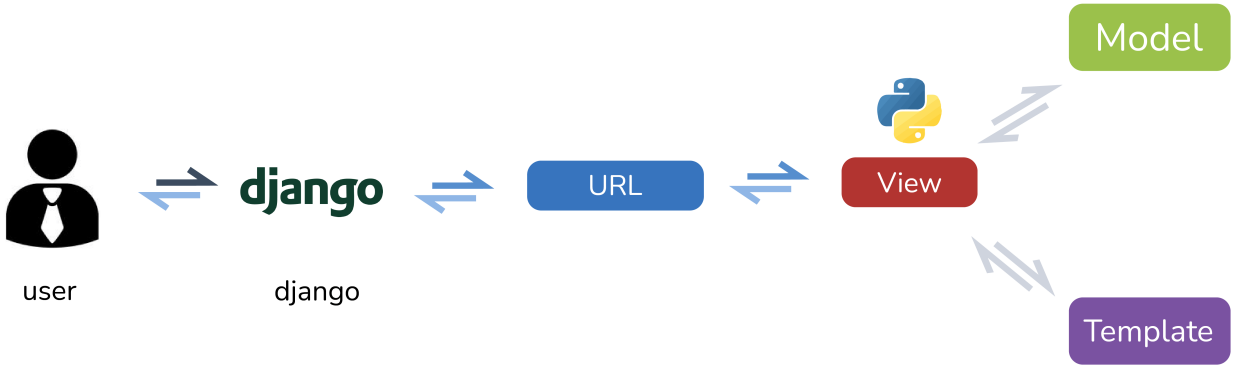
Django Models

Overview



A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table.

The basics

- Each model is a Python class that subclasses ***django.db.models.Model***
- Each attribute of the model represents a database field.
- Django gives you an automatically-generated database-access API.

A simple model

Here's a model that creates a simple db table:

```
from django.db import models

class Musician(models.Model):
    # The only required part of a model is the list of database fields it defines.
Fields are specified by class attributes. Don't use clean, save, or delete.
    first_name = models.CharField(max_length=50)
    # field = field type(field options)
    last_name = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)

class Album(models.Model):
    artist = models.ForeignKey(Musician, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars = models.IntegerField()
```

Field types

Each field in your model should be an instance of the appropriate field class.

Django uses the field class types to determine a few things:

- The column type, which tells the database what kind of data to store (e.g. INTEGER, VARCHAR, TEXT).
- The default HTML widget to use when rendering a form field (e.g. <input type="text">, <select>).
- The minimal validation requirements, used in Django's admin and in automatically-generated forms.

Django ships with dozens of built-in field types; you can find the complete list in the model field reference. You can easily write your own fields if Django's built-in ones don't do the trick.

Commonly used Field Types

- **CharField:**

A string field, for small- to large-sized strings. The default form widget for this field is a TextInput.

CharField has a required argument "max_length". (The maximum length (in characters) of the field.) The max_length is enforced at the database level and in Django's validation using MaxLengthValidator.

- **DateTimeField:**

A date, represented in Python by a datetime.date instance. The default form widget for this field is a DateInput. Has a few extra, optional arguments:

auto_now:

Automatically set the field to now every time the object is saved.

auto_now_add:

Automatically set the field to now when the object is first created.

NOTE: DateField and DateTimeField is almost similar except adding time with date in DateTimeField.

- **EmailField:**

A CharField that checks that the value is a valid email address using EmailValidator. By default max_length=254.

- **FileField:**

A file-upload field. Mostly used argument:

upload_to:

This attribute provides a way of setting the upload directory and file name.

```
# file will be uploaded to MEDIA_ROOT/uploads
upload = models.FileField(upload_to='uploads/')
```

- **ImageField:**

Inherits all attributes and methods from FileField, but also validates that the uploaded object is a valid image. In addition to the special attributes that are available for FileField, an ImageField also has height and width attributes.

- **IntegerField:**

An integer. Values from -2147483648 to 2147483647 are safe in all databases supported by Django.

It uses MinValueValidator and MaxValueValidator to validate the input based on the values that the default database supports.

The default form widget for this field is a NumberInput

- **TextField:**

A large text field. The default form widget for this field is a Textarea.

If you specify a max_length attribute, it will be reflected in the Textarea widget of the auto-generated form field. However it is not enforced at the model or database level. Use a CharField for that.

CharField(TextInput) vs TextField(TextArea)

An TextInput field is a one-line field (probably to carry something like first name or last name, a phone number, an email).

A TextArea is a multi-line field that allows you to press ENTER! They are used for addresses or others long and complex type of data (also notes, for instance).

- **ForeignKey:**

A many-to-one relationship. Requires two positional arguments: the class to which the model is related and the on_delete option.

Field types documentation

Field options

Field options (or arguments) will determine the behaviour of the field.

Commonly used Field options

- **null:**

If True, Django will store empty values as NULL in the database. Default is False.

- **blank:**

If True, the field is allowed to be blank. Default is False.

- **choices:**

A sequence consisting itself of iterables of exactly two items (e.g. [(A, B), (A, B) ...]) to use as choices for this field.

The first element in each tuple is the actual value to be set on the model, and the second element is the human-readable name.

- **default:**

The default value for the field.

- **primary_key:**

If True, this field is the primary key for the model.

- **unique:**

If True, this field must be unique throughout the table. This is enforced at the database level and by model validation.

- **verbose_name:**

A human-readable name for the field. If the verbose name isn't given, Django will automatically create it using the field's attribute name, converting underscores to spaces.

Field options documentation