Day5 (ORM with View) - CRUD

구분	Django
과목	

오늘 우리는 게시판 기능을 구현할 것이다.

게시글을 CRUD 하는 것을 연습해 볼 것이다.

todo_list_pjt.zip

오늘 실습을 위한 스켈레톤 코드이다.

가상환경 활성화, migrate 그리고 admin 계정도 새로 생성하자.

admin 페이지로 들어가 보자. admin 페이지에 접속 후

todo 게시글을 3개를 수동으로 만들어 놓고 시작하자.

가장 먼저, 단일 게시글을 볼 수 있는 기능을 (조회) 구현 해보자.

URL > VIEW > TEMPLATE 순으로 작성해 보자.

각 단일 데이터는 고유의 id(pk)가 있을 것이다.

http://127.0.0.1:8000/todos/1/

http://127.0.0.1:8000/todos/2/

http://127.0.0.1:8000/todos/3/ 이러한 형식으로 URL을 받을 것이며,

1,2,3은 각 게시글에 대한 PK번호(고유번호)가 될 것이다.

detail 함수의 경로를 수정하자.

todos > urls.py

```
path('<int:todo_pk>/', views.detail, name='detail'),
```

todos > views.py

```
from django.shortcuts import render
from .models import Todo
```

```
def detail(request, todo_pk):
    todo=Todo.objects.get(pk=todo_pk)
    context = {
        'todo': todo
    }
    return render(request, 'todos/detail.html', context)
```

todos > detail.html

```
{% extends 'base.html' %}

{% block content %}

<h1>{{ todo.work }}의 상세 페이지 입니다.</h1>
<hr />

<p>할 일 : {{ todo.work }}</p>
<p>내 용 : {{ todo.content }}</p>
<p>일 자 : {{ todo.created_at }}</p>
<p>완 료 : {{ todo.is_completed }}</p>
<hr />

<a href="{% url 'todos:index' %}">[back]</a>
{% endblock %}
```

위 소스코드들의 흐름을 잘 이해하고 기억하자.

1. 먼저 urls.py 파일에서는 각 게시글의 pk 번호를 이용한 URL 경로를 지정하기 위해서 variable routing을 사용하였고
2. views.py 파일에서는 todo=Todo.objects.get(pk=todo_pk) 를 살펴보자.

   todo라는 변수에 QuerySetApi를 이용해서 DB에 있는 특정 데이터를 저장했다.

   그리고 해당 내용을 detail.html 파일로 보냈다.
3. detail.html 파일에서는 해당 내용을 출력했다.

그 다음 게시글의 리스트를 볼 수 있는 페이지를 완성하자.

http://127.0.0.1:8000/todos/ 형식으로 요청이 들어오면

모~든 todo 게시글이 보일 수 있도록 만들어보자.

URL > VIEW > TEMPLATE 순으로 작성해 보자.

todos > urls.py

```
path('', views.index, name='index'),
```

따로 건들 것이 없다.

todos > views.py

```
from django.shortcuts import render
from . models import Todo

def index(request):
    works=Todo.objects.all()
    context = {
        'todo_list': works
    }
    return render(request, 'todos/index.html', context)
```

이번에는 DB에 단일 데이터를 조회 하는것이 아니라

Todo 테이블에 있는 모든 게시글을 조회하기 위해서 Todo.objects.all() 을 사용하였다.

즉, works라는 변수에 Todo 모델에 있는 모든 정보를 담은 후 context를 이용해서 template로 보낸다.

todos > index.html 파일을 수정하자.

```
{% extends 'base.html' %}

{% block content %}
  <h1>할 일 목록 상세 페이지</h1>

  <ul>
    {% for todo in todo_list %}
      <li>
        <a href="{% url "todos:detail" todo.pk %}">{{ todo.work }}</a>
        <p>완료 여부 : {{ todo.is_completed }}</p>
      </li>
    {% empty %}
      <li>아직 등록 된 할 일이 없습니다.</li>
    {% endfor %}

  </ul>

{% endblock %}
```

for 태그는 추가적으로 {% empty %} 태그도 지원하는데

empty구문은 list가 비어 있을 경우 실행이 되며 for문과 함께 쓰인다.

서버를 켜서 확인해 보자. http://127.0.0.1:8000/todos/

먼저 Read (조회) 기능을 완료 하였으니 그 다음은

게시글 Create(생성) 을 해보자.

게시글을 생성하는 기능을 구현하기 위해서는 2가지 세부 기능이 필요하다.

1. from 태그를 통해서 게시글을 입력받는 기능 (새로운 할 일을 작성)
    - 새로운 할 일을 작성하는 함수는 views.py에 create_todo함수를 이용할 것이다.

2. form을 통해서 입력받은 정보를 Database에 저장하는 기능 (작성된 내용을 DB에 저장)
    - 입력받은 정보를 Database에 저장하는 역할을 하는 new_todo 라는 함수는 이따가 만들 것이다.

먼저, template으로 이동 후 create_todo,html 파일을 수정해보자.
이미 작성이 되어 있는 form코드를 조금 더 그럴 듯 하게 만들어 보자.
제목(work)도 넣고 내용(content)도 따로 넣을 수 있도록 수정하자.

todos > create_todo.html

```
{% extends 'base.html' %}

{% block content %}
 <h1>이곳에 할 일을 생성합니다.</h1>

 <form action="{% url 'todos:new_todo' %}" method="POST">
  {% csrf_token %}

  <label for="work">할일(work) :</label>
  <input type="text" id="work" name="work" /> <br /><br />

  <label for="content">content :</label>
  <textarea name="content" id="content" cols="30" rows="10"></textarea>

  <input type="submit" value="제출하기" />
 </form>
{% endblock %}
```

- form 태그 안에 method="POST" 를 사용하였다. 이미 작성된 게시글을 조회 할 때에는 GET method가 되겠지만 새로 **게시글을 생성**을 요청을 할 때에는 HTTP method 중 POST method를 사용해서 요청을 한다

- 나중에 우리가 form에 직접 작성한 내용(입력이되는 내용)은 new_todo라는 경로의 new_todo 함수로 보낼 것이다.

- {% csrf_token %} 는 Django에서 제공하는 보안 기능이다. 웹에서 CSRF(Cross-Site Request Forgery) 공격을 방지하기 위한 보안 기능이다.

  Django 에서는 클라이언트가 Django 서버로 POST, PUT, DELETE 등의 요청을 보낼 때 보안을 위해서 CSRF 토큰을 반드시 포함해야 한다.

  CSRF(Cross-Site Request Forgery)가 무엇인지? PDF 뒷부분에 설명을 할 것이다. 지금은 Django에서 form을 통해서 post 요청을 보낼때 CSRF 토큰을 함께 포함해야 된다는 사실만 기억하자.

지금까지 게시글을 입력받을 form 기능을 다 구현 했다면 그 다음에는 form 을 통해서 입력받은 정보를 database에 저장을 해야 할 것이다.

이 역할을 해 줄 new_todo 라는 함수를 views.py에 만들어 보자.

(참고로 나~중에는 create_todo 함수랑 new_todo 함수를 합쳐서 하나로 작성할 것이다. ㅎㅎ)

(우리가 아직 학습하지 않았지만 나중에는 form 태그가 아니라 django에서 제공하는 form을 사용해서 사용자 입력을 받을 것이다. 우리는 Django form을 학습하기 전이니까 사용자입력받은 함수 그리고 입력받은 값을 저장하는 함수를 2개를 운영하자.)

todos > urls.py

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


app_name = 'todos'

urlpatterns = [
    path('', views.index, name='index'),
    path('create_todo/', views.create_todo, name='create_todo'),
    path('new_todo/', views.new_todo, name='new_todo'),
    path('<int:todo_pk>/', views.detail, name='detail'),
]
```

new_todo 경로를 추가 했다.

흐름을 살펴보자.

1. create_todo.html의 form 태그를 통해서 작성된 내용은
2. new_todo 함수로 전달했다. (Throw-catch 내용)
3. new_todo 함수는 create_todo에서 작성한 내용을 Database에 저장을 하는 역할을 할 것이다.

todos > views.py

```
from django.shortcuts import render, redirect

def new_todo(request):

    work = request.POST.get('work')
    content = request.POST.get('content')
    is_completed = False

    todo = Todo(work=work, content=content, is_completed=is_completed)
    todo.save()

    return redirect('todos:detail', todo.pk)
```

1. import문에 redirect 함수를 추가하자.
   그리고 create_todo 함수 아래에 new_todo함수를 새로 정의해 주자.
2. create_todo.html 에서 작성한 내용을 각각 work 그리고 content 변수에 저장한다.
3. 각 변수의 내용을 todo 객체의 속성값으로 넣은 다음
4. save() 를 통해서 database에 저장을 하였다.
5. 그리고 해당 detail 페이지가 보도록 경로를 설정했다. (redirect 함수 사용)
   a. redirect 함수는 페이지를 자동으로 이동 시킬 때 사용하는 함수다.

한번 더 정리를 해 보자면 우리는 새로운 글을 create 하기 위해서
두가지의 과정을 거쳤다.

1. 새 글의 내용을 작성할 페이지 [create_todo] → (url + view함수 + template)
2. 수정된 내용을 Datebase에 저장 [new_todo] → (url + view함수)

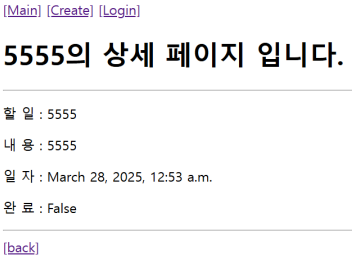
흐름은 다음과 같다.
create_todo.html에서 form을 통해서 사용자 입력을 받고 해당 내용을 csrf_token과 함께
new_todo 함수로 보냈으며
new_todo 함수의 request 매개변수에 저장된 내용을 Todo 클래스의 todo 인스턴스에 담은 후
save()를 통해 database서버로 저장하였다.

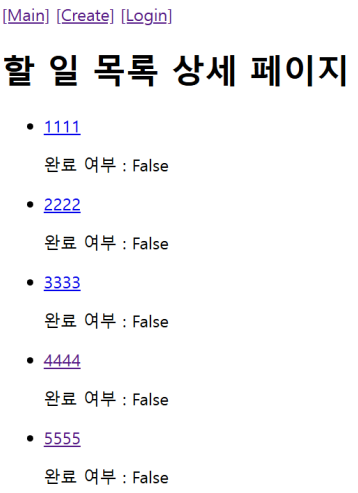
추가적으로,
render랑 redirect랑 혼동하지 말자.

`render` 는 템플릿에서 html파일을 불러올 때 사용하고 context로 값도 같이 넘길 때 사용한다.

`redirect` 는 특정 URL로 이동만 할 때 사용된다.

서버를 켜서 잘 작동 되는지 확인해 보자. [create] 버튼을 누른 후 새 글을 게시해 보자.





지금까지 게시글 조회하는 것을 보았다.

자, 이제 게시글을 삭제하는 기능을 구현해 보자.

todos > urls.py

```
    path('<int:todo_pk>/delete/', views.delete_todo, name='delete_todo'),
```

todos > views.py

```
def delete_todo(request, todo_pk):
    todo = Todo.objects.get(pk=todo_pk)
    todo.delete()
    return redirect('todos:index')
```

아직 템플릿에 삭제 버튼을 달지 않았지만

지우고자 하는 detail 페이지에서 삭제 버튼을 누르면

delete_todo 함수가 실행되어 해당 게시글을 삭제되도록 했다.

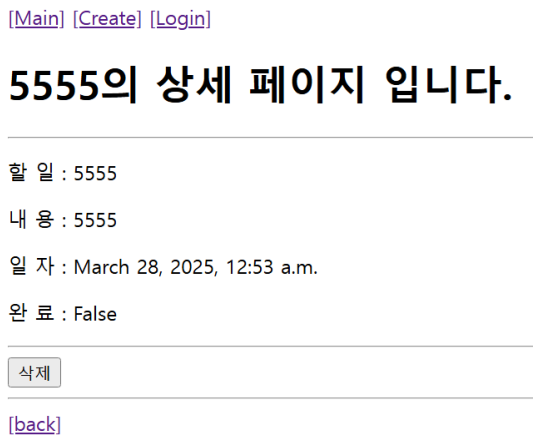
detail.html 파일에 삭제 버튼을 달아보자.

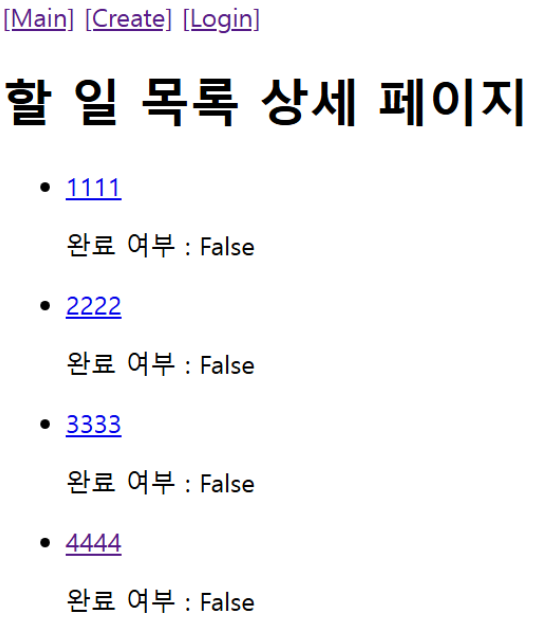
```
 ...
<p>완 료 : {{ todo.is_completed }}</p>
<hr />

<form action="{% url 'todos:delete_todo' todo.pk %}" method="POST">
 {% csrf_token %}
 <input type="submit" value="삭제" />
</form>
<hr />

<a href="{% url 'todos:index' %}">[back]</a>
{% endblock %}
```

Django에서 POST 요청을 보낼 시 항상 CSRF토큰을 달아줘야 하는 것을 잊지 말자!

서버켜서 확인해 보자.

삭제 버튼을 누르면 해당 게시글이 삭제가 될 것이다.





마지막 작성한 할일을 Update (수정) 하는 기능을 추가 해보자.

수정 기능을 구현하기 위해서는 CREATE 기능을 구현했던 것과 마찬가지로 두가지 과정을 거친다.

1. 수정될 내용을 작성
2. 수정된 내용을 Datebase에 저장

두 과정을 각각 나눠서 구현 해 볼 것이다.

todos > urls.py

```
path('<int:todo_pk>/update_todo/', views.update_todo, name='update_todo'),
```

todos > views.py

```
def update_todo(request, todo_pk):
    todo = Todo.objects.get(pk=todo_pk)
    context = {
        'todo': todo
    }
    return render(request, 'todos/update_todo.html', context)
```

1. 아직 구현을 하지 않았지만 detail.html 에 [수정] 버튼을 추가 할 것이고
2. [수정] 버튼을 누르면 해당 게시물의 PK를 views.py에 있는 update_todo 함수로 보낼 것이다.
    - 어떤 개시글을 수정할 것인지 알아야 하기 때문에 수정할 게시글의 PK를 update_todo 함수로 보내는 것이다.
3. update_todo 함수는 수정전! 내용을 DB에서 가져와 update_todo.html 파일로 보낼 것이다.
    - 우리는 [수정] 버튼을 누르면 수정 전 내용을 화면에 먼저 보여줄 것이므로 수정 전 내용을 템플릿으로 보내는 것이다.

1. 먼저 detail.html 가서 [수정] 버튼을 단 후에
2. update_todo.html 파일도 새로 구현해 보자.

1. 먼저 detail.html에 [수정] 버튼을 달아주자.

```
<p>완 료 : {{ todo.is_completed }}</p>
<hr />

<a href="{% url 'todos:update_todo' todo.pk %}"> [수정] </a>
```

2. templates/todos 폴더 안에 update_todo.html 파일을 새로 생성 후

   수정 내용을 작성할 수 있는 form을 만들어 주자.

todos > update_todo.html

```
{% extends 'base.html' %}

{% block content %}
<h1>이 곳에서 할 일을 수정합니다.</h1>

<form action="{% url 'todos:edit_todo' todo.pk %}" method="POST">
 {% csrf_token %}

 <label for="work">work :</label>
 <input type="text" name="work" id="work" value="{{ todo.work }}" /> <br />
 <label for="content">content :</label>
 <textarea name="content" id="work" cols="30" rows="10"> {{ todo.content }} </textarea >

 <input type="submit" value="수정완료" />

</form>
{% endblock %}
```

수정 한 내용을 database에 저장을 해야 할 것이다.

따라서 우리는 `edit_todo` 라는 함수를 views.py에 새로 만들 것이다.

반복해서 이야기 하자면 CREATE 기능을 구현 했을때와 같다.

2가지 프로세스를 통해서 수정 기능을 구현하고 있는 것이다.

1. 수정할 내용을 다시 작성할 페이지 [update_todo] → (url + view함수 + template)
2. 수정된 내용을 Datebase에 저장 [edit_todo] → (url + view함수)

edit_todo 함수의 url 그리고 함수를 작성 해 보자.

todos > urls.py

```
path('<int:todo_pk>/edit_todo/', views.edit_todo, name='edit_todo'),
```

todos > views.py

```
def edit_todo(request, todo_pk):
    todo = Todo.objects.get(pk=todo_pk)

    work = request.POST.get('work')
```

```
    content = request.POST.get('content')

    todo.work = work
    todo.content = content
    todo.save()

    return redirect('todos:detail', todo.pk)
```

todo_pk 의 PK는 update_todo.html 로 부터 전달받은 pk값이다.

다시 흐름을 살펴보자

- detail.html 에서 [수정] 버튼을 누르면
- 수정 게시글의 pk값 → urls.py의 update_todo 경로로 갈 것이고
- 결국 update_todo 함수의 *todo_pk 라는 이름의* 매개변수에
  수정할 게시글의 pk값이 저장 될 것이다.

- update_todo 함수에서 해당 PK의 수정 전 내용들을 update_todo.html 로 전달하면
- update_todo.html 에서 수정 전 내용들이 화면에 보여줄 것인데
- 게시글 관련 값들을 수정 후에는 그 내용을 edit_todo 함수로
  csrf 토큰과 함께 POST요청 보낼 것이다.
- edit_todo 함수에서 database에 저장을 하면서 모든 과정이 마무리가 된다.

지금까지 게시글 CRUD과정을 실습해 보았다.

아래부터는 참고사항 이지만 웹 개발자라면 이해하고 있어야 하는 내용들이다.
설명을 천천히 반복해서 읽어보고 구글링 또는 생성형AI를 이용해서라도 반드시 이해를 하도록 노력해보자.

[참고] HTTP Method

- HTTP
  - 네트워크 상에서 데이터를 주고 받기 위한 약속
- HTTP Method
  - 데이터(리소스)에 어떤 요청(행동)을 원하는지를 나타낸 것

GET

- 특정 리소스를 가져오도록 요청할 때 사용
- 반드시 데이터를 가져올 때만 사용해야 함

- DB에 변화를 주지 않음
- CRUD에서 R 역할을 담당
- GET 방식으로 데이터를 전달하면 Query String 형식으로 보내짐
- 유저가 작성할 데이터 폼을 화면에 띄울떄 GET 사용할 것임

POST

- 서버로 데이터를 전송할 때 사용
- 서버에 변경사항을 만듦
- 리소스를 생성/변경하기 위해 데이터를 HTTP body에 담아 전송
- GET의 쿼리 스트링 파라미터와 다르게 URL로 데이터를 보내지 않음
- CRUD에서 C/U/D 역할을 담당

[참고] CSRF 토큰이란?

먼저 쿠키와 세션의 개념에 대해서 이해를 해야 한다.

[쿠키]

쿠키는 방문하는 웹사이트(서버)가 유저(클라이언트)에게 보내는 작은 텍스트파일 이다.

쿠키는 유저가 방문한 웹사이트 안에서 의 활동을 추적하는데 사용한다.

예를 들어, 쿠키로 유저가 해당 사이트 내에서 “팝업창을 다시는 안본다” 라는 단추에 클릭을 했는지를 웹사이트(서버)는 유저의 쿠키를 통해서 알 수 있다.

프로세스를 보자.

1. 유저(클라이언트)가 웹사이트(서버)에 로그인을 시도했다.
2. 서버는 쿠키를 생성 한다. 그리고 쿠키에 필요한 유저관련 정보들을 담아
   로그인 성공이라는 메시지와 함께 서버가 생성한 쿠키도 함께 응답한다.
3. 유저는 넘겨 받은 쿠키를 저장하고 있다가 다시 해당 서버(사이트)에 요청을 보낼 때, 제공받았던 쿠키를 함께 전송한다.

쿠키에는 인터넷쇼핑 장바구니, 재 방문시 아이디/비번 자동 입력, 오늘 이 창을 다시보지 않기 설정 등

유저의 사이트 안에서 활동을 기억하는 역할을 한다.

[세션]

웹사이트에 한번 로그인 하고 나면 웹사이트 안에서 페이지 이동을 하는 동안에

다시 로그인 할 필요가 없는 것은 “세션” 기술 덕분이다.

세션은 웹 브라우저를 통해 웹서버에 접속한 시점으로부터

웹 브라우저를 종료하여 연결을 끝내는 시점까지

그 상태를 일정하게 유지하는 기술을 세션(session)이라고 한다.

프로세스를 하나씩 살펴 보자.

1. 회원가입이 완료된 유저(클라이언트)가 웹사이트(서버)에 로그인을 시도를 한다.
2. 로그인 요청을 받은 서버는 아이디와 비밀번호가 맞는지 확인을 하고

   로그인에 성공한다면 세션을 생성한다. 이 세션에는 사용자 인증정보 (아이비 비번)을 저장한다.
3. 그리고 저장된 세션 정보를 유저가 사용할 수 있도록 세션 id 라는 것을 생성하고 로그인 성공 메시지와 함께 응답을 한다. 세션id는 유저와 서버와 상호작용 할 때 사용한다.
4. 유저는 브라우저의 쿠키저장소에 응답받은 session id를 저장한다.
5. 유저가 서버에 요청을 보낼 때 마다 클라이언트의 session id가 함께 서버로 전송된다.
6. 서버는 session id로 유저를 식별하고 해당 세션에 연결된 상태정보를 찾을 수 있다.

세션과 쿠키의 차이점은

세션은 서버에 저장되므로 안전한 반면,

쿠키는 유저의 컴퓨터에 저장되어 탈취,변조 위험이 존재한다.

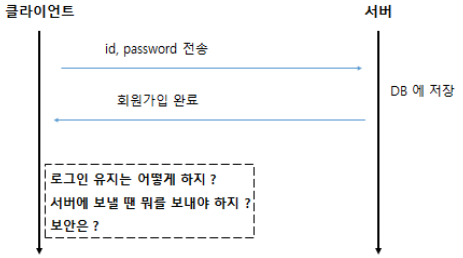
따라서 세션은 보안에 취약한 쿠키를 보완해주는 역할을 해준다고 할 수 있다.

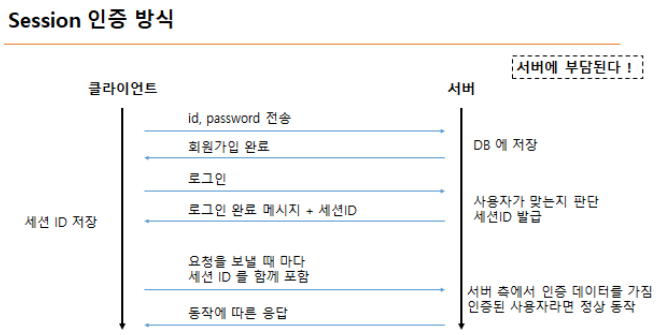
하지만 세션정보를 서보에만 저장하다보면 회원이 많아 질 경우 서버에는 큰 부담이 되므로

개발자들은 서버의 부담을 줄이기 위해서 서버가 사용자의 인증상태를 저장하는 방식이 아닌,

유저(클라이언트)가 세션정보를 저장하는 방법을 고안하였고, 그게 바로 토큰 인증 방식이다.

가장 기본적인 인증 방식 (가장 기본적인 형태)





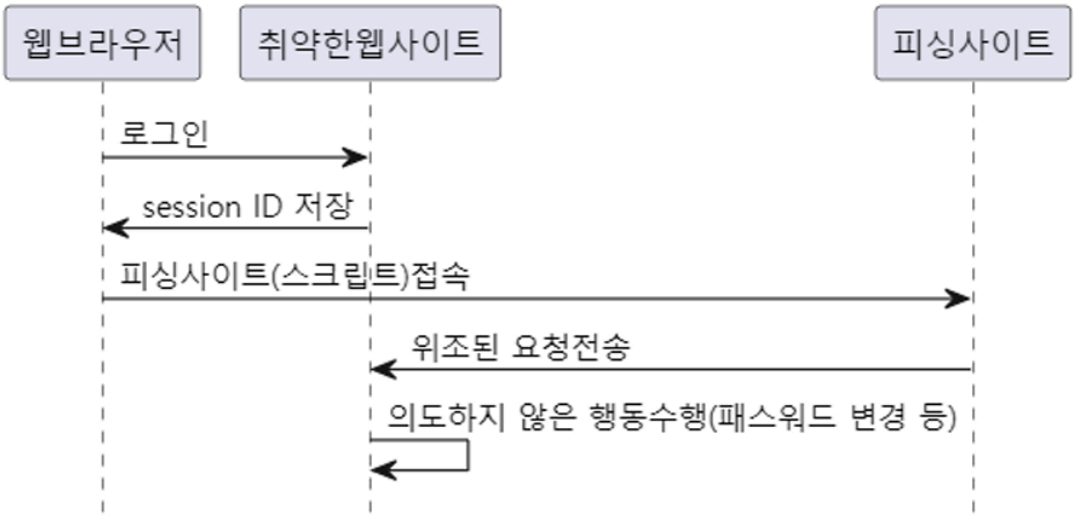
[CSRF 토큰]

CSRF 공격에 대해서 먼저 살펴보자.

1. 유저가 보안이 취약한 사이트에 로그인을 한다. 이때, 로그인 인증을 위해 세션ID가 생성된다.
2. 로그인 이후 서버에 저장된 세션 정보를 사용할 수 있는 session id가 사용자 브라우저 쿠키에 저장될 것이다.
   그 다음
3. 해커는 유저가 악성 스크립트 페이지를 누르도록 유저의 행동을 유도한다.
4. 유저가 악성 스크립트가 작성된 웹페이지를 방문시

   쿠키에 저장된 sessionID를 탈취하여 보안이 취약한 웹사이트에

   위조한 sessionID를 이용해서 비밀번호를 바꾼다.

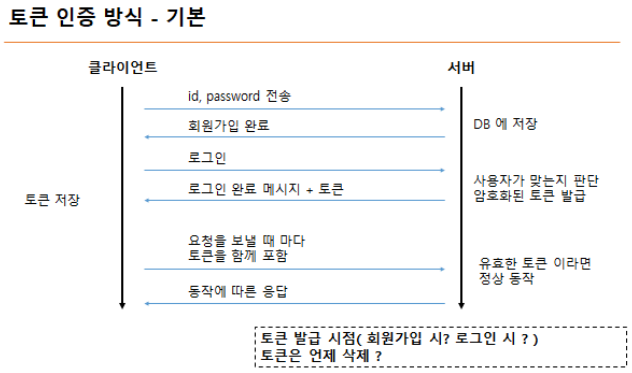


이를 CSRF 탈취를 막기위한 과정을 살펴보자.

1. 유저가 웹사이트(서버)로 요청을 보낼 때 회원 가입이 완료된 유저는

HTTP GET Method를 사용해서 로그인을 시도한다.

2. 서버는 CSRF 토큰을 생성하고 HTML 형식의 숨겨진 태그를 사용하여 유저의 HTML문서에 토큰을 삽입 해서 클라이언트에게 응답한다.
3. 클라이언트가 글을 작성 후 CSRF 토큰과 함께 서버에 요청을 보낸다. CSRF 토큰은 HTML 형식의 숨겨진 필드에 포함되어 있으므로 CSRF 토큰 값이 요청 매개 변수로 전송된다.
4. 서버는 요청으로 들어온 CSRF 토큰 값과 서버에 저장이 되어 있는 값이 동일한지 확인한다. 토큰 값이 일치하지 않으면 잘못된 요청으로 오류를 발생해 버린다.



토큰을 발급하는 시점이나 삭제 시점 등은 정형화된 것이 없다. 개발시 서버 개발자가 결정할 사항이다. 나중에 Django에서 인증과 권한에 대해서 학습할 때 Djnago에서의 토큰 인증방식에 대해서 조금 더 자세하게 살펴 볼 예정이다.

진짜로 끝!