Altıncı Bölüm

Formlar ve Kullanıcı Girişi

Django'da Form Oluşturma

Django'da formlar, kullanıcı girdilerini toplama ve doğrulama sürecini basitleştiren ve güvenli veri işleme sağlayan kritik bir bileşendir. Bu kılavuz, Django'da form oluşturma, işleme ve formlardan yararlanma konularını ele alarak formların web uygulaması geliştirmedeki önemli rolünü ortaya koymaktadır.

Django Formlarına Genel Bakış

Django'nun form sistemi, form öğelerinin oluşturulmasını otomatikleştirir, gönderimleri işler ve kullanıcı girdisini doğrularken CSRF (Siteler Arası İstek Sahteciliği) güvenliği gibi yerleşik korumaları da içerir. Formlar çerçevesi, geliştiricilerin kolayca HTML formları oluşturmasına, kullanıcı verilerini doğrulamasına ve form oluşturmayı özelleştirmesine olanak tanır.

Django'da Form Oluşturma

Django'daki formlar, genel formlar için **django.forms.Form** veya formunuz doğrudan bir Django modeline bağlıysa **django.forms.ModelForm** alt sınıfı kullanılarak tanımlanabilir.

Örnek: Basit Bir İletişim Formu Oluşturma

İşte **django.forms.Form** kullanarak temel bir iletişim formunun nasıl tanımlanacağı:

```
from django import forms

class ContactForm(forms.Form):
    name = forms.CharField(max_length=100)
    email = forms.EmailField()
    message = forms.CharField(widget=forms.Textarea)
```

Bu sınıf, uygun veri doğrulamasını sağlamak için **CharField** ve **EmailField** gibi yerleşik alanları kullanarak ad, e-posta ve mesaj alanlarına sahip bir form tanımlar.

Örnek: Model Form Oluşturma

Doğrudan Django modelleriyle ilgili formlar için **ModelForm** kullanılabilir. Aşağıdaki modele sahip olduğunuzu varsayalım:

```
from django.db import models

class Subscriber(models.Model):
    email = models.EmailField()

from django.forms import ModelForm

class SubscriberForm(ModelForm):
    class Meta:
        model = Subscriber
        fields = ['email']
```

SubscriberForm, modelin e-posta alanı için otomatik olarak bir form alanı oluşturacaktır.

Form Gönderimini Yönetme

Formlar genellikle görünümlerde işlenir. Yaygın bir yaklaşım, görünümdeki istek yöntemini kontrol etmeyi, GET istekleri için boş bir form görüntülemeyi ve POST'ta veri gönderimini işlemeyi içerir.

İşte bir iletişim formunu yöneten bir görünüm:

```python
from django.shortcuts import render, redirect
from .forms import ContactForm

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            # Handle the valid form data, such as sending an email or saving to a database
            return redirect('success_url')  # Redirect to a new URL after handling
    else:
        form = ContactForm()  # Display an empty form

    return render(request, 'contact.html', {'form': form})
```

Şablonlarda Formları Görüntüleme

Formlar şablonlarda doğrudan oluşturulabilir. Formun tamamını tek bir form olarak görüntüleyebilir veya daha hassas kontrol için her alanı ayrı ayrı işleyebilirsiniz.

Örneğin, **contact.html** şablonunda:

```html
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Send</button>
</form>
```

{{form.as_p }} yöntemi, her form alanını **<p>** etiketlerine sarılmış **olarak** oluşturur.

Formlar için Özel Doğrulama

Django kapsamlı doğrulama özelliklerini destekler. Özel doğrulama, **clean_<fieldname> ()** gibi yöntemler tanımlanarak doğrudan form alanlarına eklenebilir.

Örneğin, bir e-posta alanına özel doğrulama eklemek için:

```python
class ContactForm(forms.Form):
    ...
    def clean_email(self):
        email = self.cleaned_data.get('email')
        if "example.com" not in email:
            raise forms.ValidationError("Please use your example.com email address")
        return email
```

Sonuç

Django formları, HTML formlarını yöneterek, kullanıcı girdilerini doğrulayarak ve güvenliği artırarak karmaşık web geliştirme görevlerini kolaylaştırır. Güvenli formları hızlı bir şekilde oluşturma ve dağıtma yeteneği, geliştiricilerin kullanıcı merkezli özellikler oluşturmaya daha fazla odaklanmasına olanak tanır, böylece web uygulamalarının işlevselliğini ve etkileşimini artırır. Django formlarını etkili bir şekilde kullanmak modern, sağlam ve verimli web uygulamaları oluşturmak için çok önemlidir.

Form Doğrulama ve Güvenlik

Form doğrulama ve güvenlik, kullanıcı girdilerinin doğruluğunu teyit etmek ve uygulamaları potansiyel tehditlerden korumak için çok önemli olan web geliştirmenin temel unsurlarıdır. Yaygın olarak kullanılan bir Python web çerçevesi olan Django, bu temel görevleri basitleştiren sağlam bir dizi araç sunarak geliştiricilerin veri bütünlüğünü sağlamasına ve güvenlik önlemlerini verimli bir şekilde geliştirmesine olanak tanır.

Form Doğrulamanın Temelleri

Formların doğrulanması, kullanıcılar tarafından gönderilen verilerin işlenmeden veya saklanmadan önce beklenen parametrelerle uyumlu olup olmadığının kontrol edilmesini içerir. Bu adım, hataları önlemek ve uygulamayı SQL enjeksiyonları gibi yaygın istismarlardan korumak için hayati önem taşır.

Django, her alan türü için önceden tanımlanmış doğrulama kuralları ile donatılmış form sınıfları ile form doğrulamayı kolaylaştırır. Bu kurallar, belirli uygulamaların benzersiz ihtiyaçlarını karşılamak için özelleştirilebilir.

Temel Doğrulamanın Uygulanması

Django'nun form çerçevesi, geliştiricilerin her alan için veri türlerini ve kısıtlamaları kolayca belirlemelerine olanak tanır. İşte basit bir örnek:

```python
from django import forms

class RegistrationForm(forms.Form):
    username = forms.CharField(max_length=100)
    password = forms.CharField(widget=forms.PasswordInput)
    age = forms.IntegerField(min_value=13)
```

Bu yapılandırma, girdinin uyumluluk için gerekli olabilecek asgari yaş sınırı gibi tanımlanmış gerekliliklere uymasını sağlar.

Gelişmiş Doğrulama Yöntemleri

Django, doğrulama sürecini daha da iyileştirmek için özel doğrulama yöntemlerinin geliştirilmesini de destekler:

```python
class ContactForm(forms.Form):
    email = forms.EmailField()
    verify_email = forms.EmailField()

    def clean(self):
        cleaned_data = super().clean()
        email = cleaned_data.get("email")
        verify_email = cleaned_data.get("verify_email")

        if email != verify_email:
            raise forms.ValidationError("Please ensure the email addresses match.")
```

Bu özel yöntem, formda sağlanan iki e-posta adresinin aynı olup olmadığını kontrol ederek ek bir doğrulama katmanı ekler.

Form Güvenliğini Artırma

Güvenlik formlar karşı tehditler gibi XSS ve CSRF o kritik Web uygulamalarının bütünlüğünü ve güvenliğini korumak için.

Siteler Arası İstek Sahteciliği (CSRF) Koruması

Django, CSRF tehditlerini her formun içine benzersiz bir belirteç ekleyerek azaltır. Bu belirteç, form gönderimlerinin gerçekliğini doğrular:

```
<form method="post">
    {% csrf_token %}
    {{ form }}
    <button type="submit">Submit</button>
</form>
```

csrf_token %} etiketi CSRF korumasını otomatikleştirerek tüm form gönderimlerinin yetkisiz kaynaklara karşı güvenli olmasını sağlar.

Enjeksiyon Saldırılarını Önleme

Django'nun ORM'si, form girdilerinden gelen SQL sorgularını otomatik olarak kaçarak güvenli bir şekilde işler ve böylece SQL enjeksiyon saldırılarını önler.

XSS'ye Karşı Koruma

XSS'ye karşı koruma sağlamak için Django, şablonlardaki tüm çıktıları **{{ value }}** kullanarak otomatik olarak kaçar. Geliştiriciler, diğer araçların yanı sıra Django'nun **|escape** filtresini kullanarak çıktıları daha da güvenli hale getirebilirler.

Sonuç

Form doğrulama ve güvenlik konularında uzmanlaşmak, güvenli ve güvenilir web uygulamaları oluşturmak için vazgeçilmezdir. Django, geliştiricilere form verilerini güvenli ve verimli bir şekilde yönetmek için gelişmiş araçlar sağlar. Django'nun form işleme yeteneklerini kullanmak, uygulamaların yalnızca işlevsel olmasını değil, aynı zamanda çeşitli güvenlik açıklarına karşı güvenli olmasını sağlayarak hem kullanıcı güvenini hem de uygulama kararlılığını artırır.

GET ve POST İsteklerini İşleme

Web geliştirmede, GET ve POST isteklerinin işlenmesinde uzmanlaşmak, duyarlı web uygulamaları oluşturmak için çok önemlidir. GET istekleri genellikle sunucunun durumunu etkilemeden bilgi almak için kullanılır ve güvenli bir şekilde tekrarlanabilmelerini sağlar. POST istekleri ise sunucuya veri iletmek için kullanılır ve genellikle güncelleme veya değişikliklerle sonuçlanır. Güçlü bir Python web çerçevesi olan Django, geliştiricileri bu HTTP yöntemlerini etkili bir şekilde yönetmek için verimli araçlarla donatır.

GET ve POST İsteklerini Keşfetme

GET İstekleri, sunucudan güvenli ve boş bir şekilde veri almak için tasarlanmıştır, yani tekrarlanan istekler tek bir istekle aynı etkiye sahiptir.

POST İstekleri, yeni girişler oluşturmak veya mevcut verileri güncellemek gibi sunucu durumunu değiştirebilecek veri gönderimlerini içerir. Bu isteklerin işlenmesi, veri güvenliğini ve bütünlüğünü korumak için dikkatli olunmasını gerektirir.

Django'nun GET İsteklerine Yaklaşımı

Django, kullanıcı girdilerine dayalı veri alımını işleyen görünümler aracılığıyla GET isteklerinin yönetimini kolaylaştırır. Bir ürün araması için GET isteğini işleyen bir görünümün bu örneğini düşünün:

```
from django.shortcuts import render

def product_search(request):
    query = request.GET.get('query', '')
    if query:
        products = Product.objects.filter(name__icontains=query)  # Assuming a
            Product model is defined
        return render(request, 'product_search.html', {'products': products})
    else:
        return render(request, 'product_search.html')
```

Bu görünüm, GET isteği içinde bir 'sorgu' parametresi arar, bunu bir dizi ürünü filtrelemek için kullanır ve sonuçlarla birlikte uygun sayfayı oluşturur. Herhangi bir parametre belirtilmezse, genel bir arama sayfası yükler.

Django'da POST İsteklerini İşleme

Django, veri değişikliklerini içeren işlemler için çok önemli olan POST isteklerinin güvenli ve etkili bir şekilde işlenmesini sağlar:

```python
from django.shortcuts import render, redirect
from django.views.decorators.http import require_POST
from .forms import ContactForm

@require_POST  # Ensures this view accepts only POST requests
def submit_contact_form(request):
    form = ContactForm(request.POST)
    if form.is_valid():
        form.save()
        return redirect('success_page')  # Redirects to a confirmation page to
            prevent resubmission
    return render(request, 'contact_form.html', {'form': form})
```

Bu görünüm, **@require_POST** dekoratörü kullanılarak yalnızca POST isteklerini işleyecek şekilde güvence altına alınmıştır. Form verilerini doğrulayıp kaydettikten sonra, bir başarı sayfasına yönlendirerek yinelenen gönderim riskini azaltır.

CSRF Korumasının Uygulanması

POST işlemlerinin güvenliğini artırmak için Django, form gönderimlerinin kimliği doğrulanmış kullanıcıdan geldiğini doğrulayan CSRF koruması içerir:

```html
<form method="post">
    {% csrf_token %}
    <!-- Additional form fields here -->
    <button type="submit">Submit</button>
</form>
```

csrf_token %} etiketinin eklenmesi, forma bir CSRF belirteci ekler ve Django, isteğin meşru olduğundan emin olmak için gönderim sırasında bu belirteci doğrular.

Sonuç

GET ve POST isteklerinin düzgün bir şekilde ele alınması, dinamik ve güvenli web uygulamaları geliştirmek için kritik öneme sahiptir. Django'nun kapsamlı araçları, geliştiricilerin bu istekleri etkili bir şekilde yönetmesine olanak tanıyarak uygulamaların hem kullanıcı dostu hem de güvenli olmasını sağlar. Django'nun GET ve POST isteklerini işleme yeteneklerinden yararlanarak, geliştiriciler uygulamalarının sağlam, duyarlı ve yaygın güvenlik tehditlerine karşı korumalı olmasını sağlayabilirler.

Pratik Örnekler: Kullanıcı Kayıt Formu

Bir kullanıcı kayıt formu oluşturmak, birçok web uygulamasında standart bir gerekliliktir ve yeni kullanıcı kayıtlarının verimli bir şekilde yönetilmesine olanak tanır. Kapsamlı çerçevesiyle bilinen Django, güvenli ve işlevsel kayıt formlarının nispeten kolay bir şekilde oluşturulmasını sağlar. Bu kılavuz, Django'da formun yapılandırılması, görünüm mantığının uygulanması, şablonun hazırlanması ve gerekli güvenlik önlemlerinin uygulanmasını içeren temel bir kullanıcı kayıt formu oluşturma sürecini ele alacaktır.

Kullanıcı Kayıt Formunun Yapılandırılması

1. Formu Tanımlayın

Django'nun form yetenekleri güçlüdür, bu da onu veri gönderimlerini işlemek için ideal kılar. Tipik bir kullanıcı kayıt formu kullanıcı adı, şifre ve e-posta gibi alanlar gerektirebilir. **forms.ModelForm** veya **forms.Form** alt sınıflarını kullanarak formu ihtiyaçlarınıza göre özelleştirebilirsiniz. Django'nun yerleşik kullanıcı yönetimi ile daha iyi entegrasyon için, Django'nun kimlik doğrulama sisteminden **UserCreationForm**'u kullanmak avantajlıdır.

İşte **UserCreationForm**'u bir e-posta alanı içerecek şekilde geliştirmenin bir örneği:

```python
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User

class RegistrationForm(UserCreationForm):
    email = forms.EmailField(required=True)

    class Meta:
        model = User
        fields = ("username", "email", "password1", "password2")

    def save(self, commit=True):
        user = super().save(commit=False)
        user.email = self.cleaned_data["email"]
        if commit:
            user.save()
        return user
```

Bu özelleştirilmiş form e-posta alanı ekler ve **kaydetme** yöntemini geçersiz kılarak kullanıcının diğer verileriyle birlikte kaydedilmesini sağlar.

2. Formu Görünümde İşleme

Formun nasıl görüntüleneceğini yönetmek ve gönderim sırasında verileri işlemek için bir görünüm gereklidir. Aşağıda tam da bunu yapan basit bir görünüm yer almaktadır:

```
from django.shortcuts import render, redirect
from .forms import RegistrationForm

def register(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('/login')  # Redirects to login page after successful
                registration
    else:
        form = RegistrationForm()
    return render(request, 'register.html', {'form': form})
```

Bu görünüm formu başlatır ve işler. Verileri doğrular, yeni kullanıcıyı kaydeder ve ardından kayıt başarılı olursa oturum açma sayfasına yönlendirir. Form verileri geçersizse, hataları görüntülemek için kayıt sayfasını yeniden oluşturur.

3. Kayıt Şablonunu Oluşturun

Kayıt formunuz için şablon, form alanlarının gerçek görüntüsünü ele alacak ve gönderimi yönetecektir:

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

{% block content %}
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Register</button>
</form>
{% endblock %}
```

Bu şablon, CSRF saldırılarına karşı koruma sağlamak için **{% csrf_token %}** kullanarak temel şablonun bir bloğu içinde kayıt formunu içerir.

Güvenliğin Sağlanması

Kullanıcı kayıtlarını işlerken güvenlik çok önemlidir. İşte temel güvenlik uygulamaları:

- **CSRF Koruması:** Django'nun yerleşik CSRF koruması, form şablonlarınıza {% csrf_token %} eklenerek kullanılmalıdır.
- **Parola Güvenliği:** Django'nun UserCreationForm'unu kullanmak, parolaların uygun şekilde karma hale getirilmesini sağlayarak kullanıcı kimlik bilgilerini korur.
- **Veri Doğrulama:** Django formları, SQL enjeksiyonları gibi yaygın web saldırılarını önlemeye yardımcı olan alan türlerine göre otomatik olarak doğrulama gerçekleştirir.

Sonuç

Django'nun form işleme ve kullanıcı kimlik doğrulama özellikleri sayesinde Django'da bir kullanıcı kayıt formu uygulamak kolaydır. Form yapılandırması, görünüm kurulumu ve güvenlik için en iyi uygulamalara bağlı kalmak, güvenli, verimli ve kullanıcı dostu bir kayıt sürecinin oluşturulmasını sağlayabilir. Bu sadece web uygulamalarının işlevselliğini artırmakla kalmaz, aynı zamanda sağlam kullanıcı yönetimi ve güvenliğini de destekler.