Python

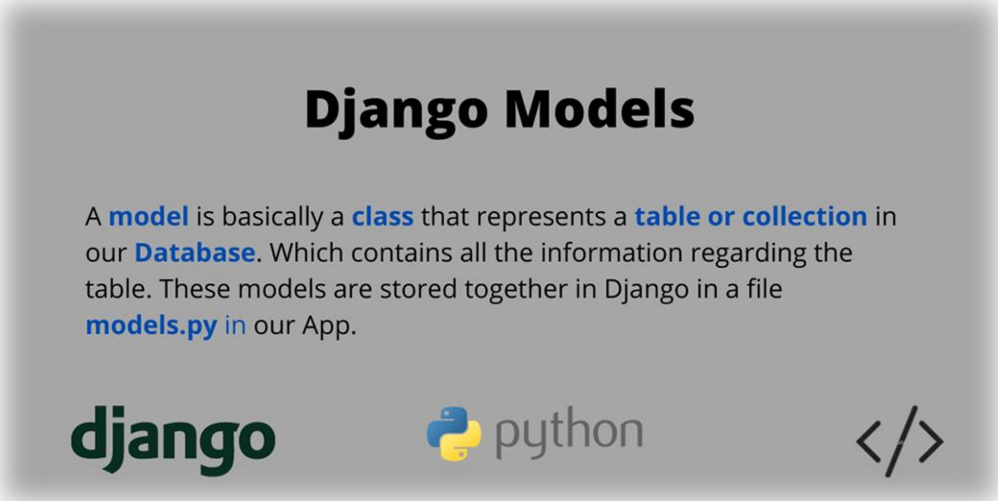
Full stack Skills Bootcamp

Introduction to Django Models

What is a Django Model ?

Django models define the structure of your data and map it to a database.

- Each model is a Python class that represents a database table.
- Models allow you to handle database operations using Python code, avoiding SQL queries.



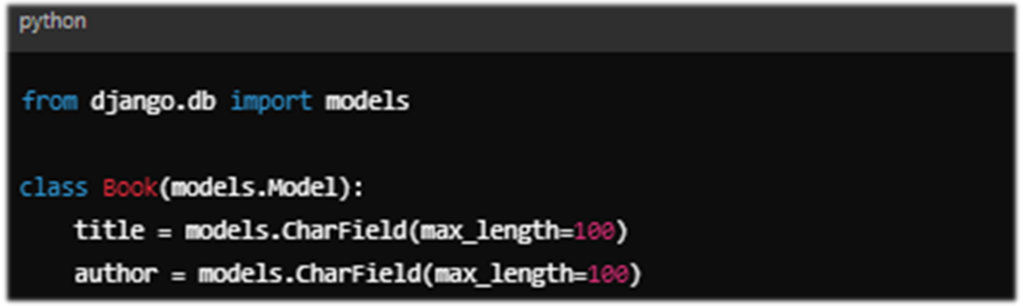
Django Models

A **model** is basically a **class** that represents a **table or collection** in our **Database**. Which contains all the information regarding the table. These models are stored together in Django in a file **models.py** in our App.

django
python
</>

Defining a Model Class

- A model class inherits from models.Model and defines fields as class attributes.
- Django provides various field types to represent data (e.g., CharField, IntegerField, DateField).

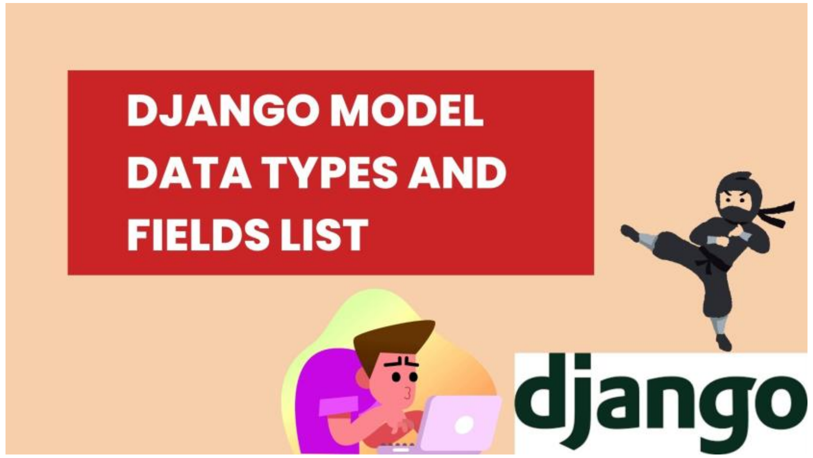


```python
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.CharField(max_length=100)
```

Field Types in Django

Django provides a variety of field types to define your data:

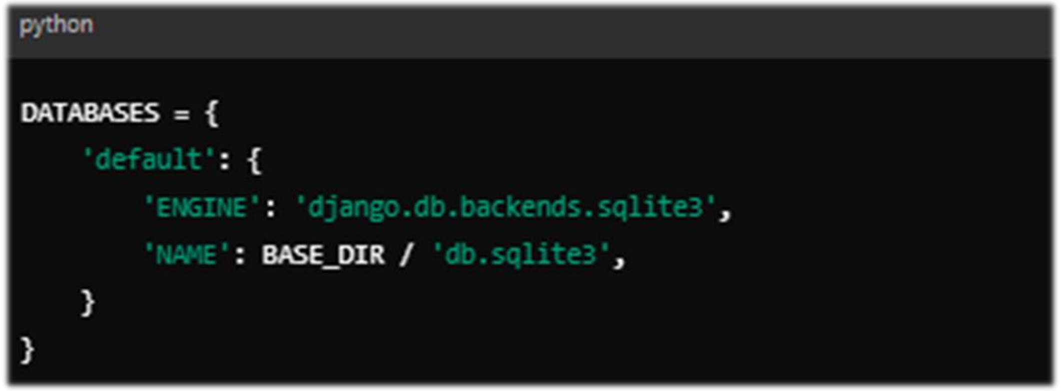
- **CharField**: For short text like names or titles.
- **TextField**: For longer text.
- **IntegerField**: For numeric values.
- **DateField**: For dates.
- **BooleanField**: For true/false values.



Each field comes with specific options like max_length, default, blank, etc.

Default Database Configuration

- Django uses SQLite as its default database.
- SQLite is a lightweight, file-based database that's easy to set up for development.
- It requires no additional configuration and works out of the box.



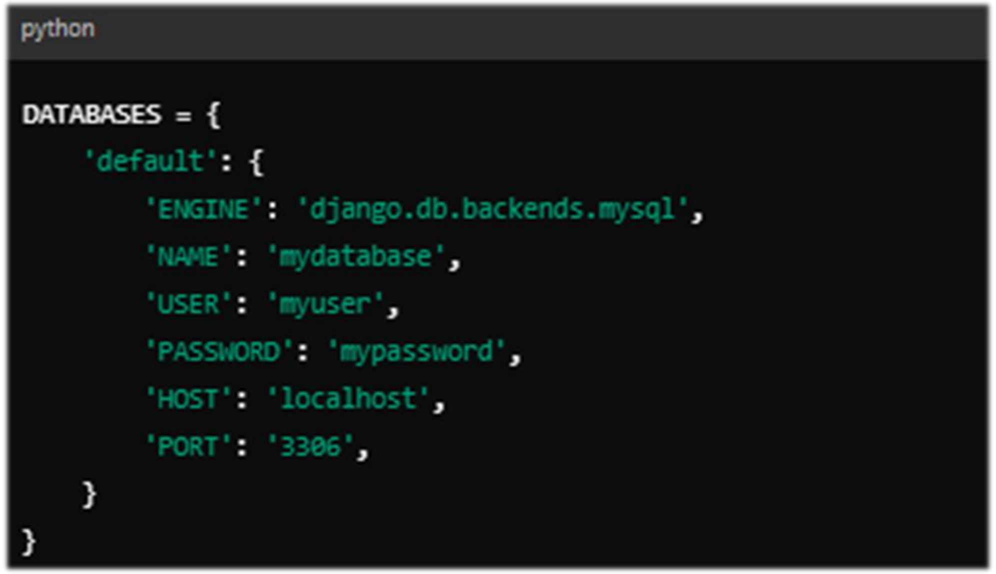
```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Step8Up Academy

Other Database Configurations

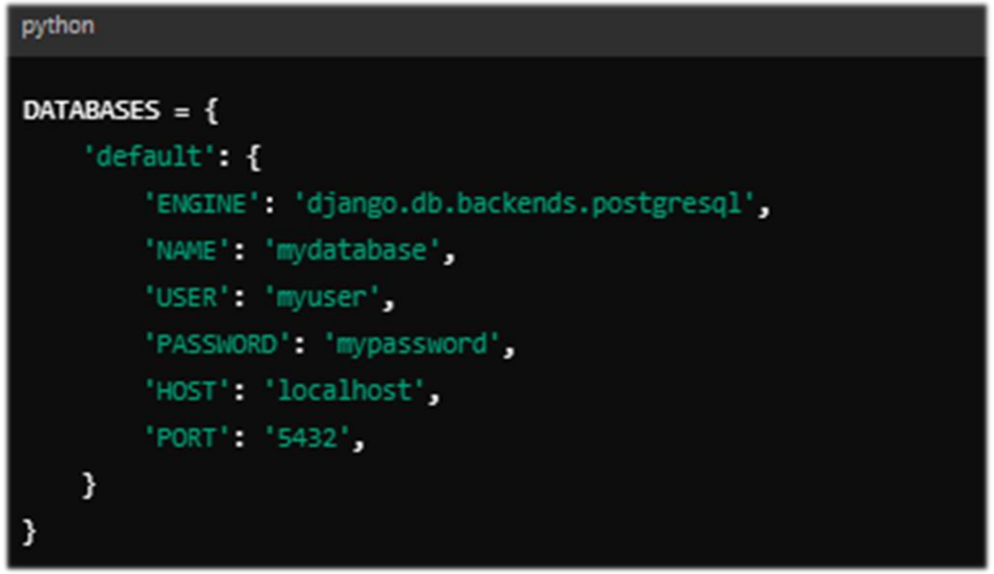
- Django supports other databases like MySQL and PostgreSQL for production use.
- To configure a different database, update the DATABASES setting in settings.py.

For MySQL:



```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mydatabase',
        'USER': 'myuser',
        'PASSWORD': 'mypassword',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

For PostgreSQL:



```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydatabase',
        'USER': 'myuser',
        'PASSWORD': 'mypassword',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Running Migrations

Migrations are Django’s way of applying changes to your database schema.

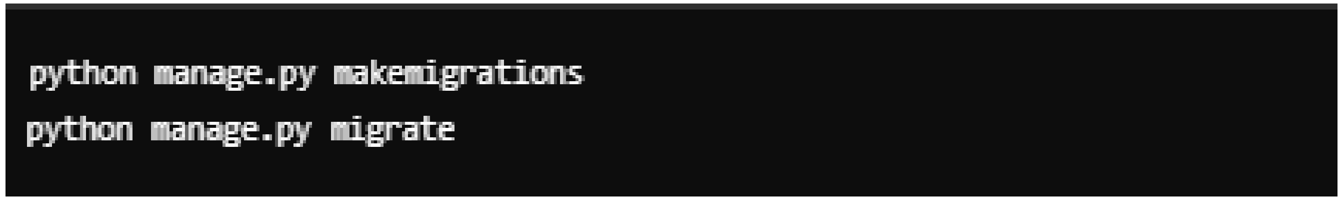
After modifying models, create migration files by running:

- python manage.py makemigrations

Apply the migrations to the database by running:

- python manage.py migrate

Migrations ensure your database structure is synchronized with your models.

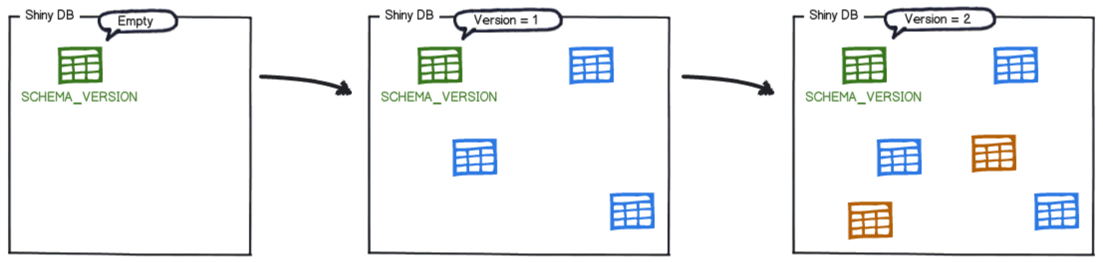


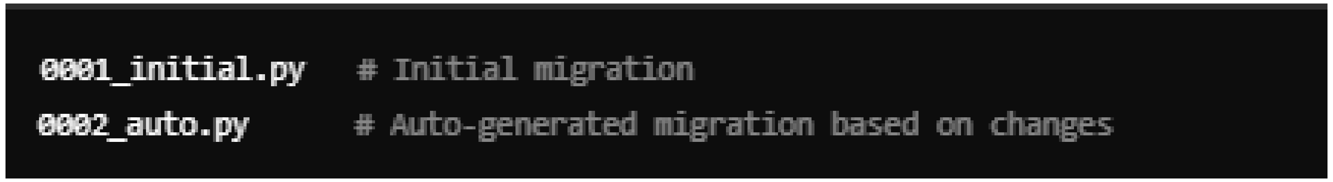
```
python manage.py makemigrations
python manage.py migrate
```

Tracking Database Changes with Migrations

Django automatically creates migration files that record changes to models.

- Django automatically creates migration files that record changes to models.
- Migration files are stored in your app’s migrations/ directory.
- These files allow you to track changes over time and apply them in a controlled manner.

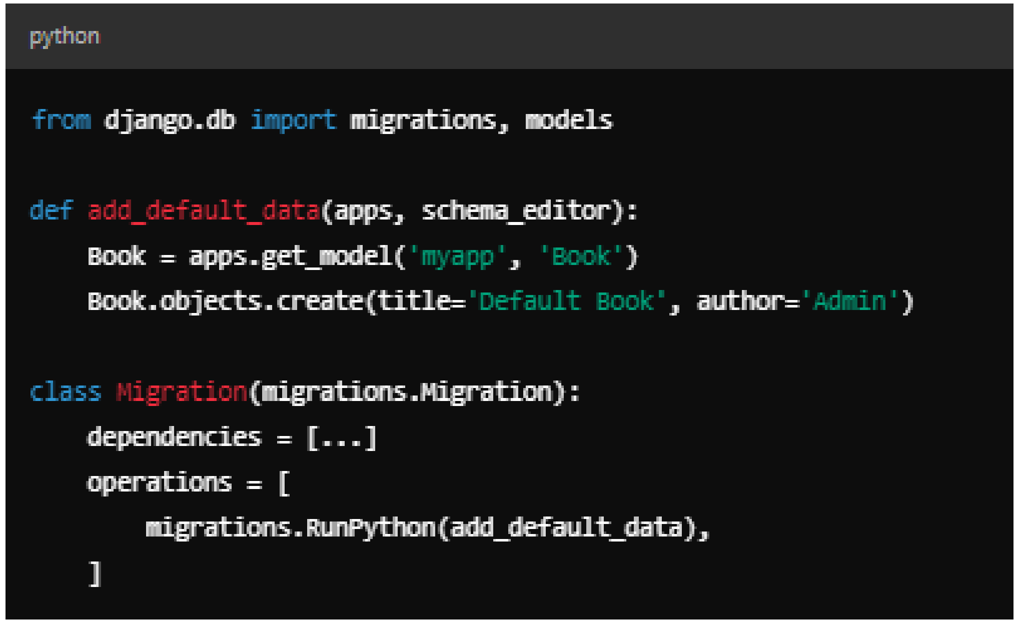




```
0001_initial.py    # Initial migration
0002_auto.py       # Auto-generated migration based on changes
```

Customizing Migrations

- You can write custom migrations to perform advanced operations (e.g., renaming columns, altering data).
- Django provides a RunPython method that lets you execute Python code during migrations.



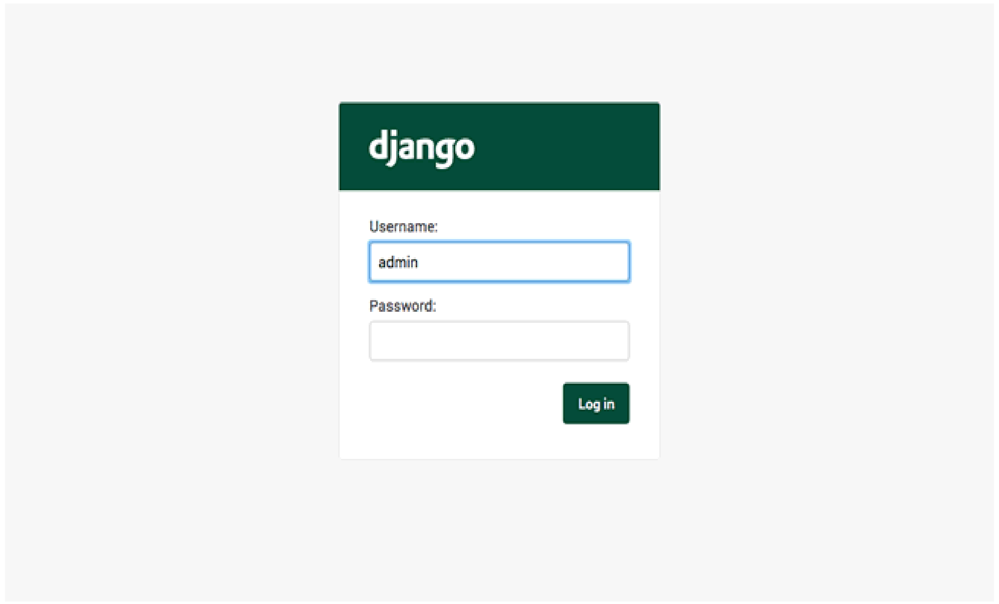
```python
from django.db import migrations, models

def add_default_data(apps, schema_editor):
    Book = apps.get_model('myapp', 'Book')
    Book.objects.create(title='Default Book', author='Admin')

class Migration(migrations.Migration):
    dependencies = [...]
    operations = [
        migrations.RunPython(add_default_data),
    ]
```

Using Django Admin with Models

- Django's admin interface provides a built-in way to manage your data through models.
- You can register your models with the admin site by adding them to admin.py.

```python
from django.contrib import admin
from .models import Book

admin.site.register(Book)
```

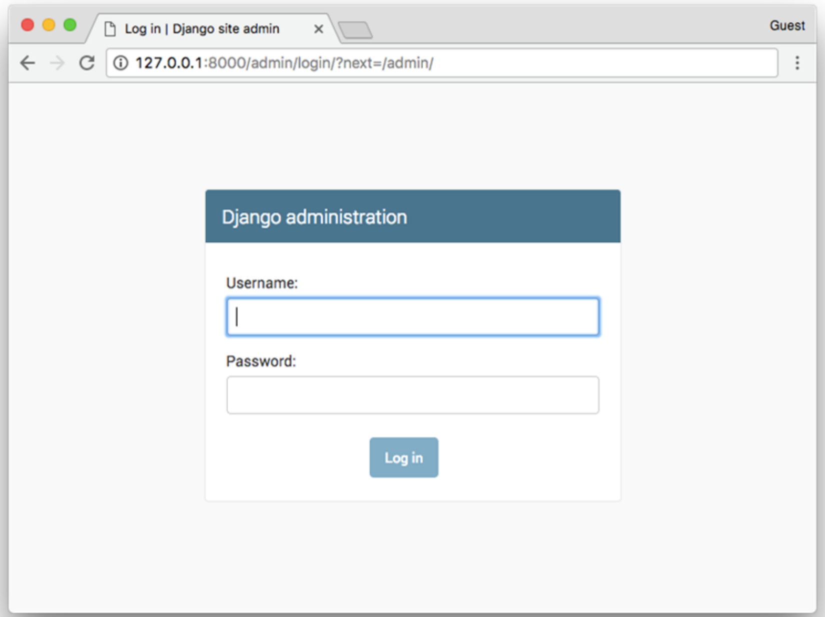
The admin interface allows you to view, add, edit, and delete records.



Introduction to Django Admin

Django's admin interface is a built-in feature that allows you to manage your application data.

- It provides a powerful and easy-to-use interface for viewing, adding, editing, and deleting data.
- The admin panel is automatically generated based on your models.


Log in | Django site admin
Guest
127.0.0.1:8000/admin/login/?next=/admin/
Django administration
Username:
Password:
Log in

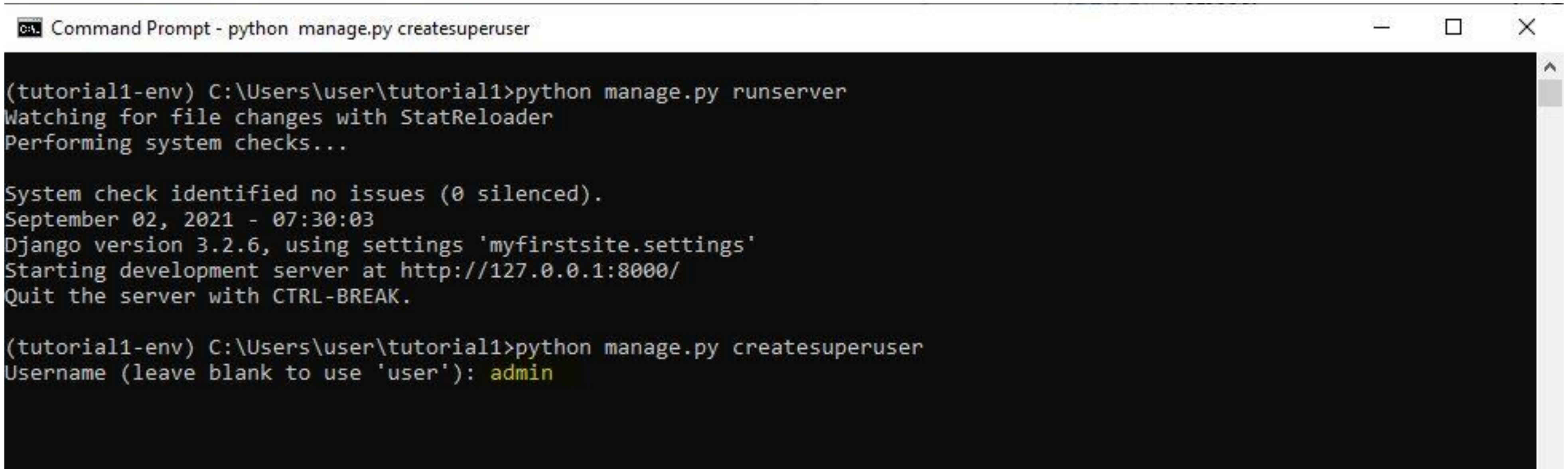
Accessing the Admin Panel

You can access the admin panel by going to http://localhost:8000/admin/ after starting your Django server.

- By default, Django includes a User and Group model in the admin panel.

You must create a superuser to log into the admin panel:

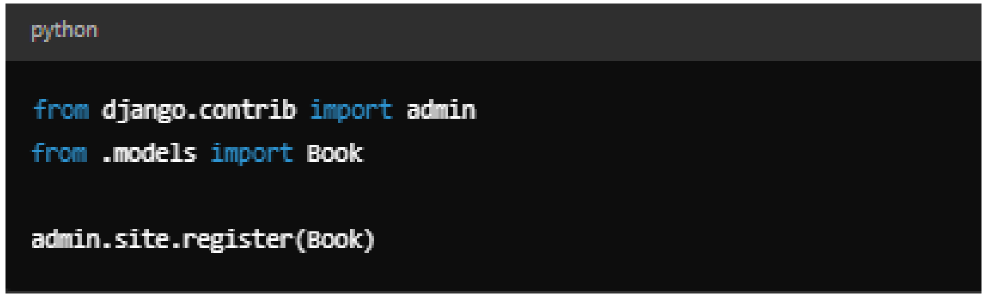
- Run “python manage.py createsuperuser” and follow the prompts to create a new admin account.



```
(tutorial1-env) C:\Users\user\tutorial1>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
September 02, 2021 - 07:30:03
Django version 3.2.6, using settings 'myfirstsite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

(tutorial1-env) C:\Users\user\tutorial1>python manage.py createsuperuser
Username (leave blank to use 'user'): admin
```

Registering Models in Admin

To make your models available in the admin panel, you need to register them in admin.py file.



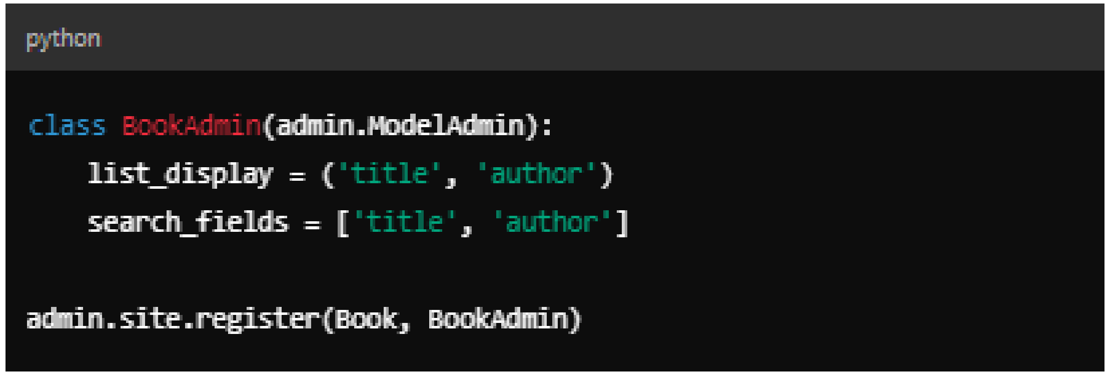
```python
from django.contrib import admin
from .models import Book

admin.site.register(Book)
```

Once registered, you can view and manage data related to that model in the admin interface

Customizing the Admin Interface

Django allows you to customize the admin interface by extending the ModelAdmin class.

- You can control how data is displayed by modifying fields, search options, filters, etc.
- This allows for a more tailored admin experience.


python

```python
class BookAdmin(admin.ModelAdmin):
    list_display = ('title', 'author')
    search_fields = ['title', 'author']

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

Step8Up Academy

Benefits of Django Admin

- Speed: Quickly manage data without writing SQL queries.
- Security: Comes with built-in authentication and authorization.
- Customization: Easily adapt the interface to your needs.
- Efficiency: Great for managing application data during development and in smaller production environments.