Practice Project: Customer360

Estimated time needed: 45 minutes

This practice project will give you first hand experience applying the skills you learned in Django application development.

Scenario

Your organization is consolidating different platforms and wants to store customer communication records in a central location. As a software engineer, you are tasked to develop this Customer360 application using Django.

A communication record stores Channel, Direction (inbound or outbound) and Summary.

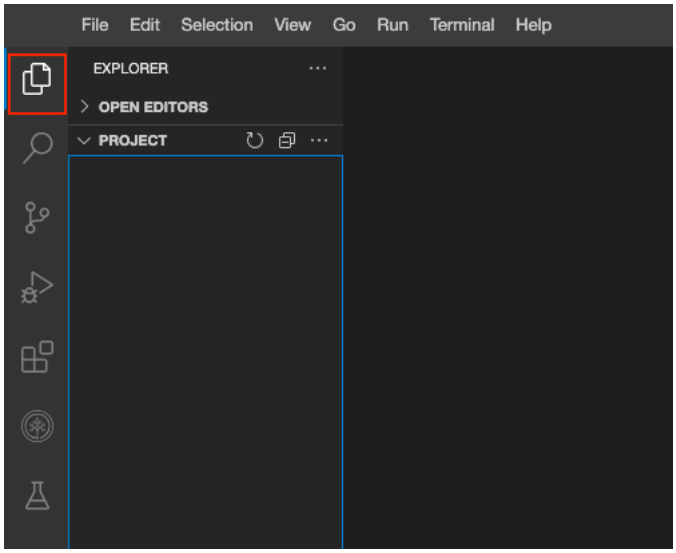
Objectives

- Develop a screen to capture communication record
- Display interaction in last 30 days
- Provide professional customer management

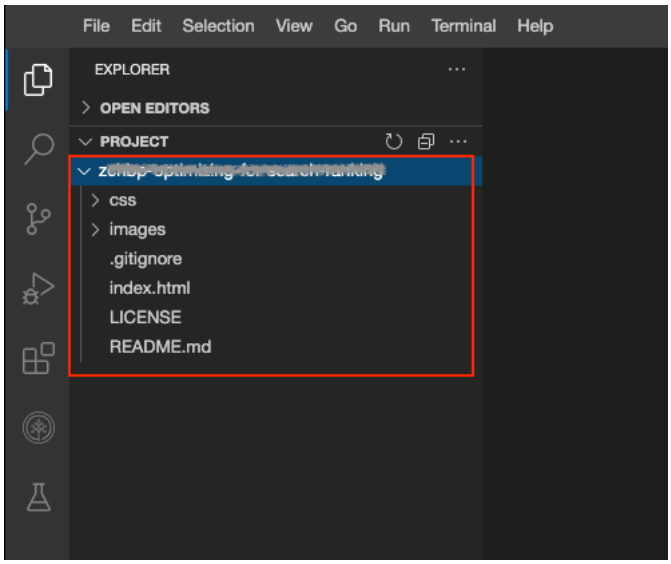
Working with Files in Cloud IDE

If you are new to Cloud IDE, this section will show you how to create and edit files that are part of your project in Cloud IDE.

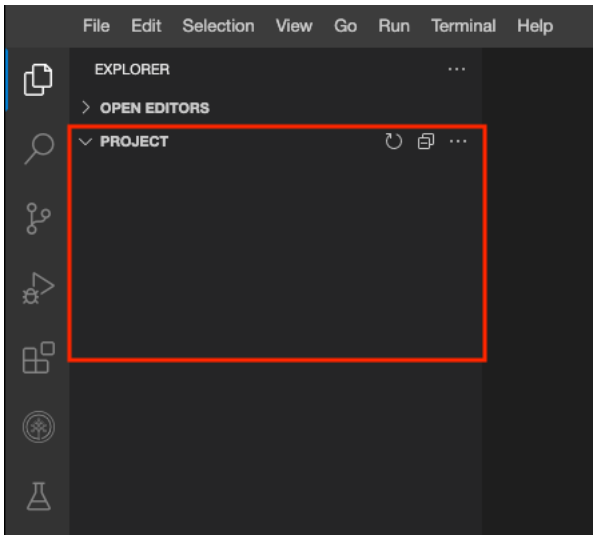
To view your files and directories inside Cloud IDE, click the file icon to reveal it.



If you have cloned (using the `git clone` command) boilerplate/starting code, then it will look like the image below:

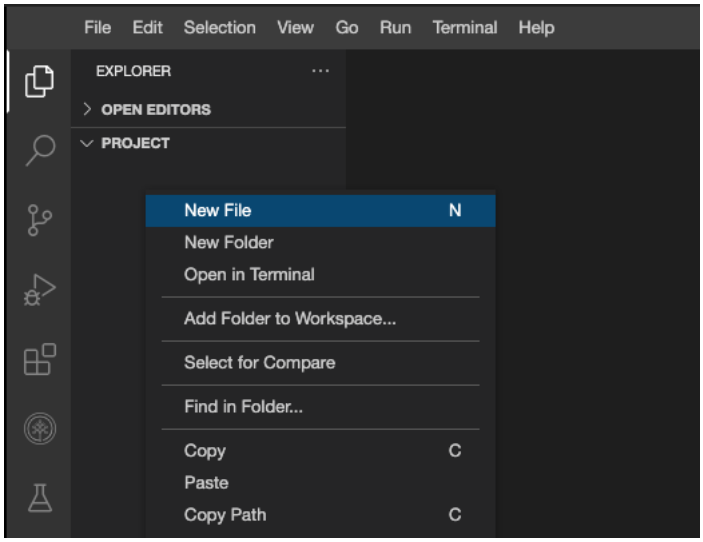


If you have not cloned and are starting with a blank project, it will look like this:

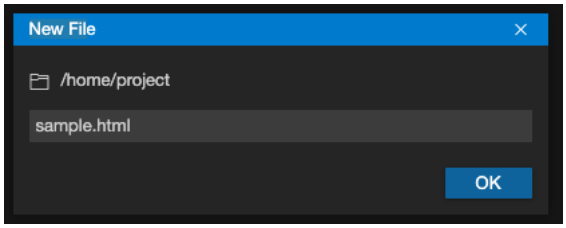


Create a New File

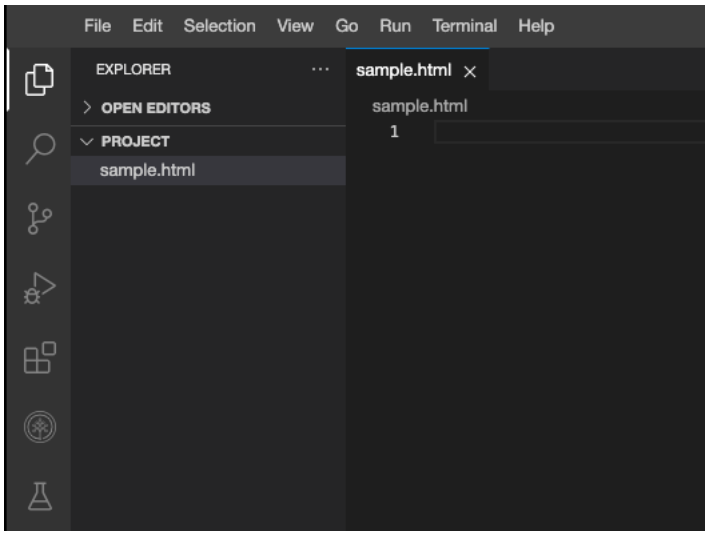
To create a new file in your project, right-click and select the **New File** option. You can also choose `File -> New File` to do the same.



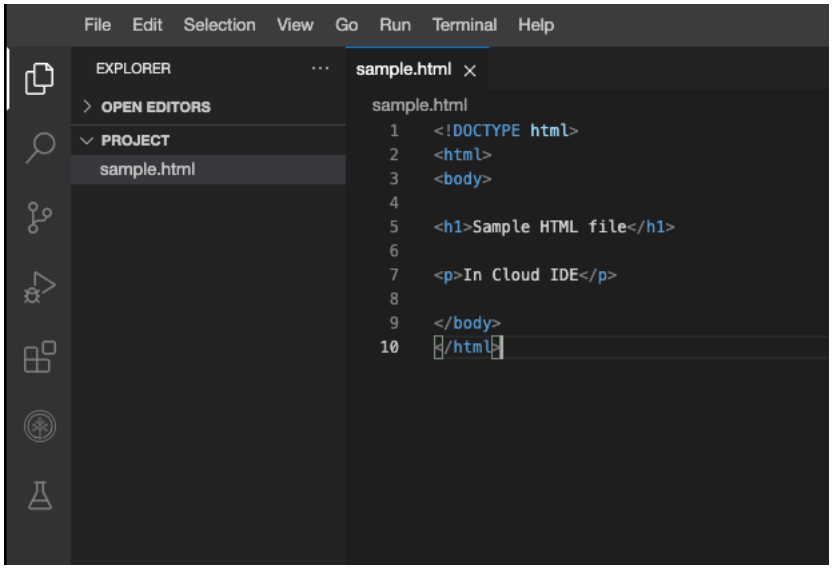
You will then be prompted to name the new file. In this scenario, let's name it `sample.html`.



Clicking the file name `sample.html` in the directory structure will open the file on the right pane. You can create all different types of files; for example, `FILE_NAME.js` for JavaScript files.

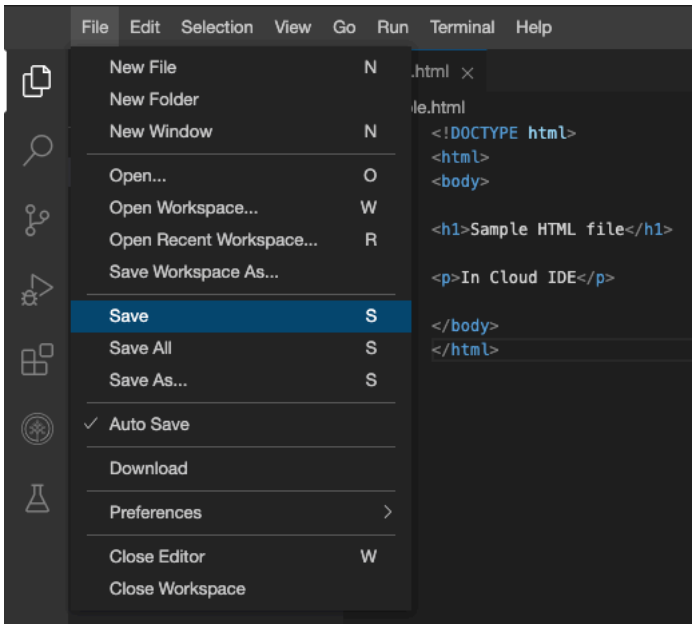


In the example below, we pasted some basic HTML code and then saved the file.



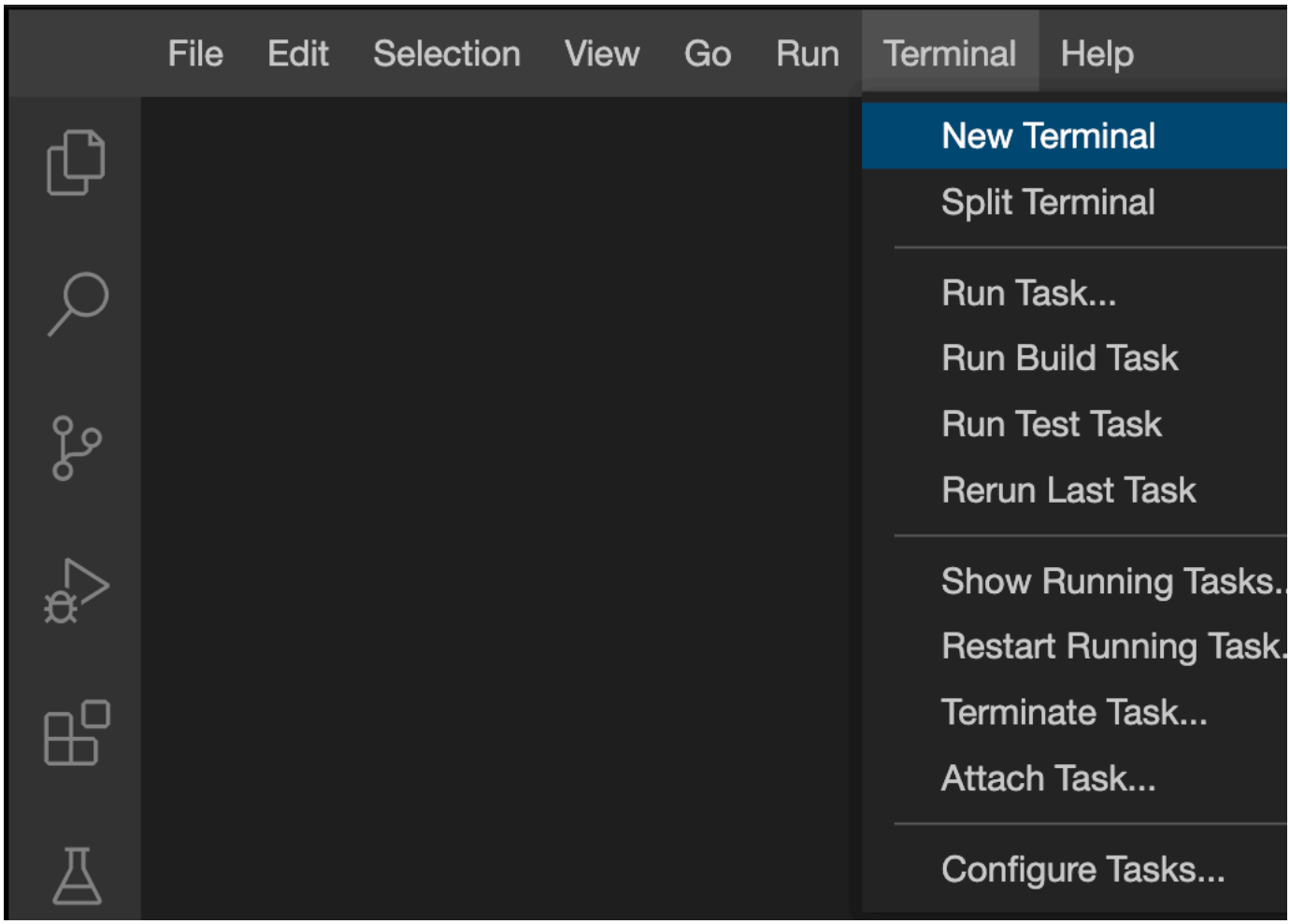
We save this file by:

- Going to the menu.
- Press `Command + S` on Mac or `CTRL + S` on Windows.
- Alternatively, it will Autosave your work as well.



Set-up: Create a Django Application

1. Open a terminal window using the editor's menu: Select **Terminal > New Terminal**.



2. If you are not currently in the project folder, copy and paste the following code to change to your project folder. Select the copy button to the right of the code to copy it.

```
cd /home/project
```

3. Ensure pip is installed.

```
python3.11 -m ensurepip
```

4. Install Django.

```
python3.11 -m pip install Django
```

5. Create a project.

```
django-admin startproject customer360
```

6. Change the directory so that it works in the lab.

```
cd customer360
```

7. Run migration before running the application for the first time.

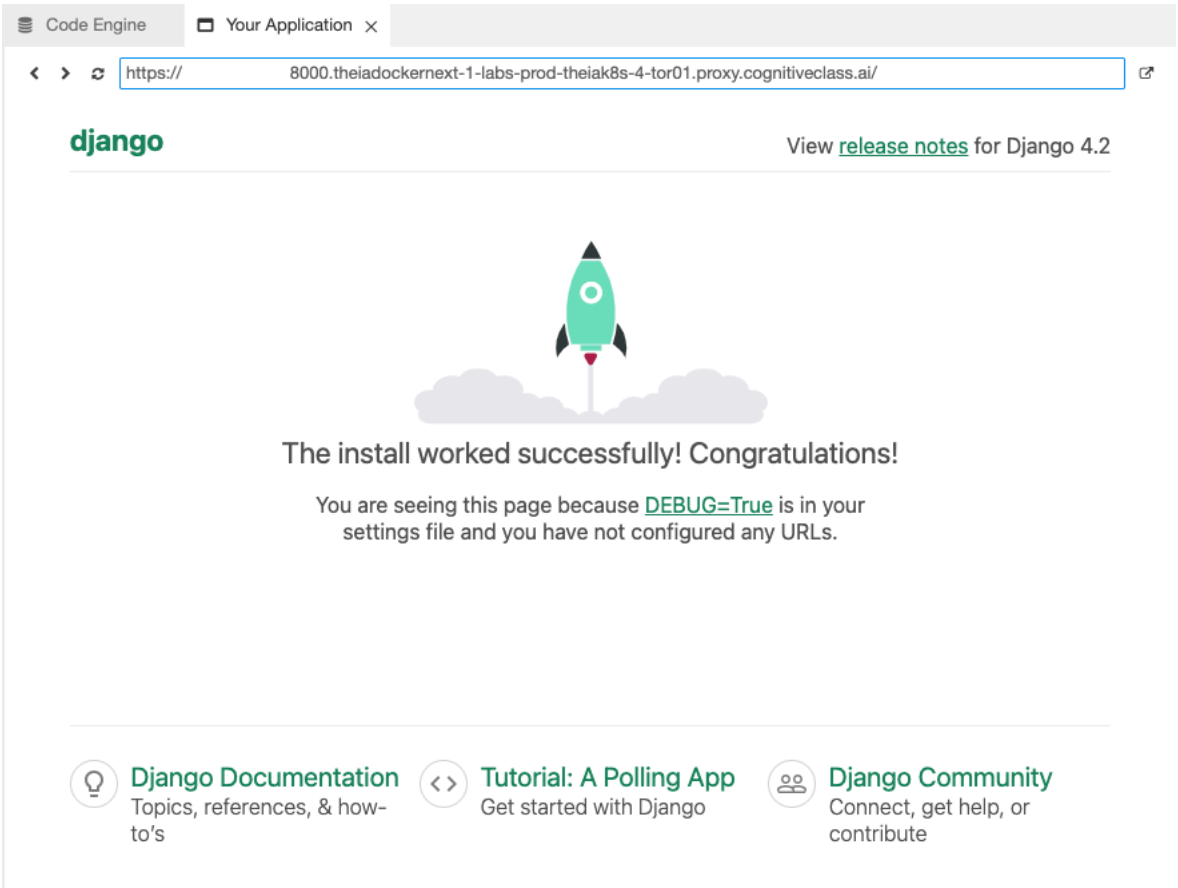
```
python3.11 manage.py migrate
```

8. Run the server successfully this time.

```
python3.11 manage.py runserver
```

Launch Application

9. It will look like the image below:



10. In your terminal, press `CTRL+C` to stop your web server.

Task 1: Modify Settings

Now that your environment project is set up, you are set to start doing work.

Each application you write in Django consists of a Python package that follows a certain convention. Django comes with a utility that automatically generates the basic directory structure of an app, so you can focus on writing code rather than creating directories.

You will make changes to `settings.py` in the customer360 app:

Open **settings.py** in IDE

Allowed Hosts

A list of strings representing the host/domain names that this Django site can serve.

```
ALLOWED_HOSTS=["*"]
```

Installed Apps

A list of strings designating all applications that are enabled in this Django installation. Each string should be a dotted Python path to:

- an application configuration class (preferred), or
- a package containing an application.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'customer360'
]
```

List of Trusted Origins

A list of trusted origins for unsafe requests (for example, POST).

For requests that include the Origin header, Django CSRF protection requires that the header match the origin present in the Host header.

```
CSRF_TRUSTED_ORIGINS = ['https://*.cognitiveclass.ai']
```

Import OS

To be able to use `path` property. You need to import the `os` module. Add an `import` statement near the top of the file after `from pathlib...`

```
from pathlib import Path
import os
```

Configure Additional Static Files Directory

This setting defines the additional locations the staticfiles app will traverse if the `FileSystemFinder` finder is enabled, for example, if you use the `collectstatic` or `findstatic` management command or use the static file serving view.

For clarity, add it after `STATIC_URL`.

```
STATICFILES_DIRS = (
    os.path.join(BASE_DIR,"static/"),
)
```

You can see the complete file below.

▼ Completed settings.py

```
"""
Django settings for customer360 project.
Generated by 'django-admin startproject' using Django 4.2.4.
For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/
For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""
from pathlib import Path
import os
# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-mxj20imb1j!8hz2!kqt*qh5^=y3q3^hyknmj**bpi9v2vuhr!p'
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
ALLOWED_HOSTS = ["*"]
CSRF_TRUSTED_ORIGINS = ['https://*.cognitiveclass.ai']
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'customer360'
]
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
ROOT_URLCONF = 'customer360.urls'
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        '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',
            ],
        },
    },
]
WSGI_APPLICATION = 'customer360.wsgi.application'
# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

```
# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_TZ = True
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/
STATIC_URL = 'static/'
STATICFILES_DIRS = (
    os.path.join(BASE_DIR,"static/"),
)
# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Task 2: Create Models

Let's define our models that will help us store data and build the UI.

> A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table.

Define models

Start with creating a model file in `customer360/customer360/models.py`. Run the following script to create the file.

```
touch /home/project/customer360/customer360/models.py
```

Open **models.py** in IDE

And define the following:

Import

```
from django.db import models
```

Customer Model

```
class Customer(models.Model):
    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=100)
    email = models.EmailField(max_length=100)
    phone = models.CharField(max_length=20)
    address = models.CharField(max_length=200)
    def __str__(self):
        return str(self.id)
```

Interaction model

```
class Interaction(models.Model):
    CHANNEL_CHOICES = [
        ('phone', 'Phone'),
        ('sms', 'SMS'),
```

```
        ('email', 'Email'),
        ('letter', 'Letter'),
    ]
    DIRECTION_CHOICES = [
        ('inbound', 'Inbound'),
        ('outbound', 'Outbound'),
    ]
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    channel = models.CharField(max_length=15, choices=CHANNEL_CHOICES)
    direction = models.CharField(max_length=10, choices=DIRECTION_CHOICES)
    interaction_date = models.DateField(auto_now_add=True)
    summary = models.TextField()
```

You can see the complete file below.

▼ Completed models.py

```
from django.db import models
# Create your models here.
class Customer(models.Model):
    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=100)
    email = models.EmailField(max_length=100)
    phone = models.CharField(max_length=20)
    address = models.CharField(max_length=200)
    def __str__(self):
        return str(self.id)
class Interaction(models.Model):
    CHANNEL_CHOICES = [
        ('phone', 'Phone'),
        ('sms', 'SMS'),
        ('email', 'Email'),
        ('letter', 'Letter'),
    ]
    DIRECTION_CHOICES = [
        ('inbound', 'Inbound'),
        ('outbound', 'Outbound'),
    ]
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    channel = models.CharField(max_length=10, choices=CHANNEL_CHOICES)
    direction = models.CharField(max_length=10, choices=DIRECTION_CHOICES)
    interaction_date = models.DateField(auto_now_add=True)
    summary = models.TextField()
```

Task 3: Create Templates

You now need to add a few HTML files that will show our models and let users interact with the application.

```
mkdir /home/project/customer360/customer360/templates
touch /home/project/customer360/customer360/templates/add.html
touch /home/project/customer360/customer360/templates/base.html
touch /home/project/customer360/customer360/templates/index.html
touch /home/project/customer360/customer360/templates/interact.html
touch /home/project/customer360/customer360/templates/summary.html
```

You should review each HTML content as you paste the models into the relevant file.

base.html

You will start with base template, aptly named as `base.html`. This file contains sections that we will fill in with other templates.

Open **base.html** in IDE

```
{% load static %}
<!DOCTYPE html>
<html>
    <head>
```

```
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
        <link rel="stylesheet" href="{% static 'css/main.css' %}">
    </head>
    <body>
        <nav class="navbar navbar-default">
                <ul class="nav navbar-nav">
                    <li>
                        <a style="color:black;" href="/">Home</a>
                    </li>
                    <li>
                        <a  style="color:black;" href="/create">New Customer</a>
                    </li>
                    <li>
                        <a style="color:black;" href="/summary">Summary</a>
                    </li>
                </ul>
        </nav>
        {% block content %}
        {% endblock %}
    </body>
</html>
```

index.html

Next you will fill the `index.html` with HTML content. This is our landing page.

Open **index.html** in IDE

```
{% extends 'base.html' %}
{% load static %}
{% block content %}
<html>
    <head>
        <title>Home Page</title>
    </head>
    <script>
       function set_customer(){
        var cinput = document.querySelector('input[name="selected_customers"]:checked');
            if (cinput){
                cid = cinput.value;
                window.location = "/interact/"+cid;
            }
            else
                alert("Please select a customer");
       }
    </script>
    <body>
        <h1>Welcome to Customer 360</h1>
        <p>Interact and Manage your Customers</p>
        <a class="btn btn-primary" style="font-weight:bold; display:inline" onclick="set_customer()">Interact</a>
        <table class="table">
            <thead>
                <tr>
                    <th>Customer ID</th>
                    <th>Name</th>
                    <th>Email</th>
                    <th>Phone</th>
                    <th>Address</th>
                    <th>Selected</th>
                </tr>
            </thead>
            <tbody>
                {% for customer in customers %}
                    <tr>
                        <td>{{customer.id }}</td>
                        <td>{{customer.name }}</td>
                        <td>{{customer.email }}</td>
                        <td>{{customer.phone }}</td>
                        <td>{{customer.address }}</td>
                        <td>
                            <input type="radio" name="selected_customers" value="{{ customer.id }}">
                        </td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    </body>
</html>
{% endblock content %}
```

add.html

You will now create the template for adding a new customer. You will notice that it extends to `base.html`.

Open **add.html** in IDE

```
{% extends 'base.html' %}
{% load static %}
{% block content %}
<html>
    <head>
        <title>Add a Customer</title>
    </head>
    <body>
        <h1>Add a new Customer</h1>
        <form class="form" method="post" action="/create/">
            {% csrf_token %}
            <div class="form-group">
                <label for="Name">Name </label>
                <input type="text" name="name" required>
            </div>
            <div class="form-group">
                <label for="Email">Email </label>
                <input type="email" name="email" required>
            </div>
            <div class="form-group">
                <label for="Phone">Phone</label>
                <input type="tel" name="phone" required>
            </div>
            <div class="form-group">
                <label for="Address">Address</label>
                <input type="text" name="address" required>
            </div>
            <button type="submit" class="btn btn-success">Add</button>
            <p> {{ msg }} </p>
        </form>
    </body>
</html>
{% endblock content %}
```

interact.html

To record an interaction with a customer, we use the template `interact.html`.

Open **interact.html** in IDE

```
{% extends 'base.html' %}
{% load static %}
{% block content %}
<html>
    <head>
        <title>Interact & Manage</title>
    </head>
    <script>
        function selectButton(element) {
            var buttons = element.parentElement.getElementsByClassName("btn");
            for (var i = 0; i < buttons.length; i++) {
                buttons[i].classList.remove("active");
            }
            element.classList.add("active");
        }
        function check_selected(){
            var dirinput = document.querySelector('input[name="direction"]:checked');
            var chaninput = document.querySelector('input[name="channel"]:checked');
            var summary = document.querySelector('textarea[name="summary"]').value;
            if (!dirinput || !chaninput || summary === ""){
                alert("Please fill all required fields");
                return false;
            }
            return true;
        }
    </script>
    <body>
        <h1>Interact With Your Customers</h1>
        <form class="form" method="post"  onsubmit="return check_selected()" action="#">
            {% csrf_token %}
            <div class="form-group">
                <label>Channel</label>
                <div class="btn-group" data-toggle="buttons">
                    {% for channel in channels %}
                        <label class="btn btn-outline-primary" onclick="selectButton(this)">
                            <input type="radio" name="channel" value="{{ channel.0 }}" required> {{ channel.1 }}
                        </label>
                    {% endfor %}
                </div>
            </div>
            <div class="form-group">
                <label>Direction</label>
                <div class="btn-group" data-toggle="buttons">
                    {% for direction in directions %}
```

```
                        <label class="btn btn-outline-primary" onclick="selectButton(this)">
                            <input type="radio" name="direction" value="{{ direction.0 }}" required> {{ direction.1 }}
                        </label>
                    {% endfor %}
                </div>
            </div>
            <div class="form-group">
                <label>Summary</label>
                <textarea name="summary"></textarea>
            </div>
            <button type="submit" class="btn btn-success">Save Interaction</button>
            <p>{{ msg }}<p>
        </form>
    </body>
</html>
{% endblock content %}
```

summary.html

Finally, you will fill the HTML template `summary.html` as follows:

Open **summary.html** in IDE

```
{% extends 'base.html' %}
{% load static %}
{% block content %}
<html>
    <body>
        <h1> Interactions in last 30 Days  </h1>
        {% if not interactions %}
            <p> there are no interactions in the last 30 days <p>
        {% else %}
            <table class="table">
                <thead>
                    <tr>
                        <th>Channel</th>
                        <th>Direction</th>
                        <th>Count</th>
                    </tr>
                </thead>
                <tbody>
                    {% for interaction in interactions %}
                        <tr>
                            <td>{{ interaction.channel }}</td>
                            <td>{{ interaction.direction }}</td>
                            <td> {{ interaction.count }} </td>
                        </tr>
                    {% endfor %}
                </tbody>
            </table>
            <h4> Total : {{ count }} </h4>
        {% endif %}
    </body>
</html>
{% endblock content %}
```

Task 4: Create Views

You will now create `customer360/customer360/views.py` to define views.

```
touch /home/project/customer360/customer360/views.py
```

Open **views.py** in IDE

Define Imports

```
from django.shortcuts import render
from datetime import date, timedelta
from django.db.models import Count
from .models import *
```

Define Index View

```
def index(request):
    customers = Customer.objects.all()
    context = {"customers":customers}
    return render(request,"index.html",context=context)
```

Define Create Customer View

```
def create_customer(request):
    if request.method == "POST":
        name = request.POST["name"]
        email = request.POST["email"]
        phone = request.POST["phone"]
        address = request.POST["address"]
        customer = Customer.objects.create(name=name,email=email,phone=phone,address=address)
        customer.save()
        msg = "Successfully Saved a Customer"
        return render(request,"add.html",context={"msg":msg})
    return render(request,"add.html")
```

Define Summary View

```
def summary(request):
    thirty_days_ago = date.today() - timedelta(days=30)
    interactions = Interaction.objects.filter(interaction_date__gte=thirty_days_ago)
    count = len(interactions)
    interactions = interactions.values("channel","direction").annotate(count=Count('channel'))
    context={
                "interactions":interactions,
                "count":count
             }
    return render(request,"summary.html",context=context)
```

Define Interaction View

```
def interact(request,cid):
    channels = Interaction.CHANNEL_CHOICES
    directions = Interaction.DIRECTION_CHOICES
    context = {"channels":channels,"directions":directions}
    if request.method == "POST":
        customer = Customer.objects.get(id=cid)
        channel = request.POST["channel"]
        direction = request.POST["direction"]
        summary = request.POST["summary"]
        interaction = Interaction.objects.create(
                                    customer=customer,
                                    channel=channel,
                                    direction=direction,
                                    summary=summary)
        interaction.save()
        context["msg"] = "Interaction Success"
        return render(request,"interact.html",context=context)
    return render(request,"interact.html",context=context)
```

Completed File

You can see the completed file below.

▼ Completed views.py

```
from django.shortcuts import render
from datetime import date, timedelta
from django.db.models import Count
from . models import *
# Create your views here.
def index(request):
    customers = Customer.objects.all()
    context = {"customers":customers}
    return render(request,"index.html",context=context)
def create_customer(request):
    if request.method == "POST":
        name = request.POST["name"]
        email = request.POST["email"]
        phone = request.POST["phone"]
        address = request.POST["address"]
        customer = Customer.objects.create(name=name,email=email,phone=phone,address=address)
        customer.save()
        msg = "Successfully Saved a Customer"
        return render(request,"add.html",context={"msg":msg})
    return render(request,"add.html")
def summary(request):
    thirty_days_ago = date.today() - timedelta(days=30)
    interactions = Interaction.objects.filter(interaction_date__gte=thirty_days_ago)
    count = len(interactions)
    interactions = interactions.values("channel","direction").annotate(count=Count('channel'))
    context={
                "interactions":interactions,
                "count":count
            }
    return render(request,"summary.html",context=context)
def interact(request,cid):
    channels = Interaction.CHANNEL_CHOICES
    directions = Interaction.DIRECTION_CHOICES
    context = {"channels":channels,"directions":directions}
    if request.method == "POST":
        customer = Customer.objects.get(id=cid)
        channel = request.POST["channel"]
        direction = request.POST["direction"]
        summary = request.POST["summary"]
        interaction = Interaction.objects.create(
                                    customer=customer,
                                    channel=channel,
                                    direction=direction,
                                    summary=summary)
        interaction.save()
        context["msg"] = "Interaction Success"
        return render(request,"interact.html",context=context)
    return render(request,"interact.html",context=context)
```

Task 5: Create URLs

A clean, elegant URL scheme is an important detail in a high-quality web application. Django lets you design URLs however you want, with no framework limitations.

You will make changes to `urls.py` in the customer360 app:

Open **urls.py** in IDE

Import Views

```
from . import views
```

Add URL Patterns

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.index,name="index"),
    path('create/',views.create_customer,name='create_customer'),
    path('interact/<int:cid>',views.interact,name='interact'),
    path('summary/',views.summary,name='summary'),
]
```

You can see the complete file below.

▼ Completed urls.py

```
from django.contrib import admin
from django.urls import path
from . import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.index,name="index"),
    path('create/',views.create_customer,name='create_customer'),
    path('interact/<int:cid>',views.interact,name='interact'),
    path('summary/',views.summary,name='summary'),
]
```

Task 6: Add Styling

To visually enhance the user experience, you are given the following CSS which you should add to your application.

```
mkdir -p /home/project/customer360/static/css
touch /home/project/customer360/static/css/main.css
```

Open **main.css** in IDE

CSS content

```
body{
    text-align: center;
    margin: auto;
}
h1,p{
    font-weight: bold;
}
table{
    margin: auto;
    border: 2px solid #000000;
    margin-top: 50px;
    border-radius: 15px;
    background-color: white;
    text-align: center;
}
th{
    text-align: center;
}
.nav{
    width: 200%;
    border: 2px solid black;
    border-radius: 10px;
}
.nav li{
    margin-right:  6%;
    margin-left:  6%;
    font-size: large;
    font-weight: bold;
}
.form{
    margin: auto;
```

```
        margin-top: 20px;
        background-color: white;
        text-align: center;
        width: 50%;
        height: 50%;
        border: 2px solid black;
        border-radius: 10px;
    }
    .form-group{
        display: flex;
        flex-direction: column;
        align-items: flex-start;
        width:100%;
        padding: 10px;
        text-align: left;
        margin-top: 10px;
        width: 100%;
    }
    input[type="text"],input[type="email"],input[type="tel"]{
        width: 100%;
    }
    textarea{
        width: 500px;
        height: 200px;
    }
    .btn{
        display: flex;
        margin-top: 10px;
        margin-bottom: 10px;
        padding-left: 50px;
        padding-right: 50px;
        text-align: left;
        margin-left: 10px;
    }
    .active {
        background-color: #007bff;
        color: #fff;
    }
```

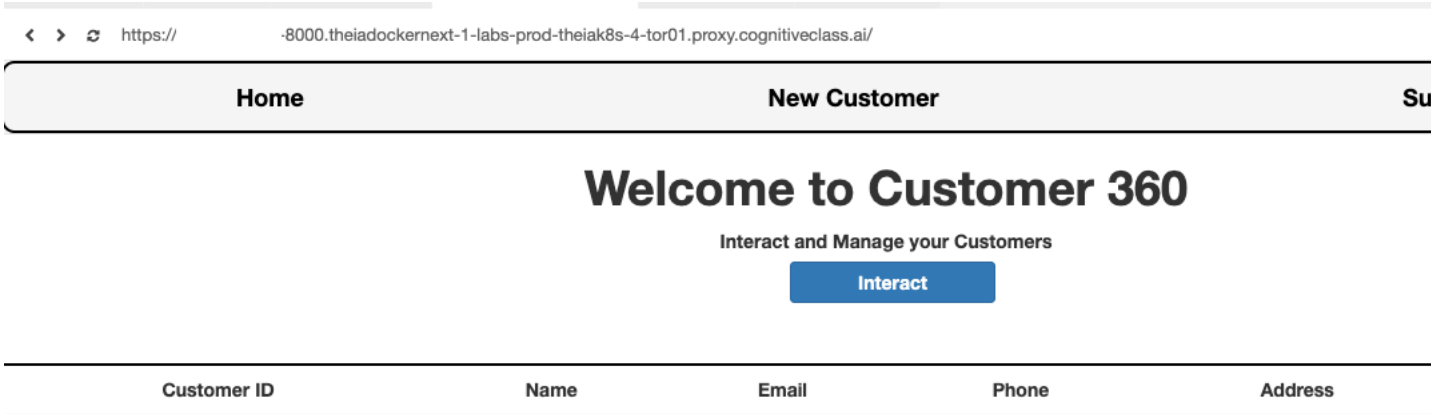
Task 7: Run the Application

```
cd /home/project/customer360
python3.11 manage.py makemigrations customer360
python3.11 manage.py migrate
python3.11 manage.py runserver
```

Launch Customer360

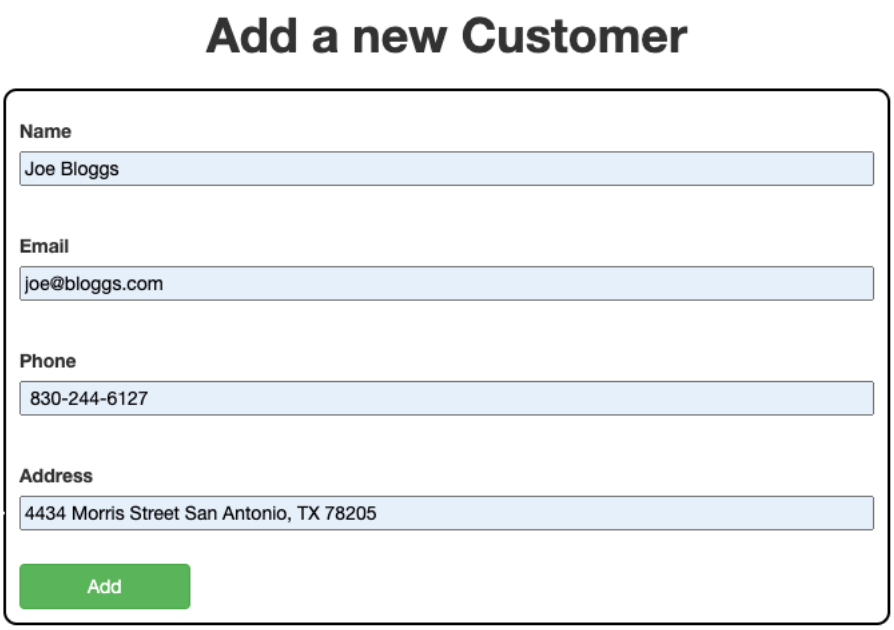
Launch

When you launch the Customer360 app you will see a similar view as shown below.



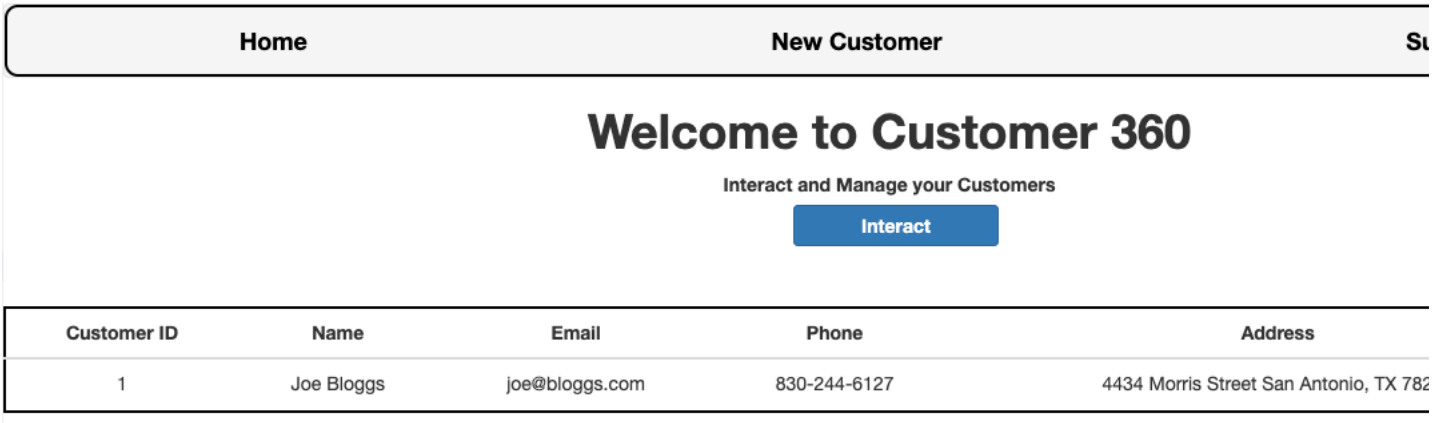
New Customer

There are no customer records because you are running it for the first time. Let' s add one.



List of Customers

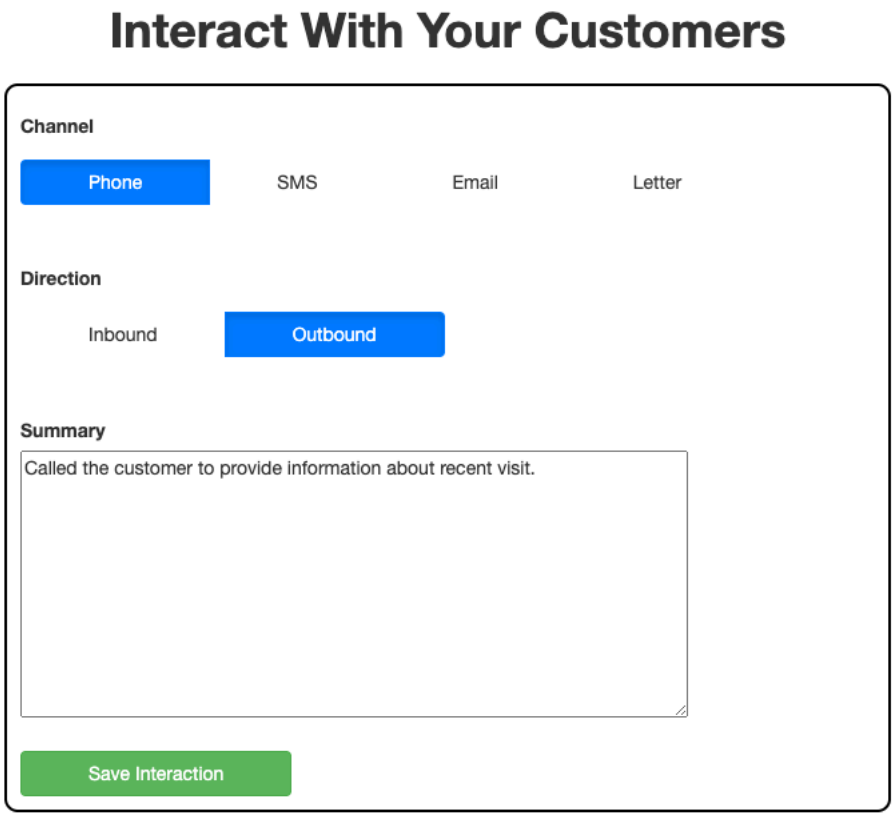
You can now see your newly added customer on the main page.



New Interaction record

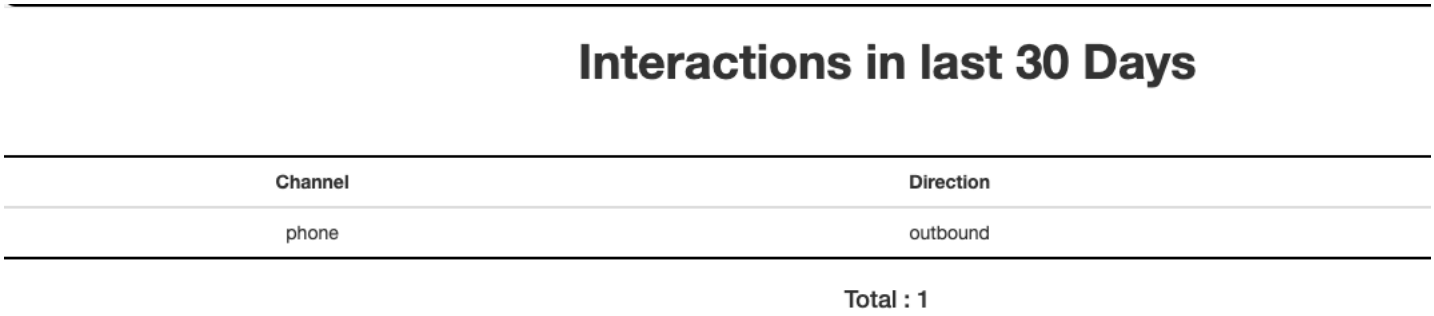
You will now record a new interaction for this customer, by selecting the customer record and then `Interact`.

Then fill in the interaction details.



Summary

You can now go to the Summary page.



Task 8: Modifications (Optional)

Your challange now is to enhance the Customer360 app and add some functionality. You can discuss these modifications with your peers.

1. Add a new field to customer model
2. Create a new interaction channel

Add social media Field to Customer

Model change

In the `Customer` model, add a new optional field called `social_media`. And display it with customer details.

▼ Hint

```
new_field = models.data_type(args)
```

▼ Solution

```
social_media = models.CharField(max_length=100, blank=True)
```

Template change

Now you need to ensure that the new field `social_media` is added to the customer create form.

▼ Hint

```
<div class="form-group">
    <label for="Field">Field Label </label>
    <input type="Type" name="Name">
</div>
```

▼ Solution

```
<div class="form-group">
    <label for="SocialMedia">Social Media </label>
    <input type="text" name="social_media">
</div>
```

And that the `social_media` field is added in the grid on the landing page.

▼ Hint

```
<th>Field Name</th>
...
<td>{{mode.field_name }}</td>
```

▼ Solution

```
<th>Social Media</th>
...
<td>{{customer.social_media }}</td>
```

View change

1. Add the code for posting as a Social media text

▼ Hint

Use the `request.POST` method

▼ Solution

```
social_media=request.POST["social_media"]
```

2. Create a new Customer object using the `Customer.objects.create()` method

▼ Hint

Add the `social_media` attribute to the existing `Customer.objects.create()` method

▼ Solution

```
customer = Customer.objects.create(name=name,email=email,phone=phone,address=address,social_media=social_media)
```

Add new interaction channel

This change is only in one place. Can you figure out where this should be?

▼ Hint
In Models file, in the Channel Choices array, add a new entry. The reason this change is only in one place is because in interact.html, we are iterating over possible channels instead of hard coding.

```
('new_choice', 'New Choice'),
```

▼ Solution

```
('social_media', 'Social Media'),
```

Test your changes

To apply these model changes, you need to run certain procedures. These procedures will apply the model changes to your database.

▼ Hint

```
# create migrations
# migrate
# run server to test
```

▼ Solution

```
python3.11 manage.py makemigrations customer360
python3.11 manage.py migrate
python3.11 manage.py runserver
```

Launch Customer360

Summary

Congratulations!

You have completed this practice project exercising the skills you learned in Django application development.

Author(s)

- Muhammad Yahya