스파르타코딩클럽 8기 5주차

🔥 **매 주차 강의자료 시작에 PDF파일과 영상 링크를 올려두었어요!**

▼ PDF 강의자료 다운받기

> https://s3-us-west-2.amazonaws.com/secure.notion-static.com/9c5b031c-f784-4b8f-8745-fa2749928781/scc-8-1.pdf

▼ 영상강의 참고하기

- 1주차 복습용 영상강의 링크

[수업 목표]

1. Flask 프레임워크를 활용해서 API를 만들 수 있다.
2. '마이 페이보릿 무비스타'를 완성한다.
3. 내 프로젝트의 기획안을 완성한다.

전반 3시간

[시작] : 체크인 & 출석체크

🔥 **튜터님은 체크인과 함께 출석 체크(링크)를 진행해주세요!**

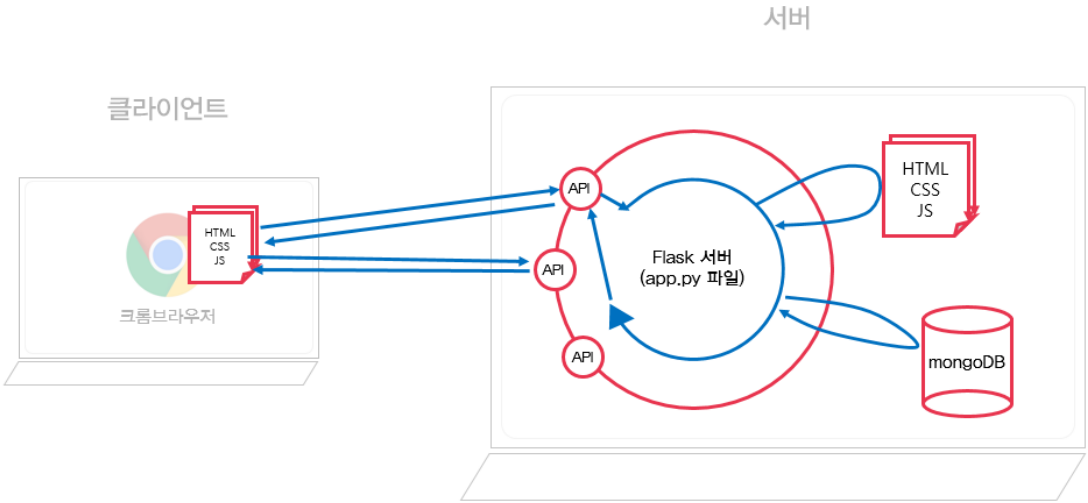
▼ "(오늘은) 30초 체크인"을 진행합니다 - 프로젝트 생김새 확인!

- 튜터는 타이머를 띄워주세요! (링크)
- 각자 30초 씩 돌아가며 프로젝트의 개요를 설명합니다.
- 튜터는 발표가 끝날 때마다 30초 내외로 피드백을 전달합니다.
  (범위, 참고해보면 좋은 서비스)

▼ 오늘 배울 것 이야기 - 5주차: 미니프로젝트3

이번 주 완성본 마이 페이보릿 무비스타 → 결과물 링크

> 👉 오늘은 아직 익숙해지지 않았을 당신을 위해! 같은 난이도의 유사한 프로젝트를 진행하며 머릿속의 퍼즐을 맞출 예정입니다.
>
> 여기까지 배웠다면, 이제 6~8주차 프로젝트 준비 완료!!



[1.5시간] : 마이 페이보릿 무비스타 GET 만들어보기

> 🔥 **VS Code 파일→폴더열기를 클릭해서,**
> **sparta > project > moviestar 폴더를 열고 시작!**

▼ 1) 완성작부터 보기!

▼ 2) 새 프로젝트니까 - 가상환경 다시 잡기

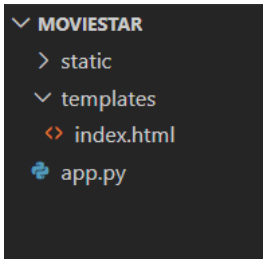
- **터미널 → 새 터미널**을 열어서, 아래를 입력!

```
python -m venv myenv
```

- 다시, **터미널 → 새 터미널**을 열면, 공구함 리로딩 완료!
- 공구함 만든 김에, 필요한 패키지를 미리 설치해둡시다!

```
pip install flask bs4 requests pymongo
```

▼ 3) 프로젝트 폴더 구조 만들기



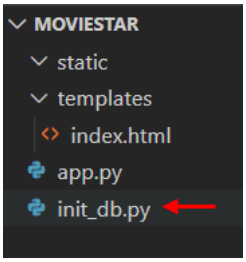
▼ 3) 크롤링으로 DB 쌓기

> 👉 **API를 구상하고 직접 만드는 것에만 집중**하실 수 있게, 데이터를 크롤링해 저장하는 과정은 저희가 미리 작성해두었습니다.

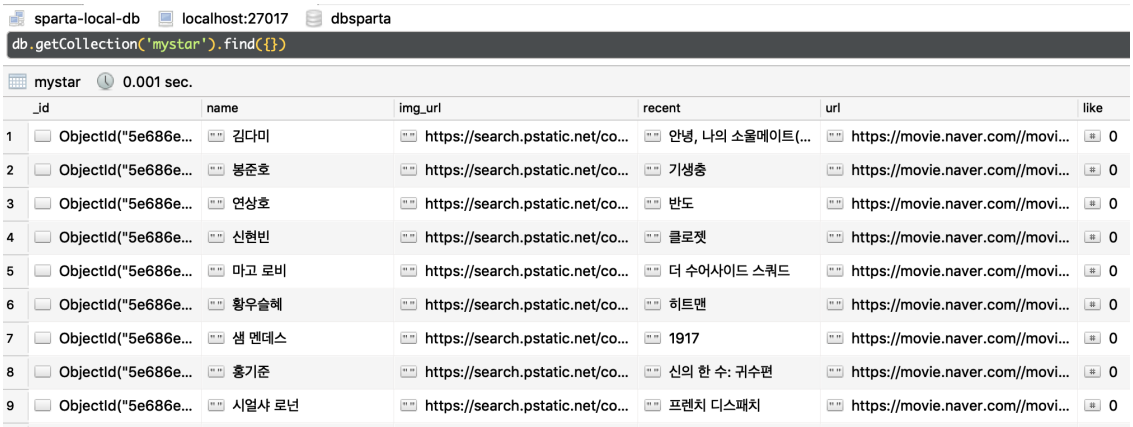
1. init_db.py 파일을 만들어 아래 코드를 복사붙여넣기 후 실행하시면 mystar 콜렉션에 영화인들의 정보가 저장됩니다. DB에 정보가 모두 입력됐으면 지워도 무방합니다.

   ▼ 코드보기

```
import requests
from bs4 import BeautifulSoup

from pymongo import MongoClient

client = MongoClient('localhost', 27017)
db = client.dbsparta

# DB에 저장할 영화인들의 출처 url을 가져옵니다.
def get_urls():
    headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)AppleWebKit/537.36 (KHTML, like Gecko) Chrom
    data = requests.get('https://movie.naver.com/movie/sdb/rank/rpeople.nhn', headers=headers)

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

    trs = soup.select('#old_content > table > tbody > tr')

    urls = []
    for tr in trs:
        a = tr.select_one('td.title > a')
        if a is not None:
            baseurl = 'https://movie.naver.com/'
            url = baseurl + a['href']
            urls.append(url)

    return urls

# 출처 url로부터 영화인들의 사진, 이름, 최근작 정보를 가져오고 mystar 콜렉션에 저장합니다.
def insert_star(url):
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.
    data = requests.get(url, headers=headers)

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

    name = soup.select_one('#content > div.article > div.mv_info_area > div.mv_info.character > h3 > a').text
    img_url = soup.select_one('#content > div.article > div.mv_info_area > div.poster > img')['src']
    recent = soup.select_one(
        '#content > div.article > div.mv_info_area > div.mv_info.character > dl > dd > a:nth-child(1)').text

    doc = {
        'name': name,
        'img_url': img_url,
        'recent': recent,
        'url': url,
        'like': 0
    }

    db.mystar.insert_one(doc)
    print('완료!', name)

# 기존 mystar 콜렉션을 삭제하고, 출처 url들을 가져온 후, 크롤링하여 DB에 저장합니다.
def insert_all():
    db.mystar.drop()  # mystar 콜렉션을 모두 지워줍니다.
    urls = get_urls()
    for url in urls:
        insert_star(url)


### 실행하기
insert_all()
```



2. mystar 콜렉션에 세팅 완료된 모습은 다음과 같습니다.



sparta-local-db localhost:27017 dbsparta

```
db.getCollection('mystar').find({})
```

mystar 0.001 sec.

	_id	name	img_url	recent	url	like
1	ObjectId("5e686e...	김다미	https://search.pstatic.net/co...	안녕, 나의 소울메이트(...	https://movie.naver.com//movi...	0
2	ObjectId("5e686e...	봉준호	https://search.pstatic.net/co...	기생충	https://movie.naver.com//movi...	0
3	ObjectId("5e686e...	연상호	https://search.pstatic.net/co...	반도	https://movie.naver.com//movi...	0
4	ObjectId("5e686e...	신현빈	https://search.pstatic.net/co...	클로젯	https://movie.naver.com//movi...	0
5	ObjectId("5e686e...	마고 로비	https://search.pstatic.net/co...	더 수어사이드 스쿼드	https://movie.naver.com//movi...	0
6	ObjectId("5e686e...	황우슬혜	https://search.pstatic.net/co...	히트맨	https://movie.naver.com//movi...	0
7	ObjectId("5e686e...	샘 멘데스	https://search.pstatic.net/co...	1917	https://movie.naver.com//movi...	0
8	ObjectId("5e686e...	홍기준	https://search.pstatic.net/co...	신의 한 수: 귀수편	https://movie.naver.com//movi...	0
9	ObjectId("5e686e...	시얼샤 로넌	https://search.pstatic.net/co...	프렌치 디스패치	https://movie.naver.com//movi...	0

▼ 4) index.html, app.py 준비하기

▼ index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>프론트-백엔드 연결 마지막 예제!</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
    <link
      rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/bulma@0.8.0/css/bulma.min.css"
    />
    <script
      defer
      src="https://use.fontawesome.com/releases/v5.3.1/js/all.js"
    ></script>
    <style>
      .make-center {
        text-align: center;
      }
      .star-list {
        width: 500px;
        margin: 20px auto 0 auto;
      }
      .star-name {
          display: inline-block;
      }
      .star-name:hover {
          text-decoration: underline;
      }
      .card {
          margin-bottom: 15px;
      }
    </style>
    <script>
      $(document).ready(function() {
        // index.html 로드가 완료되면 자동으로 show_star() 함수를 호출합니다.
        show_star();
      });

      function show_star(){
        $.ajax({
            type: 'GET',
            url:  '/api/list',
            data: {},
            success: function (response) {
              if (response['result'] == 'success') {
                let msg = response['msg'];
                alert(msg);
              }
            }
        });
      }

      function like_star(name){
          $.ajax({
              type: 'POST',
              url:  '/api/like',
              data: {},
              success: function (response) {
                  if (response['result'] == 'success') {
```

```
                    let msg = response['msg'];
                    alert(msg);
                }
            }
        });
    }

    function delete_star(name){
        $.ajax({
            type: 'POST',
            url:  '/api/delete',
            data: {},
            success: function (response) {
                if (response['result'] == 'success') {
                    let msg = response['msg'];
                    alert(msg);
                }
            }
        });
    }

  </script>
</head>
<body>
  <section class="hero is-warning">
    <div class="hero-body">
      <div class="container make-center">
        <h1 class="title">
          마이 페이보릿 무비스타😆
        </h1>
        <h2 class="subtitle">
          순위를 매겨봅시다
        </h2>
      </div>
    </div>
  </section>
  <div class="star-list" id="star-box">
    <div class="card">
      <div class="card-content">
        <div class="media">
          <div class="media-left">
            <figure class="image is-48x48">
              <img
                src="https://search.pstatic.net/common/?src=https%3A%2F%2Fssl.pstatic.net%2Fsstatic%2Fpeople%2Fportra
                alt="Placeholder image"
              />
            </figure>
          </div>
          <div class="media-content">
            <a href="#" target="_blank" class="star-name title is-4">김다미 (좋아요: 3)</a>
            <p class="subtitle is-6">안녕, 나의 소울메이트(가제)</p>
          </div>
        </div>
      </div>
      <footer class="card-footer">
        <a href="#" onclick="like_star('김다미')" class="card-footer-item has-text-info">
          위로!
          <span class="icon">
            <i class="fas fa-thumbs-up"></i>
          </span>
        </a>
        <a href="#" onclick="delete_star('김다미')" class="card-footer-item has-text-danger">
          삭제
          <span class="icon">
            <i class="fas fa-ban"></i>
          </span>
        </a>
      </footer>
    </div>
  </div>
</body>
</html>
```

▼ app.py

```
from pymongo import MongoClient

from flask import Flask, render_template, jsonify, request
app = Flask(__name__)

client = MongoClient('localhost', 27017)
db = client.dbsparta
```

```python
# HTML을 주는 부분
@app.route('/')
def home():
    return render_template('index.html')

# API 역할을 하는 부분
@app.route('/api/list', methods=['GET'])
def stars_list():
    # 1. mystar 목록 전체를 검색합니다. ID는 제외하고 like 가 많은 순으로 정렬합니다.
    # 참고) find({},{'_id':False}), sort()를 활용하면 굿!
    # 2. 성공하면 success 메시지와 함께 stars_list 목록을 클라이언트에 전달합니다.
  return jsonify({'result': 'success','msg':'list 연결되었습니다!'})


@app.route('/api/like', methods=['POST'])
def star_like():
    # 1. 클라이언트가 전달한 name_give를 name_receive 변수에 넣습니다.
    # 2. mystar 목록에서 find_one으로 name이 name_receive와 일치하는 star를 찾습니다.
    # 3. star의 like 에 1을 더해준 new_like 변수를 만듭니다.
    # 4. mystar 목록에서 name이 name_receive인 문서의 like 를 new_like로 변경합니다.
    # 참고: '$set' 활용하기!
    # 5. 성공하면 success 메시지를 반환합니다.
  return jsonify({'result': 'success','msg':'like 연결되었습니다!'})


@app.route('/api/delete', methods=['POST'])
def star_delete():
    # 1. 클라이언트가 전달한 name_give를 name_receive 변수에 넣습니다.
    # 2. mystar 목록에서 delete_one으로 name이 name_receive와 일치하는 star를 제거합니다.
    # 3. 성공하면 success 메시지를 반환합니다.
  return jsonify({'result': 'success','msg':'delete 연결되었습니다!'})


if __name__ == '__main__':
    app.run('0.0.0.0', port=5000, debug=True)
```

▼ 5) 영화인 카드의 HTML 살펴보기

👉 VS Code를 통해 index.html만을 브라우저에서 실행시켜보세요. (단축키: alt+b)

다음을 중점적으로 살펴본 다음에 **영화인 카드를 어떻게 만들 수 있을지** 생각해봅시다.

* 이름이 들어가는 부분
* 이미지 src 속성이 들어가는 부분
* 좋아요 개수와 최근작 내용이 들어가는 부분
* 좋아요 기능을 담당하는 like_star 함수의 onclick 호출 부분
* 삭제 기능을 담당하는 delete_star 함수의 onclick 호출 부분

서버가 넘겨준 정보를 이용하여 영화인 정보 카드를 만들 때는 다음 코드를 참고합니다.

```
<!-- 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://search.pstatic.net/common/?src=https%3A%2F%2Fssl.pstatic.net%2Fsstatic%2Fpeople%2Fportrait%2F20180
            alt="Placeholder image"
          />
        </figure>
      </div>
      <div class="media-content">
        <a href="https://movie.naver.com//movie/bi/pi/basic.nhn?st=1&code=397373" target="_blank" class="star-name title is
        <p class="subtitle is-6">안녕, 나의 소울메이트(가제)</p>
      </div>
    </div>
  </div>
  <footer class="card-footer">
    <a href="#" onclick="like_star('김다미')" class="card-footer-item has-text-info">
      위로!
      <span class="icon">
        <i class="fas fa-thumbs-up"></i>
      </span>
    </a>
    <a href="#" onclick="delete_star('김다미')" class="card-footer-item has-text-danger">
      삭제
      <span class="icon">
        <i class="fas fa-ban"></i>
      </span>
    </a>
  </footer>
</div>
```

▼ 6) GET 연습: 영화인 조회하기

✅ **우리가 만들 API 세 가지**

1) 조회: 영화인 정보 전체를 조회
2) 좋아요: 클라이언트에서 받은 이름(name_give)으로 찾아서 좋아요(like)를 증가
3) 삭제: 클라이언트에서 받은 이름(name_give)으로 영화인을 찾고, 해당 영화인을 삭제

▼ 1) 클라이언트와 서버 연결 확인하기

아래 클라이언트/서버 시작 코드는 쌍을 이루고 있습니다.

[서버 시작 코드]

```
@app.route('/api/list', methods=['GET'])
def stars_list():
    # 1. mystar 목록 전체를 검색합니다. ID는 제외하고 like 가 많은 순으로 정렬합니다.
    # 참고) find({},{'_id':False}), sort()를 활용하면 굿!
    # 2. 성공하면 success 메시지와 함께 stars_list 목록을 클라이언트에 전달합니다.
    return jsonify({'result': 'success','msg':'list 연결되었습니다!'})
```

[클라이언트 시작 코드]

```
function show_star(){
  // 1. #star_box의 내부 html 태그를 모두 삭제합니다.
  // 2. 서버에 1) GET 방식으로, 2) /api/list 라는 주소로 stars_list를 요청합니다.
  // 3. 서버가 돌려준 stars_list를 stars라는 변수에 저장합니다.
  // 4. for 문을 활용하여 stars 배열의 요소를 차례대로 조회합니다.
  // 5. stars[i] 요소의 name, url, img_url, recent, like 키 값을 활용하여 값을 조회합니다.
  // 6. 영화인 카드를 만듭니다.
  // 7. #star-box에 temp_html을 붙입니다.
  $.ajax({
      type: 'GET',
      url:  '/api/list',
      data: {},
      success: function (response) {
        let msg = response['msg'];
        alert(msg);
      }
```

```
  });
}
```

👉 새로고침했을 때, 'list 연결되었습니다!' 내용의 alert가 뜨면 연결에 성공한 것입니다!

▼ 2) 서버부터 만들기

💡 **API = 약속**이라고 했습니다! 이 약속을 서버에서부터 시작해보겠습니다.

영화인 정보 전체를 조회하기 위해 서버가 받을 정보는 없습니다.

따라서 서버 로직은 다음 단계로 구성되어야 합니다.
1. mystar 목록 전체를 검색합니다. ID는 제외하고 like 가 많은 순으로 정렬합니다.
2. 성공하면 success 메시지와 함께 stars_list 목록을 클라이언트에 전달합니다.

```
@app.route('/api/list', methods=['GET'])
def stars_list():
    # 1. mystar 목록 전체를 검색합니다. ID는 제외하고 like 가 많은 순으로 정렬합니다.
    # 참고) find({},{'_id':False}), sort()를 활용하면 굿!
    stars = list(db.mystar.find({},{'_id':False}).sort('like',-1))
    # 2. 성공하면 success 메시지와 함께 stars_list 목록을 클라이언트에 전달합니다.
    return jsonify({'result': 'success','stars_list':stars})
```

▼ 3) 클라이언트 만들기

💡 **API = 약속**이라고 했습니다! 이제는 클라이언트를 만들어보겠습니다.

영화인 정보 전체를 조회하기 위해 클라이언트가 전달할 정보는 없습니다.

따라서 클라이언트 로직은 다음 단계로 구성되어야 합니다.
1. #star_box의 내부 html 태그를 모두 삭제합니다.
2. 서버에 1) GET 방식으로, 2) /api/list 라는 주소로 stars_list를 요청합니다.
3. 서버가 돌려준 stars_list를 stars라는 변수에 저장합니다.
4. for 문을 활용하여 stars 배열의 요소를 차례대로 조회합니다.
5. stars[i] 요소의 name, url, img_url, recent, like 키 값을 활용하여 값을 조회합니다.
6. 영화인 카드를 만듭니다.
7. #star-box에 temp_html을 붙입니다.

```
function show_star(){
  // 1. #star_box의 내부 html 태그를 모두 삭제합니다.
  $('#star-box').empty()

  // 2. 서버에 1) GET 방식으로, 2) /api/list 라는 주소로 star_list를 요청합니다.
  $.ajax({
      type: "GET",
      url: "/api/list",
      data: {},
      success: function (response) {
          // 3. 서버가 돌려준 star_list를 star라는 변수에 저장합니다.
          let stars = response['stars_list']
          // 4. for 문을 활용하여 star 배열의 요소를 차례대로 조회합니다.
          for (let i = 0; i < stars.length; i++) {
              let star = stars[i]
              // 5. star[i] 요소의 name, url, img_url, recent, like 키 값을 활용하여 값을 조회합니다.
              let name = star['name']
              let url = star['url']
              let img_url = star['img_url']
              let recent = star['recent']
              let like = star['like']

              // 6. 영화인 카드를 만듭니다.
              let temp_html = `<div class="card">
                                  <div class="card-content">
```

```
                                <div class="media">
                                  <div class="media-left">
                                    <figure class="image is-48x48">
                                      <img
                                        src="${img_url}"
                                        alt="Placeholder image"
                                      />
                                    </figure>
                                  </div>
                                  <div class="media-content">
                                    <a href="${url}" target="_blank" class="star-name title is-4">${name} (좋아요: ${l
                                    <p class="subtitle is-6">${recent}</p>
                                  </div>
                                </div>
                              </div>
                              <footer class="card-footer">
                                <a href="#" onclick="like_star('${name}')" class="card-footer-item has-text-info">
                                  위로!
                                  <span class="icon">
                                    <i class="fas fa-thumbs-up"></i>
                                  </span>
                                </a>
                                <a href="#" onclick="delete_star('${name}')" class="card-footer-item has-text-danger"
                                  삭제
                                  <span class="icon">
                                    <i class="fas fa-ban"></i>
                                  </span>
                                </a>
                              </footer>
                            </div>`

                // 7. #star-box에 temp_html을 붙입니다.
                $('#star-box').append(temp_html)
            }
        }
    });
}
```

▼ 4) 완성 확인하기

index.html을 새로고침 했을 때 영화인 정보가 조회되는지 확인합니다.

[1.5시간] : 마이 페이보릿 무비스타 POST 완성하기

▼ 7) POST 연습: 좋아요 +1

> ✅ **우리가 만들 API 세 가지**
>
> 1) 조회: 영화인 정보 전체를 조회
> **2) 좋아요: 클라이언트에서 받은 이름(name_give)으로 찾아서 좋아요(like)를 증가**
> 3) 삭제: 클라이언트에서 받은 이름(name_give)으로 영화인을 찾고, 해당 영화인을 삭제

▼ 1) 클라이언트와 서버 연결 확인하기

아래 클라이언트/서버 시작 코드는 쌍을 이루고 있습니다.

[서버 시작 코드]

```
@app.route('/api/like', methods=['POST'])
def star_like():
    # 1. 클라이언트가 전달한 name_give를 name_receive 변수에 넣습니다.
    # 2. mystar 목록에서 find_one으로 name이 name_receive와 일치하는 star를 찾습니다.
    # 3. star의 like 에 1을 더해준 new_like 변수를 만듭니다.
    # 4. mystar 목록에서 name이 name_receive인 문서의 like 를 new_like로 변경합니다.
    # 참고: '$set' 활용하기!
    # 5. 성공하면 success 메시지를 반환합니다.
    return jsonify({'result': 'success','msg':'like 연결되었습니다!'})
```

[클라이언트 시작 코드]

```
function like_star(name){
  // 1. 서버에 1) POST 방식으로, 2) /api/like 라는 url에, 3) name_give라는 이름으로 name을 전달합니다.
  // 참고) POST 방식이므로 data: {'name_give': name} 과 같은 양식이 되어야합니다!
```

```
  // 2. '좋아요 완료!' 얼럿을 띄웁니다.
  // 3. 변경된 정보를 반영하기 위해 새로고침합니다.
  $.ajax({
      type: 'POST',
      url:  '/api/like',
      data: {},
      success: function (response) {
          if (response['result'] == 'success') {
            let msg = response['msg'];
            alert(msg);
          }
      }
  });
}
```

👉 '위로' 버튼을 눌렀을 때, 'like 연결되었습니다!' 내용의 alert가 뜨면 연결에 성공한 것입니다!

▼ 2) 서버부터 만들기

💡 **API = 약속**이라고 했습니다! 이 약속을 서버에서부터 시작해보겠습니다.

좋아요 수를 증가시키기 위해 서버가 필요한 정보는 다음과 같습니다.
1. 영화인의 이름 (name_give라고 클라이언트가 전달)

따라서 서버 로직은 다음 단계로 구성되어야 합니다.
1. 클라이언트가 전달한 name_give를 name_receive 변수에 넣습니다.
2. mystar 목록에서 find_one으로 name이 name_receive와 일치하는 star를 찾습니다.
3. star의 like 에 1을 더해준 new_like 변수를 만듭니다.
4. mystar 목록에서 name이 name_receive인 문서의 like 를 new_like로 변경합니다.

```
@app.route('/api/like', methods=['POST'])
def star_like():
    # 1. 클라이언트가 전달한 name_give를 name_receive 변수에 넣습니다.
    name_receive = request.form['name_give']

    # 2. mystar 목록에서 find_one으로 name이 name_receive와 일치하는 star를 찾습니다.
    star = db.mystar.find_one({'name':name_receive})
    # 3. star의 like 에 1을 더해준 new_like 변수를 만듭니다.
    new_like = star['like']+1

    # 4. mystar 목록에서 name이 name_receive인 문서의 like 를 new_like로 변경합니다.
    # 참고: '$set' 활용하기!
    db.mystar.update_one({'name':name_receive},{'$set':{'like':new_like}})

    # 5. 성공하면 success 메시지를 반환합니다.
    return jsonify({'result': 'success'})
```

▼ 3) 클라이언트 만들기

💡 **API = 약속**이라고 했습니다! 이제는 클라이언트를 만들어보겠습니다.

좋아요 수를 증가시키기 위해 클라이언트가 전달할 정보는 다음과 같습니다.
1. 영화인의 이름 (name_give라고 클라이언트가 전달)

따라서 클라이언트 로직은 다음 단계로 구성되어야 합니다.
1. 서버에 1) POST 방식으로, 2) /api/like 라는 url에, 3) name_give라는 이름으로 name을 전달합니다. (참고) POST 방식이므로 data: {'name_give': name} 과 같은 양식이 되어야합니다!
2. '좋아요 완료!' 얼럿을 띄웁니다.
3. 변경된 정보를 반영하기 위해 새로고침합니다.

```
function like_star(name){
    // 1. 서버에 1) POST 방식으로, 2) /api/like 라는 url에, 3) name_give라는 이름으로 name을 전달합니다.
```

```
    // 참고) POST 방식이므로 data: {'name_give': name} 과 같은 양식이 되어야합니다!
    $.ajax({
        type: "POST",
        url: "/api/like",
        data: { 'name_give':name },
        success: function (response) {
            if (response['result'] == 'success') {
                // 2. '좋아요 완료!' 얼럿을 띄웁니다.
                alert('좋아요 완료!')
                // 3. 변경된 정보를 반영하기 위해 새로고침합니다.
                window.location.reload()
            }
        }
    });
}
```

▼ 4) 완성 확인하기

'위로' 버튼을 눌렀을 때 좋아요가 증가하고 영화인 카드의 순위가 변경되는지 확인합니다.

▼ 8) POST 연습: 영화인 삭제하기

✅ **우리가 만들 API 세 가지**

1) 조회: 영화인 정보 전체를 조회
2) 좋아요: 클라이언트에서 받은 이름(name_give)으로 찾아서 좋아요(like)를 증가
3) 삭제: 클라이언트에서 받은 이름(name_give)으로 영화인을 찾고, 해당 영화인을 삭제

▼ 1) 클라이언트와 서버 연결 확인하기

아래 클라이언트/서버 시작 코드는 쌍을 이루고 있습니다.

[서버 시작 코드]

```
@app.route('/api/delete', methods=['POST'])
def star_delete():
    # 1. 클라이언트가 전달한 name_give를 name_receive 변수에 넣습니다.
    # 2. mystar 목록에서 delete_one으로 name이 name_receive와 일치하는 star를 제거합니다.
    # 3. 성공하면 success 메시지를 반환합니다.
    return jsonify({'result': 'success','msg':'delete 연결되었습니다!'})
```

[클라이언트 시작 코드]

```
function delete_star(name){
    // 1. 서버에 1) POST 방식으로, 2) /api/delete 라는 url에, 3) name_give라는 이름으로 name을 전달합니다.
    // 참고) POST 방식이므로 data: {'name_give': name} 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);
            }
        }
    });
}
```

👉 '삭제' 버튼을 눌렀을 때, 'delete 연결되었습니다!'라는 alert가 뜨면 연결에 성공한 것입니다!

▼ 2) 서버부터 만들기

💡 **API = 약속**이라고 했습니다! 이 약속을 서버에서부터 시작해보겠습니다.

영화인 카드를 삭제하기 위해 필요한 정보는 다음과 같습니다.
1. 영화인의 이름 (name_give라고 클라이언트가 전달)

따라서 서버 로직은 다음 단계로 구성되어야 합니다.
1. 클라이언트가 전달한 name_give를 name_receive 변수에 넣습니다.
2. mystar 에서 delete_one으로 name이 name_receive와 일치하는 star를 제거합니다.
3. 성공하면 success 메시지를 반환합니다.

```
@app.route('/api/delete', methods=['POST'])
def star_delete():
    # 1. 클라이언트가 전달한 name_give를 name_receive 변수에 넣습니다.
    name_receive = request.form['name_give']
    # 2. mystar 목록에서 delete_one으로 name이 name_receive와 일치하는 star를 제거합니다.
    db.mystar.delete_one({'name':name_receive})
    # 3. 성공하면 success 메시지를 반환합니다.
    return jsonify({'result': 'success'})
```

▼ 3) 클라이언트 만들기

💡 **API = 약속**이라고 했습니다! 이제는 클라이언트를 만들어보겠습니다.

영화인 카드를 삭제하기 위해 필요한 정보는 다음과 같습니다.
1. 영화인의 이름 (name_give라고 클라이언트가 전달)

따라서 클라이언트 로직은 다음 단계로 구성되어야 합니다.
1. 서버에 1) POST 방식으로, 2) /api/delete 라는 url에, 3) name_give라는 이름으로 name을 전달합니다. (참고) POST 방식이므로 data: {'name_give': name} 과 같은 양식이 되어야합니다!
2. '삭제 완료! 안녕!' 얼럿을 띄웁니다.
3. 변경된 정보를 반영하기 위해 새로고침합니다.

```
function delete_star(name){
    // 1. 서버에 1) POST 방식으로, 2) /api/delete 라는 url에, 3) name_give라는 이름으로 name을 전달합니다.
    // 참고) POST 방식이므로 data: {'name_give': name} 과 같은 양식이 되어야합니다!
    $.ajax({
        type: "POST",
        url: "/api/delete",
        data: { 'name_give':name },
        success: function (response) {
            if (response['result'] == 'success') {
                // 2. '삭제 완료! 안녕!' 얼럿을 띄웁니다.
                alert('삭제 완료! 안녕!')
                // 3. 변경된 정보를 반영하기 위해 새로고침합니다.
                window.location.reload()
            }
        }
    });
}
```

▼ 4) 완성 확인하기

삭제 버튼을 눌렀을 때 영화인 카드가 삭제되는지 확인합니다.

▼ 9) 전체 완성 코드

▼ index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>프론트-백엔드 연결 마지막 예제!</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
```

```
<link
  rel="stylesheet"
  href="https://cdn.jsdelivr.net/npm/bulma@0.8.0/css/bulma.min.css"
/>
<script
  defer
  src="https://use.fontawesome.com/releases/v5.3.1/js/all.js"
></script>
<style>
  .make-center {
    text-align: center;
  }
  .star-list {
    width: 500px;
    margin: 20px auto 0 auto;
  }
  .star-name {
      display: inline-block;
  }
  .star-name:hover {
      text-decoration: underline;
  }
  .card {
      margin-bottom: 15px;
  }
</style>
<script>
  $(document).ready(function() {
    // index.html 로드가 완료되면 자동으로 show_star() 함수를 호출합니다.
    show_star();
  });

  function show_star(){
    // 1. #star_box의 내부 html 태그를 모두 삭제합니다.
    $('#star-box').empty()

    // 2. 서버에 1) GET 방식으로, 2) /api/list 라는 주소로 star_list를 요청합니다.
    $.ajax({
        type: "GET",
        url: "/api/list",
        data: {},
        success: function (response) {
            // 3. 서버가 돌려준 star_list를 star라는 변수에 저장합니다.
            let stars = response['stars_list']
            // 4. for 문을 활용하여 star 배열의 요소를 차례대로 조회합니다.
            for (let i = 0; i < stars.length; i++) {
                let star = stars[i];
                // 5. star[i] 요소의 name, url, img_url, recent, like 키 값을 활용하여 값을 조회합니다.
                let name = star['name']
                let url = star['url']
                let img_url = star['img_url']
                let recent = star['recent']
                let like = star['like']

                // 6. 영화인 카드를 만듭니다.
                let temp_html = `<div class="card">
                                  <div class="card-content">
                                    <div class="media">
                                      <div class="media-left">
                                        <figure class="image is-48x48">
                                          <img
                                            src="${img_url}"
                                            alt="Placeholder image"
                                          />
                                        </figure>
                                      </div>
                                      <div class="media-content">
                                        <a href="${url}" target="_blank" class="star-name title is-4">${name} (좋아요
                                        <p class="subtitle is-6">${recent}</p>
                                      </div>
                                    </div>
                                  </div>
                                  <footer class="card-footer">
                                    <a href="#" onclick="like_star('${name}')" class="card-footer-item has-text-inf
                                      위로!
                                      <span class="icon">
                                        <i class="fas fa-thumbs-up"></i>
                                      </span>
                                    </a>
                                    <a href="#" onclick="delete_star('${name}')" class="card-footer-item has-text-d
                                      삭제
                                      <span class="icon">
                                        <i class="fas fa-ban"></i>
                                      </span>
                                    </a>
                                  </footer>
                                </div>`
```

```
                // 7. #star-box에 temp_html을 붙입니다.
                $('#star-box').append(temp_html)
            }
        }
      });
    }

    function like_star(name){
        // 1. 서버에 1) POST 방식으로, 2) /api/like 라는 url에, 3) name_give라는 이름으로 name을 전달합니다.
        // 참고) POST 방식이므로 data: {'name_give': name} 과 같은 양식이 되어야합니다!
        $.ajax({
            type: "POST",
            url: "/api/like",
            data: { 'name_give':name },
            success: function (response) {
                if (response['result'] == 'success') {
                    // 2. '좋아요 완료!' 얼럿을 띄웁니다.
                    alert('좋아요 완료!')
                    // 3. 변경된 정보를 반영하기 위해 새로고침합니다.
                    window.location.reload()
                }
            }
        });
    }

    function delete_star(name){
        // 1. 서버에 1) POST 방식으로, 2) /api/delete 라는 url에, 3) name_give라는 이름으로 name을 전달합니다.
        // 참고) POST 방식이므로 data: {'name_give': name} 과 같은 양식이 되어야합니다!
        $.ajax({
            type: "POST",
            url: "/api/delete",
            data: { 'name_give':name },
            success: function (response) {
                if (response['result'] == 'success') {
                    // 2. '삭제 완료! 안녕!' 얼럿을 띄웁니다.
                    alert('삭제 완료! 안녕!')
                    // 3. 변경된 정보를 반영하기 위해 새로고침합니다.
                    window.location.reload()
                }
            }
        });
    }

  </script>
</head>
<body>
  <section class="hero is-warning">
    <div class="hero-body">
      <div class="container make-center">
        <h1 class="title">
          마이 페이보릿 무비스타😆
        </h1>
        <h2 class="subtitle">
          순위를 매겨봅시다
        </h2>
      </div>
    </div>
  </section>
  <div class="star-list" id="star-box">
    <div class="card">
      <div class="card-content">
        <div class="media">
          <div class="media-left">
            <figure class="image is-48x48">
              <img
                src="https://search.pstatic.net/common/?src=https%3A%2F%2Fssl.pstatic.net%2Fsstatic%2Fpeople%2Fportra
                alt="Placeholder image"
              />
            </figure>
          </div>
          <div class="media-content">
            <a href="https://movie.naver.com//movie/bi/pi/basic.nhn?st=1&code=397373" target="_blank" class="star-nam
            <p class="subtitle is-6">안녕, 나의 소울메이트(가제)</p>
          </div>
        </div>
      </div>
      <footer class="card-footer">
        <a href="#" onclick="like_star('김다미')" class="card-footer-item has-text-info">
          위로!
          <span class="icon">
            <i class="fas fa-thumbs-up"></i>
          </span>
        </a>
        <a href="#" onclick="delete_star('김다미')" class="card-footer-item has-text-danger">
          삭제
          <span class="icon">
```

```
              <i class="fas fa-ban"></i>
            </span>
          </a>
        </footer>
      </div>
    </div>
  </body>
</html>
```

▼ app.py

```
from pymongo import MongoClient

from flask import Flask, render_template, jsonify, request
app = Flask(__name__)

client = MongoClient('localhost', 27017)
db = client.dbsparta


# HTML을 주는 부분
@app.route('/')
def home():
    return render_template('index.html')

# API 역할을 하는 부분
@app.route('/api/list', methods=['GET'])
def star_list():
    # 1. mystar 목록 전체를 검색합니다. ID는 제외하고 like 가 많은 순으로 정렬합니다.
    stars = list(db.mystar.find({},{'_id':False}).sort('like',-1))
    # 2. 성공하면 success 메시지와 함께 stars 목록을 클라이언트에 전달합니다.
    return jsonify({'result': 'success','stars_list':stars})


@app.route('/api/like', methods=['POST'])
def star_like():
    # 1. 클라이언트가 전달한 name_give를 name_receive 변수에 넣습니다.
    name_receive = request.form['name_give']

    # 2. mystar 목록에서 find_one으로 name이 name_receive와 일치하는 star를 찾습니다.
    star = db.mystar.find_one({'name':name_receive})
    # 3. star의 like 에 1을 더해준 new_like 변수를 만듭니다.
    new_like = star['like']+1

    # 4. mystar 목록에서 name이 name_receive인 문서의 like 를 new_like로 변경합니다.
    # 참고: '$set' 활용하기!
    db.mystar.update_one({'name':name_receive},{'$set':{'like':new_like}})

    # 5. 성공하면 success 메시지를 반환합니다.
    return jsonify({'result': 'success'})


@app.route('/api/delete', methods=['POST'])
def star_delete():
    # 1. 클라이언트가 전달한 name_give를 name_receive 변수에 넣습니다.
    name_receive = request.form['name_give']
    # 2. mystar 목록에서 delete_one으로 name이 name_receive와 일치하는 star를 제거합니다.
    db.mystar.delete_one({'name':name_receive})
    # 3. 성공하면 success 메시지를 반환합니다.
    return jsonify({'result': 'success'})


if __name__ == '__main__':
    app.run('0.0.0.0', port=5000, debug=True)
```

후반 3시간

[월수/화목반] : 체크인 & 출석체크

> 🔥 **튜터님은 체크인과 함께 출석 체크(링크)를 진행해주세요!**

▼ "15초 체크인"을 진행합니다.

- 튜터는 타이머를 띄워주세요! (링크)
- 본인의 감정상태와 오늘 있었던 소소한 일을 공유하는 시간입니다.

[1.5시간]: 프로젝트 개요 & 목표 설정

▼ 10) 프로젝트 1차 기획안을 작성 → 발표해보기

> 👉 일단 말하고 나면, 이루어진답니다! :-)
> 우리가 앞으로 3주 동안 지켜볼 나침반 역할을 해줄거에요!

- **공유 가능한 블로그(네이버 블로그 추천!)에 사진+글로 정리하고, 슬랙에 공유**합니다!
  - 기획안을 바탕으로 6~8주차에 계속해서 개발일지를 작성할 예정입니다.
  - 본인이 익숙한, 슬랙에 공개 가능한 블로그(Velog, Medium) 등이 있으면 사용하셔도 무방합니다.

[1.5시간]: 튜터 개별 피드백

▼ 11) 개발 범위 & 우선순위

- 튜터와 10분 씩 개별 면담을 통해 프로젝트의 **1) 개발 범위**와 **2) 기능 우선순위**에 대해 함께 검토해서 **3) 다음 시간까지 해올 Todo 리스트***를 선정합니다.

▼ **Todo 리스트*** 예시

> 👉 **예시! 입니다. 아래와 같이 할 일을 정리해보기로 해요!**
> 한번 슥 읽어보면 "뭐부터 해야할까?" 정리하는 데 도움이 될거예요!

▼ 한걸음더 페이지를 보고 필요한 기술이 있는지 확인해보기

🚶 한 걸음 더

▼ 프로젝트와 관련 있는 API 사용법을 공부해오기

- 공공데이터포털, 서울 열린데이터광장, 네이버, 카카오, 구글, 유튜브, 페이스북, ... 등등의 API 사용법을 유튜브 영상 또는 구글링으로 공부해오기
- 링크에서 내가 필요한 서비스가 API를 제공하는지 확인해봅니다. (Behance, Gmail, Google Analytics, Riot Games, Dota 2, Covid-19 등등 무궁무진한 API가 있습니다!)

▼ 크롤러가 필요하다면 어떻게 만들지 구상해오기

- 예를 들어 면세점 정보를 모아서 종합 할인가를 보여주는 서비스를 만들기 위해서는 Selenium 이라는 패키지의 사용법을 알아보아야 합니다. (위의 '한 걸음 더' 페이지를 참고하세요!)

▼ 특정 기술이 필요하다면 그에 관해 조사해오기

- 그 외 프로젝트에 필요한 기술이 있다면 조사해오고, 어떻게 공부해서 사용할지에 대한 계획이 필요합니다!

[끝]

▼ "15초 체크아웃"을 진행합니다.

- 튜터는 타이머를 띄워주세요! (링크)
- 다음 시간까지 해올 Todo 리스트에 대한 계획을 말해도 좋습니다!

숙제 & 설치

[숙제 1] - 프로젝트 기획안 완성해오기!

1. 튜터 피드백을 바탕으로 최종 프로젝트 기획안을 완성해옵니다.
2. 최종 프로젝트 기획안에는 다음 내용을 넣어주세요.

✅ 스파르타코딩클럽 8기 프로젝트 기획안

1. 프로젝트 이름/설명
2. 프로젝트 생김새
3. 개발해야 하는 기능 (우선순위별 정렬)
4. 다음 시간까지 해야 하는 Todo 리스트

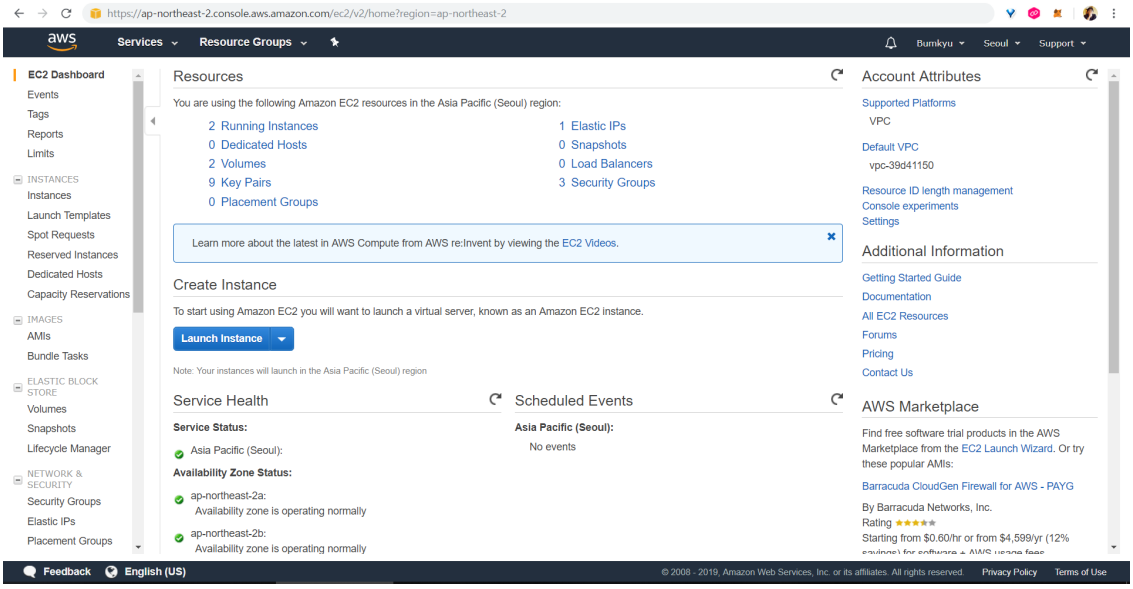
3. 공유 가능한 블로그에 사진+글로 정리하고, 슬랙에 공유합니다!

[가입/설치] - 다음 시간을 위해 미리 가입/설치해와야 할 것들

1. AWS 가입하기 (승인까지 최대 24시간이 걸리니, 미리 해주세요!)
   - 가입: https://console.aws.amazon.com/console/home
   - 해외결제가 가능한 유효한 결제 수단을 넣어야 가입이 정상적으로 이루어집니다. Visa 또는 Master 겸용의 신용카드를 추천드립니다. 가입이 정상적으로 이루어지지 않을 경우 6주차 이후 수업을 진행할 수 없으므로 사전에 해외결제가 가능한지 반드시 확인 부탁드립니다.
   - AWS는 개인에게 클라우드 환경의 가상서버를 제공합니다. 기본 사양의 서버(EC2)를 1년 동안 무료로 사용할 수 있으며, 이후 월 1만 원정도의 금액이 결제될 수 있습니다.
   - 가입 시 결제된 금액은 다시 반환됩니다. (일종의 결제 테스트 목적)

   ▼ 가입 후 아래와 같은 화면에 접속 하면 성공!

   - https://ap-northeast-2.console.aws.amazon.com/ec2/v2/home?region=ap-northeast-2



2. Filezilla 설치하기
   - 다운로드: https://filezilla-project.org/download.php
   - 다운로드 클릭 후 가장 기본 버전 (스크린샷 기준 왼쪽) 다운로드!

Please select your edition of FileZilla Client

	FileZilla	FileZilla with manual	FileZilla Pro
Standard FTP	Yes	Yes	Yes
FTP over TLS	Yes	Yes	Yes
SFTP	Yes	Yes	Yes
Comprehensive PDF manual	-	Yes	Yes
Amazon S3	-	-	Yes
Backblaze B2	-	-	Yes
Dropbox	-	-	Yes
Microsoft OneDrive	-	-	Yes
Google Drive	-	-	Yes
Google Cloud Storage	-	-	Yes
Microsoft Azure Blob and File Storage	-	-	Yes
WebDAV	-	-	Yes
OpenStack Swift	-	-	Yes
Box	-	-	Yes
Site Manager synchronization	-	-	Yes
	Download	Select	Select