# Views

Steuerung der Applikation

# Aufgaben von Views

Die sogenannte View (dt. Ansicht) kümmert sich um die Aufbereitung der Daten für den Benutzer.

Sie kann dabei auf die Models zurückgreifen und deren Daten auslesen
und verändern. Wie dabei die Benutzerschnittstelle konkret aussieht, ist der View egal.

Die Aufgabe der Views ist nur, die vom Benutzer abgefragten Daten zu ermitteln,
diese zu verarbeiten und dann an ein sogenanntes Template zu übergeben,
das die eigentliche Anzeige übernimmt.

# Erklärung

Eine Django-View ist eine Python-Funktion, die immer ein Request-Objekt und optional Parameter entgegennimmt, und immer ein Response-Objekt zurückgibt. Einzige Ausnahme ist eine Exception, die ausgelöst wird.

Ein Response-Objekt kann vieles sein: Plain Text, ein Html-Template, ein PDF, ein Bild,…

Die view-Funktion kann jegliche Logik verarbeiten, es sind keine Grenzen gesetzt. Wichtig sind nur Request und Response.

# Steuerung

Eine View dient zur Steuerung der Applikation. Sie kann mit dem Controller aus dem MVC-Pattern verglichen werden.

Beispiel:

Eine Anfrage an eine URL soll verarbeitet werden. Die Daten werden in einer Datenbank gespeichert. Andere Daten werden aus der Datenbank geholt. Es wird ein Template erstellt, welches dem User zurückgegeben wird.

# Wo sind die Views gespeichert

Views werden auf App-Ebene gespeichert, also in unserem Fall unter company. Prinzipiell können views in jeder Datei angelegt werden, aber die streng einzuhaltende Django-Konvention schreibt vor, dass Views in der Datei views.py gespeichert werden.

company/

- migrations
- models.py
- views.py

# Beispiel einer View

```
def single(request, id):
    company = Company.objects.get(pk=id)
    return HttpResponse("das ist eine firma!")
```

An die Funktion single wird das Request-Objekt und eine ID übergeben. Die ID wird aus der URL generiert.

Die View selbst gibt ein HttpResponse Objekt zurück.

Die URL für die View könnte lauten:

http://127.0.0.1/company/3

# Mapping von URLs auf Views

Django stellt einen Mechanismus bereit, um angefragte URLs auf die entsprechende View zu mappen.

Urlconf ist eine Definition von Mustern, um für angefragte Adressen die entsprechende View zu finden.

path('admin/', admin.site.urls)

Diese Zeile bedeutet, dass für jede angefragte Adresse (URL), die mit der Zeichenfolge admin/ anfängt, die entsprechenden URLs importiert werden, in denen dann wieder nachgesehen wird, ob die Adresse mit einer Zeichenkette matcht.

Die URL könnte zum Beispiel so aussehen:

http://127.0.0.1/admin/show

# Muster

Das Muster, um die URL zu erkennen und die View zu bestimmen, ist ein regulärer Ausdruck.

Views.py in my_first_app:

```
urlpatterns = [
    path('first', views.first_view, name='first_view'),
    path('second', views.second_view, name='second_view')
]
```

Im Beispiel hatten wir die urls für die App my_first_app definiert.

Der endgültige Pfad in der URL setzt sich zusammen aus den URLs in der APP und den URLs aus dem Projekt.

http://127.0.0.1:8000/company/first

# Die Path Funktion

In jeder Urls Datei (für jede App bzw. Für das Projekt) nutzen wir die path Funktion.

```
urlpatterns = [
    path('first', views.first_view, name='first_view'),
]
```

Die path-Funktion ist dafür zuständig, aus der gefundenen Route die entsprechende View zu suchen.

Um path() zu nutzen, wird diese von django.urls importiert:

from django.urls import path

Die Path-Funktion wird mit drei wichtigen Argumente aufgerufen:

Route (zb. 'first') => ist ein String, der die Route angibt. Django sucht in allen Apps nach Routen.

View (zb. views.first_view) => die angesteuerte View, die aufgerufen wird, wenn die Route gefunden wurde

Name (zb. first_view) => gibt dem Path-Objekt einen Namen. Sollte so heissen, wie die View.

# Mapping auf Views

```
from . import views


urlpatterns = [
    path('first', views.first_view, name='first_view'),
    path('second', views.second_view, name='second_view')
]
```

Das Mapping auf die konkrete View finden auch in der path-Funktion statt, und zwar als zweites, positionelles Argument. Wir importieren die Views aus der App-Ebene, und weisen der Pfad-Funktion die Funktionsreferenz zu (zb. second_view)

# URLs mit Parametern

Wir können in den URLs auch Parameter mit angeben, zum Beispiel die ID einer Firma. In Django versucht man, auf den Query-String weitesgehend zu verzichten, und alles über den URL-Pfad zu lösen.

http://127.0.0.1/company/3

Statt

http://127.0.0.1?company=3

# URLs mit Parametern

http://127.0.0.1/company/3

Ensprechend unserer Url-Conf soll diese URL auf die View single() mappen. Parameter, d.h. variable Werte werden in spitzen Klammern inkl. dem Datentyp notiert.

```
urlpatterns = [
    path('company/<int:id>', views.single, name='single'),
]
```

Zusätzlich und aus Gründen der good practice übergeben wir path() noch das benannte Argument name.

# URLs mit Parametern

```
urlpatterns = [
    path('company/<int:id>', views.single, name='single'),
]
```

Urlconf ruft die View auf, übergibt ihr das Request-Objekt und die Paramter, die in path definiert wurden. Die entsprechende View würde dann so aussehen.

```
def single(request, id):
    company = Company.objects.get(pk=id)
    return HttpResponse("das ist Firma " + company.name)
```

# URLs Zusammenfassung

Wir unterscheiden zwischen 1.) Project-URLs und 2.) App-URLs.

Projekturls werden im Projektverzeichnis definiert, dort wo die settings.py liegen. Ihre Aufgabe ist es, ein Muster in der URL zu erkennen, und darauhin die URLs der entsprechenden App zu inkludieren:

```
path('news/', include('news.urls'))
```

```
2.) Die App News hat ihre eigenen URLs, die unter urls.py in news liegen.
```

```
urlpatterns = [
    path('first', views.first_view, name='first_view')
]
```

```
Den Pfad, den Danjgo finden wird, ist /news/first
```

# Rückgabe eines Templates

In der überwiegenden Mehrheit aller Fälle geben view-Funktionen ein HTML-Template zurück, welches mit Werten gefüllt wird (rendering).

Beispiel eines Templates ohne dynamische Inhalte:

```
def hello_world(request):
    template = loader.get_template('company/single_simple.html')
    context = {}
    return HttpResponse(template.render(context, request))
```

# Rückgabe eines Templates: shortcut

Da diese Aufgabe so häufig vorkommt, gibt es für die Rückgabe des Templates einen Shortcut. Dieser wird von django.shortcuts importiert

```
from django.shortcuts import render
```

Und wird wie folgt genutzt:

```
def hello_world(request):
    return render(request, 'company/single_simple.html')
```

Render() zwei Argumente: das Request-Objekt request und den Pfad zum Template. Ein drittes Argument ist der Kontext, in dem Daten im Template gerendert werden. Dabei handelt es sich um ein Dict.

# Template mit Daten rendern

Meistens sollen Daten in ein Template gerendert werden. Dazu übergibt man der Render-Funktion die Daten in einem Kontext-Dict. Diese rendert dann das Template und gibt es als HTML an den User zurück

```
def single(request, id):
    company = company.objects.get(pk=id)
    context = {'my_company_object': company}
    return render(request, 'company/single.html', context)
```

Der Dict-Key von context steht dann im Template als Referenz auf die Objektinstanz zur Verfügung und kann dort anhand der Django-Template-Language genutzt werden. Context-Dict kann natürlich viele Keys mit Objektreferenzen haben.