Django Migrations and Django Admin

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

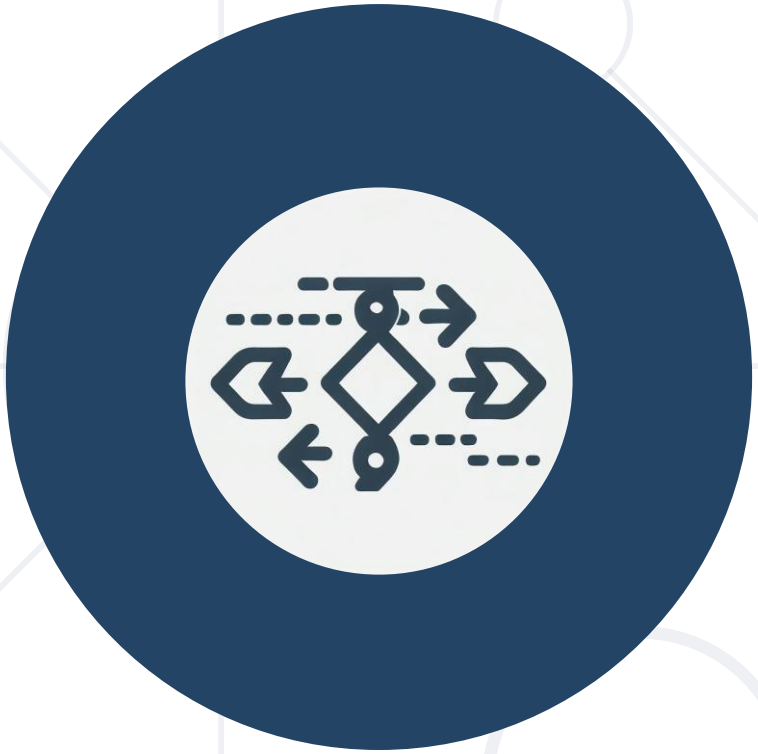
Have a Question?

Software University

sli.do

#python-db

Table of Contents



Django Migrations Advanced

Django Migrations Basics Overview

- Files with **Python code** that:
  - Propagate **changes** you make to your **models** into the **database** schema
  - Designed to be mostly automatic
- Basic Commands
  - `makemigrations`
    - Packing all **changes** into **migration files**
  - `migrate`
    - Applying **migrations** to the **database**
    - **Unapplying** migrations

Migration Files

- Python files, written in a **declarative** style

```
from django.db import migrations, models

class Migration(migrations.Migration):
    initial = True
    dependencies = []
    operations = [migrations.CreateModel(
        name='Employee',
        fields=[('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
                ('first_name', models.CharField(max_length=30)),
                ...])]
```

- It is possible to write them **manually** if needed

Applying Migrations

- To apply all migrations from all apps

```
python manage.py migrate
```

- To apply all migrations from one app

```
python manage.py migrate main_app
```

- To apply specific migration

```
python manage.py migrate main_app 0001
```

Reversing Migrations

- To revert to **a certain migration**, pass the **app name** and the **number** of the **migration** you need to revert to

```
python manage.py migrate main_app 0001
```

- To reverse **all** already applied **migrations**, use the **app name** and the name **zero** as parameters

```
python manage.py migrate main_app zero
```

- **Note:** If a migration contains any **irreversible** operations, attempting to reverse it will raise `IrreversibleError`

Problem: Migrations

- You are given an **ORM project skeleton** (you can download it from **here**) with one model called **"Product"**
- Your task is to help us **improve the model** by making some changes to the code
- A full description of the problem can be found in the Lab document **here**

Showing Migrations

- Listing **project's** migrations and their status

```
python manage.py showmigrations
```

  - Apps without migrations are also listed but have no migrations printed after them

- Listing migrations and their status for a **certain app**

```
python manage.py showmigrations main_app
```

- You can choose a format to list: `--list` or `-l`

```
python manage.py showmigrations --list
```

Optimizing Migrations Number and Size

- **squashmigrations** command
  - Reducing an existing set of (many) migrations
    - down to one or sometimes a few migrations
    - still representing the same changes

```
python manage.py squashmigrations main_app 0238
```

  - You need to pass the **app name** and the **migration number/name**
    - all previous migrations will be **squashed**

SQL Representation of a Migration

- **sqlmigrate** command
  - Prints the **SQL** for the named migration
    - requires an **active** database connection
    - must be generated against a copy of the database on which later to be applied on

```
python manage.py sqlmigrate main_app 0001_initial
```

  - You need to pass the **app name** and the **migration number/name**

Custom / Data Migrations

Custom / Data Migrations

- **Data Migrations**
  - Migrations that **alter data**
  - Best written as **separate migrations**
  - Sitting **alongside** your **schema migrations**
- Use **data** migrations to **change**
  - the **data** in the database **itself**
  - in **conjunction** with the schema if you need that

Custom / Data Migrations

- Django **cannot** automatically generate **Data Migrations**
  - It is not very hard to write them **manually**
- Migration files in Django are made up of **Operations**
  - The main **operation** to use for **data migrations** is **RunPython**

Creating an Empty Migration

- By making an **empty** migration file Django will
  - put the file in the right place
  - suggest a name
  - add **dependencies**

```
python manage.py makemigrations --empty main_app
```

- You need to pass the **app name**

Empty Migration

- The **empty** migration file would look like this:

```
from django.db import migrations


class Migration(migrations.Migration):
    dependencies = [
        ("main_app", "0002_employee_full_name"),
    ]

    operations = []
```

app name

last migration to depend on

empty list of operations

RunPython Usage

- Create a function and have `RunPython` use it
- `RunPython` expects a **callable** which takes **two** arguments
  - `apps`
    - A registry of installed applications that
      - stores configuration
      - provides introspection
      - maintains a list of available models
    - Has the historical versions of all models
  - `SchemaEditor`
    - Exposes operations as methods and turns code into SQL

RunPython Usage Example

- Problem: For all already **existing records** in the database, the new column should be **prepopulated** with data

models.py

```
from django.db import models


class Employee(models.Model):
    ...
    # additional field with default value
    full_name = models.CharField(max_length=200, default='')
```

RunPython Usage Example

Software University

0003_data_migration.py

```
from django.db import migrations

def add_full_name(apps, schema_editor):
    Employee = apps.get_model("main_app", "Employee")
    for employee in Employee.objects.all():
        employee.full_name = f"{employee.first_name} {employee.last_name}"
        employee.save()

class Migration(migrations.Migration):
    dependencies = [
        ("main_app", "0002_employee_full_name"),
    ]
    operations = [
        migrations.RunPython(add_full_name),
    ]
```

a callable that accepts apps and schema_editor

populating full_name with values for existing records

RunPython accepts the callable

Reversible Data Migration - Example

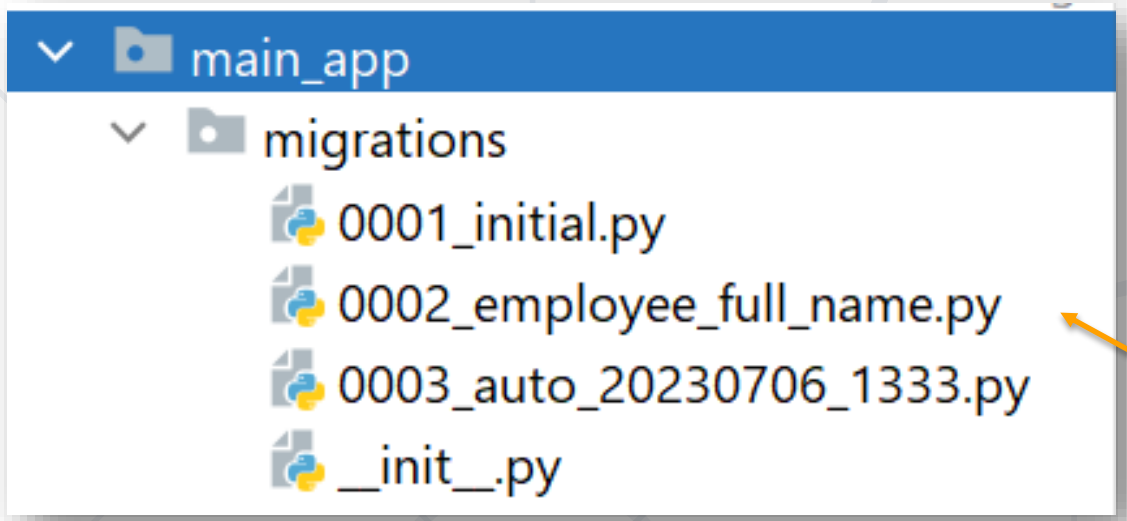
```
from django.db import migrations

def add_full_name(apps, schema_editor):
    ...

def reverse_add_full_name(apps, schema_editor):
    Employee = apps.get_model("main_app", "Employee")
    for employee in Employee.objects.all():
        employee.full_name = ''
        employee.save()

class Migration(migrations.Migration):
    dependencies = [
        ("main_app", "0002_employee_full_name"),
    ]
    operations = [
        migrations.RunPython(add_full_name, reverse_code=reverse_add_full_name),
    ]
```

Making our data migration (0003_...) reversible

Add a callable to define the reversal

RunPython accepts both callables

Data Migrations Change Data in DB




```python
class Employee(models.Model):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    job_title = models.CharField(max_length=100)
    job_level = models.CharField(max_length=50)
    email_address = models.EmailField(max_length=250)

    # additional field with default value
    full_name = models.CharField(max_length=200, default='')
```

Data Migration – populates existing records with specific data

Problem: Barcode System

- Make some more changes to the **`"Product"`** model
  - Add a new **unique integer field** called **`"barcode"`**
  - For all already applied products in the database, add a barcode value - a **random unique number** from 100 000 000 to 999 999 999 both inclusive

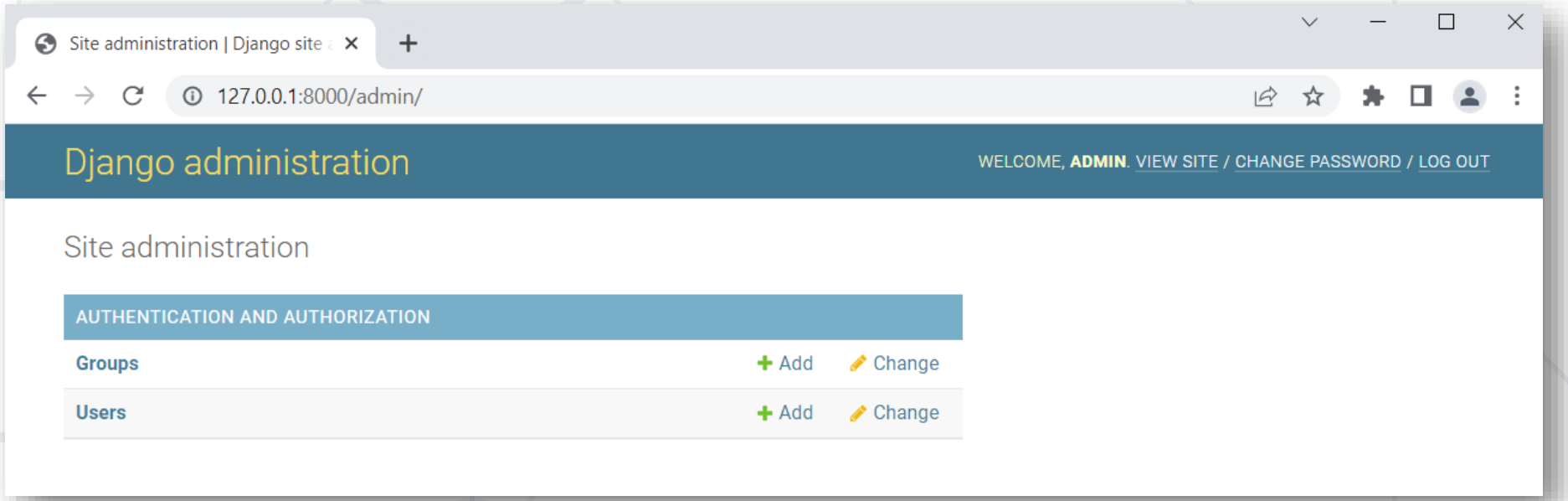
Solution: Barcode System

```
...
def add_barcode(apps, schema_editor):
    Product = apps.get_model("main_app", "Product")
    all_products = Product.objects.all()
    all_barcodes = random.sample(
        range(100000000, 1000000000),
        len(all_products)
    )
    for i in range(len(all_products)):
        product = all_products[i]
        product.barcode = all_barcodes[i]
        product.save()
...
```

Django Admin Site

Django Admin Site Introduction

- It is a **built-in admin interface**
  - Where **trusted users can manage site content**
- One of the most powerful parts of Django

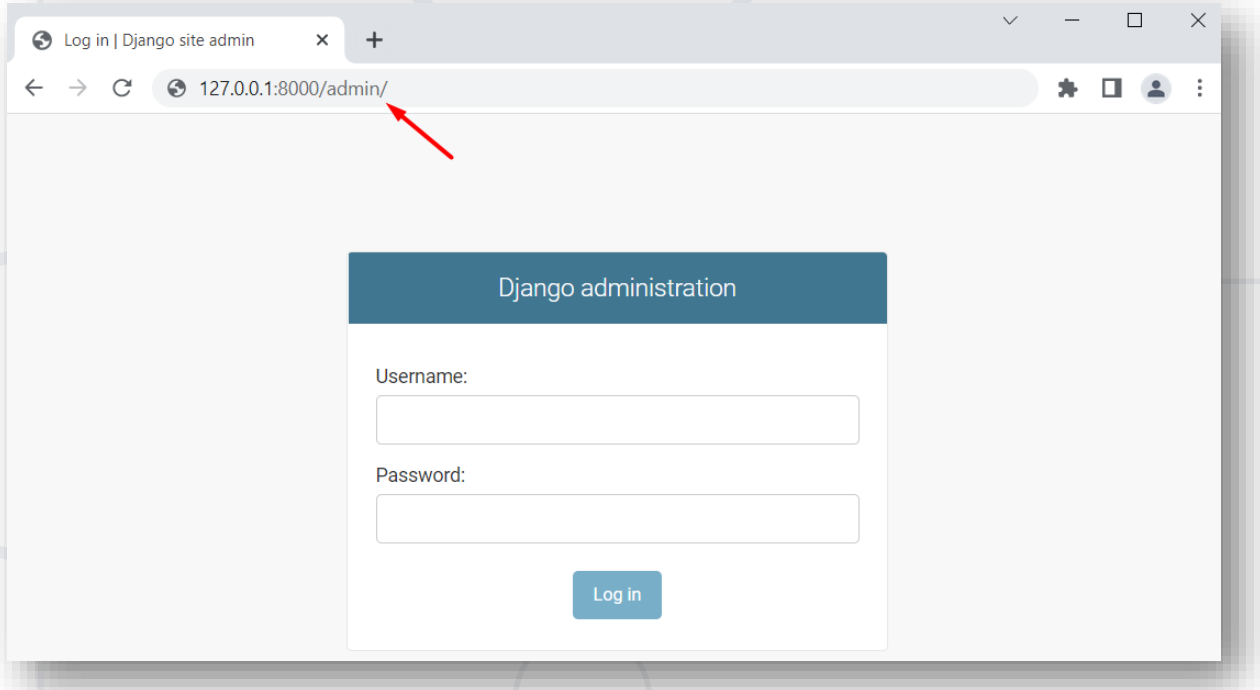


Access Django Admin Site

- First, create a **superuser to log in with**

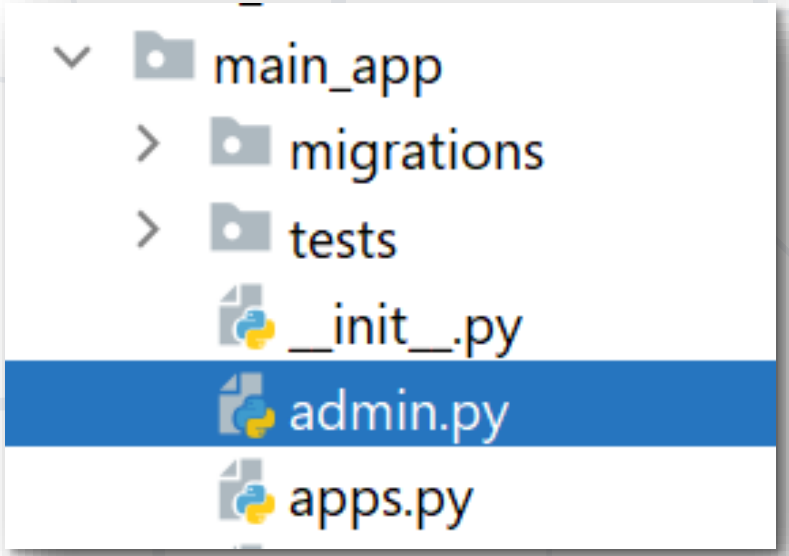
```
python manage.py createsuperuser
```

- Then, **start the server** and **navigate to the admin site**



Make the App Modifiable in the Admin

- **Register** all models in a special file in the **app** called **`admin.py`**



admin.py

```
42
43
44
45  from django.contrib import admin
46
47  from main_app.models import Employee
48
49
50  @admin.register(Employee)
51  class EmployeeAdmin(admin.ModelAdmin):
52      pass
```

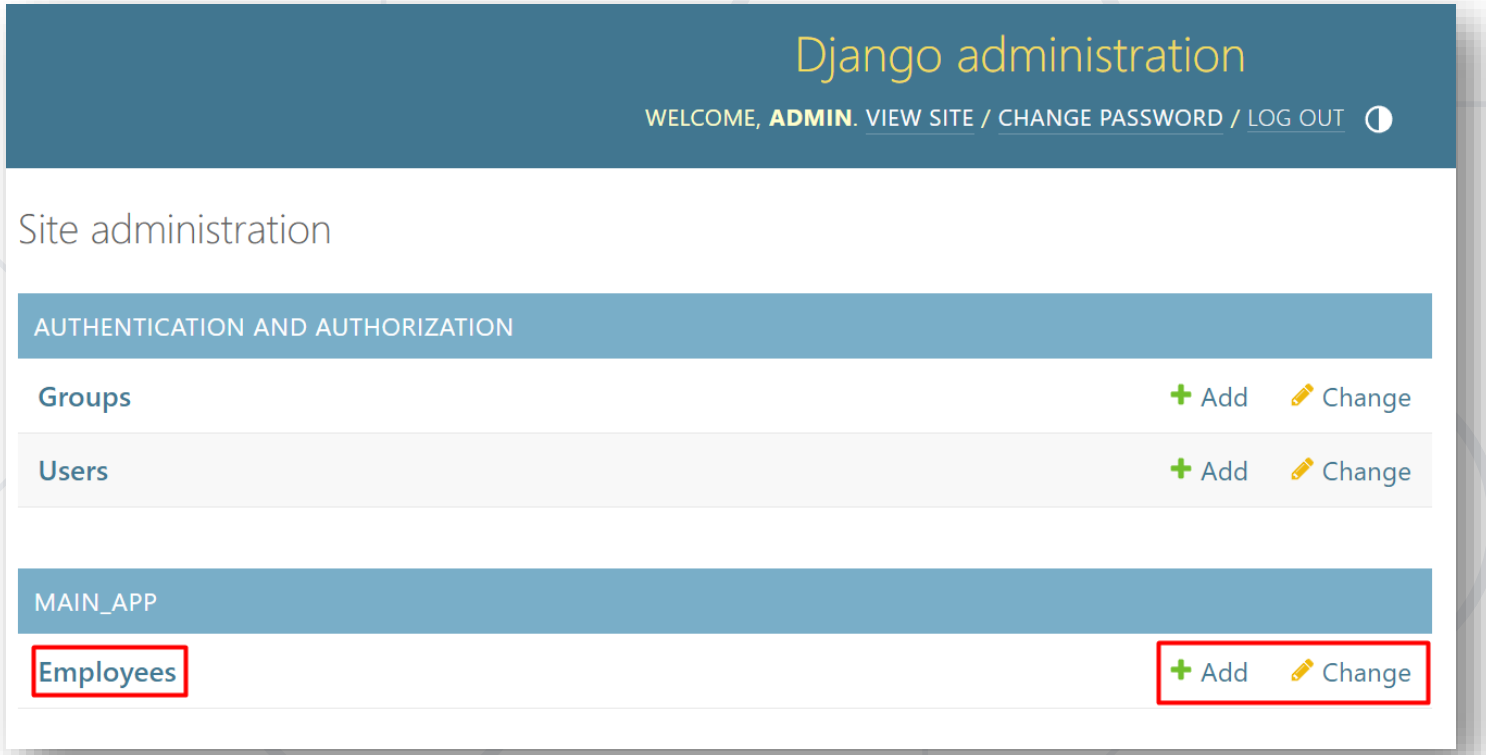
Register Models

- Use the `ModelAdmin` class
  - It represents the model in the admin site

```
from django.contrib import admin
from main_app.models import Employee

@admin.register(Employee)
class EmployeeAdmin(admin.ModelAdmin):
    pass
```

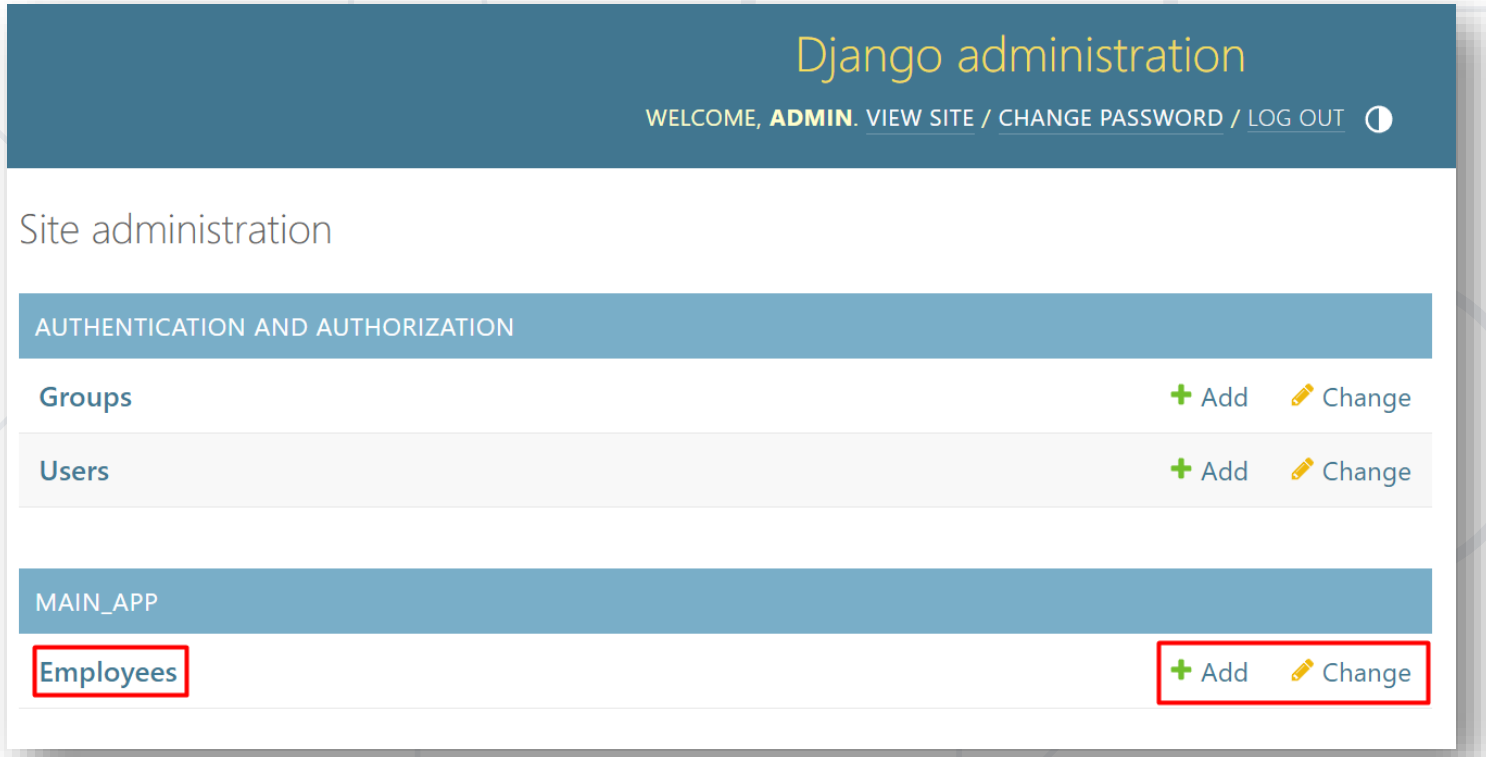
Access the Models

- Use the Django Admin site to **manage the models**



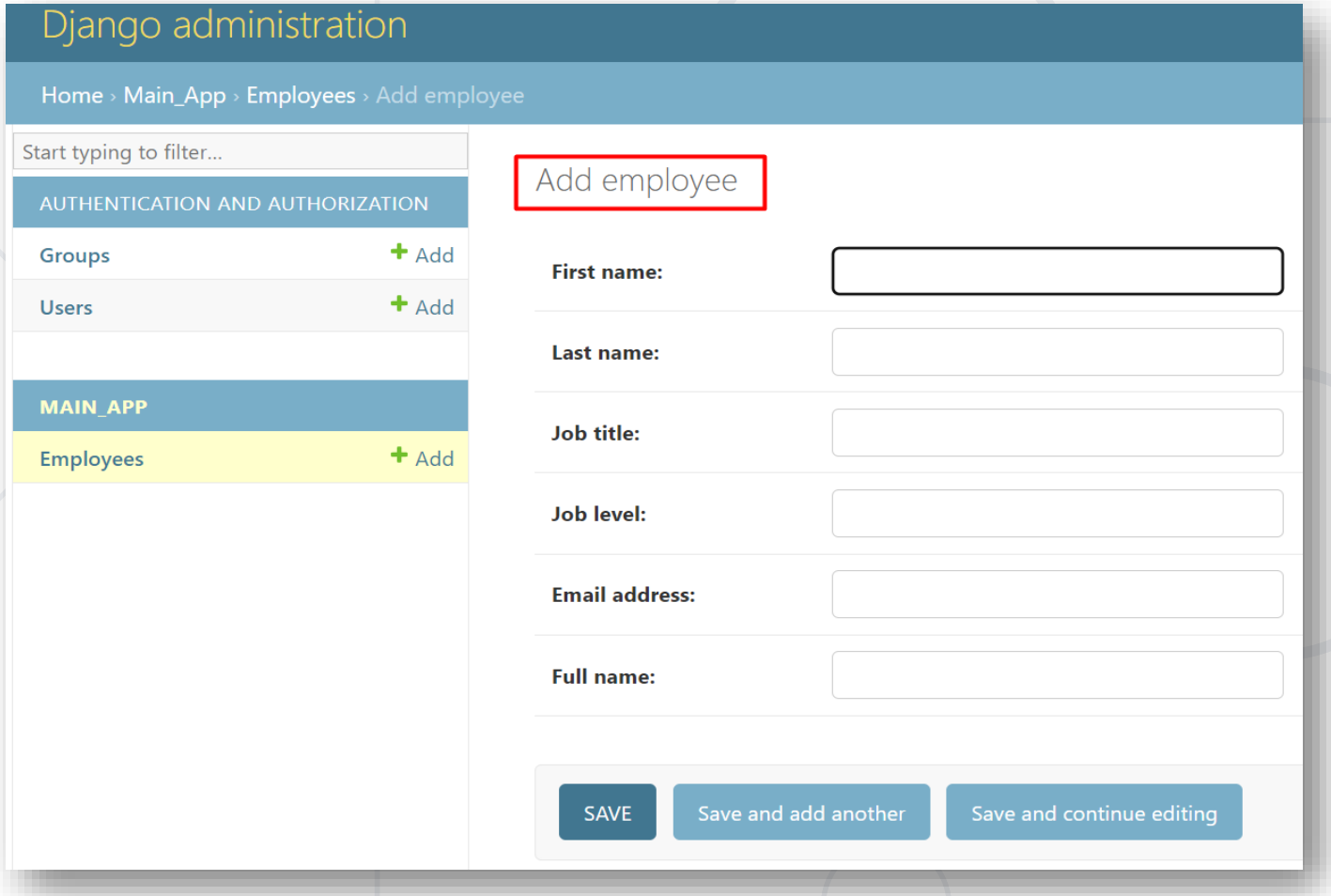
Django Admin Benefits

- Easily **manage** (create, update, delete) the data stored in the database



Django Admin Benefits

- The form is **automatically generated** from the models



Problem: 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

Solution: Register the Model

```
from django.contrib import admin
from main_app.models import Product


@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    pass
```

Django Admin Site

Customizations

Customizing Django Admin Site

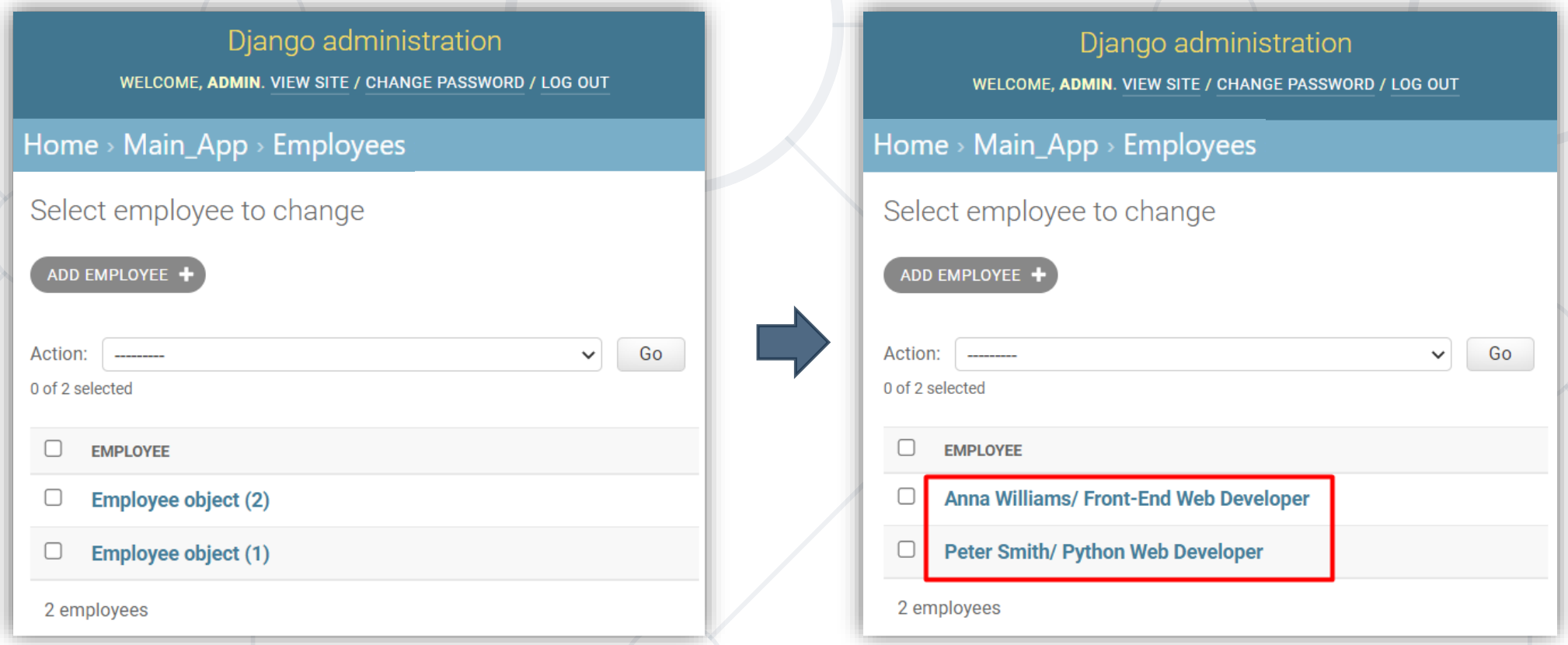
- Use the **ModelAdmin** class
  - Use its **options** to customize the admin interface

```
from django.contrib import admin
from main_app.models import Employee

@admin.register(Employee)
class EmployeeAdmin(admin.ModelAdmin):
    …
```

Add custom options here

Display the Model Objects

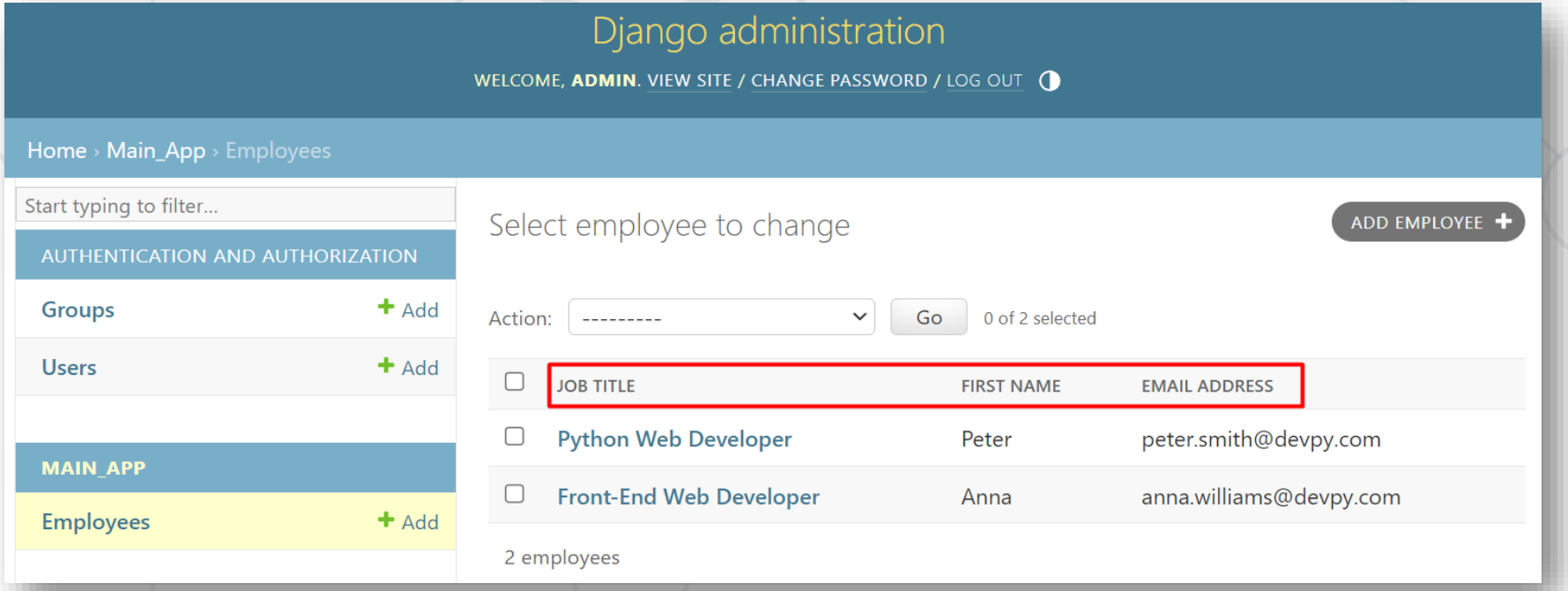
- Use **__str__()** in the Model class to return a human-readable representation of an object in the **admin site** or in the **console**



ModelAdmin Options

- **Display** the model fields

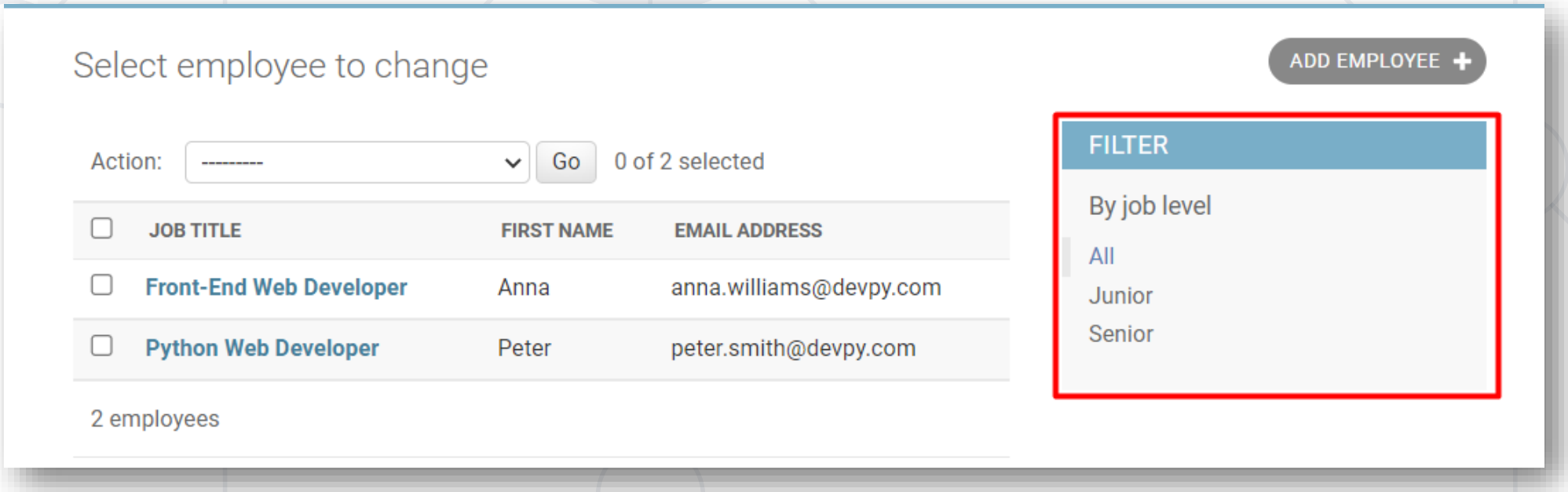
```
class EmployeeAdmin(admin.ModelAdmin):
    list_display = ['job_title', 'first_name', 'email_address']
```



ModelAdmin Options

- **Add filters** to the models

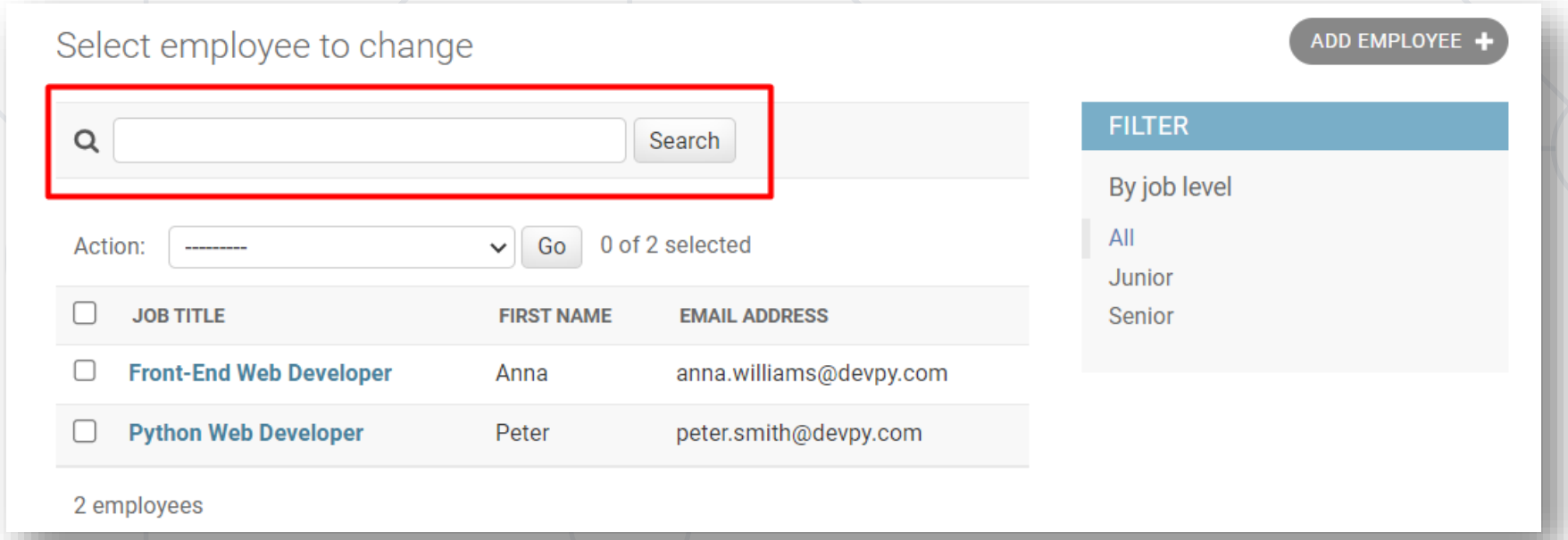
```
class EmployeeAdmin(admin.ModelAdmin):
    list_filter = ['job_level']
```



ModelAdmin Options

- Add **search box** with field names that will be searched

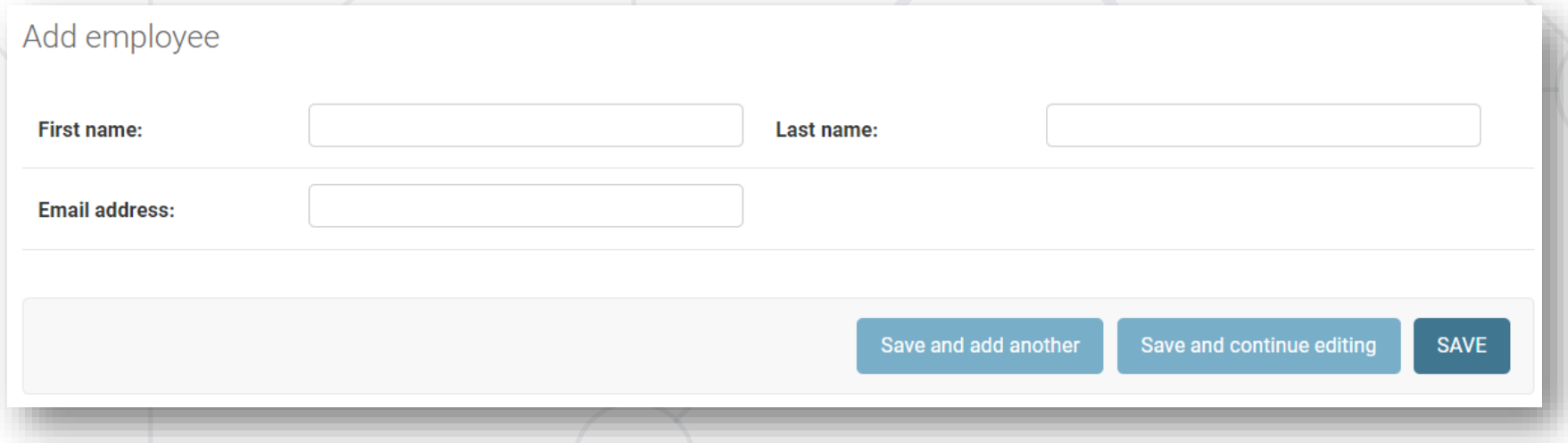
```
class EmployeeAdmin(admin.ModelAdmin):
    search_fields = ['email_address']
```



ModelAdmin Options

- Make **layout changes** on "Add" and "Change" pages

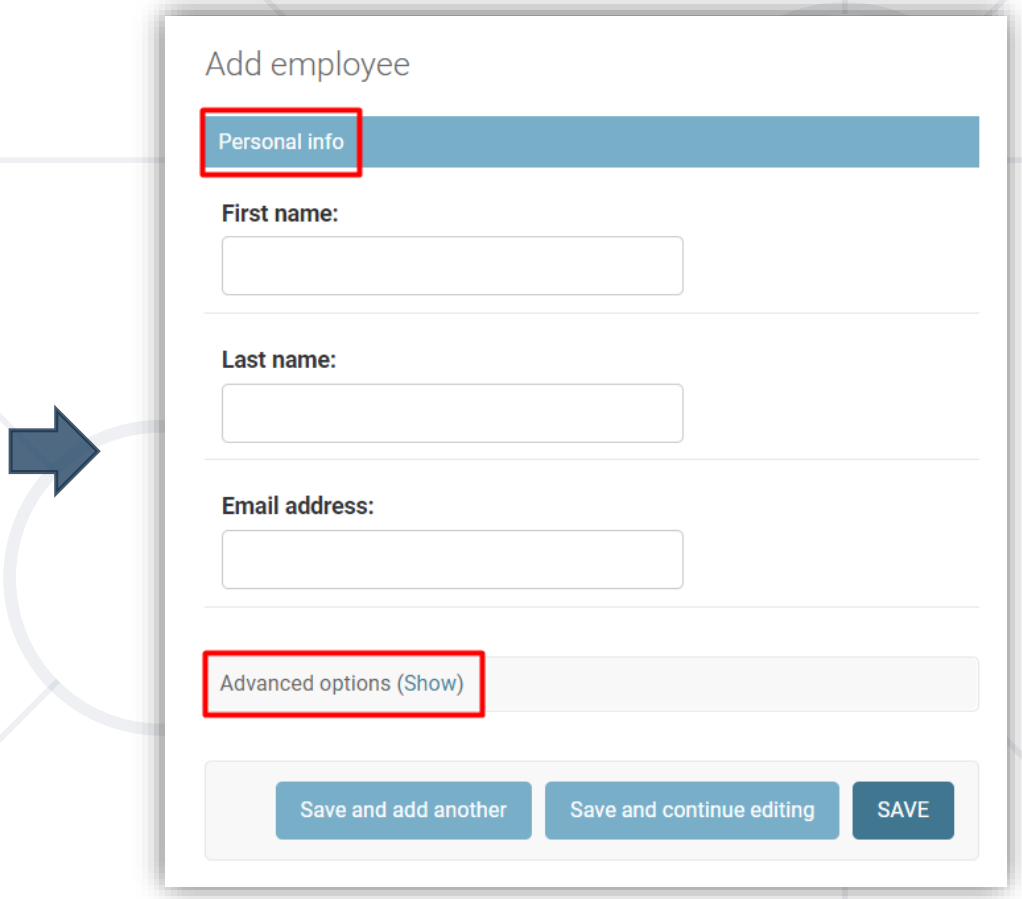
```
class EmployeeAdmin(admin.ModelAdmin):
    fields = [('first_name', 'last_name'), 'email_address']
```



ModelAdmin Options

- **Control the layout** of "Add" and "Change" pages

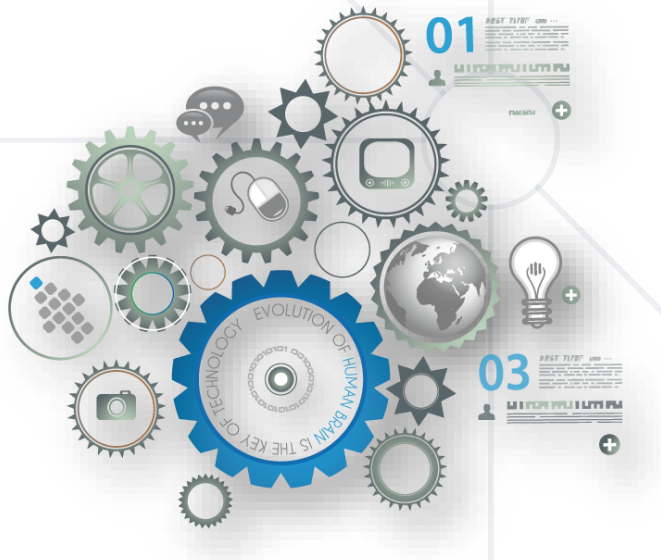
```
fieldsets = (
    ('Personal info',
     {'fields': (...)}),
    ('Advanced options',
     {'classes': ('collapse',),
      'fields': (...),}),
)
```



More ModelAdmin Options: *https://docs.djangoproject.com/en/5.0/ref/contrib/admin/#modeladmin-options*

Problem: Customize the Admin

- **Customize the interface** of the registered **"Product"** model in the Admin interface
- A full description of the problem can be found in the Lab document **here**



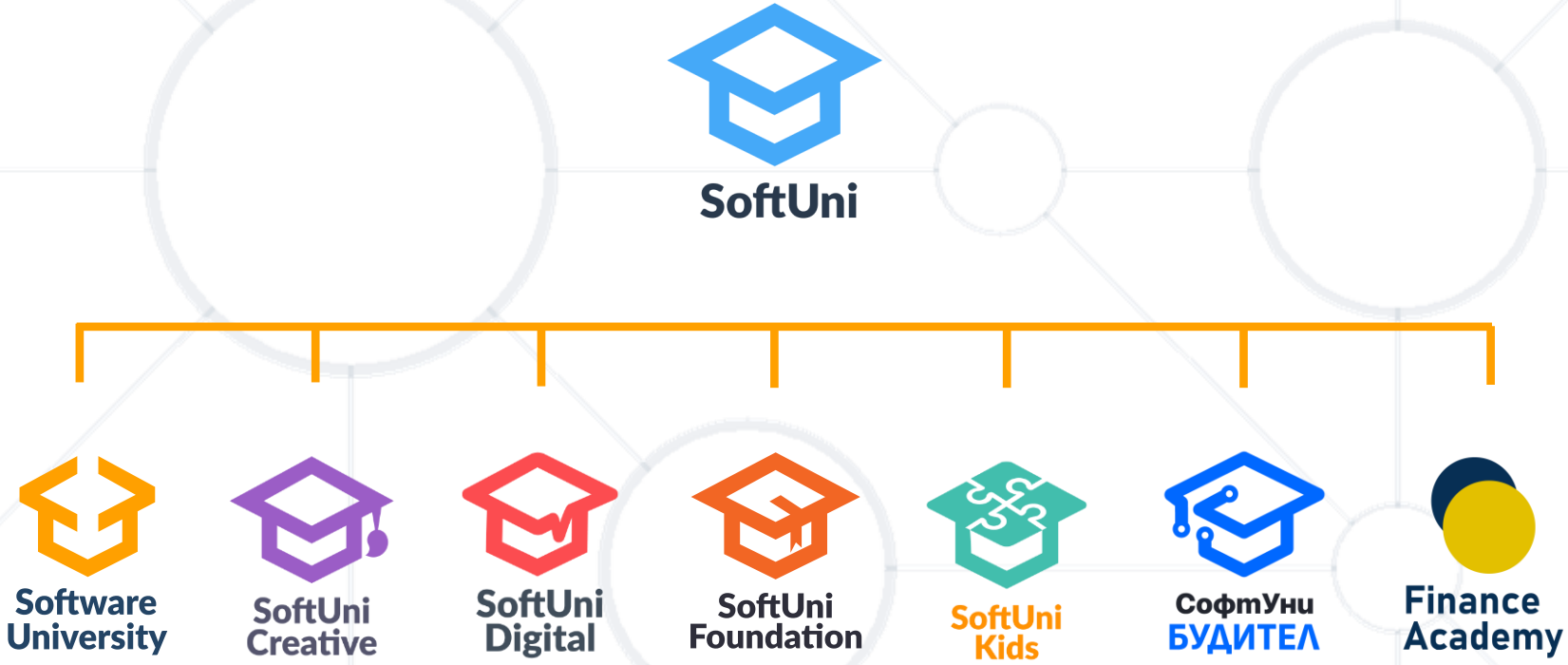
Solution: Customize the Admin

```
@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = ('name', 'category', 'price', 'created_on')
    search_fields = ('name', 'category', 'supplier')
    list_filter = ('category', 'supplier')
    fieldsets = (
        ('General Information', {
          'fields': ('name', 'description', 'price', 'barcode')}),
        ('Categorization', {'fields': ('category', 'supplier')}),
    )
    date_hierarchy = 'created_on'
```

Summary

- Migrations propagate changes
- Data Migrations
  - Alter data in DB
- Django Admin Site
  - Access - createsuperuser
  - Customizations

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

SOFTWARE GROUP

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction, or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg