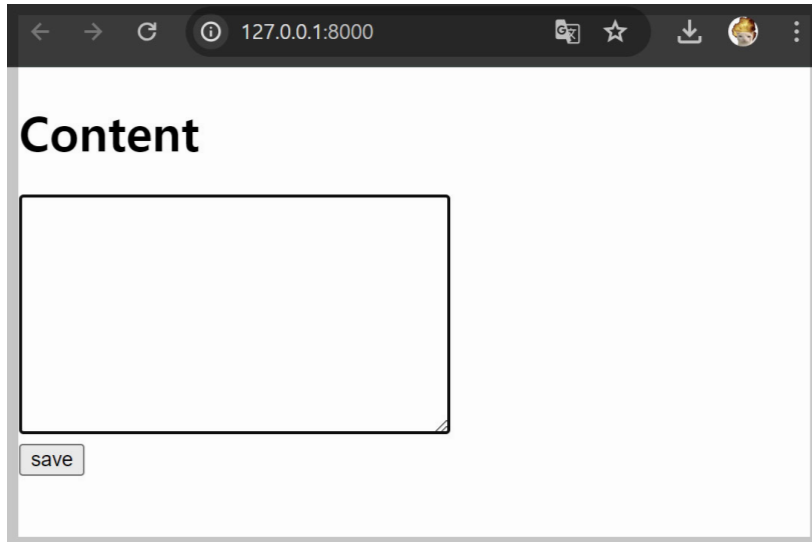


Django 04. forms.py와 HTML 사이트 구현하기.



swing 32기 실습 스터디에서 진행한 해당 사이트를 구현해볼 것이다.

forms.py

C:/Django/mysite/main/forms.py

```
from django import forms
```

해당 명령어를 통해 애플리케이션 디렉토리에 forms.py를 생성한다.

장고의 폼에는 두 가지 종류가 있다.

- `forms.form` : 폼
- `forms.ModelForm` : 모델 폼

```

main > forms.py > ContentForm > Meta
1  from django import forms
2  from .models import Content
3
4  class ContentForm(forms.ModelForm):
5      def __init__(self, *args, **kwargs):
6          super(ContentForm, self).__init__(*args, **kwargs)
7          self.fields['content'].widget.attrs.update({
8              'class': 'form-control',
9              'autofocus': True
10         })
11
12     class Meta:
13         model = Content
14         fields = ['content']

```

스터디에서 진행한 form은 form.ModelForm을 상속받아 폼을 만들었다는 것을 알 수 있다.

모델 폼은 모델과 연결된 폼이며, 모델폼의 객체를 저장한다는 것은 연결된 모델의 데이터를 저장할 수 있다는 뜻이다.

`class Meta:`
forms.ModelForm을 상속받아 만든 폼은 반드시 Meta 클래스를 가져야 한다.
이 공간에는 모델 폼이 사용할 모델과 모델의 필드들을 적어야 한다.

표준 HTML

```

{% load static %}
<!doctype html>
<html lang="ko">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, install-scale=1, shrink-to-fit=no">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" type="text/css" href="{% static 'bootstrap.min.css' %}">
    <!-- pybo CSS -->
    <link rel="stylesheet" type="text/css" href="{% static 'style.css' %}">
    <title>Hello, pybo!</title>
</head>
<body>
    ( ... 생략 ... )
</html>

```

적어둔 구조는 장고에서 사용되는 HTML의 표준 구조이다.

1. html, head, body 엘리먼트가 있어야 한다.
2. CSS 파일은 head 엘리먼트 안에 있어야 한다.

3. head 엘리먼트 안에는 meta, title 엘리먼트 등이 포함되어야 한다.

index.html

```

<> index.html X
main > templates > main > <> index.html
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <!-- Required meta tags -->
5      <meta charset="utf-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
7      <title>SWING32</title>
8  </head>
9  <body>
10     <h1>Content</h1>
11     <form method="post">
12         {% csrf_token %}
13         <div>{{form.content}}</div>
14         <button type="submit">save</button>
15     </form>
16 </body>
17 </html>

```

```
<meta charset="utf-8">
```

문서 문자 인코딩을 UTF-8로 설정한다.

```
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

반응형 디자인을 지원하는 설정이다.

문서의 viewport 설정을 통해 모바일 장치에서의 렌더링 방식을 제어한다.

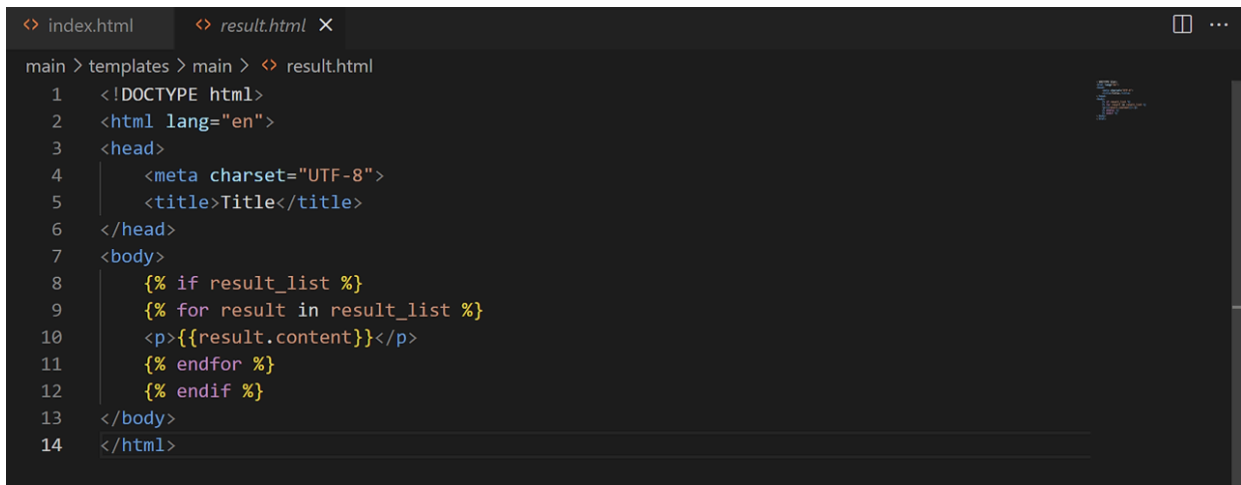
`initial-scale=1` : 초기 스케일을 1로 설정한다.

`width=device-width` : 뷰포트 너비를 장치의 너비와 동일하게 설정한다.

```
<title>SWING32</title>
```

title은 브라우저의 제목 표시줄이나 탭에 표시될 문서의 제목을 설정한다.

result.html



```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8     {% if result_list %}
9     {% for result in result_list %}
10    <p>{{result.content}}</p>
11    {% endfor %}
12    {% endif %}
13 </body>
14 </html>

```

각각의 html에 작성해준 내용이다.

```

{% if result_list %}
{% for result in result_list %}
<p>{{ result.content }}</p>
{% endfor %}
{% endif %}

```

result.html에 이 부분은 조건문과 반복문을 의미한다.

```

{% if result_list %}
{% endif %}

```

해당 코드는 result_list가 존재하고 비어있지 않다면 아래의 내용을 실행하는 조건문이다.

```

{% for result in result_list %}
{% endfor %}

```

해당 코드는 result_list의 각 항목에 대한 반복문을 실행하는 코드이다.

구현을 위해 html을 설정해준 다음의 과정이다.

urls.py

```
main > 📄 urls.py > ...
1  from django.urls import path
2
3  from . import views
4
5  urlpatterns = [
6      path('', views.index, name='index'),
7      path('result/', views.result, name='result'),
8  ]
```

- urlpatterns : URL의 패턴 리스트. URL 패턴과 패턴이 매칭될 때 호출될 views.py 함수를 정의한다.
- path : 개별 URL 패턴을 뜻한다.
 - ('', views.index, name='index')
 - 순서대로 루트 URL에 접근했을 때 매칭될 패턴 정의, views 모듈의 index 뷰 함수를 호출, URL 패턴에 이름을 붙이는 작업을 의미한다.

루트 URL('/')

이곳에 접근하면 views.index 뷰 함수가 호출된다.

결과 URL 'result/'

이곳에 접근하면 views.result 뷰 함수가 호출된다.

views.py

```

main > views.py > index
1  from django.shortcuts import render, redirect
2  from django.contrib import messages
3  from .forms import ContentForm
4  from .models import Content
5
6  # Create your views here.
7  def index(request):
8      if request.method == "POST": #2 요청이 POST 형식일 때
9          form = ContentForm(request.POST)
10         if form.is_valid():
11             content_form = form.save(commit=False) # 바로 저장하는 것을 방지
12             content_form.save()
13             return redirect('result/')
14         else:
15             messages.error(request, "error")
16             return redirect('main/index.html')
17
18     else: #1 index.html 페이지에 접속하면 여기가 먼저 실행된다.
19         form = ContentForm()
20         context = {
21             'form': form
22         }
23         return render(request, 'main/index.html', context)
24
25
26 def result(request):
27     result_list = Content.objects.all()
28     context = {
29         'result_list' : result_list
30     }
31     return render(request, 'main/result.html', context)

```

최종적으로 views.py에서 index와 result라는 뷰 함수를 정의하게 된다.

이 두 개의 view 함수들은 특정한 URL에 접근될 때 호출되어 특정한 작업을 수행한다.

그리고 template에 적어있던 내용을 렌더링하여 사용자에게 응답하는 역할을 수행하고 있다.

/index 페이지에서 출력되는 과정

```

else: #1 index.html 페이지에 접속하면 여기가 먼저 실행된다.
    form = ContentForm()
    context = {
        'form': form
    }
    return render(request, 'main/index.html', context)

```

1. 사용자가 '/' URL에 접속하면, GET 요청으로 인해 실행된다.
2. ContentForm 객체를 생성한다.
3. 2번의 객체를 context 딕셔너리에 담아 main/index.html 템플릿에 전달한다.
4. main/index.html 템플릿이 렌더링되어 사용자에게 forms.py가 표시된다.

/result 페이지에서 출력되는 과정

```
if request.method == "POST": #2 요청이 POST 형식일 때
    form = ContentForm(request.POST)
    if form.is_valid():
        content_form = form.save(commit=False) # 바로 저장하는 것을 방지
        content_form.save()
        return redirect('result/')
    else:
        messages.error(request, "error")
        return redirect('main/index.html')
```

1. 사용자가 forms.py를 작성하고 제출하면, POST 요청이 발생한다.
2. views.py에서 이 부분이 실행된다.
3. ContentForm 객체를 생성하고 forms.py가 유효한지 확인한다.
4. form.save(commit=False)를 사용하여 DB에 저장하지 않고 content_form 객체를 반환한다.
5. content_form.save()를 호출하여 데이터를 데이터베이스에 저장한다.
6. '/result/' URL로 리디렉션한다.
7. 만약 폼이 유효하지 않으면 에러 문구를 출력하고 main/index.html로 리디렉션한다.