

# Lab: Migrations and Django Admin

This document defines the problems for the in-class lab for the **Python ORM course** @ SoftUni Global.

For this lab, you are given an **ORM project skeleton** (you can download it from the current lesson's resources) with **one model** called **"Product"**. That model consists of the fields: **"name"**, **"description"**, **"price"**, **"category"**, and **"supplier"**.

## 1. Migrations

**Note: The problem cannot be submitted in the Judge system.**

Your task is to help us **improve the model** by making some **changes to the code**:

- First, we want you to **migrate the model** to the database.
- Next, we need to **add a couple of records** to the created database table. To help us, you need to **open the caller.py** file and **execute the code on line 80**.
- We just remembered that we need to **put two more columns** in the table:
  - **created\_on** - a **date and time field**, that should be **set to the current date and time when the object is first created** and should **not be editable**.
  - **last\_edited\_on** - a **date and time field**, that should be **set to the current date and time every time the object is saved** and should **not be editable**.

**Add the fields to the model** and **migrate the changes**.

- We are willing to **add one more record**. **Run only the code on line 82** in the **caller.py** file.
- Now, we come up with the idea to **add the following field**:
  - **count** - a **positive integer field** with a **default value of 0** (zero).

We need you to **refactor the code** and **migrate the changes** to the database.

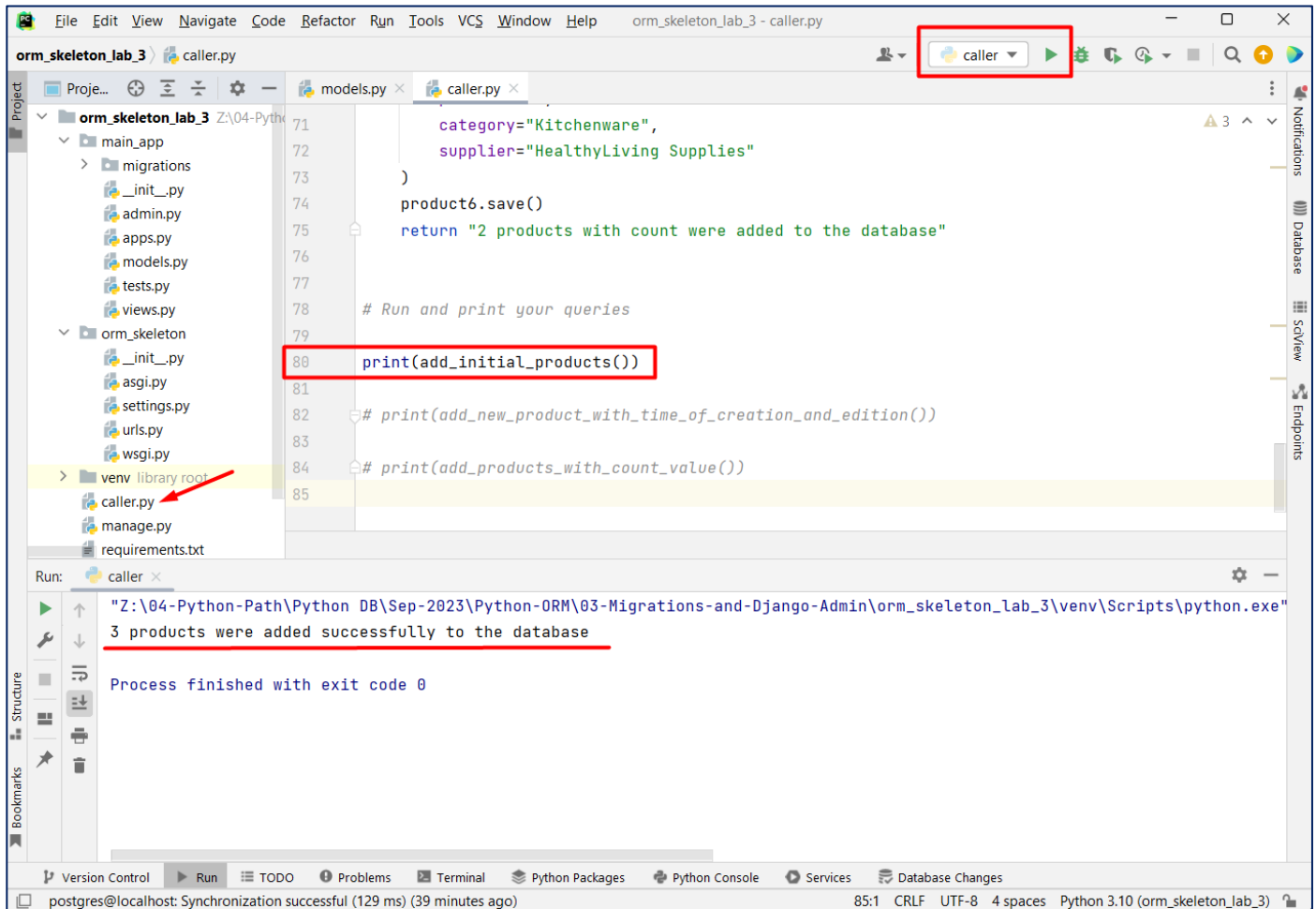
- We want to **add a couple of records**. **Run only the code on line 84** in the **caller.py** file.
- We saw that it was a poor idea and now we want you to **undo the changes from the previous task** (reverse the **last migration** and remove the **count** field).
- Now, we have come to the opinion that making the fields **"category"** and **"supplier"** **required** (rather than optional) is beneficial. **Make the changes to the code** and **migrate them** to the database.
- For now, we think we are satisfied with the model.

## Hint

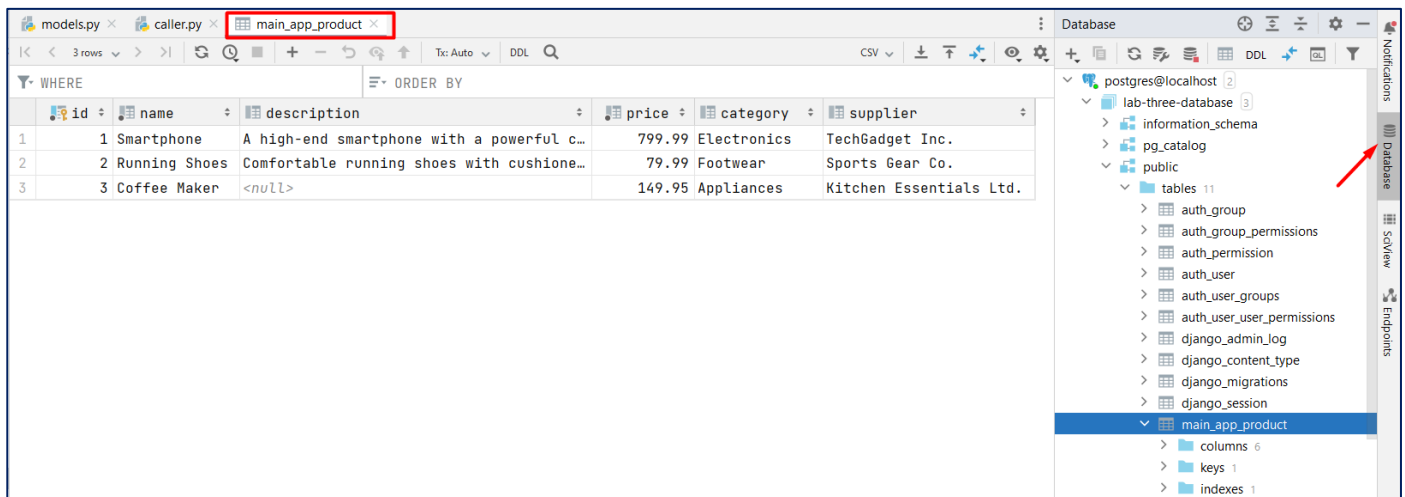
**Note: Before start solving the problem, set up the database.**

First, we need to migrate the model to the database. We will use the terminal commands **"python manage.py makemigrations"** and **"python manage.py migrate"** successively.

Next, we will go to the **caller.py** file, and execute the code on **line 80**. If the products are added successfully to the database as record, we should see the following output on the console:

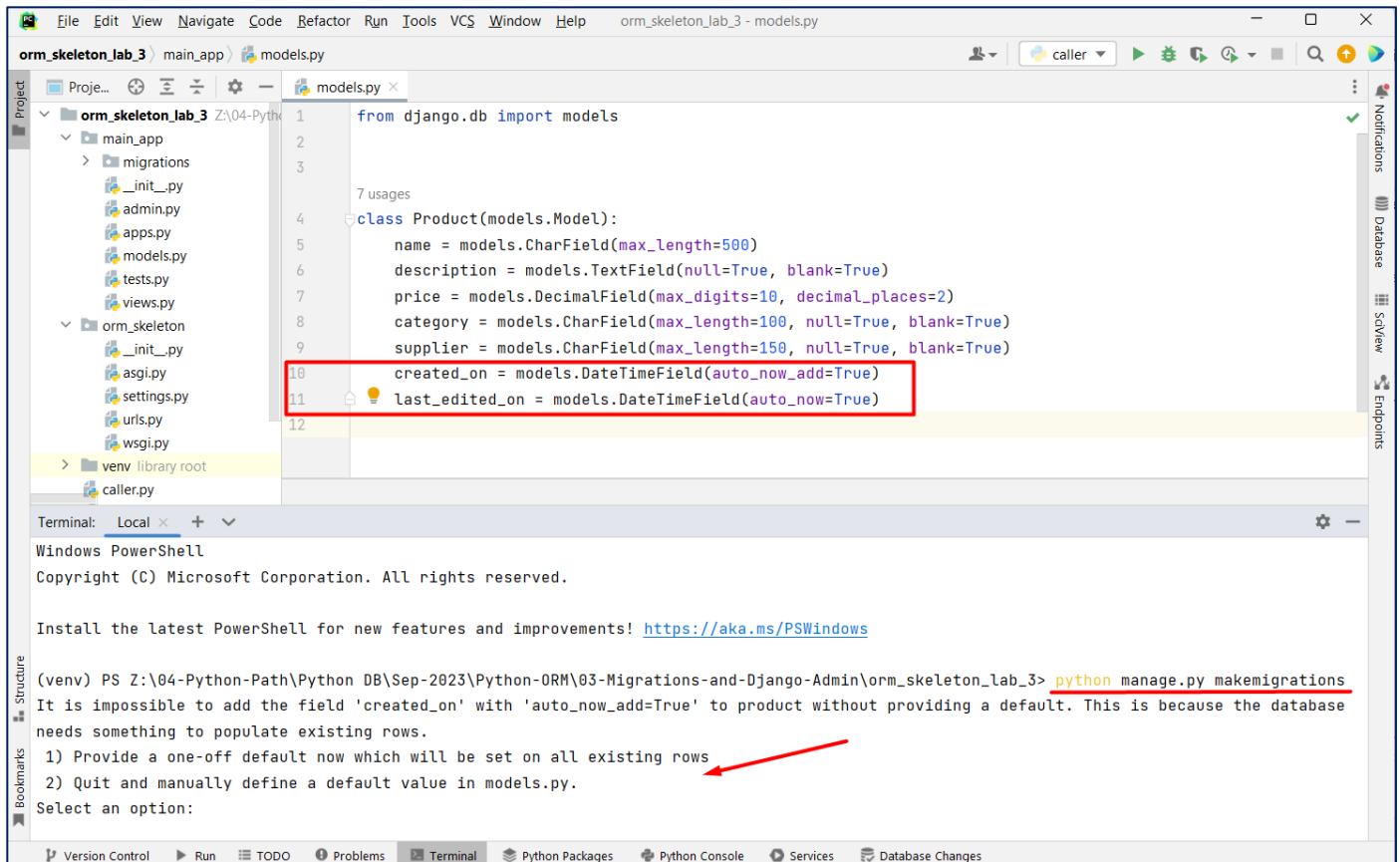


We can also **check the table data**, using the PyCharm Database tool:



Now, our next task is to add **two more fields** to the model and **migrate the changes**. We can see that, when we try to make the migration files, Django raises an error. That is because the database needs something to populate existing rows. We have **two options** - we can **provide a one-off value** that will be set only on the already existing

rows, or we can **exit the operation and set a default value** in the model fields:



The screenshot shows an IDE with a project named 'orm\_skeleton\_lab\_3'. The file 'models.py' is open, showing a Django model class 'Product' with fields: 'name', 'description', 'price', 'category', 'supplier', 'created\_on', and 'last\_edited\_on'. The 'created\_on' and 'last\_edited\_on' fields are highlighted with a red box. Below the code editor, a terminal window shows the command 'python manage.py makemigrations' and an error message: 'It is impossible to add the field 'created\_on' with 'auto\_now\_add=True' to product without providing a default. This is because the database needs something to populate existing rows.' The terminal also shows two options: '1) Provide a one-off default now which will be set on all existing rows' and '2) Quit and manually define a default value in models.py.' A red arrow points to option 1.

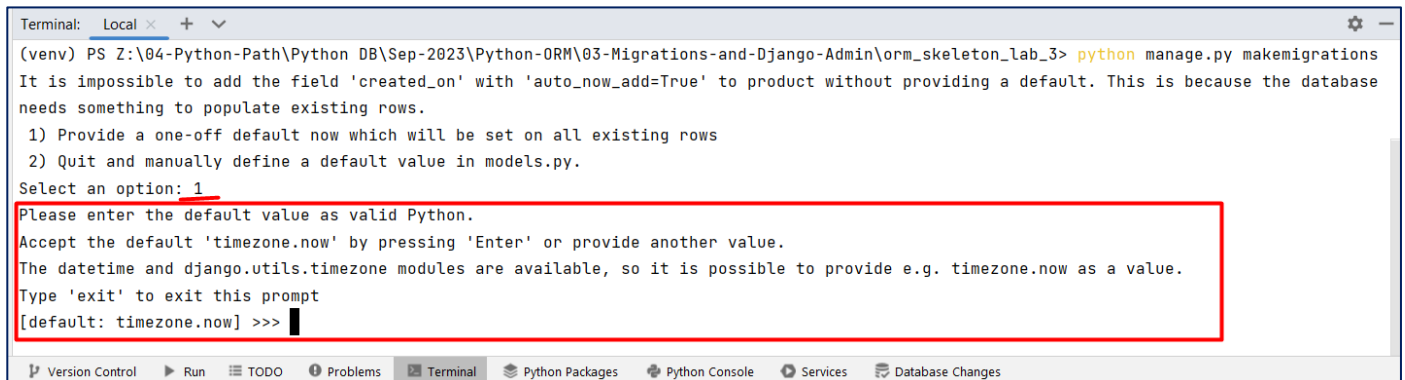
```
1 from django.db import models
2
3
4 class Product(models.Model):
5     name = models.CharField(max_length=500)
6     description = models.TextField(null=True, blank=True)
7     price = models.DecimalField(max_digits=10, decimal_places=2)
8     category = models.CharField(max_length=100, null=True, blank=True)
9     supplier = models.CharField(max_length=150, null=True, blank=True)
10    created_on = models.DateTimeField(auto_now_add=True)
11    last_edited_on = models.DateTimeField(auto_now=True)
12
```

```
Terminal: Local x + v
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

(venv) PS Z:\04-Python-Path\Python DB\Sep-2023\Python-ORM\03-Migrations-and-Django-Admin\orm_skeleton_lab_3> python manage.py makemigrations
It is impossible to add the field 'created_on' with 'auto_now_add=True' to product without providing a default. This is because the database
needs something to populate existing rows.
1) Provide a one-off default now which will be set on all existing rows
2) Quit and manually define a default value in models.py.
Select an option:
```

Let us **choose option 1** for this example. We can enter a **default value** manually or we can choose to use the **now()** function provided by Django. Let us just **set the default value to now()** by pressing Enter:



The screenshot shows a terminal window with the same command and error message as the previous screenshot. The user has selected option 1, and the terminal prompts for a default value. The user enters 'now()', and the terminal shows the prompt '[default: timezone.now] >>>'.

```
Terminal: Local x + v
(venv) PS Z:\04-Python-Path\Python DB\Sep-2023\Python-ORM\03-Migrations-and-Django-Admin\orm_skeleton_lab_3> python manage.py makemigrations
It is impossible to add the field 'created_on' with 'auto_now_add=True' to product without providing a default. This is because the database
needs something to populate existing rows.
1) Provide a one-off default now which will be set on all existing rows
2) Quit and manually define a default value in models.py.
Select an option: 1
Please enter the default value as valid Python.
Accept the default 'timezone.now' by pressing 'Enter' or provide another value.
The datetime and django.utils.timezone modules are available, so it is possible to provide e.g. timezone.now as a value.
Type 'exit' to exit this prompt
[default: timezone.now] >>>
```

If we look at the migration file, we can see the default value added:

```
models.py x 0002_product_created_on_product_last_edited_on.py x
6
7 class Migration(migrations.Migration):
8
9     dependencies = [
10         ('main_app', '0001_initial'),
11     ]
12
13     operations = [
14         migrations.AddField(
15             model_name='product',
16             name='created_on',
17             field=models.DateTimeField(auto_now_add=True, default=django.utils.timezone.now),
18             preserve_default=False,
19         ),
20         migrations.AddField(
```

We can now easily migrate the changes to the database using the "**python manage.py migrate**" command.

Let us **add one more record** to the database using the code in the **caller.py**:

```
Project models.py x caller.py x
75 return "2 products with count were added to the database"
76
77
78 # Run and print your queries
79
80 # print(add_initial_products())
81
82 print(add_new_product_with_time_of_creation_and_edition())
83
84 # print(add_products_with_count_value())
85 |

Run: caller x
"Z:\04-Python-Path\Python DB\Sep-2023\Python-ORM\03-Migrations-and-Django-Admin\orm_sk
1 product with time of creation and edition was added to the database
Process finished with exit code 0
```

It is time to **execute our next task**. We need to **add a new field called count**:

```
models.py x
1 from django.db import models
2
3
4 7 usages
5 class Product(models.Model):
6     name = models.CharField(max_length=500)
7     description = models.TextField(null=True, blank=True)
8     price = models.DecimalField(max_digits=10, decimal_places=2)
9     category = models.CharField(max_length=100, null=True, blank=True)
10    supplier = models.CharField(max_length=150, null=True, blank=True)
11    created_on = models.DateTimeField(auto_now_add=True)
12    last_edited_on = models.DateTimeField(auto_now=True)
13    count = models.PositiveIntegerField(default=0)
```

**Migrate the changes** to the database. Then, execute only the **84<sup>th</sup> line**.

Unfortunately, we are given more work to do. Now, we need to **undo our changes**. First, let us **reverse the last migration** by using the command **"python manage.py migrate main\_app 0002"**:

```
Terminal: Local x + v
(venv) PS Z:\04-Python-Path\Python DB\Sep-2023\Python-ORM\03-Migrations-and-Django-Admin\orm_skeleton_lab_3> python manage.py migrate main_app 0002
Operations to perform:
  Target specific migration: 0002_product_created_on_product_last_edited_on, from main_app
Running migrations:
  Rendering model states... DONE
  Unapplying main_app.0003_product_count... OK
(venv) PS Z:\04-Python-Path\Python DB\Sep-2023\Python-ORM\03-Migrations-and-Django-Admin\orm_skeleton_lab_3> |
```

Next, we should **delete the field "count"**, so that it will not be added to the table as a column anymore. And, finally, we **do not need the migration file 0003**, so we can delete it. If we do not delete it, the next time we migrate the changes, Django will execute the migration file again.

Finally, **make the last changes** to the model and **migrate them**.

## 2. Barcode System

**Note:** The problem cannot be submitted in the Judge system.

We've reconsidered it, and now we want to **make some more changes to the "Product" model**. Add a new **unique integer field** called **"barcode"** to the model. It will consist of the barcode of each product. For **all already applied products in the database**, we should add the barcode value - a random unique number from **100 000 000 to 999 999 999 both inclusive**.

### Hint

First, let us add the field **"barcode"** to the model:

```
models.py x
1  from django.db import models
2
3
4  7 usages
5  class Product(models.Model):
6      name = models.CharField(max_length=500)
7      description = models.TextField(null=True, blank=True)
8      price = models.DecimalField(max_digits=10, decimal_places=2)
9      category = models.CharField(max_length=100)
10     supplier = models.CharField(max_length=150)
11     created_on = models.DateTimeField(auto_now_add=True)
12     last_edited_on = models.DateTimeField(auto_now=True)
13     barcode = models.IntegerField()
```

Next, let us **migrate the changes** to the database, and then on a separate migration file we can write our custom logic. First, we need to **open the terminal** and write the command "**python manage.py makemigrations --empty main\_app**" to generate a new empty migration file. Then, we can **populate the file**. Note, that we can use **random.sample(range, count)** Python function to get only random unique values:

```
0005_auto_20230910_1737.py x
1  import random
2  from django.db import migrations
3
4
5  1 usage
6  def add_barcode(apps, schema_editor):
7      Product = apps.get_model("main_app", "Product")
8      all_products = Product.objects.all()
9      all_barcodes = random.sample(
10         range(100000000, 1000000000),
11         len(all_products))
12
13     for i in range(len(all_products)):
14         product = all_products[i]
15         product.barcode = all_barcodes[i]
16         product.save()
17
18  1 usage
19  def reverse_add_barcode(apps, schema_editor):
20      Product = apps.get_model("main_app", "Product")
21      for product in Product.objects.all():
22         product.barcode = 0
23         product.save()
24
25  class Migration(migrations.Migration):
26      dependencies = [('main_app', '0004_product_barcode')]
27      operations = [migrations.RunPython(add_barcode, reverse_code=reverse_add_barcode)]
```

### 3. Register the Model

Register the model "Product" in the Django Admin site. **Open the interface** and **create one record** for each model. Familiarize yourself with the functionalities of the admin site. **Submit your project to the Judge system.**

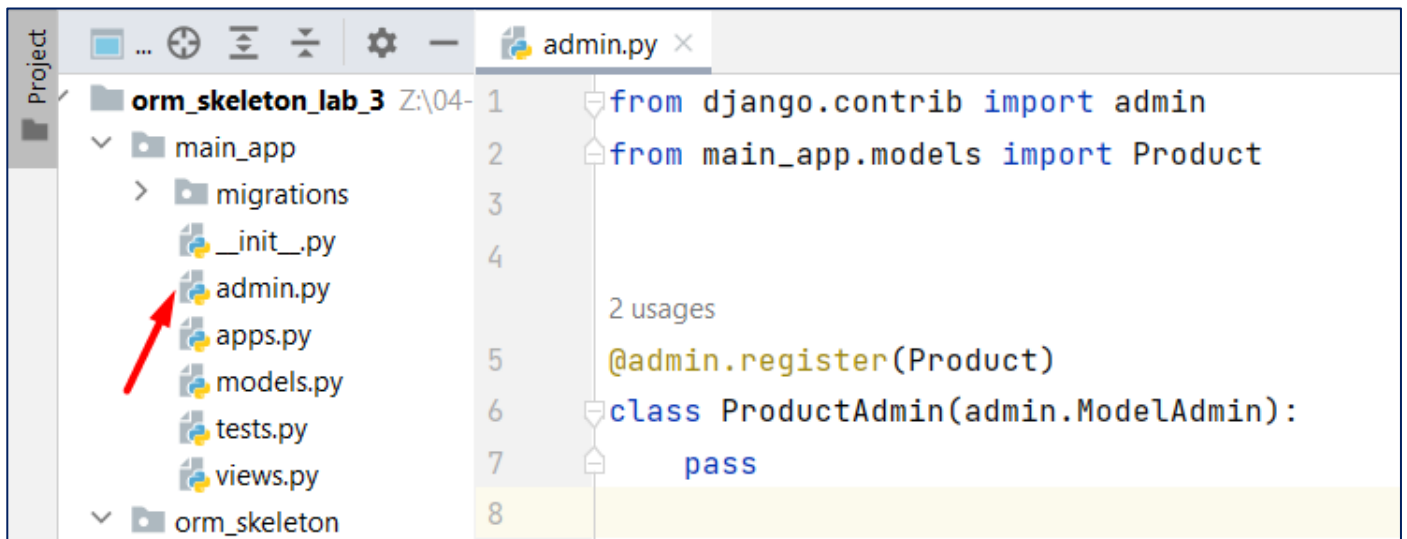
#### Hint

First, let us open the **admin.py** file in the **main\_app** and import the model "Product". Now, we have two ways of registering the model in the admin site. One way to register the model is by directly providing the model. We use this when we **do not want to define any custom values** and **we are ok with the default admin interface**:



```
1 from django.contrib import admin
2 from main_app.models import Product
3
4
5 admin.site.register(Product)
6
```

Second way - we can **create a class that inherits from the ModelAdmin class** to register our model:



```
1 from django.contrib import admin
2 from main_app.models import Product
3
4
5 @admin.register(Product)
6 class ProductAdmin(admin.ModelAdmin):
7     pass
8
```

To work with the admin site, we need to create a superuser. We achieve it by executing the terminal command **"python manage.py createsuperuser"**. Then, we need to pass the name and the password of the user.

### 4. Customize the Admin

**Customize the interface** of the registered model. In a **ProductAdmin** class add the following functionality:

- **Display** the fields "name", "category", "price", and "created\_on" (in that order) in the admin panel.
- Enable **searching** for "name", "category", and "supplier" fields in the admin panel.
- Create **filters** for "category" and "supplier" fields in the admin panel.
- **Control the layout** of "Add" and "Change" pages by grouping related fields within different sections:
  - Group **"General Information"** with fields "name", "description", "price", "barcode".



- Group **"Categorization"** with fields **"category"** and **"supplier"**.
- Enable date-based drill-down navigation by the **"created\_on"** field.

## Hint

To include a date-based drill-down navigation by the field **created\_on**, we should use the option **date\_hierarchy** and set it to **"created\_on"** value:

```
admin.py x
1  from django.contrib import admin
2  from main_app.models import Product
3
4
5  2 usages
6  @admin.register(Product)
7  class ProductAdmin(admin.ModelAdmin):
8      list_display = ('name', 'category', 'price', 'created_on')
9      search_fields = ('name', 'category', 'supplier')
10     list_filter = ('category', 'supplier')
11     fieldsets = (
12         ('General Information', {
13             'fields': ('name', 'description', 'price', 'barcode')
14         }),
15         ('Categorization', {
16             'fields': ('category', 'supplier')
17         }),
18     )
19     date_hierarchy = 'created_on'
```