retrieval Augmented Generation (RAG).
- Spring Boot Auto Configuration and Starters for all AI Models and Vector Stores - use the start.spring.io to select the Model or Vector-store of choice.

결론 및 제언

AI는 GIS 전문가를 대체할까요?

전문가는 더 강력한 무기를 갖게 됩니다.

AI가 FOSS4G의 마지막 장벽(SQL)을 허물고 있습니다. 아이디어와 데이터만 있다면,
SQL을 모르는 '아저씨'도, '지자체장'도, '일반 시민'도 누구나 강력한 PostGIS의 힘을 활용할 수 있습니다.

AI와 FOSS4G의 결합은 '공간 정보의 민주화'를 완성시킬 것입니다.

감사합니다.

"이제 여러분도 AI에게 GIS를 물어보세요!"

시연 URL: https://thlee33.github.io/ai_sql_map/

GitHub 소스: https://github.com/thlee33/ai_sql_map

Email: geodata357@gmail.com