Django Форми

Форми — це основа для взаємодії користувача з веб-застосунком. Django надає потужні інструменти для роботи з ними, що значно спрощує обробку, валідацію та рендеринг даних.

Відправка HTML-форм

Одним із способів надсилання даних на сервер є використання стандартних HTML-форм, як правило, за допомогою POST-запиту. У Django, щоб отримати ці дані, використовується об'єкт request.POST.

Приклад: Базова HTML-форма

Створимо просту форму для введення імені та віку користувача.

- **Шаблон index.html**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>User Form</title>
</head>
<body>
    <h2>Форма користувача</h2>
    <form method="post" action="postuser/">
        {% csrf_token %}
        <p>Ім'я:<br> <input name="name" /></p>
        <p>Вік:<br> <input name="age" type="number" /></p>
        <input type="submit" value="Відправити" />
    </form>
</body>
</html>
```

Зверніть увагу на тег {% csrf_token %}. Він є обов'язковим для захисту від CSRF-атак і додає приховане поле до форми. Django автоматично перевіряє його наявність у POST-запитах.

- **Файл views.py**

```
from django.shortcuts import render
from django.http import HttpResponse

def index(request):
    return render(request, "index.html")

def postuser(request):
    # Отримуємо дані з об'єкта request.POST
    name = request.POST.get("name", "Undefined")
    age = request.POST.get("age", 1)
    return HttpResponse(f"<h2>Ім'я: {name}  Вік: {age}</h2>")
```

Метод request.POST.get() дозволяє отримати значення поля за його іменем (name або age), а другий аргумент є значенням за замовчуванням, якщо поле не було відправлено.

- **Файл urls.py**

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

urlpatterns = [
    path("", views.index),
    path("postuser/", views.postuser),
]
```

Завдання 1: Відправка HTML-форми

Створіть застосунок, що містить форму для реєстрації на подію.

- **Файл views.py**: Додайте функцію register для обробки POST-запиту. Вона має отримувати ім'я, прізвище та електронну пошту, а потім повертати HTML-сторінку з підтвердженням реєстрації.
- **Файл urls.py**: Створіть відповідні маршрути для форми та сторінки обробки.
- **Файл register_form.html**: Створіть шаблон з формою, яка включає поля для імені, прізвища та електронної пошти.

Визначення форм Django

Django надає спеціальні класи для форм, які допомагають уникнути дублювання коду,

спрощують валідацію та забезпечують повторне використання. Кожна Django-форма — це клас, що успадковується від forms.Form.

Створення та відображення форми

Зазвичай, класи форм зберігаються в окремому файлі forms.py у вашому застосунку.

- **Файл forms.py**

```
from django import forms

class UserForm(forms.Form):
    # Визначаємо поля форми, які автоматично перетворюються в HTML-елементи
    name = forms.CharField()
    age = forms.IntegerField()
```

Тут CharField створює текстове поле, а IntegerField — поле для чисел.

- **Файл views.py**

```
from django.shortcuts import render
from .forms import UserForm

def index(request):
    # Створюємо екземпляр форми
    userform = UserForm()
    # Передаємо об'єкт форми в шаблон
    return render(request, "index.html", {"form": userform})
```

- **Шаблон index.html**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django Forms</title>
</head>
```

```
<body>
    <form method="POST">
        {% csrf_token %}
        <table>
            {{ form.as_table }}
        </table>
        <input type="submit" value="Відправити" >
    </form>
</body>
</html>
```

Використання {{ form.as_table }} генерує HTML-розмітку форми у вигляді таблиці, що значно спрощує її відображення. Існують також інші методи: as_ul() (як список), as_p() (як параграфи) або as_div() (як блоки div).

Завдання 2: Створення та відображення форми

Створіть форму для введення інформації про книгу.

- **Файл forms.py**: Визначте клас BookForm, який містить поля для назви книги, автора та року видання.
- **Файл views.py**: Створіть функцію book_form, яка відображає цю форму.
- **Файл urls.py**: Створіть маршрут для форми.
- **Файл book_form.html**: Створіть шаблон, який відображає форму у вигляді списку (as_ul).

Валідація даних у формах Django

Валідація — це процес перевірки даних на коректність. Django надає вбудовані механізми валідації на стороні сервера, які не дозволять зберегти некоректні дані, навіть якщо клієнтська перевірка була вимкнена.

Основним методом для валідації є **is_valid()**. Він перевіряє дані, передані у форму, і повертає True, якщо вони коректні, і False у протилежному випадку.

Приклад: Валідація з is_valid()

- **Файл views.py**

```
from django.shortcuts import render
from django.http import HttpResponse
from .forms import UserForm
```

```
def index(request):
    if request.method == "POST":
        # Заповнюємо форму даними з POST-запиту
        userform = UserForm(request.POST)
        # Перевіряємо валідність даних
        if userform.is_valid():
            # Якщо дані коректні, вони доступні через cleaned_data
            name = userform.cleaned_data["name"]
            return HttpResponse(f"<h2>Привіт, {name}</h2>")
        else:
            # Якщо дані некоректні, показуємо форму з помилками
            return render(request, "index.html", {"form": userform})
    else:
        # Якщо запит GET, створюємо порожню форму
        userform = UserForm()
        return render(request, "index.html", {"form": userform})
```

Зверніть увагу, що коли is_valid() повертає False, об'єкт форми містить інформацію про помилки, які будуть автоматично відображені в шаблоні.

Основні параметри валідації

- **required=False**: робить поле необов'язковим.
- **min_length / max_length**: для текстових полів, задає мінімальну та максимальну довжину.
- **min_value / max_value**: для числових полів, задає мінімальне та максимальне значення.

Завдання 3: Валідація даних

Доповніть форму реєстрації на подію з першого завдання валідацією.

- **Файл forms.py**: Створіть клас RegistrationForm з полями для імені, прізвища та електронної пошти. Встановіть мінімальну довжину для імені (3 символи) і переконайтесь, що електронна пошта є обов'язковим полем.
- **Файл views.py**: Обробляйте POST-запит. Якщо форма валідна, перенаправте користувача на сторінку підтвердження, інакше знову покажіть форму з помилками.
- **Файл urls.py**: Оновіть маршрути відповідно до вашої логіки.

Стилізація форм

Хоча Django генерує HTML-розмітку для форм, ви можете повністю контролювати їх

зовнішній вигляд за допомогою CSS.

Способи стилізації:

1. **Ручне виведення полів**: Ви можете перебрати поля форми в циклі та застосувати власні класи та стилі.

```
{% for field in form %}
    <div class="form-group">
        {{ field.label_tag }}
        <div>{{ field }}</div>
        {% if field.errors %}
            {% for error in field.errors %}
                <div class="alert alert-danger">
                    {{ error }}
                </div>
            {% endfor %}
        {% endif %}
    </div>
{% endfor %}
```

2. **Атрибути required_css_class та error_css_class**: Дозволяють автоматично додавати класи для обов'язкових полів та полів з помилками.

```
# forms.py
class UserForm(forms.Form):
    name = forms.CharField()
    age = forms.IntegerField()
    required_css_class = "field"
    error_css_class = "error"
```

3. **Використання віджетів**: Ви можете вказати віджет для поля, який додасть потрібні HTML-атрибути, такі як class або style.

```
# forms.py
```

```
class UserForm(forms.Form):
    name = forms.CharField(widget=forms.TextInput(attrs={"class":
"myfield"}))
    age = forms.IntegerField(widget=forms.NumberInput(attrs={"class":
"myfield"}))
```

Завдання 4: Стилізація форми

Візьміть форму з попереднього завдання і застосуйте до неї власні стилі.

- **Файл forms.py**: Задайте віджет forms.EmailInput для поля електронної пошти і додайте йому клас form-control.
- **Файл register_form.html**: Переберіть поля форми в циклі та застосуйте власну розмітку. Використайте field.label_tag та field для відображення мітки та самого поля. Додайте блок для виведення помилок.
- **Файл urls.py**: Переконайтеся, що маршрути працюють коректно.

Додаткові теми

1. Відображення помилок валідації

Django автоматично додає повідомлення про помилки, але ви можете контролювати, як вони відображаються.

- **form.errors**: Цей словник містить помилки, пов'язані з кожним полем. Ви можете перебрати його в циклі.

```
{% if form.errors %}
    <h2>Будь ласка, виправте наступні помилки:</h2>
    <ul>
    {% for field, errors in form.errors.items %}
        <li>{{ field }}: {{ errors|join:", " }}</li>
    {% endfor %}
    </ul>
{% endif %}
```

- **field.errors**: Це спеціальний об'єкт, який містить список помилок для конкретного поля. Це зручно, коли ви вручну відображаєте кожне поле.

2. Більше прикладів полів та отримання даних

Ось ще кілька корисних типів полів і приклади того, як отримати з них значення за допомогою form.cleaned_data.

- **Файл forms.py**

```
from django import forms

class SurveyForm(forms.Form):
    # Поле для вибору з кількох варіантів (випадаючий список)
    favorite_language = forms.ChoiceField(
        label="Улюблена мова програмування",
        choices=[('py', 'Python'), ('js', 'JavaScript'), ('c#', 'C#')]
    )

    # Поле для вибору кількох варіантів (множинний вибір)
    frameworks = forms.MultipleChoiceField(
        label="Знайомі фреймворки",
        choices=[('dj', 'Django'), ('re', 'React'), ('fl', 'Flask')],
        widget=forms.CheckboxSelectMultiple
    )

    # Поле для булевого значення (прапорець)
    has_experience = forms.BooleanField(
        label="Чи маєте досвід?",
        required=False  # Робимо необов'язковим
    )

    # Поле для дати
    start_date = forms.DateField(
        label="Дата початку",
        widget=forms.DateInput(attrs={'type': 'date'})
    )
```

- **Файл views.py**

```
from django.shortcuts import render
from django.http import HttpResponse
from .forms import SurveyForm
```

```
def survey_view(request):
    if request.method == "POST":
        form = SurveyForm(request.POST)
        if form.is_valid():
            # Отримуємо дані з різних полів
            lang = form.cleaned_data.get('favorite_language') # 'py', 'js'
або 'c#'
            frameworks = form.cleaned_data.get('frameworks')   # ['dj',
're', 'fl']
            experience = form.cleaned_data.get('has_experience') # True або
False
            start_date = form.cleaned_data.get('start_date')   #
datetime.date object

            # Приклад виведення даних
            response_text = f"<h2>Ваш вибір:</h2>"
            response_text += f"<p>Мова: {lang}</p>"
            response_text += f"<p>Фреймворки: {', '.join(frameworks)}</p>"
            response_text += f"<p>Досвід: {'Так' if experience else
'Ні'}</p>"
            response_text += f"<p>Дата: {start_date}</p>"
            return HttpResponse(response_text)
        else:
            return render(request, 'survey_form.html', {'form': form})
    else:
        form = SurveyForm()
    return render(request, 'survey_form.html', {'form': form})
```

cleaned_data — це найважливіший об'єкт для отримання даних. Він гарантує, що дані були **провалідовані** та перетворені у відповідний тип (наприклад, рядок у булеве значення або об'єкт дати).

Завдання 5: Обробка різних полів

Створіть застосунок для опитування.

- **Файл forms.py**: Визначте клас FeedbackForm з такими полями:
  1. rating (IntegerField) з min_value=1 та max_value=5.
  2. favorite_color (CharField).
  3. newsletter (BooleanField) з label="Підписатись на розсилку", необов'язкове.
- **Файл views.py**: Створіть функцію feedback_view, яка обробляє POST-запити. Використайте is_valid() і отримайте значення з усіх полів за допомогою

form.cleaned_data.

- **Файл urls.py**: Створіть відповідні маршрути.
- **Файл feedback_form.html**: Створіть шаблон, який відображає форму. Після відправки виведіть отримані значення на екран.