Flask 게시판 프로젝트

Python · Flask · SQLite · Jinja2

간단한 게시판 웹 애플리케이션 구현 발표

사용 기술

- **Python 3**
- **Flask**
- **SQLite**
- **Jinja2**

4	3	1
파일	라우트	DB 테이블

목 차

01 기술 스택

Server Language

Python 3

이 프로젝트의 서버 언어입니다.
sqlite3, Flask 등 모든 백엔드 로직을
Python으로 작성합니다.

Web Framework

Flask

Python 기반 경량 웹 프레임워크입니다.
라우팅, 요청 처리, 템플릿 렌더링을
간결한 코드로 처리합니다.

Database

SQLite

파일 하나(board.db)로 동작하는 관계형 데이터베이스입니다.
별도 서버 설치가 필요 없습니다.

Template Engine

Jinja2

Flask 내장 HTML 템플릿 엔진입니다.
{{ 변수 }}, {% for %} 등의 문법으로
Python 데이터를 HTML에 삽입합니다.

02 폴더 구조

```
flask_board_project/

├── app.py
│
└── templates/
    ├── index.html
    ├── write.html
    └── detail.html
```

app.py

프로젝트의 핵심 파일

Flask 앱 생성, 라우팅, DB 연결, 데이터 조회·삽입 로직이 모두 여기에 있습니다.

index.html

게시글 목록 페이지

Jinja2 반복문으로 posts 데이터를 목록 형태로 렌더링합니다.

write.html

글 작성 페이지

제목·내용 입력 폼을 제공하고 POST 방식으로 데이터를 전송합니다.

detail.html

상세보기 페이지

선택한 게시글의 제목과 본문을 상세하게 표시합니다.

03 게시판 작동 흐름 — 전체 구조

브라우저 (Client)

① URL 입력 / 링크 클릭

⑥ HTML 수신
화면에 표시

Flask 서버

② 라우터 매칭
@app.route(...)

③ DB 쿼리 실행
cursor.execute()

⑤ render_template()
HTML 완성 후 반환

SQLite (DB)

④ 데이터 반환
fetchall() / fetchone()

03 게시판 작동 흐름 — 글 작성 상세 흐름

1 **GET /write**
/write 접속 (브라우저)

2 **라우터 수신**
write() 함수 실행 시작

3 **HTML 렌더**
write.html 반환

4 **사용자 입력**
제목·내용 작성 버튼 클릭

5 **POST /write**
폼 데이터 서버로 전송

6 **request.form**
title, content 값 추출

7 **INSERT INTO**
DB에 게시글 저장

8 **redirect('/')**
목록 페이지로 자동 이동

핵심 포인트

✓ GET → 빈 폼 표시

✓ POST → DB에 저장

✓ ? 바인딩으로 SQL Injection 방어

✓ redirect()로 중복 제출 방지

04 핵심 주요 기능 ① 게시글 목록 조회

GET /

DB에서 전체 게시글(id, title)을 최신순으로 불러와
목록 형태로 화면에 표시하는 기능입니다.

최신순 정렬

ORDER BY id DESC 쿼리로
최근 글이 위에 표시됩니다.

제목 클릭 링크

각 제목 클릭 시
/post/<id> 로 이동합니다.

글쓰기 버튼

/write 링크로
새 글 작성 페이지로 이동합니다.

```
<!-- index.html -->


<ul>
  {% for post in posts %}
  <li>
    <a href="/post/{{ post[0] }}">
      {{ post[1] }}
    </a>
  </li>
  {% endfor %}
</ul>
```

post[0] = id / post[1] = title (튜플 인덱스)

04 핵심 주요 기능 ② 게시글 작성

GET / POST /write

하나의 라우트가 GET·POST 두 방식을 모두 처리합니다.
GET이면 빈 폼을 표시하고, POST이면 DB에 저장합니다.

GET 요청

write.html 렌더링
빈 입력 폼을 표시합니다.

POST 요청

request.form으로 데이터 수신
DB에 INSERT 후 리다이렉트합니다.

SQL Injection 방어

VALUES (?, ?) 형식으로
입력값을 안전하게 처리합니다.

```
<!-- write.html -->

<form method="POST">
  제목:
  <input type="text"
         name="title">

  내용:
  <textarea name="content"
    rows="5" cols="30">
  </textarea>

  <button type="submit">
    작성
  </button>
</form>
```

name 속성값 = request.form['키'] 로 Flask에서 수신

04 핵심 주요 기능 ③ 게시글 상세보기

GET /post/<int:id>

URL의 <int:id> 값으로 DB에서 해당 게시글을 조회해
제목과 본문을 화면에 상세히 표시합니다.

URL 파라미터 변환

<int:id>: URL 숫자를
자동으로 정수(int)로 변환합니다.

단건 조회

fetchone()으로
해당 글 하나만 가져옵니다.

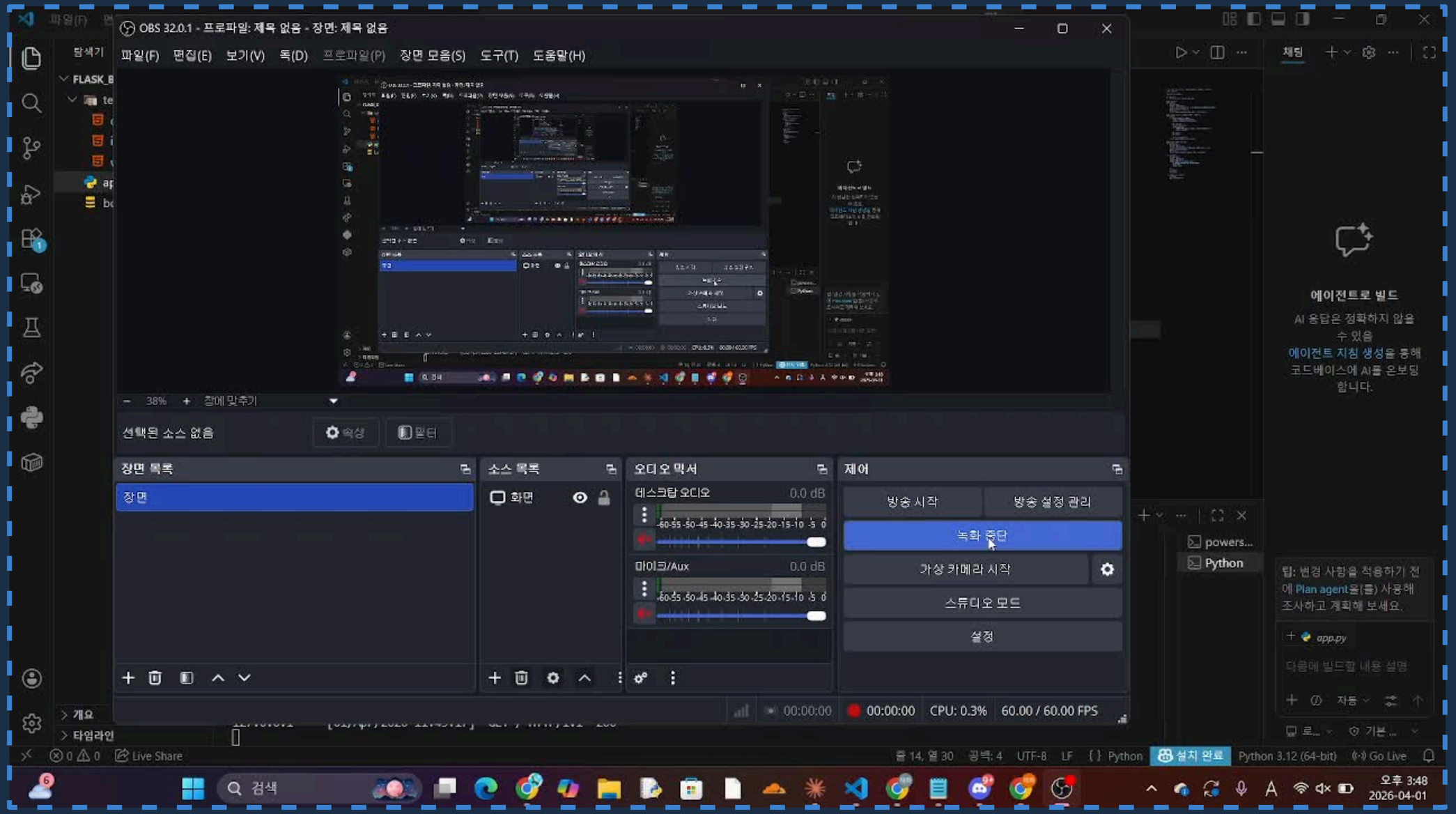
목록으로 돌아가기

<a href="/">으로
목록 페이지로 이동합니다.

```
<!-- detail.html -->


<h1>{{ post[1] }}</h1>


<p>{{ post[2] }}</p>


<a href="/">목록으로</a>
```

```
SELECT * 결과 튜플 구조

post[0] = id     post[1] = title     post[2] = content
```

05 시연 영상


OBS 32.0.1 - 프로파일: 제목 없음 - 장면: 제목 없음
파일(F) 편집(E) 보기(V) 독(D) 프로파일(P) 장면 모음(S) 도구(T) 도움말(H)
선택된 소스 없음
속성
필터
장면 목록
장면
소스 목록
화면
오디오 믹서
데스크탑 오디오 0.0 dB
마이크/Aux 0.0 dB
제어
방송 시작
방송 설정 관리
녹화 중단
가상 카메라 시작
스튜디오 모드
설정
00:00:00
00:00:00
CPU: 0.3%
60.00 / 60.00 FPS
에이전트로 빌드
AI 응답은 정확하지 않을 수 있음
에이전트 지침 생성을 통해 코드베이스에 AI를 온보딩합니다.
팁: 변경 사항을 적용하기 전에 Plan agent을(를) 사용해 조사하고 계획해 보세요.
app.py
다음에 빌드할 내용 설명
Live Share
Python
설치 완료
Python 3.12 (64-bit)
Go Live
검색
오후 3:48
2026-04-01

06 HTML 템플릿 — index.html (게시글 목록)

```
<!DOCTYPE html>
<html>
<head>
  <title>게시판</title>
</head>
<body>
  <h1>게시판</h1>
  <a href="/write">글쓰기</a>

  <ul>
    {% for post in posts %}
    <li>
      <a href="/post/{{ post[0] }}">
        {{ post[1] }}
      </a>
    </li>
    {% endfor %}
  </ul>
</body></html>
```

{% for post in posts %}

Flask가 전달한 posts 리스트를 순서대로 반복 출력합니다.
{% endfor %}로 반복을 종료합니다.

{{ post[0] }} / {{ post[1] }}

post는 DB 결과 튜플입니다.
[0] = id (링크 URL), [1] = title (텍스트)
로 각각 사용됩니다.

글쓰기

순수 HTML 링크 하나로
글쓰기 페이지로 이동합니다.
Jinja2 변수 없이 사용합니다.

06 HTML 템플릿 — write.html (글 작성)

```
<!DOCTYPE html>
<html>
<head>
  <title>글쓰기</title>
</head>
<body>
  <h1>글쓰기</h1>
  <form method="POST">
    제목:
    <input type="text" name="title">
    내용:
    <textarea name="content"
      rows="5" cols="30">
    </textarea>
    <button type="submit">
      작성
    </button>
  </form>
</body></html>
```

method="POST"

폼 제출 방식을 POST로 지정합니다.
GET은 데이터가 URL에 노출되므로
입력 데이터는 항상 POST를 사용합니다.

name="title" / name="content"

name 속성값이 Flask에서
request.form["title"] 처럼
키(key)로 사용됩니다.

action 속성 없음

action을 생략하면 현재 URL인
/write 로 POST 전송됩니다.
같은 라우트 함수가 처리합니다.

06 HTML 템플릿 — detail.html (상세보기)

```
<!DOCTYPE html>
<html>
<head>
  <title>상세보기</title>
</head>
<body>
  <h1>
    {{ post[1] }}
  </h1>

  <p>
    {{ post[2] }}
  </p>

  <a href="/">목록으로</a>
</body></html>
```

{{ post[1] }} — 제목

DB의 SELECT * 결과 튜플에서
인덱스 1이 title 컬럼입니다.
<h1> 태그로 큰 제목으로 표시합니다.

{{ post[2] }} — 본문

인덱스 2가 content 컬럼입니다.
<p> 태그로 본문 단락으로 표시합니다.
Jinja2가 HTML 특수문자를 자동 처리합니다.

목록으로

href="/"로 루트(목록) 페이지로 이동합니다.
Jinja2 변수 없이 순수 HTML만 사용합니다.

07 Flask 소스코드 — import & 앱 생성

```
from flask import Flask,
    render_template,
    request, redirect
import sqlite3


app = Flask(__name__)
```

Flask — Flask 클래스

웹 앱 인스턴스를 만드는 클래스입니다.
app = Flask(...)로 앱 본체를 생성합니다.

Flask Flask 클래스

웹 앱 인스턴스를 만드는 클래스입니다.
app = Flask(...)로 앱 본체를 생성합니다.

render_template render_template

templates/ 폴더의 HTML을 찾아
변수를 주입하고 완성된 HTML로 반환합니다.

request request

브라우저 요청 정보 객체입니다.
request.form으로 POST 데이터를 꺼냅니다.

redirect redirect

지정 URL로 브라우저를 이동시킵니다.
글 저장 후 목록 페이지로 보낼 때 씁니다.

07 Flask 소스코드 — import 상세 & 앱 생성

```
from flask import Flask, render_template, request, redirect

import sqlite3



app = Flask(__name__)
```

Flask

Flask 클래스
앱 인스턴스를 만드는 핵심 클래스입니다.

render_template

HTML 렌더러
templates/ 폴더 HTML을 찾아 변수를 주입합니다.

request

요청 객체
request.form 으로 POST 데이터를 꺼냅니다.

redirect

리다이렉트
지정 URL로 브라우저를 이동시킵니다.

sqlite3

DB 라이브러리
Python 내장 모듈로 pip 설치 불필요합니다.

Flask(__name__)

앱 생성
__name__으로 현재 파일 경로를 기준으로 설정합니다.

07 Flask 소스코드 — get_db() DB 연결 함수

```
def get_db():

    return sqlite3.connect("board.db")
```

sqlite3.connect()

"board.db" 파일에 연결합니다.
파일이 없으면 SQLite가 자동으로 생성하고,
이미 있으면 기존 데이터를 그대로 유지합니다.

return 으로 반환

연결 객체(conn)를 반환합니다.
라우트마다 get_db()를 호출하면
항상 신선한 DB 연결을 사용할 수 있습니다.

매번 연결 & 닫기 패턴

각 라우트: get_db() → 쿼리 → db.close()
단순한 구조로 소규모 프로젝트에 충분합니다.
실제 서비스에서는 커넥션 풀 방식을 사용합니다.

07 Flask 소스코드 — index() 게시글 목록 라우트

```
@app.route("/")
def index():
    db = get_db()
    cursor = db.cursor()
    cursor.execute(
        "SELECT id, title
         FROM posts
         ORDER BY id DESC"
    )
    posts = cursor.fetchall()
    db.close()
    return render_template(
        "index.html", posts=posts)
```

1행 · @app.route("/")

URL "/"로 들어오는 GET 요청을
이 함수가 처리하도록 Flask에 등록합니다.

2행 · def index():

라우트에 연결된 뷰 함수입니다.
요청이 오면 Flask가 자동으로 호출합니다.

3-4행 · get_db() / cursor()

DB 연결 객체를 얻고,
cursor로 SQL 명령 실행 준비를 합니다.

5-9행 · SELECT id, title … ORDER BY id DESC

id와 title만 선택적으로 가져옵니다.
ORDER BY id DESC = 최신 글이 먼저 표시됩니다.

10행 · cursor.fetchall()

쿼리 결과 전체를
리스트(튜플 목록) 형태로 반환합니다.

12-13행 · render_template()

"index.html"을 찾아 렌더링합니다.
posts=posts 로 DB 데이터를 HTML에 전달합니다.

07 Flask 소스코드 — write() ① 라우트 등록 & GET 처리

```python
@app.route("/write", methods=["GET","POST"])
def write():
    if request.method == "POST":
        title   = request.form["title"]
        content = request.form["content"]
        db = get_db()
        cursor = db.cursor()
        cursor.execute(
            "INSERT INTO posts (title, content)
             VALUES (?, ?)",
            (title, content)
        )
        db.commit()
        db.close()
        return redirect("/")
    return render_template("write.html")
```

methods=["GET", "POST"]

기본적으로 라우트는 GET만 허용합니다.
methods에 "POST"를 추가해야
폼 제출(POST)도 이 함수가 받을 수 있습니다.

def write() — 뷰 함수

하나의 함수가 GET·POST 두 경우를
모두 처리합니다.
if 분기로 로직을 구분합니다.

render_template("write.html") ← GET

이 줄은 if 블록 바깥에 있습니다.
GET 요청일 때만 실행되어
빈 write.html 폼을 반환합니다.

07 Flask 소스코드 — write() ② POST 처리 (데이터 저장)

```python
@app.route("/write", methods=["GET","POST"])
def write():
    if request.method == "POST":
        title   = request.form["title"]
        content = request.form["content"]
        db = get_db()
        cursor = db.cursor()
        cursor.execute(
            "INSERT INTO posts (title, content)
             VALUES (?, ?)",
            (title, content)
        )
        db.commit()
        db.close()
        return redirect("/")
    return render_template("write.html")
```

request.form["title"]

HTML 폼의 name 속성값을 키로 씁니다.
브라우저가 전송한 POST 데이터를
딕셔너리처럼 꺼냅니다.

VALUES (?, ?) — 바인딩 파라미터

? 는 플레이스홀더입니다.
뒤의 튜플로 ?를 순서대로 채우며
SQL Injection을 완전히 방어합니다.

db.commit() / redirect("/")

commit()으로 INSERT를 DB에 확정 저장합니다.
redirect로 목록 페이지로 이동시켜
새로고침 시 중복 제출을 방지합니다.

07 Flask 소스코드 — detail() 게시글 상세보기 라우트

```python
@app.route("/post/<int:id>")
def detail(id):
    db = get_db()
    cursor = db.cursor()
    cursor.execute(
        "SELECT * FROM posts WHERE id=?",
        (id,)
    )
    post = cursor.fetchone()
    db.close()
    return render_template(
        "detail.html", post=post)
```

/post/<int:id> URL 컨버터

"<int:id>" 부분이 URL의 숫자를
Python 정수(int)로 자동 변환해
함수 인자 id로 전달합니다.

WHERE id=? / (id,) 이유

id 값으로 특정 게시글만 조회합니다.
(id,) 처럼 쉼표가 있는 이유는
1개짜리 튜플을 만들기 위해서입니다.

fetchone() vs fetchall()

fetchall(): 전체 결과를 리스트로 반환합니다.

07 Flask 소스코드 — init_db() DB 초기화

```
def init_db():
    db = get_db()
    cursor = db.cursor()
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS posts (
        id      INTEGER PRIMARY KEY AUTOINCREMENT,
        title   TEXT,
        content TEXT
    )""")
    db.commit()
    db.close()
```

CREATE TABLE IF NOT EXISTS

테이블이 이미 있으면 건너뜁니다.
앱을 다시 실행해도 오류 없이
안전하게 초기화됩니다.

PRIMARY KEY AUTOINCREMENT

글을 등록할 때 id를 직접 지정 불필요합니다.
SQLite가 1, 2, 3... 자동으로 부여합니다.

TEXT 타입

SQLite의 TEXT는 가변 길이 문자열입니다.
title, content 모두 길이 제한 없이 저장합니다.

07 Flask 소스코드 — if __name__ & app.run()

```
if __name__ == "__main__":

    init_db()

    app.run(debug=True)
```

if __name__ == "__main__"

Python 파일을 직접 실행(python app.py)할 때만
아래 코드가 동작합니다.

다른 파일에서 import app 으로 불러올 때는
이 블록이 실행되지 않아 의도치 않은
자동 실행을 막을 수 있습니다.

init_db()

앱 시작 시 DB 테이블이 없으면
자동으로 생성합니다.
IF NOT EXISTS로 이미 있으면
건너뜁니다.

app.run(debug=True)

Flask 개발 서버를 시작합니다.

debug=True 효과:
· 코드 수정 시 자동 재시작
· 오류 상세 정보 표시
· 배포 시 반드시 False로!

08 데이터베이스 — SQLite & posts 테이블

```
CREATE TABLE IF NOT EXISTS posts (
    id      INTEGER PRIMARY KEY AUTOINCREMENT,
    title   TEXT,
    content TEXT
)
```

board.db → posts 테이블

id	title	content
1	Flask 소개	Flask는 Python으로 만든 경량 웹 프레임워크입니다.
2	SQLite란?	파일 기반의 관계형 데이터베이스입니다.
3	Jinja2 문법	{{ 변수 }} 형식으로 Python 데이터를 삽입합니다.

id (INTEGER)

PRIMARY KEY AUTOINCREMENT

게시글 고유 번호입니다.
글 등록 시 자동으로 1씩 증가하며
직접 입력할 필요가 없습니다.

title (TEXT)

가변 길이 문자열

게시글 제목을 저장합니다.
TEXT 타입이므로 길이 제한 없이 저장합니다.

content (TEXT)

가변 길이 문자열

게시글 본문 내용입니다.
줄바꿈 포함 긴 텍스트도 문제없이 저장합니다.

app.py — 전체 코드 한눈에 보기

```
from flask import Flask, render_template, request, redirect
import sqlite3
app = Flask(__name__)

def get_db():
    return sqlite3.connect("board.db")

@app.route("/")
def index():
    db=get_db(); cursor=db.cursor()
    cursor.execute("SELECT id,title FROM posts ORDER BY id DESC")
    posts=cursor.fetchall(); db.close()
    return render_template("index.html", posts=posts)

@app.route("/write", methods=["GET","POST"])
def write():
    if request.method == "POST":
        title=request.form["title"]; content=request.form["content"]
        db=get_db(); cursor=db.cursor()
        cursor.execute("INSERT INTO posts(title,content) VALUES(?,?)",(title,content))
        db.commit(); db.close()
        return redirect("/")
    return render_template("write.html")

@app.route("/post/<int:id>")
def detail(id):
    db=get_db(); cursor=db.cursor()
    cursor.execute("SELECT * FROM posts WHERE id=?",(id,))
    post=cursor.fetchone(); db.close()
    return render_template("detail.html", post=post)
```

추가적인 이야기....

질문

감사합니다

Flask 라우팅 & 요청 처리

SQLite DB 연동 (CRUD)

Jinja2 템플릿 렌더링

SQL Injection 방어 (? 바인딩)