**SW사관학교 정글 - Chapter 4**

**[수업 목표]**

1. Flask 프레임워크를 활용해서 API를 만들 수 있다.

**01. Flask**

flask란?

* 📖 파이썬으로 쓰인 웹 프레임워크로, 서버를 구동하는 데 필요한 여러 기능들을 제공합니다.

flask 기초: 서버 실행하기

* 1.VSCode에서 chapter\_4 폴더를 Open Folder...로 열여준다.
* 2.python3 -m venv .venv로 가상 환경을 만들고 .venv/scripts/activate로 가상 환경을 활성화시킨다. Interpreter도 가상환경의 것으로 바꾼다.
* 3.project interpreter에서 flask 패키지를 설치한다.
* 4.app.py라는 새 파일을 만든 후 아래 코드를 붙여넣는다.
  + flask 시작 코드
    - from flask import Flask
    - app = Flask(\_\_name\_\_)
    - @app.route('/')
    - def home():

return 'This is Home!'

* + - if \_\_name\_\_ == '\_\_main\_\_':

app.run('0.0.0.0',port=5000,debug=True)

* + 👉 파일 이름은 아무렇게나 해도 상관없지만, 통상적으로 flask 서버를 돌리는 파일은 app.py라고 이름 짓습니다.
* 5.파일을 실행해 터미널에 아래와 같은 메시지가 뜨면 성공! 서버를 유지하는 동안 파이썬 코드가 계속 실행 중이므로 Process finished with exit code 0.가 나오지 않습니다.텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

  자동 생성된 설명
* 6.이제 크롬에서 <http://localhost:5000/> 으로 접속했을 때 화면에 This is Home!이 나오면 성공!
  + 👉 터미널 창을 클릭하시고, ctrl + c 을 누르시면 서버를 종료할 수 있습니다. 맥에서도 Cmd가 아니라 Ctrl!

Flask 기초: URL 나누기

* 1.아래 코드를 이용하여 URL마다 다른 글씨를 띄우도록 해보겠습니다.
  + from flask import Flask
  + app = Flask(\_\_name\_\_)
  + @app.route('/')
  + def home():

return 'This is Home!'

* + @app.route('/mypage')
  + def mypage():

return 'This is My Page!'

* + if \_\_name\_\_ == '\_\_main\_\_':

app.run('0.0.0.0',port=5000,debug=True)

* 2.크롬 브라우저에서 [http://localhost:5000/와](http://localhost:5000/%EC%99%80) [http://localhost:5000/mypage에](http://localhost:5000/mypage%EC%97%90) 가보면 다른 글씨가 뜨죠?
  + 👉 url 별로 함수명이 같거나, route('/')내의 주소가 같으면 안돼요!

Flask 기초: 기본 폴더구조

* Flask 서버를 만들 때는 항상 프로젝트 폴더 안에 static, templates 폴더와 app.py를 먼저 만들고 시작해주세요.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* + ⚠️ .venv는 가상환경에 대한 설정 정보가 담긴 폴더로, 안의 내용을 바꾸지 않는 것이 좋기 때문에 없는 것처럼 생각하는 것이 좋습니다.

Flask 기초: HTML 파일 불러오기

* 👉 templates 폴더는 HTML 파일을 담아두고 불러오는 역할을 합니다.
* 1.간단한 index.html 파일을 templates 안에 만들어주세요.
  + 예제코드
    - <!DOCTYPE html>
    - <html lang="en">
    - <head>
    - <meta charset="UTF-8">
    - <meta name="viewport" content="width=device-width, initial-scale=1.0">
    - <script src="<https://code.jquery.com/jquery-3.5.1.slim.min.js>"
    - integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
    - crossorigin="anonymous"></script>
    - <title>Document</title>
    - </head>
    - <body>
    - <h1>서버를 만들었다!</h1>
    - </body>
    - </html>

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* 2.html 파일을 app.py에서 불러온다. flask 프레임워크에 이미 구현되어 있는 내장함수 render\_template를 이용할 수 있습니다.
  + from flask import Flask, render\_template
  + app = Flask(\_\_name\_\_)
  + ## URL 별로 함수명이 같거나,
  + ## route('/') 등의 주소가 같으면 안됩니다.
  + @app.route('/')
  + def home():

return render\_template('index.html')

* + if \_\_name\_\_ == '\_\_main\_\_':

app.run('0.0.0.0', port=5000, debug=True)

* 3.<http://localhost:5000/에서> 해당 html이 뜨는 것을 확인한다.

Flask 기초: HTML 파일 내 이미지를 불러오기

* 👉 static 폴더는 html 파일 외에 이미지, css파일과 같은 파일을 담아두는 역할을 합니다.
* 1.static 폴더에 이미지를 넣는다.
  + 이미지

건물, 하늘, 경기장, 구름이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* 2.일반 HTML에서 이 파일을 불러온다면 img의 src를 이런 식으로 쓸 것입니다. rome.jpg의 경우 index.html이 있는 templates 폴더의 상위 폴더(코드로는 ..라고 씁니다) 안의 static 폴더 안에 있기 때문입니다
  + <!DOCTYPE html>
  + <html lang="en">
  + <head>
  + <meta charset="UTF-8">
  + <meta name="viewport" content="width=device-width, initial-scale=1.0">
  + <script src="<https://code.jquery.com/jquery-3.5.1.slim.min.js>"
  + integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
  + crossorigin="anonymous"></script>
  + <title>Document</title>
  + </head>
  + <body>
  + <h1>서버를 만들었다!</h1>
  + <img src="../static/rome.jpg"/>
  + </body>
  + </html>
* 3.하지만 flask에서 띄울 때는 flask에서 미리 정의된 방법으로 경로를 입력해주어야 합니다. 그래서 ststic 폴더 안의 rome.jpg의 경로는 아래와 같이 써줍니다.
  + <!DOCTYPE html>
  + <html lang="en">
  + <head>
  + <meta charset="UTF-8">
  + <meta name="viewport" content="width=device-width, initial-scale=1.0">
  + <script src="<https://code.jquery.com/jquery-3.5.1.slim.min.js>"
  + integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
  + crossorigin="anonymous"></script>
  + <title>Document</title>
  + </head>
  + <body>
  + <h1>서버를 만들었다!</h1>
  + <img src="{{ url\_for('static', filename='rome.jpg') }}"/>
  + </body>
  + </html>
  + 💡 위와 같이 적으면 flask가 html 정보를 보내줄 때 {{ }} 안의 내용을 상황에 맞게 바꿔 보내줍니다. 따옴표와 중괄호의 위치에 주의해 주세요.
* 4.app.py를 실행하고 <http://localhost:5000/>에 가서 개발자 도구 > Elements에 가서 보면 아래처럼 들어가 있죠?
  + <img src="/static/rome.jpg"/>
  + 💡 이미지 뿐만 아니라 헤드에 넣는 CSS 파일 등의 링크도 형식에 맞게 바꿔 넣어야합니다.

API 만들기

* 들어가기 전에: GET, POST 요청타입 – 리마인드
  + 👉 리마인드! 은행의 창구가 API와 같다는 것을 기억하시나요? \*\*\*\*같은 예금 창구에서도 개인 고객이냐 기업 고객이냐에 따라 처리하는 것이 다른 것처럼,
  + **클라이언트가 요청 할 때에도, "방식"이 존재합니다.** HTTP 라는 통신 규약을 따른다는 거 잊지 않으셨죠? 클라이언트는 요청할 때 HTTP request method(요청 메소드)를 통해, 어떤 요청 종류인지 응답하는 서버 쪽에 정보를 알려주는 거에요.
  + 👉 **GET, POST 방식**
  + 여러 방식([링크](https://developer.mozilla.org/ko/docs/Web/HTTP/Methods))이 존재하지만 우리는 가장 많이 쓰이는 GET, POST 방식에 대해 다루겠습니다.
    - GET → 통상적으로! 데이터 조회(Read)를 요청할 때 예) 영화 목록 조회 → **데이터 전달** : URL 뒤에 물음표를 붙여 key=value로 전달 → 예: [google.com?q=북극곰](http://google.com?q=%EB%B6%81%EA%B7%B9%EA%B3%B0)
    - POST → 통상적으로! 데이터 생성(Create), 변경(Update), 삭제(Delete) 요청 할 때 예) 회원가입, 회원탈퇴, 비밀번호 수정 → **데이터 전달 :** 바로 보이지 않는 HTML body에 key:value 형태로 전달

GET, POST 요청에서 클라이언트의 데이터를 받는 방법

* 예를 들어, 클라이언트에서 서버에 title\_give란 키 값으로 데이터를 들고왔다고 생각합시다. (주민등록번호 라는 키 값으로 850120- .. 을 가져온 것과 같은 의미)
  + 👉 받은 값을 개발자가 볼 수 있게 print 로 찍어볼 수 있게 했습니다. 실전에선 print로 찍어주는 것 외에, 여러가지 작업을 할 수 있겠죠?
  + GET 요청 API코드
    - @app.route('/test', methods=['GET'])
    - def test\_get():

title\_receive = request.args.get('title\_give')

print(title\_receive)

return jsonify({'result':'success', 'msg': '이 요청은 GET!'})

* + GET 요청 확인 Ajax코드
    - $.ajax({
    - type: "GET",
    - url: "/test?title\_give=봄날은간다",
    - data: {},
    - success: function(response){
    - console.log(response)
    - }
    - })
  + POST 요청 API코드
    - @app.route('/test', methods=['POST'])
    - def test\_post():
    - title\_receive = request.form['title\_give']
    - print(title\_receive)
    - return jsonify({'result':'success', 'msg': '이 요청은 POST!'})
  + POST 요청 확인 Ajax코드
    - $.ajax({
    - type: "POST",
    - url: "/test",
    - data: { title\_give:'봄날은간다' },
    - success: function(response){
    - console.log(response)
    - }
    - })

**02. 나홀로 링크 메모장 완성**

프로젝트 준비 - flask 폴더 구조 만들기

* 1.jungle 폴더 안에 alonememo 폴더를 만들고 VSCode에서 Open Folder...로 열어준다.
* 2.alonememo 폴더안에 가상환경을 만들고 static, templates 폴더와 [app.py](http://app.py) 파일을 만든다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

문제 분석 - 필요한 기능 생각하기

* 완성작을 보면서 어떤 기능이 필요한지 생각해봅시다! [링크](http://spartacodingclub.shop/)
* 1.페이지가 로딩되고 나면 DB에 저장되어 있는 기사 정보를 받아와 카드로 만들어 붙여준다.
  + 페이지 로딩 후 함수가 실행되게 한다.
  + 서버로 추가 정보 없이 GET 요청을 보낸다.
  + 서버는 DB에서 모든 기사의 제목, 설명, URL, 이미지URL, 코멘트 받아와 넘겨준다.
  + 받아온 정보로 각 기사에 대하여 카드를 만들어 붙인다.
* 2.포스팅 버튼을 누르면 입력한 url과 코멘트를 받아 DB에 제목, 설명, 이미지 URL까지 다 저장하고 페이지 새로고침한다. → 페이지 새로 로딩되면서 추가된 카드까지 나온다.
  + 포스팅 버튼을 눌러 함수가 실행되게 한다.
  + url과 코멘트를 잘 입력했는지 확인한다.
  + 입력 받은 url과 코멘트를 서버로 POST 요청을 보내준다.
  + 서버에서 url과 코멘트를 받아 제목, 설명 이미지 URL을 크롤링한다.
  + 5개 정보를 DB에 도큐먼트로 저장한다.
  + 성공 메시지를 반환하면 페이지를 새로고침한다.

프로젝트 설계 - 만들 API 설계

* ✅ 포스팅API - 카드 생성 (Create)
* **A. 요청 정보**
  + 요청 URL= /memo , 요청 방식 = POST
  + 요청 데이터 : URL(url\_give), 코멘트(comment\_give)
* **B. 서버가 제공할 기능**
  + URL의 meta태그 정보를 바탕으로 제목, 설명, 이미지URL 스크래핑
  + (제목, 설명, URL, 이미지URL, 코멘트) 정보를 모두 DB에 저장
* **C. 응답 데이터**
  + API가 정상적으로 작동하는지 클라이언트에게 알려주기 위해서 성공 메시지 보내기
  + (JSON 형식) 'result'= 'success'
* ✅ 리스팅API - 저장된 카드 보여주기 (Read)
* **A. 요청 정보**
  + 요청 URL= /memo , 요청 방식 = GET
  + 요청 데이터 : 없음
* **B. 서버가 제공할 기능**
  + DB에 저장돼 있는 모든 (제목, 설명, URL, 이미지URL, 코멘트) 정보를 가져오기
* **C. 응답 데이터**
  + API 동작 잘했다는 성공 메시지, 아티클(기사)들의 정보(제목, 설명, URL, 이미지URL, 코멘트)
  + (JSON 형식) 'result'= 'success', 'articles': 아티클 정보

프로젝트 준비 - URL로부터 페이지 정보 받아오는 기능 구현

* 사용자가 입력한 것은 URL과 코멘트 밖에 없는데, 만들어진 카드를 보면 기사 제목, 썸네일 이미지, 내용이 같이 나오는 것을 볼 수 있습니다. 이 값들을 어떻게 만들어낼 것인지 다른 파이썬 파일에서 작성해 실행해보고 잘 구현되면 코드를 본 파일로 붙여넣어봅시다!
* 카카오톡/페이스북 등 많은 곳에서 이렇게 페이지 URL만 있어도 자동으로 썸네일 이미지와 제목, 간단한 설명 등을 카드에 띄워주는데, 이들 모두 해당 html 파일 head에 있는 '메타태그'의 값을 참조한 것입니다.

텍스트, 인간의 얼굴, 스크린샷, 사람이(가) 표시된 사진

자동 생성된 설명

* 기사를 직접 브라우저에 띄워보면([링크](https://platum.kr/archives/120958)) head 안에 아래와 같이 추가정보들이 들어있는 것을 확인할 수 있습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* 우리는 그 중 og:image / og:title / og:description 을 크롤링해올 것입니다.
* meta 태그 스크래핑 하기
  + 연습을 위해 meta\_prac.py 파일을 만들어봅니다. 기본 준비를 합니다.
    - import requests
    - from bs4 import BeautifulSoup
    - url = '<https://platum.kr/archives/120958>'
    - headers = {'User-Agent' : 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36'}
    - data = requests.get(url,headers=headers)
    - soup = BeautifulSoup(data.text, 'html.parser')
    - # 여기에 코딩을 해서 meta tag를 먼저 가져와보겠습니다.
  + select\_one()을 이용해 meta tag를 먼저 가져와봅니다.
    - 👉 select\_one의 새로운 사용법! 이렇게 또 알아가네요~!
    - og\_image = soup.select\_one('meta[property="og:image"]')
    - og\_title = soup.select\_one('meta[property="og:title"]')
    - og\_description = soup.select\_one('meta[property="og:description"]')
    - print(og\_image)
    - print(og\_title)
    - print(og\_description)
  + 가져온 meta tag의 content를 가져와봅시다.
    - url\_image = og\_image['content']
    - url\_title = og\_title['content']
    - url\_description = og\_description['content']
    - print(url\_image)
    - print(url\_title)
    - print(url\_description)

프로젝트 준비 - app.py,index.html 준비하기

* 나홀로메모장 - [app.py](http://app.py)
  + from flask import Flask, render\_template, jsonify, request
  + app = Flask(\_\_name\_\_)
  + import requests
  + from bs4 import BeautifulSoup
  + from pymongo import MongoClient
  + client = MongoClient('MongoDB Atlas Link')
  + db = client.dbjungle
  + ## HTML을 주는 부분
  + @app.route('/')
  + def home():

return render\_template('index.html')

* + @app.route('/memo', methods=['GET'])
  + def listing():

# 1. 모든 document 찾기 & \_id 값은 출력에서 제외하기

# 2. articles라는 키 값으로 도서정보 내려주기

return jsonify({'result':'success', 'msg':'GET 연결되었습니다!'})

* + ## API 역할을 하는 부분
  + @app.route('/memo', methods=['POST'])
  + def saving():

# 1. 클라이언트로부터 데이터를 받기

# 2. meta tag를 스크래핑하기

# 3. mongoDB에 데이터 넣기

return jsonify({'result': 'success', 'msg':'POST 연결되었습니다!'})

* + if \_\_name\_\_ == '\_\_main\_\_':

app.run('0.0.0.0',port=5000,debug=True)

* 나홀로메모장 - index.html
  + <!DOCTYPE html>
  + <html lang="ko">
  + <head>
  + <!-- Required meta tags -->
  + <meta charset="utf-8">
  + <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  + <!-- Bootstrap CSS -->
  + <link rel="stylesheet" href="<https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css"

integrity="sha384-B0vP5xmATw1+K9KRQjQERJvTumQW0nPEzvF6L/Z6nronJ3oUOFUFpCjEUQouq2+l"

crossorigin="anonymous">

* + <!-- JS -->
  + <script src="<https://cdn.jsdelivr.net/npm/jquery@3.5.1/dist/jquery.min.js"

integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="

crossorigin="anonymous"></script>

* + <script src="<https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js>"

integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7lN"

crossorigin="anonymous"></script>

* + <!-- 구글폰트 -->
  + <link href="<https://fonts.googleapis.com/css?family=Stylish&display=swap>" rel="stylesheet">
  + <title>SW사관학교 정글 | 나홀로 메모장</title>
  + <!-- style -->
  + <style type="text/css">
  + \* {

font-family: "Stylish", sans-serif;

}

* + .wrap {

width: 900px;

margin: auto;

}

* + .comment {

color: blue;

font-weight: bold;

}

* + #post-box {

width: 500px;

margin: 20px auto;

padding: 50px;

border: black solid;

border-radius: 5px;

}

* + </style>
  + <script>
  + $(document).ready(function () {

showArticles();

});

* + function openClose() {

// id 값 post-box의 display 값이 block 이면(= 눈에 보이면)

if ($("#post-box").css("display") == "block") {

// post-box를 가리고

$("#post-box").hide();

// 다시 버튼을 클릭하면, 박스 열기를 할 수 있게 텍스트 바꿔두기

$("#btn-post-box").text("포스팅 박스 열기");

} else {

// 아니면(눈에 보이지 않으면) post-box를 펴라

$("#post-box").show();

// 다시 버튼을 클릭하면, 박스 닫기를 할 수 있게 텍스트 바꿔두기

$("#btn-post-box").text("포스팅 박스 닫기");

}

}

* + function postArticle() {

$.ajax({

type: "POST",

url: "/memo",

data: {},

success: function (response) { // 성공하면

if (response["result"] == "success") {

alert(response["msg"]);

}

}

})

}

* + function showArticles() {

$("#cards-box").html("");

$.ajax({

type: "GET",

url: "/memo",

data: {},

success: function (response) {

if (response["result"] == "success") {

alert(response["msg"]);

}

}

})

}

* + function makeCard(url, title, desc, comment, image) {

}

* + </script>
  + </head>
  + <body>
  + <div class="wrap">
  + <div class="jumbotron">
  + <h1 class="display-4">나홀로 링크 메모장!</h1>
  + <p class="lead">중요한 링크를 저장해두고, 나중에 볼 수 있는 공간입니다</p>
  + <hr class="my-4">
  + <p class="lead">
  + <button onclick="openClose()" id="btn-post-box" type="button" class="btn btn-primary">포스팅 박스 열기
  + </button>
  + </p>
  + </div>
  + <div id="post-box" class="form-post" style="display:none">
  + <div>
  + <div class="form-group">
  + <label for="post-url">아티클 URL</label>
  + <input id="post-url" class="form-control" placeholder="">
  + </div>
  + <div class="form-group">
  + <label for="post-comment">간단 코멘트</label>
  + <textarea id="post-comment" class="form-control" rows="2"></textarea>
  + </div>
  + <button type="button" class="btn btn-primary" onclick="postArticle()">기사저장</button>
  + </div>
  + </div>
  + <div id="cards-box" class="card-columns">
  + <div class="card">
  + <img class="card-img-top" src="<https://www.eurail.com/content/dam/images/eurail/Italy%20OCP%20Promo%20Block.adaptive.767.1535627244182.jpg" alt="Card image cap">
  + <div class="card-body">
  + <a href="#" class="card-title">여기 기사 제목이 들어가죠</a>
  + <p class="card-text">기사의 요약 내용이 들어갑니다. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세 무궁화 삼천리 화려강산...</p>
  + <p class="card-text comment">여기에 코멘트가 들어갑니다.</p>
  + </div>
  + </div>
  + <div class="card">
  + <img class="card-img-top" src="<https://www.eurail.com/content/dam/images/eurail/Italy%20OCP%20Promo%20Block.adaptive.767.1535627244182.jpg" alt="Card image cap">
  + <div class="card-body">
  + <a href="#" class="card-title">여기 기사 제목이 들어가죠</a>
  + <p class="card-text">기사의 요약 내용이 들어갑니다. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세 무궁화 삼천리 화려강산...</p>
  + <p class="card-text comment">여기에 코멘트가 들어갑니다.</p>
  + </div>
  + </div>
  + <div class="card">
  + <img class="card-img-top" src="<https://www.eurail.com/content/dam/images/eurail/Italy%20OCP%20Promo%20Block.adaptive.767.1535627244182.jpg" alt="Card image cap">
  + <div class="card-body">
  + <a href="#" class="card-title">여기 기사 제목이 들어가죠</a>
  + <p class="card-text">기사의 요약 내용이 들어갑니다. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세 무궁화 삼천리 화려강산...</p>
  + <p class="card-text comment">여기에 코멘트가 들어갑니다.</p>
  + </div>
  + </div>
  + <div class="card">
  + <img class="card-img-top" src="<https://www.eurail.com/content/dam/images/eurail/Italy%20OCP%20Promo%20Block.adaptive.767.1535627244182.jpg" alt="Card image cap">
  + <div class="card-body">
  + <a href="#" class="card-title">여기 기사 제목이 들어가죠</a>
  + <p class="card-text">기사의 요약 내용이 들어갑니다. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세 무궁화 삼천리 화려강산...</p>
  + <p class="card-text comment">여기에 코멘트가 들어갑니다.</p>
  + </div>
  + </div>
  + <div class="card">
  + <img class="card-img-top" src="<https://www.eurail.com/content/dam/images/eurail/Italy%20OCP%20Promo%20Block.adaptive.767.1535627244182.jpg" alt="Card image cap">
  + <div class="card-body">
  + <a href="#" class="card-title">여기 기사 제목이 들어가죠</a>
  + <p class="card-text">기사의 요약 내용이 들어갑니다. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세 무궁화 삼천리 화려강산...</p>
  + <p class="card-text comment">여기에 코멘트가 들어갑니다.</p>
  + </div>
  + </div>
  + <div class="card">
  + <img class="card-img-top" src="<https://www.eurail.com/content/dam/images/eurail/Italy%20OCP%20Promo%20Block.adaptive.767.1535627244182.jpg" alt="Card image cap">
  + <div class="card-body">
  + <a href="#" class="card-title">여기 기사 제목이 들어가죠</a>
  + <p class="card-text">기사의 요약 내용이 들어갑니다. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세 무궁화 삼천리 화려강산...</p>
  + <p class="card-text comment">여기에 코멘트가 들어갑니다.</p>
  + </div>
  + </div>
  + </div>
  + </div>
  + </body>
  + </html>

API 만들고 사용하기 - 포스팅API (Create → **POST**)

* ✅ **우리가 만들 API 두 가지**
* **1) 포스팅API - 카드 생성 (Create) : 클라이언트에서 받은 url, comment를 이용해서 페이지 정보를 찾고 저장하기**
* 2) 리스팅API - 저장된 카드 보여주기 (Read)
* 1.클라이언트와 서버 연결 확인하기
  + 여기서는 미리 적혀 있는 쌍으로 되어있는 서버-클라이언트 코드를 확인하고 갈게요.
  + 분홍 형광펜 부분이 서로 어떻게 매칭되는지 확인해보세요!
  + [서버 코드 - app.py]
    - @app.route('/memo', methods=['POST'])
    - def post\_articles():
      * # 1. 클라이언트로부터 데이터를 받기
      * # 2. meta tag를 스크래핑하기
      * # 3. mongoDB에 데이터 넣기
      * return jsonify({'result': 'success', 'msg': 'POST 연결되었습니다!'})
  + [클라이언트 코드 - index.html]
    - function postArticle() {
    - $.ajax({
    - type: "POST",
    - url: "/memo",
    - data: {},
    - success: function (response) { // 성공하면
    - if (response['result'] == 'success') {
    - alert(response['msg']);
    - }
    - }
    - })
    - }
    - <button type="button" class="btn btn-primary" onclick="postArticle()">기사저장</button>
  + 👉 **동작 테스트** '기사저장' 버튼을 클릭했을 때, 'POST 연결되었습니다!' alert창이 뜨면 클라이언트 코드와 서버 코드가 연결 되어있는 것입니다.
* 2.서버부터 만들기
  + 👉 API 는 약속이라고 했습니다. 위에 미리 설계해 둔 API 정보를 보고 만들어보죠!
  + 메모를 작성하기 위해 서버가 전달받아야하는 정보는 다음 두 가지입니다.
    - URL(url\_give)
    - 코멘트(comment\_give)
  + 그리고 URL를 meta tag를 스크래핑해서 아래 데이터를 저장(Create)합니다.
    - URL(url)
    - 제목(title)
    - 설명(desc)
    - 이미지URL(image)
    - 코멘트(comment)
  + 따라서 서버 로직은 다음 단계로 구성되어야 합니다.
    - 클라이언트로부터 데이터를 받기
    - meta tag를 스크래핑하기
    - mongoDB에 데이터를 넣기
  + @app.route('/memo', methods=['POST'])

def post\_article():

# 1. 클라이언트로부터 데이터를 받기

url\_receive = request.form['url\_give'] # 클라이언트로부터 url을 받는 부분

comment\_receive = request.form['comment\_give'] # 클라이언트로부터 comment를 받는 부분

# 2. meta tag를 스크래핑하기

headers = {

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36'}

data = requests.get(url\_receive, headers=headers)

soup = BeautifulSoup(data.text, 'html.parser')

og\_image = soup.select\_one('meta[property="og:image"]')

og\_title = soup.select\_one('meta[property="og:title"]')

og\_description = soup.select\_one('meta[property="og:description"]')

url\_title = og\_title['content']

url\_description = og\_description['content']

url\_image = og\_image['content']

article = {'url': url\_receive, 'title': url\_title, 'desc': url\_description, 'image': url\_image,

'comment': comment\_receive}

# 3. mongoDB에 데이터를 넣기

db.articles.insert\_one(article)

return jsonify({'result': 'success'})

* 3.클라이언트 만들기
  + 👉 API 는 약속이라고 했습니다. API를 사용할 클라이언트를 만들어보죠!
  + 메모를 작성하기 위해 서버에게 주어야하는 정보는 다음 두 가지입니다.
    - URL (url\_give) : meta tag를 가져올 url
    - comment (comment\_give) : 유저가 입력한 코멘트
  + 따라서 클라이언트 로직은 다음 단계로 구성되어야 합니다.
    - 유저가 입력한 데이터를 #post-url과 #post-comment에서 가져오기
    - /memo에 POST 방식으로 메모 생성 요청하기
    - 성공 시 리스트 새로고침하기
  + function postArticle() {

// 1. 유저가 입력한 데이터를 #post-url과 #post-comment에서 가져오기

let url = $("#post-url").val();

let comment = $("#post-comment").val();

// 2. memo에 POST 방식으로 메모 생성 요청하기

$.ajax({

type: "POST", // POST 방식으로 요청하겠다.

url: "/memo", // /memo라는 url에 요청하겠다.

data: { url\_give: url, comment\_give: comment}, // 데이터를 주는 방법

success: function(response){ // 성공하면

if (response["result"] == "success") {

alert("포스팅 성공!");

// 3. 성공 시 리스트 새로고침하기

showArticles();

} else {

alert("서버 오류!")

}

}

})

}

* 4.완성 확인하기
  + 👉 **동작 테스트** <https://platum.kr/archives/129737> ← 이 URL을 입력하고 기사저장을 눌렀을 때, '포스팅 성공!' alert창이 뜨는지 확인합니다. (우리는 스크래핑을 사용해 정보를 저장하고 있으니까, meta tag 가 있는 사이트만 저장이 제대로 되겠죠?)
  + 참고! 지금은 카드가 보이지 않습니다. 아직 카드를 보여주는 리스팅 API 를 만들지 않았기 때문이죠.

API 만들고 사용하기 - 리스팅 API (Read → **GET**)

* ✅ **우리가 만들 API 두 가지**
  + 포스팅API - 카드 생성 (Create) : 클라이언트에서 받은 url, comment를 이용해서 페이지 정보를 찾고 저장하기
  + **리스팅API - 저장된 카드 보여주기 (Read)**
* 1.클라이언트와 서버 연결 확인하기
  + 여기서는 미리 적혀 있는 쌍으로 되어있는 서버-클라이언트 코드를 확인하고 갈게요.
  + 분홍 형광펜 부분이 서로 어떻게 매칭되는지 확인해보세요!
  + [서버 코드 - app.py]
  + @app.route('/memo', methods=['GET'])

def read\_articles():

# 1. 모든 document 찾기 & \_id 값은 출력에서 제외하기

# 2. articles라는 키 값으로 영화정보 내려주기

return jsonify({'result':'success', 'msg':'GET 연결되었습니다!'})

[클라이언트 코드 - index.html]

function showArticles() {

$.ajax({

type: "GET",

url: "/memo",

data: {},

success: function (response) {

if (response["result"] == "success") {

alert(response["msg"]);

}

}

})

}

* + 👉 **동작 테스트** 새로고침했을 때, 'GET 연결되었습니다!' alert창이 뜨면 클라이언트 코드와 서버 코드가 연결 되어있는 것입니다.
* 2.서버부터 만들기
  + 👉 API 는 약속이라고 했습니다. 위에 미리 설계해 둔 API 정보를 보고 만들어보죠!
  + 메모를 보여주기 위해 서버가 추가로 전달받아야하는 정보는 없습니다. 조건없이 모든 메모를 보여줄 꺼니까요!
  + 따라서 서버 로직은 다음 단계로 구성되어야 합니다.
    - mongoDB에서 \_id 값을 제외한 모든 데이터 조회해오기 (Read)
    - articles라는 키 값으로 articles 정보 보내주기

@app.route('/memo', methods=['GET'])

def read\_articles():

# 1. mongoDB에서 \_id 값을 제외한 모든 데이터 조회해오기 (Read)

result = list(db.articles.find({}, {'\_id': 0}))

# 2. articles라는 키 값으로 article 정보 보내주기

return jsonify({'result': 'success', 'articles': result})

* + 👉 **중간 테스트** [http://localhost:5000/memo에](http://localhost:5000/memo%EC%97%90) 접속했을 때, 결과가 JSON 형식으로 보이면 성공입니다. 브라우저에서 주소창에 엔터를 치는 것이 GET 요청과 같다는 사실, 기억하시죠??
* 3.클라이언트 만들기
  + 👉 API 는 약속이라고 했습니다. API를 사용할 클라이언트를 만들어보죠!
  + 메모를 작성하기 위해 서버에게 주어야하는 정보는 없습니다. 조건없이 모든 메모를 가져오기 때문입니다.
  + 따라서 클라이언트 로직은 다음 단계로 구성되어야 합니다.
    - /memo에 GET 방식으로 메모 정보 요청하고 articles로 메모 정보 받기
    - , makeCard 함수를 이용해서 카드 HTML 붙이기 (→ 2주차 Ajax 연습과 같습니다!)

function showArticles() {

$.ajax({

type: "GET",

url: "/memo",

data: {},

success: function(response){

let articles = response["articles"];

for (let i = 0; i < articles.length; i++) {

makeCard(articles[i]["image"], articles[i]["url"], articles[i]["title"], articles[i]["desc"], articles[i]["comment"])

}

}

})

}

function makeCard(image, url, title, desc, comment) {

let temp\_html = `<div class="card">

<img class="card-img-top" src="${image}" alt="Card image cap">

<div class="card-body">

<a href="${url}" target="\_blank" class="card-title">${title}</a>

<p class="card-text">${desc}</p>

<p class="card-text comment">${comment}</p>

</div>

</div>`;

$("#cards-box").append(temp\_html);

}

* 4.완성 확인하기
  + 👉 **동작 테스트** 새로고침했을 때, 앞 포스팅 API를 만들고 테스트했던 메모가 보이면 성공입니다.
  + 참고. \*\*\*\*card가 정렬되는 순서는 위에서 아래로 채워지고, 왼쪽부터 오른쪽으로 순서대로 채워집니다. 부트스트랩 컴퍼넌트 페이지에 적혀있어요. "*Cards are ordered from top to bottom and left to right.*" ([컴퍼넌트 페이지 링크](https://getbootstrap.com/docs/4.6/components/card/#card-columns))

전체 완성 코드

* 클라이언트 코드 index.html
  + <!Doctype html>
  + <html lang="ko">
  + <head>
  + <!-- Required meta tags -->
  + <meta charset="utf-8">
  + <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  + <!-- Bootstrap CSS -->
  + <link rel="stylesheet" href="[https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css" integrity="sha384-B0vP5xmATw1+K9KRQjQERJvTumQW0nPEzvF6L/Z6nronJ3oUOFUFpCjEUQouq2+l" crossorigin="anonymous"](https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css%22%20integrity=%22sha384-B0vP5xmATw1+K9KRQjQERJvTumQW0nPEzvF6L/Z6nronJ3oUOFUFpCjEUQouq2+l%22%20crossorigin=%22anonymous%22)
  + <!-- JS -->
  + <script src="<https://cdn.jsdelivr.net/npm/jquery@3.5.1/dist/jquery.min.js" integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0=" crossorigin="anonymous"></script>
  + <script src="<https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js" integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7lN" crossorigin="anonymous"></script>
  + <!-- 구글폰트 -->
  + <link href="<https://fonts.googleapis.com/css?family=Stylish&display=swap>" rel="stylesheet">
  + <title>SW사관학교 정글 | 나홀로 메모장</title>
  + <!-- style -->
  + <style type="text/css">
  + \* {

font-family: "Stylish", sans-serif;

}

* 1. .wrap {
  2. width: 900px;
  3. margin: auto;
  4. }
  5. .comment {
  6. color: blue;
  7. font-weight: bold;
  8. }
  9. #post-box {
  10. width: 500px;
  11. margin: 20px auto;
  12. padding: 50px;
  13. border: black solid;
  14. border-radius: 5px;
  15. }
  16. </style>
  17. <script>
  18. $(document).ready(function () {
  19. showArticles();
  20. });
  21. function openClose() {
  22. // id 값 post-box의 display 값이 block 이면(= 눈에 보이면)
  23. if ($("#post-box").css("display") == "block") {
  24. // post-box를 가리고
  25. $("#post-box").hide();
  26. // 다시 버튼을 클릭하면, 박스 열기를 할 수 있게 텍스트 바꿔두기
  27. $("#btn-post-box").text("포스팅 박스 열기");
  28. } else {
  29. // 아니면(눈에 보이지 않으면) post-box를 펴라
  30. $("#post-box").show();
  31. // 다시 버튼을 클릭하면, 박스 닫기를 할 수 있게 텍스트 바꿔두기
  32. $("#btn-post-box").text("포스팅 박스 닫기");
  33. }
  34. }
  35. function postArticle() {
  36. let url = $("#post-url").val();
  37. let comment = $("#post-comment").val();
  38. // 2. memo에 POST 방식으로 메모 생성 요청하기
  39. $.ajax({
  40. type: "POST", // POST 방식으로 요청하겠다.
  41. url: "/memo", // /memo라는 url에 요청하겠다.
  42. data: {url\_give: url, comment\_give: comment}, // 데이터를 주는 방법
  43. success: function (response) { // 성공하면
  44. if (response["result"] == "success") {
  45. alert("포스팅 성공!");
  46. // 3. 성공 시 리스트 새로고침하기
  47. showArticles();
  48. } else {
  49. alert("서버 오류!")
  50. }
  51. }
  52. })
  53. }
  54. function showArticles() {
  55. $("#cards-box").html("");
  56. $.ajax({
  57. type: "GET",
  58. url: "/memo",
  59. data: {},
  60. success: function (response) {
  61. let articles = response["articles"];
  62. console.log(articles);
  63. for (let i = 0; i < articles.length; i++) {
  64. makeCard(articles[i]["image"], articles[i]["url"], articles[i]["title"], articles[i]["desc"], articles[i]["comment"]);
  65. }
  66. }
  67. });
  68. }
  69. function makeCard(image, url, title, desc, comment) {
  70. let tempHtml = `<div class="card">
  71. <img class="card-img-top" src="${image}" alt="Card image cap">
  72. <div class="card-body">
  73. <a href="${url}" target="\_blank" class="card-title">${title}</a>
  74. <p class="card-text">${desc}</p>
  75. <p class="card-text comment">${comment}</p>
  76. </div>
  77. </div>`;
  78. $("#cards-box").append(tempHtml);
  79. }
  80. </script>
  81. </head>
  82. <body>
  83. <div class="wrap">
  84. <div class="jumbotron">
  85. <h1 class="display-4">나홀로 링크 메모장!</h1>
  86. <p class="lead">중요한 링크를 저장해두고, 나중에 볼 수 있는 공간입니다</p>
  87. <hr class="my-4">
  88. <p class="lead">
  89. <button onclick="openClose()" id="btn-post-box" type="button" class="btn btn-primary">포스팅 박스 열기
  90. </button>
  91. </p>
  92. </div>
  93. <div id="post-box" class="form-post" style="display:none">
  94. <div>
  95. <div class="form-group">
  96. <label for="post-url">아티클 URL</label>
  97. <input id="post-url" class="form-control" placeholder="">
  98. </div>
  99. <div class="form-group">
  100. <label for="post-comment">간단 코멘트</label>
  101. <textarea id="post-comment" class="form-control" rows="2"></textarea>
  102. </div>
  103. <button type="button" class="btn btn-primary" onclick="postArticle()">기사저장</button>
  104. </div>
  105. </div>
  106. <div id="cards-box" class="card-columns">
  107. <div class="card">
  108. <img class="card-img-top"
  109. src="<https://www.eurail.com/content/dam/images/eurail/Italy%20OCP%20Promo%20Block.adaptive.767.1535627244182.jpg>"
  110. alt="Card image cap">
  111. <div class="card-body">
  112. <a href="#" class="card-title">여기 기사 제목이 들어가죠</a>
  113. <p class="card-text">기사의 요약 내용이 들어갑니다. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세 무궁화 삼천리 화려강산...</p>
  114. <p class="card-text comment">여기에 코멘트가 들어갑니다.</p>
  115. </div>
  116. </div>
  117. <div class="card">
  118. <img class="card-img-top"
  119. src="<https://www.eurail.com/content/dam/images/eurail/Italy%20OCP%20Promo%20Block.adaptive.767.1535627244182.jpg>"
  120. alt="Card image cap">
  121. <div class="card-body">
  122. <a href="#" class="card-title">여기 기사 제목이 들어가죠</a>
  123. <p class="card-text">기사의 요약 내용이 들어갑니다. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세 무궁화 삼천리 화려강산...</p>
  124. <p class="card-text comment">여기에 코멘트가 들어갑니다.</p>
  125. </div>
  126. </div>
  127. <div class="card">
  128. <img class="card-img-top"
  129. src="<https://www.eurail.com/content/dam/images/eurail/Italy%20OCP%20Promo%20Block.adaptive.767.1535627244182.jpg>"
  130. alt="Card image cap">
  131. <div class="card-body">
  132. <a href="#" class="card-title">여기 기사 제목이 들어가죠</a>
  133. <p class="card-text">기사의 요약 내용이 들어갑니다. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세 무궁화 삼천리 화려강산...</p>
  134. <p class="card-text comment">여기에 코멘트가 들어갑니다.</p>
  135. </div>
  136. </div>
  137. <div class="card">
  138. <img class="card-img-top"
  139. src="<https://www.eurail.com/content/dam/images/eurail/Italy%20OCP%20Promo%20Block.adaptive.767.1535627244182.jpg>"
  140. alt="Card image cap">
  141. <div class="card-body">
  142. <a href="#" class="card-title">여기 기사 제목이 들어가죠</a>
  143. <p class="card-text">기사의 요약 내용이 들어갑니다. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세 무궁화 삼천리 화려강산...</p>
  144. <p class="card-text comment">여기에 코멘트가 들어갑니다.</p>
  145. </div>
  146. </div>
  147. <div class="card">
  148. <img class="card-img-top"
  149. src="<https://www.eurail.com/content/dam/images/eurail/Italy%20OCP%20Promo%20Block.adaptive.767.1535627244182.jpg>"
  150. alt="Card image cap">
  151. <div class="card-body">
  152. <a href="#" class="card-title">여기 기사 제목이 들어가죠</a>
  153. <p class="card-text">기사의 요약 내용이 들어갑니다. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세 무궁화 삼천리 화려강산...</p>
  154. <p class="card-text comment">여기에 코멘트가 들어갑니다.</p>
  155. </div>
  156. </div>
  157. <div class="card">
  158. <img class="card-img-top"
  159. src="<https://www.eurail.com/content/dam/images/eurail/Italy%20OCP%20Promo%20Block.adaptive.767.1535627244182.jpg>"
  160. alt="Card image cap">
  161. <div class="card-body">
  162. <a href="#" class="card-title">여기 기사 제목이 들어가죠</a>
  163. <p class="card-text">기사의 요약 내용이 들어갑니다. 동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라만세 무궁화 삼천리 화려강산...</p>
  164. <p class="card-text comment">여기에 코멘트가 들어갑니다.</p>
  165. </div>
  166. </div>
  167. </div>
  168. </div>
  169. </body>
  170. </html>
  171. 서버 코드 app.py
  172. from flask import Flask, render\_template, jsonify, request
  173. import requests
  174. from bs4 import BeautifulSoup
  175. from pymongo import MongoClient # pymongo를 임포트 하기(패키지 인스톨 먼저 해야겠죠?)
  176. app = Flask(\_\_name\_\_)
  177. client = MongoClient('MongoDB Atlas Link') # MongoDB Altas 사이트의 링크를 가져옵니다
  178. db = client.dbjungle # 'dbjungle'이라는 이름의 db를 만들거나 사용합니다.
  179. @app.route('/')
  180. def home():
  181. return render\_template('index.html')
  182. @app.route('/memo', methods=['POST'])
  183. def post\_article():
  184. # 1. 클라이언트로부터 데이터를 받기
  185. url\_receive = request.form['url\_give'] # 클라이언트로부터 url을 받는 부분
  186. comment\_receive = request.form['comment\_give'] # 클라이언트로부터 comment를 받는 부분
  187. # 2. meta tag를 스크래핑하기
  188. headers = {
  189. 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36'}
  190. data = requests.get(url\_receive, headers=headers)
  191. soup = BeautifulSoup(data.text, 'html.parser')
  192. og\_image = soup.select\_one('meta[property="og:image"]')
  193. og\_title = soup.select\_one('meta[property="og:title"]')
  194. og\_description = soup.select\_one('meta[property="og:description"]')
  195. url\_title = og\_title['content']
  196. url\_description = og\_description['content']
  197. url\_image = og\_image['content']
  198. article = {'url': url\_receive, 'title': url\_title, 'desc': url\_description, 'image': url\_image,
  199. 'comment': comment\_receive}
  200. # 3. mongoDB에 데이터를 넣기
  201. db.articles.insert\_one(article)
  202. return jsonify({'result': 'success'})
  203. @app.route('/memo', methods=['GET'])
  204. def read\_articles():
  205. # 1. mongoDB에서 \_id 값을 제외한 모든 데이터 조회해오기 (Read)
  206. result = list(db.articles.find({}, {'\_id': 0}))
  207. # 2. articles라는 키 값으로 article 정보 보내주기
  208. return jsonify({'result': 'success', 'articles': result})
  209. if \_\_name\_\_ == '\_\_main\_\_':
  210. app.run('0.0.0.0', port=5000, debug=True)

**03. 마이페이보릿북 완성**

프로젝트 준비 - flask 폴더 구조 만들기

* 🔥 새 프로젝트는 새 가상환경에!
  + 1.jungle 폴더 안에 books폴더를 만들고 VSCode에서 Open Folder로 열어준다.
  + 2.books폴더 안에 가상환경을 만들고 static, templates 폴더와 [app.py](http://app.py) 파일을 만든다

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명.

문제 분석 - 완성작부터 보기!

* 완성작을 보면서 어떤 기능이 필요한지 생각해봅시다! [링크](https://edu.spartacodingclub.kr/jungle/books)
* 1.우선 책 정보(도서명, 썸네일, 작가명, 책 정보 URL)가 있어야 한다.
* 2.페이지가 로딩되고 나면 DB에 저장되어 있는 도서 정보를 받아와 카드로 만들어 붙여준다.
  + 페이지 로딩 후 함수가 실행되게 한다.
  + 서버로 추가 정보 없이 GET 요청을 보낸다.
  + 서버는 DB에서 모든 도서명, 썸네일 URL, 작가명, 도서정보 URL, 좋아요 수 받아와 '**좋아요가 많은 순으로**' 넘겨준다.
  + 받아온 정보로 각 도서에 대하여 카드를 만들어 붙인다.
* 3.위로! 버튼을 누르면 도서명을 받아 DB에 좋아요 수를 업데이트하고 페이지 새로고침한다. → 페이지 새로 로딩되면서 추가된 카드까지 나온다.
  + 위로! 버튼을 눌러 함수가 실행되게 한다.
  + 입력 받은 도서명을 서버로 POST 요청을 보내준다.
  + DB에서 도서명을 받아 현재 좋아요 개수를 찾는다.
  + 현재 좋아요 개수에 1을 더해 DB에 저장한다.
  + 성공 메시지를 반환하면 페이지를 새로고침한다.
* 4.삭제 버튼을 누르면 도서명을 받아 DB에 좋아요 수를 업데이트하고 페이지 새로고침한다. → 페이지 새로 로딩되면서 추가된 카드까지 나온다.
  + 삭제 버튼을 눌러 함수가 실행되게 한다.
  + 입력 받은 도서 이름을 서버로 POST 요청을 보내준다.
  + DB에서 도서명을 받아 삭제해준다.
  + 성공 메시지를 반환하면 페이지를 새로고침한다.

프로젝트 준비 - 프로젝트에서 사용할 데이터 넣기(웹 스크래핑)

* 👉 **API를 설계하고 만드는 것에 집중**할 수 있게,
  + 사용할 데이터를 웹 스크래핑해서,
  + 데이터베이스에 저장하는 코드를 미리 작성해두었어요.
* books 폴더 안에 \*\*\*\*init\_db.py 파일을 만들어 아래 코드를 복사-붙여넣기 해주세요.
  + 코드
    - import requests
    - from bs4 import BeautifulSoup
    - from pymongo import MongoClient
    - client = MongoClient('MongoDB Atlas Link')
    - db = client.dbjungle
    - # 타겟 URL을 읽어서 HTML를 받아오고,
    - headers = {'User-Agent' : 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36'}
    - url = '<https://www.yes24.com/Product/Category/BestSeller?categoryNumber=001&pageNumber=1&pageSize=50>'
    - data = requests.get(url,headers=headers)
    - # HTML을 BeautifulSoup이라는 라이브러리를 활용해 검색하기 용이한 상태로 만듦
    - # soup이라는 변수에 "파싱 용이해진 html"이 담긴 상태가 됨
    - # 이제 코딩을 통해 필요한 부분을 추출하면 된다.
    - soup = BeautifulSoup(data.text, 'html.parser')
    - books = soup.select('#yesBestList > li')
    - db.books.drop()
    - for i in books:
    - title = i.select\_one('a.gd\_name').text.strip()
    - img\_url = i.select\_one('em.img\_bdr > img')['data-original']
    - auth = i.select\_one('span.authPub > a').text
    - href = i.select\_one('a.gd\_name')['href']
    - href = '<https://yes24.com>' + href
    - # print(rank, title, auth)
    - db.books.insert\_one({"title" : title, "img\_url" : img\_url, "auth" : auth, "href" : href, "like" : 0})
    - print("mission complete!")
* init\_db.py 파일을 실행하면 내 mongoDB의 dbjungle.bookscollection에 도서 정보가 저장됩니다.
* dbjungle.books콜렉션에 데이터가 저장된 모습은 다음과 같습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

프로젝트 준비 - index.html, app.py 준비하기

* 페이보릿북 - index.html
  1. <!DOCTYPE html>
  2. <html lang="ko">
  3. <head>
  4. <meta charset="UTF-8"/>
  5. <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  6. <title>마이 페이보릿 북 | 프론트-백엔드 연결 마지막 예제!</title>
  7. <script src="<https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js>"></script>
  8. <link rel="stylesheet" href="<https://cdn.jsdelivr.net/npm/bulma@0.8.0/css/bulma.min.css>"/>
  9. <script defer src="<https://use.fontawesome.com/releases/v5.3.1/js/all.js>"></script>
  10. <style>
  11. .center {
  12. text-align: center;
  13. }
  14. .book-list {
  15. width: 500px;
  16. margin: 20px auto 0 auto;
  17. }
  18. .star-name {
  19. display: inline-block;
  20. }
  21. .star-name:hover {
  22. text-decoration: underline;
  23. }
  24. .card {
  25. margin-bottom: 15px;
  26. }
  27. </style>
  28. <script>
  29. $(document).ready(function () {
  30. // index.html 로드가 완료되면 자동으로 showBooks() 함수를 호출합니다.
  31. showBooks();
  32. });
  33. function showBooks() {
  34. $.ajax({
  35. type: 'GET',
  36. url: '/api/books/list',
  37. data: {},
  38. success: function (response) {
  39. if (response['result'] == 'success') {
  40. let msg = response['msg'];
  41. alert(msg);
  42. }
  43. }
  44. });
  45. }
  46. function likeBooks(name) {
  47. $.ajax({
  48. type: 'POST',
  49. url: '/api/books/like',
  50. data: {},
  51. success: function (response) {
  52. if (response['result'] == 'success') {
  53. let msg = response['msg'];
  54. alert(msg);
  55. }
  56. }
  57. });
  58. }
  59. function deleteBooks(name) {
  60. $.ajax({
  61. type: 'POST',
  62. url: '/api/books/delete',
  63. data: {},
  64. success: function (response) {
  65. if (response['result'] == 'success') {
  66. let msg = response['msg'];
  67. alert(msg);
  68. }
  69. }
  70. });
  71. }
  72. </script>
  73. </head>
  74. <body>
  75. <section class="hero is-warning">
  76. <div class="hero-body">
  77. <div class="container center">
  78. <h1 class="title">
  79. 마이 페이보릿 북😆
  80. </h1>
  81. <h2 class="subtitle">
  82. 순위를 매겨봅시다
  83. </h2>
  84. </div>
  85. </div>
  86. </section>
  87. <div class="book-list" id="book-box">
  88. <div class="card">
  89. <div class="card-content">
  90. <div class="media">
  91. <div class="media-left">
  92. <figure class="image is-48x48">
  93. <img
  94. src="<https://image.yes24.com/goods/122120495/L>"
  95. alt="Placeholder image"
  96. />
  97. </figure>
  98. </div>
  99. <div class="media-content">
  100. <a href="#" target="\_blank" class="star-name title is-4">강용수 (좋아요: 3)</a>
  101. <p class="subtitle is-6">마흔에 읽는 쇼펜하우어</p>
  102. </div>
  103. </div>
  104. </div>
  105. <footer class="card-footer">
  106. <a href="#" onclick="likeBooks('강용수')" class="card-footer-item has-text-info">
  107. 위로!
  108. <span class="icon">
  109. <i class="fas fa-thumbs-up"></i>
  110. </span>
  111. </a>
  112. <a href="#" onclick="deleteBooks('강용수')" class="card-footer-item has-text-danger">
  113. 삭제
  114. <span class="icon">
  115. <i class="fas fa-ban"></i>
  116. </span>
  117. </a>
  118. </footer>
  119. </div>
  120. </div>
  121. </body>
  122. </html>
* 페이보릿북 - app.py
  1. from pymongo import MongoClient
  2. from flask import Flask, render\_template, jsonify, request
  3. app = Flask(\_\_name\_\_)
  4. client = MongoClient('mongodb://test:test@localhost', 27017)
  5. db = client.dbjungle
  6. # HTML 화면 보여주기
  7. @app.route('/')
  8. def home():
  9. return render\_template('index.html')
  10. # API 역할을 하는 부분
  11. @app.route('/api/books/list', methods=['GET'])
  12. def show\_books():
  13. # 1. db에서 books 목록 전체를 검색합니다. ID는 제외하고 like 가 많은 순으로 정렬합니다.
  14. # 참고) find({},{'\_id':False}), sort()를 활용하면 굿!
  15. # 2. 성공하면 success 메시지와 함께 all\_books 목록을 클라이언트에 전달합니다.
  16. return jsonify({'result': 'success', 'msg': 'list 연결되었습니다!'})
  17. @app.route('/api/books/like', methods=['POST'])
  18. def like\_book():
  19. # 1. 클라이언트가 전달한 title\_give를 title\_receive 변수에 넣습니다.
  20. # 2. books 목록에서 find\_one으로 title이 title\_receive와 일치하는 book을 찾습니다.
  21. # 3. star의 like 에 1을 더해준 new\_like 변수를 만듭니다.
  22. # 4. books 목록에서 title이 title\_receive인 문서의 like 를 new\_like로 변경합니다.
  23. # 참고: '$set' 활용하기!
  24. # 5. 성공하면 success 메시지를 반환합니다.
  25. return jsonify({'result': 'success', 'msg': 'like 연결되었습니다!'})
  26. @app.route('/api/books/delete', methods=['POST'])
  27. def delete\_book():
  28. # 1. 클라이언트가 전달한 title\_give를 title\_receive 변수에 넣습니다.
  29. # 2. books 목록에서 delete\_one으로 title이 title\_receive와 일치하는 book을 제거합니다.
  30. # 3. 성공하면 success 메시지를 반환합니다.
  31. return jsonify({'result': 'success', 'msg': 'delete 연결되었습니다!'})
  32. if \_\_name\_\_ == '\_\_main\_\_':
  33. app.run('0.0.0.0', port=5000, debug=True)

문제 분석 - 화면과 동작 살펴보기

* 우리가 만들 기능은 도서 정보를 카드로 보여주기(Read)입니다.
* **화면에 어떤 데이터가 어떤 부분에 보여지는지** 도서 카드 화면 코드를 보며 분석해보겠습니다.
  + 도서명
  + 도서 이미지 : 이미지 src 속성
  + 좋아요 개수
  + 작가명
* 👉 index.html을 크롬에서 실행시켜 크롬 개발자도구 - 검사하기(Inspector)로 어떤 요소에 어떤 데이터가 보일지 분석해보세요.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명<!-- 다음 코드가 하나의 카드를 이루는 div 입니다. -->

<div class="card">

<div class="card-content">

<div class="media">

<div class="media-left">

<figure class="image is-48x48">

<img

src="<https://image.yes24.com/goods/122120495/L>"

alt="Placeholder image"

/>

</figure>

</div>

<div class="media-content">

<a href="#" target="\_blank" class="book-name title is-4">마흔에 읽는 쇼펜하우어 (좋아요: 3)</a>

<p class="subtitle is-6">강용수</p>

</div>

</div>

</div>

<footer class="card-footer">

<a href="#" onclick="likeBook('마흔에 읽는 쇼펜하우어')" class="card-footer-item has-text-info">

위로!

<span class="icon">

<i class="fas fa-thumbs-up"></i>

</span>

</a>

<a href="#" onclick="deleteBook('마흔에 읽는 쇼펜하우어')" class="card-footer-item has-text-danger">

삭제

<span class="icon">

<i class="fas fa-ban"></i>

</span>

</a>

</footer>

</div>

API 만들고 사용하기 - 책 정보 조회 API (Read → **GET**)

* 👉 만들 API
  + **1.조회(Read) 기능**: 책 정보 전체를 조회
  + **2.좋아요(Update) 기능**: 클라이언트에서 받은 이름(title\_give)으로 찾아서 좋아요(like)를 증가
  + **3.삭제(Delete) 기능**: 클라이언트에서 받은 이름(title\_give)으로 책 정보를 찾고, 해당 책 정보를 삭제

- 👉 정리하면, **만들 API 정보**는 아래와 같습니다.

**A. 요청 정보**

* 1. 요청 URL= /api/books/list , 요청 방식 = GET
  2. 요청 데이터 : 없음

**B. 서버가 제공할 기능** : 데이터베이스에 책 정보를 조회(Read)하고, 책 정보를 응답 데이터로 보냄

C\*\*. 응답 데이터 :\*\* (JSON 형식) 'result'= 'success', 'all\_books'= 책 정보 리스트

* 1.클라이언트와 서버 연결 확인하기
  + 여기서는 미리 적혀 있는 쌍으로 되어있는 서버-클라이언트 코드를 확인하고 갈게요.
  + 분홍 형광펜 부분이 서로 어떻게 매칭되는지 확인해보세요!
  + [서버 코드 - app.py]

@app.route('/api/books/list', methods=['GET'])

def show\_books():

# 1. db에서 books 목록 전체를 검색합니다. ID는 제외하고 like 가 많은 순으로 정렬합니다.

# 참고) find({},{'\_id':False}), sort()를 활용하면 굿!

# 2. 성공하면 success 메시지와 함께 all\_books 목록을 클라이언트에 전달합니다.

return jsonify({'result': 'success', 'msg': 'list 연결되었습니다!'})

* + [클라이언트 코드 - index.html]

function showBook(){

// 1. #book\_box의 내부 html 태그를 모두 삭제합니다.

// 2. 서버에 1) GET 방식으로, 2) /api/list 라는 주소로 all\_books를 요청합니다.

// 3. 서버가 돌려준 all\_books를 all\_books라는 변수에 저장합니다.

// 4. for 문을 활용하여 all\_books 배열의 요소를 차례대로 조회합니다.

// 5. all\_books[i] 요소의 title, img\_url, auth, href, like 키 값을 활용하여 값을 조회합니다.

// 6. 도서 카드를 만듭니다.

// 7. #book-box에 tempHtml을 붙입니다.

$.ajax({

type: 'GET',

url: '/api/books/list',

data: {},

success: function (response) {

let msg = response['msg'];

alert(msg);

}

});

}

* + 👉 **동작 테스트** 새로고침했을 때, 'list 연결되었습니다.' 라는 메시지가 뜨면 동작하는 것입니다.
* 2.서버부터 만들기
  + 👉 API 는 약속이라고 했습니다. 위에 미리 설계해 둔 API 정보를 보고 만들어보죠!
  + 도서 정보 전체를 조회하기 위해 서버가 받을 정보는 없습니다. 조건없이 모든 정보를 보여줄 것이니까요!
  + 따라서 서버 로직은 다음 단계로 구성되어야 합니다.
    - all\_books 목록 전체를 검색합니다. ID는 제외하고 like 가 많은 순으로 정렬
    - 성공하면 success 메시지와 함께 all\_books목록을 클라이언트에 전달

@app.route('/api/books/list', methods=['GET'])

def stars\_list():

# 1. db에서 books 목록 전체를 검색합니다. ID는 제외하고 like 가 많은 순으로 정렬합니다.

# 참고) find({},{'\_id':False}), sort()를 활용하면 굿!

all\_books = list(db.books.find({},{"\_id" : False}).sort("like", -1))

# 2. 성공하면 success 메시지와 함께 all\_books 목록을 클라이언트에 전달합니다.

return jsonify({"result" : "success", "all\_books" : all\_books})

* + 👉 **중간 테스트** <http://localhost:5000/api/books/list>에 접속했을 때, 결과가 JSON 형식으로 보이면 성공입니다!
* 3.클라이언트 만들기
  + 👉 API 는 약속이라고 했습니다. API를 사용할 클라이언트를 만들어보죠!
  + 도서 정보 전체를 조회하기 위해 서버가 받을 정보는 없습니다. 조건없이 모든 정보를 보여줄 것이니까요!
  + 따라서 클라이언트 로직은 다음 단계로 구성되어야 합니다.
    - #book\_box의 내부 html 태그를 모두 삭제
    - 서버에 1) GET 방식으로, 2) /api/books/list 라는 주소로 all\_books를 요청
    - 서버가 돌려준 all\_books를 books라는 변수에 저장
    - for 문을 활용하여 all\_books배열의 요소를 차례대로 조회
    - all\_books[i] 요소의 title, img\_url, auth, href, like 키 값을 활용하여 값 조회
    - 도서 카드 코드 만들어 #book-box에 붙이기

$(document).ready(function () {

// index.html 로드가 완료되면 자동으로 showBook() 함수를 호출합니다.

showStar();

});

function showStar() {

$.ajax({

type: 'GET',

url: '/api/books/list',

data: {},

success: function (response) {

if (response['result'] == 'success') {

$("#book-box").empty()

let all\_books = response['all\_books'];

for (i=0; i<all\_books.length; i++) {

let row = all\_books[i]

let title = row['title']

let auth = row['auth']

let href = row['href']

let img\_url = row['img\_url']

let like = row['like']

let temp\_html = `<div class="card">

<div class="card-content">

<div class="media">

<div class="media-left">

<figure class="image is-48x48">

<img src="${img\_url}" alt="Placeholder image">

</figure>

</div>

<div class="media-content">

<a href="#" target="\_blank" class="star-name title is-4">${auth} (좋아요: ${like})</a>

<p class="subtitle is-6">${title}</p>

</div>

</div>

</div>

<footer class="card-footer">

<a href="#" onclick="likeStar('${title}')" class="card-footer-item has-text-info">

위로!

<span class="icon">

<svg class="svg-inline--fa fa-thumbs-up fa-w-16" aria-hidden="true" data-prefix="fas" data-icon="thumbs-up" role="img" xmlns="<http://www.w3.org/2000/svg>" viewBox="0 0 512 512" data-fa-i2svg=""><path fill="currentColor" d="M104 224H24c-13.255 0-24 10.745-24 24v240c0 13.255 10.745 24 24 24h80c13.255 0 24-10.745 24-24V248c0-13.255-10.745-24-24-24zM64 472c-13.255 0-24-10.745-24-24s10.745-24 24-24 24 10.745 24 24-10.745 24-24 24zM384 81.452c0 42.416-25.97 66.208-33.277 94.548h101.723c33.397 0 59.397 27.746 59.553 58.098.084 17.938-7.546 37.249-19.439 49.197l-.11.11c9.836 23.337 8.237 56.037-9.308 79.469 8.681 25.895-.069 57.704-16.382 74.757 4.298 17.598 2.244 32.575-6.148 44.632C440.202 511.587 389.616 512 346.839 512l-2.845-.001c-48.287-.017-87.806-17.598-119.56-31.725-15.957-7.099-36.821-15.887-52.651-16.178-6.54-.12-11.783-5.457-11.783-11.998v-213.77c0-3.2 1.282-6.271 3.558-8.521 39.614-39.144 56.648-80.587 89.117-113.111 14.804-14.832 20.188-37.236 25.393-58.902C282.515 39.293 291.817 0 312 0c24 0 72 8 72 81.452z"></path></svg><!-- <i class="fas fa-thumbs-up"></i> -->

</span>

</a>

<a href="#" onclick="deleteStar('${title}')" class="card-footer-item has-text-danger">

삭제

<span class="icon">

<svg class="svg-inline--fa fa-ban fa-w-16" aria-hidden="true" data-prefix="fas" data-icon="ban" role="img" xmlns="<http://www.w3.org/2000/svg>" viewBox="0 0 512 512" data-fa-i2svg=""><path fill="currentColor" d="M256 8C119.034 8 8 119.033 8 256s111.034 248 248 248 248-111.034 248-248S392.967 8 256 8zm130.108 117.892c65.448 65.448 70 165.481 20.677 235.637L150.47 105.216c70.204-49.356 170.226-44.735 235.638 20.676zM125.892 386.108c-65.448-65.448-70-165.481-20.677-235.637L361.53 406.784c-70.203 49.356-170.226 44.736-235.638-20.676z"></path></svg><!-- <i class="fas fa-ban"></i> -->

</span>

</a>

</footer>

</div>`

$("#book-box").append(temp\_html)

}

}

}

});

}

* 4.완성 확인하기
  + 👉 **동작 테스트** 화면을 새로고침 했을 때 도 정보가 조회되는지 확인합니다.

API 만들고 사용하기 - 좋아요 API (Update → **POST**)

* 👉 **만들 API**
  + 조회: 도서 정보 전체를 조회
  + **좋아요: 클라이언트에서 받은 이름(title\_give)으로 찾아서 좋아요(like)를 증가**
  + 삭제: 클라이언트에서 받은 이름(title\_give)으로 도서명을 찾고, 해당 도서명을 삭제

- 👉 정리하면, **만들 API 정보**는 아래와 같습니다.

**A. 요청 정보**

* 1. 요청 URL= /api/like , 요청 방식 = POST
  2. 요청 데이터 : 도서명(title\_give)

**B. 서버가 제공할 기능** : 도서명(요청 데이터)과 일치하는 도서명 정보의 좋아요 수를 한 개 증가시켜 데이터베이스에 업데이트하고(Update), 성공했다고 응답 메세지를 보냄

C\*\*. 응답 데이터 :\*\* (JSON 형식) 'result'= 'success'

* 1.클라이언트와 서버 연결 확인하기
  + 여기서는 미리 적혀 있는 쌍으로 되어있는 서버-클라이언트 코드를 확인하고 갈게요.
  + 분홍 형광펜 부분이 서로 어떻게 매칭되는지 확인해보세요!
  + [서버 코드 - app.py]

@app.route('/api/books/like', methods=['POST'])

def like\_star():

# 1. 클라이언트가 전달한 title\_give를 title\_receive 변수에 넣습니다.

# 2. books목록에서 find\_one으로 name이 title\_receive와 일치하는 book를 찾습니다.

# 3. book의 like 에 1을 더해준 new\_like 변수를 만듭니다.

# 4. books목록에서 title이 title\_receive인 문서의 like 를 new\_like로 변경합니다.

# 참고: '$set' 활용하기!

# 5. 성공하면 success 메시지를 반환합니다.

return jsonify({'result': 'success','msg':'like 연결되었습니다!'})

* + [클라이언트 코드 - index.html]

function likeStar(name){

// 1. 서버에 1) POST 방식으로, 2) /api/like 라는 url에, 3) title\_give라는 이름으로 title을 전달합니다.

// 참고) POST 방식이므로 data: {'title\_give': title} 과 같은 양식이 되어야합니다!

// 2. '좋아요 완료!' 얼럿을 띄웁니다.

// 3. 변경된 정보를 반영하기 위해 새로고침합니다.

$.ajax({

type: 'POST',

url: '/api/books/like',

data: {},

success: function (response) {

if (response['result'] == 'success') {

let msg = response['msg'];

alert(msg);

}

}

});

}

* + 👉 **동작 테스트** '위로' 버튼을 눌렀을 때, 'like 연결되었습니다!' 내용의 alert창이 뜨면 제대로 동작하는 것입니다.
* 2.서버부터 만들기
  + 👉 API 는 약속이라고 했습니다. 위에 미리 설계해 둔 API 정보를 보고 만들어보죠!
  + 도서 카드의 좋아요 수를 증가시키기 위해 서버가 클라이언트에게 전달받아야하는 정보는 다음과 같습니다.
    - 도서의 이름 (title\_give)
  + 따라서 서버 로직은 다음 단계로 구성되어야 합니다.
    - 1.클라이언트가 전달한 title\_give를 title\_receive 변수에 넣습니다.
    - 2.mystar 목록에서 find\_one으로 title이 title\_receive와 일치하는 book를 찾습니다.
    - 3.book의 like 에 1을 더해준 new\_like 변수를 만듭니다.
    - 4.books 목록에서 title이 title\_receive인 문서의 like 를 new\_like로 변경합니다.

@app.route('/api/books/like', methods=['POST'])

def like\_book():

# 1. 클라이언트가 전달한 title\_give를 title\_receive 변수에 넣습니다.

title\_receive = request.form['title\_give']

# 2. books 목록에서 find\_one으로 title이 title\_receive와 일치하는 book을 찾습니다.

book = db.books.find\_one({"title" : title\_receive})

# 3. book의 like 에 1을 더해준 new\_like 변수를 만듭니다.

new\_like = book['like'] + 1

# 4. books 목록에서 title이 title\_receive인 문서의 like 를 new\_like로 변경합니다.

# 참고: '$set' 활용하기!

db.books.update\_one({"title" : title\_receive}, {'$set' : {"like" : new\_like}})

# 5. 성공하면 success 메시지를 반환합니다.

return jsonify({"result" : "success", "msg" : "좋아요 완료!"})

* 3.클라이언트 만들기
  + 👉 API 는 약속이라고 했습니다. API를 사용할 클라이언트를 만들어보죠!
  + 좋아요 수를 증가시키기 위해 클라이언트가 전달할 정보는 다음과 같습니다.
    - 도서명의 이름 (title\_give)
  + 따라서 클라이언트 로직은 다음 단계로 구성되어야 합니다.
    - 1.서버에
      * 1) POST 방식으로,
      * 2) /api/like 라는 url에,
      * 3) title\_give라는 이름으로 title을 전달합니다. (참고) POST 방식이므로 data: {'title\_give': title} 사용
    - 2.'좋아요 완료!' alert 창을 띄웁니다.
    - 3.변경된 정보를 반영하기 위해 카드 리스트 재생성

function likeBook(title) {

$.ajax({

type: 'POST',

url: '/api/books/like',

data: {'title\_give' : title},

success: function (response) {

if (response['result'] == 'success') {

let msg = response['msg'];

alert(msg);

showBooks();

}

}

});

}

* 4.완성 확인하기
  + 👉 **동작 테스트** '위로' 버튼을 눌렀을 때 좋아요 수가 증가하고 도 카드의 순위가 변경되는지 확인합니다.

API 만들고 사용하기 - 카드 삭제 API (Delete → **POST**)

* 👉 만들 API
  + **조회(Read) 기능**: 도 정보 전체를 조회
  + **좋아요(Update) 기능**: 클라이언트에서 받은 이름(title\_give)으로 찾아서 좋아요(like)를 증가
  + **삭제(Delete) 기능**: 클라이언트에서 받은 이름(title\_give)으로 도서를 찾고, 해당 도서를 삭제

- 👉 정리하면, **만들 API 정보**는 아래와 같습니다.

**A. 요청 정보**

* 1. 요청 URL= /api/delete , 요청 방식 = POST
  2. 요청 데이터 : 도서 이름(title\_give)

**B. 서버가 제공할 기능** : 도서 이름(요청 데이터)와 일치하는 도 정보를 데이터베이스에서 삭제(Delete)하고, 성공했다고 응답 메세지를 보냄

C\*\*. 응답 데이터 :\*\* (JSON 형식) 'result'= 'success'

* 1.클라이언트와 서버 연결 확인하기
  + 여기서는 미리 적혀 있는 쌍으로 되어있는 서버-클라이언트 코드를 확인하고 갈게요.
  + 분홍 형광펜 부분이 서로 어떻게 매칭되는지 확인해보세요!
  + [서버 코드 - app.py]

@app.route('/api/delete', methods=['POST'])

def delete\_book():

# 1. 클라이언트가 전달한 title\_give를 title\_receive 변수에 넣습니다.

# 2. books목록에서 delete\_one으로 title이 title\_receive와 일치하는 book를 제거합니다.

# 3. 성공하면 success 메시지를 반환합니다.

return jsonify({'result': 'success','msg':'delete 연결되었습니다!'})

* + [클라이언트 코드 - index.html]

function deleteStar(name){

// 1. 서버에 1) POST 방식으로, 2) /api/delete 라는 url에, 3) title\_give라는 이름으로 title을 전달합니다.

// 참고) POST 방식이므로 data: {'title\_give': title} 과 같은 양식이 되어야합니다!

// 2. '삭제 완료! 안녕!' 얼럿을 띄웁니다.

// 3. 변경된 정보를 반영하기 위해 새로고침합니다.

$.ajax({

type: 'POST',

url: '/api/delete',

data: {},

success: function (response) {

if (response['result'] == 'success') {

let msg = response['msg'];

alert(msg);

}

}

});

}

* + 👉 **동작 테스트** '삭제' 버튼을 눌렀을 때, 'delete 연결되었습니다!' alert창이 뜨면 클라이언트 코드와 서버 코드가 연결 되어있는 것입니다.
* 2.서버부터 만들기
  + 👉 API 는 약속이라고 했습니다. 위에 미리 설계해 둔 API 정보를 보고 만들어보죠!
  + 도서 카드를 삭제하기 위해 필요한 정보는 다음과 같습니다.
    - 도서의 이름 (title\_give)
  + 따라서 서버 로직은 다음 단계로 구성되어야 합니다.
    - 1.클라이언트가 전달한 title\_give를 title\_receive 변수에 넣기
    - 2.books에서 delete\_one으로 title이 title\_receive와 일치하는 star를 제거
    - 3.성공하면 success 메시지를 반환

@app.route('/api/books/delete', methods=['POST'])

def delete\_book():

# 1. 클라이언트가 전달한 title\_give를 title\_receive 변수에 넣습니다.

title\_receive = request.form['title\_give']

# 2. books 목록에서 delete\_one으로 title이 title\_receive와 일치하는 book을 제거합니다.

db.books.delete\_one({"title" : title\_receive})

# 3. 성공하면 success 메시지를 반환합니다.

return jsonify({"result" : "success", "msg" : "삭제 완료!"})

* 3.클라이언트 만들기
  + 👉 API 는 약속이라고 했습니다. API를 사용할 클라이언트를 만들어보죠!
  + 도서 카드를 삭제하기 위해 필요한 정보는 다음과 같습니다.
    - 도서의 이름 (title\_give)
  + 따라서 클라이언트 로직은 다음 단계로 구성되어야 합니다.
    - 1.서버에
      * 1) POST 방식으로,
      * 2) /api/delete 라는 url에,
      * 3) title\_give라는 이름으로 title을 전달 (참고) POST 방식이므로 data: {'title\_give': title}
    - 2.'삭제 완료! 안녕!' alert창 띄우기
    - 3.변경된 정보를 반영하기 위해 카드 리스트 재생성

function deleteBook(title) {

$.ajax({

type: 'POST',

url: '/api/books/delete',

data: {'title\_give' : title},

success: function (response) {

if (response['result'] == 'success') {

let msg = response['msg'];

alert(msg);

showBooks();

}

}

});

}

* 4.완성 확인하기
  + 👉 **동작 테스트** 삭제 버튼을 눌렀을 때 도서 카드가 삭제되는지 확인합니다.

전체 완성 코드

* 클라이언트 코드 - index.html
  1. <!DOCTYPE html>
  2. <html lang="ko">
  3. <head>
  4. <meta charset="UTF-8"/>
  5. <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  6. <title>마이 페이보릿 북 | 프론트-백엔드 연결 마지막 예제!</title>
  7. <script src="<https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js>"></script>
  8. <link rel="stylesheet" href="<https://cdn.jsdelivr.net/npm/bulma@0.8.0/css/bulma.min.css>"/>
  9. <script defer src="<https://use.fontawesome.com/releases/v5.3.1/js/all.js>"></script>
  10. <style>
  11. .center {
  12. text-align: center;
  13. }
  14. .book-list {
  15. width: 500px;
  16. margin: 20px auto 0 auto;
  17. }
  18. .book-name {
  19. display: inline-block;
  20. }
  21. .book-name:hover {
  22. text-decoration: underline;
  23. }
  24. .card {
  25. margin-bottom: 15px;
  26. }
  27. </style>
  28. <script>
  29. $(document).ready(function () {
  30. // index.html 로드가 완료되면 자동으로 showBooks() 함수를 호출합니다.
  31. showBooks();
  32. });
  33. function showBooks() {
  34. $.ajax({
  35. type: 'GET',
  36. url: '/api/books/list',
  37. data: {},
  38. success: function (response) {
  39. if (response['result'] == 'success') {
  40. $("#book-box").empty()
  41. let all\_books = response['all\_books'];
  42. for (i=0; i<all\_books.length; i++) {
  43. let row = all\_books[i]
  44. let title = row['title']
  45. let auth = row['auth']
  46. let href = row['href']
  47. let img\_url = row['img\_url']
  48. let like = row['like']
  49. let temp\_html = `<div class="card">
  50. <div class="card-content">
  51. <div class="media">
  52. <div class="media-left">
  53. <figure class="image is-48x48">
  54. <img src="${img\_url}" alt="Placeholder image">
  55. </figure>
  56. </div>
  57. <div class="media-content">
  58. <a href="${href}" target="\_blank" class="book-name title is-4">${title} (좋아요: ${like})</a>
  59. <p class="subtitle is-6">${auth}</p>
  60. </div>
  61. </div>
  62. </div>
  63. <footer class="card-footer">
  64. <a href="#" onclick="likeBook('${title}')" class="card-footer-item has-text-info">
  65. 위로!
  66. <span class="icon">
  67. <i class="fas fa-thumbs-up"></i>
  68. </span>
  69. </a>
  70. <a href="#" onclick="deleteBook('${title}')" class="card-footer-item has-text-danger">
  71. 삭제
  72. <span class="icon">
  73. <i class="fas fa-ban"></i>
  74. </span>
  75. </a>
  76. </footer>
  77. </div>`
  78. $("#book-box").append(temp\_html)
  79. }
  80. }
  81. }
  82. });
  83. }
  84. function likeBook(title) {
  85. $.ajax({
  86. type: 'POST',
  87. url: '/api/books/like',
  88. data: {'title\_give' : title},
  89. success: function (response) {
  90. if (response['result'] == 'success') {
  91. let msg = response['msg'];
  92. alert(msg);
  93. showBooks();
  94. }
  95. }
  96. });
  97. }
  98. function deleteBook(title) {
  99. $.ajax({
  100. type: 'POST',
  101. url: '/api/books/delete',
  102. data: {'title\_give' : title},
  103. success: function (response) {
  104. if (response['result'] == 'success') {
  105. let msg = response['msg'];
  106. alert(msg);
  107. showBooks();
  108. }
  109. }
  110. });
  111. }
  112. </script>
  113. </head>
  114. <body>
  115. <section class="hero is-warning">
  116. <div class="hero-body">
  117. <div class="container center">
  118. <h1 class="title">
  119. 마이 페이보릿 북😆
  120. </h1>
  121. <h2 class="subtitle">
  122. 순위를 매겨봅시다
  123. </h2>
  124. </div>
  125. </div>
  126. </section>
  127. <div class="book-list" id="book-box">
  128. <div class="card">
  129. <div class="card-content">
  130. <div class="media">
  131. <div class="media-left">
  132. <figure class="image is-48x48">
  133. <img
  134. src="<https://image.yes24.com/goods/122120495/L>"
  135. alt="Placeholder image"
  136. />
  137. </figure>
  138. </div>
  139. <div class="media-content">
  140. <a href="#" target="\_blank" class="book-name title is-4">마흔에 읽는 쇼펜하우어 (좋아요: 3)</a>
  141. <p class="subtitle is-6">강용수</p>
  142. </div>
  143. </div>
  144. </div>
  145. <footer class="card-footer">
  146. <a href="#" onclick="likeBook('마흔에 읽는 쇼펜하우어')" class="card-footer-item has-text-info">
  147. 위로!
  148. <span class="icon">
  149. <i class="fas fa-thumbs-up"></i>
  150. </span>
  151. </a>
  152. <a href="#" onclick="deleteBook('마흔에 읽는 쇼펜하우어')" class="card-footer-item has-text-danger">
  153. 삭제
  154. <span class="icon">
  155. <i class="fas fa-ban"></i>
  156. </span>
  157. </a>
  158. </footer>
  159. </div>
  160. </div>
  161. </body>
  162. </html>
* 서버 코드 - app.py
  1. from pymongo import MongoClient
  2. from flask import Flask, render\_template, jsonify, request
  3. app = Flask(\_\_name\_\_)
  4. client = MongoClient('mongodb://test:test@localhost', 27017)
  5. db = client.dbjungle
  6. # HTML 화면 보여주기
  7. @app.route('/')
  8. def home():
  9. return render\_template('index-final.html')
  10. # API 역할을 하는 부분
  11. @app.route('/api/books/list', methods=['GET'])
  12. def show\_books():
  13. # 1. db에서 books 목록 전체를 검색합니다. ID는 제외하고 like 가 많은 순으로 정렬합니다.
  14. # 참고) find({},{'\_id':False}), sort()를 활용하면 굿!
  15. all\_books = list(db.books.find({},{"\_id" : False}).sort("like", -1))
  16. # 2. 성공하면 success 메시지와 함께 all\_books 목록을 클라이언트에 전달합니다.
  17. return jsonify({"result" : "success", "all\_books" : all\_books})
  18. @app.route('/api/books/like', methods=['POST'])
  19. def like\_book():
  20. # 1. 클라이언트가 전달한 title\_give를 title\_receive 변수에 넣습니다.
  21. title\_receive = request.form['title\_give']
  22. # 2. books 목록에서 find\_one으로 title이 title\_receive와 일치하는 book을 찾습니다.
  23. target\_like = db.books.find\_one({"title" : title\_receive})["like"]
  24. # 3. star의 like 에 1을 더해준 new\_like 변수를 만듭니다.
  25. new\_like = target\_like + 1
  26. # 4. books 목록에서 title이 title\_receive인 문서의 like 를 new\_like로 변경합니다.
  27. # 참고: '$set' 활용하기!
  28. db.books.update\_one({"title" : title\_receive}, {'$set' : {"like" : new\_like}})
  29. # 5. 성공하면 success 메시지를 반환합니다.
  30. return jsonify({"result" : "success", "msg" : "좋아요 완료!"})
  31. @app.route('/api/books/delete', methods=['POST'])
  32. def delete\_book():
  33. # 1. 클라이언트가 전달한 title\_give를 title\_receive 변수에 넣습니다.
  34. title\_receive = request.form['title\_give']
  35. # 2. books 목록에서 delete\_one으로 title이 title\_receive와 일치하는 book을 제거합니다.
  36. db.books.delete\_one({"title" : title\_receive})
  37. # 3. 성공하면 success 메시지를 반환합니다.
  38. return jsonify({"result" : "success", "msg" : "삭제 완료!"})
  39. if \_\_name\_\_ == '\_\_main\_\_':
  40. app.run('0.0.0.0', port=5000, debug=True)
* 🔥 와, API 사용하고 만들기 한 번 더 복습하니 조금은 익숙해지죠? 마이페이보릿 북 프로젝트도 완성!