



BackEnd Workshop-3

Clarusway



Subject: Django CRUD Operations

Learning Goal

- Practice Django to implement CRUD (Create, Read, Update, Delete) operations.

Introduction

In this workshop, we will create a "Student Register" Django app.

- The root url will be a Student Register form. Users can submit a new student using this form.
- After submission, the user will be linked to "list" page. Or they can go to the list page with a direct link.
- In list page, users will see the "Full Name" and the "Path" of the student. Here there will be "Add New", "Update", and "Delete" buttons.
 - "Add New" will go to the "Student Register" form.
 - "Update" will go to the related students filled form, which can be updated.
 - "Delete" will erase the related student, and continue to show the list page.

Model:

In this project, you need to create a Student model with fields:

- fullname
- number
- mobile
- email

- gender
- path

You can use choices option with gender and path.

Form :

In this project, you need to create a forms.py file including Student form using Student model.

Views:

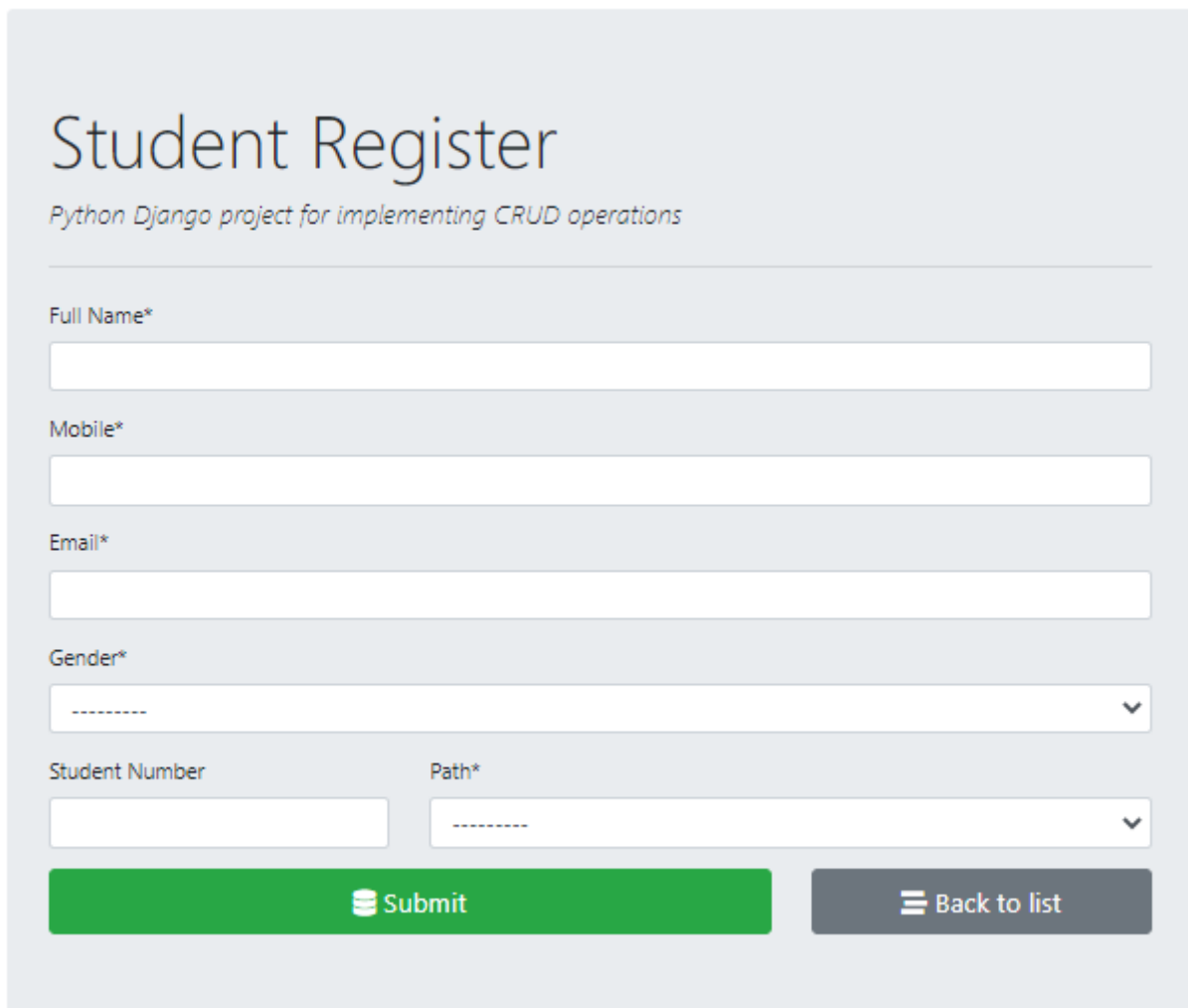
In this project, you need to create three views:

- student_add_update # Including Student form, to create and update student object.
- student_list # To read all student objects on the template.
- student_delete # To delete a student object.

Templates:

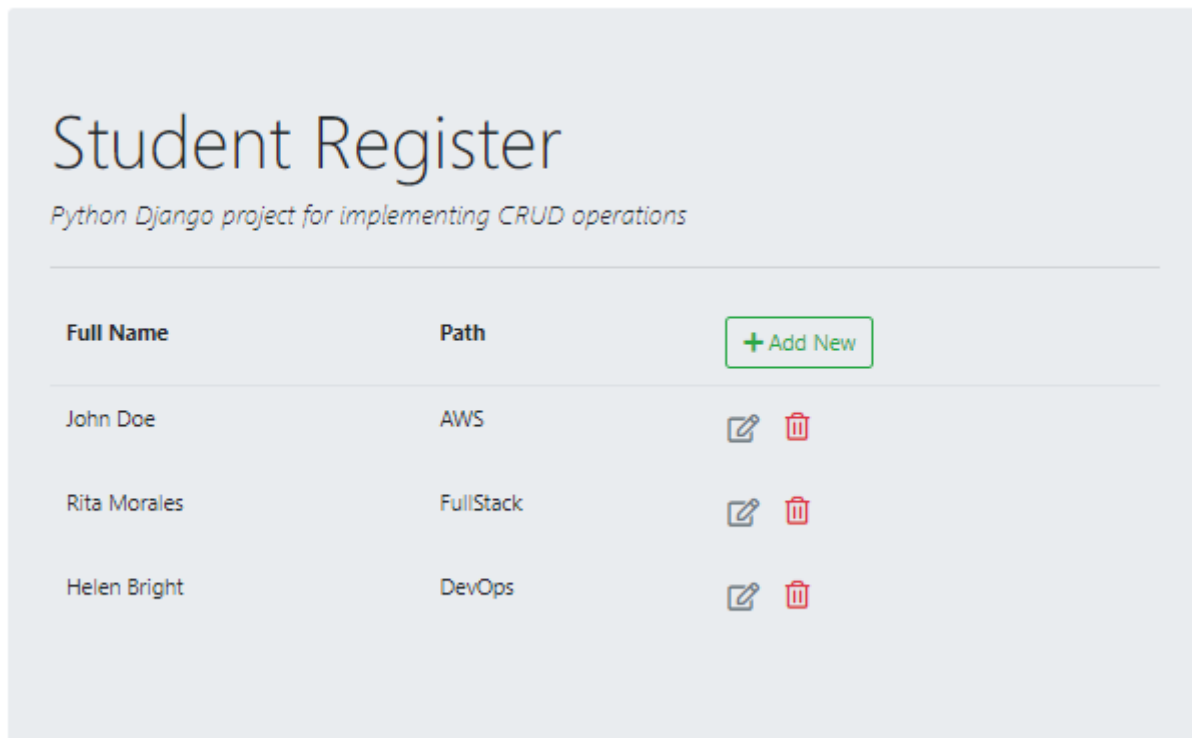
In this project, you need to create two templates:

- student_form.html



The screenshot shows a web form titled "Student Register" with the subtitle "Python Django project for implementing CRUD operations". The form is set against a light gray background. It contains several input fields: "Full Name*" (a single-line text box), "Mobile*" (a single-line text box), "Email*" (a single-line text box), "Gender*" (a dropdown menu with a dashed line as a placeholder), "Student Number" (a single-line text box), and "Path*" (a dropdown menu with a dashed line as a placeholder). At the bottom of the form, there are two buttons: a green "Submit" button with a database icon and a dark gray "Back to list" button with a hamburger menu icon.

- student_list.html



And one optional template for base.html.

URLs:

In this project, you need to create three urls:

- " : root path will return main page with student form.
- '<int:id>/' : While update, the same form will be displayed with student info of related id.
- 'delete/<int:id>/' : delete student.
- 'list/' : list view to show all student objects.

Project Folder Structure

At the end of the project, the folder sturcture will be like:

```
workshop3
├─ student_project
│  ├─ asgi.py
│  ├─ settings.py
│  ├─ urls.py
│  ├─ wsgi.py
│  └─ __init__.py
├─ student_register
│  └─ templates
│     └─ student_register
│        ├─ base.html
│        ├─ student_form.html
│        └─ student_list.html
├─ admin.py
├─ apps.py
├─ forms.py
├─ models.py
├─ tests.py
├─ urls.py
├─ views.py
└─ __init__.py
├─ .env
├─ .gitignore
├─ db.sqlite3
├─ manage.py
└─ requirements.txt
```

Code:

Part 1 - Create first project and app

1. Create a working directory, cd to new directory.
2. Create virtual environment as a best practice:

```
python3 -m venv env # for Windows or
python -m venv env # for Windows
virtualenv env # for Mac/Linux or;
virtualenv env -p python3 # for Mac/Linux
```

3. Activate scripts:

```
.\env\Scripts\activate # for Windows, may need to switch powershell on this
operation.
source env/bin/activate # for MAC/Linux
```

See the (env) sign before your command prompt.

4. Install django:

```
pip install django # or
py -m pip install django
```

5. (Optional) See installed python packages:

```
pip freeze

# you will see:

# asgiref==3.5.0
# Django==4.0.4
# sqlparse==0.4.2
# tzdata==2022.1

# If you see lots of things here, that means there is a problem with your virtual
env activation. Activate scripts again!
```

6. Create the requirements.txt on your working directory, send your installed packages to this file, requirements file must be up to date:

```
pip freeze > requirements.txt
# In this project we will not use this file. But this is a standard procedure to
learn.
``

<br>
7. Create project:

```py
django-admin startproject student_project .
With . it creates a single project folder.
Avoiding nested folders.

django-admin startproject student_project
Without . at the end it will create nested folders.
```

```
Another naming as "main":
django-admin startproject main .

Naming depends on the company, team and the project.
```

Various files has been created inside project folder "student\_project"!

- manage.py - A command-line utility that allows you to interact with your Django project
- \_\_init\_\_.py - An empty file that tells Python that the current directory should be considered as a Python package
- settings.py - Comprises the configurations of the current project like DB connections.
- urls.py - All the URLs of the project are present here
- wsgi.py - This is an entry point for your application which is used by the web servers to serve the project you have created.

8. (Optional) If you created your project without "." at the end, change the name of the project main directory as src to distinguish from subfolder with the same name:

```
Be careful to use your own folder names.
mv student_project src
```

9. Let's create first application:

Go to the same level with manage.py file if you create your project with nested folders at step 7:

```
cd student_project # or, if you changed;
cd src
``

Start app
```py
# Using app name as "student_register"
python manage.py startapp student_register # or
py -m manage.py startapp student_register

# Another naming:
python manage.py startapp home
py -m manage.py startapp home
```

10. Go to settings.py and add another line as below under INSTALLED_APPS:

```
'student_register',
```

11. Run your project:

```
python manage.py runserver # or  
py -m manage.py runserver
```

Go to <http://localhost:8000/> in your browser, and you should see Django rocket, which means you successfully created project.

The install worked successfully! Congratulations!

Part 2 - Create Student Model

1. Go to models.py and create your student model like below:

```
from django.db import models  
  
class Student(models.Model):  
    fullname = models.CharField(max_length=100)  
    number = models.CharField(max_length=15)  
    mobile = models.CharField(max_length=15)  
    email = models.CharField(max_length=50)  
    GENDER = (  
        ('Female', 'Female'),  
        ('Male', 'Male'),  
        ('Other', 'Other'),  
        ('Prefer', 'Prefer not to say')  
    )  
    gender = models.CharField(max_length=50, choices=GENDER)  
  
    PATH = (  
        ('Full Stack', 'Full Stack'),  
        ('Data Science', 'Data Science'),  
        ('DevOps', 'DevOps'),  
        ('AWS', 'AWS'),  
        ('ITF', 'ITF')  
    )  
    path = models.CharField(max_length=50, choices=PATH)  
  
    def __str__(self):  
        return self.fullname
```

2. See how to use choices option.

3. Discuss using other field options like EmailField.

4. Manage migrations:

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

Part 3 - Create Student Form

1. Create forms.py under student_register app and create your student form like below:

```
from django import forms
from .models import Student

class StudentFrom(forms.ModelForm):

    class Meta:
        model = Student
        fields = ('fullname', 'number', 'mobile', 'email', 'gender', 'path')
        labels = {
            'fullname': 'Full Name',
            'number': 'Student Number'
        }

    def __init__(self, *args, **kwargs):
        super(StudentFrom, self).__init__(*args, **kwargs)
        self.fields['path'].empty_label = "Select"
        self.fields['number'].required = False
```

2. Discuss about __init__ method. How we can overwrite ModelForm?

Part 4 - Create Views and URLs

1. Go to views.py under "student_register" directory, and create your views like below:

```
from django.shortcuts import render, redirect
from .forms import StudentFrom
from .models import Student

def student_list(request):
    context = {'student_list': Student.objects.all()}
    return render(request, "student_register/student_list.html", context)

def student_add_update(request, id=0):
    if request.method == "GET":
        if id == 0:
            form = StudentFrom()
        else:
```



```

        student = Student.objects.get(pk=id)
        form = StudentForm(instance=student)
        return render(request, "student_register/student_form.html", {'form':
form})
    else:
        if id == 0:
            form = StudentForm(request.POST)
        else:
            student = Student.objects.get(pk=id)
            form = StudentForm(request.POST,instance= student)
        if form.is_valid():
            form.save()
        return redirect('student_list')

def student_delete(request,id):
    student = Student.objects.get(pk=id)
    student.delete()
    return redirect('student_list')

```

2. Discuss about redirect function.

3. Include URL path of the new app to the project url list, go to urls.py and add:

```

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('student_register.urls')),
]

```

4. Add urls.py file under student_register directory and add:

```

from django.urls import path,include
from .views import student_add_update, student_delete, student_list

urlpatterns = [
    path('', student_add_update, name='student_add_update'), # get and post req.
for insert operation
    path('<int:id>/', student_add_update, name='student_update'), # get and post
req. for update operation
    path('delete/<int:id>/',student_delete,name='student_delete'),
    path('list/',student_list,name='student_list') # get req. to retrieve and
display all records
]

```

Part 5 - Install Crispy Forms and Create Templates

1. Install crispy forms according to this link:

[Crispy Forms Installation](#)

2. Create templates/student_register folder and under that folder, create base.html file as a best practice. Create this base template including bootstrap like below:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Student Register</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.11.2/css/all.min.css">
</head>

<body>
    <div class="container">
        <div class="col-md-10 offset-md-1 mt-5">
            <div class="jumbotron">
                <h1 class="display-4">Student Register</h1>
                <p class="lead font-italic">Python Django project for implementing
CRUD operations</p>
                <hr class="my-4"> {% block content %} {% endblock content %}
            </div>
        </div>
    </div>

    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-
q8i/X+965Dz00rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
integrity="sha384-
U02eT0CpHqdSJQ6hJty5KVphtPhzWj9W01c1HTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
```

```

integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy60rQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
</body>

</html>

```

3. Discuss using block tag.

4. Under templates/student_register folder create student_form.html as the second template, this will serve Student Form:

```

{% extends "student_register/base.html" %}

{% load crispy_forms_tags %}

{% block content %}

<form action="" method="post" autocomplete="off">
    {% csrf_token %}
    {{form.fullname|as_crispy_field}}
    {{form.mobile|as_crispy_field}}
    {{form.email|as_crispy_field}}
    {{form.gender|as_crispy_field}}
    <div class="row">
        <div class="col-md-4">
            {{form.number|as_crispy_field}}
        </div>
        <div class="col-md-8">
            {{form.path|as_crispy_field}}
        </div>
    </div>
    <div class="row">
        <div class="col-md-8">
            <button type="submit" class="btn btn-success btn-block btn-lg"><i
class="fas fa-database"></i>
            Submit</button>
        </div>
        <div class="col-md-4">
            <a href="{% url 'student_list' %}" class="btn btn-secondary btn-block
btn-lg">
                <i class="fas fa-stream"></i> Back to list
            </a>
        </div>
    </div>
</form>

{% endblock content %}

```

5. Under templates/student_register folder create student_list.html as the third template, this will show all students:

```
{% extends "student_register/base.html" %}

{% block content %}

<table class="table table-borderless">
  <thead class="border-bottom font-weight-bold">
    <tr>
      <td>Full Name</td>
      <td>Path</td>
      <td>
        <a href="{% url 'student_add_update' %}" class="btn btn-outline-
success">
          <i class="fas fa-plus"></i> Add New
        </a>
      </td>
    </tr>
  </thead>
  <tbody>
    {% for student in student_list %}
      <tr>
        <td>{{student.fullname}}</td>
        <td>{{student.path}}</td>
        <td>
          <a href="{% url 'student_update' student.id %}" class="btn text-
secondary px-0">
            <i class="far fa-edit fa-lg"></i>
          </a>
          <form action="{% url 'student_delete' student.id %}" method="post"
class="d-inline">
            {% csrf_token %}
            <button type="submit" class="btn">
              <i class="far fa-trash-alt fa-lg text-danger float-
right"></i>
            </button>
          </form>
        </td>
      </tr>
    {% endfor %}
  </tbody>
</table>

{% endblock content %}
```

6. Run our project again to see our view:

```
python manage.py runserver  
py -m manage.py runserver
```

7. Go to <http://localhost:8000/> in your browser, and check the functionality of your website. To stop the server use "CTRL + C"

8. If you want to send this project to your Github repo, do not forget to add .gitignore file, and secure your sensitive information like keys storing them locally in .env file and using python-decouple module.

9. Update the requirements.txt file after all your installations!

😊 **Thanks for Attending** 🙌

Clarusway

