das Model

Schnittstelle zur Datenbank

Was ist ein Modell?

Ein Model ist der Ort, an dem Informationen gespeichert werden. Es beinhaltet alle wichtigen Felder und Verhaltensweisen dieser Informationen.

Modelle existieren nur auf App-Ebene und dort jeweils in der Datei models.py. Da eine App durchaus mehr als ein Modell hat (zb. Artikel und Kommentare), werden in der models.py mehrere Klassen stehen.

Grundsätzlich wird jedes Modell auf genau eine Datenbank-Tabelle abgebildet.

Beispiel für ein Model

Ein Blog-Artikel könnte ein Model sein. Der Artikel hat eine Headline, eine Subline, Content, ein Bild, einen Autor usw. Alle diese Attribute werden in einem Model verwendet.

Technisch gesehen ist ein Modell eine Python-Klasse, die von models.Model erbt und in models.py definiert ist.

```
class Article(models.Model):
    headline = models.CharField(max_length=100)
    subline = models.CharField(max_length=200)
    article_img = models.CharField(max_length=40)
    content = models.TextField()
    pub_date = models.DateTimeField(auto_now_add=True)
```

SQL

Auf Basis der Model-Definition wird eine Datenbank-Tabelle angelegt.
Das SQL dazu sieht etwa so aus:

```
CREATE TABLE pets_pet (
    "id" serial NOT NULL PRIMARY KEY,
    "headline" varchar(100) NOT NULL,
    "subline" varchar(200) NOT NULL,
    "article_img" varchar(40) NOT NULL,
    "content" TEXT NOT NULL,
    "pub_date" DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Migration

Um die Modell-Definition in das SQL für die Datenbank zu übertragen, d.h. daraus ein SQL-Statement zu formen, das die entsprechende Datenbanktabelle erstellt (oder ändert), sind sogenannte Migrationen notwendig.

Datenbank-Änderungen werden NIEMALS manuell vorgenommen.

Migrationen sind Dateien, die auf Basis der Modell-Definition erstellt und dann ausgeführt werden.

Vorteil von Migrationen

Datenbank-Tabellen entsprechen immer genau den Models (Klassen)

Migrationen werden in der Software versioniert und können jederzeit ausgeführt werden, zum Beispiel wenn ein anderer Entwickler das Projekt bei sich lokal installiert.

Migrationen sind abstrakt, d.h. sie können auf jeder von Django unterstützten DB ausgeführt werden (ORM).

Eine Migration erstellen

Migrationen erstellt man appspezifisch mit dem Verwaltungstool manage.py und dem Subkommando makemigrations, gefolgt von dem Namen der Django App

python manage.py makemigrations pets

Unter pets/migrations wurde jetzt eine Datei erstellt:

0001_initial.py

der Präfix 0001 ist eine fortlaufende Nummer und wird automatisch hochgezählt

Beispiel einer Migration

```
from django.db import migrations, models

class Migration(migrations.Migration):
    initial = True
    dependencies = [
    ]
    operations = [
        migrations.CreateModel(
            name='FirstModel',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('title', models.CharField(max_length=100)),
                ('text', models.TextField(verbose_name='Meldungstext')),
            ],
        ),
    ]
```

Migrationsdateien werden nur in seltenen Ausnahmen manuell geändert!

SQL

Um sich das generierte SQL anzusehen, welches eine Migration erzeugen würde, können wir das Subkommando sqlmigrate nutzen.

Zwingend notwendig sind hier a) die Angabe der app (pets) und b) der Nummer der Migration (d. ist der Dateipräfix der Migrationsdatei)

python manage.py sqlmigrate pets 0001

Migration in Datenbank übertragen

Mit makemigrations haben wir nur eine Migrationsdatei erstellt! Um diese in ein SQL-Statement zu überführen, nutzen wir das Subkommando migrate des Managementools manage.py

python manage.py migrate pets

Jetzt wurde in der Datenbank eine neue Tabelle erstellt (oder verändert, wenn sie schon existiert hat).