**PROJECT TITLE**: CRM PROJECT USING DJANGO

**TASK 1**: DOCUMENTATION FOR CODE FLOW OF PROJECT

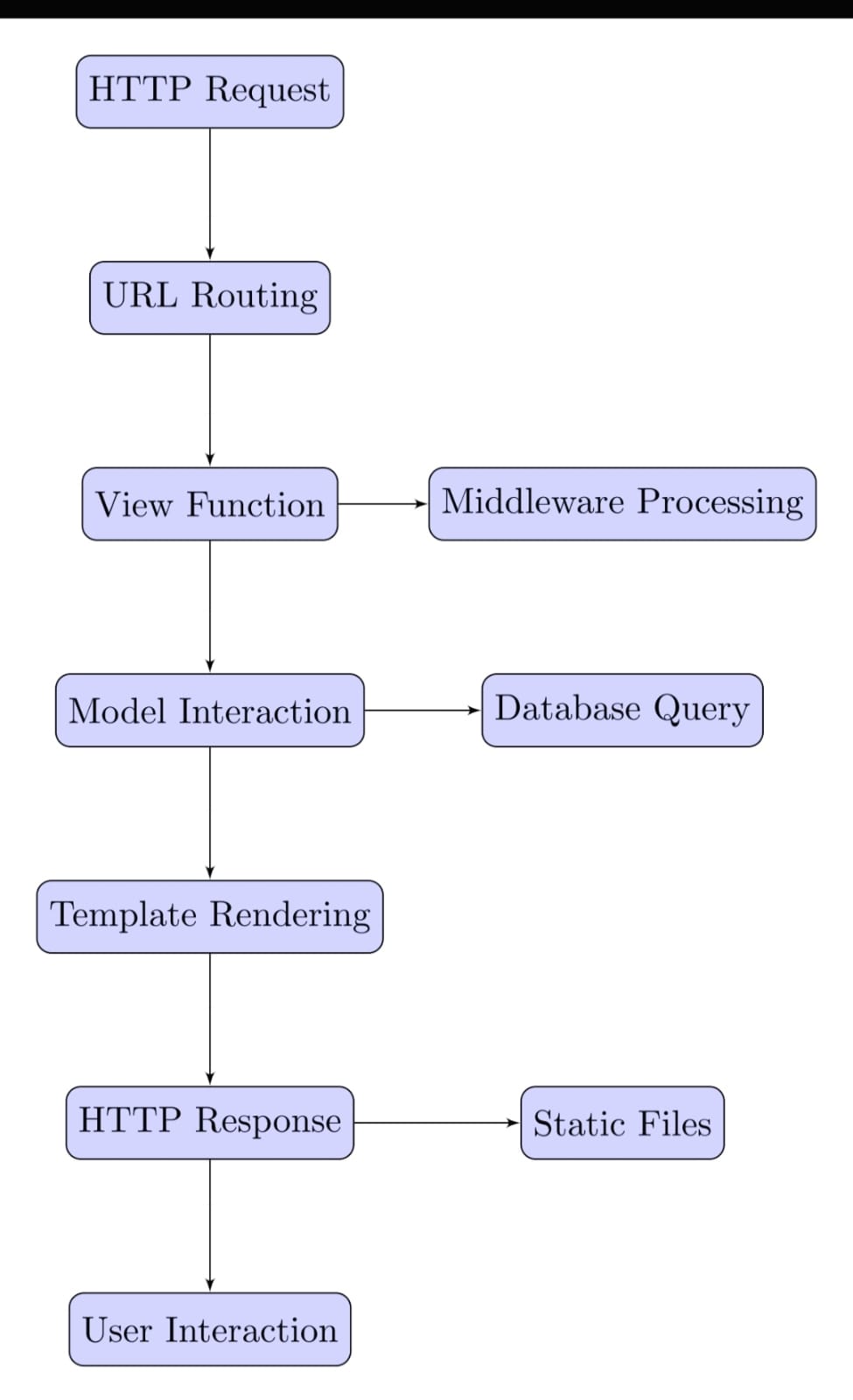
PRESENTED BY:

HARSHAVARDHAN MANAVALAN

**TABLE OF CONTENTS:**

1. Flow chart of django framework
2. Description of CRM Project django framework
3. Code flow

**FLOW CHART OF DJANGO FRAMEWORK:**



**Description of CRM Project django framework:**

**Request Handling:**

The process begins when a user sends an HTTP request to the Django web server.

**URL Routing:**

Django uses a URL dispatcher to match the incoming URL with a view function.

The **urls.py** file in each Django app defines the URL patterns and maps them to corresponding views.

from django.urls import path

from . import views

urlpatterns = [

    path('', views.home, name='home'),

    #path('login/', views.login\_user, name='login'),

    path('logout/', views.logout\_user, name='logout'),

    path('register/', views.register\_user, name='register'),

    path('record/<int:pk>', views.customer\_record, name='record'),

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

    path('add\_record/', views.add\_record, name='add\_record'),

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

]

**View Function:**

Once the URL is matched, the associated view function is called.

Views handle the request and return an HTTP response, often rendering a template.

* def home(request):
* def logout\_user(request):
* def register\_user(request):
* def customer\_record(request, pk):
* def delete\_record(request, pk):
* def add\_record(request):
* def update\_record(request, pk):

**Template Rendering:**

If the view returns an HTML template, Django processes it to generate dynamic content.

Templates may include variables, control structures, and template tags to render dynamic data.

* add\_record.html
* base.html(bootstrap and django code)
* home.html(also for login page )
* navbar.html
* record.html
* register.html
* update\_record.html

**Model Interaction:**

Views often interact with Django models to retrieve or update data in the database.

Models represent the data structure and are defined in the **models.py** file.

from django.db import models

class Record(models.Model):

    created\_at = models.DateTimeField(auto\_now\_add=True)

    first\_name = models.CharField(max\_length=50)

    last\_name =  models.CharField(max\_length=50)

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

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

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

    city =  models.CharField(max\_length=50)

    state =  models.CharField(max\_length=50)

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

    def \_\_str\_\_(self):

        return(f"{self.first\_name} {self.last\_name}")

**Database Query:**

When interacting with models, Django ORM (Object-Relational Mapping) translates high-level code into SQL queries.

These queries are executed on the database to fetch or modify data.

**Form Handling:**

If the view involves form submission, Django handles form validation, processing, and saving data to the database.

Forms are defined in the **forms.py** file within each app.

**Middleware Processing:**

Middleware components can intercept and process requests or responses globally.

Examples include authentication, security, and caching middleware.

**Response Generation:**

The final response is generated by the view and can include HTML content, JSON, or other formats.

**HTTP Response:**

The HTTP response is sent back to the user's browser.

**Static Files:**

Static files (CSS, JavaScript, images) are served by Django or a separate server in production.

**User Interaction:**

The user interacts with the web page based on the rendered content.

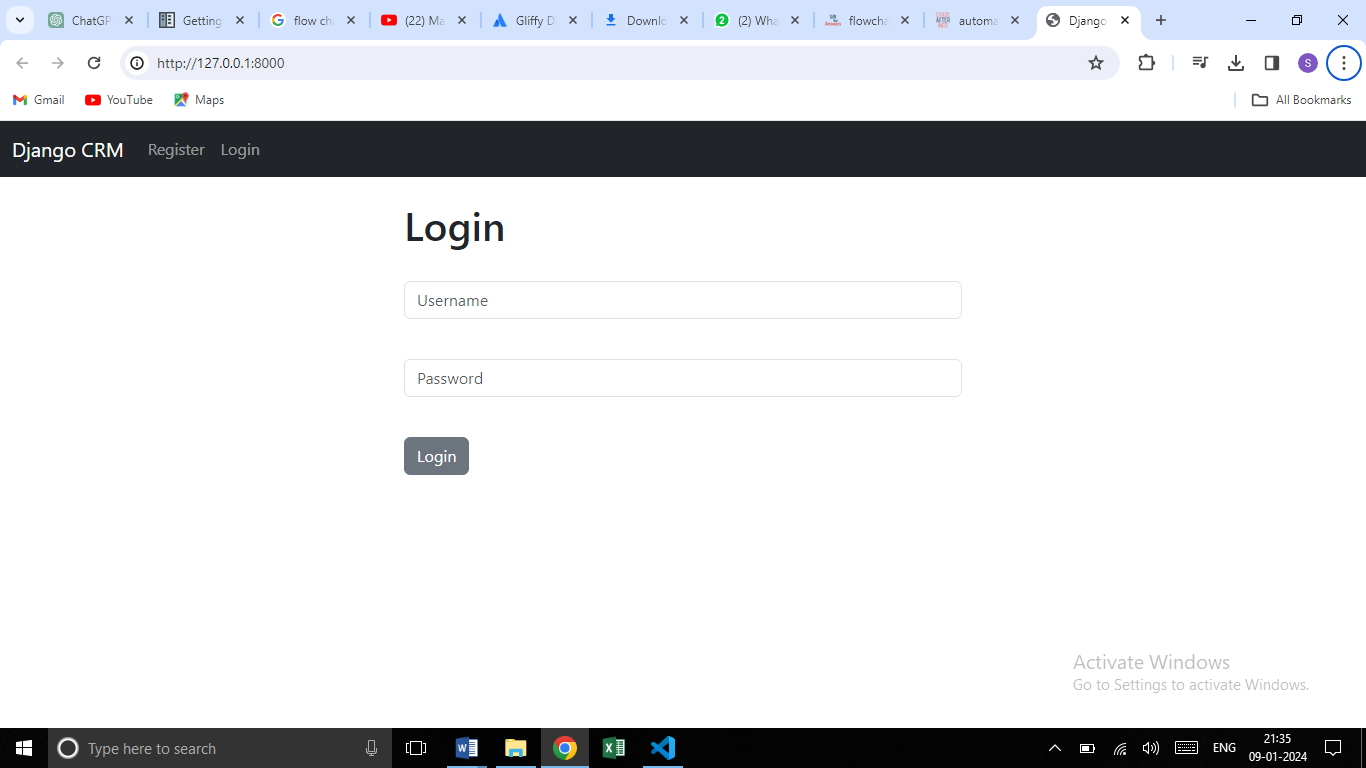
**Repeat:**

The process repeats for each user interaction, creating a dynamic and responsive web application.

**Code flow:**

When **python manage.py runserver,** manage.py runs and **main()** runs start processing and shows Starting development server at <http://127.0.0.1:8000/>

The website shows



When login admin user is logged in, def home(request): runs

def home(request):

    records = Record.objects.all()

    # Check to see if logging in

    if request.method == 'POST':

        username = request.POST['username']

        password = request.POST['password']

        # Authenticate

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

        if user is not None:

            login(request, user)

            messages.success(request, "You Have Been Logged In!")

            return redirect('home')

        else:

            messages.success(request, "There Was An Error Logging In, Please Try Again...")

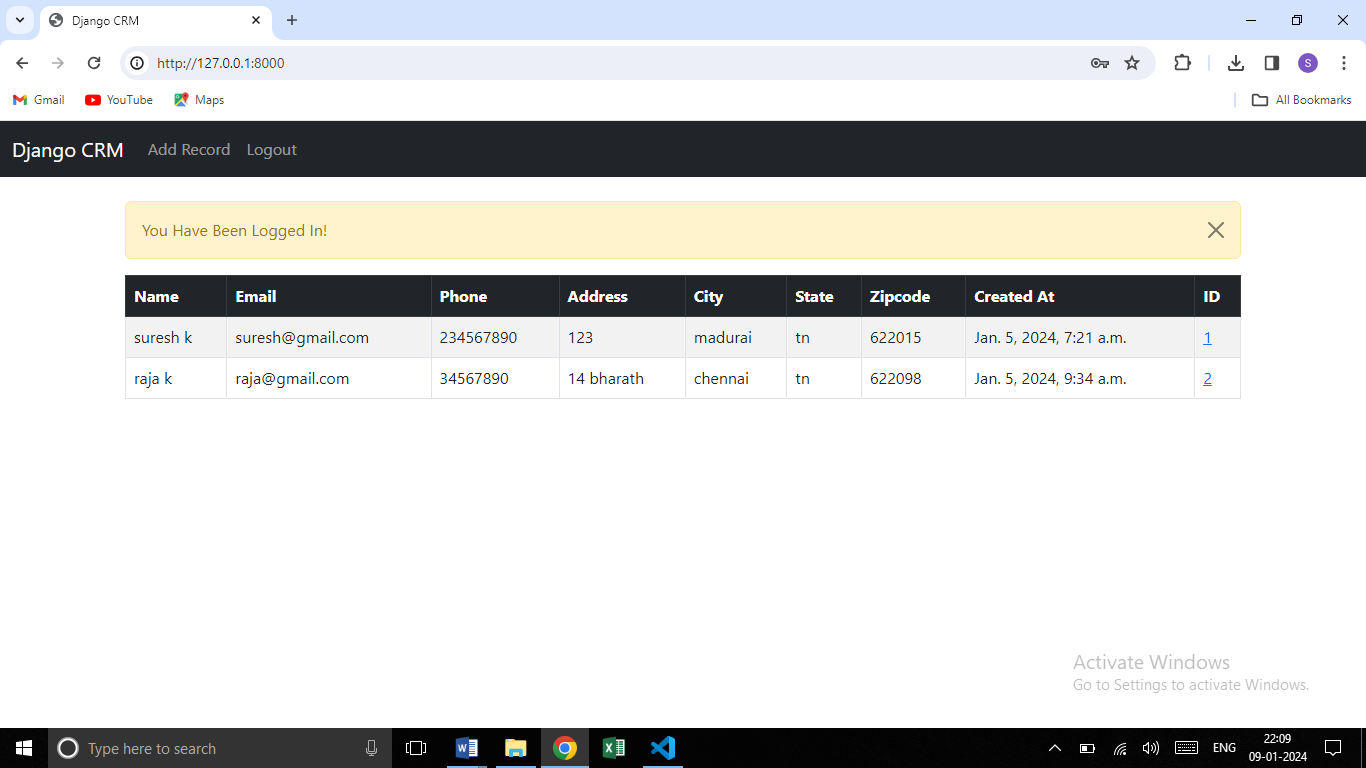
            return redirect('home')

    else:

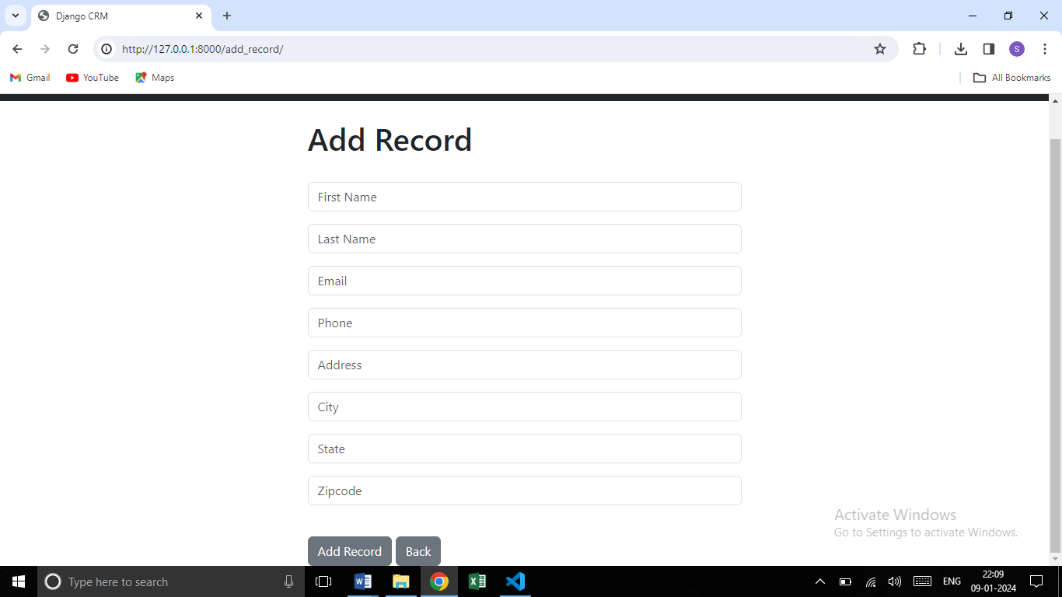
        return render(request, 'home.html', {'records':records})

If username and password is correct it shows success message and show records.

Else message as error and return to home page

****

When add record is clicked ,it shows

****

After inserting data into it and clicked add record,

def add\_record(request): runs add\_record.html file and form.py runs

def add\_record(request):

    form = AddRecordForm(request.POST or None)

    if request.user.is\_authenticated:

        if request.method == "POST":

            if form.is\_valid():

                add\_record = form.save()

                messages.success(request, "Record Added...")

                return redirect('home')

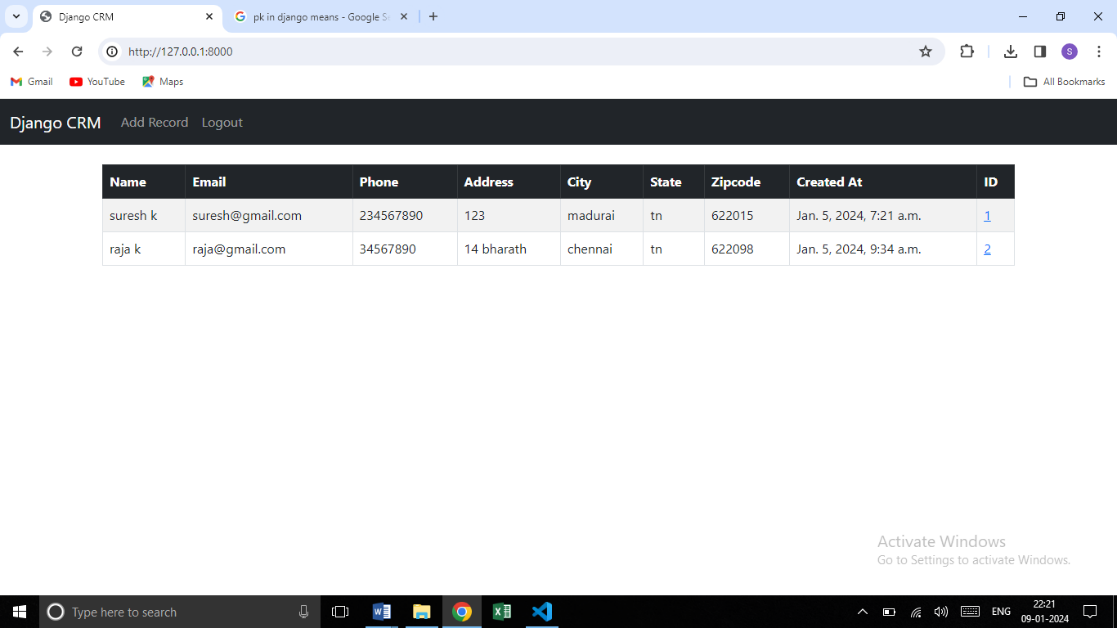
        return render(request, 'add\_record.html', {'form':form})

    else:

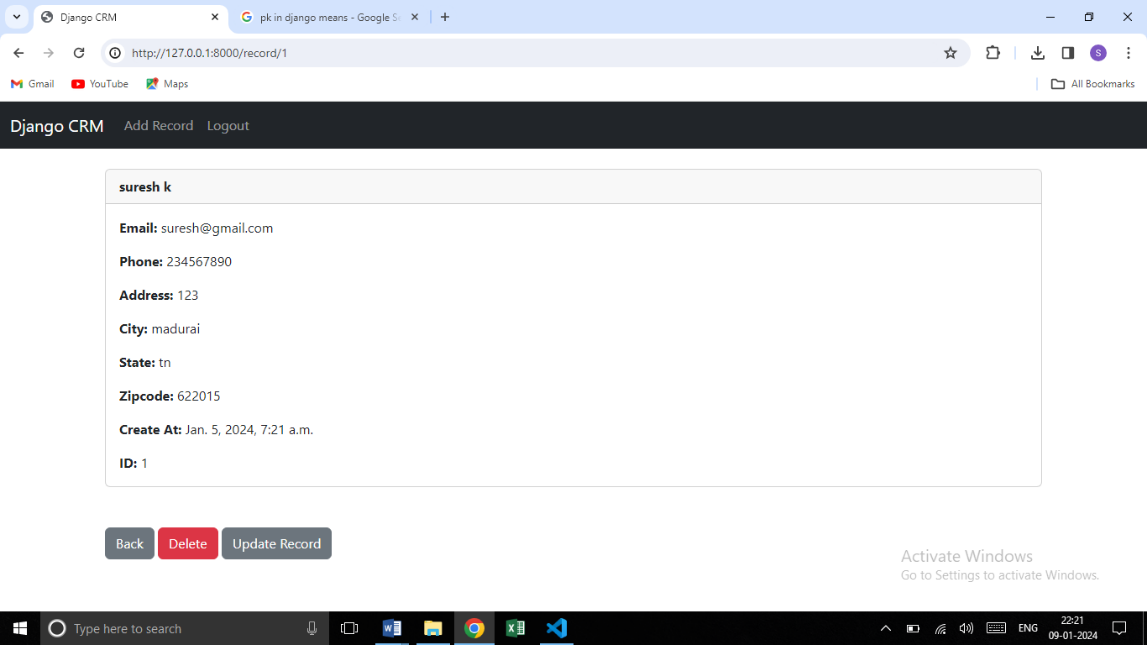
        messages.success(request, "You Must Be Logged In...")

        return redirect('home')

When id is clicked, it shows details of customer record def\_customer: and record.html runs in it.



Delete button and update button to run respective functions to do.



def customer\_record(request, pk):

    if request.user.is\_authenticated:

        # Look Up Records

        customer\_record = Record.objects.get(id=pk)

        return render(request, 'record.html', {'customer\_record':customer\_record})

    else:

        messages.success(request, "You Must Be Logged In To View That Page...")

        return redirect('home')

def delete\_record(request, pk):

    if request.user.is\_authenticated:

        delete\_it = Record.objects.get(id=pk)

        delete\_it.delete()

        messages.success(request, "Record Deleted Successfully...")

        return redirect('home')

    else:

        messages.success(request, "You Must Be Logged In To Do That...")

        return redirect('home')

def update\_record(request, pk):

    if request.user.is\_authenticated:

        current\_record = Record.objects.get(id=pk)

        form = AddRecordForm(request.POST or None, instance=current\_record)

        if form.is\_valid():

            form.save()

            messages.success(request, "Record Has Been Updated!")

            return redirect('home')

        return render(request, 'update\_record.html', {'form':form})

    else:

        messages.success(request, "You Must Be Logged In...")

        return redirect('home')

When logout nav bar is clicked, def logout\_user(request):

Runs and log outed.

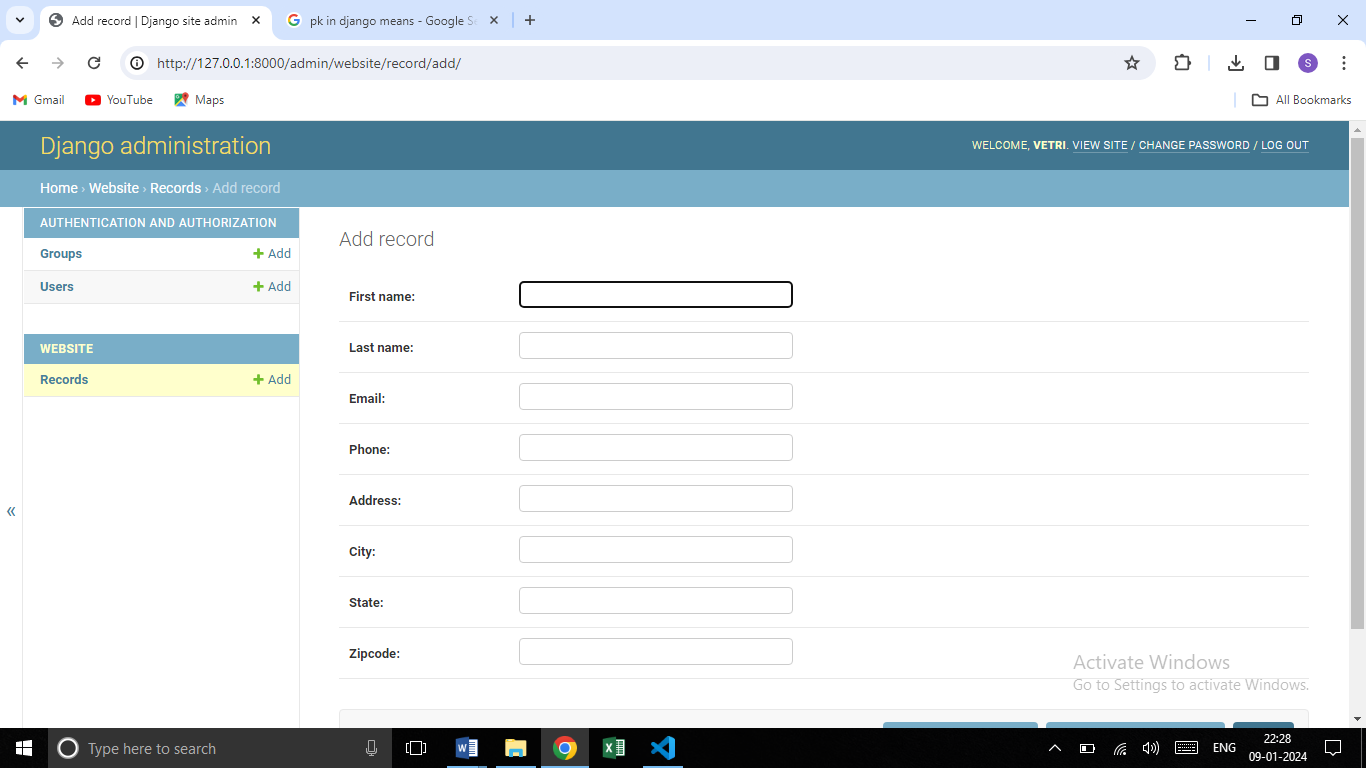
def logout\_user(request):

    logout(request)

    messages.success(request, "You Have Been Logged Out...")

    return redirect('home')

We can add and update record in admin page also



THANK YOU