**Relationship In Django:**

There are 3 types of relationship :

1. One-to-One Relationship
2. One-to-Many Relationship
3. Many-to-Many Relationship

**One-to-One Relationship**

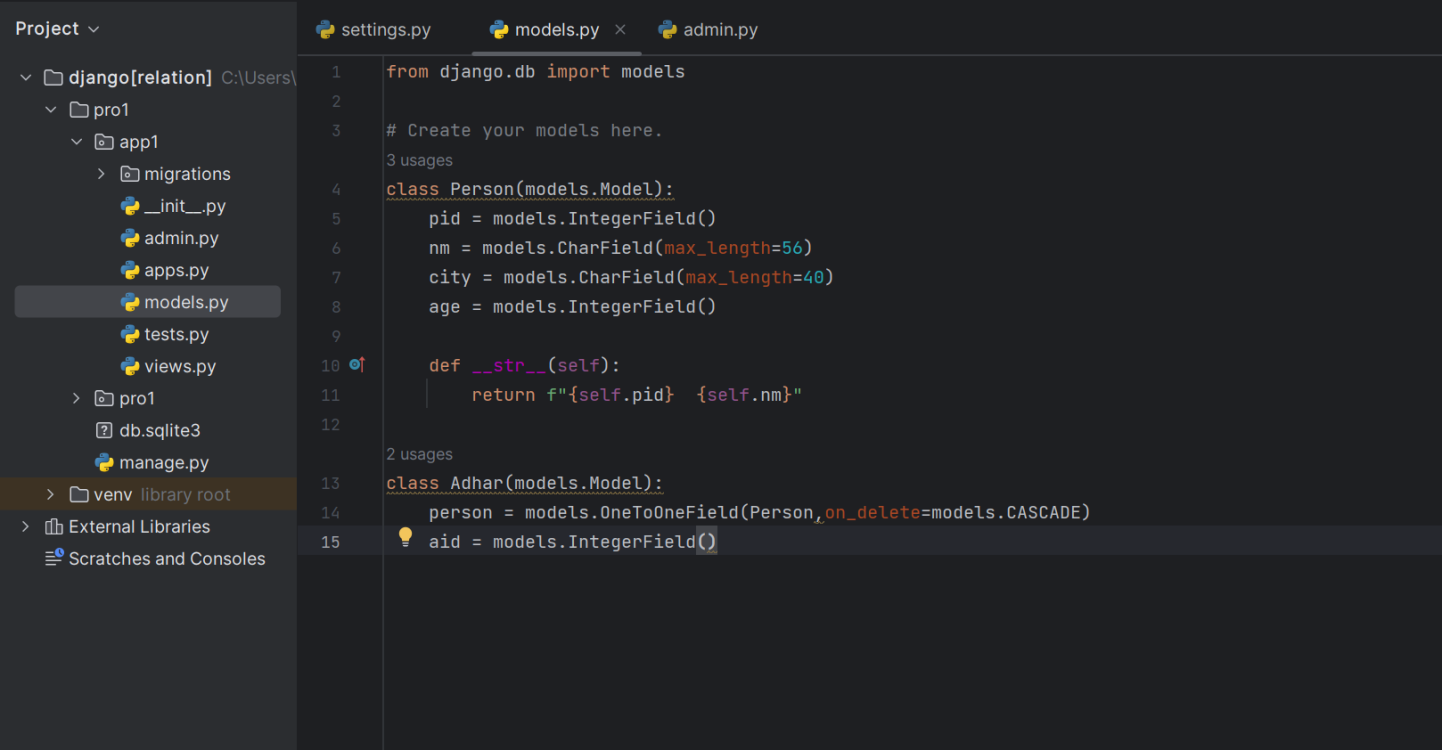
A **one-to-one** relationship means that a record in one table relates to a single record in another table.

* To define a one-to-one relationship, use [**OneToOneField**](https://docs.djangoproject.com/en/5.0/ref/models/fields/#django.db.models.OneToOneField)**.**
* Generally we keep FK field name in lowercase of parent Model.

Let's create an example with a Person model and an associated Adhar model. Each person will have only one Adhar number.

Step 1: Let's create the Adhar model with a one-to-one relationship to the Person model:

models.py file



In this example, the Adhar model has a OneToOneField named person, which links it to the Person model. The on\_delete=models.CASCADE argument specifies that if a Person instance is deleted, the associated Adhar instance will also be deleted.

**Step 2**: Insert data in database we have to use admin panel

A screenshot of a computer

Description automatically generated

**Step 3:** Command

1. py manage.py makemigrations(migration)
2. py manage.py migrate
3. py manage.py createsuperuser(create superuser)
4. py manage.py runserver(server run )
5. enter admin url ([Log in | Django site admin](http://127.0.0.1:8000/admin/login/?next=/admin/) 🡪http://127.0.0.1:8000/admin/)

A screenshot of a computer

Description automatically generated

3 Record added in person table we refer ‘person’ table to add record in ‘adhar’ table.

A screenshot of a computer

Description automatically generated

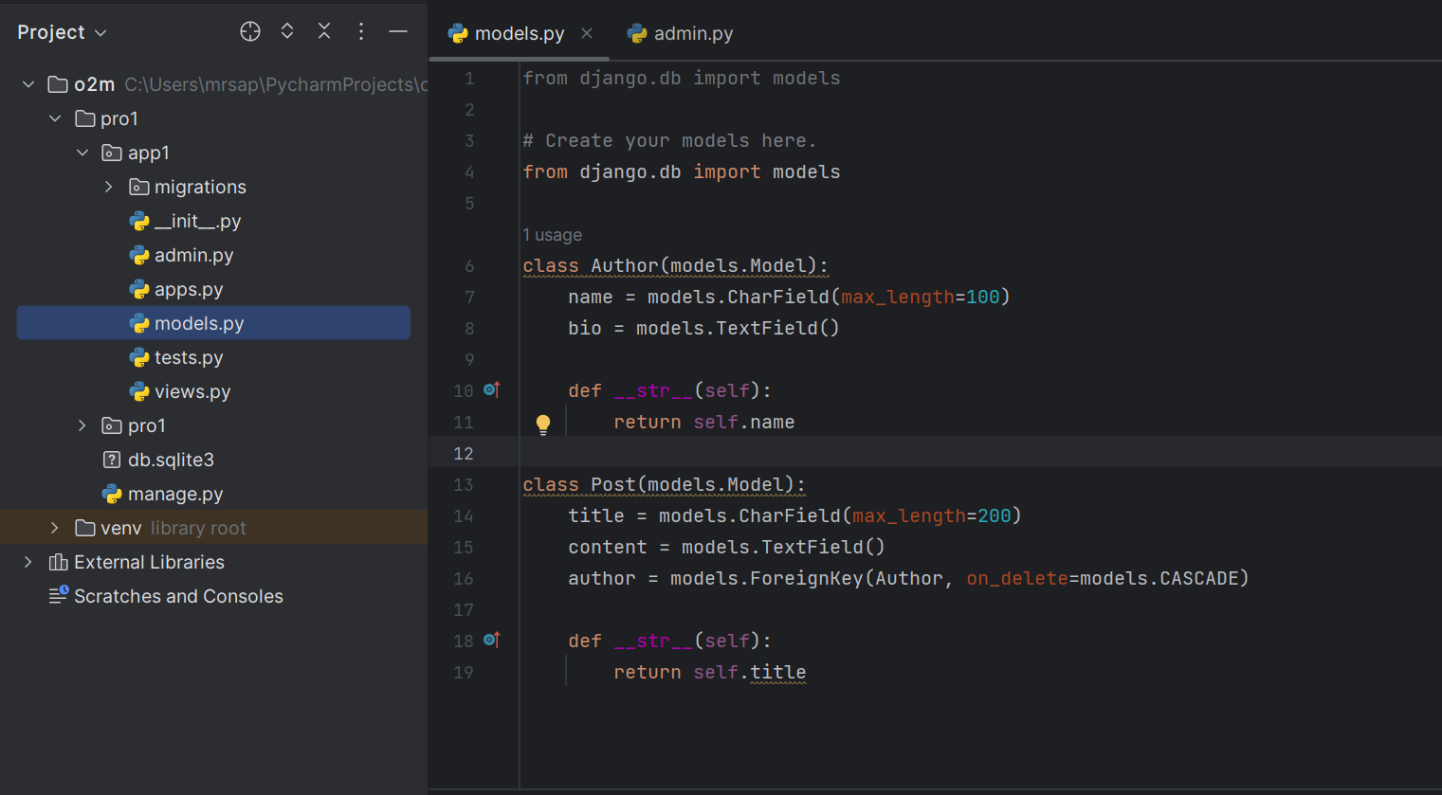
**One-to-Many Relationship**

Django, a one-to-many relationship is established between two models when one instance of a model (the "parent" or "one" side) can be related to multiple instances of another model (the "child" or "many" side).

* record in one table relates to multiple records in another table.
* To define a *one-to-many* relationship, use [**ForeignKey**](https://docs.djangoproject.com/en/5.0/ref/models/fields/#django.db.models.ForeignKey).

Let's take an example of a simple blogging platform to illustrate this relationship. We'll have two models: Author and Post. Each author can have multiple posts, creating a one-to-many relationship between authors and posts.

**Step 1:** Let's create the Author model with a one-to-many relationship to the Post model using field ForeignKey:

models.py file 

Author model: Represents an author with a name and bio.

Post model: Represents a blog post with a title, content, and a ForeignKey to Author.

In the Post model, author field is defined as a ForeignKey to Author. This indicates a many-to-one relationship, where each post is associated with a single author.

**key points**

One Author, Many Posts: An instance of the Author model can be associated with multiple instances of the Post model through the ForeignKey.

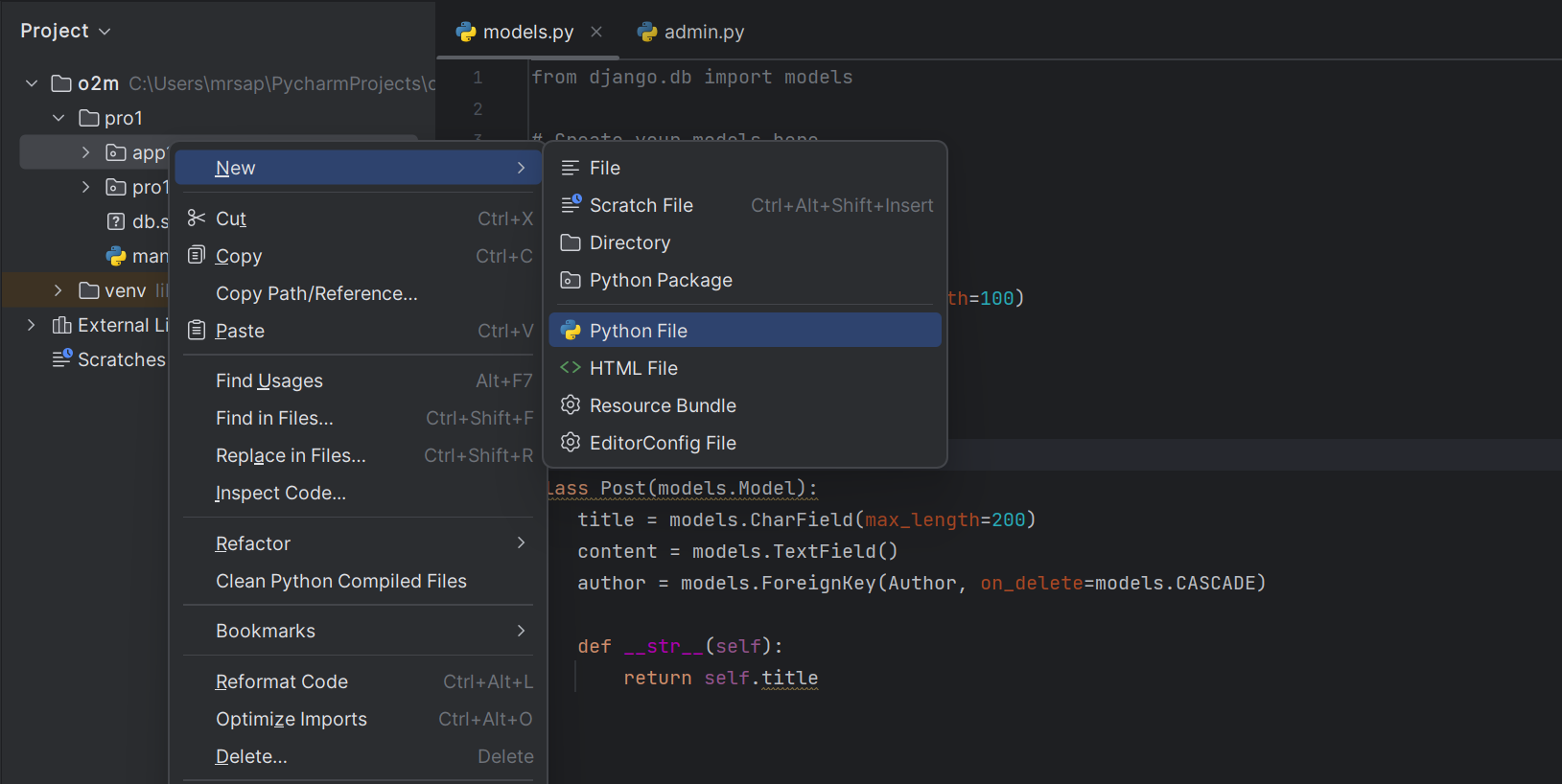
**Step 2**: you need to tell Django that you’re going to use them by registering the application name in the INSTALLED\_APPS list in the settings.py of the project:(Register app into the INSTALLED\_APPS inside settings.py file.

**Step 3:** Migration Command

**py manage.py makemigrations**

**py manage.py migrate**

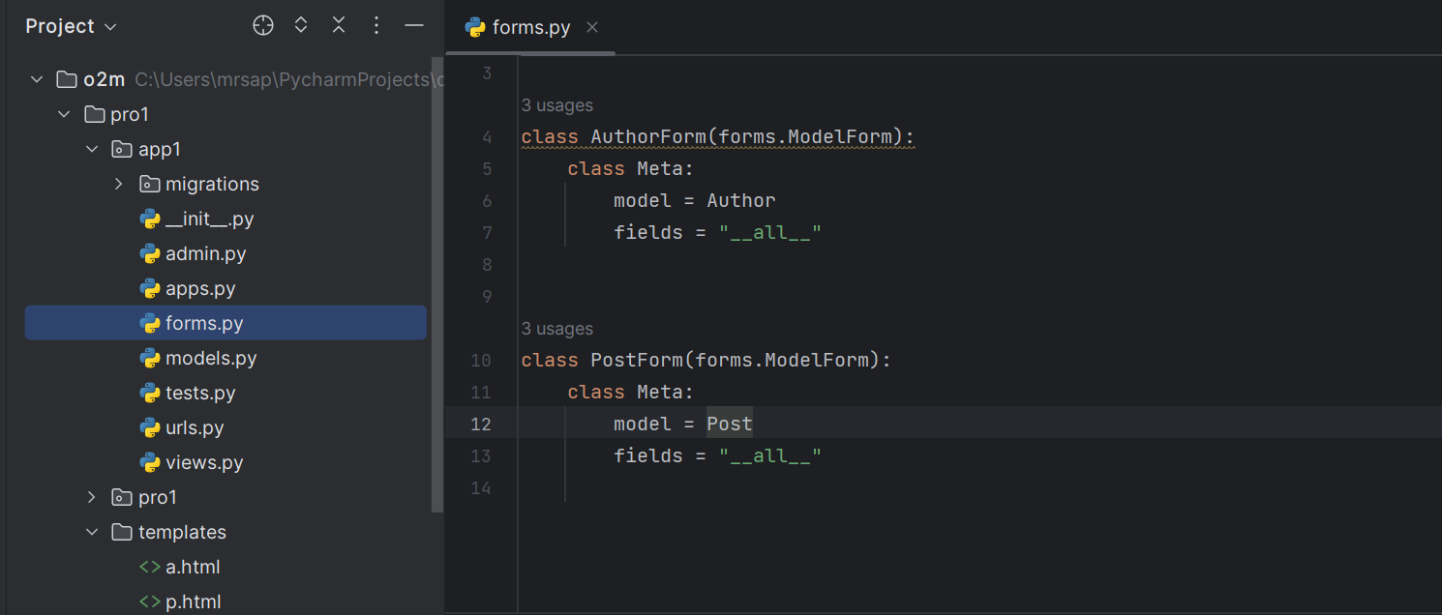
**Step 4:** Create forms.py file in app directory



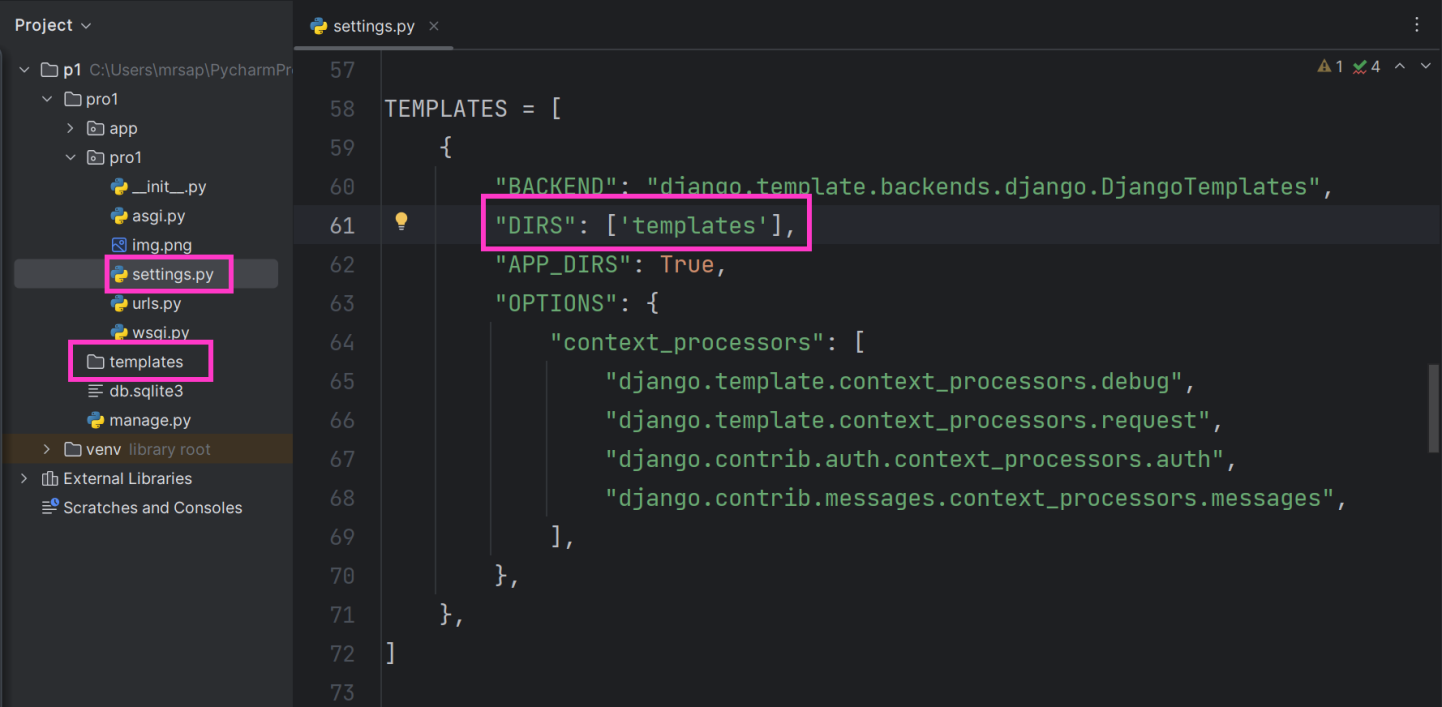
**Step 5:** Next, create Django forms for entering data entry for Author and Post:

AuthorForm allows entering author details, while PostForm includes a dropdown to select an author when creating a new post.

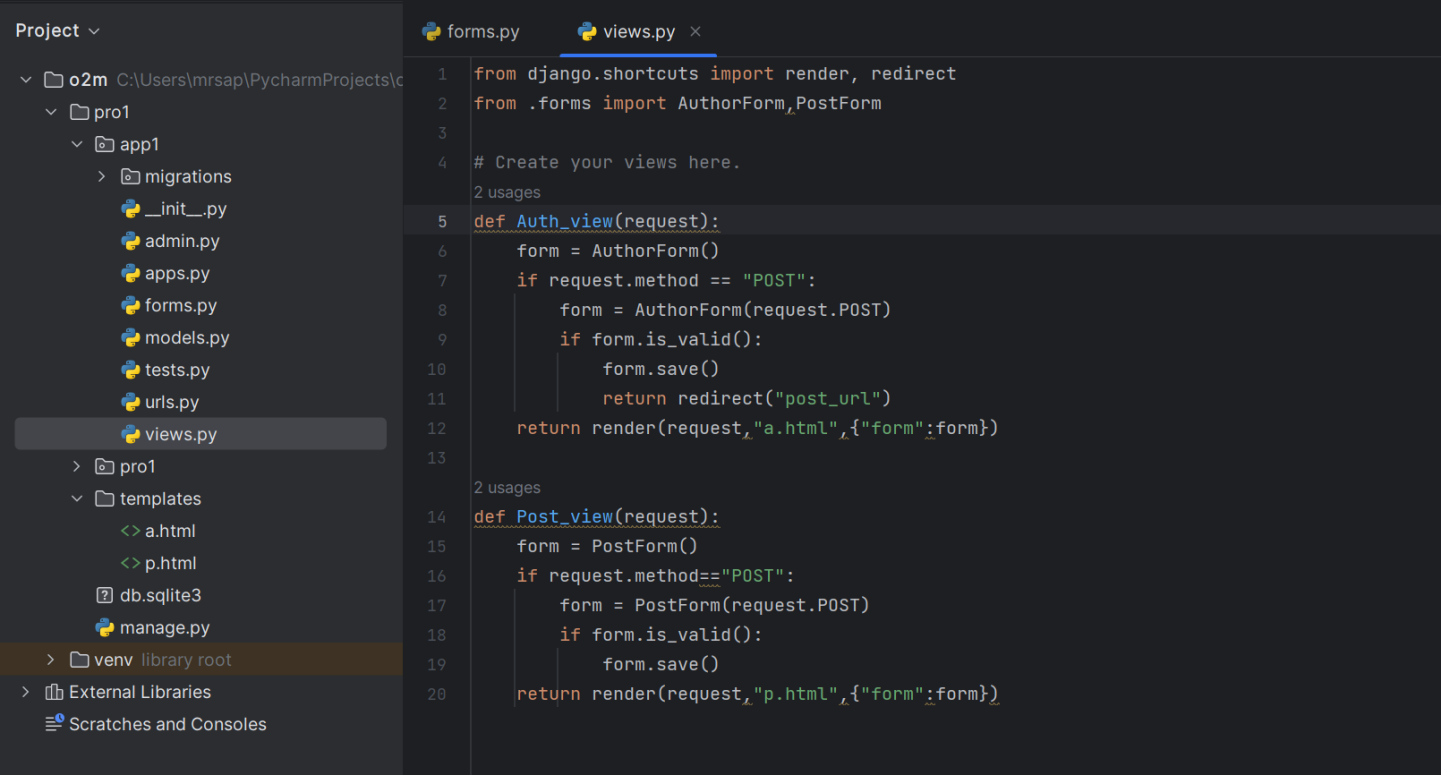
Forms.py file



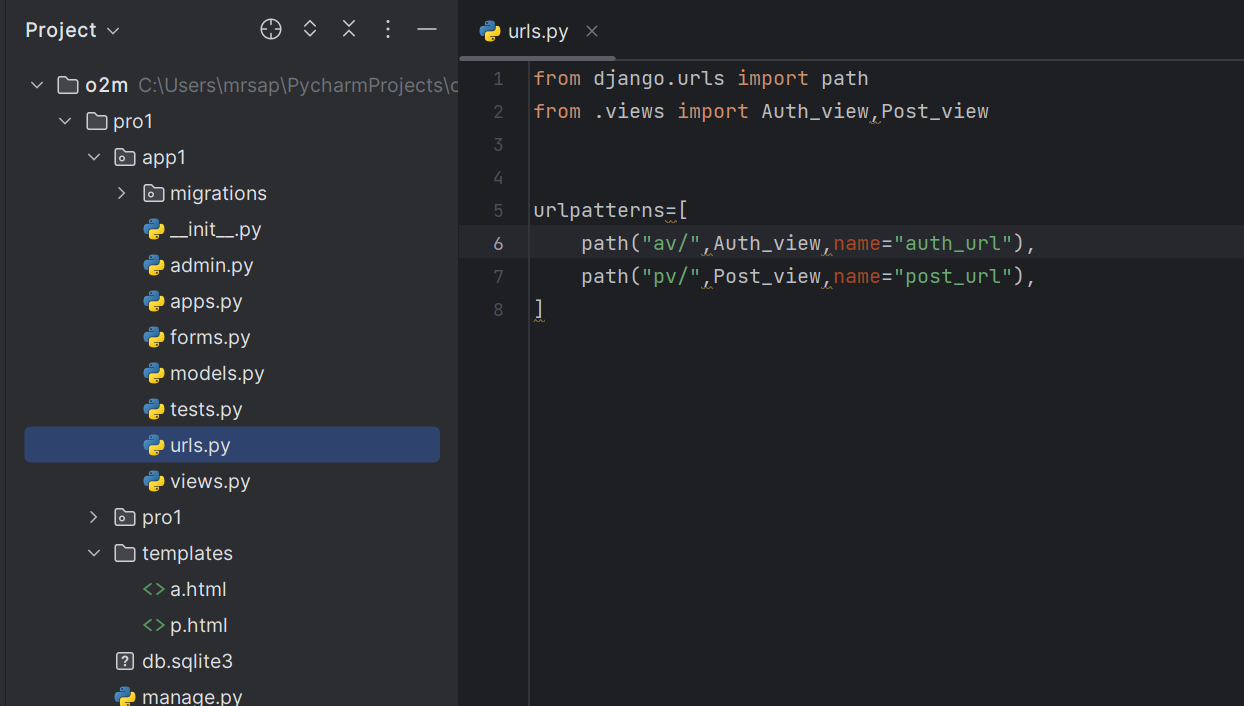
**Step 6:** Create templates directory inside that directory we have to create new .html file and Register our templates folder in setting.py inside the TEMPLATES list .So that Django will search for html files inside templates directory



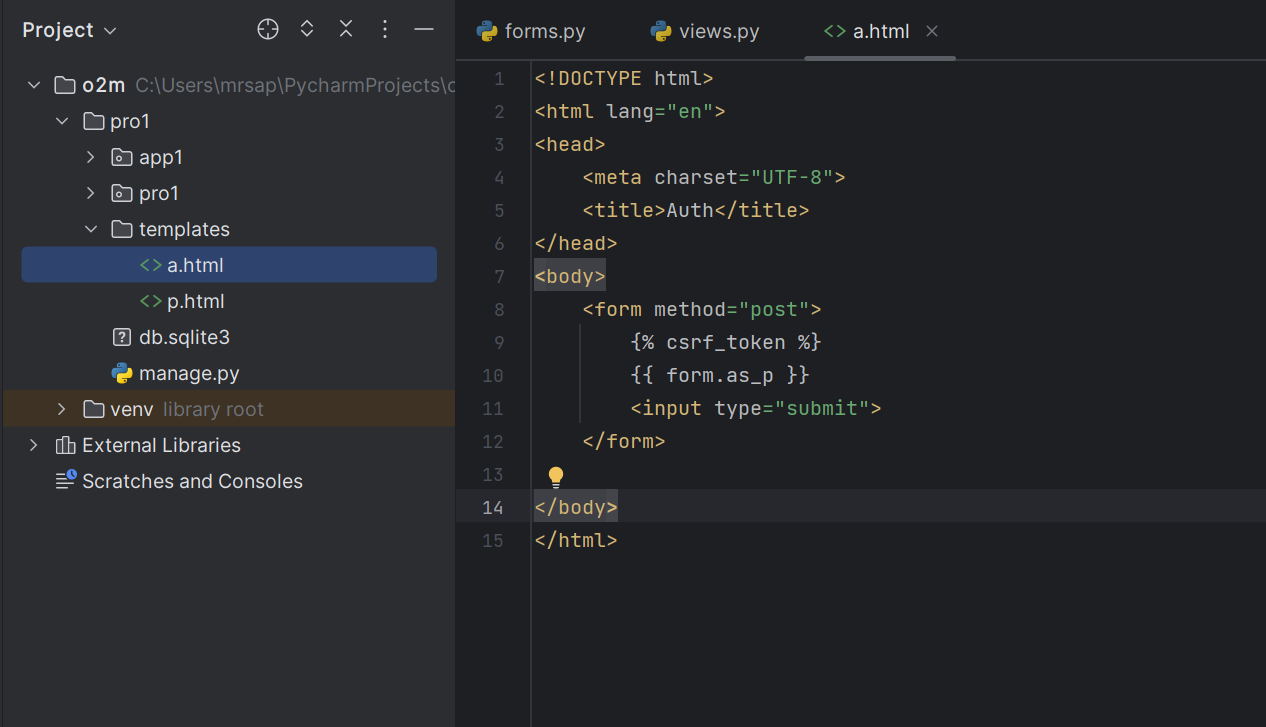
**Step 7:** Define view for Author and Post in views.py file



**Step 8:** Define url in urls.py file

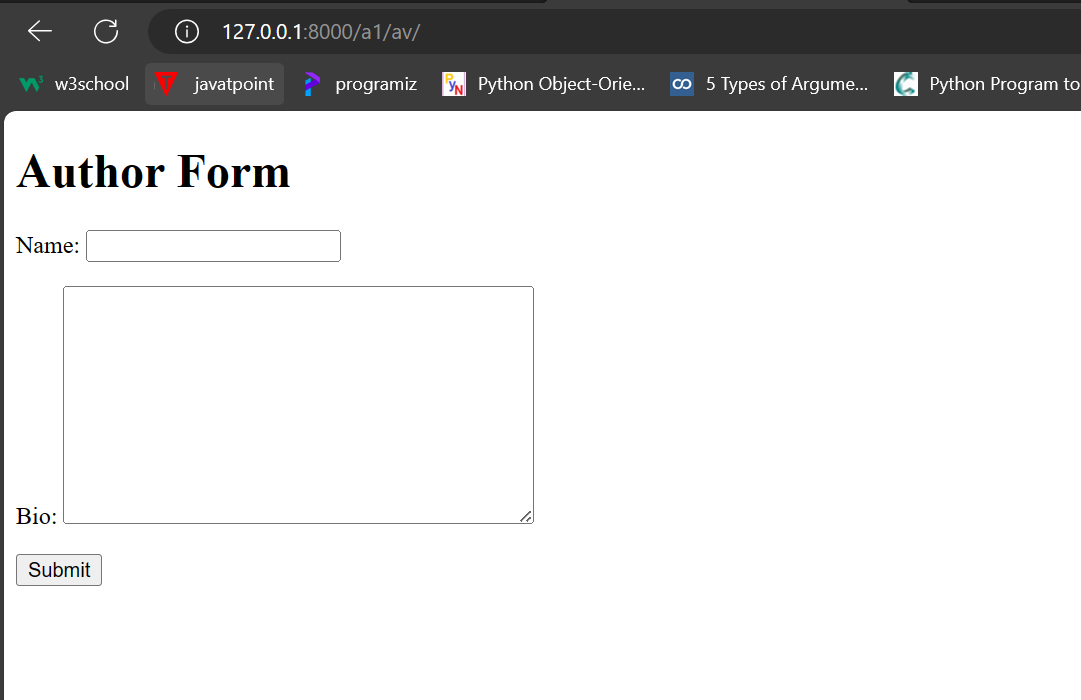


**Step 9:** Next step to define templates for auth.html and post.html.



**Step 10: Run server**

**Py managr .py runserver**



**Post Form**

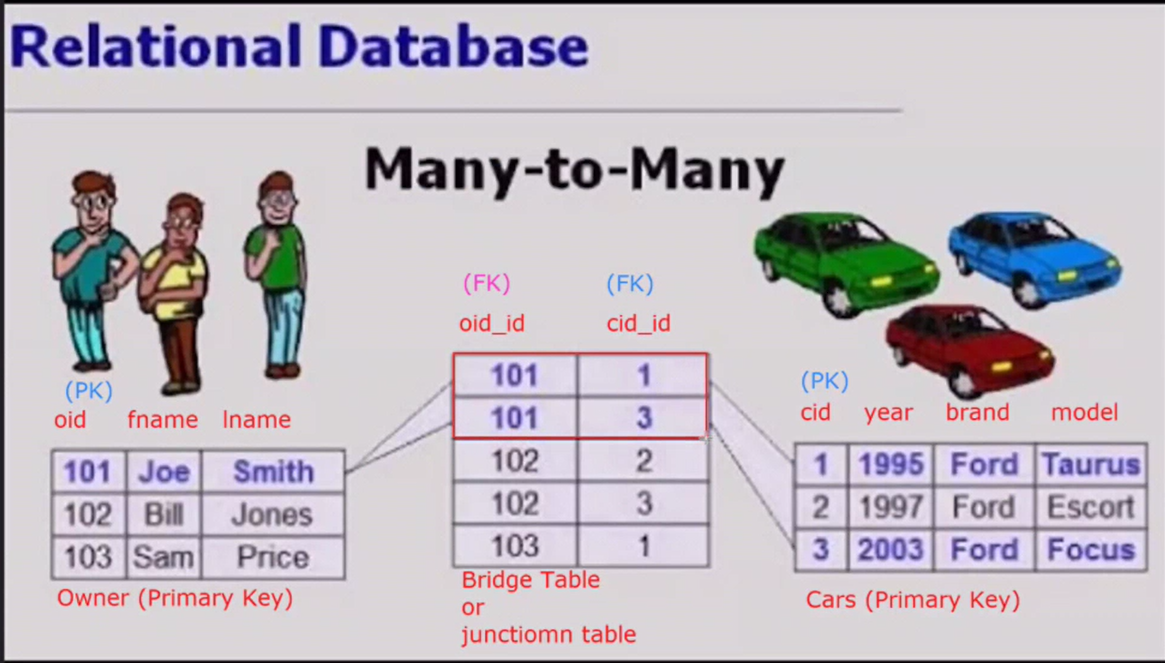
A screenshot of a computer

Description automatically generated

**Many-to-Many Relationship**

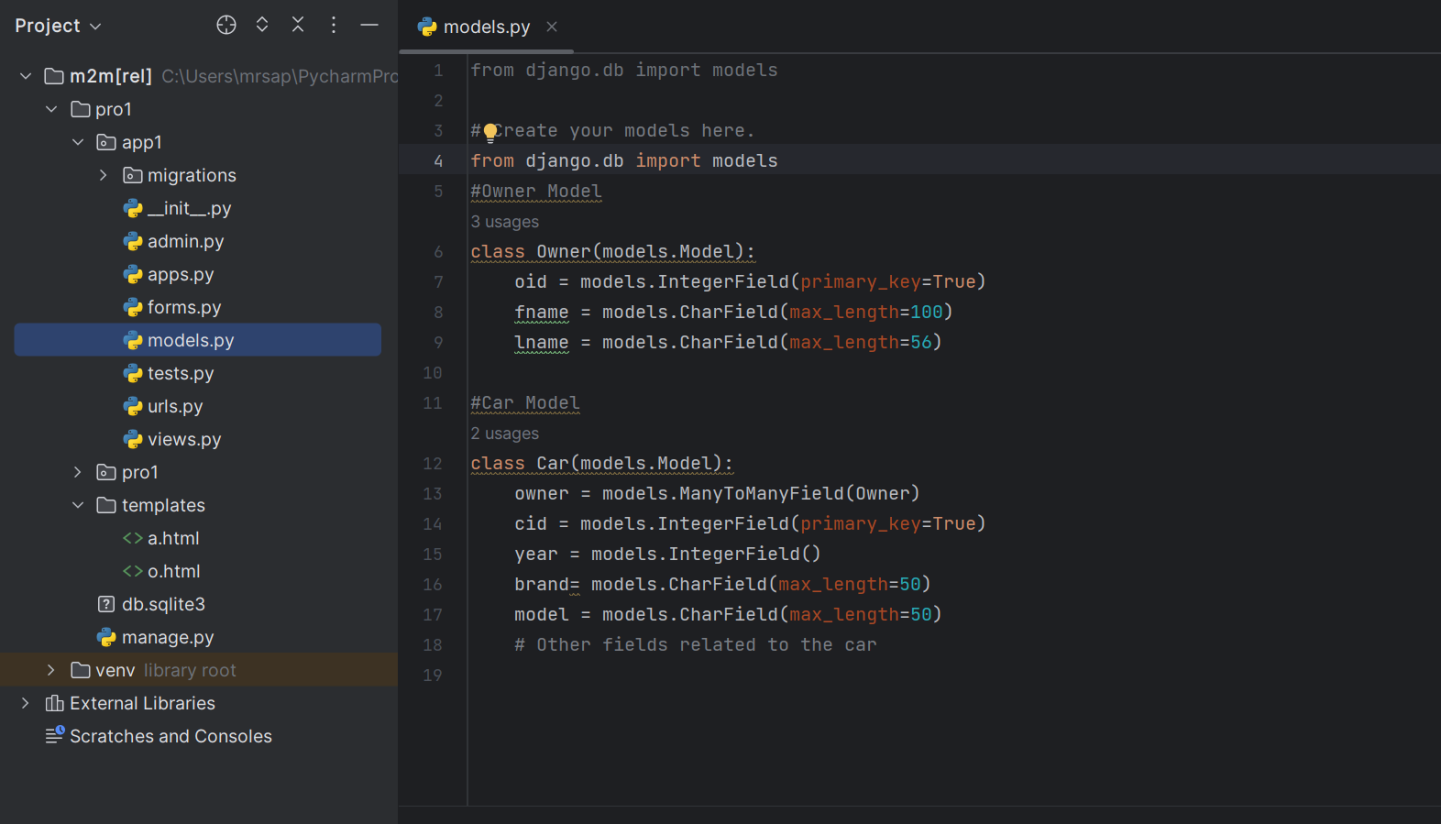
In Django, a many-to-many relationship between two models, such as an "Owner" and "Car," implies that each owner can possess multiple cars, and each car can belong to multiple owners. To implement this, you'd use a ManyToManyField which creates an intermediate table to manage the relationship between the two models.

* In a **many-to-many** relationship, multiple records in one table relate to many records in another table.
* To define a many-to-many relationship, use [**ManyToManyField**](https://docs.djangoproject.com/en/5.0/ref/models/fields/#django.db.models.ManyToManyField).



**Step 1:** Establishing the Many-to-Many Relationship:

To create the many-to-many relationship between Owner and Car, modify either model to include a ManyToManyField pointing to the other model.



**Step 2:** Applying Changes and Migrations:

After making these changes to your models, run the following commands in your terminal to create and apply the migrations:

**python manage.py makemigrations**

**python manage.py migrate**

**Note:**

In many-to-many relationship django automatically create a junction table of Car and Owner model by the name app\_nm\_Car\_Owner. When working with databases with Django, it automatically creates 10 default tables in the database.

