⑦（発展）Web APIの提供

さらなる発展課題です。ここまでの全ての手順をクリアした方は、トライしてみましょう

「いいね」機能の更なる改善

- 今のサンプルは「いいね」リンクを押す度に、画面遷移が起こってしまいます

- 本当は「いいね」リンクを押しても、数字だけが書き換わり、画面が切り替わらないほうが親切です
- このことは「いいね」の機能をREST API化し、前期に学習したfetch APIを使えば実現できます

(再掲)REST (Representational State Transfer)

- 近年では、URLをリソースの識別子(ID)ととらえ、そのリソースに対して各HTTPメソッドの意味に沿った機能を提供する Web API の設計が主流です
  - 例: Googleカレンダ
    - GET : イベントの情報(日時や参加者など)を取得
    - PUT : イベントの情報を更新
    - POST : 新規のイベントをカレンダ上に追加
    - DELETE : イベントをカレンダから削除

- このような設計で作られた Web API を REST API と呼びます
  - 上記は厳密ではありません; 興味のある方はフィールディングの論文を読んでみてください
    - https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

手順

1. URLディスパッチャへのパターンを追加する
   - → `blog/urls.py`
2. APIを処理するControllerを追加する
   - → `api/views.py`
3. JavaScriptを追加する
   - → `blog/static/js/index.js`
4. Viewを修正する
   - → `blog/templates/blog/index.html`

1. URLディスパッチャの追加：blog/urls.py

- URLディスパッチャが、以下のURLを受け付けるようにします
  - http://127.0.0.1:8000/api/articles/123/like
  - → views.api_like を呼び出す

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
...
    path('<int:article_id>/update', views.update, name='update'),
    path('<int:article_id>/like', views.like, name='like'),
    path('api/articles/<int:article_id>/like', views.api_like),
]
```

2. Controllerへの処理の追加：blog/views.py

- blog/views.py の最後に、views.api_like 関数を定義する
  1. Article モデルから与えられたIDに対応したオブジェクトを取り出す
  2. オブジェクトの変数likeを変更し、保存する
  3. 結果をJSONレスポンスとして返す

```
from django.shortcuts import render, redirect
from django.http import HttpResponse
from django.http import Http404, JsonResponse
from django.utils import timezone
import random
from blog.models import Article, Comment
...
```

```
...
def api_like(request, article_id):
    try:
        article = Article.objects.get(pk=article_id)
        article.like += 1
        article.save()
    except Article.DoesNotExist:
        raise Http404("Article does not exist")
    result = {
        'id' : article_id,
        'like' : article.like
    }
    return JsonResponse(result)
```

JSONについて

- JSON (JavaScript Object Notation)
  - もとは、JavaScriptにおけるオブジェクトの表記法をベースとしたデータの記述方法
  - JavaScriptの配列やオブジェクトの書き方だが、オブジェクトのプロパティは文字列でなければならない
  - JavaScriptにおいて簡単に扱える

- 正しく実装できていれば、以下のURLをブラウザで開くと右図のような結果が返る
  - http://127.0.0.1:8000/api/articles/ID/like
  - ※IDには実在する記事のIDを指定すること

```
{"id": 1, "like": 3}
```

3. JavaScriptの追加

- blog/static/blog/js/index.js を作成し、以下のJavaScriptを記述する

API呼び出しが完了した時に呼ばれるコールバック関数
HTML中の'like1','like2'…といったIDをもつ要素の内容をレスポンス中のlikeの値で置き替える

```
function callback(json) {
    let element = document.getElementById('like' + json.id);
    element.textContent = json.like;
}

function like(article_id) {
    fetch('/api/articles/' + article_id + '/like')
    .then(response => response.json())
    .then(callback)
}
```

関数が呼ばれた際に、今回作成したAPIを呼び出す

4. テンプレートの修正

- blog/templates/blog/index.html を以下のように編集する
  - JavaScript読み込みのタグを追加
  - 「いいね」リンクをJavaScriptのlike関数への呼び出しに変更

```
{% load static %}
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <link href="{% static 'blog/css/default.css' %}" rel="stylesheet">
    <script type="text/javascript" src="{% static 'blog/js/index.js' %}" ></script>
  </head>
...
```

```
      <div>
        <a onclick="like({{article.id}});">
          Like: <span id="like{{article.id}}">{{ article.like }}</span>
        </a>
      </div>
```