



Technologies and Web Programming

The Django Platform



Django Platform

Models

Django's Database Layer



model

- MTV – *Model, Template, View*
- Model
 - Consists of the data access layer - “*Data Access Layer*”
 - This layer allows you to define, in relation to data:
 - the access;
 - the validity;
 - the behavior;
 - Relationships between data.

DB Configuration



- Defining access to data starts with configuring access to the database.
- In the “settings.py” file, look for the variable `DATABASES` and define the parameters `ENGINE`, `NAME`, `USER`, `PASSWORD`, `HOST`, `PORT`.
 - Some of these parameters can be omitted, depending on the DB to be accessed.
- By default, when creating the project, access to the local SQLite database is configured. For other DBs, consult the documentation:
 - <https://docs.djangoproject.com/en/?..?/topics/db/multi-db/>

Creating a model (i)



- In the “models.py” file
from the “app” folder,
define the classes of
data model a
to use.

```
models.py x
1 from django.db import models
2
3 class Author(models.Model):
4     name = models.CharField(max_length=70)
5     email = models.EmailField()
6     def __str__(self):
7         return self.name
8
9
10 class Publisher(models.Model):
11     name = models.CharField(max_length=70)
12     city = models.CharField(max_length=50)
13     country = models.CharField(max_length=50)
14     website = models.URLField()
15     def __str__(self):
16         return self.name
17
18
19 class Book(models.Model):
20     title = models.CharField(max_length=100)
21     date = models.DateField()
22     authors = models.ManyToManyField(Author)
23     publisher = models.ForeignKey(Publisher,
24                                   on_delete=models.CASCADE)
25     def __str__(self):
26         return self.title
```



Creating a model (ii)

- For each class attribute, a “Field” type object and/or subtype is instantiated, such as: CharField, DataField, etc.
 - See documentation at:
<https://docs.djangoproject.com/en/?.?/ref/models/#field-types>
- Some attributes represent the creation of relationships between classes, having the effect of creating columns with foreign keys (1:1, 1:M, M:1) or association tables (M:N)
 - See documentation at:
<https://docs.djangoproject.com/en/?.?/topics/db/models/#relationships>



Creating a model (iii)

- Relations between classes:
- 1:M and M:1
 - Achieved with an attribute of the “models.ForeignKey” class
 - Example of 1:M and M:1
 - Publisher (1) : (M) Book or Book (M) : (1) Publisher
 - The attribute is placed in the Book (M) class, one that represents many objects for one.
- 1:1 (single)
 - Achieved with an attribute of the “models.OneToOne” class
 - Example: Book (1) : (1) Place
 - The attribute “must” be placed on the class that “needs” the most another – which one??



Creating a model (iv)

- M:N
 - Achieved with an attribute of the “models.ManyToManyField” class
- Example
 - Book (M) : (N) Author
 - The attribute “must” be placed in the class that “needs” the other the most, as in the 1:1 relationship
 - Which one??

Creating a model (v)



- The “Model” base class, from which all model classes are derived, has all the necessary mechanisms to interact with the database.
- Each derived class is implemented in the DB in the form of a table and its attributes are implemented in the form of columns (fields) of the table.
- Example of the “Author” class, which corresponds to:

```
CREATE TABLE “app_author” (  
    “id” integer NOT NULL PRIMARY KEY AUTOINCREMENT,  
    “name” varchar (70) NOT NULL,  
    “email” varchar (254) NOT NULL );
```

Creating a model (vi)



- In order to activate the model, the web application, “app”, must be included in the `INSTALLED_APPS` variable of the “settings.py” file. If the application is already being included, indirectly, it is not necessary.
- Next, the model must be validated (syntax and logic) and to do so, execute the following command in the console:
 - `python manage.py check`
- Inside the “app/migrations” folder, if any, delete them. all files, with the exception of “__init__.py” and execute the following commands in the console:
 - `python manage.py makemigrations app` (produces migration code)
 - `python manage.py sqlmigrate app 0001` (optional: shows SQL code)
 - `python manage.py migrate` (produces the tables in the DB)

Data Management (i)

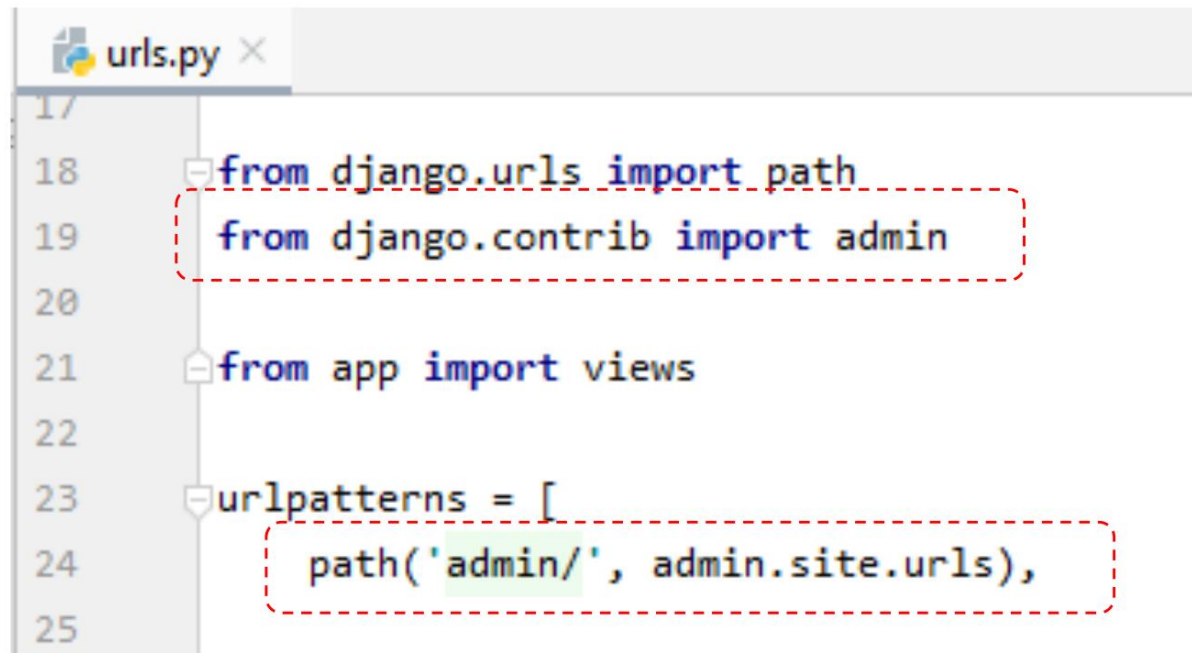


- The Django platform has a mechanism which enables very easy management of all data belonging to the data model: Django Admin Site
- The URL = <http://localhost:{port}/admin> gives access to the administrative area which allows, by default, managing the site's users
- In this area, it is also possible to access and manage the data defined in the model



Data Management (ii)

- Settings
 - In the “urls.py” file, add the following:

A screenshot of a code editor window titled 'urls.py'. The code is as follows:

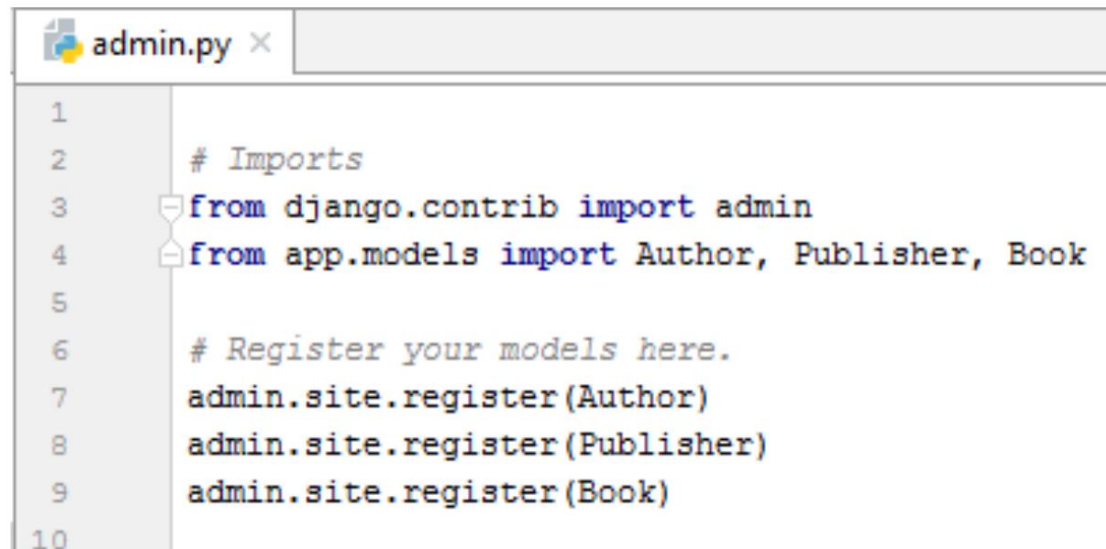
```
17  
18 from django.urls import path  
19 from django.contrib import admin  
20  
21 from app import views  
22  
23 urlpatterns = [  
24     path('admin/', admin.site.urls),  
25 ]
```

Red dashed boxes highlight the import statements on lines 18-19 and the path configuration on line 24. The string 'admin/' in the path is highlighted in green.



Data Management (iii)

- Add the data model to the Admin Site
 - In the “admin.py” file in the “app” folder, register the classes that you want to manage

A screenshot of a code editor window titled 'admin.py'. The code is as follows:

```
1
2     # Imports
3     from django.contrib import admin
4     from app.models import Author, Publisher, Book
5
6     # Register your models here.
7     admin.site.register(Author)
8     admin.site.register(Publisher)
9     admin.site.register(Book)
10
```

- From the administration page, the data model becomes accessible

Data Management (iv)



- Account to access the Django Admin Site:
 - Create the administration account, with the command:
 - `python manage.py createsuperuser`
- Test the Django Admin Site
 - Run the project and access the link: • <http://localhost:{porto}/admin>



Data Management (v)

- Programming:

- Insert an object

```
a = Author(name='Antonio Pedro',  
            email='apedro@email.com')  
a.save()
```

- Modify an object

```
a.email = 'antonio.pedro@email.com' a.save()
```

- Select all objects

```
Author.objects.all()
```

- Filter objects (by name)

```
Author.objects.filter(name='Author1')
```



Data Management (vi)

- Filter by name and email

```
Author.objects.filter(name='Author1', email='...')
```

- Filter by similar name

```
Author.objects.filter(name_contains='Author')
```

- Access a single object

```
Author.objects.get(email='autor1 @email.com')
```

- Ordering

```
Publisher.objects.order_by("city", "country")
```

- Filtering and Sorting

```
Publisher.objects.filter(country='Portugal').order_by("-city")
```




Data Management (vii)

- Select the first results

```
Publisher.objects.order_by("city", "country")[0]
```

```
Publisher.objects.order_by("city", "country")[0:4]
```

- Negative indices are not allowed

- Remove an object

```
Author.objects.get(email='autor1@email.com').delete()
```

- Many alternate forms are possible. See documentation at:

- <http://docs.djangoproject.com/en/?..?/topics/db/queries>