# Campus Recruitment Training (CRT) classes

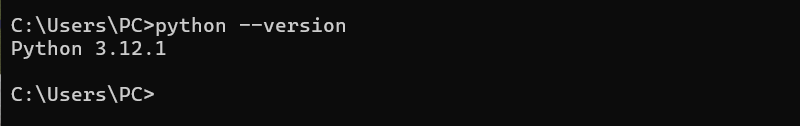
**Projects Included In this Programme:**

1. To-Do-List Project
2. Student Report Project
3. FastApi

***Python Installation***:

**Commands to run**:

**>>** python –version.



***Django Installation and Setup***:

Installing and setting up Django is a straightforward process. Below are the step-by-step instructions to install Django and set up a new Django project on your system.

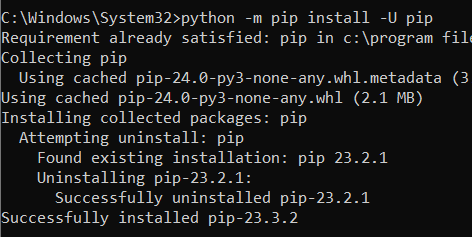
**Install Pip**

ToInstall Django in Linux and Mac is similar, here I am showing it in Windows for Linux and Mac just open the terminal in place of the command prompt and go through the following commands.

**Install Pip**

Open the command prompt and enter the following command-

**>> python -m pip install -U pip**



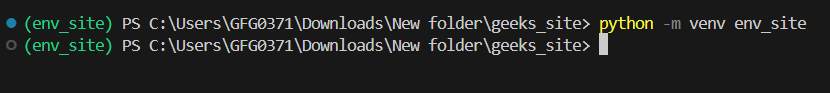
**Set Virtual environment:**

Setting up the virtual environment will allow you to edit the dependency which generally your system wouldn't allow. Follow these steps to set up a virtual environment-

**Create virtual environment in Django:**

We should first go the directory where we want to create the [virtual environment](https://www.geeksforgeeks.org/python-virtual-environment/) then we type the following command to create virtual environment in django.

**python -m venv env\_site**



then we need to activate virtual environment in Django

**Activate the virtual environment:**

Run the activation script located in the bin directory within the virtual environment folder

* **For Windows:**

**.\env\_site\Scripts\activate.ps1**

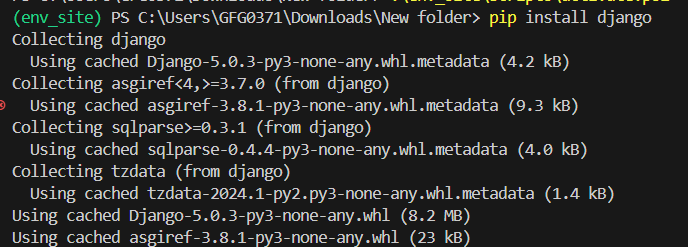
* **For MacOs/Linux:**

**source env\_site/bin/activate**

**Install Django:**

Install django by giving following command

**pip install django**



**Django Setup**

Once Django is installed, we can start to create a new Django project.

**Step 1: Start a new Django Project**

Start a project by following command-

**django-admin startproject geeks\_site**

4

**Step 2: Navigate to the Project Directory**

Change directory to geeks\_site

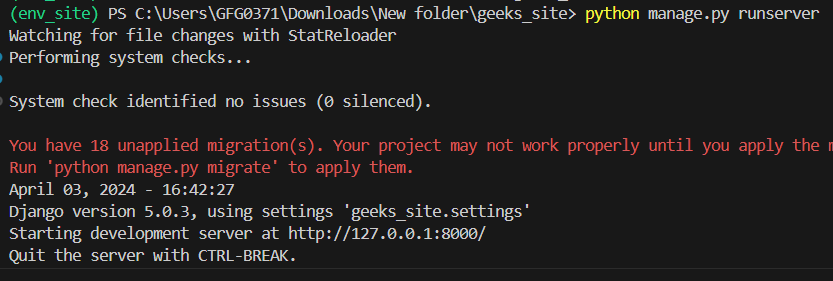
**cd geeks\_site**

4

**Step 3: Start the server**

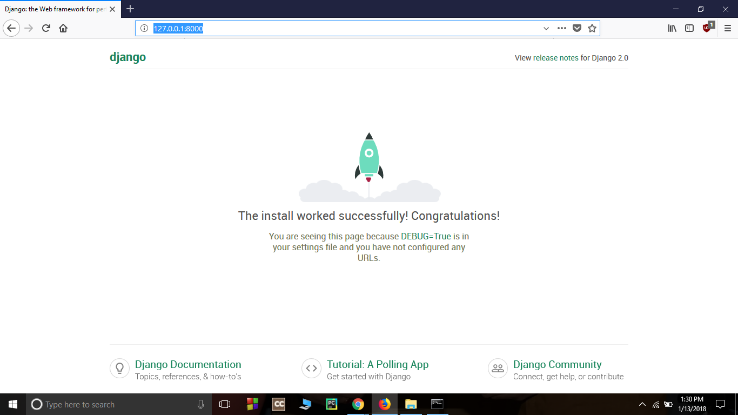
Start the server by typing following command in cmd-

**python manage.py runserver**



**Step 4: Verify Server Status**

To check whether server is running or not go to web browser and enter **http://127.0.0.1:8000/** as URL.



**When to Use Django? Comparison with other Development Stacks**

Django is a **high-level Python web framework** which allow us to quickly create web applications without all of the installation or dependency problems that we normally face with other frameworks.

One should be using Django for web development in the following cases:

* For developing a Web Application or API Backend.
* For Rapid Development of some web application.
* Deploying the application Fast and Scaling it according to your needs
* A Perfect ORM for working with databases instead of database queries

| **Django** | **MEAN/MERN** | **Spring** |
| --- | --- | --- |
| Python | JavaScript | Java |
| Open Source | NoSQL Databases | Open Source |
| Great Community | Modularity | Powerful |
| Easy to Learn | Mongoose | Dependency Injection |
| MVT Based | Organized | Stability |
| Batteries Included Framework | Dynamic Front End | MVC |

* To develop a secure single-page application for either retrieving data or posting data.

**Applications**

* Django’s built-in ORM and migrations make handling large databases easy.
* The admin panel helps manage data without extra setup.

**2. Applications with User Functionality**

* Built-in support for user registration, login, forms, and permissions.

**3. Quick Project Launches**

* Django follows a “batteries-included” philosophy- most tools are built-in, reducing the need for extra libraries.

**4. Python Developers**

* If you're already comfortable with Python, Django is a natural fit.

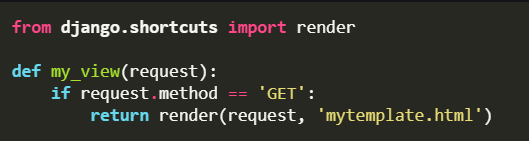
**When Django Might Not Be the Best Choice**

We should consider some alternative development frameworks when:

* **Building a small or simple site/SPA:** Use JavaScript frameworks like React or Angular.
* **Creating real-time apps:** Node.js handles concurrency better with its non-blocking I/O.
* **Developing a mobile app:** Use tools like React Native or Flutter

**Below are examples of a simple Django view function:**

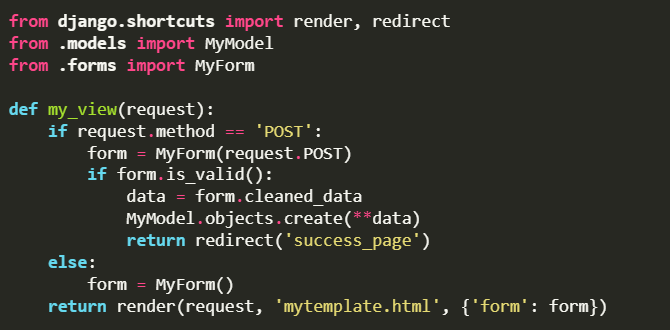
**Handling a GET Request:**

****

**Explanation:**

* render() returns an HTML template response.
* This view only processes GET requests and loads mytemplate.html.

**Handling a POST Request:**

****

**Explanation:**

* On POST, it initializes a form with submitted data.
* If valid, it creates a new model entry and redirects to a success page.
* On GET, it renders an empty form in the template.

**Companies using Django:**

* Instagram
* Disqus
* Pinterest
* Mozilla Firefox
* Spotify
* YouTube

**How to Create a Basic Project using MVT in Django ?**

**Create a basic Project:**

* To initiate a project of Django on Your PC, open Terminal and Enter the following command

**>>django-admin startproject projectName**

* A New Folder with the name *projectName*will be created. To enter in the project using the terminal enter command  >>**cd projectName**
* Create a new file views.py inside the project folder where settings.py, urls.py and other files are stored and save the following code in it-

**Example:**

**Project name:** Student record**.**

**In terminal:**

>>python -m Django startproject studentrecord

>>cd studentrecord

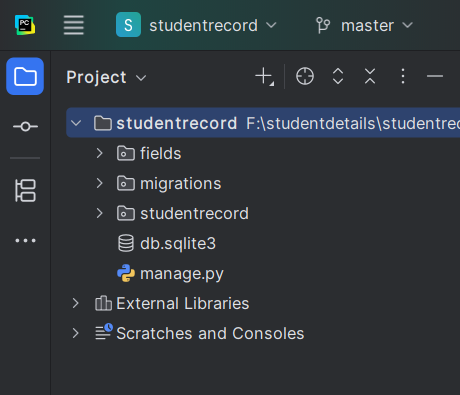
>>python manage.py startapp studentrecord\_app

>>python manage.py make migrations

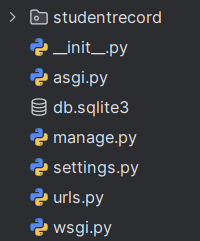
Or  
>>python manage.py migrate

>>python manage.py runserver

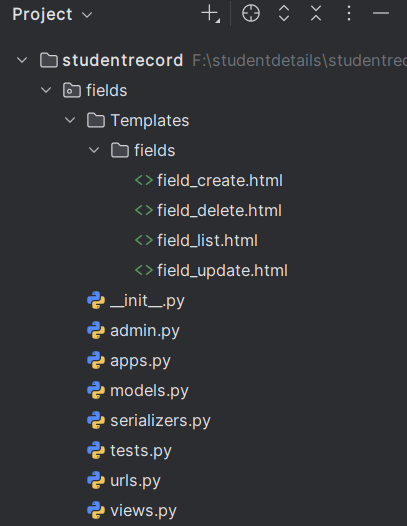
The Overview We Will Get:



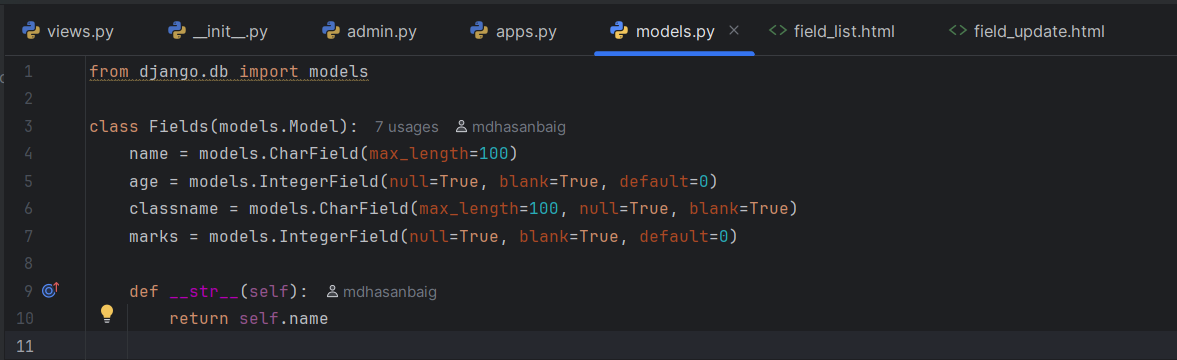
The App Contains Files:



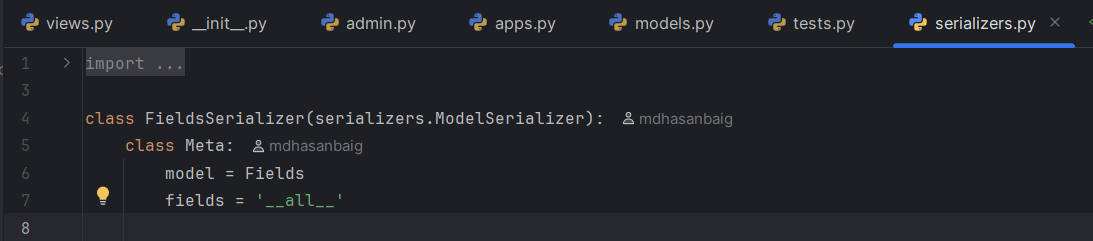
The Project Contains Files:



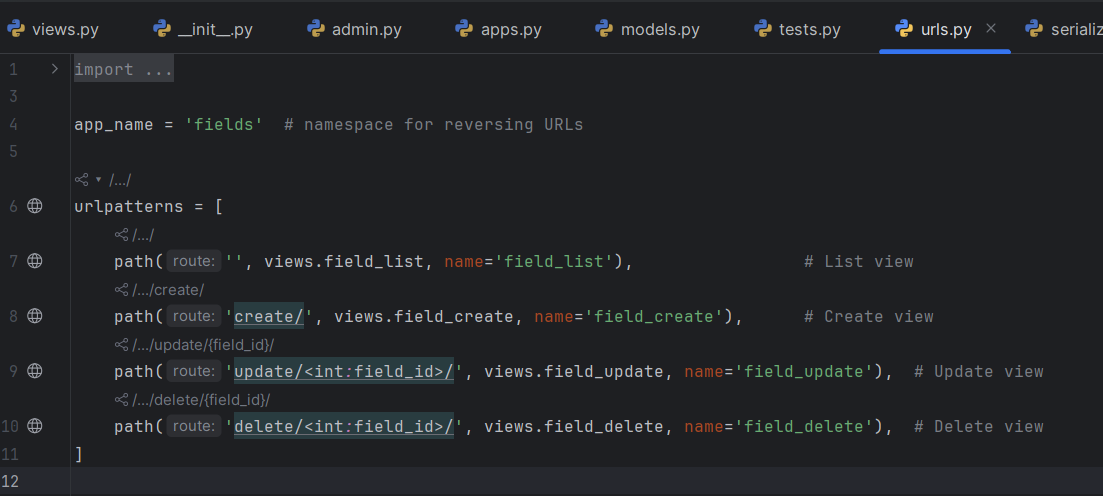
Models.py file:



Serializers.py file:



Urls.py file:



Views.py file:

from django.shortcuts import render, redirect, get\_object\_or\_404  
from .models import Fields  
  
# List all records with optional filtering  
def field\_list(request):  
 fields = Fields.objects.all()  
  
 # Apply filters based on query parameters  
 if 'name' in request.GET:  
 fields = fields.filter(name\_\_icontains=request.GET['name'])  
 if 'age' in request.GET:  
 fields = fields.filter(age=request.GET['age'])  
 if 'classname' in request.GET:  
 fields = fields.filter(classname\_\_icontains=request.GET['classname'])  
 if 'marks' in request.GET:  
 fields = fields.filter(marks=request.GET['marks'])  
  
 return render(request, 'fields/field\_list.html', {'fields': fields})  
  
# Create new record  
def field\_create(request):  
 if request.method == 'POST':  
 name = request.POST.get('name')  
 age = request.POST.get('age')  
 classname = request.POST.get('classname')  
 marks = request.POST.get('marks')  
 Fields.objects.create(name=name, age=age, classname=classname, marks=marks)  
 return redirect('fields:field\_list')  
 return render(request, 'fields/field\_create.html')  
  
# Update existing record  
def field\_update(request, field\_id):  
 field = get\_object\_or\_404(Fields, id=field\_id)  
 if request.method == 'POST':  
 field.name = request.POST.get('name')  
 field.age = request.POST.get('age')  
 field.classname = request.POST.get('classname')  
 field.marks = request.POST.get('marks')  
 field.save()  
 return redirect('fields:field\_list')  
 return render(request, 'fields/field\_update.html', {'field': field})  
  
# Delete record  
def field\_delete(request, field\_id):  
 field = get\_object\_or\_404(Fields, id=field\_id)  
 if request.method == 'POST':  
 field.deletex()  
 return redirect('fields:field\_list')  
 return render(request, 'fields/field\_delete.html', {'field': field})

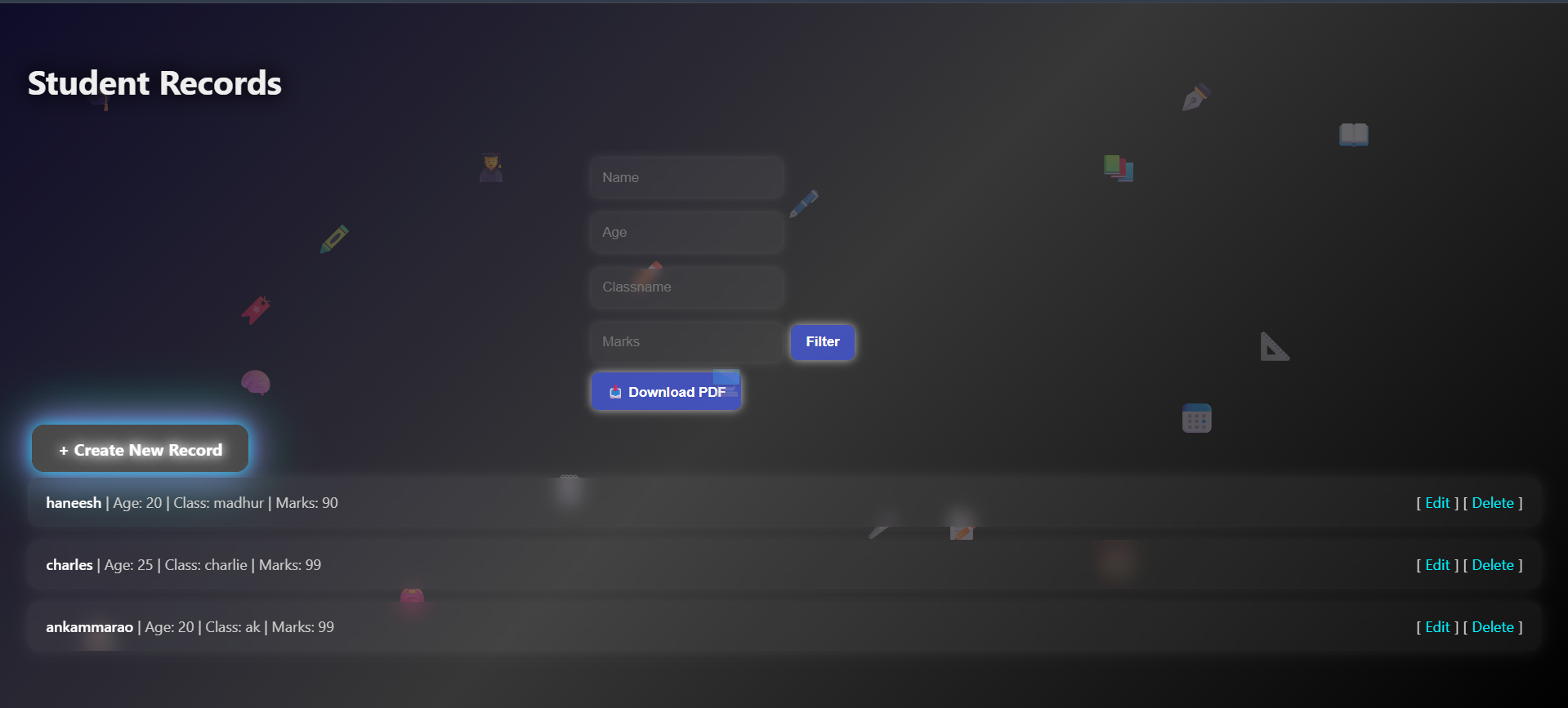
**Accessing the API Endpoints:**

Because in your **projectName/urls.py** you included your app’s URLs with the prefix **api/**, all your API endpoints will be accessed through URLs starting with **/api/**.

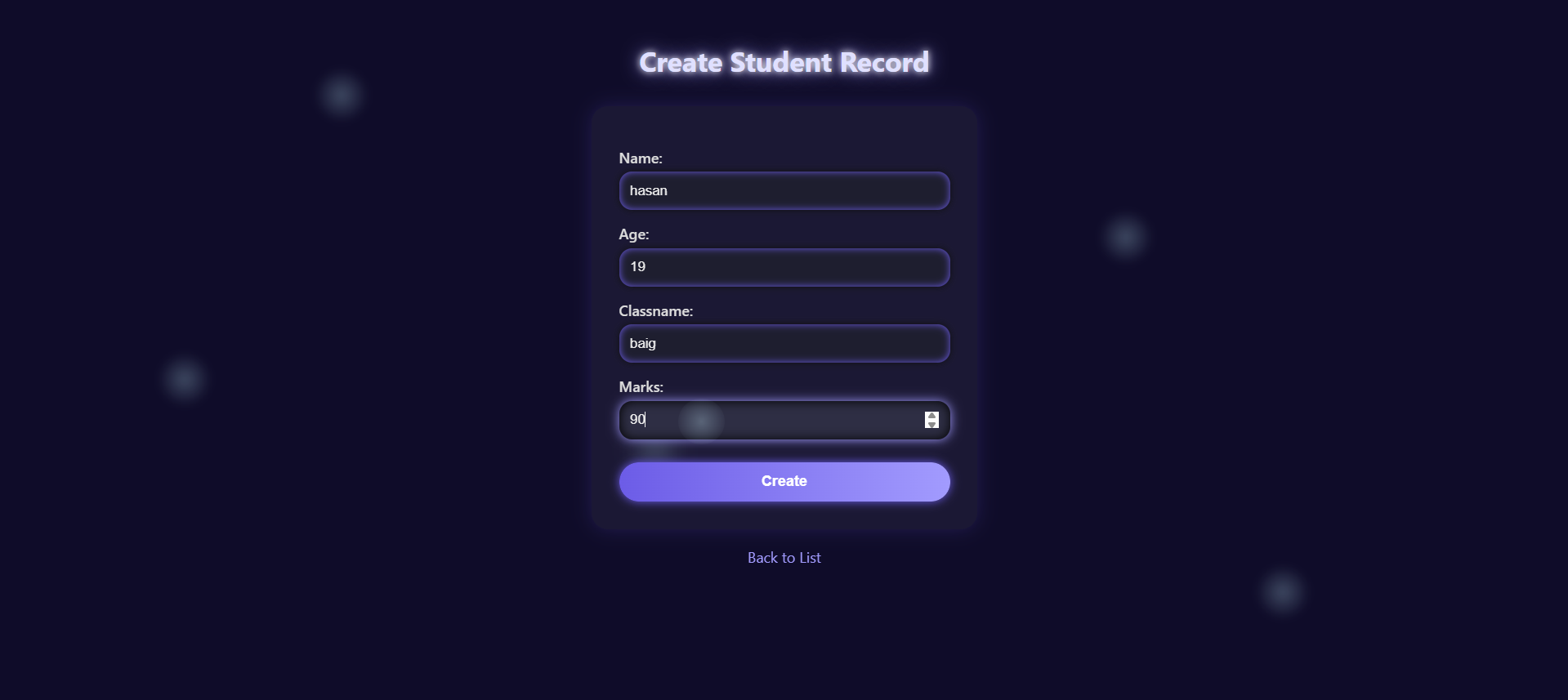
For example, the DefaultRouter in your **app/urls.py** registered the **BookViewSet**with the route books. So, the endpoints you can use are:

| **HTTP Method** | **Endpoint URL** | **Description** |
| --- | --- | --- |
| **GET** | http://127.0.0.1:8000/api/studentrecord/ | List all books |
| **POST** | http://127.0.0.1:8000/api/ studentrecord / | Create a new student |
| **GET** | http://127.0.0.1:8000/api/ studentrecord /{id}/ | Retrieve a student by its ID |
| **PUT** | http://127.0.0.1:8000/api/ studentrecord /{id}/ | Update a student by its ID |
| **DELETE** | http://127.0.0.1:8000/api/ studentrecord /{id}/ | Delete a student by its ID |

Final Overview:



Creating a Student record:



Created View:

