**Django CRM Application**

**Basic Set Up:**

1. **Git Hub Set Up:**

* Go to your git hub account
* Create a new repository
* Open the created repo and copy the link of this repo

1. **Visual Studio Set Up:**

* Open Vs code and open any empty folder from pc
* Clone the create repo by git command -> git clone copied link
* Initialize git -> git init

1. **Virtualenv Set Up:**

* In vs terminal install virtual environment -> pip install virtualenv
* Create a virtualenv file -> virtualenv venv
* Activate venv -> venv\scripts\activate
* To deactivate this venv -> deactivate

1. **Install Django:**

* In terminal install Django while venv is activated -> pip install Django
* To see the list of packages -> pip list

1. **Django Create Project Set Up:**

* In terminal create a Django project -> Django-admin startproject project­­\_name
* cd into your project to run server
* To run server -> python manage.py runserver

1. **Django Create App Set Up:**

* In terminal to create a Django app -> Django-admin startapp app\_name
* Register your app in setting.py of your main project folder
* Go to settings and find installed apps and register there -> ‘app\_name’,
* Now in terminal migrate it to save your app -> python manage.py migrate

1. **Create Super User for Admin Login:**

* Create a super user to access Django admin pannel gui –> python manage.py createsuperuser
* Give username, enter to skip email, set password twice
* Super User Created

1. **Create and Configure urls.py file:**

* Create a urls.py file in your app and configure the url
* Include app url in main project url
* Create a url route to test and create a view as http response to see it is running

1. **Create Static Folder and Configure in Settings:**

* Create a static folder inside the main project
* Create 2 sub folders css and js
* In settings of main project scroll down to static and configure -> STATICFILES\_DIRS = [BASE\_DIR / ‘static’ ]

1. **Templates Set Up:**

* Two types project level and app level templates
* Project level is configuring templates in project settings under templates -> ‘DIRS’ : [BASE\_DIR, ‘templates’ ]
* App level is creating folder named templates inside app folder
* Also create sub folder with same as app name inside templates folder of app level

1. **Simple Views:**

* from django.shortcuts import render
* from django.http import HttpResponse
* def home(request):
* return render(request, 'crud\_app/index.html')

1. **Base HTML File(DRY):**

* Create a base html in templates
* This is a DRY page don’t repeat yourself
* This is written once and extended in other html pages

{% load static %} = this tag is used to extend or render the css files in html page

{% block content %}

<!-- Your content goes here -->

{% endblock %}

* This above block content will be used in base html page to inherit the functionality of this base html into other pages
* To extend the functionalities in other pages from base below code is used

{% extends 'crud\_app/base.html' %}

{% block content %}

<h1>Hello World</h1>

{% endblock %}

1. **Bootstrap Themes:**

* Use bootswatch.com to use free themes provided by bootstrap

1. **Create Required Pages:**

* Create remaining html pages
  + **Register page** for new user register
  + **Login page** for user login
  + **Dashboard page** for admin to control
  + **Create Record page** to create user record
  + **View Record page** to view or display the created record

1. **Create Forms:**

* Create a form in your app
* Forms are created to take user data
* Import forms built in with Django itself

from django.contrib.auth.forms import UserCreationForm, AuthenticationForm # used to creeate form with authentication

from django.contrib.auth.models import User  # used for registering username and password

from django.forms.widgets import PasswordInput, TextInput, EmailInput  # used for password and text input based on imported form

from django import forms

* Do the rest thing and create login credentials or refer the source code

1. **Index Page Set Up:**

* Import DRY code in index page and design according to your need

1. **Navbar page**

* Create new page navbar.html in templates
* Go to getbootstrap and copy the code for navbar and paste it in navbar page and modify
* To render this navbar in base html page just use Django dry code and link the navbar url like this

<!--create navbar here-->

{% include 'crud\_app/navbar.html' %}

1. **Create Register Form in View:**

* Import forms created in forms.py in views to create a post register form
* Write the logic in the register view refer code below

#Register form rendering in register page

def register(request):

    form = CustomUserCreationForm()                     #import the form in a variable

    if request.method == 'POST':                        #check whether the request method is post

        form = CustomUserCreationForm(request.POST)     #post all the data filled into this form

        if form.is\_valid():                             #verifying whether the form is valid or not

            form.save()                                 #posting or saving the form details in database

            return redirect('login\_page')               #redirecting to login page if form is valid

    context = {'form':form}                             #creating context of form to render on html page

    return render(request, 'crud\_app/register\_page.html', context)

* Make a form for user to register as post method as it is sending data to database
* If any form is post method then csrf token should be added in that form in Django as this

{% csrf\_token %}

* Create a css for form to look good
* Install Django crispy forms to alter the input field boxes to look good
* Install older version to work properly -> pip install django-crispy-forms==1.14.0
* Go to settings file to add in installed apps as -> ‘crispy\_forms’
* Also add crispy template pack as bootstrap -> CRISPY\_TEMPLATE\_PACK = ‘bootstrap4’
* To use this add it above the block content -> {% load crispy\_forms\_tags %} and add crispy in the form by removing as\_p -> {{form | crispy }}

1. **Create Login in view:**

#User registration form handling

def login(request):

    form = LoginForm()                            #import the form in a variable

    if request.method == "POST":

        form = LoginForm(request, data=request.POST)

        if form.is\_valid():

            username = request.POST.get('username')

            password = request.POST.get('password')

            user = authenticate(request, username=username, password=password)

            if user is not None:

                auth.login(request, user)

                messages.success(request, 'You Are Now Logged In Successfully!')

                return redirect('dashboard')

            else:

                messages.info(request, 'Username or Password is incorrect')

                return redirect('login')

        else:

            messages.info(request, 'Username not registered, Register Now!')

            return redirect('register')

    context = {'form':form}

    return render(request, 'crud\_app/login\_page.html', context)

* Use this form in frontend to take the user data and authenticate to login the user
* To use this form just use the Django syntax like this -> {{ form.as\_p }}
* This will render the form which is built in inside Django

1. **Create Logout View:**

#Logout user

def logout(request):

    auth.logout(request)

    messages.success(request, 'You havve been logged out!')

    return redirect('/')

* Create its url as well and just link the url in the logout button
* On clicking the button it will logout the user

**CRUD OPERATIONS**

1. **Create Records Model and Display In Frontend:**

* Create a model to create user record
* Register this model in the admin.py file

from django.db import models

# Create your models here.

from django.db import models

# Create your models here.

class CreateRecord(models.Model):

    date\_create = models.DateField(auto\_now\_add=True)

    first\_name = models.CharField(max\_length=100)

    last\_name = models.CharField(max\_length=100)

    email = models.EmailField(max\_length=255)

    phone = models.CharField(max\_length=25)

    address = models.CharField(max\_length=255)

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

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

    country = models.CharField(max\_length=120)

    def \_\_str\_\_(self):

        return f'{self.first\_name} {self.last\_name}'

* In views take all the records in an object and show in frontend

#user dashboard

@login\_required(login\_url='login')

def dashboard(request):

    my\_records = CreateRecord.objects.all()

    context = {'my\_records': my\_records}

    return render(request, 'crud\_app/dashboard.html', context)

* create a table with a logic to display the details of user or employee created

<div class="container">

<div class="row">

<center>

<div class="col-8"><br/><br/>

{% if my\_records %}

<table class="table table-bordered border-primary table-primary table-hover table-striped">

<thead class="table-light">

<tr>

<th scope="col">Employee Id</th>

<th scope="col">Employee Name</th>

<th scope="col">Employee Email</th>

<th scope="col">Employee Email</th>

<th scope="col">Employee Phone</th>

<th scope="col">Date Joined</th>

<th scope="col">View</th>

</tr>

</thead>

<tbody class="table-group-divider">

{% for records in my\_records %}

<tr>

<td>{{records.id}}</td>

<td>{{records.first\_name}}</td>

<td>{{records.last\_name}}</td>

<td>{{records.email}}</td>

<td>{{records.phone}}</td>

<td>{{records.date\_create}}</td>

</tr>

{% endfor %}

</tbody>

</table>

{% else %}

<h1>No Records Found!</h1>

{% endif %}

</div>

</center>

</div>

</div>

</body>

1. **Create a Create Record Operation:**

* Import the model created in froms.py and create a form to create record

#Create Record Form

class CreateRecordForm(forms.ModelForm):

    class Meta:

        model = CreateRecord

        fields = ['first\_name', 'last\_name', 'email', 'phone', 'address', 'city', 'province', 'country']

* Create a view to import that form and validate

@login\_required(login\_url='login')

def createrecord(request):

    form = CreateRecordForm()

    if request.method == 'POST':

        form = CreateRecordForm(request.POST)

        if form.is\_valid():

            form.save()

            return redirect('dashboard')

    context = {'form':form}

    return render(request, 'crud\_app/create\_record.html', context)

* Create a url for this view

urlpatterns = [

    path('', views.index, name="index"),

    path('login/', views.login, name="login"),

    path('logout/', views.logout, name="logout"),

    path('register/', views.register, name="register"),

    path('dashboard/', views.dashboard, name="dashboard"),

    path('create\_record/', views.createrecord, name="createrecord"),    #<<------

]

* Create a html page and display the form

<div class="container bg-light p-5 shadow-md form-layout">

    <div class="col-md-8 offset-md-2">

    <form method="POST" autocomplete="off">

        {% csrf\_token %}

        {{form|crispy}}

        <br/>

        <button type="submit" class="btn btn-primary w-100 btn-block p2">Create</button>

    </form>

    </div>

</div>

1. **Create a Update Record Operation:**

* Create a form in forms and import the Create Record form

#Upodate Record Form

class UpdateRecordForm(forms.ModelForm):

    class Meta:

        model = CreateRecord

        fields = ['first\_name', 'last\_name', 'email', 'phone', 'address', 'city', 'province', 'country']

* Create a dynamic url to update record

    path('update\_record/<int:pk>', views.updaterecord, name="updaterecord"),

* Create a view for update record by passing primary key as pk used in url

#Update a record

@login\_required(login\_url='login')

def updaterecord(request, pk):

    record = CreateRecord.objects.get(id=pk)

    form = UpdateRecordForm(instance=record)

    if request.method == 'POST':

        form = UpdateRecordForm(request.POST, instance=record)

        if form.is\_valid:

            form.save()

            messages.success(request, 'Updated Successfully!')

            return redirect('dashboard')

    context = {'form':form}

    return render(request, 'crud\_app/update\_record.html', context)

* Create a page for update record and use that updated form

1. **Create View Individual Record:**

* Create a dynamic url for this to view single record

    path('view\_record/<int:pk>', views.viewrecord, name="viewrecord"),

* Create a view for this

#View individual records

@login\_required(login\_url='login')

def viewrecord(request, pk):

    all\_records = CreateRecord.objects.get(id=pk)

    context = {'records': all\_records}

    return render(request, 'crud\_app/view\_record.html', context)

* Create a page to view this and add back button, update button with id, and delete button

<div class="card w-50 text-bg-light mb-3 col-md-6 offset-md-3" style="width: 25rem;">

        <div class="card-header fs-3 text-uppercase">

          <strong>{{records.first\_name}} {{records.last\_name}}</strong>

        </div>

        <div class="card-body fs-5">

          <h5 class="card-title"><b>Employee Details</b></h5></br>

          <p class="card-text"><strong> ID : </strong>{{records.id}}</p>

          <p class="card-text"><strong> Email : </strong>{{records.email}}</p>

          <p class="card-text"><strong> Creation Date : </strong>{{records.date\_create}}</p>

          <p class="card-text"><strong> Address : </strong>{{records.address}}</p>

          <p class="card-text"><strong> City : </strong>{{records.city}}</p>

          <p class="card-text"><strong> Province : </strong>{{records.province}}</p>

          <p class="card-text"><strong> Country : </strong>{{records.country}}</p></br>

          <a href="{% url 'dashboard' %}" class="btn btn-seondary">Back</a>

          <a href="{% url 'updaterecord' records.id %}" class="btn btn-outline-info">Update Record</a>

          <a href="#" class="btn btn-outline-danger">Delete</a>

        </div>

      </div>

1. **Create Delete Function:**

@login\_required(login\_url='login')

def deleterecord(request, pk):

    record = CreateRecord.objects.get(id=pk)

    record.delete()

    messages.error(request, 'Record Deleted')

    return redirect('dashboard')