# Lab: ORM Introduction

This document defines the problems for the in-class lab for the [Python ORM course @ Software University](https://softuni.bg/trainings/4253/python-orm-october-2023).

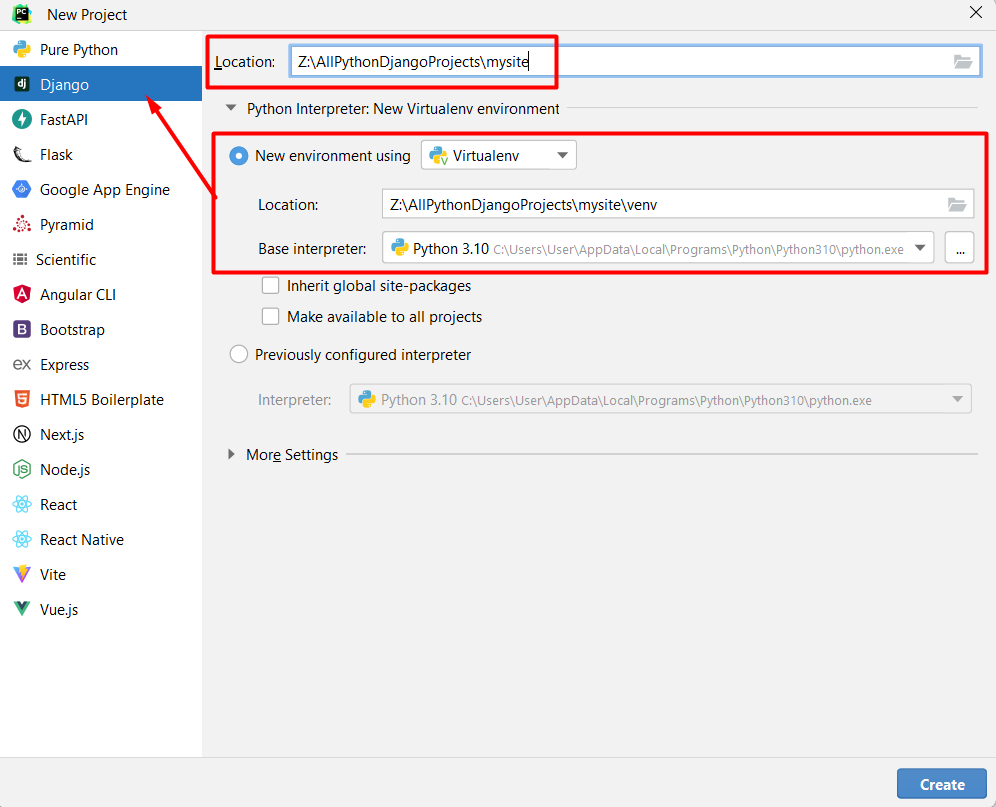
## Creating a Django Project

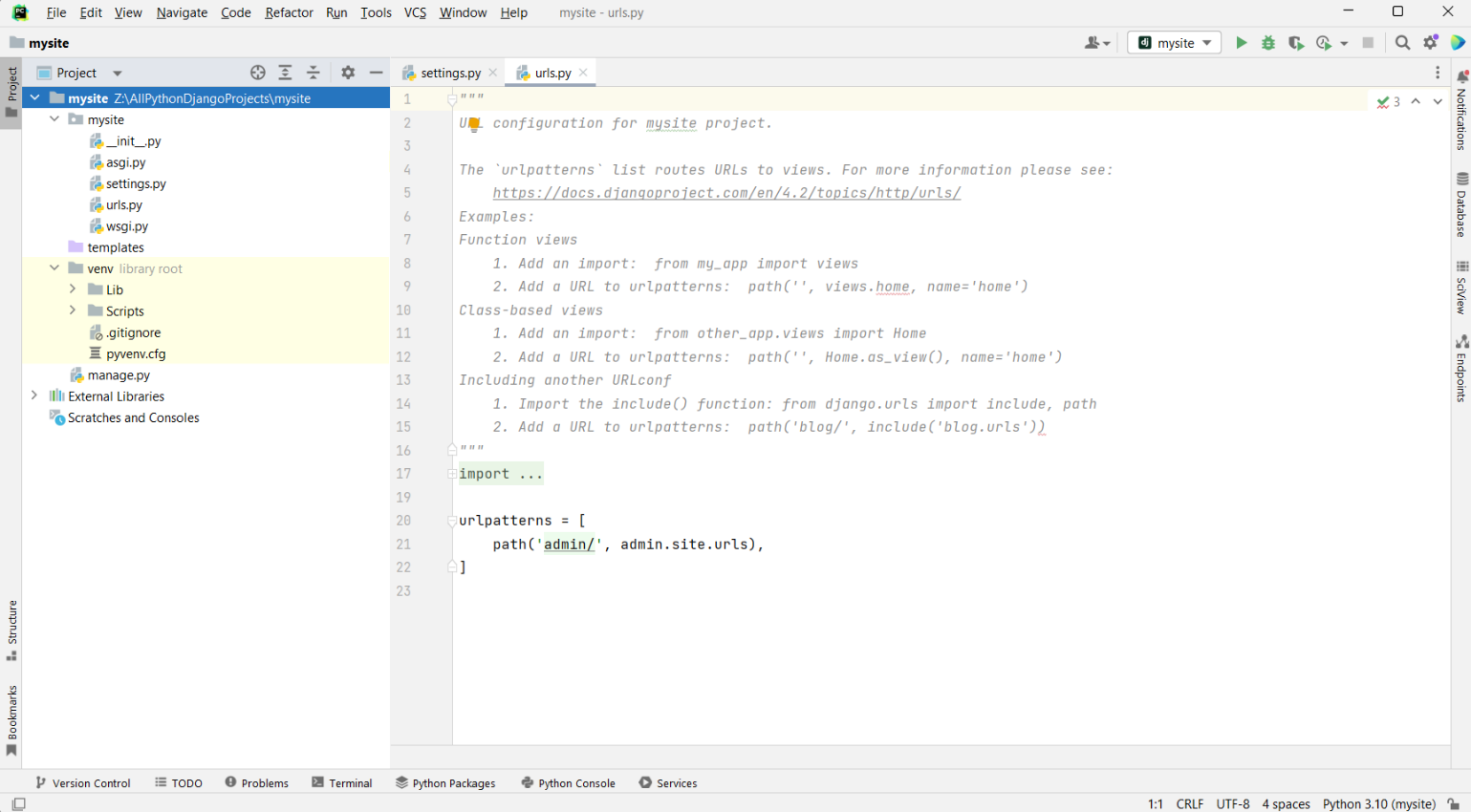
We are going to create our first Django project. A Django project is like the **blueprint for building a website** or web app. It **includes** all the **important** **instructions**, **tools**, and **parts** needed to create your online creation. It helps you **organize your code**, **manage website addresses**, **handle where your data is stored**, and do many other important tasks. Just like a foundation for a house, a Django project **gives you the basic structure** to make your website work and do different things.

To create the project, we will be using **PyCharm Professional** as our integrated development environment. Let us **launch the tool** and **navigate to File -> New Project**:

A screenshot of a computer

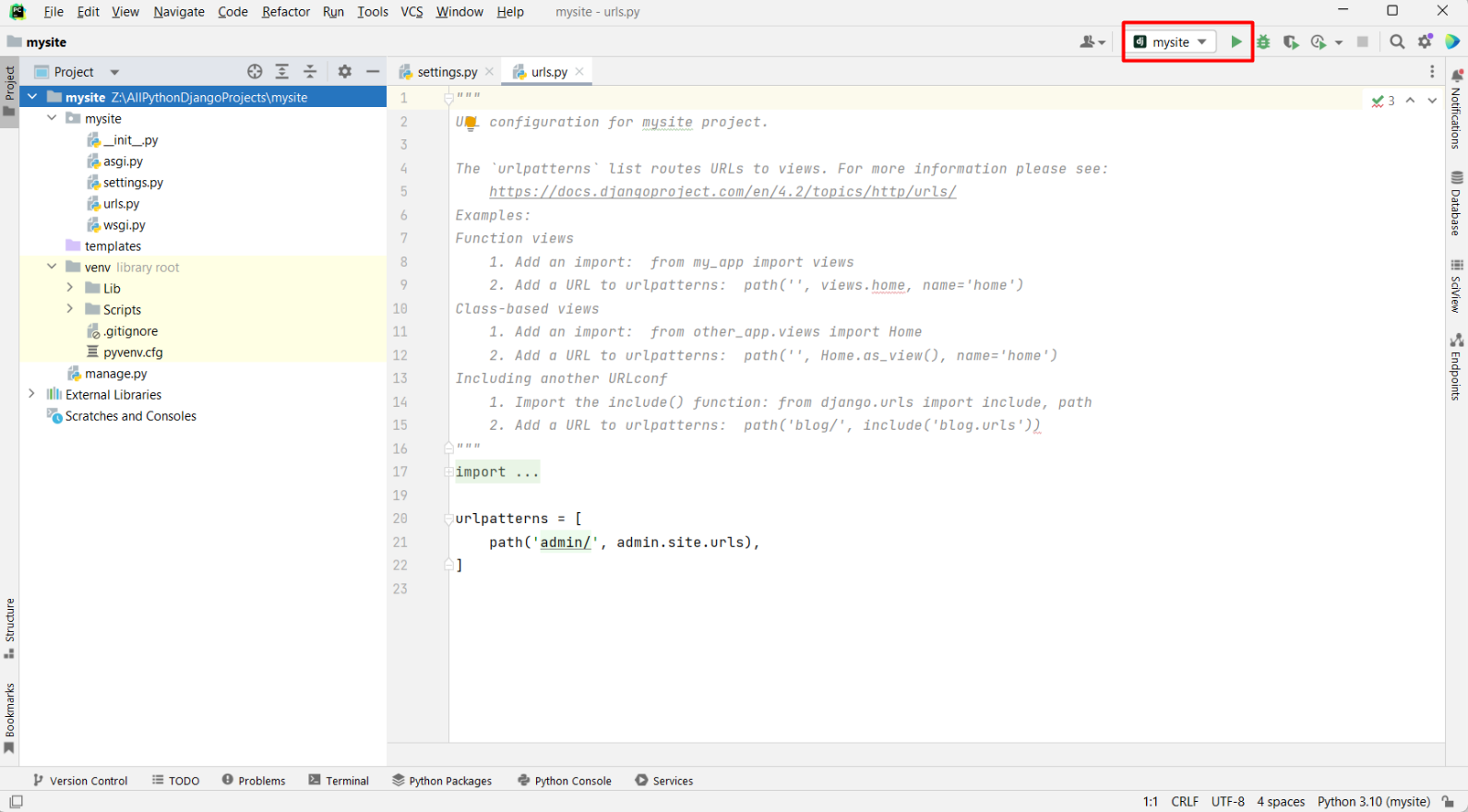
Description automatically generated

In the pop-up window, we will **choose to create a Django project**. **Configure the project's** **location** and **name**, ensuring the **virtual environment is set up** as well: 

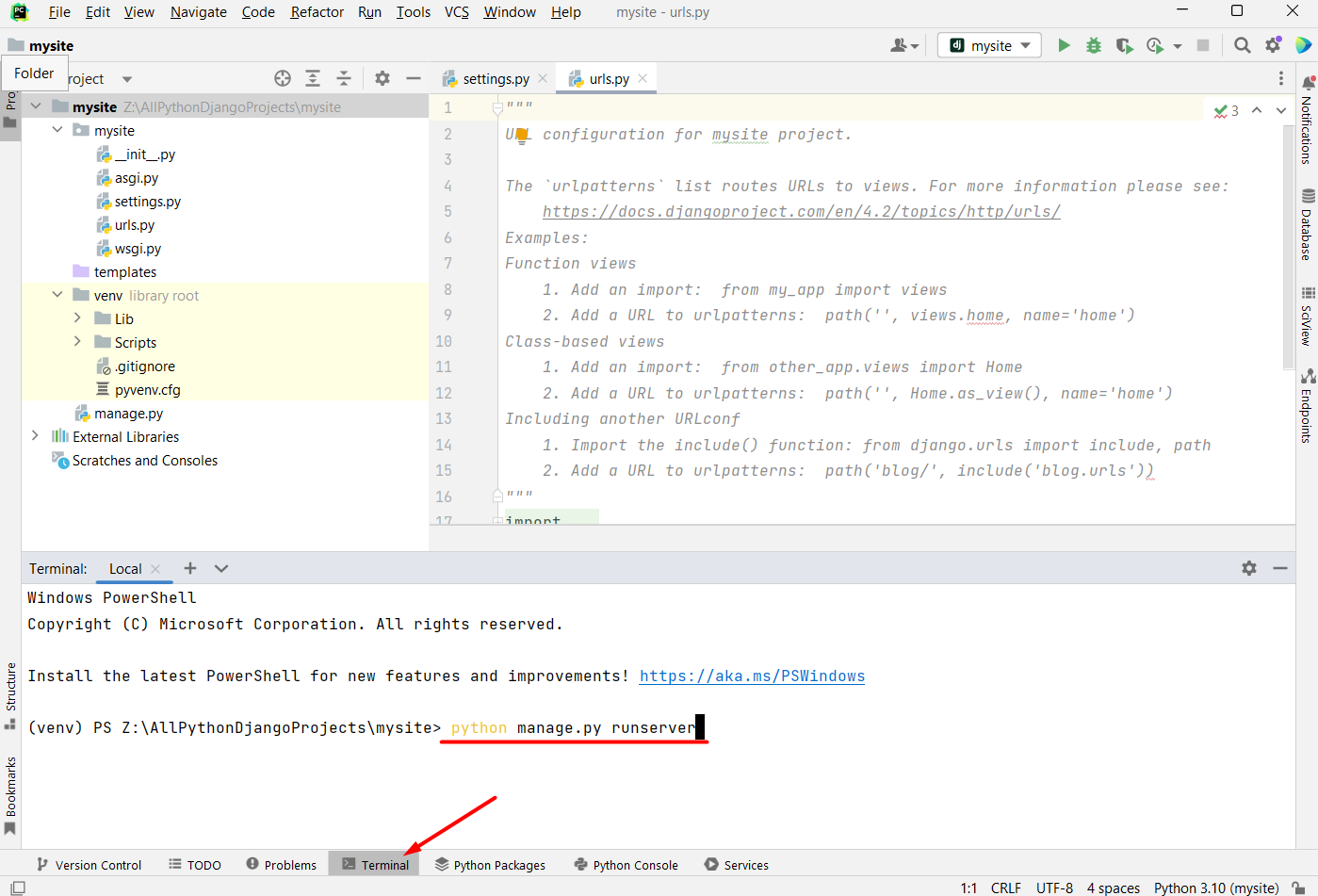
Click the **"Create"** **button** to initiate the project creation process: 

Next, let us **check if the project is launching correctly**. There are **3 ways of doing that:**

* We can **click on the** **PyCharm Run** **button**. Note: Ensure the Django project is selected:



* We can start the project using the **keyboard shortcut [Shift + F10]**
* We can **open the local terminal** and **type down the command** **"python manage.py runserver"**:



Either way, we should **see the** **following output on the command line**: 

**Click on the link** and **check** if you see the following content: A screenshot of a computer

Description automatically generated

## Creating a Django Application

Now, let us create our first Django application. In Django, an app is like a **building block within a larger website** or **web application**. It is a **self-contained unit** that contains all the **code** and **functionality** needed to **perform a specific task** or **provide a particular feature**. For example, an app can handle user authentication, manage blog posts, or process payments.

Let us **access the local terminal** again and **type down the command** **"python manage.py startapp tasks"**. This will **generate** the desired Django **application** **named "tasks"**:

A screenshot of a computer

Description automatically generated

Django will **automatically generate the essential components** for the app: A screenshot of a computer

Description automatically generated

To **integrate** the app **with the project**, add its **reference** to **the INSTALLED\_APPS list within the settings.py**:A screenshot of a computer

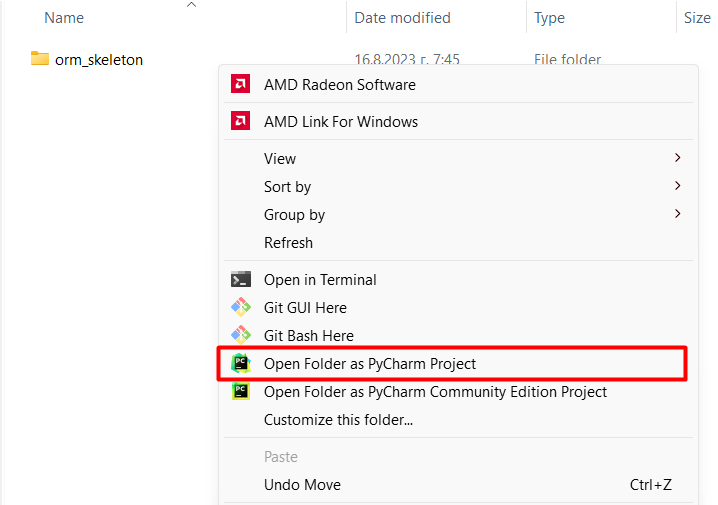
Description automatically generated

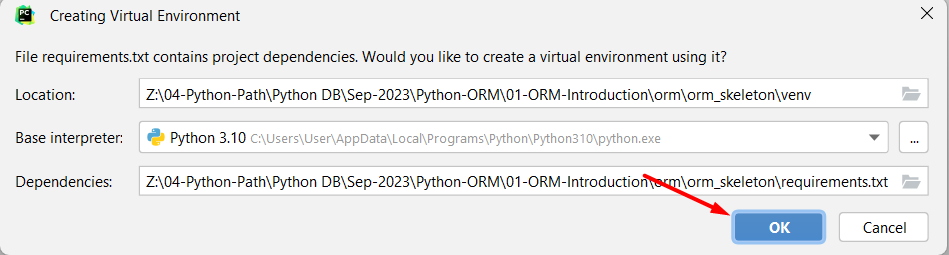
Note: If we chose to **move the app inside the root directory** of our Django project, we should **consider the difference** when integrating it:A screenshot of a computer

Description automatically generated

## Introduction to the Ready-to-Use Skeleton

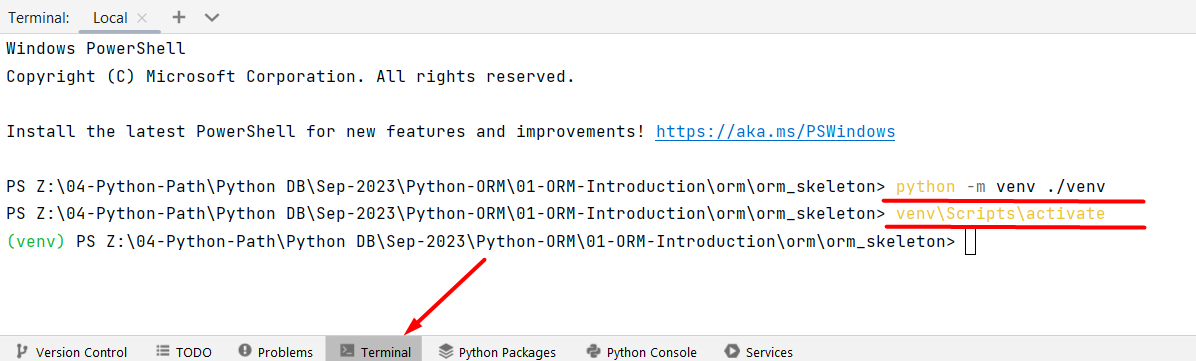
### 3.1 Opening the Project and Installing the Dependencies

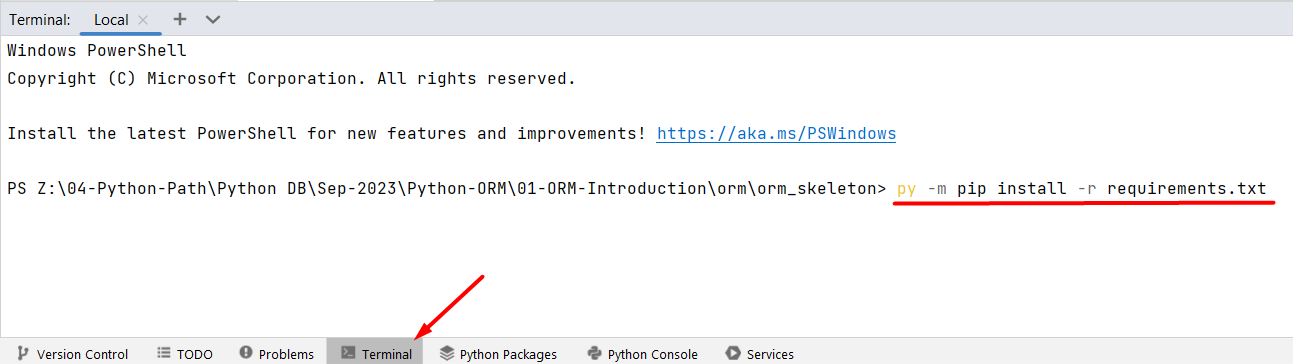
*For your convenience, we have prepared a ready-to-use skeleton for the needs of this course.* Let us access the skeleton. **Note: You can open the skeleton by right-clicking on the root folder if you have chosen "Update Context Menu / Add "Open Folder as Project" during PyCharm installation:**

As the skeleton is opened, we need to **ensure dependencies are met**. We can **accept** PyCharm's **suggested settings** for creating a virtual environment with the base interpreter and required dependencies. The **window should pop up** **automatically** after opening the project:

However, we can set it up **manually**. First, **choose your preferred version of Python for a base interpreter**: A screen shot of a computer

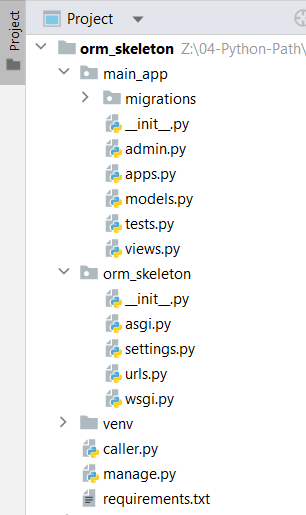
Description automatically generated

Afterward, we need to **set up a virtual environment** and **activate** **it**. This is done by **executing two commands**: **"python -m venv ./venv"** and **"venv\Scripts\activate"**. Note that the first command **creates a location** for the virtual environment of the project; we the **"./"** we indicate that this venv should be created in the **directory just above our current location**: 

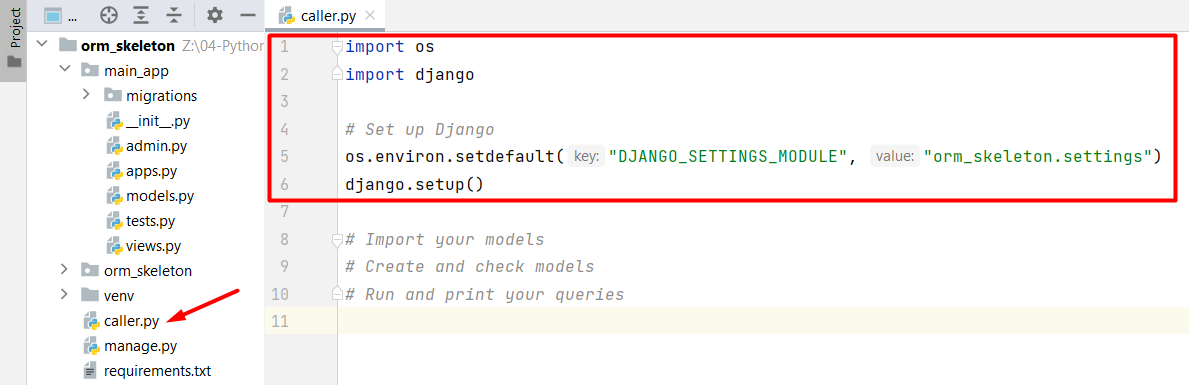
Once our virtual environment is ready, we proceed to **install the required dependencies** listed in the **requirements.txt** file. This is accomplished by **entering** the **command** **"py -m pip install -r requirements.txt"** in the terminal:  
We can **launch the development server** (by clicking on the **Run** **button**) and check if the project is functioning correctly.

### 3.2 Skeleton Structure

Like a typical Django project, the skeleton structure has a **root project directory** called **"orm\_skeleton"** and **one pre-configured** Django **application** directory called **"main\_app"**. **You must not change the name of the directories!**



Also, there's an **important file** named **"caller.py"** **included** in the project. It **sets up some necessary parts** of Django which you **MUST** **NOT** **erase**. Throughout this course, we'll use the **"caller.py"** file to **help us** **see** the **queries we run** using **Python code** more **clearly**. Here we will **practice working with queries** that we have learned during the previous course, but we will not use direct PostgreSQL code in the pgAdmin interface. Instead, we will **use Python code**. We will **create**, **read**, **update**, and **delete** database records (rows):



## Configuring a PostgreSQL Database

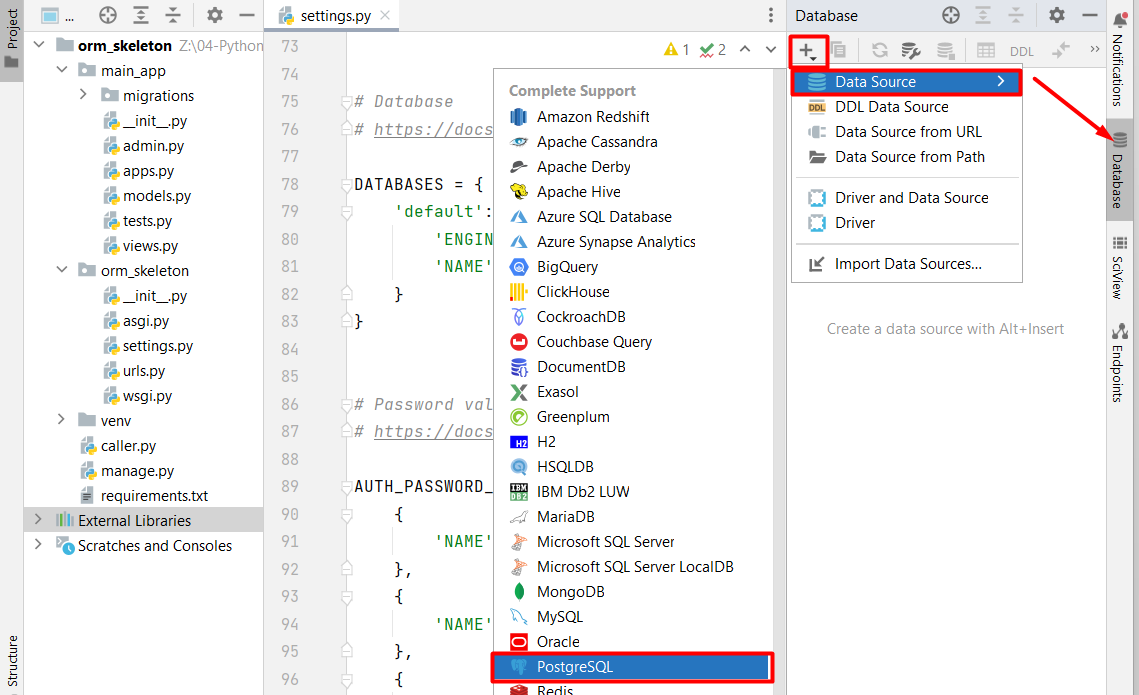
### 4.1. Create a New Database using PyCharm Tool Window

As you see, in **both** the conventional Django project and the provided skeleton **feature a database called** **sqlite3**:



It is a **database** that Django uses by default to **store** and **manage** **data** for the web application. **SQLite** is a lightweight, serverless, and self-contained **relational database management system**, and Django supports it as one of its database backends. While SQLite is a **great choice for small to medium-sized projects**, larger projects might require **more robust database systems** like **PostgreSQL**, **MySQL**, or **Oracle**.

So, now we are going to **configure a PostgreSQL database** in the **provided skeleton**. First, let us **access PostgreSQL**. Click on the **"Database"** tool located on the **PyCharm Tool Window**. Since no data sources are currently defined, **click** **on the** **"+" icon** (meaning "New") to **create one**:



A dialog window will pop up on the screen, where we can **add our credentials** - the **username** and **password** that we **use to connect to PostgreSQL**: A screenshot of a computer

Description automatically generated

As we have connected to the data source, we now need to **create a database** in which all the data from the current project will be saved. **Right-click** on the source and **choose to create a new database**:A screenshot of a computer

Description automatically generated

We will **name the database** **"my-site-database"**, but you are free to name it with your individual preferences: A screenshot of a computer

Description automatically generated

To see the changes made, **right-click on the data source** and **refresh the interface**: A screenshot of a computer

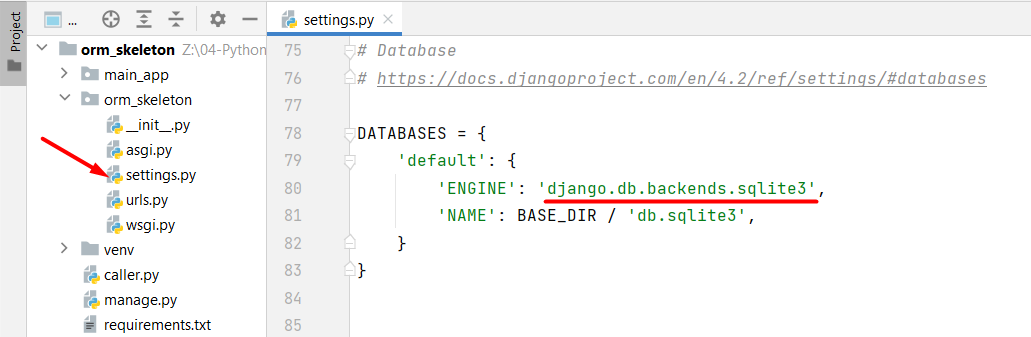
Description automatically generated

Now, we will be able to **see the newly created** **database**:

A screenshot of a computer

Description automatically generated

### 4.2. Configure the Database Settings

We are already aware that Django comes with a built-in database called SQLite by default. However, for our purposes, we want to **switch to using PostgreSQL**. To make this change, we will **go to the** **"settings.py"** file. Inside the **"DATABASES"** **section**, there's a **piece of code** **that** **shows the project is currently set to use SQLite**: 

We need to **modify that part**, so it **uses the PostgreSQL** **database** we have just created. **Add the following code**:

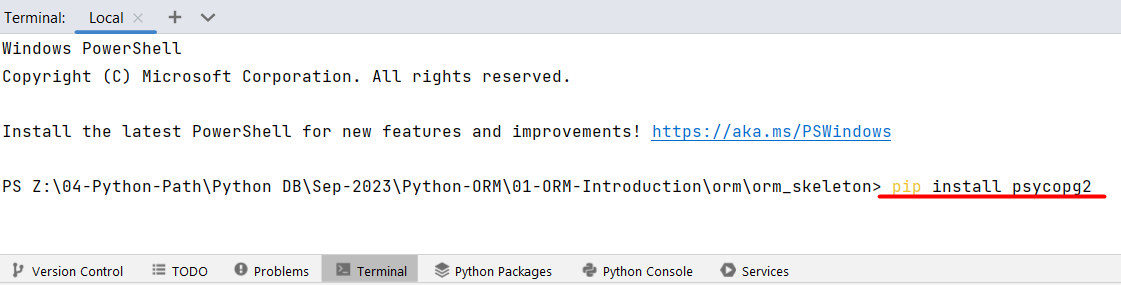
|  |
| --- |
| **DATABASES = {**  **'default': {**  **'ENGINE': 'django.db.backends.postgresql',**  **'NAME': 'my-site-database',**  **'USER': 'postgres-user',**  **'PASSWORD': 'postgres',**  **'HOST': 'localhost',**  **'PORT': '5432',**  **}**  **}** |

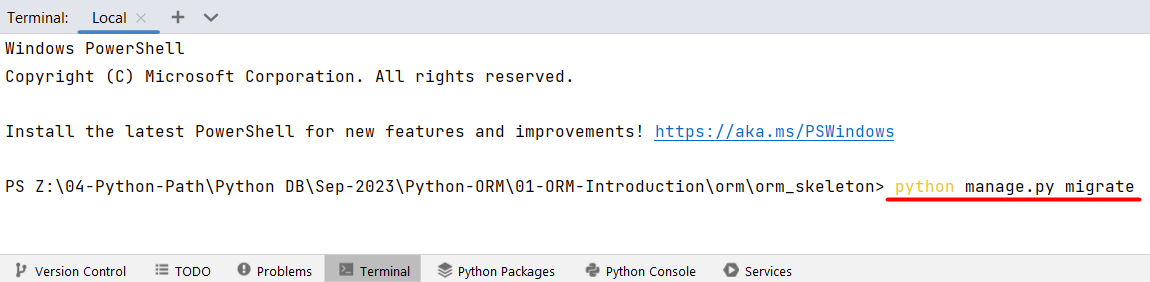
Let's **break down each key-value pair** in this configuration:

* **'ENGINE'**: Specifies the **database backend to use**. In this case, **'django.db.backends.postgresql'** indicates that the PostgreSQL database engine will be used. Note that this engine **needs the** **psycopg2** **adapter** to connect to PostgreSQL.
* **'NAME'**: This is the **name of the database** we want to connect to. In our case, it's set to **'my-site-database'**. This is where **our project's data will be stored**.
* **'USER'**: Specifies the **username** **to use** when connecting to the PostgreSQL database. Replace **'postgres-user'** with the actual username you will use.
* **'PASSWORD'**: Specifies the **password** **for the provided username**. Replace **'postgres'** with the actual password.
* **'HOST'**: Indicates the **host where the PostgreSQL database server is running**. In this case, it's set to **'localhost'**, meaning the database server is **on the same machine** as our Django project.
* **'PORT'**: Specifies the **port number on which the PostgreSQL database server is listening** - it is set to **'5432'** - the **default port** for PostgreSQL.

### 4.3. Install the Database Driver

Now, we can **install the PostgreSQL adapter psycopg2** with the command **"pip install psycopg2"** using the local terminal:

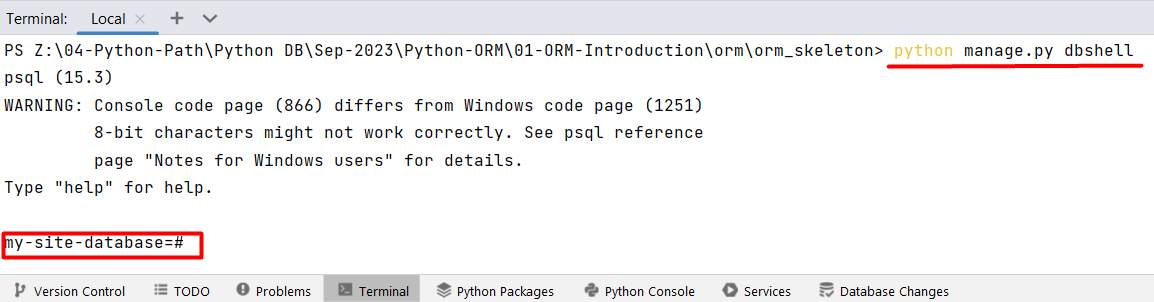


Finally, we are **ready to use the database** in our Django project. We can make a **fast check** by typing the command **"python manage.py migrate"** which will **create all the tables** that were predefined in the Django project: 

**Note: The command will be discussed in detail in the upcoming lessons. At the moment, you do not need to concern yourself with understanding it further.**

## Django DB Shell

### 5.1. Opening the Django dbshell

It is time to **dive into the Django dbshell** and practice our knowledge. **Open the terminal** and ty**pe down the command** which starts the dbshell **"python manage.py dbshell"**:

### 5.2. Receiving CommandError for Windows Users

If you are using Windows OS, you might encounter a common issue when you **first run the command**. You might see a **CommandError** saying **"You appear not to have the 'psql' program installed or on your path."**: A screenshot of a computer

Description automatically generated

To solve this problem, you will need to adjust your computer's settings.

First, let us **locate our PostgreSQL binary path**. For example, Windows users usually find it at **"C:\Program Files\PostgreSQL\15\bin"** (it can vary based on the PostgreSQL versions or where you have saved the installation files).

Next, **access the system settings** and search the **"Edit the system environment variables"** window. Afterward, **click on the "Environment Variables" button** in the **"Advanced"** **System Properties**:

A screenshot of a computer program

Description automatically generated

**Find and click on the** **"Path"** **system variables**, so we can **edit them**: A screenshot of a computer

Description automatically generated

Finally, we need to **add the path to the PostgreSQL installation files**. **Click on the "New" button and add it:** A screenshot of a computer

Description automatically generated

After making the addition, we just need to **confirm the changes** **by** **clicking on the** **"OK"** **button**:

A screenshot of a computer

Description automatically generated

**Note: We** **MUST** **restart the IDE in order to execute the command correctly**:

A screenshot of a computer

Description automatically generated

### 5.3. Receiving CommandError for Mac Users

If you are using Windows OS, you might encounter a common issue when you **first run the command**. You might see a **CommandError** saying **"You appear not to have the 'psql' program installed or on your path."**: A screenshot of a computer

Description automatically generated

To solve this problem, you will need to run the command **"brew install postgres"**. In case your machine does not have brew installed, you need to install it.

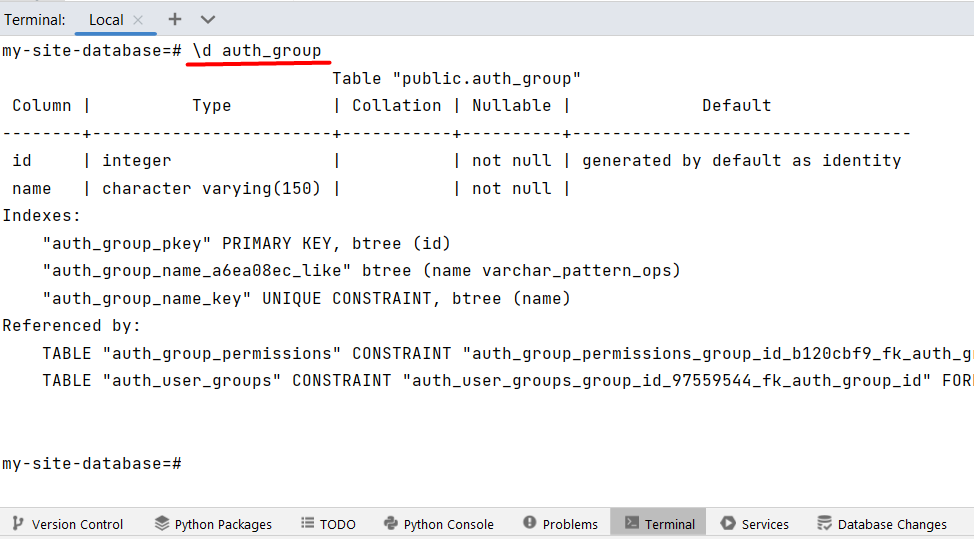
### 5.4. Working with the Django dbshell

Let begin by **displaying all the tables in our database** running the commands **"\dt"**:A screenshot of a computer

Description automatically generated

These are the **tables** **predefined within every Django project**. **Note: We will examine them closely in the next Django course.**

Moreover, we can **access information about a specific table** by executing the command **"\d <table\_name>"**. Let us take a closer look at the **"auth\_group"** **table**:



The fundamental aspect of using the **dbshell** is that we can **execute raw SQL queries** **directly** within our project's database **through the local terminal**. For example, we can **retrieve all the data from the initial table**: A screenshot of a computer

Description automatically generated

The **table appears empty** because we have **not added any records** to it yet.

For the purposes of this lab, we will **create a table** named **"people"** **using raw SQL code**. Note that **we can write the SQL statement on multiple lines**. As long as the statement is **not concluded** with a **";"** the **program treats** it as a **single instruction**:

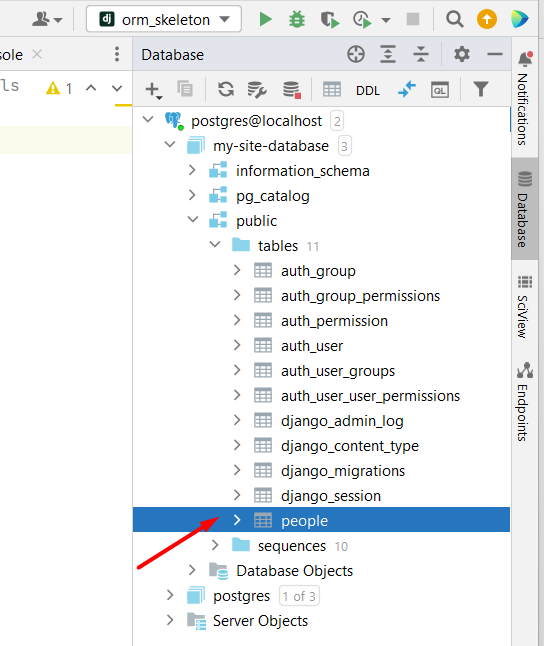
A screen shot of a computer

Description automatically generated

We can check if the **table is created properly** by typing the command **"\d people"**:A screenshot of a computer

Description automatically generated

Furthermore, the table is visible within the Database PyCharm Tool:



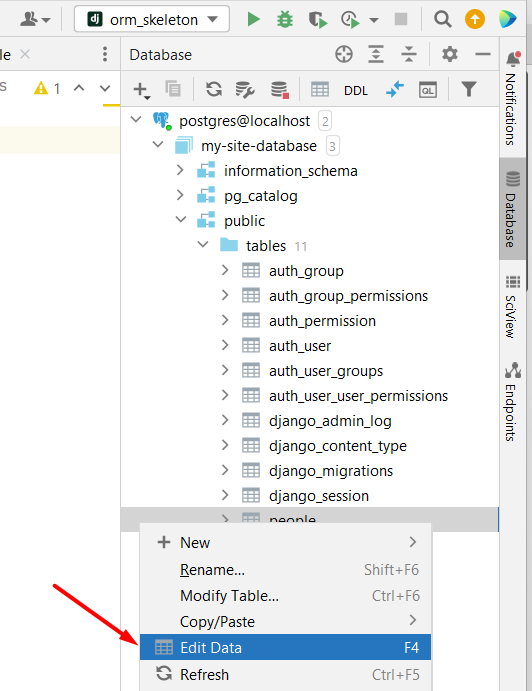
Now, let us **add some records** (rows) to the **"people"** **table**: A screenshot of a computer

Description automatically generated

Subsequently, we can **verify the inserted data**: A screenshot of a computer

Description automatically generated

For a **more interactive exploration** of the inserted data, the **Database PyCharm Tool** comes in handy. By **right-clicking** **on the** **"people"** **table** and **selecting** **"Edit Data"** **from the menu**, we gain **access to a user-friendly** **interface**:



Here we can **find our data**: A screenshot of a computer

Description automatically generated

From this interface, we can **add rows of data**: A screenshot of a computer

Description automatically generated

We need to **submit the newly added data**:

A screenshot of a computer

Description automatically generated

*Our exploration of the dbshell within the context of Django has revealed a valuable tool for direct interaction with the database. We have learned that the dbshell enables us to execute raw SQL commands directly through the terminal, granting us direct control over database operations. This newfound understanding of the dbshell equips us with an effective method for navigating and interacting with databases within the Django framework.*