Python - Django

Django official documentation page - https://docs.djangoproject.com/en/4.0/

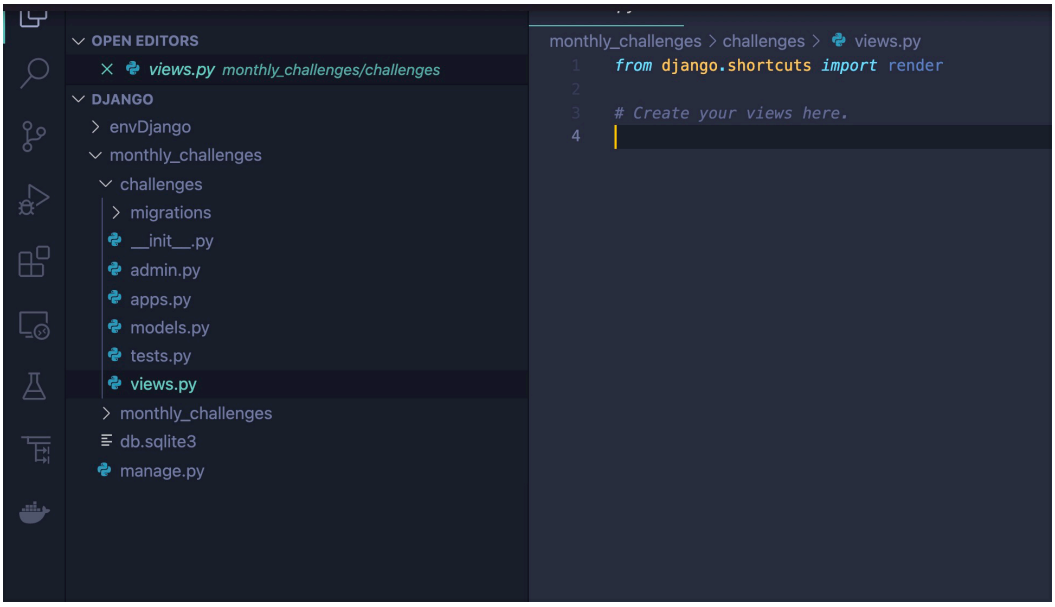
“django-admin startproject monthly_challenges”

“python3 manage.py runserver” -> runs the server locally on port 8000.
(run this in command line).

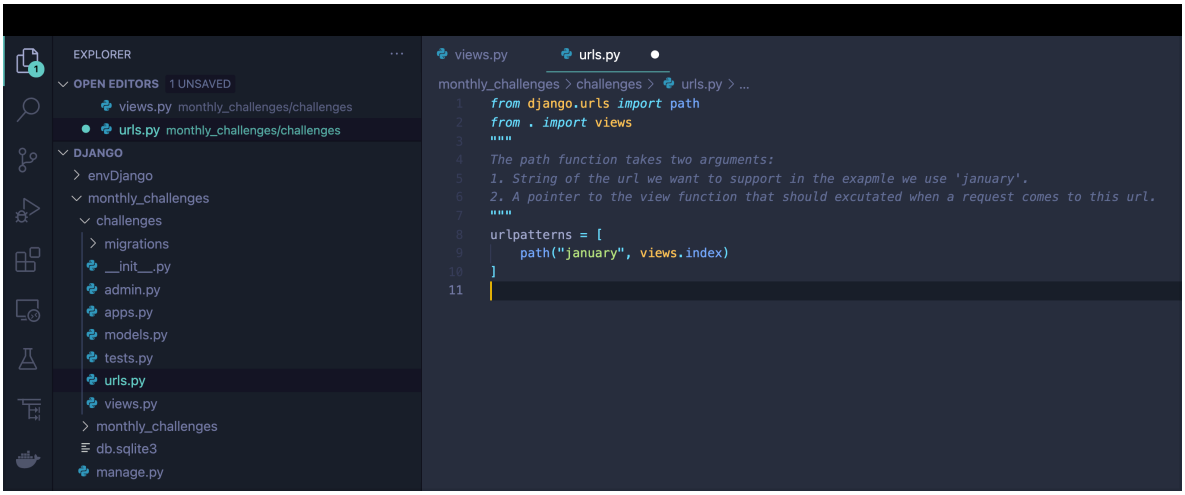
“python3 manage.py startapp challenges” -> build app “Django” app.
(run this in command line).

To Start a venv - “source tutorial-env/bin/activate” if the name of the venv is “tutorial-env”.
EXAMPLE - “source /Users/remez19/vsProjects/Django/envDjango/bin/activate ”

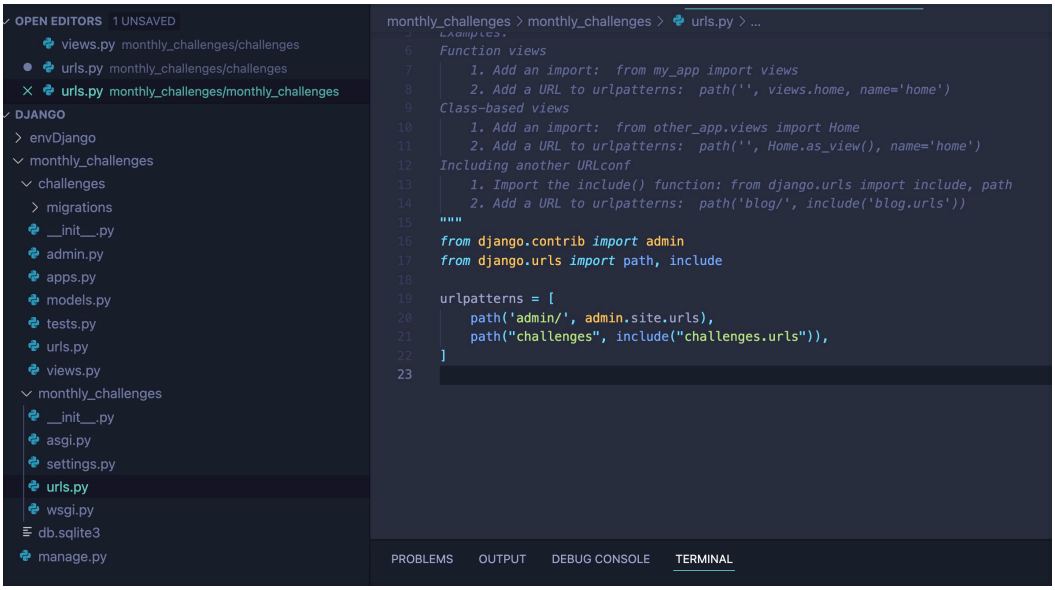
Views - in Django views are defined by functions or classes.
We write the code in the “App” directory in the views.py file.



We want to return respone to the client. the function name can be anything we like.
The return type is an Object called - "HttpRespone(), inside the () we pass the data we want to respone with."
The paramter that the function gets is a request. Django handle the request and calls the right function.
Inorder to bind the function with the event that trrigers it we create a new file "urls.py".
Inside this file we create a list of all the urls we want in our website.



Next we need to connect the “App” to the project - we do this by adding a path to the ‘urls.py’ file in the project directory.
Like this: ** Add ‘/’ after challenges.



Dynamic Path Segments & Captured Values:
Using '<>' let django to know that we don't care about the value of the url that comes after challenges,
Instead we tell him that all the urls that starting with 'challenges/' should handle by the same function.
Then we need to add an argument to the views.monthly_challenges function with the name we chose in this case "month”.

```
urlpatterns = [
    path("<month>", views.monthly_challenges),

]
```

Path Converters:
With this we can specify what we do expect the url to be (str, int , ...).
The order in with we write this patterns matter. In the exapmle below we first try to see if the url can be an int, then as a string.

```
urlpatterns = [
    path("<int:month>", views.monthly_challenges_number),
    path("<str:month>", views.monthly_challenges),
```

Redirects:
To redirect to other views we use the “HttpResponseRedirect()”, as an argument we pass the url we want to redirect to.

```
def monthly_challenges_number(request, month):
    if month == 1:
        return HttpResponse(the_remez_david_var)
    else:
        return HttpResponseRedirect("/challenges/boyohboy")
```

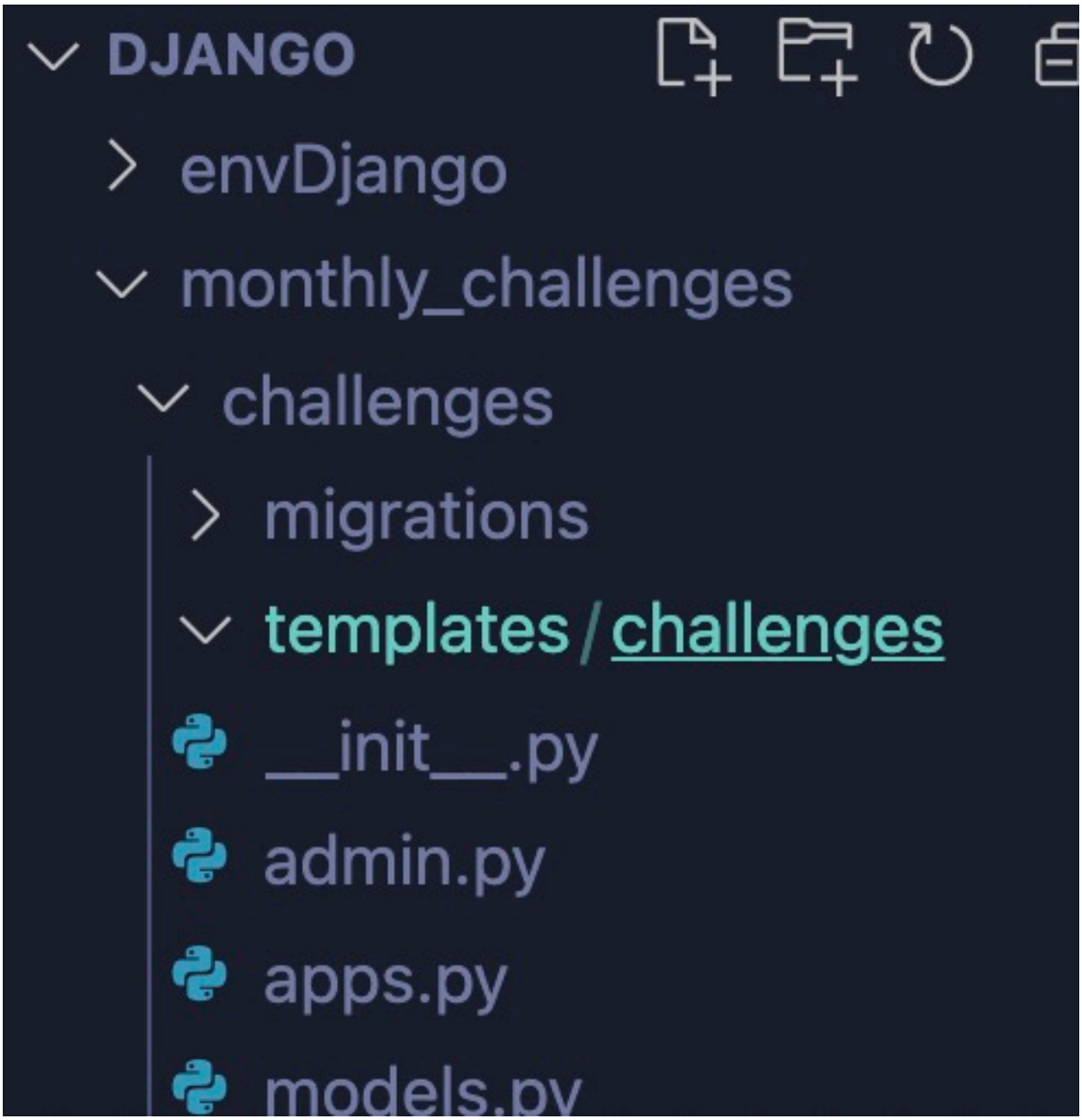
The Reverse Function & Named URLs:
A method to make redirections more dynamic and not hard coded example below:

```
urlpatterns = [
    path("<int:month>", views.monthly_challenges_number),
    path("<str:month>", views.monthly_challenges, name='month_challenges'),
]
```

```
def monthly_challenges_number(request, month):
    if month == 404:
        return HttpResponseNotFound("MotherFucker not supported!")
    if month == 1:
        return HttpResponse(the_remez_david_var)
    else:
        # return HttpResponseRedirect("/challenges/Shame") Can be more dynamic by switching to :
        # effectively builds up a path -> /challenges/Shame
        redirect_path = reverse("month_challenges", args=["Shame"])
        return HttpResponseRedirect(redirect_path)
```

Sending html files and injecting data into them.

Adding & Registering Templates:
The templates are html flies that we can inject some data into them.
First we create a sub folder named “templates” inside the relevant Django app (This is a convension).
Then we create another sub-folder inside the “templates” folder. The name of this folder should be the name of the app.

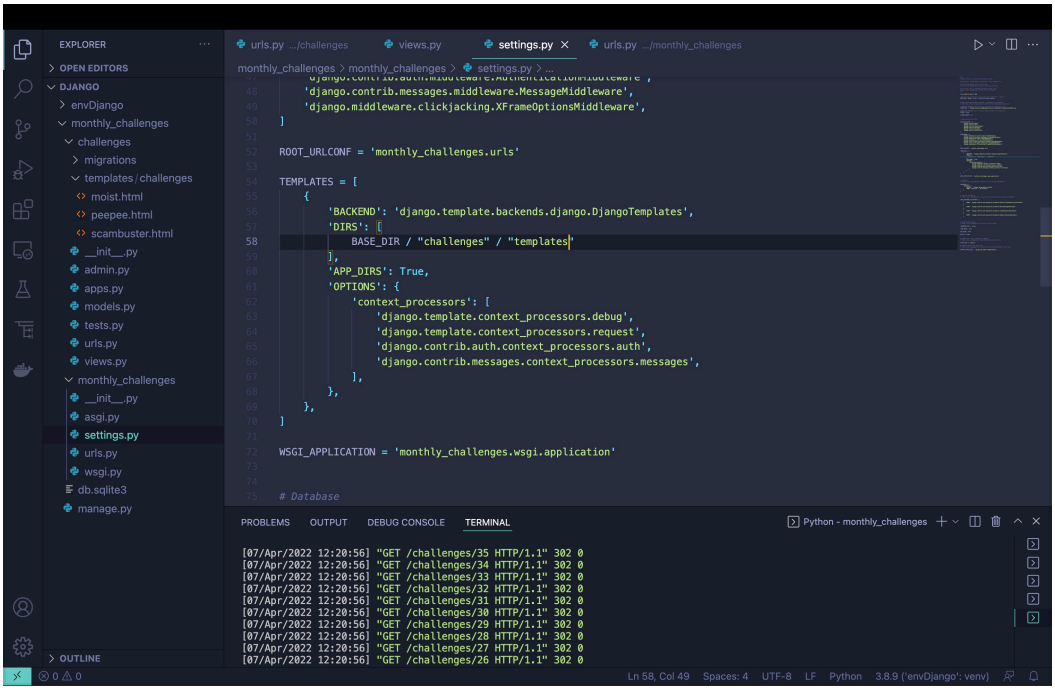



```
from django.template.loader import render_to_string
```

In order to tell Django to look for our templates with need to change the “settings.py ” file and add the relevant paths as in the example:
First way : (Making Django aware of our app)

```
SECRET_KEY = 'django-insecure-qb9@zig6*l$n7*olof+*owb72z#t!facqmsl3a*55dt#5n)rgv'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'challenges',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Second option - adding the path to the templates like this:



After that we can use the like this:

```
def remez_page(request, month):
    response_to_send = ""
    youtubers = list(youtube_dic.keys())
    if month in youtubers:
        response_to_send = render_to_string(youtube_dic[month])
        return HttpResponse(response_to_send)
    else:
        return HttpResponseNotFound("Not a valid page!")

# Template
```

Or we can do it like this:

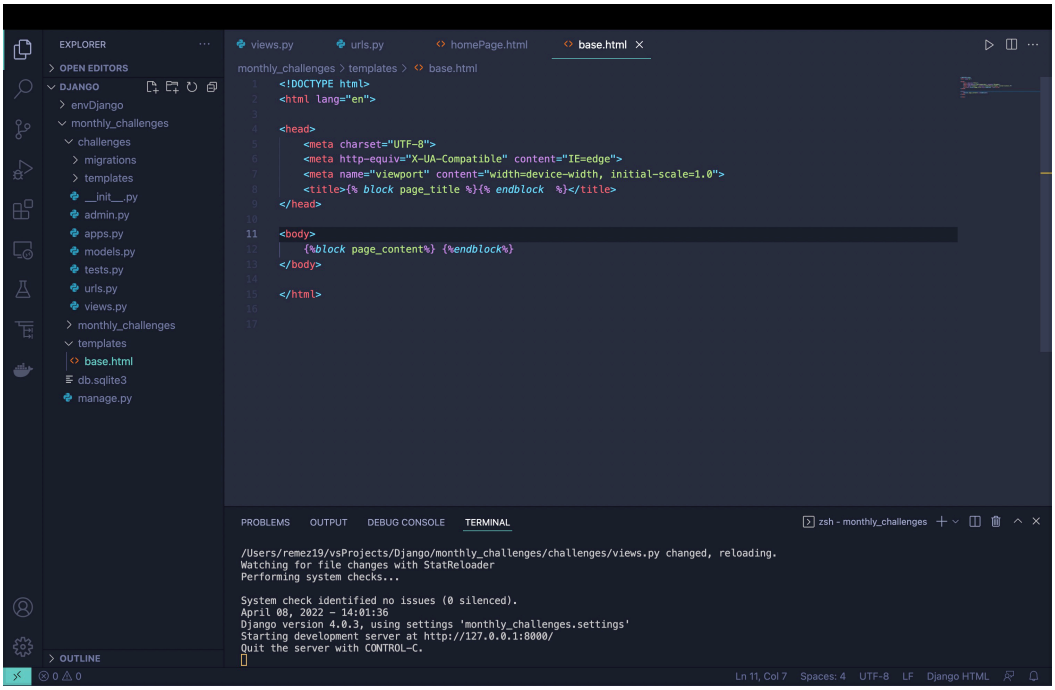
```
def remez_page(request, month):
    youtubers = list(youtube_dic.keys())
    if month in youtubers:
        return HttpResponse(render(request, youtube_dic[month]))
    else:
        return HttpResponseNotFound("Not a valid page!")
```

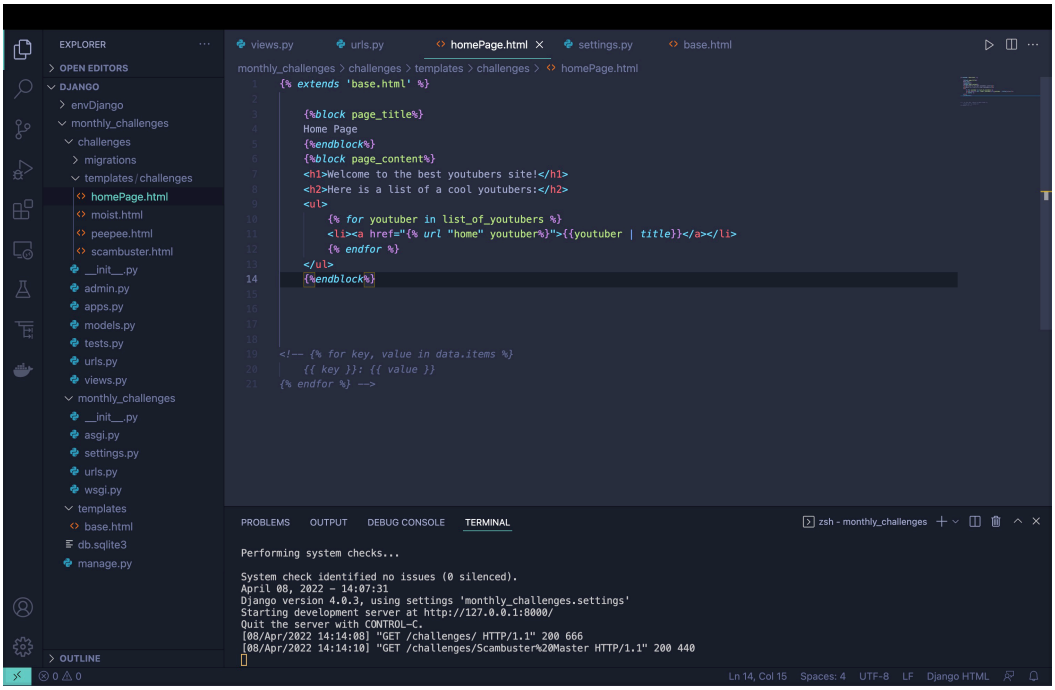
Django built in filters - https://docs.djangoproject.com/en/4.0/ref/templates/builtins/

Creating dynamic urls the right way using the built in DTL

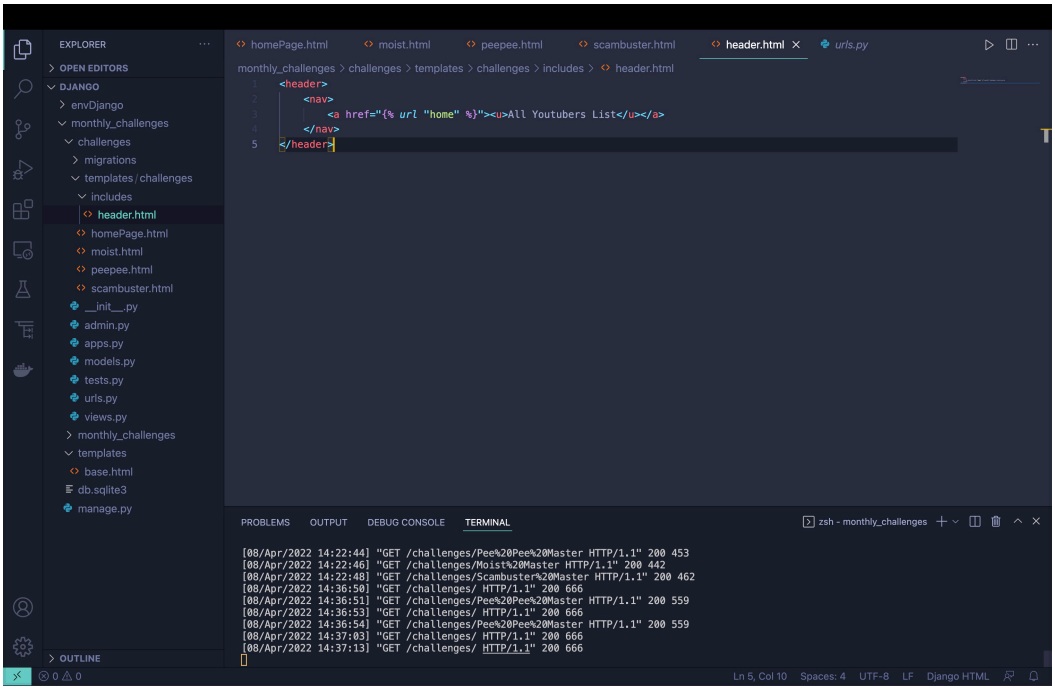
```
<ul>
    {% for youtuber in list_of_youtubers %}
    <li><a href="{% url "home" youtuber%}">{{youtuber | title}}</a></li>
    {% endfor %}
</ul>
```

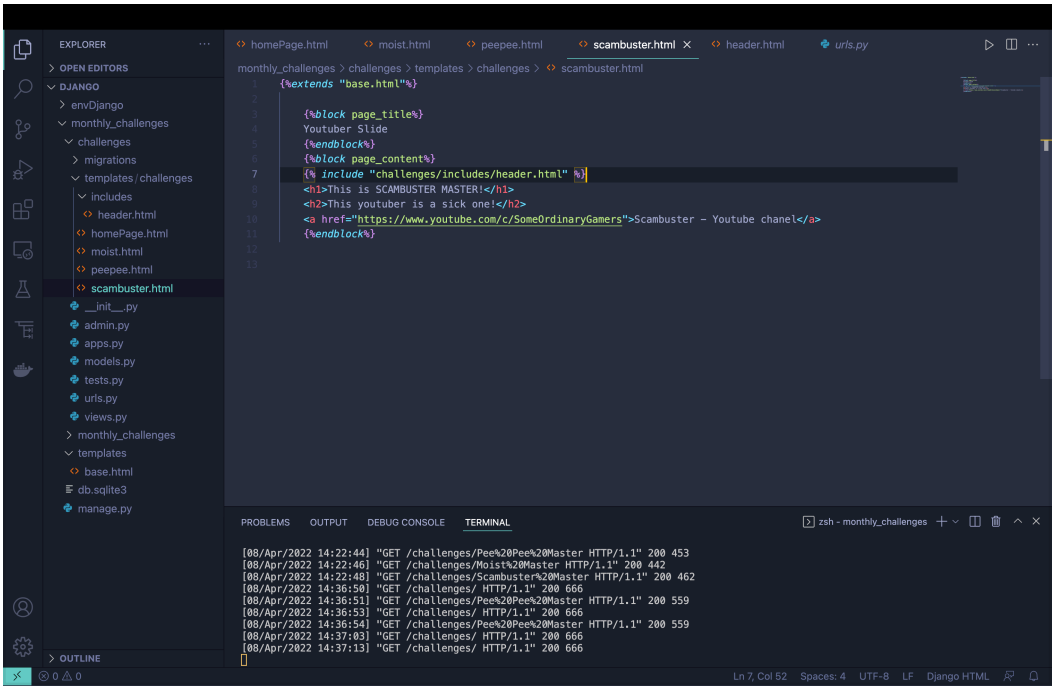
Template Inheritance:
We this we can inject a block of data into a HTML file like this:
1. Adding a folder named templates.
2. Creating the files that we want other templates to inherent from.
3. Adding the dir for Django to look for in the “settings.py” file.
4. Finally we tell Django where to inject our data.





Including Partial Template Snippets:
Important to know that vars that we can use the vars that are in other files that use include
Example including a header:

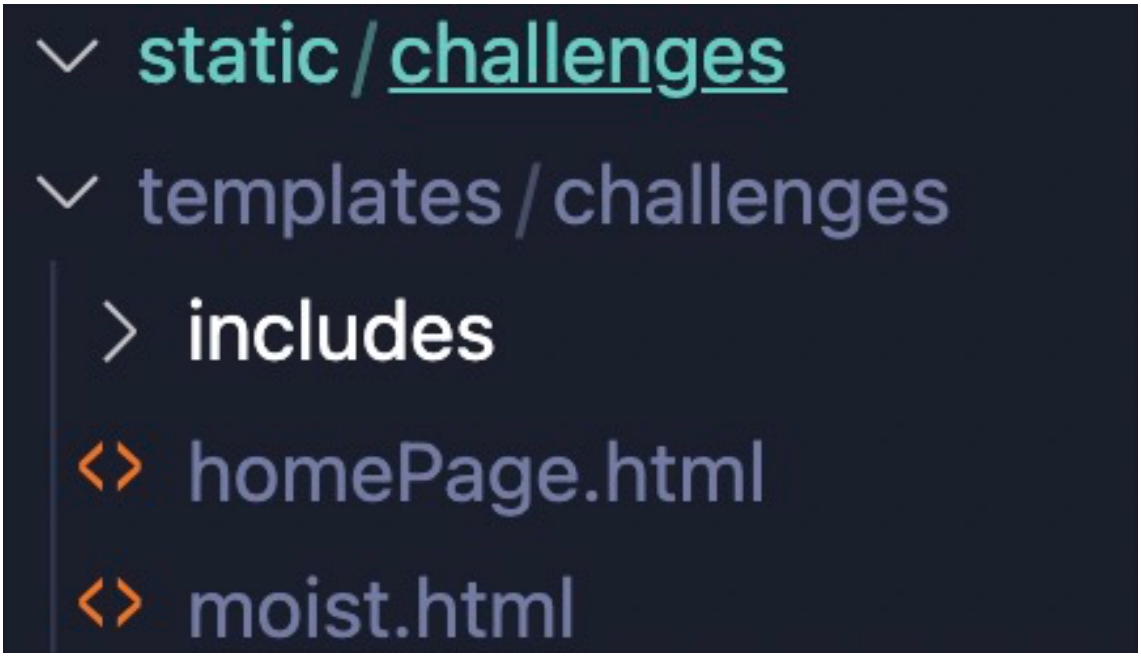


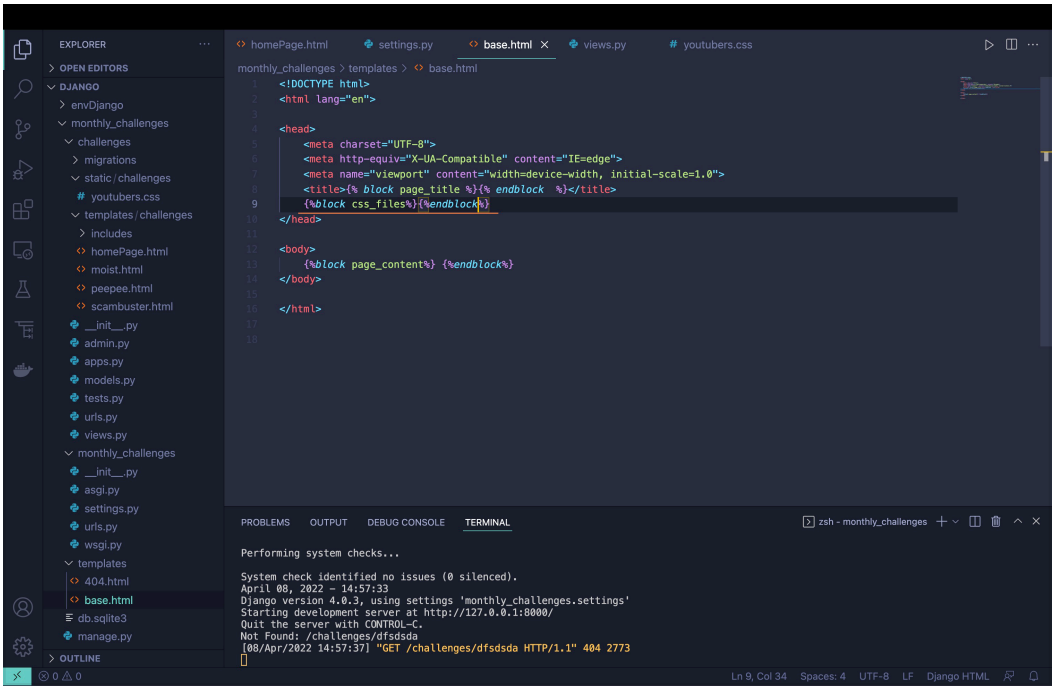


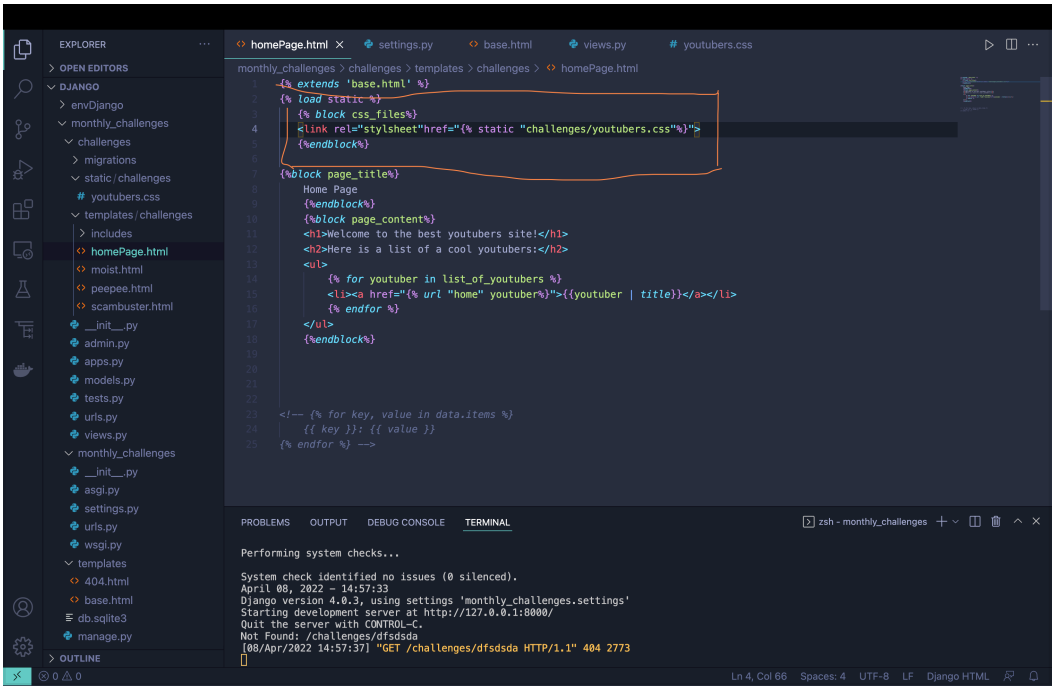
404 Template:
Adding a 404.html file to the “templates” folder in the root project and using “raise Http404()” will load our custom 404.html file to present the relevant data.

Adding Static Files (CSS files):

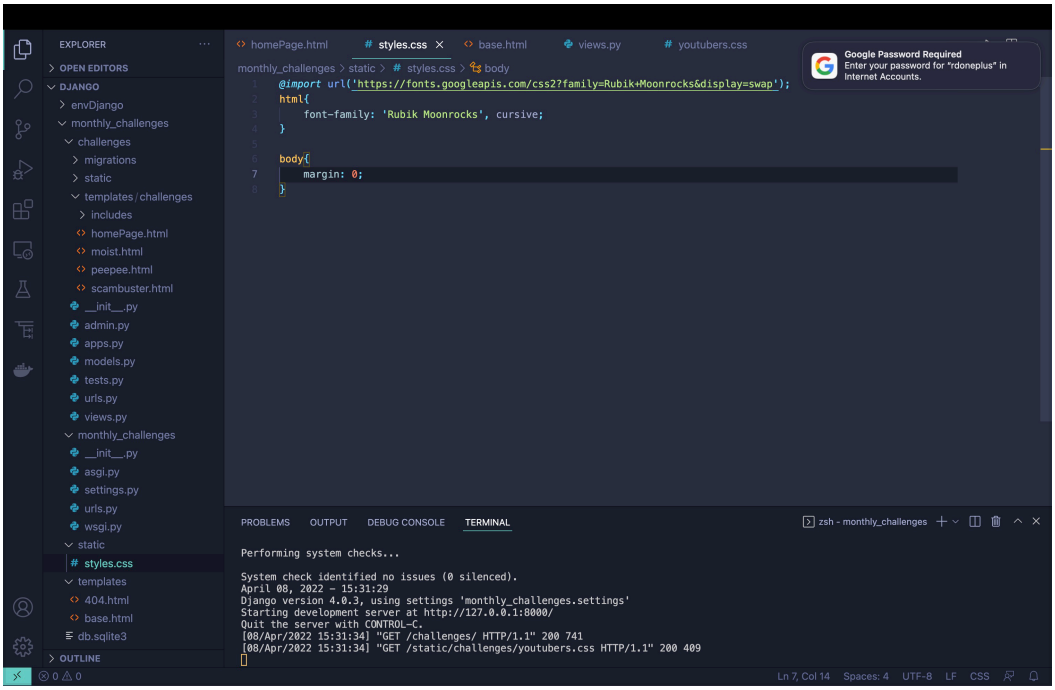
1. Adding a folder named “static” to the app we want. (Same concept as the templates folder for the html)
2. Best practice is to create another sub-folder inside the static folder with name of our app.

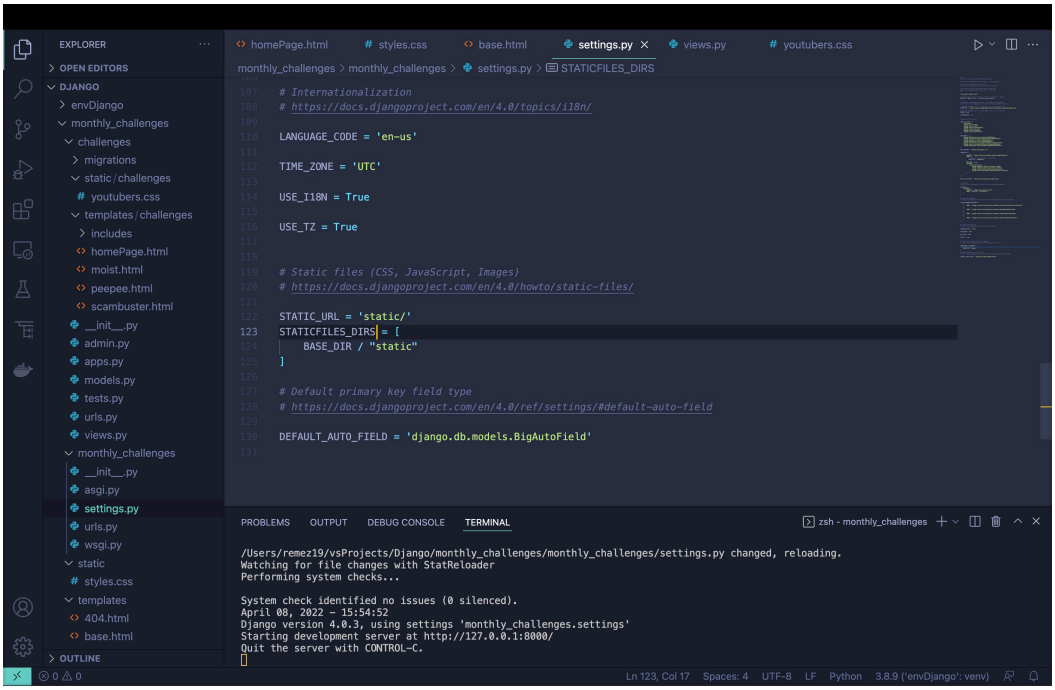

static / challenges
templates / challenges
includes
homePage.html
moist.html


EXPLORER
OPEN EDITORS
DJANGO
envDjango
monthly_challenges
challenges
migrations
static / challenges
youtubers.css
templates / challenges
includes
homePage.html
moist.html
peepee.html
scambuster.html
__init__.py
admin.py
apps.py
models.py
tests.py
urls.py
views.py
monthly_challenges
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
templates
404.html
base.html
db.sqlite3
manage.py
OUTLINE
homePage.html
settings.py
base.html
views.py
youtubers.css
monthly_challenges > templates > base.html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>{% block page_title %}{% endblock %}</title>
{%block css_files%}{%endblock%}
</head>
<body>
{%block page_content%} {%endblock%}
</body>
</html>
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
zsh - monthly_challenges
Performing system checks...
System check identified no issues (0 silenced).
April 08, 2022 - 14:57:33
Django version 4.0.3, using settings 'monthly_challenges.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
Not Found: /challenges/dfsdsda
[08/Apr/2022 14:57:37] "GET /challenges/dfsdsda HTTP/1.1" 404 2773
Ln 9, Col 34
Spaces: 4
UTF-8
LF
Django HTML



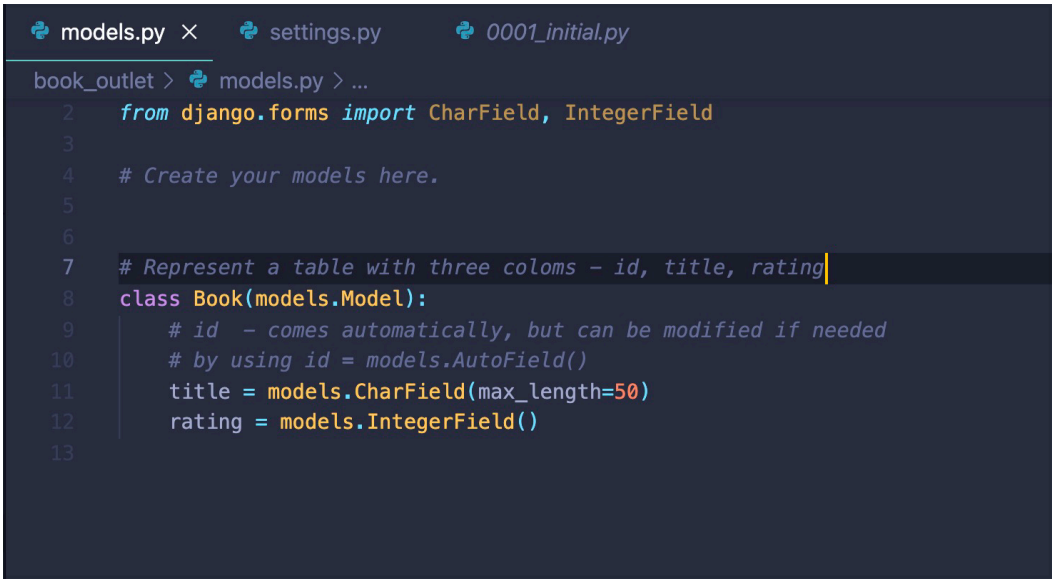
Adding Global Static Files:
Same idea as the global templates file.
https://fonts.google.com
As an example we add a global css file to control our overall font style:





Migrations:

Creating classes that eventually Django will transform into a data base tables as such:



Next we need to make Django aware of this classes so we use the following command:

“python3 manage.py makemigrations”

Than we follow up this command by another command:

“python3 manage.py migrate”

With this command Django goes over all the migration folders (of each app) And make sure to run all migrations that not have been executed yet.

To work with the data base that we created we use the following command:

“Python3 manage.py shell” this will open up a interactive shell that will let us insert data or retrive data from the database that we created.

The pic below shows how we can sore data in the data base and **save** it.

```
remez19 book_store %python3 manage.py shell
Python 3.8.9 (default, Feb 18 2022, 07:45:33)
[Clang 13.1.6 (clang-1316.0.21.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from book_outlet.models import Book
>>> harry_poter = Book(title="Harry poter 1", rating=5)
>>> harry_poter.save()
```

Getting all Entries

Using the “all()” method will give us back all the data entries that we created

```
>>> Book.objects.all()
<QuerySet [<Book: Book object (1)>, <Book: Book object (2)>]>
>>>
```

Overriding the str function in our classes:
We override the merthod with the name “__str__” (similar to toString() in Java)

```
6
7   # Represent a table with three coloms - id, title, rating
8   class Book(models.Model):
9       # id  - comes automatically, but can be modified if need
10      # by using id = models.AutoField()
11      title = models.CharField(max_length=50)
12      rating = models.IntegerField()
13
14      def __str__(self) -> str:
15          return f"{self.title} ({self.rating})"
16
```

Output Change after the overide.

```
remez19 book_store %ls
book_outlet     book_store      db.sqlite3      manage.py
remez19 book_store %python3 manage.py shell
Python 3.8.9 (default, Mar 30 2022, 13:51:16)
[Clang 13.1.6 (clang-1316.0.21.2.3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from book_outlet.models import Book
>>> Book.objects.all()
<QuerySet [<Book: Harry poter 1 (5)>, <Book: The Lord Of The Rings (4)>]>
>>>
```

Using Validators:

We want to use validators in order to control the data we want to insert.
As an example in this book project we would like to control the rating field (1<= rating <= 5):
Import built in validators (We can also construct our own validators see - https://docs.djangoproject.com/en/4.0/ref/validators/):

```python
from django.core.validators import MinValueValidator, MaxValueValidator

# Create your models here.
```

```python
    # Represent a table with three coloms - id, title, rating
    class Book(models.Model):
        # id  - comes automatically, but can be modified if n
        # by using id = models.AutoField()
        title = models.CharField(max_length=50)
        rating = models.IntegerField(validators=[
            MinValueValidator(1),
            MaxValueValidator(5),
        ])

        def __str__(self) -> str:
            return f"{self.title} ({self.rating})"
```

If we already have records in our database that didn't have the files we added when we first insert them, than we have to set a default
Value for them using: "null=" (Setting null=True) key-argument or the "default=" key-argument.
Also we can use the "blank=True" to allow empty fields in our database

```python
class Book(models.Model):
    # id  - comes automatically, but can be modified if needed
    # by using id = models.AutoField()
    title = models.CharField(max_length=50)
    rating = models.IntegerField(validators=[
        MinValueValidator(1),
        MaxValueValidator(5),
    ])
    author = models.CharField(null=True, max_length=100)
    is_bestselling = models.BooleanField(default=False)

    def __str__(self) -> str:
        return f"{self.title} ({self.rating})"

```

After this changes we run two commands:

1. python3 manage.py makemigrations.
2. python3 manage.py migrate.

Updating Data:

Updating the two new fields we added - “author” and “is_bestselling”:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

remez19 book_store %python3 manage.py shell
Python 3.8.9 (default, Mar 30 2022, 13:51:16)
[Clang 13.1.6 (clang-1316.0.21.2.3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from book_outlet.models import Book
>>> harry_poter = Book.objects.all()[0]
>>> harry_poter.title
'Harry poter 1'
>>> lord_of_the_rings = Book.objects.all()[1]
>>> lord_of_the_rings.title
'The Lord Of The Rings'
>>> harry_poter.author = "J.K Rowling"
>>> harry_poter.is_bestselling = True
>>> harry_poter.save()
>>> lord_of_the_rings.save()
>>> Book.objects.all()[0].author
'J.K Rowling'
>>>
```

Deleting Data:

```
>>> harry_poter = Book.objects.all()[0]
>>> harry_poter.delete()
(1, {'book_outlet.Book': 1})
>>>
```

Create Instead of Save:

```
>>> Book.objects.create(title="Harry Potter 1", author= "J.K Rowlling", rating=5, is_bestselling=True)
<Book:
        Book Ttitle: Harry Potter 1
        Book author: J.K Rowlling
        Book Rating: (5)
        Book Best Selling: True
        >
```

Querying & Filtering Data:

By id:

```
>>> from book_outlet.models import Book
>>> Book.objects.get(id=3)
<Book:
        Book Ttitle: Harry Potter 1
        Book author: J.K Rowlling
        Book Rating: (5)
        Book Best Selling: True
        >
>>>
```

By title:

```
>>> Book.objects.get(title="My Story")
<Book:
        Book Ttitle: My Story
        Book author: Remez
        Book Rating: (3)
        Book Best Selling: False
        >
>>>
```

And just like this for every field we create.

Get will always give just one entry even if we have multiple entries that have the same data.

If their are more than one entry than get will raise an error!

If we want to get all the entries that fulfill some conditions that we will want to use **“filter”:**

```
>>> Book.objects.filter(is_bestselling=False)
<QuerySet [<Book:
        Book Ttitle: The Lord Of The Rings
        Book author: Remez
        Book Rating: (4)
        Book Best Selling: False
        >, <Book:
        Book Ttitle: My Story
        Book author: Remez
        Book Rating: (3)
        Book Best Selling: False
        >, <Book:
        Book Ttitle: Gigo's Story
        Book author: Gigo
        Book Rating: (1)
        Book Best Selling: False
        >]>
```

```
>>> Book.objects.filter(is_bestselling=False,rating=4)
<QuerySet [<Book:
        Book Ttitle: The Lord Of The Rings
        Book author: Remez
        Book Rating: (4)
        Book Best Selling: False
        >]>
```

“rating__lte = 4” => rating<=4

```
>]>
>>> Book.objects.filter(rating__lte=4)
<QuerySet [<Book:
        Book Ttitle: The Lord Of The Rings
        Book author: Remez
        Book Rating: (4)
        Book Best Selling: False
        >, <Book:
        Book Ttitle: My Story
        Book author: Remez
        Book Rating: (3)
        Book Best Selling: False
        >, <Book:
        Book Ttitle: Gigo's Story
        Book author: Gigo
        Book Rating: (1)
        Book Best Selling: False
        >]>
>>>
```

Official filters documintation : **https://docs.djangoproject.com/en/4.0/topics/db/queries/**

"or" Conditions

Starting by importing the built-in module “Q” and as follows in the example.
For using “or” we use -> “|” , for “and” we use -> “,”

```
>>> from django.db.models import Q
>>> Book.objects.filter(Q(rating__lt=3) | Q(is_bestselling=True))
<QuerySet [<Book:
        Book Ttitle: Harry Potter 1
        Book author: J.K Rowlling
        Book Rating: (5)
        Book Best Selling: True
        >, <Book:
        Book Ttitle: Gigo's Story
        Book author: Gigo
        Book Rating: (1)
        Book Best Selling: False
        >]>
>>>
```

Rendering Queried Data in the Template:

Getting the list of books from our Book model and passing the data to the “index.html” template.

views.py index.html

book_outlet > views.py > ...

```python
from django.shortcuts import render
from .models import Book
# Create your views here.


def index(request):
    books = Book.objects.all()
    return render(request, "book_outlet/index.html", {
        "books": books,
    })

```

Then we continue to loop thru all the list of books in the “index.html” template.

views.py index.html

book_outlet > templates > book_outlet > index.html

```
{% extends 'book_outlet/base.html' %}
{% block  title%}
All Books
{% endblock  %}
{% block content %}
    <ul>
        {% for book in books%}
        <li>{{book.title}} (Rating: {{book.rating}})</li>
        {% endfor %}
    </ul>

{% endblock  %}
```

Another examples:

views.py urls.py index.html book_detail.html

book_outlet > views.py > book_detail

```python
def index(request):
    books = Book.objects.all()
    return render(request, "book_outlet/index.html", {
        "books": books,
    })


def book_detail(request, id):
    book = Book.objects.get(pk=id)
    return render(request, "book_outlet/book_detail.html", {
        "title": book.title,
        "author": book.author,
        "rating": book.rating,
        "is_bestselling": book.is_bestselling,
    })
```

views.py urls.py index.html book_detail.html

book_outlet > templates > book_outlet > book_detail.html

```
{% extends 'book_outlet/base.html' %}
{% block title %}
    {{title}}
{% endblock title %}

{% block content %}
 <h1>{{title}}</h1>
 <h2>{{author}}</h2>
 <p>The book has a rating of {{rating}}
    {% if is_bestselling %}
       and is a bestseller.
    {% else %}
       but isn't a bestseller.
    {% endif %}
 </p>
{% endblock content %}
```

Model URLs:

views.py urls.py index.html models.py book_de

book_outlet > models.py > Book > get_absolute_url

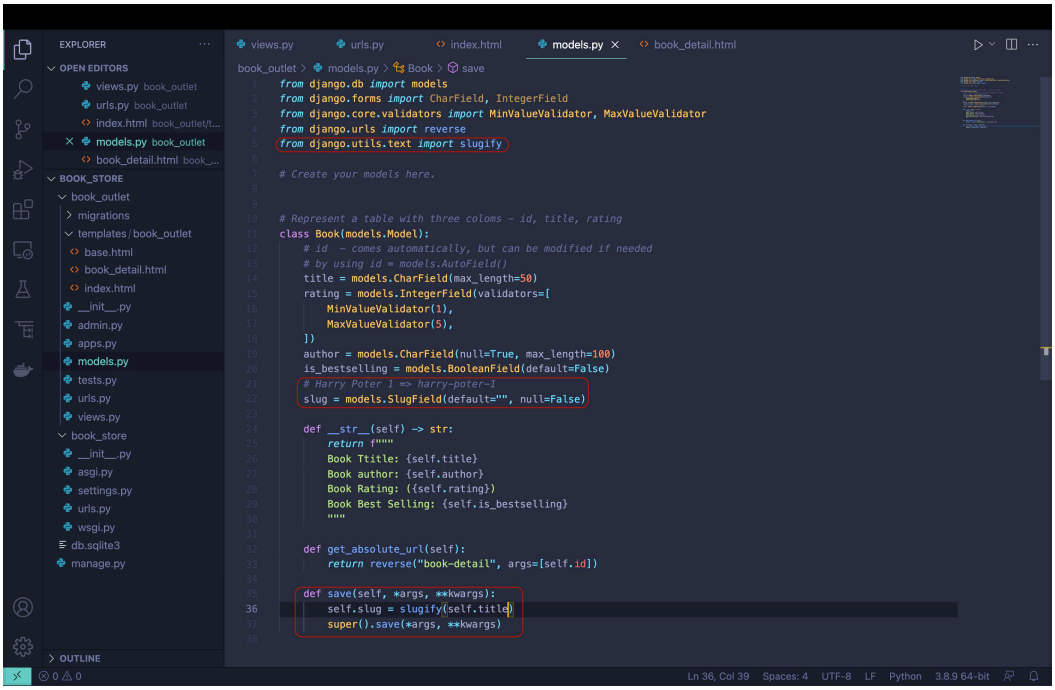
```python
            MaxValueValidator(5),
        ])
        author = models.CharField(null=True, max_length=100)
        is_bestselling = models.BooleanField(default=False)

        def __str__(self) -> str:
            return f"""
            Book Ttitle: {self.title}
            Book author: {self.author}
            Book Rating: ({self.rating})
            Book Best Selling: {self.is_bestselling}
            """

        def get_absolute_url(self):
            return reverse("book-detail", kwargs={"pk": self.pk})
```

views.py urls.py index.html models.py book_detail.html

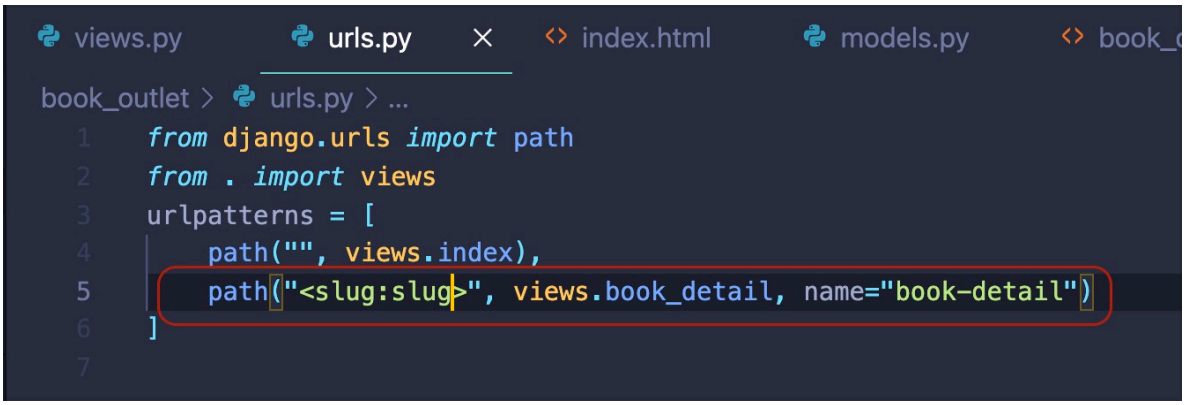
book_outlet > templates > book_outlet > index.html

```
{% extends 'book_outlet/base.html' %}
{% block  title%}
All Books
{% endblock  %}
{% block content %}
    <ul>
        {% for book in books%}
        <li><a href="{{book.get_absolute_url}}">{{book.title}}</a> (Rating: {{book.rating}})</li>
        {% endfor %}
    </ul>

{% endblock  %}
```

Adding a Slugfield & Overwriting Save:

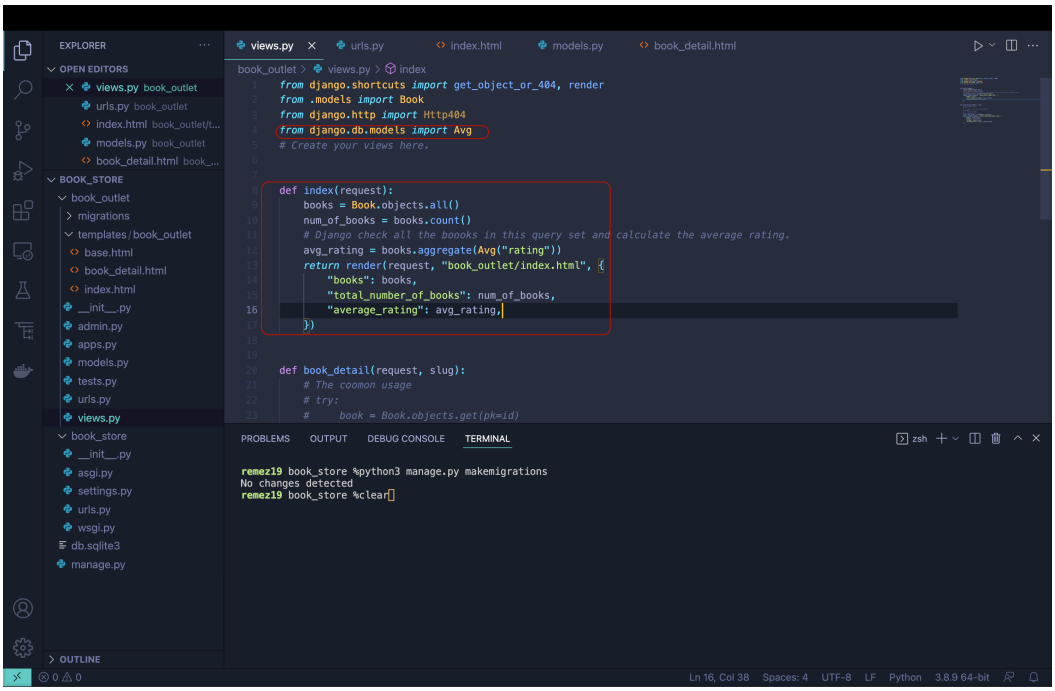


Using the Slug & Updating Field Options:



```python
def book_detail(request, slug):
    # The coomon usage
    # try:
    #     book = Book.objects.get(pk=id)
    # except:
    #     raise Http404()
    # Alternative way:
    book = get_object_or_404(Book, slug=slug)
    return render(request, "book_outlet/book_detail.html
        "title": book.title,
        "author": book.author,
        "rating": book.rating,
        "is_bestselling": book.is_bestselling,
    })
```

Aggregation & Ordering:



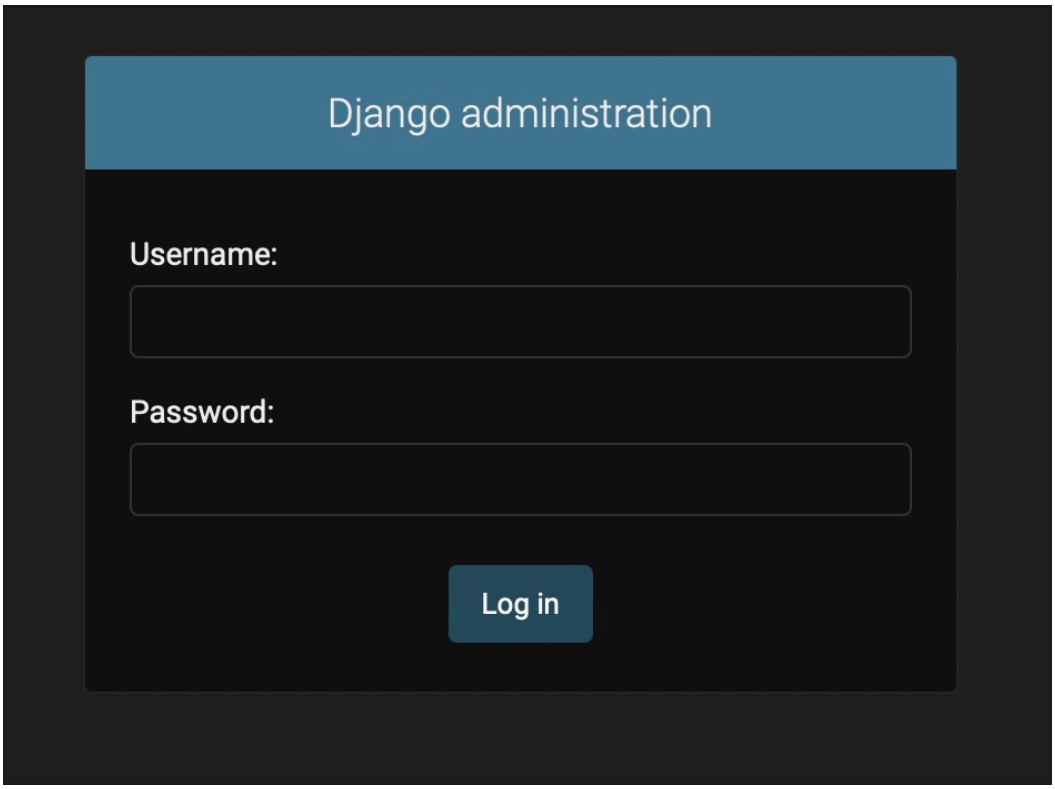
views.py urls.py index.html models.py

book_outlet > templates > book_outlet > index.html

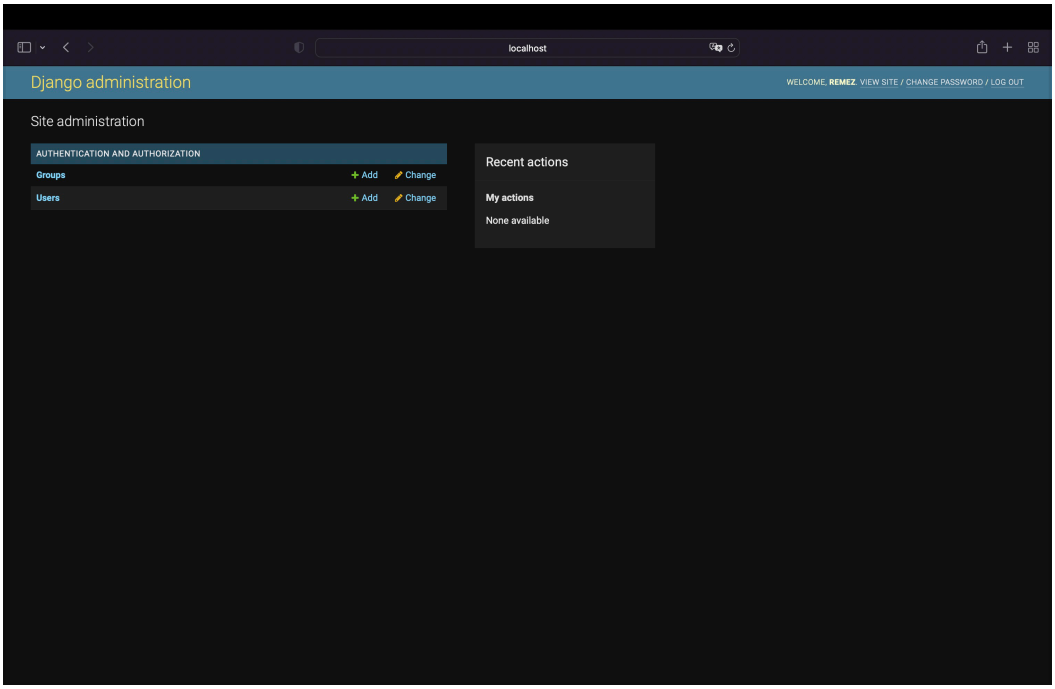
```
{% extends 'book_outlet/base.html' %}
{% block  title%}
All Books
{% endblock  %}
{% block content %}
    <ul>
       {% for book in books%}
       <li><a href="{{book.get_absolute_url}}">{{book.title}}</
       {% endfor %}
    </ul>
    <hr>
    <p>
        Total Number Of Books: {{total_number_of_books}}
    </p>
    <p>Average Rating: {{average_rating.rating__avg}}</p>
{% endblock  %}
```

Logging Data Into the Admin Panel:

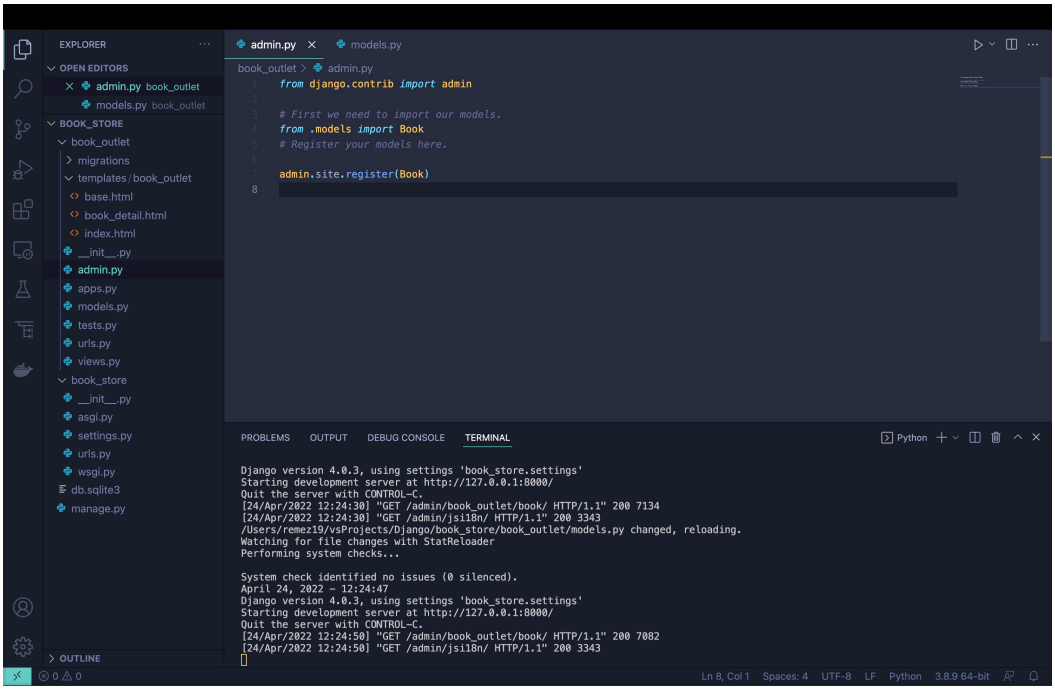
First we need to run the command - “python3 manage.py createsuperuser”. This command let us create a new super user “admin”. With this user we can log into the Django admin section.



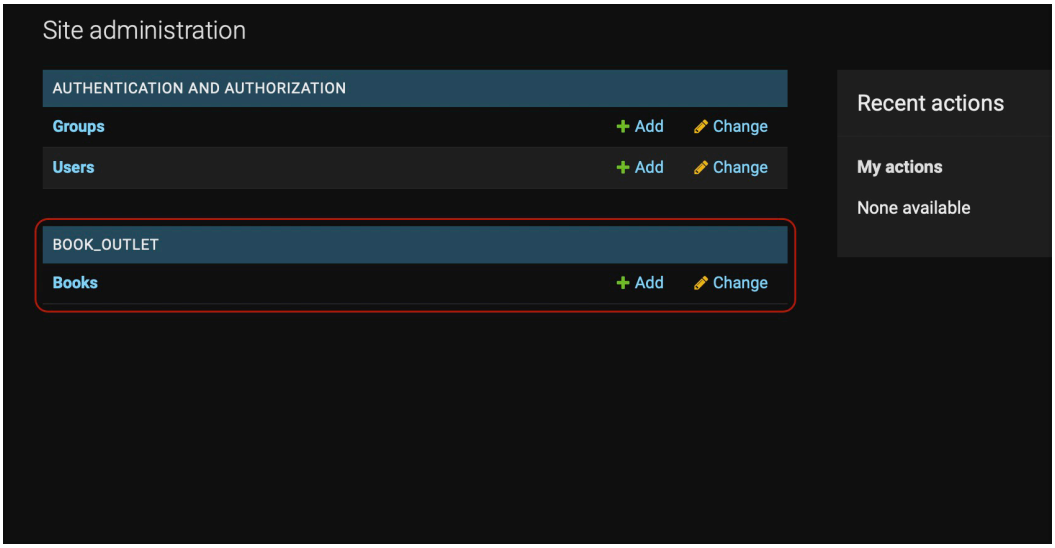
After login in:



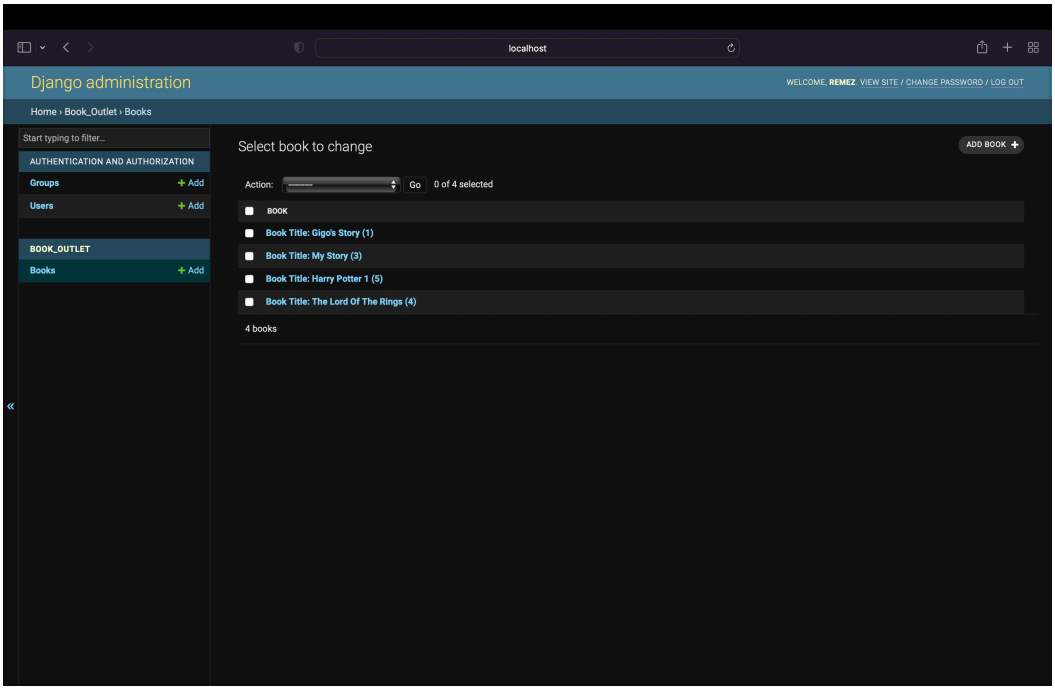
Adding Models to the Admin Area:

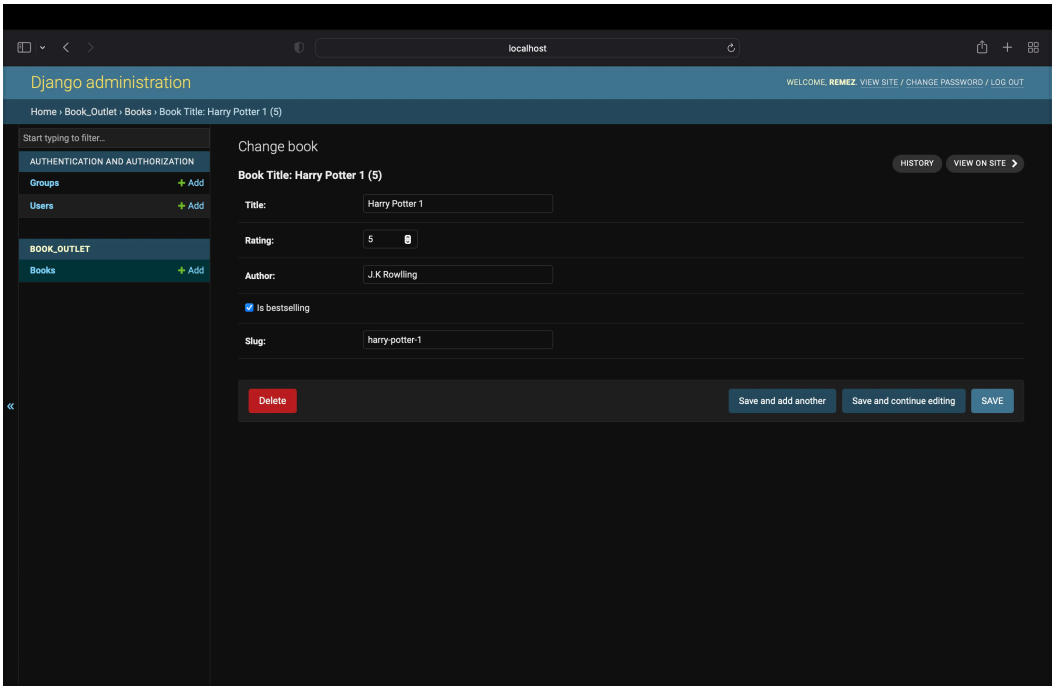


Now we will have this tab we can click on and see the “book” table:

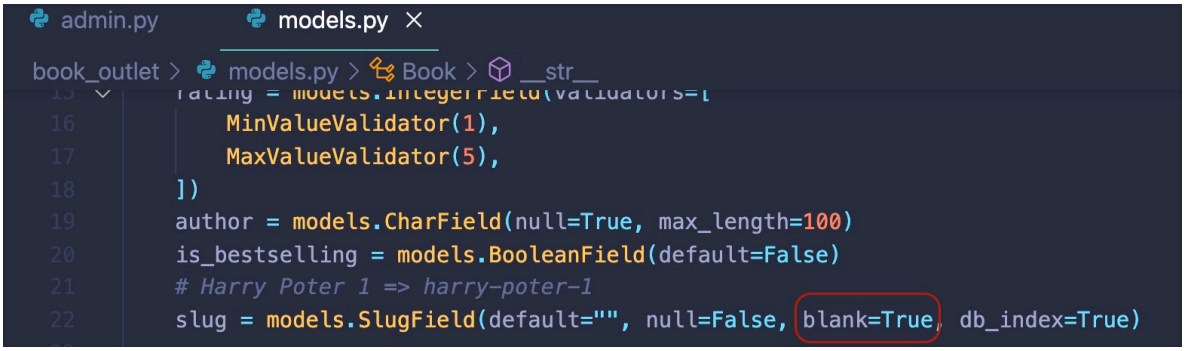


We this we can click on “Books” and see all our books the are currently in our table and, we can also edit book data if we want to:

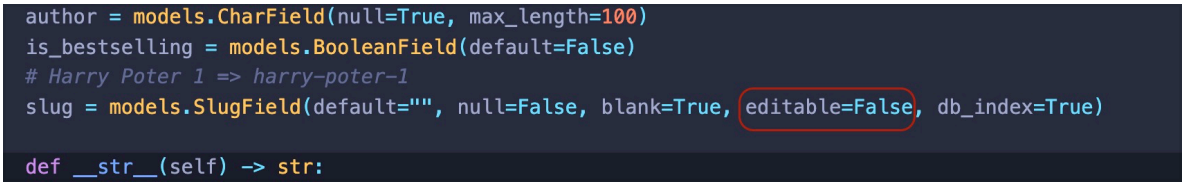




Configuring Model Fields:



With this field set to True - we can leave this field empty when we creating a new book thru the Django admin interface.
We can also make fields uneditable by:



Configuring the Admin Settings:

admin.py × models.py

book_outlet > admin.py > BookAdmin

```python
from django.contrib import admin

# First we need to import our models.
from .models import Book
# Register your models here.


class BookAdmin(admin.ModelAdmin):
    readonly_fields = ("slug",)


admin.site.register(Book, BookAdmin)
```

We can also make fields auto-populated by another fields as in this example:

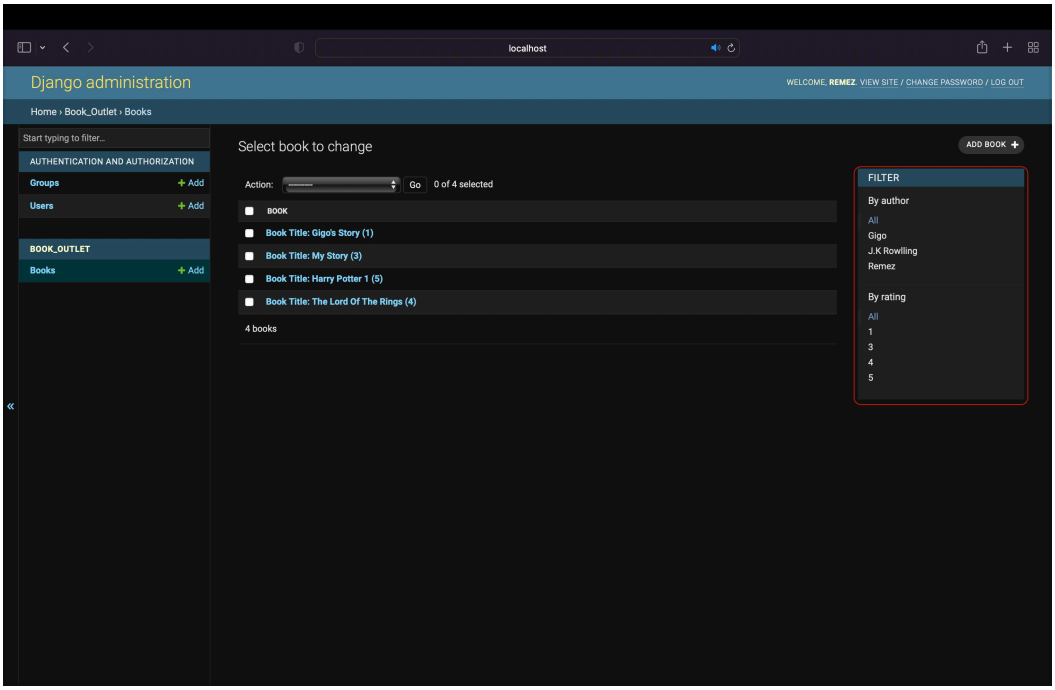
```python
class BookAdmin(admin.ModelAdmin):
    # readonly_fields = ("slug",)
    prepopulated_fields = {
        "slug": ("title",)
    }
```

We can also add filters as such:

admin.py × models.py

book_outlet > admin.py > BookAdmin

```python
from django.contrib import admin

# First we need to import our models.
from .models import Book
# Register your models here.


class BookAdmin(admin.ModelAdmin):
    # readonly_fields = ("slug",)
    prepopulated_fields = {
        "slug": ("title",)
    }
    # The filter we want to use to present the
    list_filter = ("author", "rating",)


admin.site.register(Book, BookAdmin)
```

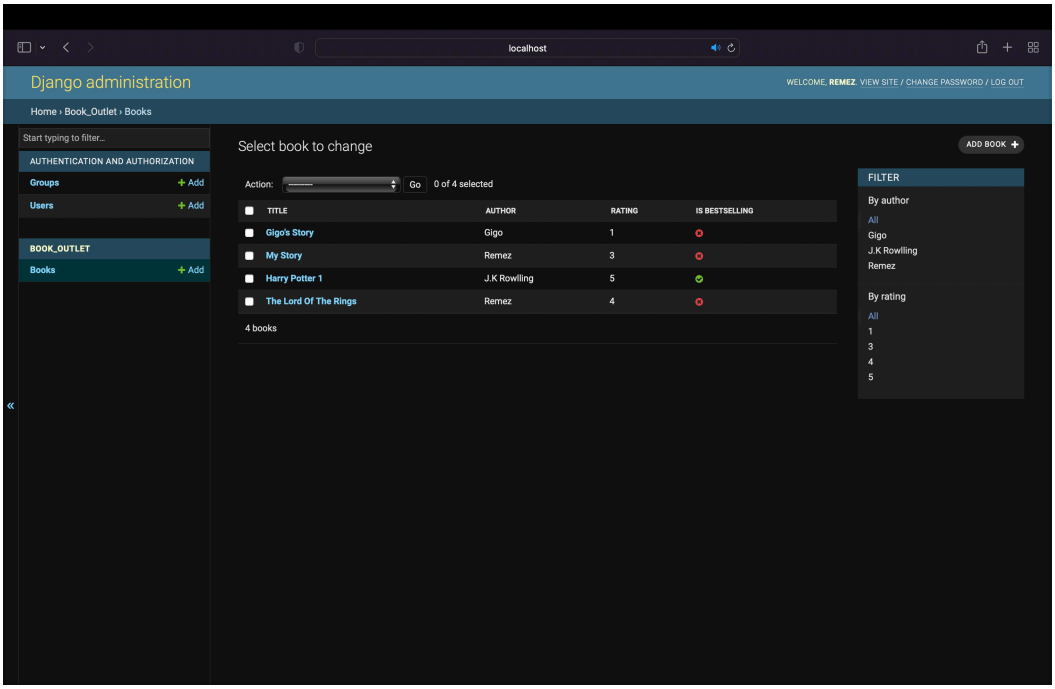


We can also change the way we display the data as such:

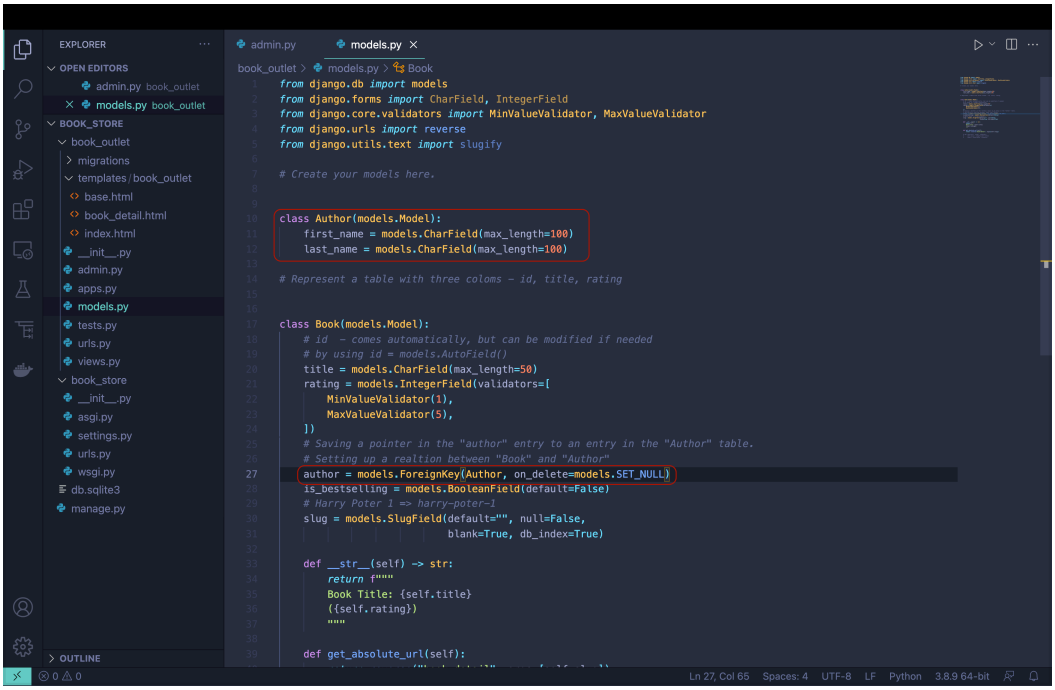
admin.py models.py

book_outlet > admin.py > BookAdmin

```python
from django.contrib import admin

# First we need to import our models.
from .models import Book
# Register your models here.


class BookAdmin(admin.ModelAdmin):
    # readonly_fields = ("slug",)
    prepopulated_fields = {
        "slug": ("title",)
    }
    # The filter we want to use to present the books in the admin se
    list_filter = ("author", "rating", )
    list_display = ("title", "author", "rating", "is_bestselling")
```



Understanding Relationship Types:
Adding a one-to-many Relation & Migrations:



We have a lot of options for ”on_delete=“. Each option define what happens when an author is deleted. In other words - what happens to the book if he doesn’t have an author.

Working with Relations in Python Code:

```
>>> from book_outlet.models import Book, Author
>>> jkrowling = Author(first_name="J.K.", last_name="Rowling")
>>> jkrowling.save()
>>> Author.objects.all()
<QuerySet [<Author: Author object (1)>]>
>>> hp1 = Book(title= "Harry Potter 1",rating=5, is_bestsellig=True, slug="harry-potter-1", author=jkrowling)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "/Users/remez19/Library/Python/3.8/lib/python/site-packages/django/db/models/base.py", line 559, in __init__
    raise TypeError(
TypeError: Book() got an unexpected keyword argument 'is_bestsellig'
>>> hp1 = Book(title= "Harry Potter 1",rating=5, is_bestselling=True, slug="harry-potter-1", author=jkrowling)
>>> hp1.save()
>>> Book.objects.all()
<QuerySet [<Book:
        Book Title: Harry Potter 1
        (5)
        >]>
>>> harrypotter = Book.objects.all()[0]
>>> harrypotter.title
'Harry Potter 1'
>>> harrypotter.author
<Author: Author object (1)>
>>> harrypotter.author.first_name
'J.K.'
```

Cross Model Queries:

```
>>> from book_outlet.models import Book, Author
>>> book_by_rowling = Book.objects.filter(author__last_name="Rowling")
>>> book_by_rowling
<QuerySet [<Book:
        Book Title: Harry Potter 1
        (5)
        >]>
>>>
```

In the next example we can use that we can “chain” query’s by “__” -> we step inside the author model and can access its fields like “last_name”:

```
>> >>> book_by_rowling = Book.objects.filter(author__last_name__contains="ling")
>> book_by_rowling
QuerySet [<Book:
       Book Title: Harry Potter 1
       (5)
       >]>
>>
```

Adding a one-to-one Relation:

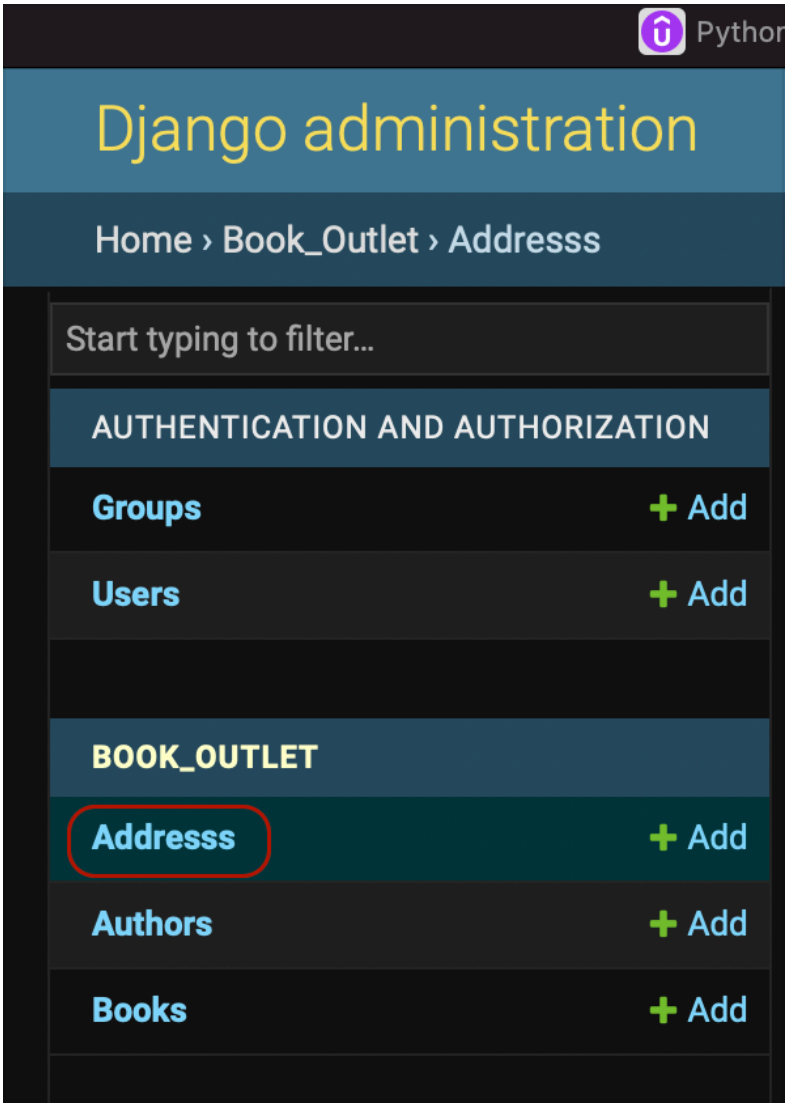
Creating address and assigning them to author.

```python
class Address(models.Model):
    street = models.CharField(max_length=80)
    postal_code = models.CharField(max_length=5)
    city = models.CharField(max_length=50)

class Author(models.Model):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    address = models.OneToOneField(Address, on_delete=models.CASCADE) # setup a one to one relation
```

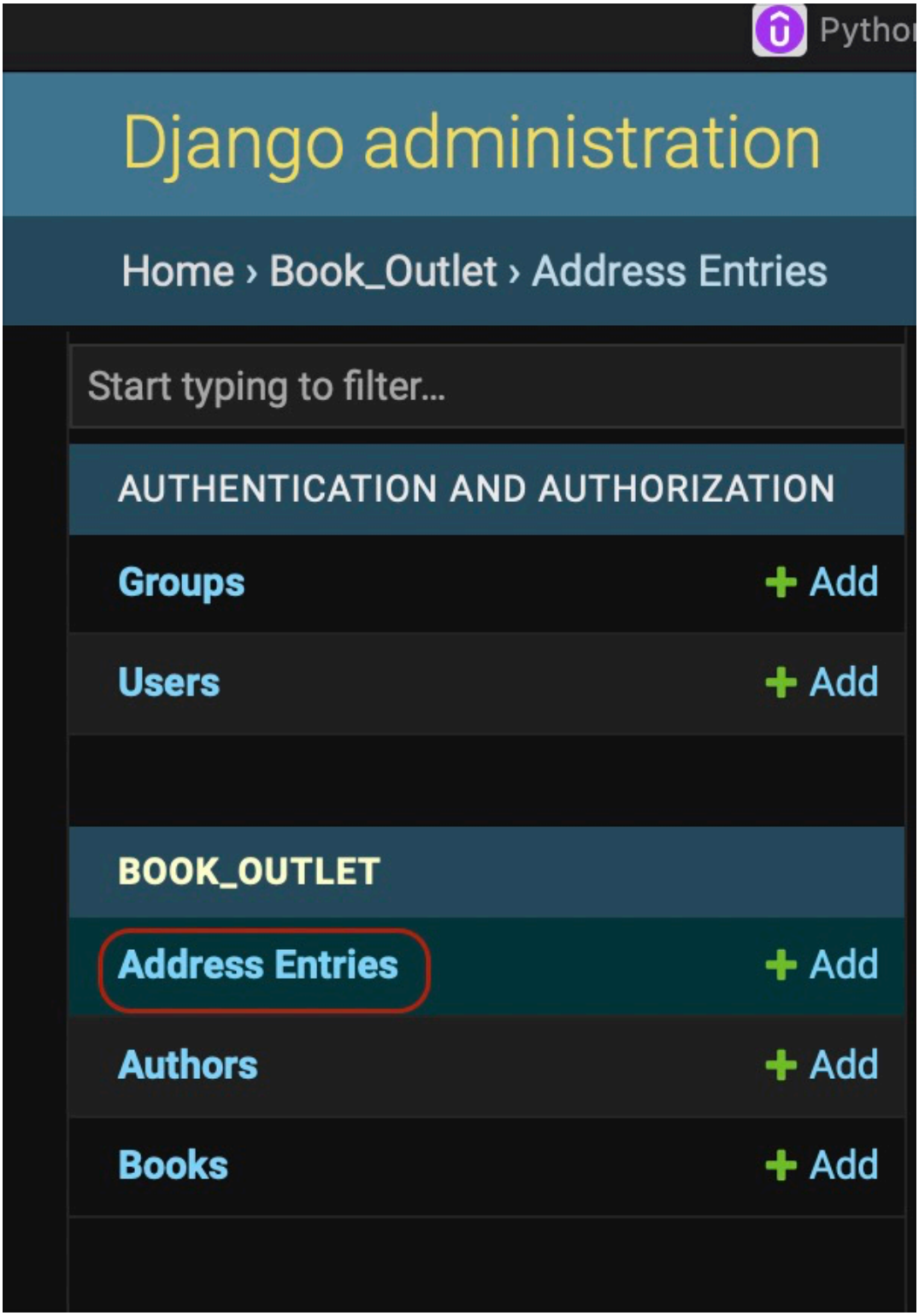
```
>>> from book_outlet.models import Author, Address, Book
>>> addr1 = Address(street="some street", postal_code="some postal", city="some city")
>>> addr2 = Address(street="another street", postal_code="12345", city="New York")
>>> addr1.save()
>>> addr2.save()
>>> Author.objects.all()
<QuerySet [<Author: J.K. Rowling>]>
>>> jkr = Author.objects.get(first_name="J.K.")
>>> jkr.address
>>> jkr.address = addr1
>>> jkr.save()
>>>
```

Change the model presentation in the admin section:

From this:


Pythor
Django administration
Home › Book_Outlet › Addresss
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
BOOK_OUTLET
Addresss + Add
Authors + Add
Books + Add

To this:



By creating a nested loop with the name “Meta”:

admin.py models.py ●

book_outline > models.py > Address > Meta

```python
from django.forms import CharField, IntegerField
from django.core.validators import MinValueValidator, Max
from django.urls import reverse
from django.utils.text import slugify

# Create your models here.


class Address(models.Model):
    street = models.CharField(max_length=80)
    postal_code = models.CharField(max_length=5)
    city = models.CharField(max_length=50)

    def __str__(self):
        return f"{self.street}, {self.postal_code}, {self

    class Meta:
        # how model output should look in the admin side
        verbose_name_plural = "Address Entries"
```

Setting-up many-to-many:

```python
class Country(models.Model):
    name = models.CharField(max_length=80)
    code = models.CharField(max_length=2)
```

```python
class Book(models.Model):
    # id  - comes automatically, but can be modified if needed
    # by using id = models.AutoField()
    title = models.CharField(max_length=50)
    rating = models.IntegerField(validators=[
        MinValueValidator(1),
        MaxValueValidator(5),
    ])
    # Saving a pointer in the "author" entry to an entry in the "Author" tabl
    # Setting up a realtion between "Book" and "Author"
    author = models.ForeignKey(Author, on_delete=models.CASCADE, null=True)
    is_bestselling = models.BooleanField(default=False)
    # Harry Poter 1 => harry-poter-1
    slug = models.SlugField(default="", null=False,
                            blank=True, db_index=True)
    published_countries = models.ManyToManyField(Country)
```

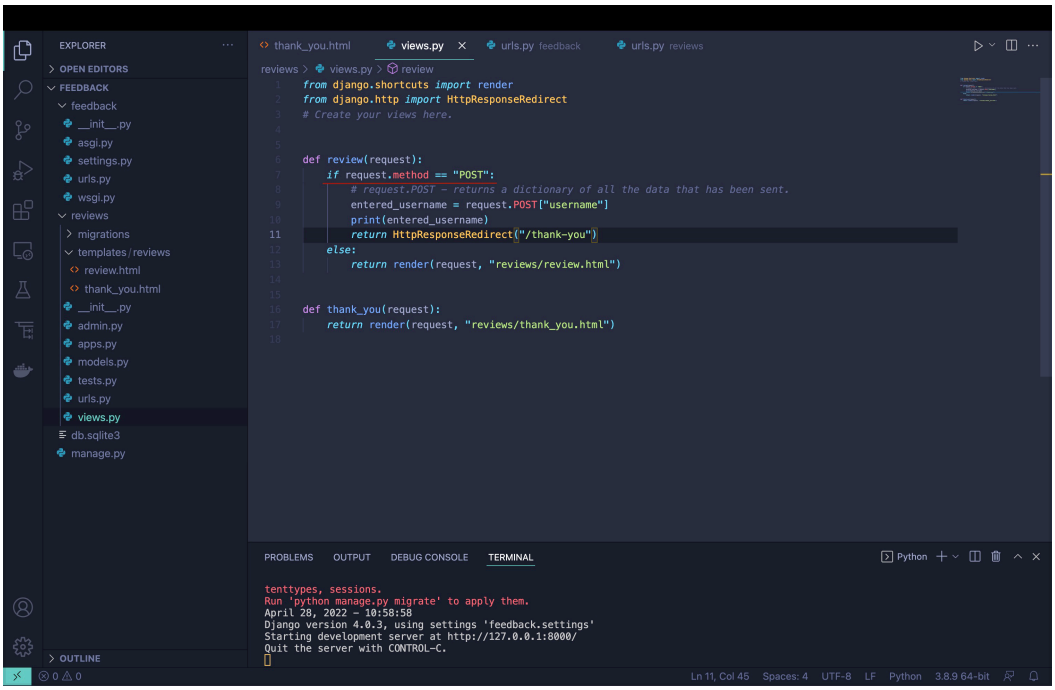
```
>>> germany = Country(name="Germany", code="DE")
>>> hp1.published_countries.add(germany)
```

Circular Relations & Lazy Relations:
To see that go to right tab of the course and read about it.

Fetching Posts for Starting Page:

Section 10 - Forms:
CSRF Protection:
Django provides a csrf token to check post requests that comes in to the server.
We want to use this token in order to validate that the data we getting from users, really came from our website.

Handling Form Submission & Extracting Data:



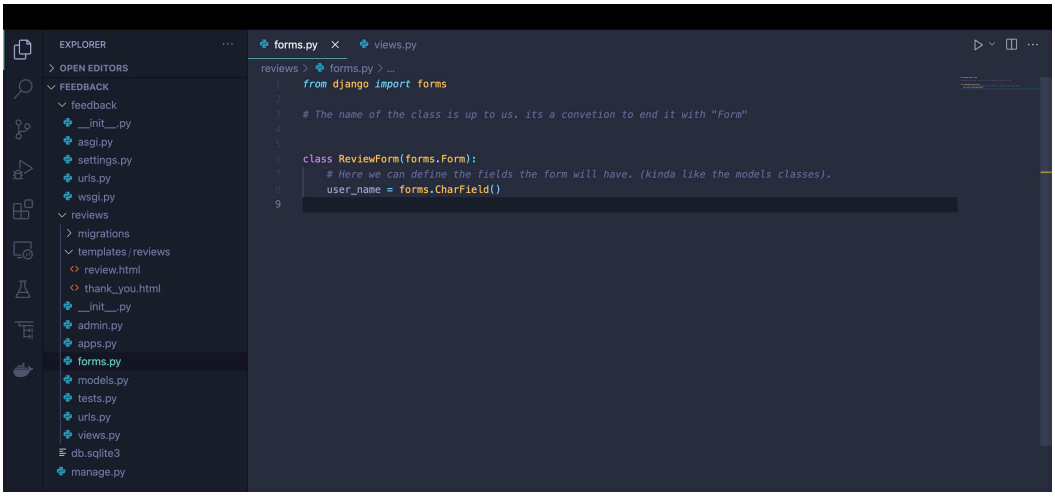
We can also validate our data:

```python
def review(request):
    if request.method == "POST":
        # request.POST - returns a dictionary of all the data that has been sent.
        entered_username = request.POST["username"]
        if entered_username == "":
            return render(request, "reviews/review.html", {
                "has_error": True,
            })
        print(entered_username)
        return HttpResponseRedirect("/thank-you")
    else:
        return render(request, "reviews/review.html", {
            "has_error": False,
        })
```

```
thank_you.html   review.html   views.py   urls.py feedback   urls.
reviews > templates > reviews > review.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Your Review</title>
</head>
<body>
    <form method="POST">
        {% csrf_token %}
        {% if has_error %}
        <p>This form is invalid - Please enter a vild user name!</p>
        {% endif %}
        <label for="username">Your name</label>
        <input type="text" id="username" name="username">
        <button>Send</button>
    </form>
</body>
</html>
```

But its not the way - we do not check data manually.

Using the Django Form Class:



```python
from .forms import ReviewForm
# Create your views here.


def review(request):
    # if request.method == "POST":
    #     # request.POST - returns a dictionary of all the
    #     entered_username = request.POST["username"]
    #     if entered_username == "":
    #         return render(request, "reviews/review.html"
    #             "has_error": True,
    #         })
    #     print(entered_username)
    #     return HttpResponseRedirect("/thank-you")
    form = ReviewForm()

    return render(request, "reviews/review.html", {
            "form": form,
        })
```

Presenting the form in the relevant template.

forms.py views.py review.html ×

reviews > templates > reviews > review.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Your Review</title>
</head>
<body>
    <form method="POST">
        {% csrf_token %}
        {{form}}
        <button>Send</button>
    </form>
</body>
</html>
```

forms.py views.py review.html

reviews > views.py > review

```python
from django.shortcuts import render
from django.http import HttpResponseRedirect
from .forms import ReviewForm
# Create your views here.


def review(request):
    if request.method == "POST":
        form = ReviewForm(request.POST)
        # Validate the input, return True if ok False otherwise. also, auto fill a field name "cleaned_data"
        if form.is_valid():
            print(form.cleaned_data)
            return HttpResponseRedirect("/thank-you")
    form = ReviewForm()
    return render(request, "reviews/review.html", {
        "form": form,
    })

```

Customizing the Form Controls:

```python
class ReviewForm(forms.Form):
    # Here we can define the fields the form will have. (kinda like the models classes).
    # label - the text besides the input (a label with text we decide)
    # max_length - the max length of data the user can enter into this field
    # error_messages - Dictionary where the keys are the error names django knows and values are the error messages
    # we want to present to the user.
    user_name = forms.CharField(
        label="Remez19 is the best gamer", max_length=20, error_messages={
            "required": "Dame you left the field empty!"
        })
```

(parameter) max_length: int | None

Customizing the Rendered HTML:

forms.py views.py review.html

reviews > templates > reviews > review.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Your Review</title>
</head>
<body>
    <form method="POST">
        {% csrf_token %}
        {{ form.user_name.label_tag }}
        {{ form.user_name }}
        {{form.user_name.errors}}
        <button>Send</button>
    </form>
</body>
</html>
```

Adding More Form Controls:

```python
class ReviewForm(forms.Form):
    # Here we can define the fields the form will have. (kinda like the models classes).
    # label - the text besides the input (a label with text we decide)
    # max_length - the max length of data the user can enter into this field
    # error_messages - Dictionary where the keys are the error names django knows and values
    # we want to present to the user.
    user_name = forms.CharField(
        label="User Name:", max_length=20, error_messages={
            "required": "Dame you left the field empty!"
        })
    review_text = forms.CharField(
        label="Your Feedback", widget=forms.Textarea, max_length=200)
    rating = forms.IntegerField(label="Your Rating", min_value=1, max_value=5)
```

Looping over the form fields:

forms.py views.py review.html styles.css

reviews > templates > reviews > review.html

```html
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Your Review</title>
    <link rel="stylesheet" href="{% static 'reviews/styles.css' %}">
</head>
<body>
    <form method="POST">
        {% csrf_token %}
        {% for field in form %}
        <div class="form-control {% if field.errors%} errors{% endif %}">
            {{ field.label_tag }}
            {{ field }}
            {{field.errors}}
        </div>
        {% endfor %}
        <button>Send</button>
    </form>
</body>
</html>
```

Storing Form Data in a Database:

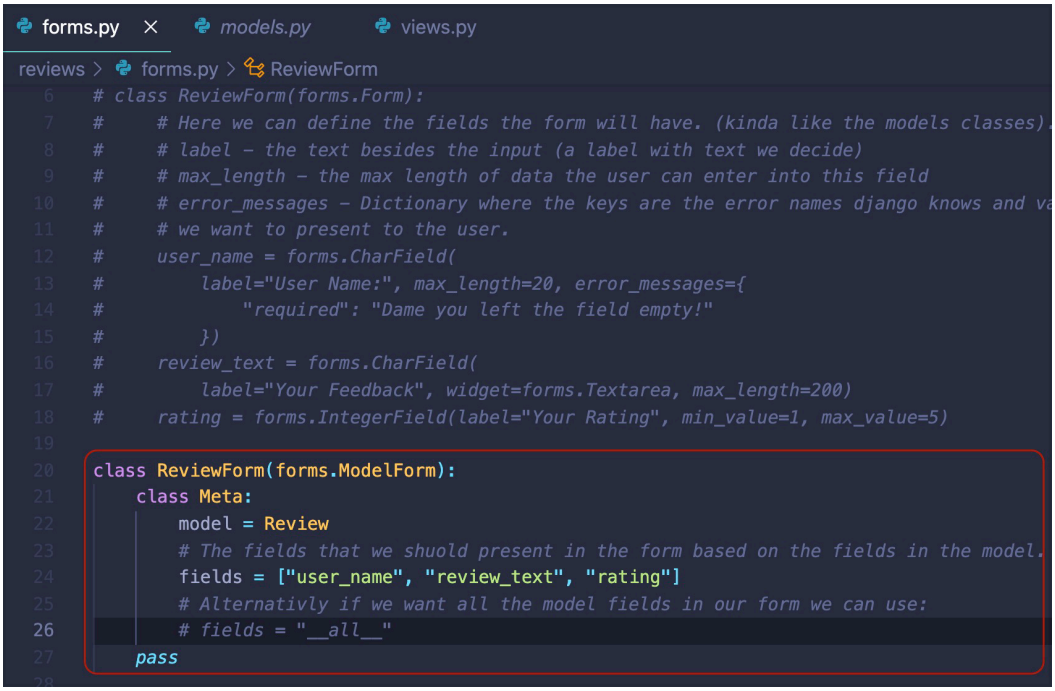
forms.py views.py models.py

reviews > models.py > ...

```python
from django.db import models

# Create your models here.


class Review(models.Model):
    user_name = models.CharField(max_length=100)
    erview_text = models.TextField()
    rating = models.IntegerField()

```

```python
def review(request):
    if request.method == "POST":
        form = ReviewForm(request.POST)
        # Validate the input, return True if ok False otherwise
        if form.is_valid():
            review = Review(
                user_name=form.cleaned_data["user_name"],
                review_text=form.cleaned_data["review_text"],
                rating=form.cleaned_data["rating"])
            review.save()
            return HttpResponseRedirect("/thank-you")
    else:
        form = ReviewForm()
    return render(request, "reviews/review.html", {
        "form": form,
    })


def thank_you(request):
    return render(request, "reviews/thank_you.html")
```

Introducing Modelforms:



Configuring the Modelform:

Saving Data with a Modelform:

```python
def review(request):
    if request.method == "POST":
        form = ReviewForm(request.POST)
        # Validate the input, return True if ok False o
        if form.is_valid():
            form.save()
            return HttpResponseRedirect("/thank-you")
    else:
        form = ReviewForm()
    return render(request, "reviews/review.html", {
        "form": form,
    })
```

Class Based Views:

forms.py models.py views.py

reviews > views.py > ...

```python
from django.views import View
# Create your views here.


class ReviewView(View):

    # Because we extened the View class we get access to this two functions.
    # We no longer need to check if its an "GET" or a "POST" request.
    def get(self, request):
        form = ReviewForm()
        return render(request, "reviews/review.html", {
            "form": form,
        })

    def post(self, request):
        form = ReviewForm(request.POST)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect("/thank-you")
        return render(request, "reviews/review.html", {
            "form": form,
        })
```


forms.py
views.py
urls.py
reviews > urls.py > ...
from django.urls import path
from . import views
urlpatterns = [
path("", views.ReviewView.as_view()),
path("thank-you", views.thank_you)
]