Working with forms

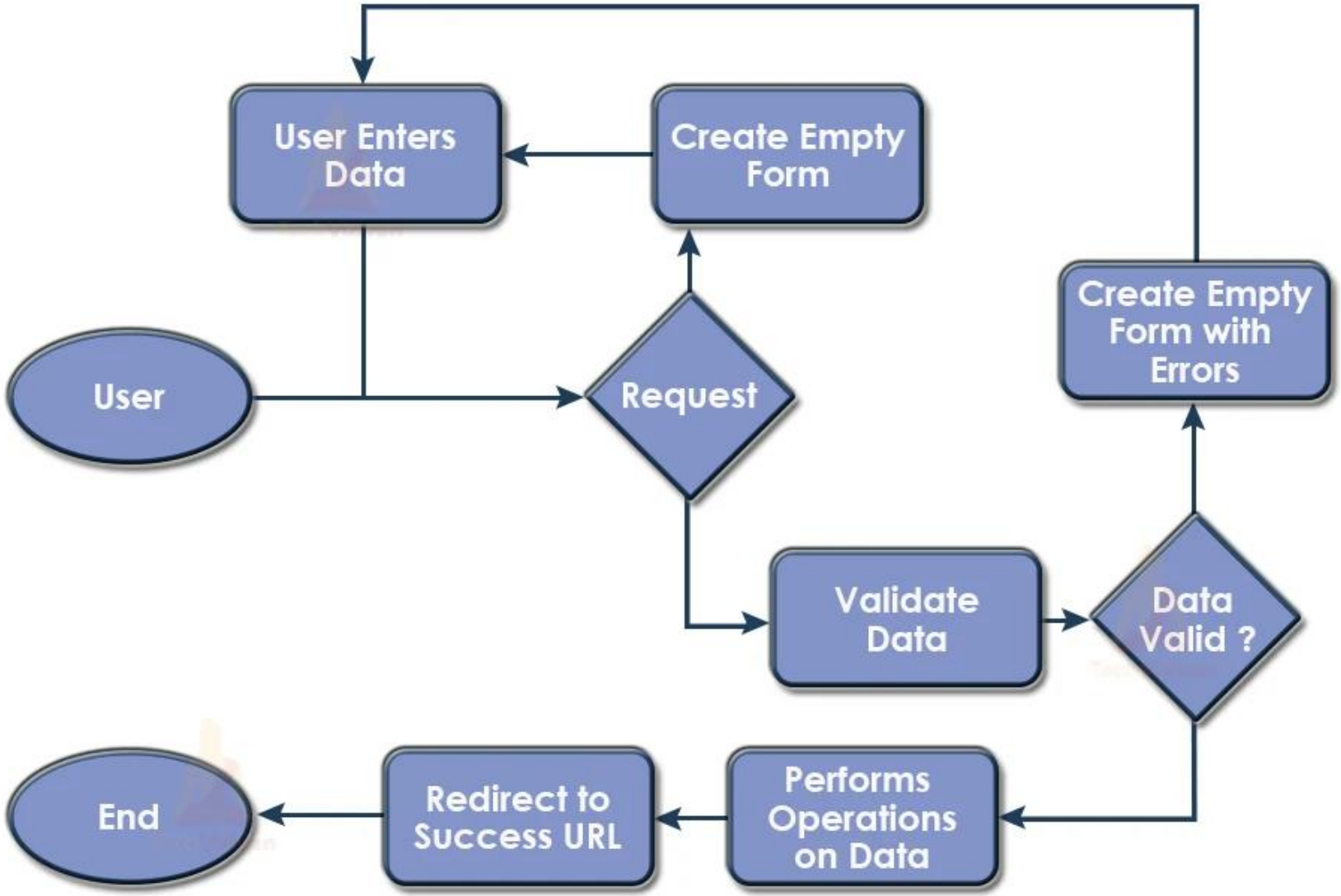
Formulare definieren und nutzen

Was sind Formulare?

In Django werden Formulare verwendet, um Benutzereingaben zu sammeln und zu validieren. Die Validierung ist ein wichtiger Schritt, um sicherzustellen, dass die Daten, die in Ihre Anwendung eingehen, korrekt und sicher sind.

Django bietet eine umfangreiche Infrastruktur für Formulare, einschließlich automatischer Generierung von Formularen aus Modellen, Validierung und Bereinigung von Formulardaten.

Form Handling Process in Django



ein Objekt einfügen

Um ein neues Objekt in die Datenbank einzufügen, bleiben neben der Django-Shell und der Admin-Oberfläche noch weitere Möglichkeiten.

Zum Beispiel über ein eigenes HTML-Formular.

Dieses Formular wird als Template angelegt, mit der HTTP-Methode POST abgesendet und von einer View verarbeitet.

Im Fehlerfall wird das Formular samt Inhalt wieder an den User zurückgesendet, ansonsten wird das neue Objekt in die Datenbank eingetragen.

Der Formularprozess

Um ein neues Objekt mit Hilfe eines Formulars anzulegen, sind mindestens folgende Teile notwendig:

- eine URL in urls.py
- eine View in views.py
- eine Formularklasse in forms.py
- ein Template

Das Modelform

Wie wir gesehen haben, können wir von einem Model ein Datenbank-Schema ableiten. Wir können aber noch viel mehr, zum Beispiel ein Formular davon ableiten. Wir benötigen nur eine Klasse, die von forms.ModelForm erbt

Dazu legen wir unter company/forms.py eine Formularklasse an:

```
from django import forms
from .models import Company

class CompanyForm(forms.ModelForm):
    class Meta:
        model = Company
        fields = "__all__"
```

und sind damit schon fertig. Diese Formularklasse können wir im Template nutzen, um ein HTML-Formular zu generieren. Sie kann aber noch viel mehr, zum Beispiel die eingehenden Daten validieren. Dazu werden wir später Validierungsfunktionen anlegen.

Die View zum Anlegen eines Objektes

Als nächstes legen wir eine View an, die zum Anlegen eines Objektes dient.

Hier im Beispiel erstellen wir im Fall eines GET-Requests ein leeres Formular und übergeben es dem Kontext der Render-Funktion.

Außerdem legen wir den Pfad zu einem Template fest. Wird die View per HTTP POST aufgerufen, wird das Objekt in der Datenbank gespeichert, falls es valid ist.

```
from .forms import CompanyForm


def company_create(request):
    if request.method == "POST":
        form = CompanyForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect(reverse("company:companies")
    else:
          form = CompanyForm()
     return render(request,
                   "company/company_create.html",
                   {"form":form"})
```

Detail

Sehen wir uns folgenden Teil nochmal genauer an

Das Formular wurde per Submit-Button abgesendet, deshalb ist die Methode POST.

- CompanyForm(request.POST) befüllt ein leeres Formular mit den POST-Daten.
- is_valid() prüft, ob die Eingangsdaten valide sind.
- save() speichert eine neue Instanz des Objekts in der Datenbank und gibt diese Referenz auch gleich zurück.
- redirect("company:companies") führt einen Redirect auf die Company-Übersichtsseite aus.

```
if request.method == "POST":
        form = CompanyForm(request.POST)
        if form.is_valid():
            instance = form.save()
            return redirect("company:companies")
```

Validierung der Formularfelder

In Django kann die Validierung von Formulareingaben anpasst werden, indem die clean_<fieldname>() Methode in der Form-Klasse überschrieben wird.

Diese Methode ermöglicht es, spezifische Validierungslogik für ein bestimmtes Feld zu implementieren. Wenn die Validierung fehlschlägt, kann ein ValidationError mit einer Fehlermeldung geworfen werden.

Beispiel clean_name()

In diesem Beispiel wird bei Absenden des Formulars geprüft, ob der Name ausschließlich Buchstaben enthält.
Die clean-Funktionen müssen den Feldwert zurückgeben!

```
from django.core.exceptions import ValidationError


class EmployeeForm(forms.ModelForm):
    class Meta:
        model = Employee
        fields = ['name', 'company']


    def clean_name(self):
        name = self.cleaned_data['name']
        if not name.isalpha():  # Prüft, ob der Name nur Buchstaben enthält
            raise ValidationError("Der Name darf nur Buchstaben enthalten.")
        return name
```