



Django-Erweiterungen: Schluss mit Routineaufgaben

Keine Wiederholungen

Christian Leubner

Python hat einen kometenhaften Aufstieg hingelegt. In Unternehmen hat sich die Programmiersprache jedoch noch nicht richtig durchgesetzt, dabei fehlt zum produktiven Einsatz meist nicht viel. Django und vor allem seine zahlreichen Add-ons können hier helfen.

-TRACT

- Django ist ein ausgereiftes Framework, das Entwicklern beim Erstellen von Webanwendungen für Unternehmen viel Arbeit abnimmt.



Wer Django benutzen will, muss bereit sein für eine längere Einarbeitung. Für das Gestalten der Frontends benötigt man ein zusätzliches Framework wie Bootstrap.

Das nach dem Jazzgitarristen Django Reinhardt benannte Webframework Django blickt auf eine mehr als 15-jährige Geschichte zurück, seit es „for perfectionists with deadlines“ – so der eigene Slogan – erstmals veröffentlicht wurde. Tatsächlich nimmt es den Entwicklern eine Menge lästige Routinearbeit ab, indem es sich konsequent an die Parole „Don’t Repeat Yourself“ hält. Wie das durch das Verwenden eines einmal deklarierten Models funktioniert, erklärt der Kasten „Django in Kürze“.

An dieser Stelle ist der grundsätzliche Aufbau einer Django-Anwendung nebensächlich. Tutorials dazu finden sich auf der Projektseite (siehe ix.de/zsw6) oder in Büchern [1]. Hier soll es um einige hilfreiche Django-Packages gehen, die beliebige Anwendungen mit wenig Aufwand unternehmenstauglich machen. Als Beispiel dient eine einfache Aufgabenverwaltung, deren Quellcode online verfügbar ist (siehe ix.de/zsw6). Getestet ist die Applikation mit Python 3.8 und Django 3.1. Wer sie ausprobieren möchte, legt den Quellcode in ein neues Verzeichnis, erzeugt eine virtuelle Umgebung mit dem Python-Modul `venv` und installiert anschließend die erforderlichen Packages mit `pip install -r requirements/dev.txt`. Der für die Entwicklung eingebaute Webserver startet dann durch `python manage.py runserver`. Voreingestellt ist ein Benutzer „root“ mit dem Kennwort „root“.

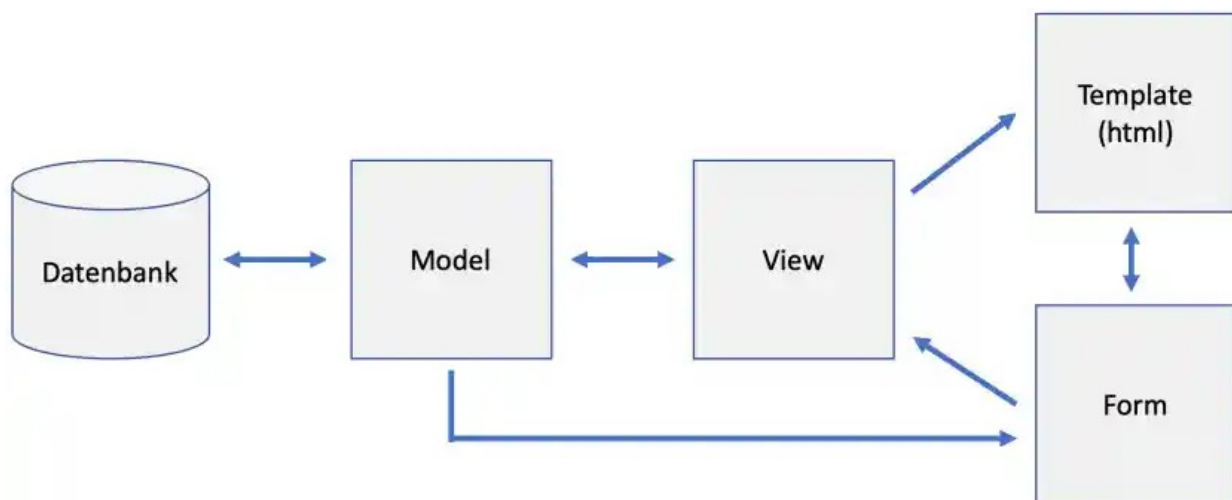
Unternehmensanwendungen müssen Anwender eindeutig authentifizieren und mit den richtigen Berechtigungen ausstatten. Django bringt dazu eine vollständige Benutzer- und Berechtigungsverwaltung mit. Jedoch sind Anwender nur selten begeistert, wenn sie sich einen weiteren Benutzernamen und noch ein Passwort merken müssen. Glücklicherweise bietet das Package `django-python3-ldap` (siehe ix.de/zsw6) eine einfache Möglichkeit, die Anwendung an ein bestehendes Active Directory anzubinden. In der `settings.py` wird unter `AUTHENTICATION_BACKENDS` das Package als „`django_python3_ldap.auth.LDAPBackend`“ hinzugefügt und schon gehen die Log-in-Daten zur Bestätigung an das Active Directory. Netter Nebeneffekt: Die Django-Benutzerverwaltung läuft weiter und lässt sich zusätzlich zum Einrichten lokaler User verwenden.

Berechtigungen pflegt der Admin in einem separaten Bereich, der auf einem gestarteten Webserver im Pfad `/admin` erreichbar ist, bei gestartetem Entwicklungsserver unter `http://localhost:8000/admin`. Die Berechtigungen orientieren sich am Model und ermöglichen für jedes Objekt das Unterteilen nach dem CRUD-Schema: Create, Read, Update, Delete. Jedes de-



Django in Kürze

Ein herausragendes Feature von Django ist das objektrelationale Mapping: Der Entwickler beschreibt seine Objekte (Model) lediglich einmal in Python, um die persistente Ablage kümmert sich das Framework automatisch. Welche Datenbank im Hintergrund arbeitet, ist nebensächlich. Django bietet Plug-ins unter anderem für MySQL und PostgreSQL, neu angelegte Projekte nutzen voreingestellt SQLite.



Django nutzt konsequent das Model und erspart dem Entwickler so viel Arbeit (Abb. 6).

Methoden wie `save` und `get` wandeln Abfragen und Veränderungen an Django-Objekten in SQL-Statements. Auch Fremdschlüsselbeziehungen und Tabellen-Joins lassen sich durch entsprechende Deklaration im Model einfach umsetzen. Letzteres legt die Klassenstruktur und Beziehungen zwischen den Objekten fest, der Abgleich mit der Datenbank erfolgt im Hintergrund (Abbildung 6). Das einmal deklarierte Model lässt sich anschließend als Vorlage für Views und Forms nutzen. Bei den Views sind typische Anwendungssituationen wie das Bearbeiten eines Objekts oder Listenansichten schon im Framework vorhanden und erfordern im Bedarfsfall nur kleine Anpassungen.

Das Darstellen im Browser übernehmen Templates, die aus HTML, CSS und JavaScript bestehen können. Angereichert werden sie mit Djangos Template Language, die Daten in den HTML-Code ausgibt. Durch Schleifen- und `if`-Befehle lassen sich Inhalte dynamisch im Template erzeugen. Für ein einheitliches und modernes Erscheinungsbild im Browser empfiehlt sich der Einsatz eines Frontend-Frameworks wie Bootstrap.

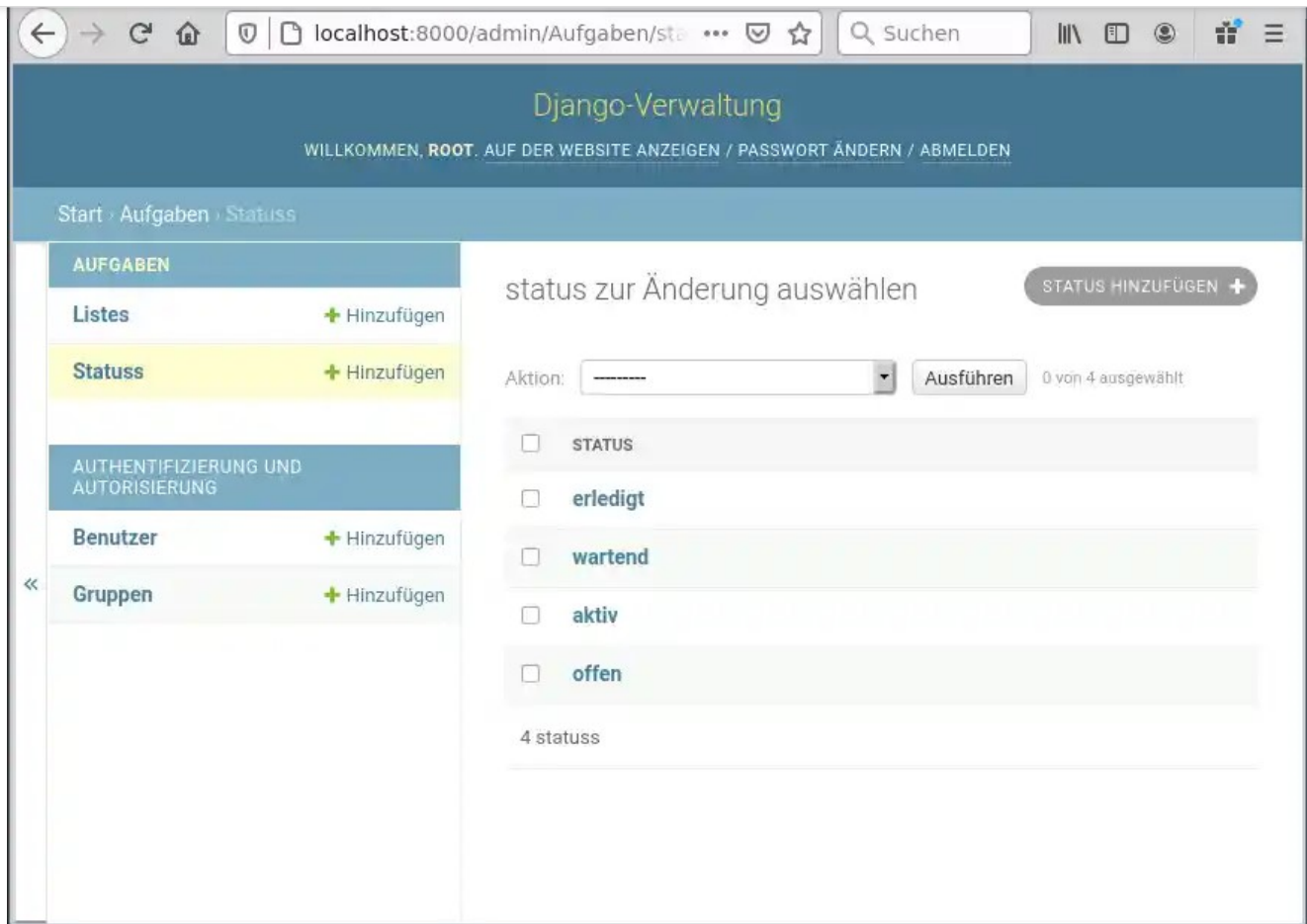
Formulare kann die Entwicklerin ebenfalls einfach mit Bezug zum Model erstellen.

ben. Darüber hinaus stehen für jeden Datentyp passende UI-Elemente (Widgets) wie Drop-down-Auswahlmenüs oder Textfelder zur Verfügung.

Unternehmensanwendungen profitieren von Features zur Mehrsprachigkeit und der Berücksichtigung unterschiedlicher Zeitzonen. Und spezielle Anforderungen können durch Vererbung und Anpassung individuell geändert und ergänzt werden. Dabei unterstützt eine aktive Community, etwa bei Stack Overflow. Zudem existieren zahlreiche Apps, die zusätzliche Funktionen bereitstellen.

Nicht alle sollen Zutritt bekommen

Der Admin-Bereich dient nicht nur der Benutzerpflege. Das Hinzufügen einer Model-Klasse in `admin.py` erzeugt automatisch Anzeige-, Bearbeitungs- und Listendialoge. Im Beispielprojekt ist dies für Status und Liste durch `admin.site.register(Status)` und `admin.site.register(Liste)` erfolgt. Abbildung 1 zeigt die Einträge für Status, die der Admin hier ändern, erweitern und löschen darf. Endanwender haben in diesem Konfigurationsbereich nichts zu suchen. Unabhängig vom sonstigen, an Objekten orientierten Berechtigungsmodell wird der Zugang zu diesem Bereich im Benutzerstamm über die Checkbox „Mitarbeiter“ gewährt. Die Bezeichnung ist unglücklich gewählt, „Key-User“ oder „Anwendungskoordinator“ wären treffendere Rollenbezeichnungen.

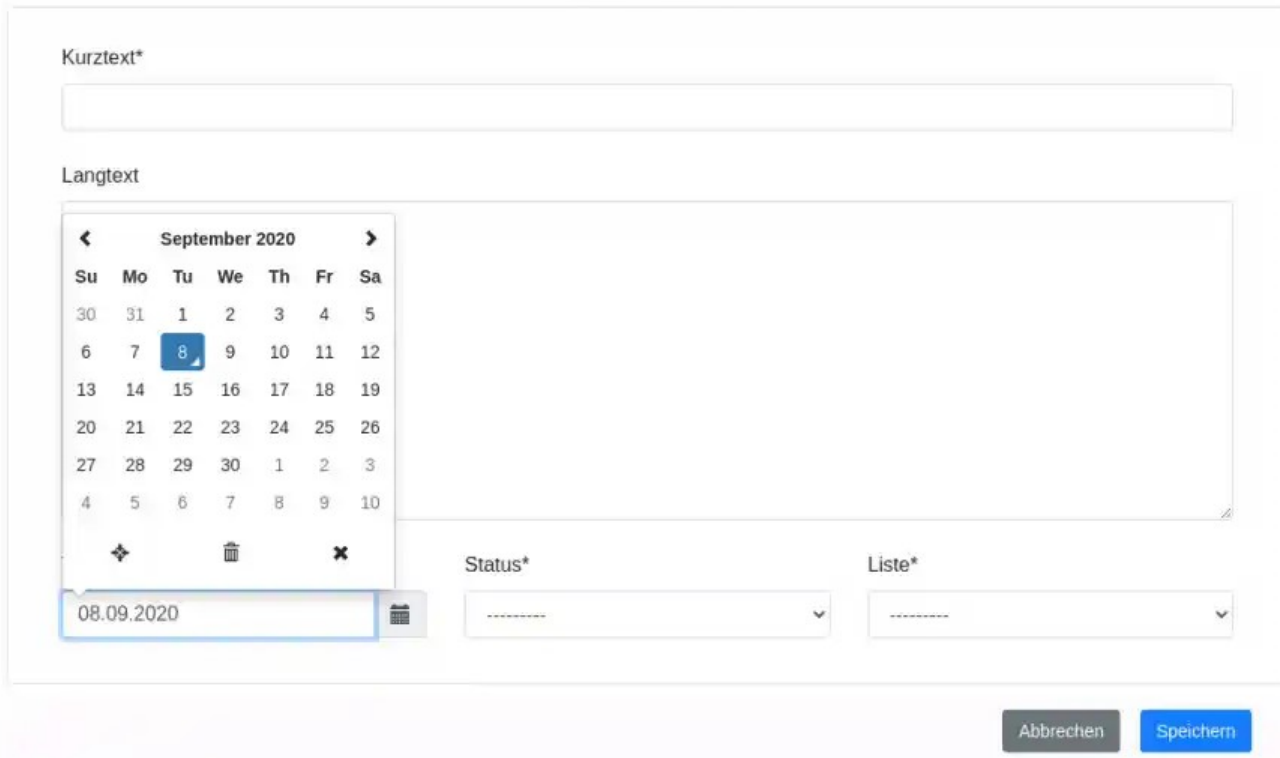


Im Admin-Bereich lassen sich neben Benutzern und Berechtigungen auch beliebige andere Objekte zum Bearbeiten einbinden (Abb. 1).

Wenn die Weboberflächen nett aussehen und auch die User Experience nicht zu kurz kommen soll, helfen gängige CSS-/JavaScript-Frameworks. Insbesondere Bootstrap ist im Django-Universum verbreitet, und viele Add-on-Packages verwenden es (siehe ix.de/zsw6). Während der Kreativität beim Gestalten nur wenige Grenzen gesetzt sind, reichen für Unternehmenszwecke in der Regel aufgeräumte Strukturen, die sich einheitlich durch die Anwendung ziehen und möglichst intuitiv bedienbar sind.

Im Beispielprojekt steht dafür ein fixierter Headerbereich mit Menü zur Verfügung. Die einzelnen Dialoge sind im Wesentlichen mit flexiblen Containern – in Bootstrap auch „fluid“ genannt – gestaltet, die sich dynamisch der Fenstergröße anpassen. Das Anpassen der Oberfläche für verschiedene Endgeräte (Responsive Design) ist beispielhaft für die Überschriften der Dialoge umgesetzt. Beim Verkleinern des Browserfensters springt die Darstellung zwischen einer Lang- und einer Kurzform der Überschriften um (Abbildung 2).

Aufgabe anlegen



Kurztext*

Langtext

September 2020

Su	Mo	Tu	We	Th	Fr	Sa
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3
4	5	6	7	8	9	10

08.09.2020

Status*

Liste*

Abbrechen Speichern

Mithilfe von Bootstrap und Crispy Forms entstehen aufgeräumte Oberflächen (Abb. 2).

Hat die Entwicklerin den grundsätzlichen Rahmen mit Templates gestaltet, kann Django wieder übernehmen. Formulare auf Webseiten lassen sich mit Bezug zu einem Model definieren. Im Beispielprojekt ist das Formular für die Aufgaben in `forms.py` beschrieben. Passende Widgets für Datentypen und Überprüfung auf Pflichteingaben leitet Django automatisch aus dem referenzierten Model ab. Lediglich das Vorlagedatum wird abweichend zum Standard vorgegeben und soll den etwas hübscheren DatePicker aus dem Package `django-bootstrap-datepicker-plus` nutzen (siehe ix.de/zsw6). Mit nur wenigen Codezeilen lässt sich anschließend der komplette View für das Anlegen und Ändern von Aufgaben in `views.py` erstellen. Sowohl der `AufgabeCreateView` als auch der `AufgabeDetailView` erben von Djangos Standard-Views und erledigen ihren Job inklusive Datenabruf und -ablage in der Datenbank ohne zusätzliches Coding. Autorisierungs- und Berechtigungsprüfung erfolgen durch Vererben von `LoginRequiredMixin` und `PermissionRequiredMixin`.

Es geht um das gute Aussehen

Views verbinden Formulare und Daten der Anwendung mit den Templates, die wiederum die HTML-Darstellung und die Interaktion mit den Anwendern abwickeln. Der Austausch ge-

gleiche Template zum Anlegen und Bearbeiten nutzen und die Überschrift durch den aufrufenden View anpassen.

Nicht auslassen sollte man das Package Crispy Forms (siehe ix.de/zsw6), das Formulare aufpeppt, wahlweise mithilfe von Bootstrap. Die komplette HTML-Generierung nach Bootstrap-Manier für ein definiertes Formular passiert im Template via `{% crispy form %}`.

Wer mehr Einfluss auf Darstellung und Anordnung der Formularfelder haben möchte, definiert in HTML eine individuelle Anordnung und greift dafür explizit auf einzelne Felder zu. Für das Feld kurztext zum Beispiel durch `{{ form.kurztext|as_crispy_field }}`.

Eine weitere Aufgabe, die es in nahezu jeder Anwendung zu meistern gilt, ist die Abfrage des Datenmodells, meistens nach Vorgaben und Selektionskriterien, die der Anwender im Dialog bestimmt. Das Ergebnis der Abfrage erscheint in der Regel als Liste oder Tabelle. Auch hier helfen ergänzende Packages, die dem Entwickler viel lästige Tipperei abnehmen. Gut aufeinander abgestimmt sind die Packages `django-filter` und `django-tables2`, die im Beispiel für die Abfrage und tabellarische Auflistung der Aufgaben zum Einsatz kommen (siehe ix.de/zsw6).

Beim Festlegen von Filterkriterien hilft wiederum der Bezug zum Model, sodass in der `filters.py` lediglich die zu berücksichtigenden Felder zu benennen sind. Zusätzlich ergänzen lässt sich der Filtertyp: Während `exact` für Status und Liste die eindeutige Übereinstimmung vorgibt, stößt `icontains` bei Kurztext eine Durchsuchung des Felds nach dem eingegebenen Suchbegriff an. Beim Vorlagedatum weicht das Beispiel wieder vom Standard ab, indem es das bereits zuvor erwähnte `DatePicker`-Widget festlegt.

Das Beispiel greift die gerade für Datumsfelder häufige Situation auf, dass nicht nach einem exakten Datum gesucht werden soll, sondern nach einem Intervall („irgendwann zwischen 01.03.2020 und 15.03.2020“). Mit dem `DatePicker` lassen sich zwei Datumsfelder durch den Zusatz `start_of()` und `end_of()` verknüpfen, sodass bei der Eingabe nur eine sinnvolle Auswahl möglich ist. Der mit nur wenigen Zeilen Code definierte Filter wird in eine aus `FilterView` vererbte Klasse gesteckt und dort mit einem Template verknüpft. Die HTML-Generierung lässt sich ebenso wie bei den Forms mit Crispy Forms aufhübschen.

Zahlreiche sinnvolle Zusatzfunktionen

Das tabellarische Aufbereiten der Ausgabe erledigt das Package `django-tables2`. Auch beim Definieren der Tabellendarstellung wird wieder der Bezug zum Model hergestellt. Seitenweises Blättern bei vielen Ergebnissen, Sortierreihenfolge sowie Darstellungsoptionen lassen sich

thoden `render_kurztext` und `render_vorlage_datum`, dass der Standardzelleninhalt verwendet wird. Der Kurztext wird in einen Link umgewandelt, der in den Dialog zur Bearbeitung der Aufgabe wechselt. Beim Vorlagedatum prüft man, ob es in der Vergangenheit liegt, und färbt es in diesem Fall rot ein.

Listing 1: AufgabeFilterView in wenigen Zeilen

```
class AufgabeFilterView(LoginRequiredMixin, PermissionRequiredMixin, SingleTableMixin):
    template_name = 'liste.html'
    permission_required = 'Aufgaben.view_aufgabe'
    filterset_class = AufgabeFilter
    model = Aufgabe
    table_class = AufgabeTable
```

Das Ergebnis `AufgabeFilterView` in `views.py` besteht inklusive Log-in- und Berechtigungsprüfung, Verweis auf Template, Filter und Tabelle sowie Aufgaben-Model aus gerade einmal sechs Zeilen Code (Listing 1). Dafür kann sich das Ergebnis durchaus sehen lassen (Abbildung 3). Die Drop-down-Boxen für Status und Liste sind ohne weiteres Zutun mit den möglichen Werten aus dem Datenmodell vorbelegt. Die Datumsfelder öffnen bei Klick auf das Kalendersymbol einen DatePicker, der durch die Verknüpfung der beiden Felder darauf achtet, dass das „Von Datum“ vor oder gleich dem „Bis Datum“ ist. Auch hinter der HTML-Darstellung der Tabelle liegt ein Template, von denen das Package `django-tables2` mehrere zur Auswahl stellt. Standardmäßig wird auch hier ein Bootstrap-Template verwendet.


Aufgabensuche

Kurztext enthält


Status

Liste

Von Datum...



...bis Datum



Abbrechen

Suchen

Kurztext	Vorlage	Status	Liste
Müll rausbringen	01.09.2020	erledigt	Zuhause
Beiträge für Kaffeekasse einsammeln	13.09.2020	wartend	Büro
Fenster putzen	05.10.2020	aktiv	Zuhause
Flur putzen	05.10.2020	aktiv	Zuhause

Die Packages `django-filter` und `django-tables2` ergänzen sich ideal beim Erstellen flexibler Abfragen und tabellarischer Ausgaben (Abb. 3).

Während die Entwicklerin auf der Programmierenebene mit erstaunlich wenigen Zeilen Code respektable Ergebnisse hervorzaubern kann, muss sie für die HTML-Templates und das Frontend deutlich mehr Arbeit investieren. Auf der anderen Seite gibt es gerade für HTML5 zahlreiche Frameworks wie Vue.js oder Angular, die sich dank intensivem JavaScript-Einsatz deutlich mehr nach lokaler Anwendung als nach Webseite anfühlen. Auch wenn man ein Python-Programm in eine Webservice-Landschaft einbinden will, für die es schon Oberflächen, Benutzerverwaltung und Ähnliches gibt, kann Django mit dem Package Django-Rest-Framework helfen (siehe ix.de/zsw6). Wie der Name vermuten lässt, rüstet dieses Package Anwendungen mit REST-Schnittstellen auf, die von anderen Anwendungen konsumiert werden können.

Nach dem Installieren des Package wird für die gewünschten Objekte ein Serializer in `serializers.py` angelegt. Auch hier reduziert sich die Tipparbeit des Entwicklers, wenn er den Bezug zu einem Model herstellt. Der vorliegende Fall zeigt das Vorgehen beispielhaft für das Aufgabenobjekt und gibt die im Datenmodell verknüpften Informationen zu Status und Liste als verschachteltes Objekt zurück. Daher werden hier auch Serializer für Status und Liste angelegt, auf die der Aufgaben-Serializer verweist (Listing 2)



```
class AufgabeSerializer(serializers.ModelSerializer):
    status = StatusSerializer(
        many=False,
        read_only=True,
    )
    liste = ListeSerializer(
        many=False,
        read_only=True,
    )

    class Meta:
        model = Aufgabe
        fields = ('id',
                  'kurztext',
                  'langtext',
                  'vorlage_datum',
                  'liste',
                  'status',
                  )
```

Nun braucht man nur noch Views, die die HTTP-Anfragen annehmen und verarbeiten. Hier sind dies `AufgabeApiListe` und `AufgabeApiDetail`, die entweder eine Liste aller Aufgaben zurückgeben oder über das Feld `id` eine bestimmte Aufgabe bereitstellen. Beide nutzen dafür den Aufgaben-Serializer.

Welche Operationen auf der Liste und dem einzelnen Objekt über REST verfügbar sein sollen, legt die Basisklasse fest. `AufgabeApiListe` erbt von `ListCreateAPIView` und ermöglicht auch die Neuanlage einer Aufgabe per POST. `AufgabeApiDetail` erbt von `RetrieveUpdateDestroyAPIView`, sodass neben dem Abrufen einer Aufgabe auch das Bearbeiten und Löschen möglich ist. Der Entwickler profitiert davon, weil er die REST-Schnittstellen direkt im Browser testen kann. Im Beispielprojekt ist die URL `http://localhost:8000/aufgabe/api` definiert (Abbildung 4).

Django REST framework

root

Aufgabe Api List

Aufgabe Api List

OPTIONS

GET

GET /aufgabe/api/

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

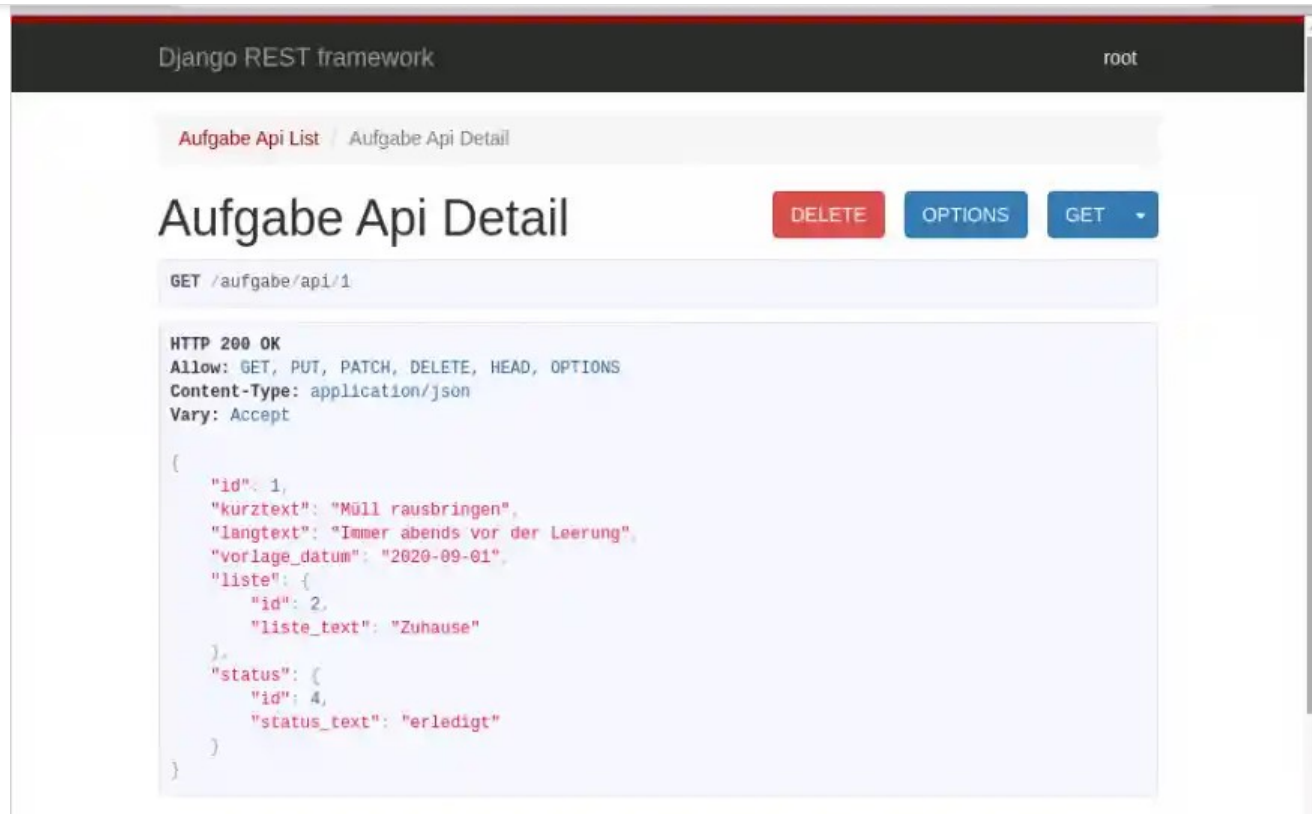
```
[
  {
    "id": 1,
    "kurztext": "Müll rausbringen",
    "langtext": "Immer abends vor der Leerung",
    "vorlage_datum": "2020-09-01",
    "liste": {
      "id": 2,
      "liste_text": "Zuhause"
    },
    "status": {
      "id": 4,
      "status_text": "erledigt"
    }
  }
]
```

REST-Schnittstellen sind nicht nur schnell angelegt, sondern können auch im Browser geprüft werden (Abb. 4).

Weiter unten und nicht mehr im Bild zu sehen liegen außerdem Eingabefelder für die Neuanlage per POST. Einzelne Aufgaben, zum Beispiel die mit der Id 1, lassen sich nach REST-Logik direkt ansprechen: `http://localhost:8000/aufgabe/api/1` (Abbildung 5). Wohlgedenkt gibt es auch diese Oberflächen frei Haus und ohne weiteres Handanlegen. Beide Views sind mit nur wenigen Zeilen erstellt (Listing 3) und regeln neben den möglichen REST-Operationen auch die Nutzerauthentifizierung und die Berechtigungsprüfung. Wer zusätzlich eine Filtermöglichkeit benötigt, kann sie über das Package `django-filter` einbauen, das gut mit dem REST-Framework zusammenspielt.

Listing 3: Views mit wenigen Zeilen erstellt

```
class AufgabeApiDetail(generics.RetrieveUpdateDestroyAPIView):
    authentication_classes = (BasicAuthentication, SessionAuthentication,)
    permission_classes = (
        permissions.DjangoModelPermissions,
    )
    queryset = Aufgabe.objects.all()
    serializer_class = AufgabeSerializer
```



Je nach genutzter Basisklasse ist auch das Ändern und Löschen möglich, inklusive Berechtigungsprüfung (Abb. 5).

Fazit

Django hat sich einen guten Ruf erworben. Durch den modellbasierten Entwicklungsansatz, das objektrelationale Mapping und die konsequente Bezugnahme auf das Model entstehen mit wenig Aufwand solide und gut wartbare Webanwendungen. Erweiterungspakete ergänzen die im Standard fehlenden Funktionen, die neben der Anbindung eines Active Directory auch flexible Datenselektionen und tabellarische Ausgaben zur Verfügung stellen.

Hakeliger gestaltet sich die Sache bei den HTML-Frontends, da Django's Standard-HTML nicht überzeugt. Im Beispielprojekt kam daher Bootstrap zum Einsatz. Alternativ lassen sich recht einfach REST-Schnittstellen erstellen, die nicht nur als Schnittstelle zu weiteren Anwendungen dienen können, sondern auch zum Einbinden in andere Frontend-Frameworks. Hat man sich einmal im Django-Universum eingewöhnt, lassen sich Python-Programme mit vergleichsweise wenig Aufwand in eine zuverlässige unternehmenstaugliche Hülle stecken. (jd@ix.de)

Quellen

[1] Tom Aratyn; Building Django 2.0 Web Applications; Packt Publishing, 2018



berät Unternehmen im Bereich Instandhaltung und Materialwirtschaft.

■ Kommentieren



Leserbrief schreiben



Auf Facebook teilen



Auf Twitter teilen

Kontakt

Impressum

Datenschutzhinweis

Nutzungsbedingungen

Metadaten

