**Django CRUD (Create Read Update Delete) Example**

**Prerequisites before using Django-**

Applications need to install- Python and python virtual environment

1. Download and Install python

2. Create virtual environment(pip should be installed(check version- pip --version))

Install either virtualenvwrapper or virtualenv(for more info goto- [link](https://docs.google.com/document/d/1NEI7Oq5KY3wwqz0nZEh0V8ZSKQagPyO92W5WEhBaEbI/edit))

install virtual environment- pip install virtualenv

Create virtual environment- virtualenv <virtual-env>

Now activate virtual env - cd <virtual-env>/Scripts/activate

3. Install Django using command -

pip install django

Check django version command-

python -m django --version

**To create a Django application that performs CRUD operations, follow the following steps.**

1. Create a Project

$ django-admin startproject crudexample

2. Create an App

$ python3 manage.py startapp employee

3**. Project Structure**

Initially, our project looks like this:

crudexample

\_\_init\_\_.py

settings.py

urls.py

wsgi.py

employee

migrations

\_\_init\_\_.py

\_\_init\_\_.py

urls.py

admin.py

apps.py

models.py

tests.py

views.py

manage.py

**4. Database Setup**

Create a database djangodb in mysql, and configure into the settings.py file of django project. See the example.

// settings.py

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.mysql',

        'NAME': 'djangodb',

        'USER':'root',

        'PASSWORD':'mysql',

        'HOST':'localhost',

        'PORT':'3306'

    }

}

**5. Create a Model**

Put the following code into models.py file.

// models.py

from django.db import models

class Employee(models.Model):

    eid = models.CharField(max\_length=20)

    ename = models.CharField(max\_length=100)

    eemail = models.EmailField()

    econtact = models.CharField(max\_length=15)

    class Meta:

        db\_table = "employee"

**6. Create a ModelForm**

// forms.py

from django import forms

from employee.models import Employee

class EmployeeForm(forms.ModelForm):

    class Meta:

        model = Employee

        fields = "\_\_all\_\_"

**7. Create View Functions**

// views.py

from django.shortcuts import render, redirect

from employee.forms import EmployeeForm

from employee.models import Employee

Create your views here.

**def emp(request):**

    if request.method == "POST":

        form = EmployeeForm(request.POST)

        if form.is\_valid():

            try:

                form.save()

                return redirect('/show')

            except:

                pass

    else:

        form = EmployeeForm()

    return render(request,'index.html',{'form':form})

**def show(request):**

    employees = Employee.objects.all()

    return render(request,"show.html",{'employees':employees})

**def edit(request, id):**

    employee = Employee.objects.get(id=id)

    return render(request,'edit.html', {'employee':employee})

**def update(request, id):**

    employee = Employee.objects.get(id=id)

    form = EmployeeForm(request.POST, instance = employee)

    if form.is\_valid():

        form.save()

        return redirect("/show")

    return render(request, 'edit.html', {'employee': employee})

**def destroy(request, id):**

    employee = Employee.objects.get(id=id)

    employee.delete()

    return redirect("/show")

**8. Provide Routing**

Provide URL patterns to map with views function.

// urls.py

from django.contrib import admin

from django.urls import path

from employee import views

urlpatterns = [

    path('admin/', admin.site.urls),

 path('', include('employee.urls')),

]

**8.1 Creating routing for employee urls**

// employee/urls.py

from django.urls import path  
from . import views  
  
urlpatterns = [  
 path('', views.show),  
 path('emp', views.emp),  
 path('show', views.show),  
 path('edit/<int:id>', views.edit),  
 path('update/<int:id>', views.update),  
 path('delete/<int:id>', views.destroy),  
]

**9. Organize Templates**

create a base template for all files- templates/base.html

To tell django about this file, do some changes in settings.py-

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [os.path.join(BASE\_DIR, 'templates')],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.debug',

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

Create a templates folder inside the employee app and create three (index, edit, show) html files inside the directory. The code for each is given below.

// index.html

{% extends 'base.html' %}  
  
{% block content %}  
<form method="POST" class="post-form" action="/emp">  
 {% csrf\_token %}  
 <div class="container">  
<br>  
 <div class="form-group row">  
 <label class="col-sm-1 col-form-label"></label>  
 <div class="col-sm-4">  
 <h3>Enter Details</h3>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label class="col-sm-2 col-form-label">Employee Id:</label>  
 <div class="col-sm-4">  
 {{ form.eid }}  
 </div>  
 </div>  
 <div class="form-group row">  
 <label class="col-sm-2 col-form-label">Employee Name:</label>  
 <div class="col-sm-4">  
 {{ form.ename }}  
 </div>  
 </div>  
 <div class="form-group row">  
 <label class="col-sm-2 col-form-label">Employee Email:</label>  
 <div class="col-sm-4">  
 {{ form.eemail }}  
 </div>  
 </div>  
 <div class="form-group row">  
 <label class="col-sm-2 col-form-label">Employee Contact:</label>  
 <div class="col-sm-4">  
 {{ form.econtact }}  
 </div>  
 </div>  
 <div class="form-group row">  
 <label class="col-sm-1 col-form-label"></label>  
 <div class="col-sm-4">  
 <button type="submit" class="btn btn-primary">Submit</button>  
 </div>  
 </div>  
 </div>  
</form>  
{% endblock content %}

// show.html

{% extends 'base.html' %}  
  
{% block content %}  
<table class="table table-striped table-bordered table-sm">  
 <thead class="thead-dark">  
 <tr>  
 <th>Employee ID</th>  
 <th>Employee Name</th>  
 <th>Employee Email</th>  
 <th>Employee Contact</th>  
 <th>Actions</th>  
 </tr>  
 </thead>  
 <tbody>  
{% for employee in employees %}  
 <tr>  
 <td>{{ employee.eid }}</td>  
 <td>{{ employee.ename }}</td>  
 <td>{{ employee.eemail }}</td>  
 <td>{{ employee.econtact }}</td>  
 <td>  
 <a href="/edit/{{ employee.id }}"><span class="glyphicon glyphicon-pencil" >Edit</span></a>  
 <a href="/delete/{{ employee.id }}">Delete</a>  
 </td>  
 </tr>  
{% endfor %}  
 </tbody>  
</table>  
<br>  
<br>  
<center>  
 <a href="/emp" class="btn btn-primary">Add New Record</a>  
</center>  
{% endblock content %}

// edit.html

{% extends 'base.html' %}  
  
{% block content %}  
<form method="POST" class="post-form" action="/update/{{employee.id}}">  
 {% csrf\_token %}  
 <div class="container">  
<br>  
 <div class="form-group row">  
 <label class="col-sm-1 col-form-label"></label>  
 <div class="col-sm-4">  
 <h3>Update Details</h3>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label class="col-sm-2 col-form-label">Employee Id:</label>  
 <div class="col-sm-4">  
 <input type="text" name="eid" id="id\_eid" required maxlength="20" value="{{ employee.eid }}"/>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label class="col-sm-2 col-form-label">Employee Name:</label>  
 <div class="col-sm-4">  
 <input type="text" name="ename" id="id\_ename" required maxlength="100" value="{{ employee.ename }}" />  
 </div>  
 </div>  
 <div class="form-group row">  
 <label class="col-sm-2 col-form-label">Employee Email:</label>  
 <div class="col-sm-4">  
 <input type="email" name="eemail" id="id\_eemail" required maxlength="254" value="{{ employee.eemail }}" />  
 </div>  
 </div>  
 <div class="form-group row">  
 <label class="col-sm-2 col-form-label">Employee Contact:</label>  
 <div class="col-sm-4">  
 <input type="text" name="econtact" id="id\_econtact" required maxlength="15" value="{{ employee.econtact }}" />  
 </div>  
 </div>  
 <div class="form-group row">  
 <label class="col-sm-1 col-form-label"></label>  
 <div class="col-sm-4">  
 <button type="submit" class="btn btn-success">Update</button>  
 </div>  
 </div>  
 </div>  
</form>  
{% endblock content %}

**10. Static Files Handling**

changes to do at the end of settings.py-

STATICFILES\_DIRS = [

    os.path.join(BASE\_DIR, 'static')

]

STATIC\_ROOT = os.path.join(BASE\_DIR, 'assets')

and now tell djnago about your css file, run command-

python manage.py collectstatic

(it will create a folder assets in main directory with your css file in it. You should run this command always after making changes in your css, js or images.)

Create a folder static/css/style.css inside the employee app and put a css inside it.

Css-

body {font:12px/1.4 Verdana,Arial; background:#A9A9A9; height:100%; margin:25px 0; padding:0}  
h1 {font:24px Georgia,Verdana; margin:0}  
h2 {font-size:12px; font-weight:normal; font-style:italic; margin:0 0 20px}  
p {margin-top:0}  
ul {margin:0; padding-left:20px}  
  
#testdiv {width:600px; margin:0 auto; border:1px solid #ccc; padding:20px 25px; background:#fff}  
  
#tinybox {position:absolute; display:none; padding:10px; background:#fff url(images/preload.gif) no-repeat 50% 50%; border:10px solid #e3e3e3; z-index:2000}  
#tinymask {position:absolute; display:none; top:0; left:0; height:100%; width:100%; background:#000; z-index:1500}  
#tinycontent {background:#fff}  
  
.button {font:14px Georgia,Verdana; margin-bottom:10px; padding:8px 10px 9px; border:1px solid #ccc; background:#eee; cursor:pointer}  
.button:hover {border:1px solid #bbb; background:#e3e3e3}

**11. Project Structure**

Crudexample->

\_\_init\_\_.py

settings.py

urls.py

wsgi.py

templates->

base.html

employee->

**migrations->**

\_\_init\_\_.py

\_\_init\_\_.py

urls.py

admin.py

apps.py

models.py

tests.py

views.py

static->

css->

style.css

templates->

index.html

show.html

edit.html

manage.py

**12. Create Migrations**

Create migrations for the created model employee, use the following command.

$ python3 manage.py makemigrations

After migrations, execute one more command to reflect the migration into the database. But before it, mention name of app (employee) in INSTALLED\_APPS of settings.py file.

// settings.py

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'employee'

]

**Run the command to migrate the migrations.**

$ python3 manage.py migrate

Now, our application has successfully connected and created tables in database. It creates 10 default tables for handling project (session, authentication etc) and one table of our model that we created.

See list of tables created after migrate command.

**Run Server**

To run server use the following command.

**$ python3 manage.py runserver**

**Access to the Browser**

Access the application by entering localhost:8000/, it will show all the available employee records.

Initially, there is no record. So, it shows no record message.

Well, we have successfully created a CRUD application using Django.

This complete project can be downloaded [here](https://github.com/diwamishra21/Django-crud-application).( <https://github.com/diwamishra21/Django-crud-application>)

**Debugging-**

1. MySQL connection Error-

Solution-

django.core.exceptions.ImproperlyConfigured: Error loading MySQLdb module.

Did you install mysqlclient?

Ans-

pip install pymysql

Then, edit the \_\_init\_\_.py file in your project origin dir(the same as settings.py)

add:

import pymysql

pymysql.install\_as\_MySQLdb()

2. 'staticfiles' is not a registered tag library

Solution-

It's due to upgrading to Django3.0, use as mentioned above.

use-

{% load static %}