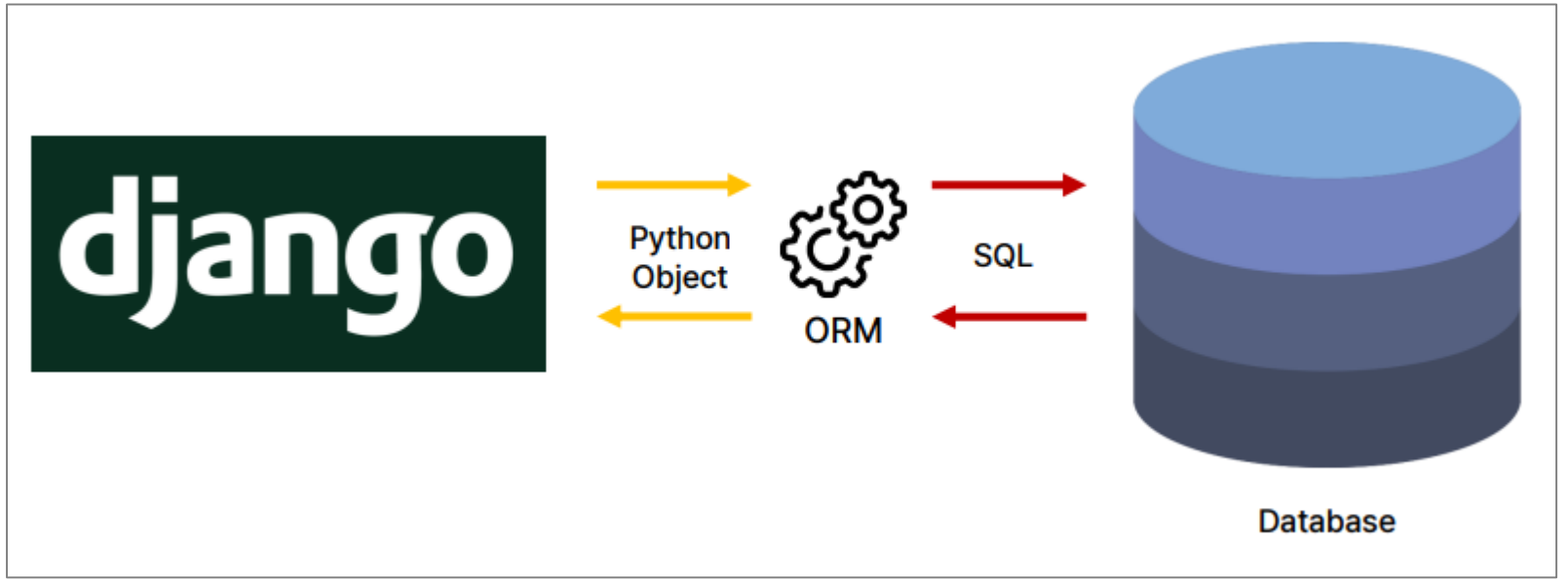
Django ORM

ORM

ORM(Object Relational Mapping)

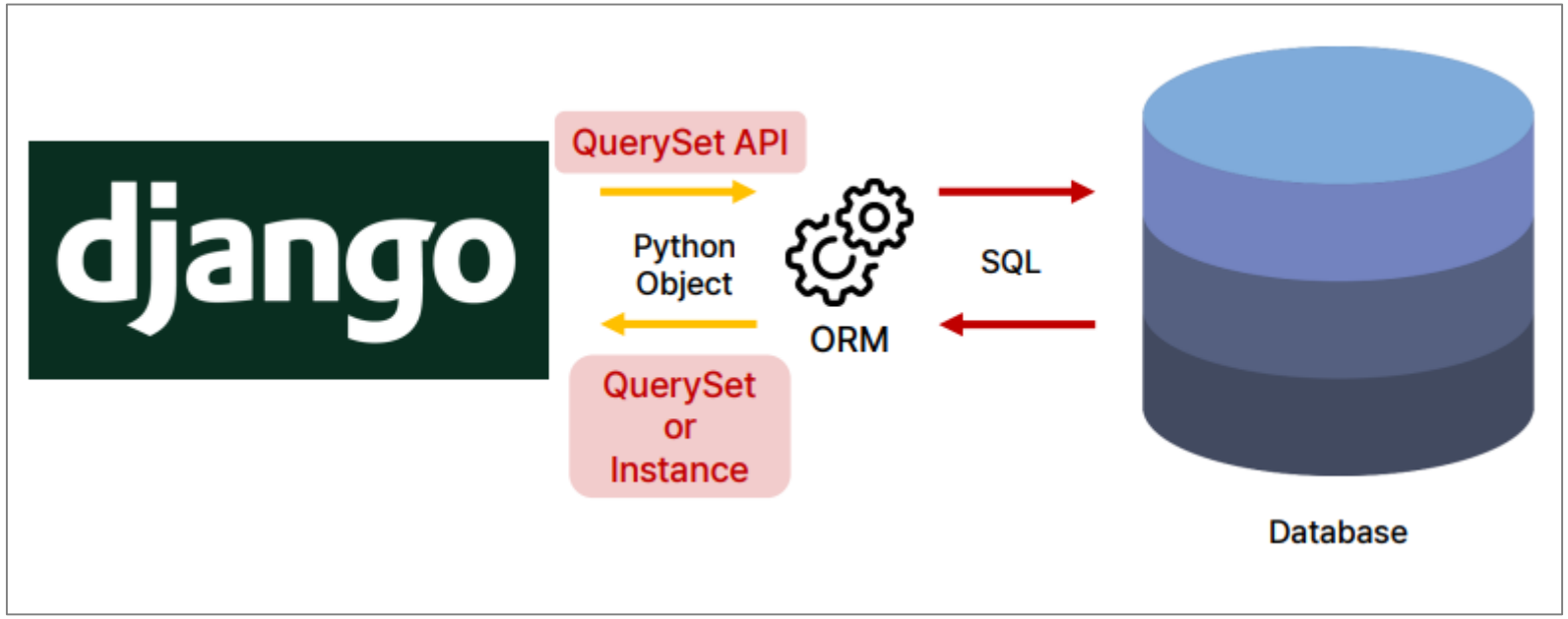
- DB와 객체 지향 프로그래밍 언어 간의 데이터 변환을 자동화하여 SQL문을 작성하지 않고도 데이터를 다룰 수 있게 도와주는 도구



Django ORM

QuerySet API

- DB로부터 데이터를 검색, 필터링, 정렬, 그룹화 등의 작업을 수행하는 도구
- API를 사용하여 SQL이 아닌 Python 코드로 데이터를 조작



Django QuerySet API

- QuerySet API 구문

Model Class . **Manager** . **QuerySet API**

- QuerySet API 예시

Article . **objects** . **all()**

Django QuerySet API 실습 전 준비

외부 라이브러리 설치 및 설정

- ipython 설치

```
$ pip install ipython
```

- django-extensions 설치

```
$ pip install django-extensions
```

```
# my_project/settings.py
INSTALLED_APPS = [
    'articles',
    'django_extensions',
    ...
]
```

Django shell

Django 환경 안에서 실행되는 python shell

- 입력하는 QuerySet API가 프로젝트에 바로 영향을 줌
- 아래 명령어로 실행

```
$ python manage.py shell_plus
```

- 기본적인 Django Setting이 되어 있어 학습 및 테스트에 유용

```
# Shell Plus Model Imports
from articles.models import Article
from django.contrib.admin.models import LogEntry
from django.contrib.auth.models import Group, Permission, User
from django.contrib.contenttypes.models import ContentType
from django.contrib.sessions.models import Session
# Shell Plus Django Imports
from django.core.cache import cache
from django.conf import settings
from django.contrib.auth import get_user_model
from django.db import transaction
from django.db.models import Avg, Case, Count, F, Max, Min, Prefetch, Q, Sum, When
from django.utils import timezone
from django.urls import reverse
from django.db.models import Exists, OuterRef, Subquery
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.23.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

Django 데이터 생성

1. 클래스로부터 인스턴스를 생성한 후 인스턴스를 DB에 저장

```
In [1]: # Article(Class) 로부터 article(Instance) 생성
   ...: article =  Article()

In [2]: # article(Instance) 생성 완료

In [3]: article
Out[3]: <Article: Article object (None)>
```

```
In [8]: # 인스턴스 변수 할당(집어넣을 데이터 값)

In [9]: article.title = "test_title"

In [10]: article.content = "text_content"
```

```
In [12]: Article.objects.all()
Out[12]: <QuerySet []>
```

```
In [13]: article.save()

In [14]: Article.objects.all()
Out[14]: <QuerySet [<Article: Article object (2)>]>
```

Django 데이터 생성

2. QuerySet API 중 create() 메소드 활용

- id는 Django가 알아서 생성하며, 괄호에 있는 값이 id

```
In [15]: Article.objects.create(title="test_title2", content="test_content2")
Out[15]: <Article: Article object (3)>
```

```
In [16]: Article.objects.all()
Out[16]: <QuerySet [<Article: Article object (2)>, <Article: Article object (3)>]>
```

Django 데이터 조회

1. 전체 데이터 조회

- QuerySet 데이터를 반환

- QuerySet

  - DB에게서 받은 객체 목록(여러 개)

  - Django ORM을 통해 만든 자료형이며, 순회가능한 객체

- 아래 코드로 전체 데이터 조회 가능

```
Article.objects.all()
```

```
<QuerySet [<Article: Article object (2)>, <Article: Article object (3)>]>
```

Django 데이터 조회

2. 단일 데이터 조회

- Instance(DB에게서 받은 객체 하나) 데이터를 반환
- 객체를 찾을 수 없으면 DoesNotExist 에러 발생
- 2개 이상의 객체를 찾으면 MultipleObjectsReturned 에러 발생
- 한 개의 데이터를 조회해야 하기 때문에 고유키를 사용하여 조회해야 함
- 아래 코드로 단일 데이터 조회 가능

```
Article.objects.get(pk=2)
```

```
<Article: Article object (2)>
```

Django 데이터 조회

3. 데이터 필터

- 특정 조건을 설정한 후에 데이터를 조회
- 다양한 QuerySet API 및 필터(Field lookups)는 Django 공식 홈페이지 참고
- 아래 코드로 데이터를 필터 후 조회 가능

```
Article.objects.filter(title="test_title")
```

```
<QuerySet [<Article: Article object (2)>]>
```

Django 데이터 업데이트

업데이트할 인스턴스 조회 후 update() 메소드 활용

```
Article.objects.filter(pk=2).update(content="update_content")
```

```
Article.objects.get(pk=2).content
'update_content'
```

Django 데이터 삭제

업데이트할 인스턴스 조회 후 delete() 메소드 활용

```
Article.objects.filter(pk=2).delete()
```

```
Article.objects.all()
<QuerySet [<Article: Article object (3)>]>
```

Django ORM 활용

게시글 목록 구현


```python
from .models import Article

# articles/views.py
def index(request):
    articles = Article.objects.all()
    context = {
        'articles': articles
    }

    return render(request, 'articles/index.html', context)
```

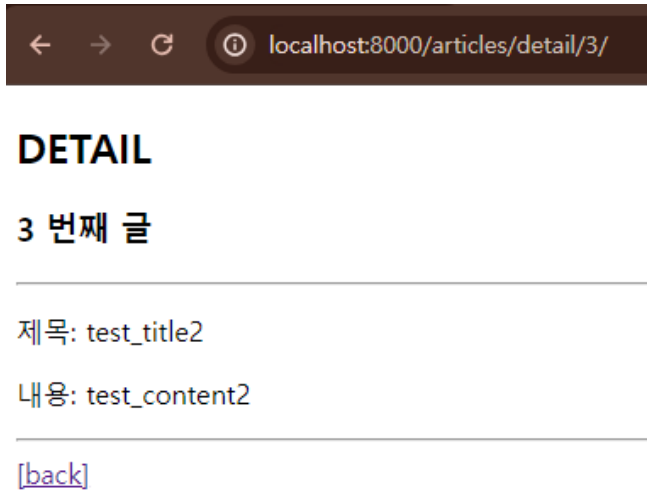
```
<!-- templates/articles/index.html-->
{% extends "articles/base.html" %}

{% block content %}
  <h1>Articles</h1>
  <hr>
  {% for article in articles %}
    <p>글 번호: {{ article.pk }}</p>
    <p>글 제목: {{ article.title }}</p>
    <p>글 내용: {{ article.content }}</p>
    <hr>
  {% endfor %}
{% endblock content %}
```

```python
# articles/urls.py
app_name = 'articles'
urlpatterns = [
    path('index/', views.index, name='index'),
    ...
]
```

```python
# artices/models.py
from django.db import models

class Article(models.Model):
    title = models.CharField(max_length=20)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

Django ORM 활용

게시글 상세보기 구현




```python
# articles/views.py
def detail(request, pk):
    article = Article.objects.get(pk=pk)
    context = {
        'article': article
    }
    return render(request, 'articles/detail.html', context)
```

```python
# articles/urls.py
app_name = 'articles'
urlpatterns = [
    path('detail/<int:pk>/', views.detail, name='detail'),
    ...
]
```

```
<!-- templates/articles/detail.html-->
{% extends "articles/base.html" %}

{% block content %}
    <h2>DETAIL</h2>
    <h3>{{ article.pk }} 번째 글</h3>
    <hr>
    <p>제목: {{ article.title }}</p>
    <p>내용: {{ article.content }}</p>
    <hr>
    <a href="{% url 'articles:index' %}">[back]</a>
{% endblock content %}
```

Django ORM 활용

게시글 목록에서 상세보기로 이동하는 링크 추가하기


```
<!-- templates/articles/index.html-->
{% extends "articles/base.html" %}

{% block content %}
  <h1>Articles</h1>
  <hr>
  {% for article in articles %}
    <p>글 번호: {{ article.pk }}</p>
    <a href="{% url 'articles:detail' pk=article.pk %}">
      <p>글 제목: {{ article.title }}</p>
    </a>
    <p>글 내용: {{ article.content }}</p>
    <hr>
  {% endfor %}
{% endblock content %}
```

Django ORM 활용

게시글 작성 페이지 구현


```python
# articles/views.py
def create(request):
    return render(request, 'articles/create.html')
```

```python
# articles/urls.py
app_name = 'articles'
urlpatterns = [
    path('create/', views.create, name='create')
]
```

```html
<!-- templates/articles/create.html-->
{% extends "articles/base.html" %}

{% block content %}
    <form action="{% url 'articles:save' %}" method="GET">
        <div>
            <label for="title">Title: </label>
            <input type="text" name="title" id="title">
        </div><br>
        <div>
            <label for="content">Content: </label>
            <textarea name="content" id="content"></textarea>
        </div><br>
        <input type="submit">
    </form>
    <hr>
    <a href="{% url 'articles:index' %}">[back]</a>
{% endblock content %}
```

Django ORM 활용

게시글 목록에서 게시글 작성 하이퍼링크 구현


```
<!-- templates/articles/index.html-->
{% extends "articles/base.html" %}

{% block content %}
  <h1>Articles</h1>
  <a href="{% url 'articles:create' %}">게시글 작성하기</a>
  <hr>
  {% for article in articles %}
    <p>글 번호: {{ article.pk }}</p>
    <a href="{% url 'articles:detail' pk=article.pk %}">
      <p>글 제목: {{ article.title }}</p>
    </a>
    <p>글 내용: {{ article.content }}</p>
    <hr>
  {% endfor %}
{% endblock content %}
```

Django ORM 활용

게시글 작성 페이지에서 게시글 작성 기능 구현


```python
# articles/views.py
def save(request):
    data = request.GET
    # 방법 1) 인스턴스 생성해서 데이터 추가
    # article = Article()
    # article.title = data.get('title')
    # article.content = data.get('content')
    # article.save()

    # 방법 2) QuerySet API 이용해서 데이터 추가
    Article.objects.create(title=data.get('title'), content=data.get('content'))

    return render(request, 'articles/save.html')
```

```python
# articles/urls.py
app_name = 'articles'
urlpatterns = [
    path('save/', views.save, name='save'),
    ...
]
```

```html
<!-- templates/articles/save.html-->
{% extends "articles/base.html" %}

{% block content %}
  <h1>게시글 작성 완료</h1>
{% endblock content %}
```

Django ORM 활용

게시글 작성 후 작성 게시글 상페 페이지로 돌아가는 기능 구현


```python
from django.shortcuts import render, redirect
from .models import Article

# articles/views.py
def save(request):
    data = request.GET
    # 방법 1) 인스턴스 생성해서 데이터 추가
    # article = Article()
    # article.title = data.get('title')
    # article.content = data.get('content')
    # article.save()

    # 방법 2) QuerySet API 이용해서 데이터 추가
    new_article = Article.objects.create(title=data.get('title'), content=data.get('content'))

    return redirect('articles:detail', pk=new_article.pk)
```

Django ORM 활용

게시글 작성은 데이터 생성이므로 HTML form method를 post를 사용

- 기억이 안난다면 다음 페이지 참고

```
<!-- templates/articles/create.html-->
{% extends "articles/base.html" %}

{% block content %}
    <form action="{% url 'articles:save' %}" method="GET">
        <div>
            <label for="title">Title: </label>
            <input type="text" name="title" id="title">
        </div><br>
        <div>
            <label for="content">Content: </label>
            <textarea name="content" id="content"></textarea>
        </div><br>
        <input type="submit">
    </form>
    <hr>
    <a href="{% url 'articles:index' %}">[back]</a>
{% endblock content %}
```

```
<!-- templates/articles/create.html-->
{% extends "articles/base.html" %}

{% block content %}
    <form action="{% url 'articles:save' %}" method="POST">
        <div>
            <label for="title">Title: </label>
            <input type="text" name="title" id="title">
        </div><br>
```

```
# articles/views.py
def save(request):
    # data = request.GET
    # 데이터 생성은 POST 메소드를 활용해야 함
    data = request.POST
    ...
```

HTML form 형식 (참고 페이지)

6. method

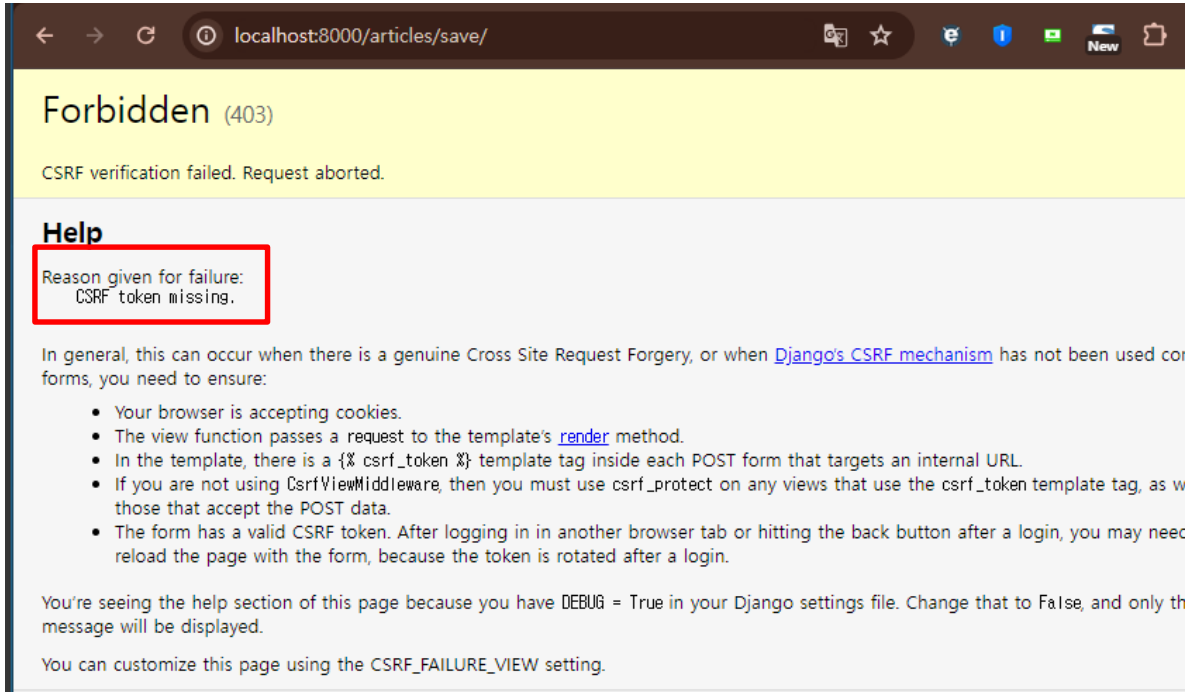
- 데이터를 어떤 방식으로 요청할 지 설정
- get, post 으로 구분
- default 는 get 으로 설정
- 일반적으로 GET 은 조회, POST 는 변경(생성,수정,삭제) 일 때 사용

```
<form action="/save_review/" method="GET">
    <label for="review_text_id">리뷰: </label>
    <input type="text" name="review_text" id="review_text_id">
    <input type="submit" value="submit">
</form>
```

Django ORM 활용

게시글 작성하면 403 에러가 나타남을 확인할 수 있음

- 서버에 요청이 전달되었지만, 권한 때문에 거절됨
- 원인은 csrf token 누락



CSRF(Cross Site Request Forgery)

다른 사용자가 마치 본인의 요청처럼 의도적으로 숨겨서 공격하는 방식

=> 위 문제를 해결하기 위해서 CSRF Token을 적용

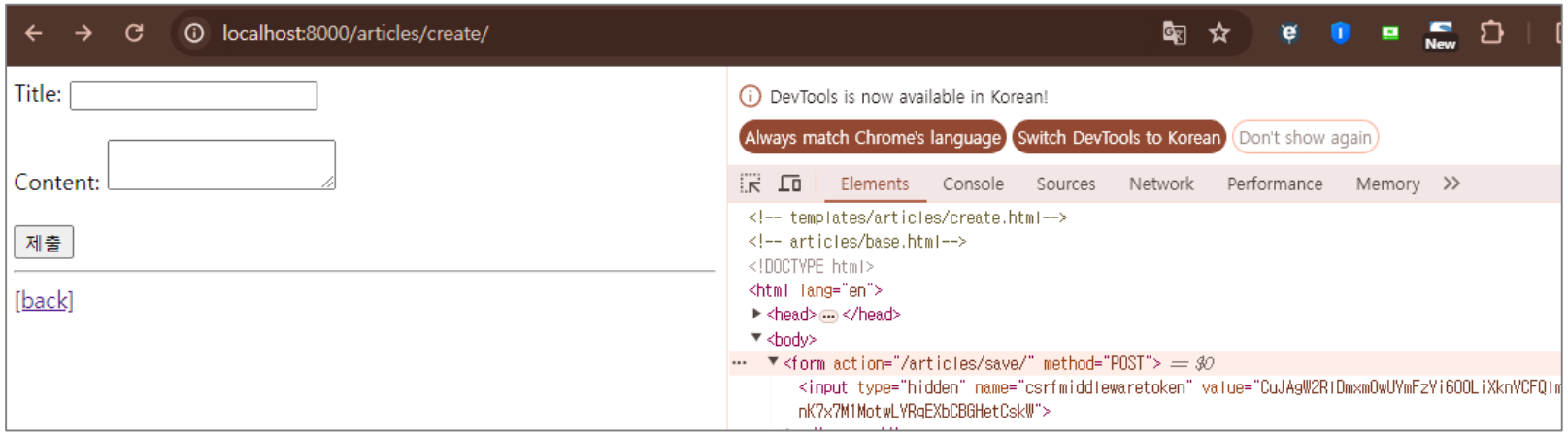
CSRF Token

1. 서버는 사용자 요청에 따라 CSRF Token을 발급해준다.
2. 사용자는 POST 요청일 경우 발급받은 CSRF Token 서버에 같이 전달한다.
3. 서버는 서버에 저장된 세션별 CSRF Token과 전달받은 Token을 비교한다.

Django ORM 활용

HTML form method가 post인 경우, csrf token을 추가해서 전송

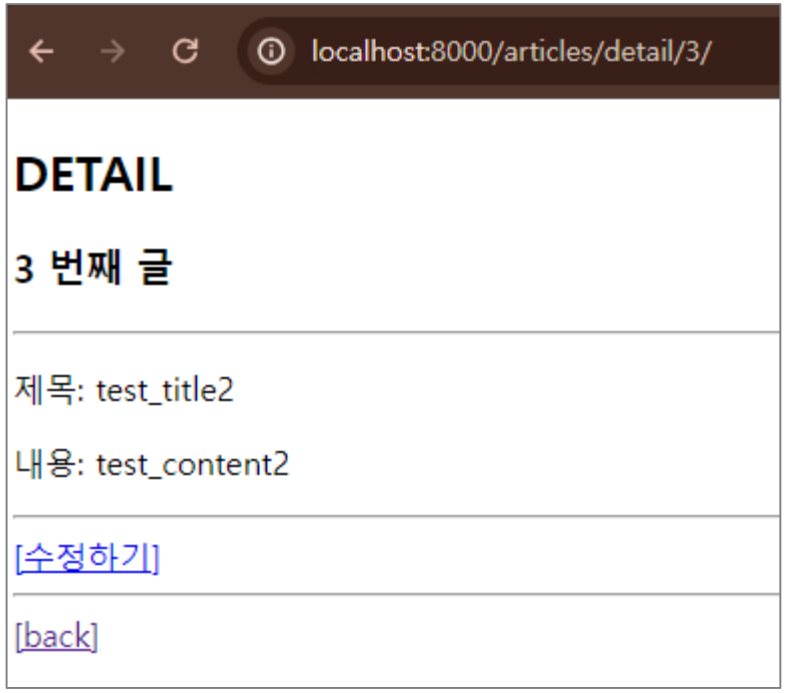
```
<!-- templates/articles/create.html-->
{% extends "articles/base.html" %}

{% block content %}
    <form action="{% url 'articles:save' %}" method="POST">
        {% csrf_token %}
        ...
```



Forestsoft

Django ORM 활용

게시글 상세보기 페이지에서 수정페이지 이동 하이퍼링크 구현



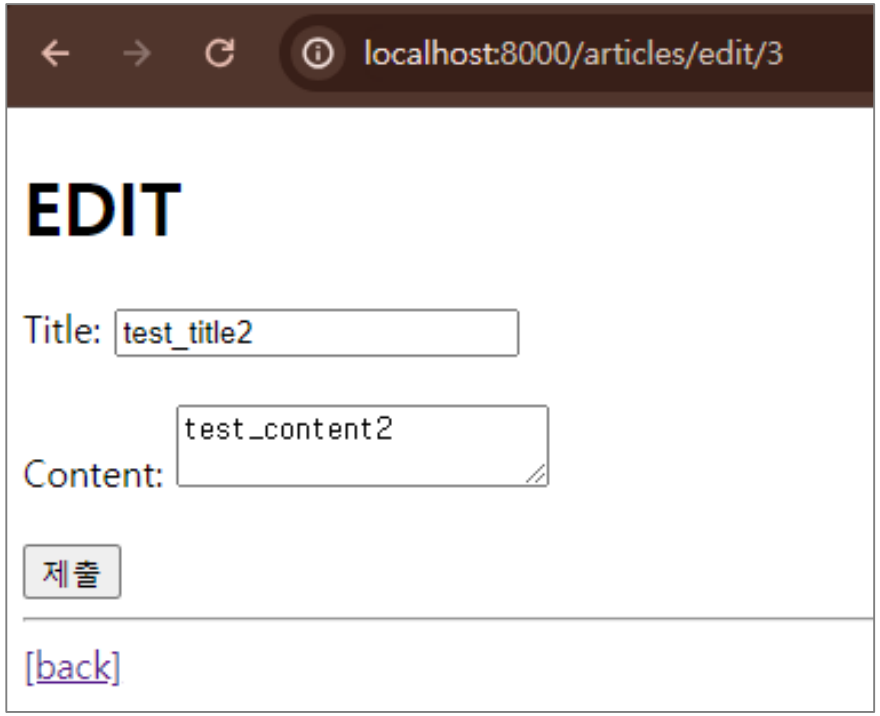

```python
# articles/views.py
def edit(request, pk):
    article = Article.objects.get(pk=pk)
    context = {
        'article':article
    }
    return render(request, 'articles/edit.html', context)
```

```python
# articles/urls.py
app_name = 'articles'
urlpatterns = [
    path('edit/<int:pk>', views.edit, name='edit'),
    ...
]
```

```
<!-- templates/articles/detail.html-->
{% extends "articles/base.html" %}

{% block content %}
    <h2>DETAIL</h2>
    <h3>{{ article.pk }} 번째 글</h3>
    <hr>
    <p>제목: {{ article.title }}</p>
    <p>내용: {{ article.content }}</p>
    <hr>
    <a href="{% url 'articles:edit' pk=article.pk %}">[수정하기]</a>
    <hr>
    <a href="{% url 'articles:index' %}">[back]</a>
{% endblock content %}
```

Django ORM 활용

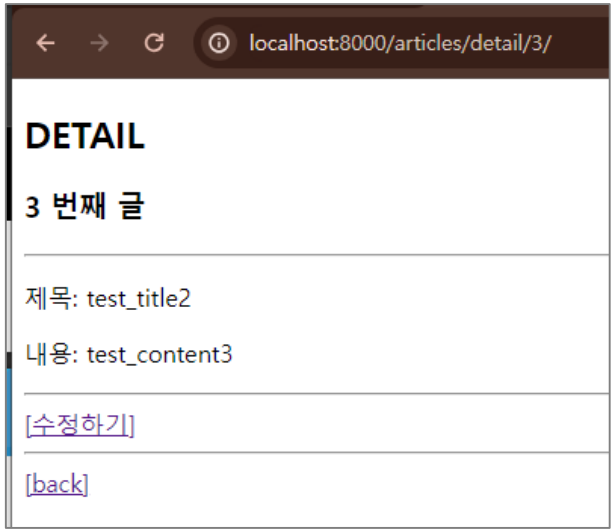
게시글 상세보기 페이지에서 수정페이지 구현




```
<!-- templates/articles/edit.html-->
{% extends "articles/base.html" %}

{% block content %}
    <h1>EDIT</h1>
    <form action="#" method="POST">
    {% csrf_token %}
    <div>
        <label for="title">Title: </label>
        <input type="text" name="title" id="title" value="{{ article.title }}">
    </div><br>
    <div>
        <label for="content">Content: </label>
        <textarea name="content" id="content">{{ article.content }}</textarea>
    </div><br>
    <input type="submit" value="수정하기">
    </form>
    <hr>
    <a href="{% url 'articles:index' %}">[back]</a>
{% endblock content %}
```

Django ORM 활용

게시글 수정 기능 구현



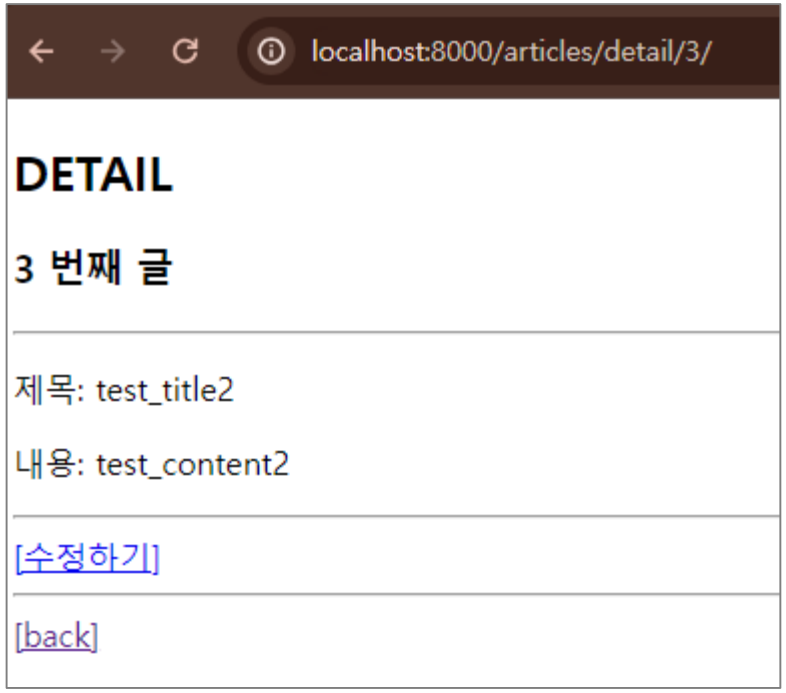

```python
# articles/views.py
def update(request, pk):
    data = request.POST
    update_article = Article.objects.filter(pk=pk).update(
        title=data.get('title'), content=data.get('content'))
    return redirect('articles:detail', pk=pk)
```

```python
# articles/urls.py
app_name = 'articles'
urlpatterns = [
    path('update/<int:pk>', views.update, name='update'),
    ...
]
```

```
<!-- templates/articles/edit.html-->
{% extends "articles/base.html" %}

{% block content %}
    <h1>EDIT</h1>
    <form action="{% url 'articles:update' pk=article.pk %}" method="POST">
    {% csrf_token %}
    ...
```

Django ORM 활용

게시글 상세보기 페이지에서 삭제 기능 구현




```python
# articles/views.py
def delete(request, pk):
    data = request.POST
    delete_article = Article.objects.filter(pk=pk).delete()
    return redirect('articles:index')
```

```python
# articles/urls.py
app_name = 'articles'
urlpatterns = [
    path('delete/<int:pk>', views.delete, name='delete'),
    ...
]
```

```
<!-- templates/articles/detail.html-->
{% extends "articles/base.html" %}

{% block content %}
    <h2>DETAIL</h2>
    <h3>{{ article.pk }} 번째 글</h3>
    <hr>
    <p>제목: {{ article.title }}</p>
    <p>내용: {{ article.content }}</p>
    <hr>
    <a href="{% url 'articles:edit' pk=article.pk %}">[수정하기]</a>
    <hr>
    <a href="{% url 'articles:delete' pk=article.pk %}">[삭제하기]</a>
    <hr>
    <a href="{% url 'articles:index' %}">[back]</a>
{% endblock content %}
```