# Django Models

A Django model is a table in your database.

1)Set Environment

venv/Scripts/activate

2)Install Django

ensure you have Django installed.

pip install django

3)Create a new Django project using the following command:

django-admin startproject ModelBase .

4)Create a Django App

Create a new Django app using the following command:

python manage.py startapp ModelApp

5)Add app name into settings.py INSTALLED\_APP

6)Define the Model

In the models.py file inside your app directory (yourappname), define your model. Here's your example model:

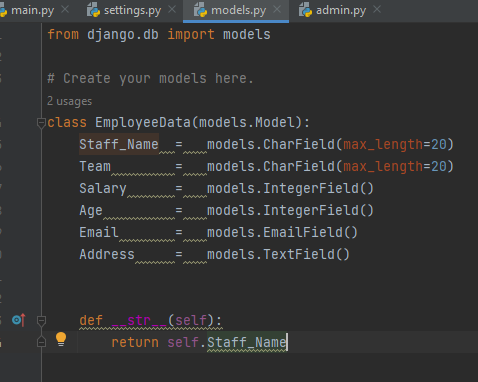
ango model named EmployeeData with fields for staff name, team, salary, age, email, and address. This model represents a basic structure for storing employee data in a Django application.

Here's a brief overview of the fields you've defined:

* Staff\_Name: CharField with a maximum length of 20 characters, representing the name of the staff.
* Team: CharField with a maximum length of 20 characters, representing the team to which the staff belongs.
* Salary: IntegerField, representing the salary of the staff.
* Age: IntegerField, representing the age of the staff.
* Email: EmailField, representing the email address of the staff.
* Address: TextField, representing the address of the staff.

The \_\_str\_\_ method is implemented to return the staff name as the string representation of the model instance.

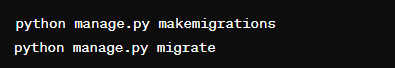
This is a good starting point for a Django model. Depending on your application's requirements, you might want to consider adding more fields or tweaking the existing ones. Additionally, you can create relationships between models if needed.



This model (EmployeeData) has fields like Staff\_Name, Team, Salary, Age, Email, and Address.

7)Run Migrations

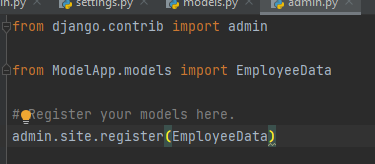
Run the following commands to create and apply migrations:



8)Registering your model with the Django Admin site allows you to manage instances of that model using the Django Admin interface. Here's a brief explanation:

### Step: Registering the Model in admin.py

In your app directory (e.g., yourappname), you typically have an admin.py file. If it doesn't exist, you can create one. Inside admin.py, you register your model with the Django Admin site:



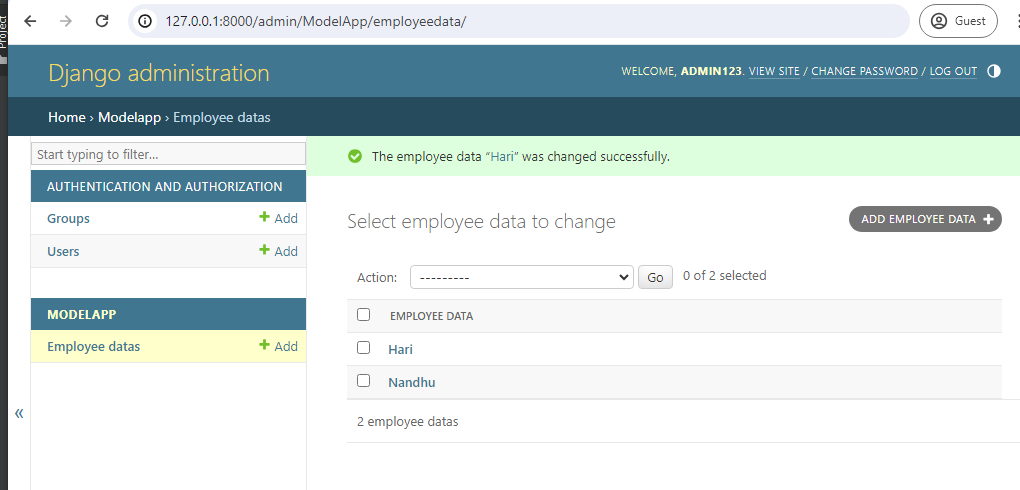
Explanation:

* from django.contrib import admin: Import the admin module from django.contrib.
* from .models import EmployeeData: Import your EmployeeData model. The . denotes the current directory.
* admin.site.register(EmployeeData): Register the EmployeeData model with the Django Admin site. This line tells Django to make this model available in the Admin interface.

9)

Remember to create a superuser account if you haven't already using python manage.py createsuperuser. This account will give you access to the Django Admin interface with administrative privileges.

10)when you run your Django development server (python manage.py runserver), you can navigate to http://127.0.0.1:8000/admin/ and log in with a superuser account. You should see your EmployeeData model listed there, allowing you to perform CRUD operations (Create, Read, Update, Delete) on instances of the model.

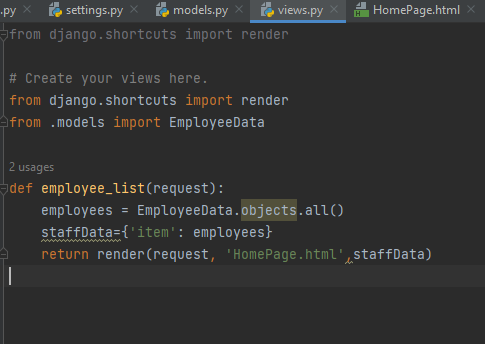


10)To display the data from your EmployeeData model in an HTML page, you need to follow these steps:

* Create a Django view: Define a view function in your Django app that retrieves the data from the database and passes it to the template.
* Create a Django template: Design an HTML template that will be used to display the data. You can use Django template tags to dynamically render the data in your HTML.
* Configure URL patterns: Map a URL to your view so that when a user accesses a specific URL, the associated view is called.

Here's an example implementation:

Views (in views.py):



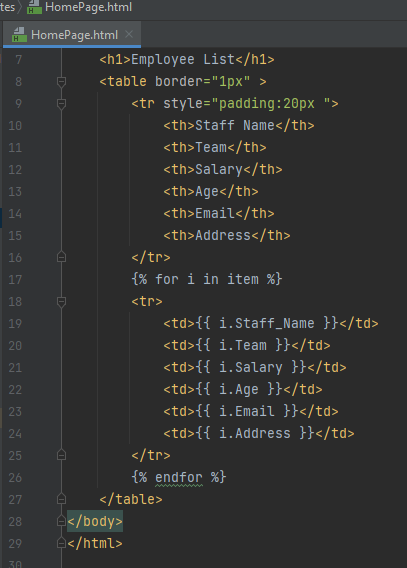
importing the render function from Django, which is used to render HTML templates, and you're importing the EmployeeData model from your app

the employee\_list view function:

* **employees = EmployeeData.objects.all():** This line queries all instances of the EmployeeData model from the database and stores them in the employees variable.
* **staffData = {'item': employees}:** This line creates a dictionary named staffData with the key 'item' mapped to the queryset of all employees retrieved from the database. This dictionary will be passed to the template.
* **return render(request, 'HomePage.html', staffData):** This line uses the render function to render the HTML template named 'HomePage.html'. The template will receive the data in the staffData dictionary. The request object is passed to the render function to generate the complete HTTP response.

this code defines a view function (employee\_list) that retrieves all instances of the EmployeeData model from the database, creates a dictionary (staffData) with the key 'item' and the queryset of employees, and renders the 'HomePage.html' template with this data. The rendered HTML page will display the list of employees in a table.

11)

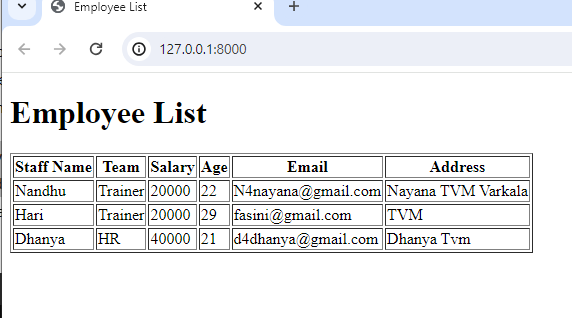


HTML template (HomePage.html) step by step:

* Document Type Declaration (<!DOCTYPE html>):
  + This declaration specifies the document type and version of HTML being used. In this case, it's HTML5.
* HTML Document Structure (<html>):
  + The <html> tag is the root element of the HTML document.
* Head Section (<head>):
  + The <head> section contains metadata about the HTML document, such as the document title.
* Title (<title>):
  + The <title> tag sets the title of the HTML document, which appears in the browser's title bar or tab.
* Body Section (<body>):
  + The <body> tag contains the content of the HTML document, which is what users see in the browser.
* Heading (<h1>):
  + The <h1> tag defines a top-level heading, indicating that this is the main title of the page. In this case, it's "Employee List."
* Table (<table>):
  + The <table> tag is used to create an HTML table.
* Table Border Attribute (border="1px"):
  + This sets a border of 1 pixel around the entire table.
* Table Header Row (<tr>):
  + The <tr> tag represents a table row. This row contains the headers for each column.
* Table Header Cells (<th>):
  + The <th> tags define table header cells. They represent the headers of each column in the table.
* Inline Style for Header Row (style="padding:20px"):
  + This sets an inline style for the header row, adding 20 pixels of padding to each side.
* Data Loop ({% for i in item %}):
  + This is a Django template tag used to loop through the items in the item variable (which contains the queryset of employees passed from the view).
* Table Data Rows (<tr>):
  + The <tr> tag represents a table row for each employee.
* Table Data Cells (<td>):
  + The <td> tags define table data cells. They represent the values for each attribute (Staff Name, Team, Salary, Age, Email, Address) of an employee.
* Django Template Variables ({{ i.Staff\_Name }}):
  + These template variables display the values of the corresponding attributes for each employee.
* Data Loop End ({% endfor %}):
  + Marks the end of the loop through the employee data.

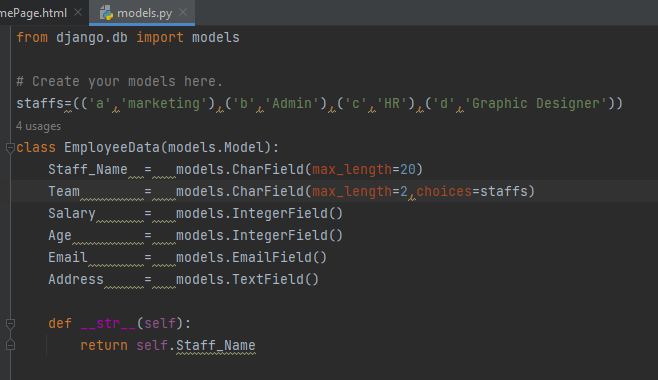
In summary, this HTML template creates a structured page with a table that displays employee data. The data is retrieved from the item variable, which is a queryset of employees passed from the Django view. The template uses Django template tags to dynamically generate HTML based on the data received from the view. The styling includes a border around the table and additional padding for the header row.

When you run the **python manage.py runserver** command, you should see output in the terminal or command prompt

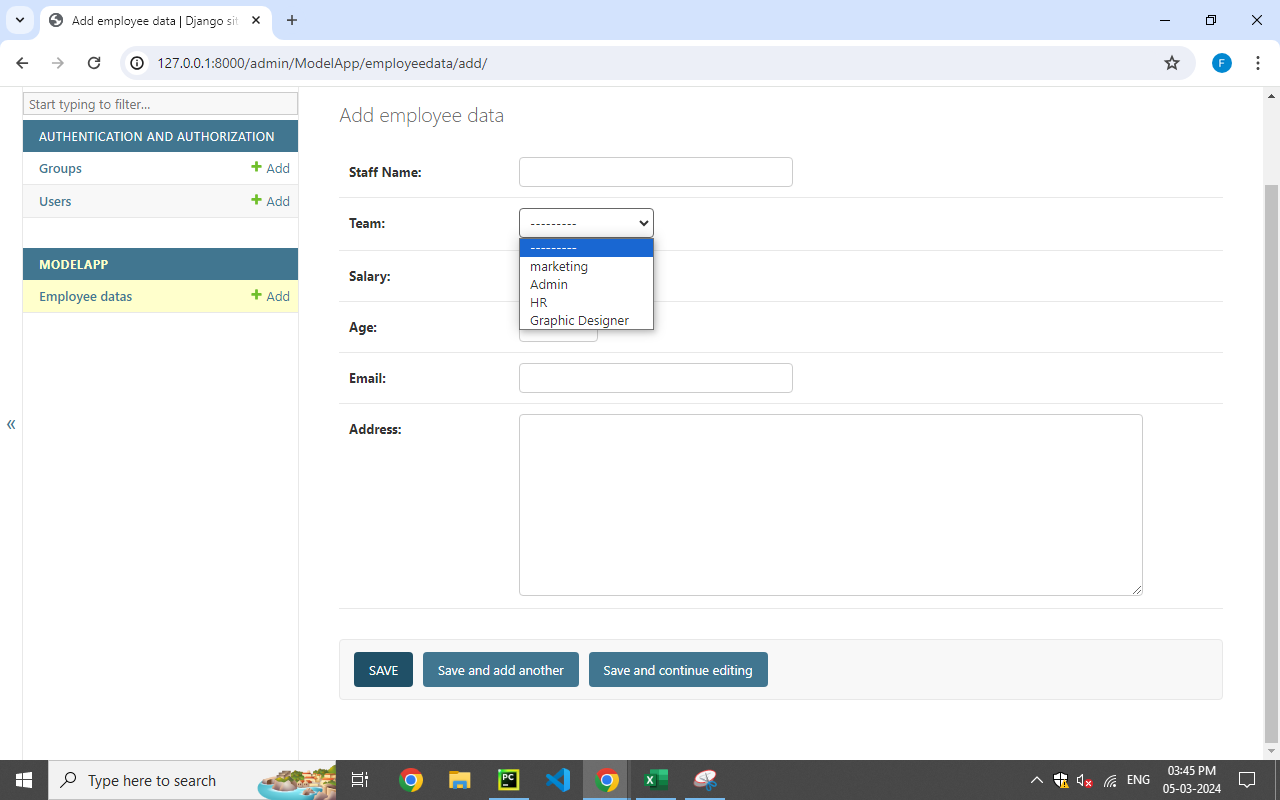


added a choices parameter to the Team field in your EmployeeData model. This is a good approach when you want to restrict the possible values for a field to a predefined set.

In your example, you've defined a tuple staffs containing choices for the "Team" field. Each choice is a tuple with two elements: the first element is the actual value stored in the database, and the second element is the human-readable name displayed in forms and the admin.



With this setup, when you create or edit an EmployeeData object using Django forms or the admin panel, the "Team" field will be presented as a dropdown list with the options 'Marketing', 'Admin', 'HR', and 'Graphic Designer'. The corresponding values ('a', 'b', 'c', 'd') will be stored in the database.



Admin page look like this

When you use this model in a Django form or the admin panel, Django will automatically render a dropdown list for the "Team" field, allowing users to select one of the predefined options.