문제 2: FastAPI + Jinja2 템플릿 활용 웹 애플리케이션

📋 문제 개요

Jinja2 템플릿 엔진을 사용하여 **할 일 관리(Todo List)** 웹 애플리케이션을 구현하세요. 사용자가 웹 브라우저에서 직접 할 일을 추가, 조회, 완료 처리할 수 있는 기능을 제공해야 합니다.

🎯 학습 목표

- Jinja2 템플릿 엔진 사용법 습득
- HTML 폼과 FastAPI 연동 방법 학습
- 템플릿 변수 전달 및 활용
- 폼 데이터 처리 (Form 클래스 사용)
- 리다이렉트 응답 처리
- 템플릿에서 조건문과 반복문 활용

📊 데이터 구조

할 일(Todo) 정보

- **id**: 정수형, 자동 증가하는 고유 식별자
- **title**: 문자열, 할 일 제목 (필수, 1-100자)
- **description**: 문자열, 할 일 상세 설명 (선택사항, 최대 500자)
- **completed**: 불린형, 완료 여부 (기본값: False)
- **created_at**: 문자열, 생성 시간 (YYYY-MM-DD HH:MM:SS 형식)
- **priority**: 문자열, 우선순위 ("높음", "보통", "낮음" 중 하나)

🛠️ 구현해야 할 기능

1. 메인 페이지 (할 일 목록 + 추가 폼)

- **경로**: `GET /`
- **템플릿**: `index.html`
- **기능**:
  - 등록된 모든 할 일 목록 표시
  - 새 할 일 추가를 위한 폼 제공
  - 완료된 할 일과 미완료 할 일 구분 표시
  - 전체 통계 정보 표시 (전체/완료/미완료 개수)

2. 할 일 추가 처리

- **경로**: `POST /add`
- **기능**: 폼에서 전송된 새 할 일 정보를 저장
- **폼 필드**:
  - `title` (필수): 할 일 제목
  - `description` (선택): 상세 설명
  - `priority` (필수): 우선순위 선택
- **처리 후**: 메인 페이지로 리다이렉트

3. 할 일 완료/미완료 토글

- **경로**: `POST /toggle/{todo_id}`
- **기능**: 특정 할 일의 완료 상태를 변경
- **처리 후**: 메인 페이지로 리다이렉트

4. 할 일 삭제

- **경로**: `POST /delete/{todo_id}`
- **기능**: 특정 할 일을 완전히 삭제
- **처리 후**: 메인 페이지로 리다이렉트

5. 통계 페이지 (선택사항)

- **경로**: `GET /stats`
- **템플릿**: `stats.html`
- **기능**: 우선순위별, 완료 상태별 통계 정보 표시

📝 구현 요구사항

1. 필수 라이브러리 설치

bash

```
pip install fastapi uvicorn jinja2 python-multipart
```

2. 프로젝트 구조

```
todo_project/
├── main.py                 # FastAPI 앱 메인 파일
├── templates/              # 템플릿 폴더
│   ├── index.html          # 메인 페이지 템플릿
│   └── stats.html          # 통계 페이지 템플릿 (선택사항)
└── static/                 # 정적 파일 폴더 (선택사항)
    └── style.css           # CSS 스타일
```

3. 템플릿 설정

python

```python
from fastapi.templating import Jinja2Templates
templates = Jinja2Templates(directory="templates")
```

4. 폼 데이터 처리

python

```python
from fastapi import Form

@app.post("/add")
async def add_todo(
    title: str = Form(...),
    description: str = Form(""),
    priority: str = Form(...)
):
    # 폼 데이터 처리 로직
```

🔧 Jinja2 템플릿 문법 가이드

1. 기본 문법 구조

```html
<!-- 변수 출력 -->
{{ variable_name }}
{{ todo.title }}
{{ user.name }}

<!-- 제어 구조 -->
{% if condition %}
    <!-- 조건이 참일 때 실행 -->
{% elif other_condition %}
    <!-- 다른 조건이 참일 때 실행 -->
{% else %}
    <!-- 모든 조건이 거짓일 때 실행 -->
{% endif %}

<!-- 반복문 -->
{% for item in items %}
    <!-- 각 항목에 대해 실행 -->
{% endfor %}

<!-- 주석 -->
{# 이것은 주석입니다 #}
```

2. 조건문 활용 패턴

```html
<!-- 리스트가 비어있는지 확인 -->
{% if todos %}
    <!-- 할 일이 있을 때 -->
    {% for todo in todos %}
        <div>{{ todo.title }}</div>
    {% endfor %}
{% else %}
    <!-- 할 일이 없을 때 -->
    <p>등록된 할 일이 없습니다.</p>
{% endif %}

<!-- 조건부 클래스 적용 -->
<div class="todo-item {% if todo.completed %}completed{% endif %}">
    {{ todo.title }}
</div>

<!-- 복잡한 조건 -->
{% if todo.priority == "높음" and not todo.completed %}
    <span class="urgent">긴급!</span>
{% endif %}
```

3. 반복문 고급 활용

```html
<!-- 반복문에서 인덱스 사용 -->
{% for todo in todos %}
    <div>{{ loop.index }}. {{ todo.title }}</div>
    <!-- loop.index: 1부터 시작하는 번호 -->
    <!-- loop.index0: 0부터 시작하는 번호 -->
    <!-- loop.first: 첫 번째 항목이면 True -->
    <!-- loop.last: 마지막 항목이면 True -->
{% endfor %}

<!-- 빈 리스트 처리 -->
{% for todo in todos %}
    <div>{{ todo.title }}</div>
{% else %}
    <p>할 일이 없습니다.</p>
{% endfor %}
```

4. 필터 사용법

html

```
<!-- 문자열 필터 -->
{{ todo.title|upper }}           <!-- 대문자로 변환 -->
{{ todo.description|truncate(50) }} <!-- 50자로 자르기 -->
{{ todo.created_at|default("날짜 없음") }} <!-- 기본값 설정 -->

<!-- 리스트 필터 -->
{{ todos|length }}               <!-- 리스트 길이 -->
{{ todos|first }}                <!-- 첫 번째 항목 -->
{{ todos|last }}                 <!-- 마지막 항목 -->
```

💡 템플릿 활용 팁

1. 템플릿 변수 전달 시 주의사항

python

```
# ✅ 올바른 방법
return templates.TemplateResponse("index.html", {
    "request": request,  # 반드시 포함!
    "todos": todos,
    "total_count": len(todos)
})

# ❌ 잘못된 방법 - request 누락
return templates.TemplateResponse("index.html", {
    "todos": todos
})
```

2. 폼 데이터 처리 팁

python

```
# 선택적 필드 처리
@app.post("/add")
async def add_todo(
    title: str = Form(...),                    # 필수 필드
    description: str = Form(""),               # 선택 필드 (기본값 빈 문자열)
    priority: str = Form("보통")               # 선택 필드 (기본값 설정)
):
    # 빈 값 처리
    if not title.strip():
        # 에러 처리 또는 기본값 설정
        pass
```

3. 템플릿에서 안전한 데이터 출력

```html
<!-- 자동 이스케이프 (기본 동작) -->
{{ user_input }}  <!-- HTML 태그가 자동으로 이스케이프됨 -->

<!-- 이스케이프 하지 않기 (주의해서 사용) -->
{{ trusted_html|safe }}

<!-- 조건부 출력 -->
{{ todo.description if todo.description else "설명 없음" }}
```

4. URL 생성 및 폼 처리

```html
<!-- 동적 URL 생성 -->
<form action="/toggle/{{ todo.id }}" method="post">
    <button type="submit">완료 토글</button>
</form>

<!-- 확인 대화상자 -->
<form action="/delete/{{ todo.id }}" method="post"
      onsubmit="return confirm('정말 삭제하시겠습니까?')">
    <button type="submit">삭제</button>
</form>
```

🔍 디버깅 및 개발 팁

1. 템플릿 디버깅

```html
<!-- 변수 내용 확인 (개발 시에만 사용) -->
<pre>{{ todos|pprint }}</pre>

<!-- 조건문 디버깅 -->
{% if debug_mode %}
    <div>현재 todos 개수: {{ todos|length }}</div>
    <div>완료된 할 일: {{ completed_todos }}</div>
{% endif %}
```

2. 폼 상태 유지

html

```html
<!-- 폼 제출 후 입력값 유지 (에러 시 유용) -->
<input type="text" name="title" value="{{ request.form.get('title', '') }}">

<!-- 선택 옵션 유지 -->
<select name="priority">
    <option value="높음" {% if request.form.get('priority') == '높음' %}selected{% endif %}>높음
    <option value="보통" {% if request.form.get('priority') == '보통' %}selected{% endif %}>보통
</select>
```

3. 에러 처리 및 피드백

python

```python
# 에러 메시지를 템플릿에 전달
@app.post("/add")
async def add_todo(request: Request, title: str = Form(...)):
    if not title.strip():
        return templates.TemplateResponse("index.html", {
            "request": request,
            "todos": todos,
            "error_message": "제목을 입력해주세요"
        })
```

html

```html
<!-- 템플릿에서 에러 메시지 표시 -->
{% if error_message %}
    <div class="error">{{ error_message }}</div>
{% endif %}
```

4. 성능 최적화 팁

python

```python
# 템플릿에 전달할 데이터 미리 계산
@app.get("/")
async def read_todos(request: Request):
    completed_count = sum(1 for todo in todos if todo["completed"])
    pending_count = len(todos) - completed_count

    return templates.TemplateResponse("index.html", {
        "request": request,
        "todos": todos,
        "stats": {
            "total": len(todos),
            "completed": completed_count,
            "pending": pending_count
        }
    })
```

5. 템플릿 구조화 팁

html

```html
<!-- 반복되는 HTML 블록을 매크로로 만들기 -->
{% macro render_todo(todo) %}
    <div class="todo-item {% if todo.completed %}completed{% endif %}">
        <h3>{{ todo.title }}</h3>
        <p>{{ todo.description or "설명 없음" }}</p>
        <!-- 버튼들 -->
    </div>
{% endmacro %}

<!-- 매크로 사용 -->
{% for todo in todos %}
    {{ render_todo(todo) }}
{% endfor %}
```

🎨 템플릿 구현 가이드

index.html 필수 포함 요소

1. 기본 HTML 구조

- DOCTYPE html, head, body 태그
- UTF-8 인코딩 설정
- 반응형 viewport 설정
- 페이지 제목: "할 일 관리"

2. 통계 정보 섹션

html

```
<div class="stats">
    <h2>📊 할 일 통계</h2>
    <p>전체: {{ total_todos }}개</p>
    <p>완료: {{ completed_todos }}개</p>
    <p>미완료: {{ pending_todos }}개</p>
</div>
```

3. 할 일 추가 폼

html

```
<form action="/add" method="post">
    <div>
        <label for="title">할 일 제목:</label>
        <input type="text" name="title" required maxlength="100">
    </div>
    <div>
        <label for="description">상세 설명:</label>
        <textarea name="description" maxlength="500"></textarea>
    </div>
    <div>
        <label for="priority">우선순위:</label>
        <select name="priority" required>
            <option value="높음">높음</option>
            <option value="보통" selected>보통</option>
            <option value="낮음">낮음</option>
        </select>
    </div>
    <button type="submit">할 일 추가</button>
</form>
```

4. 할 일 목록 표시

```html
<div class="todo-list">
    {% if todos %}
        {% for todo in todos %}
        <div class="todo-item {% if todo.completed %}completed{% endif %}">
            <h3>{{ todo.title }}</h3>
            <p>{{ todo.description }}</p>
            <div class="todo-meta">
                <span class="priority priority-{{ todo.priority }}">{{ todo.priority }}</span>
                <span class="created-at">{{ todo.created_at }}</span>
            </div>
            <div class="actions">
                <form action="/toggle/{{ todo.id }}" method="post" style="display: inline;">
                    <button type="submit">
                        {% if todo.completed %}미완료로 변경{% else %}완료{% endif %}
                    </button>
                </form>
                <form action="/delete/{{ todo.id }}" method="post" style="display: inline;">
                    <button type="submit" class="delete-btn"
                            onclick="return confirm('정말 삭제하시겠습니까?')">삭제</button>
                </form>
            </div>
        </div>
        {% endfor %}
    {% else %}
        <div class="no-todos">
            <p>아직 등록된 할 일이 없습니다.</p>
            <p>첫 번째 할 일을 추가해보세요! 📝</p>
        </div>
    {% endif %}
</div>
```

🎨 CSS 스타일링 가이드

기본 스타일링 요구사항

1. **반응형 디자인**: 모바일과 데스크톱 모두 지원
2. **완료된 할 일 구분**: 완료된 항목은 흐리게 표시하고 취소선 적용
3. **우선순위 색상**: 높음(빨강), 보통(노랑), 낮음(초록) 색상으로 구분
4. **호버 효과**: 버튼과 할 일 항목에 마우스 호버 시 시각적 피드백
5. **폼 스타일링**: 깔끔하고 사용하기 쉬운 폼 디자인

추천 CSS 클래스

```css
.todo-item.completed {
    opacity: 0.6;
    text-decoration: line-through;
}

.priority-높음 { color: #dc3545; }
.priority-보통 { color: #ffc107; }
.priority-낮음 { color: #28a745; }

.delete-btn {
    background-color: #dc3545;
    color: white;
}
```

🔍 테스트 시나리오

1. 개발 환경 설정 테스트

1. 서버 시작: `uvicorn main:app --reload`
2. 브라우저에서 `http://localhost:8000` 접속
3. 페이지가 정상적으로 로드되는지 확인
4. 개발자 도구(F12)에서 콘솔 에러 확인

2. 템플릿 렌더링 테스트

1. **초기 상태 확인**:
   - 빈 할 일 목록이 적절히 표시되는가?
   - "할 일이 없습니다" 메시지가 나타나는가?
   - 폼이 정상적으로 표시되는가?
2. **데이터 표시 테스트**:
   - 할 일 추가 후 목록에 즉시 반영되는가?
   - 우선순위가 올바르게 표시되는가?
   - 생성 시간이 정확히 표시되는가?

3. 폼 기능 테스트

1. **정상 입력**:
   - 제목만 입력하여 할 일 추가
   - 제목 + 설명 입력하여 할 일 추가
   - 모든 필드 입력하여 할 일 추가

2. **비정상 입력**:
   - 빈 제목으로 제출 시도 (에러 처리 확인)
   - 매우 긴 텍스트 입력 시도
   - 특수문자 포함 텍스트 입력
3. **폼 상호작용**:
   - 우선순위 드롭다운 선택 확인
   - 폼 제출 후 입력 필드 초기화 확인

4. 템플릿 조건부 렌더링 테스트

1. **완료 상태 토글**:
   - 미완료 → 완료 변경 시 시각적 변화 확인
   - 완료 → 미완료 변경 시 원상복구 확인
   - 버튼 텍스트가 상태에 따라 변경되는가?
2. **조건부 클래스/스타일**:
   - 완료된 할 일에 `completed` 클래스 적용 확인
   - 우선순위별 색상 구분 확인
   - 호버 효과 작동 확인

5. 반응형 및 사용성 테스트

1. **모바일 테스트**:
   - 브라우저 개발자 도구에서 모바일 뷰 확인
   - 터치 인터페이스에서 버튼 클릭 테스트
   - 가로/세로 화면 전환 테스트
2. **접근성 테스트**:
   - 폼 라벨과 입력 필드 연결 확인
   - 키보드만으로 모든 기능 사용 가능한지 확인
   - 화면 확대 시 레이아웃 깨짐 없는지 확인

6. 브라우저 호환성 테스트

- Chrome, Firefox, Safari에서 동일하게 작동하는가?
- 구형 브라우저에서도 기본 기능은 작동하는가?

🐛 일반적인 오류 및 해결법

1. 템플릿 관련 오류

```
TemplateNotFound: index.html
```

해결법: `templates` 폴더가 `main.py`와 같은 위치에 있는지 확인

```
TemplateError: 'request' is undefined
```

해결법: 템플릿 응답에 `request` 객체 포함했는지 확인

2. 폼 처리 오류

```
422 Unprocessable Entity
```

해결법:

- `python-multipart` 설치 확인
- 폼 필드명과 함수 매개변수명 일치 확인
- `enctype="multipart/form-data"` 추가 (파일 업로드 시)

3. 템플릿 문법 오류

```
html
<!-- ❌ 잘못된 문법 -->
{{ if todo.completed }}완료{{ endif }}

<!-- ✅ 올바른 문법 -->
{% if todo.completed %}완료{% endif %}
```

4. 변수 스코프 오류

```
html
<!-- ❌ 반복문 외부에서 loop 변수 사용 -->
{% for todo in todos %}{% endfor %}
{{ loop.index }}  <!-- 오류! -->

<!-- ✅ 반복문 내부에서 사용 -->
{% for todo in todos %}
    {{ loop.index }}. {{ todo.title }}
{% endfor %}
```

5. CSS/JS 로딩 문제

- 정적 파일 서빙 설정: `app.mount("/static", StaticFiles(directory="static"), name="static")`

- 캐시 문제: 브라우저 새로고침(Ctrl+F5) 또는 개발자 도구에서 캐시 비활성화

💡 구현 힌트

1. 템플릿 변수 전달 패턴

python

```python
@app.get("/", response_class=HTMLResponse)
async def read_todos(request: Request):
    # 통계 데이터 미리 계산
    completed_count = sum(1 for todo in todos if todo["completed"])
    pending_count = len(todos) - completed_count

    return templates.TemplateResponse("index.html", {
        "request": request,        # 필수!
        "todos": todos,
        "total_todos": len(todos),
        "completed_todos": completed_count,
        "pending_todos": pending_count
    })
```

2. 폼 데이터 검증 및 처리

python

```python
@app.post("/add")
async def add_todo(
    request: Request,
    title: str = Form(...),
    description: str = Form(""),
    priority: str = Form(...)
):
    # 입력값 정제
    title = title.strip()
    description = description.strip()

    # 간단한 유효성 검사
    if not title:
        return templates.TemplateResponse("index.html", {
            "request": request,
            "todos": todos,
            "error": "제목을 입력해주세요"
        })

    # 새 할 일 생성
    new_todo = {
        "id": len(todos) + 1,
        "title": title,
        "description": description,
        "completed": False,
        "created_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "priority": priority
    }

    todos.append(new_todo)
    return RedirectResponse(url="/", status_code=303)
```

3. 날짜/시간 처리

python

```python
from datetime import datetime

# 현재 시간을 문자열로 저장
created_at = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

# 템플릿에서 날짜 포맷팅을 위한 함수 (선택사항)
def format_date(date_string):
    return datetime.strptime(date_string, "%Y-%m-%d %H:%M:%S").strftime("%m월 %d일 %H:%M")
```

4. 리다이렉트 처리

python

```python
from fastapi.responses import RedirectResponse

# POST 요청 후 리다이렉트 (PRG 패턴)
return RedirectResponse(url="/", status_code=303)
```

5. 에러 처리 및 사용자 피드백

python

```python
# 존재하지 않는 할 일 처리
@app.post("/toggle/{todo_id}")
async def toggle_todo(todo_id: int):
    for todo in todos:
        if todo["id"] == todo_id:
            todo["completed"] = not todo["completed"]
            break
    else:
        # 찾지 못한 경우 - 조용히 무시하거나 에러 처리
        pass

    return RedirectResponse(url="/", status_code=303)
```

📋 체크리스트

템플릿 기본 기능

- [ ] **템플릿 렌더링**: HTML 페이지가 정상적으로 표시되는가?
- [ ] **변수 출력**: 템플릿에서 Python 변수가 올바르게 출력되는가?
- [ ] **조건문**: `{% if %}` 구문이 정상 작동하는가?
- [ ] **반복문**: `{% for %}` 구문으로 목록이 올바르게 표시되는가?
- [ ] **템플릿 상속**: 기본 레이아웃이 있다면 상속이 작동하는가?

폼 처리 기능

- [ ] **폼 제출**: POST 요청이 정상적으로 처리되는가?
- [ ] **폼 데이터**: Form() 클래스로 데이터가 올바르게 전달되는가?
- [ ] **리다이렉트**: 폼 제출 후 적절한 페이지로 이동하는가?
- [ ] **에러 처리**: 잘못된 입력에 대한 피드백이 있는가?

웹 애플리케이션 기능

- [ ] **할 일 추가**: 새 할 일이 정상적으로 추가되는가?

- [ ] **할 일 표시**: 추가된 할 일이 목록에 표시되는가?
- [ ] **상태 토글**: 완료/미완료 상태 변경이 작동하는가?
- [ ] **할 일 삭제**: 삭제 기능이 정상 작동하는가?
- [ ] **통계 표시**: 전체/완료/미완료 개수가 정확히 표시되는가?

사용자 경험 (UX)

- [ ] **시각적 구분**: 완료된 할 일이 시각적으로 구분되는가?
- [ ] **우선순위 표시**: 우선순위가 색상으로 구분되는가?
- [ ] **반응형 디자인**: 모바일에서도 사용하기 편한가?
- [ ] **확인 대화상자**: 삭제 시 확인 메시지가 나타나는가?
- [ ] **폼 사용성**: 입력 필드가 직관적이고 사용하기 쉬운가?

코드 품질

- [ ] **템플릿 구조**: HTML 구조가 의미적으로 올바른가?
- [ ] **변수 명명**: 템플릿 변수명이 명확하고 일관된가?
- [ ] **에러 핸들링**: 예외 상황이 적절히 처리되는가?
- [ ] **코드 가독성**: 템플릿과 Python 코드가 읽기 쉬운가?

완성된 프로젝트는 FastAPI와 Jinja2 템플릿을 활용한 실용적인 웹 애플리케이션의 기초가 될 것입니다. 템플릿 문법과 폼 처리 방법을 충분히 연습해보세요!