**Getting started with Django**

**Introduction**

Here, we will be learning everything about Django basics. We will be making a CRM (Customer Relationship Management) project using Django.

**Prerequisites**  
First, let’s creating a python virtual environment. Make sure to install python virtual environment using following prompt using following prompt in our terminal/ command prompt.

|  |
| --- |
| pip install virtualenv |

After installing python virtual environment, we can create it using the following prompt.

|  |
| --- |
| python -m venv env |

Now, let’s activate the virtual environment so that we can install the necessary packages for this project.

|  |
| --- |
| env\scripts\activate |

After activating the virtual environment, we will be installing the package called ‘Django’. We will be installing 3.1.4 version specifically.

|  |
| --- |
| pip install django==3.1.4 |

**First Django project**

Now, let’s create a project using Django.

|  |
| --- |
| django-admin startproject djcrm |

**Server & Migrations**

Now, let’s run the server for our Django project. And then we can see our webpage in the url ‘127.0.0.1:8000/’.

|  |
| --- |
| python manage.py runserver |

|  |
| --- |
|  |

Now, let’s run our first migration.

|  |
| --- |
| python manage.py migrate |

**Creating our app**

Now, we need to create an app by using the following prompt. Let’s name it “leads”.

|  |
| --- |
| django-admin startapp leads |

|  |
| --- |
| Workflow |

Now, for Django to register and recognize an app that has been created, we need to mention it in ‘settings.py’ file which we can find it in ‘djcrm’ folder.

|  |
| --- |
|  |

**Django Models**

We are going to look at the most important concepts called models. Basically, models are a representation of a database schema.

As an example, let’s imagine what kinds of tables we would have in our database. Suppose we have tables of leads, and each lead has their own first name, last name, age, address, contact number, etc. That database table or CSV file or whatever the format of the data be, we use Django models to essentially model or represent the structure of that data.  
Now the way to write the model, we use python class keyword.

|  |
| --- |
|  |

In the above code snippet, we can see that we named the class ‘Lead’ as we will be dealing with leads. ‘models’ is the file name we imported from ‘django.db’ in line 1. ‘.Models’ is the class from the file ‘models’ which we are going to inherit. This is how we create a model.

Now, in our model, we are going to create different properties that will help us to represent how the data inside the ‘Lead’ will look like. So, usually in the databases, first name and last name will be strings, age will be integers, salaries could be float and so on. So, for first name, we will create a field and that will be ‘first\_name=’ in the above code snippet. Following with ‘models’ which help to create a datatype of this field and we will add ‘.CharField(max\_length=20)’ because first name can only be a string and with max characters of 20. For the charField, we need to always mention ‘max\_length’ and 20 would be enough for the names.

Last name will be like the first name as it is also a string datatype. We should mention the max length here as well.

Now let’s create an age field. For numbers, we can use integer and float datatype. Since age is an integer datatype (a person’s age is measured in integer), we will be using integer datatype. So, write ‘age = models.IntegerField(default=0)’. We will also put the default value as 0.

Now, we need to do two things to solidify it as a part of database schema and create it inside the database. In command prompt, we will run the following prompt.

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py makemigrations  Migrations for 'leads':  leads\migrations\0001\_initial.py  - Create model Lead |

This prompt will create a schema according to the model we just created. Running this file, we can see that inside the folder ‘leads’ and inside the folder ‘migrations’, a new file ‘0001\_initial.py’ has been created.

|  |
| --- |
| 0001\_initial.py |

Here, we can see the data and time this migration was done. In operations, we can see ‘CreateModel’ which will create the table. The name of the model is ‘Lead’ and below it, we can see the fields. As we can see, ‘id’ has been created though we did not mention it anywhere before. Django creates an id field automatically as it makes things easier to work with. But this is still just a blueprint as we have yet to do work on the database.

Now, the second thing that we need to do is to migrate. So, we run the following prompt.

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py migrate  Operations to perform:  Apply all migrations: admin, auth, contenttypes, leads, sessions  Running migrations:  Applying leads.0001\_initial... OK |

Running this prompt, Django is going to go through all the files inside the ‘migration’ folder and run it. So, if there are models which have already been applied will be skipped and if there are models which are yet to be applied are executed such as ‘0001\_initial.py’ file.

Now the ‘0001\_initial.py’ is not only a blueprint but it has now been applied to the database. We want to open ‘db.sqlite3’ file, we would see the table named ‘lead’ and we would see all the column mentioned in ‘0001\_initial.py’ file.

|  |
| --- |
|  |

We can check it using SQLite extension.

Here we can see a lot of tables, but ‘leads\_lead’ is the one we created.

**Django Model Fields**

Till now we have created two datatype fields in the model. There are a lot more we can use. Let’s use a Boolean datatype field for creating the next field.

|  |
| --- |
|  |

We will create a new field called phoned which can be used as a measure to know whether the lead has been phoned or not. The values in Boolean field default can be either ‘True’ or ‘False’. We set the default value to ‘False’ in this case.

Now, we can also store the source of the lead. We can know where it came from (it could be from YouTube, google, newsletter, etc.).

|  |
| --- |
|  |

In this case of creating field ‘source’, we are going to provide choices which can be selected by the user. As we can from the code snippet above, we have used string data (CharField) and inside we have mentioned choices. Choices can be written using python tupples, but the way we are doing in the code above is cleaner way of doing it. By using choices, we can in future use the tupples in the webpage form. And lastly, we added max length here as well.

Now we can also give a profile picture to the lead. We will use ‘ImageField’ datatype for this field.

|  |
| --- |
|  |

Here, we can see, ‘blank=True’ which indicates that the user has submitted an empty string whereas ‘null=True’ indicates there is no value in the database. These two things are different, and we will go through them again in the future.

Now, let’s add ‘special\_files’ field where we can let user to add files related to the lead.

|  |
| --- |
|  |

|  |
| --- |
|  |

The code snippet above is only for understanding purposes. We will not be making this as a part of our project. But if we would want to make these fields a part of our database schema, then just like before we will run ‘makemigrations’ and then ‘migrate’ in our command prompt.

**Foreign Key**

Now we will use foreign keys and understand the relationship between models. Just like in database tables, foreign key is used to connect one table to another.

Let’s create a model named ‘agent’. Let this agent be part of the team. Like the previous model, we will create a class and then create first name and last name fields. Now let’s assign the lead to the agent (communication).

To set a lead to an agent, we need to use the foreign key in lead.

Now, let’s create a foreign key field in the lead model. We must move the agent model above the lead model so that it can then inherit it.

|  |
| --- |
|  |

There is another way to do it as well i.e. by using quotation in the foreign key. This would be a better way to do it as we do not fret about the ordering complications.

|  |
| --- |
|  |

Then we need to add ‘on\_delete’ in the parameters or we will get an error when running the server. Foreign key should always be passed through ‘on\_delete’. Basically ‘on\_delete’ tells Django how to handle things when an instance is deleted. There are couple of ways to handle it.

|  |
| --- |
|  |

One method is to use ‘models.CASCADE’. It deletes the lead when the agent is deleted.

|  |
| --- |
|  |

Another method is to ‘models.SET\_NULL’ which sets the value of the foreign to null, but it only works if we set ‘null=True’.

Another method is ‘models.SET\_DEFAULT’ which needs to have default set.

We are to going ‘CASCADE’ in this case.

The thing to understand is that we put a foreign key in lead rather than in agent because an agent could handle a lot of leads. If we had put foreign key in agent, then it would mean that agent is only able to handle one lead.

**Custom User Model**

Django has a lot of built in functionality to handle authentication and one of them is the user model.

|  |
| --- |
|  |

First, we are going to import a built in function from ‘django.contrib.auth’. Then we will create a model named ‘User’ which we will call in agent class. In class ‘Agent’ we can see that we have used ‘OneToOneField’ datatype, which is a better option, but its functionality is like foreign key that we used before.

This is a built-in user model which works fine but in the case of bigger projects, creating a custom user model is better. For creating custom user models, there are some changes we must make to the code.

|  |
| --- |
|  |

First, we will import ‘AbstractUser’ from different path and then we will create a new class called ‘User’. The rest of the code where we create a field called ‘cellphone\_number’ is understandable.

After creating a custom user model, we should always remember to mention it in ‘settings.py’ file.

|  |
| --- |
|  |

Now in ‘models.py’ file, if we press ctrl key and then click on ‘AbstractUser’, a file called ‘models.py’ which exists in folder in ‘env’ folder. There we can see a lot of classes being created. Here is some code we will be able to see if we go to this file.

|  |
| --- |
| A screenshot of a computer program  Description automatically generated |

For now, let’s put pass in class ‘User’ and in class 'Agent’ we will remove first and last name because we can use from ‘AbstractClass’ by inheriting it.

|  |
| --- |
|  |

Now let’s delete the database because we already started with the default model in the beginning when we migrated. So, we will delete ‘dbsqlite3’ file. And, we will delete ‘0001\_initial.py’ file.

Now let’s make migrations again in the terminal. Doing this will help us with saving the custom user model we created.

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py makemigrations  Migrations for 'leads':  leads\migrations\0001\_initial.py  - Create model User  - Create model Agent  - Create model Lead |

Here, we can see three models have been created. We can once again check the ‘0001\_initial.py’ file. Now, just like before, after running ‘makemigrations’ we must run migrate.

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py migrate  Operations to perform:  Apply all migrations: admin, auth, contenttypes, leads, sessions  Running migrations:  Applying contenttypes.0001\_initial... OK  Applying contenttypes.0002\_remove\_content\_type\_name... OK  Applying auth.0001\_initial... OK  Applying auth.0002\_alter\_permission\_name\_max\_length... OK  Applying auth.0003\_alter\_user\_email\_max\_length... OK  Applying auth.0004\_alter\_user\_username\_opts... OK  Applying auth.0005\_alter\_user\_last\_login\_null... OK  Applying auth.0006\_require\_contenttypes\_0002... OK  Applying auth.0007\_alter\_validators\_add\_error\_messages... OK  Applying auth.0008\_alter\_user\_username\_max\_length... OK  Applying auth.0009\_alter\_user\_last\_name\_max\_length... OK  Applying auth.0010\_alter\_group\_name\_max\_length... OK  Applying auth.0011\_update\_proxy\_permissions... OK  Applying auth.0012\_alter\_user\_first\_name\_max\_length... OK  Applying leads.0001\_initial... OK  Applying admin.0001\_initial... OK  Applying admin.0002\_logentry\_remove\_auto\_add... OK  Applying admin.0003\_logentry\_add\_action\_flag\_choices... OK  Applying sessions.0001\_initial... OK |

**Query sets & Managers**

Now we will run the following prompt in our terminal. It basically let’s us use python shell inside our Django project where we can do a lot of things with our models as well.

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py shell  Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32  Type "help", "copyright", "credits" or "license" for more information.  (InteractiveConsole)  >>> |

When we run the following prompt, the query set will become empty. We can exit the shell by using ‘exit()’.

|  |
| --- |
| (InteractiveConsole)  >>> from leads.models import Lead  >>> Lead.objects.all()  <QuerySet []>  >>>exit() |

Now let’s create a super user. Django provides us with an Admin Panel for its users. So, we need not worry about creating a separate Admin page or providing authentication feature as Django provides us that feature. Before using this feature, you must have migrated your project, otherwise the superuser database will not be created.

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py createsuperuser  Username: sam  Email address: raisambidh@gmail.com  Password:  Password (again):  Superuser created successfully. |

Now let’s try doing it again in Django shell.

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py shell  Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32  Type "help", "copyright", "credits" or "license" for more information.  (InteractiveConsole)  >>> from django.contrib.auth import get\_user\_model  >>> User = get\_user\_model()  >>> User.objects.all()  <QuerySet [<User: sam>]>  >>> |

Now let’s import Agent. Let’s run the following prompt.

|  |
| --- |
| >>> from leads.models import Agent  >>> admin\_user = User.objects.get(username="sam")  >>> admin\_user  <User: sam> |

By running this user:sam is no longer a query set like before.

Now, let’s create an agent.

|  |
| --- |
| >>> agent = Agent.objects.create(user=admin\_user)  >>> agent  <Agent: Agent object (1)> |

|  |
| --- |
|  |

‘Agent object (1)’ we see in the above prompt is a datatype. We can change by making some changes to the code.

Here we add ‘str’ method. The method ‘str’ allows us to convert an object into a string representation. This method is a dunder method which is usually used when creating models in Django, but it’s also used in other places. Let’s return the user’s email.

|  |
| --- |
| >>> from leads.models import Agent  >>> Agent.objects.all()  <QuerySet [<Agent: raisambidh@gmail.com>]> |

After making a change to the code, let’s run the prompt in Django shell again.

Lastly. We are going to create a lead.

|  |
| --- |
| >>> from leads.models import Lead  >>> sam\_agent = Agent.objects.get(user\_\_email="raisambidh@gmail.com")  >>> sam\_agent  <Agent: raisambidh@gmail.com>  >>> Lead.objects.create(first\_name="Gunja", last\_name="Rai", age=57, agent=sam\_agent)  <Lead: Lead object (1)>  >>> |

Let’s add the str method in lead class as well.

|  |
| --- |
|  |

Now we will restart the shell and try it again. This time we will get a different outcome.

|  |
| --- |
| >>> from leads.models import Lead  >>> Lead.objects.all()  <QuerySet [<Lead: GunjaRai>]>  >>> |

**Django Administrator**

The Django admin site, one of the most powerful parts of Django is the automatic admin interface. It reads metadata from your models to provide a quick, model-centric interface where trusted users can manage content on your site. The admin's recommended use is limited to an organization's internal management tool.

For using default Django admin panel, first we need to run the server using the following prompt.

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py runserver |

Next, we go to the URL ‘http://127.0.0.1:8000/admin’ in our browser.

|  |
| --- |
|  |

This is a default Django admin panel and by logging in using the username and password of super user that we had created; we can use Django provided CRUD features.

|  |
| --- |
|  |

We can use all the features we can see in the image above, but we are going to add all the models we have created.

|  |
| --- |
|  |

For adding models to the admin panel, we need to import the models file in ‘admin.py’ file in leads app.

From the figure above, we can understand that this is the way to import the model classes we created in ‘models.py’. We import all the models to our ‘admin.py’ file.

Now to register the imported models, we will have to use the code we can see in the picture above. Now, let’s check the Django admin panel once again.

|  |
| --- |
|  |

Now we see a new section for the app i.e. ‘Leads’ contains a model called ‘Users’. This is how we register the created models in Django admin panel. Now let’s do similarly for the rest of the models.

|  |
| --- |
|  |

Now we can see the following.

|  |
| --- |
|  |

Now if we go to ‘users’, we will see ‘sam’ and if we go to ‘Leads’, we will see ‘GunjaRai’ and if we go to ‘Agent’, we will see ‘raisambidh@gmail.com’.

|  |
| --- |
|  |

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

|  |
| --- |
|  |

These are the result of whatever had returned to the str method in ‘models.py’ file.

|  |
| --- |
|  |

If we change the return line to any other field, then we will be given output respectively. For example, try replacing ‘email’ to ‘username’ in line 23 and then refresh the Agent page in Django admin panel.

We can play around with Django admin panel. We can change the data we provided to our liking. We can add new and delete the old one.

**Django View**

A view function, or view for short, is a Python function that takes a web request and returns a web response. This response can be the HTML contents of a web page, or a redirect, or a 404 error, or an XML document, or an image . . . or anything, really.

The view itself contains whatever arbitrary logic is necessary to return that response. This code can live anywhere you want, as long as it’s on your Python path. There’s no other requirement–no “magic,” so to speak. For the sake of putting the code somewhere, the convention is to put views in a file called views.py, placed in your project or application directory.

Let’s open ‘views.py’ which we can find in djcrm folder. Let’s first import ‘HttpResponse’ which basically lets us pass a message just like in image bellow.

|  |
| --- |
|  |

After creating a function which passes a message, we then need to mention it in ‘urls.py’ file in leads app.

|  |
| --- |
|  |

Here in this picture, we can see that we have imported the function we just created in ‘views.py’ file. Then we mention the path from which we can check the message on the webpage.

We need to go ‘127.0.0.1:8000’ to see the message.

|  |
| --- |
|  |

**Templates**

Django’s template engine provides a powerful mini-language for defining the user-facing layer of your application, encouraging a clean separation of application and presentation logic. Templates can be maintained by anyone with an understanding of HTML; no knowledge of Python is required.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

First, we will create a folder called ‘templates’ inside our app i.e. ‘leads’. Then we will create another folder called ‘leads’ inside the folder ‘templates. It is standard practice. Django is made to look for templates in this manner.

We then create a html page called ‘home\_page.html’ and create a simple html structure.

|  |
| --- |
|  |

We add h1 tag and p tag. Now let’s edit the in ‘views.py’ file.

|  |
| --- |
|  |

This is how we can return the home page through ‘views.py’ file using render. Now let’s check the 127.0.0.1:8000 link after refreshing it.

|  |
| --- |
|  |

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Now there is another way of doing it which is very common i.e. to create a templates folder into the root of the project as well.

Now this folder is not discovered by default, so we need to tell Django about this folder. We can do that by adding some code in ‘settings.py’ file in djcrm folder.

|  |
| --- |
|  |

In the ‘settings.py’ file, in the templates section we need to add the code we can in code snippet above in line 47. In this way, the new templates can be mentioned to Django.

Now let’s create a new html file in this folder and let’s name it ‘second\_page.html’. We will make a few changes to the code. And just like before, we will need to mention it in ‘views.py’ file.

|  |
| --- |
|  |

|  |
| --- |
|  |

Notice that we haven’t written app names just like before because now it is recognized by Django. Now, let’s see the 127.0.0.1:8000 link once again.

|  |
| --- |
|  |

So, this kind of setup is a little bit different, and it is up to us to use whichever way we prefer. But the second way can be used for organization of html files which we will find out later.

**Context**

This concept is used in Django views all the time. In views.py file, we created a function called ‘home\_page’ and returned render. In the parameter we need to add a new parameter called context. Context is simply a python dictionary in which we can pass keys and values.

|  |
| --- |
|  |

We can also put things this way.

|  |
| --- |
|  |

As we can see there is some data in the context dictionary. Now we will show it in ‘second\_page.html’ file.

|  |
| --- |
|  |

Here in line 12, we have written the key ‘name’ which is from the dictionary inside 2 curly brackets. If we run the server and see, there will be the value of the key.

|  |
| --- |
|  |

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Here we mention the name key and if we had mention age key instead then we would see the value of the age key.

|  |
| --- |
|  |

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

So simply, context is dictionary of data, and we access this data using key and we can show the value of the key in the webpage.

Now we will empty the dictionary then add new data. We will import the model called ‘Lead’ then use calling the query set.

|  |
| --- |
|  |

We can see the same output we saw in the terminal. But it is not the prettiest way of displaying it. Let’s try using for loops with html list.

|  |
| --- |
|  |

We used python for statement and html unordered lists in this way and the output will be as the following.

|  |
| --- |
|  |

Now we know a little bit about Django context and the use of python programming language in the html page. We will be doing these things more throughout the course.

**URL Namespaces**

We recently learned about ‘urls.py’ where we set path for webpages. There is another way to organize URLs. We will be creating a new ‘urls.py’ file inside our app. With a set of URLs within an app, we can include() the URLs of that app in the main project (or within other apps, if we choose!). Then, when we change URLs for the app, you don't need to make changes to the main urls.py for the project.

|  |
| --- |
|  |

|  |
| --- |
| A screen shot of a computer program  Description automatically generated |

In urls.py file in leads app, we add these codes. We will import path and then views here. And then in ‘urls.py’ in djcrm folder we will remove import of views then we import include. The include tag allows you to include a template inside the current template.

In the second picture, in path we see ‘leads/’ which help Django to know the locations of the webpage which we have put in urls.py file inside leads app.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Now it is necessary to specify app name in lead’s URLs file.

|  |
| --- |
|  |

Now let’s go to this URL ‘127.0.0.1:8000/leads/all’.

Lead list and Detail View

First let’s leave the path blank in urls.py because I want to display all the leads in ‘127.0.0.1:8000/leads’. Check the link after changing the code. Also let’s change the name ‘home\_page’ to ‘lead\_list’.

|  |
| --- |
|  |

In ‘views.py’ file, let’s change the name and file path as well. We will also be changing the name from ‘home\_page’ to ‘lead\_list.html’ inside the leads app.

|  |
| --- |
|  |

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

We will then be displaying the leads and then also styling a bit. The codes will be below.

|  |
| --- |
|  |

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Here we have used python ‘for statement’ to display all the available leads in our database. We can display name, first name, last name, age and agent which we had mentioned in our ‘models.py’ file.

Now let’s go to ‘127.0.0.1/8000/admin’ and add new leads.

|  |
| --- |
|  |

I have added a new lead and now let’s check the ‘127.0.0.1:8000/leads’ URL.

|  |
| --- |
|  |

Now we have listed all the list, it would good if we put a link into it. For that, we will need a new view which will handle the request when we go to the link. We will also need a new html file to display. Let’s open ‘views.py’ file first.

In ‘views.py’ file, we will a new function called ‘lead\_detail’ which will help to show detail of the lead in the webpage. In this function, we will be passing ‘request’ and then ‘pk’ which stands for primary key. For now, we will be returning HttpResponse and we will also be printing the ‘pk’ just to see what it contains.

|  |
| --- |
|  |

|  |
| --- |
| A screen shot of a computer program  Description automatically generated |

Now just like before, we always need to remember that after creating a view function, we need to mention it in ‘urls.py’ file.

In the above picture, we import the ‘lead\_detail’ function and then we mentioned the path. In the path, we have mentioned the ‘pk’ inside the angled brackets. Django understands this argument ‘<pk>’ which basically mentions the primary key of the record in which it is stored in the database. So, the URL will be created in this form ‘127.0.0.1:8000/leads/1’ or ‘127.0.0.1:8000/leads/2’ and so on.

|  |
| --- |
|  |

This is how we will be getting the output but to see in primary key, we need to check out the terminal.

|  |
| --- |
|  |

Here in the picture above, number 2 is being displayed which is the primary key. We can try and change the URL to any number like ‘127.0.0.1:8000/leads/3’ which will display number 3 in the terminal.

We can display various things other than id.

|  |
| --- |
|  |

|  |
| --- |
|  |

Now we will be creating a link on each lead which will forward us to the new URL so that we won’t need to type it manually. To do that, let’s open ‘lead\_list.html’ file to create a HTML anchor tag. The code will be below.

|  |
| --- |
|  |

In line 26, we have mentioned the link in this manner. Though we have mentioned the link inside the quotations, Django will understand it and send us to the intended webpage.

Now we want more than a HttpResponse. To do that, we will create a context and then render it to a webpage. We will also be creating a new html file called ‘lead\_detail.html’.

|  |
| --- |
|  |

In ‘lead\_detail.html’ file, we will copy everything from ‘lead\_list.html’ file and then edit some codes as shown below.

|  |
| --- |
|  |

Now in ‘views.py’ file, we will need to render.

|  |
| --- |
|  |

Now doing all this, we will be able to click the link of the lead and it will send to ‘lead\_detail.html’ where we can see the lead’s detail.

|  |
| --- |
|  |

Now let’s also create a back button so we will be sent to the main ‘lead\_list.html’ file. We will add this code in the ‘lead\_detail.html’ file.

|  |
| --- |
|  |

|  |
| --- |
|  |

Forms and Create View

Now we will be making a form through which we will create more leads like the Django admin page. For that we will first create a new function in ‘views.py’ file and then we will create a new template called ‘lead\_create.html’.

|  |
| --- |
|  |

Now in ‘lead\_create.html’ file, we will add the following code.

|  |
| --- |
|  |

Lastly, we will mention these things in ‘urls.py’ file inside ‘leads’ app folder.

|  |
| --- |
|  |

We will first import the function ‘lead\_create’ then we will put the path right above the path consisting of primary key otherwise we will get an error. We will get an error because the path for ‘lead\_create’ file will also be taken as a primary key by Django, so we do it this way.

Now let’s run the server and check ‘127.0.0.1:8000/leads/create’.

|  |
| --- |
|  |

Now we can get started with form logic. We will create a new file inside our app called ‘forms.py’. We will create models inside ‘forms.py’. Basically, we create ‘forms.py’ to create form specific models.

|  |
| --- |
|  |

Now in ‘views.py’ we will add context in ‘lead\_create’ function.

|  |
| --- |
|  |

In ‘lead\_create.html’, we will add the code which we can see in line 13 in the picture below.

|  |
| --- |
|  |

Let’s run the server then check the ‘127.0.0.1:8000/leads/create’ again.

|  |
| --- |
|  |

We can also change the way the forms are, by doing the following edit. We will also be putting the Django pull of form inside html form which will help us POST or GET the data to the database.

|  |
| --- |
|  |

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Whenever we POST the form, we will have to use ‘csrf\_token’. When a user is authenticated and surfing on the website, Django generates a unique CSRF token for each session. This token is included in forms or requests sent by the user and is checked by the server to verify that the request is coming from the authenticated user and not from a malicious source.

Let’s also create a button so we can submit it.

|  |
| --- |
|  |

Now, if we were to hit the submit button, nothing would happen. Let’s check it by printing in ‘views.py’.

|  |
| --- |
|  |

Let’s enter something the form and then hit the submit button. Nothing will happen on the webpage but in the terminal, we can see the following.

|  |
| --- |
|  |

This is how data comes using the POST method. Now we can use its data to create a lead.

Now we will be using python ‘if statement’ where if the form submitted is posted then it will create a lead and if not, I will make it empty.

Now since we have the form variable, we can check if it is valid or not. We are going to print a simple message and the cleaned data of the form. Basically, cleaned data is a nicer formatted version of displaying the form data.

|  |
| --- |
|  |

Now if we submit the form again, we will be able to see the following output in the terminal.

|  |
| --- |
|  |

Now using this information, we will be able to create a lead. Understanding this logic is crucial.

|  |
| --- |
|  |

Now we need an agent field as well because in ‘models.py’, we have mentioned first name, last name, age and agent. Though we have not mentioned agent in ‘forms.py’ file, we are going to see how we can mention it in ‘views.py’. First, we need to import ‘Agent’ and then write the following code.

|  |
| --- |
|  |

|  |
| --- |
|  |

In line 29, we have called the agent and then ‘first method’ which will basically grab the first existing record. Then we create a lead where we have equaled the first and last names, age and agent which are parameters equaled to cleaned data.   
If we were to submit the form, a new lead will be created. Now we will want to redirect us to the leads viewing page and not the form page again. For that we have imported redirect and then in line 36, we have returned redirect to the ‘/leads’ page.

|  |  |
| --- | --- |
|  |  |

**Model Forms**

We have understood the basics of Django forms, now we are going to see how to make it simpler using Django’s model form. Till now we have been using simple Django forms.

In ‘forms.py’ file, we will create a new class called ‘LeadModelForm’ where will create a class called meta.

Meta class is an inner class in Django models. Which contain Meta options(metadata) that are used to change the behavior of your model fields like changing order options, whether the model is abstract or not, singular and plural versions of the name etc.

|  |
| --- |
|  |

Now we will have to edit ‘views.py’ file. First, we will import ‘LeadModelForm’ and then edit the ‘form’ variable.

|  |
| --- |
|  |

Now in ‘leads\_list.html’ file, we will create a link through which we can go to create page.

|  |  |
| --- | --- |
|  |  |

Now let’s go to the create link.

|  |
| --- |
|  |

Here we can see ‘Agent’ field here. And if we drop down, it is like the form in Django admin.

Now in ‘views.py’ file, we will in agent variable, we will call the form data using clean data just like other variables.

|  |
| --- |
|  |

Now, let’s create a new lead.

|  |  |
| --- | --- |
|  |  |

Now a new lead has been created. Let’s add the agent’s information in ‘lead\_detail.html’ file.

|  |  |
| --- | --- |
|  |  |

In conclusion, Django model forms have a bunch of extra functionalities that allow us to save the form and the data that was passed into that form will be saved as a new model in the database. So finally we are going to remove some code and replace it with ‘form.save()’ and it is going to do the same as the commented codes below.

|  |
| --- |
|  |

It works the same way because we specified what model we were working on in ‘forms.py’ file. Now try creating a new lead using the form again.

**Lead Update View**

|  |
| --- |
| A screen shot of a computer screen  Description automatically generated |

Till now we know how to create a lead and display it, now we will try to update it. First, we will create a new html file called ‘lead\_update.html’, secondly, we will create a new function is ‘views.py’ file and lastly, we will mention the URL in ‘urls.py’ file.

|  |
| --- |
| A computer screen shot of a program  Description automatically generated |

|  |
| --- |
|  |

Now to check the update page, we will go ‘127.0.0.1:8000/lead/5/update’ in our browser and then update the page. The number 5 in the URL is an id of the record and we may put any other record’s id number to update it

|  |
| --- |
|  |

|  |
| --- |
| A screen shot of a computer program  Description automatically generated |

This is one of the ways of creating a function but just like before, there is another simpler way to do it and that is ‘LeadModelForm’.

Now let’s check again the URL ‘127.0.0.1:8000/leads/5/update’.

|  |
| --- |
|  |

The form is already filled up because of the use of instance in our code in ‘views.py’.

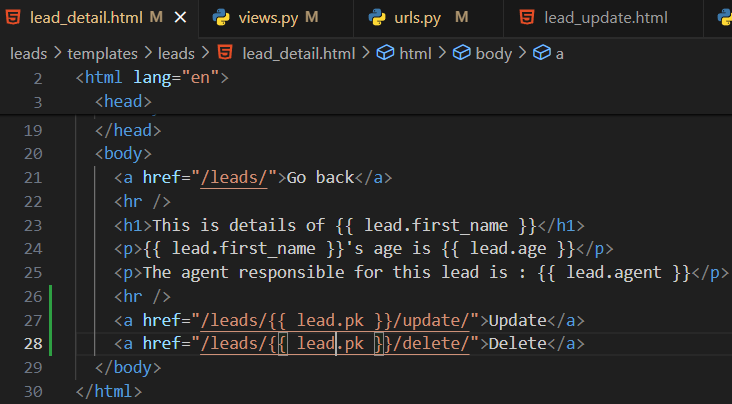
**Lead Delete View**

Now for deleting leads, we could do the same just like we did for creating and updating i.e. creating a html page then create a view function and finally mentioning it in ‘urls.py’ file. But we will use ‘lead\_detail.html’ this time.

|  |
| --- |
|  |

First, we will create a view function just like before but with minor changes. ‘lead.delete()’ is an in built Django function which is self-explanatory. Since we have not created a separate html page to delete a lead, we will redirect us back to the leads page rather than rendering.

|  |
| --- |
|  |

We will mention it in ‘urls.py’ file. And then we will create an anchor tag in the following manner.

|  |
| --- |
|  |

|  |
| --- |
|  |

Trying going to update and delete link to see what happens. It works.

**URL Names**

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.

In previous lessons, we had gone through ‘namespaces’ and now we will know why they are important.  
In ‘urls.py’ file in leads app, we can give each path their names.

|  |
| --- |
|  |

As we can see in this image above, that name has been given to each path and it allows us to reference these paths a lot easier.

For instance, in ‘lead\_list.html’ in the anchor tag’s ‘href’, we can see a link of hard coded line of code.

|  |
| --- |
|  |

If we would want to change the path name in ‘urls.py’ file, then we will have to change the ‘href’ in the entire project wherever it has been called. In the case of a big project, it will take a lot of time to make this change. Now what we can do as an alternative is to use the names, we had given to the URL path.

|  |
| --- |
|  |

Here we have used a new way of writing the link. First, we use curly brackets then ‘%’ sign and then we will write ‘url’, ‘leads’ which is the folder that the html files exist and finally the name of the path. Now we can refresh the webpage, and everything will work seamlessly. Now let’s change it in all the other files as well.

We will change similarly in all the html pages.

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

Extending Templates

Till now we have been repeating ourselves while creating all the html files. In Django, we can create a base html file where we can put all the necessary content and then we can extend it to other html files. Basically, I want all the html files to inherit the content from base html file so that we don’t need to repeat ourselves.

First let’s create a html file called ‘base.html’ in base templates folder.

|  |
| --- |
|  |

|  |
| --- |
|  |

Let’s copy the style tag from ‘lead\_detail.html’ file. We will then delete everything except the body and then extend the base file. We will need to put the body inside block content Django tags.

Now we can use this logic in all the other templates.

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | |
| A screen shot of a computer program  Description automatically generated | |

**Tailwind CSS**

Tailwind CSS is a design system implementation in pure CSS. It is also configurable. It gives developers superpowers. It allows them to build websites with a clean consistent UI out of the box.

|  |
| --- |
|  |

|  |
| --- |
| A close-up of a white screen  Description automatically generated |

Now we will use a great website called ‘www.tailblocks.cc’ where we will first use headers for the navbar.

|  |
| --- |
|  |

We will then view code and then copy the code to a new html file called ‘navbar.html’ inside base template.

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

After this we need to include the navbar html file inside ‘base.html’ like in line 14. Save the file and refresh the browser.

|  |
| --- |
|  |

Now we will remove some of the links from the navbar and then rename the links to Sign up and Login for future use.

We will then add a body from this website.

|  |
| --- |
|  |

We will copy the code and then paste it in a new html page in base html file called ‘landing.html’.

|  |
| --- |
|  |

In ‘landing.html’, we will first extend the ‘base.html’ and inside the block content we will paste the code from tailblock’s website.

Now if we remember those things we did before, if we create a new file, we will have to create a new function if ‘views.py’ file and then mention it in ‘urls.py’. This is what we are going to do.

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

Now let’s create a new link in the navbar so we can go to the leads page. We have created a link for the leads page at line 27. Inside curly braces, we will write the URL namespace of ‘lead-list’.

|  |
| --- |
|  |

|  |
| --- |
|  |

Now let’s style the ‘lead-list’ page a bit. In ‘https://tailblocks.cc/’, inside the features tab, we will be using the following style. Let’s copy the code in ‘lead-list.html’ file. I will be keeping only one box.

|  |
| --- |
|  |

|  |
| --- |
|  |

Now, we will be putting ‘for loops’ inside the ‘tailblock’ codes.

The code for ‘lead-list.html’ is below.

|  |
| --- |
| {% extends "base.html" %} {% block content %}  <section class="text-gray-600 body-font">    <div class="container px-5 py-24 mx-auto flex flex-wrap">      <a href="{% url 'leads:lead-create' %}">Create a new Lead</a>      <hr />      <div class="flex flex-wrap -m-4">        {% for lead in leads %}        <div class="p-4 lg:w-1/2 md:w-full">          <div            class="flex border-2 rounded-lg border-gray-200 border-opacity-50 p-8 sm:flex-row flex-col"          >            <div              class="w-16 h-16 sm:mr-8 sm:mb-0 mb-4 inline-flex items-center justify-center rounded-full bg-indigo-100 text-indigo-500 flex-shrink-0"            >              <svg                fill="none"                stroke="currentColor"                stroke-linecap="round"                stroke-linejoin="round"                stroke-width="2"                class="w-8 h-8"                viewBox="0 0 24 24"              >                <path d="M22 12h-4l-3 9L9 3l-3 9H2"></path>              </svg>            </div>            <div class="flex-grow">              <h2 class="text-gray-900 text-lg title-font font-medium mb-3">                {{ lead.first\_name }} {{ lead.last\_name }}              </h2>              <p class="leading-relaxed text-base">Age: {{ lead.age }}</p>              <a                href="{% url 'leads:lead-detail' lead.pk %}"                class="mt-3 text-indigo-500 inline-flex items-center"                >View this Lead                <svg                  fill="none"                  stroke="currentColor"                  stroke-linecap="round"                  stroke-linejoin="round"                  stroke-width="2"                  class="w-4 h-4 ml-2"                  viewBox="0 0 24 24"                >                  <path d="M5 12h14M12 5l7 7-7 7"></path>                </svg>              </a>            </div>          </div>        </div>        {% endfor %}      </div>    </div>  </section>  {% endblock content %} |

|  |
| --- |
|  |

Now let’s style the create link as well. We will also add the heading tag in ‘lead-list.html’.

|  |
| --- |
| {% extends "base.html" %} {% block content %}  <section class="text-gray-600 body-font">    <div class="container px-5 py-24 mx-auto flex flex-wrap">      <div        class="w-full mb-6 py-6 flex justify-between items-center border-b border-gray-200"      >        <div><h1 class="text-4xl text-gray-800">Leads</h1></div>        <div>          <a            class="text-gray-500 hover:text-blue-500"            href="{% url 'leads:lead-create' %}"            >Create a new Lead</a          >        </div>      </div>      <div class="flex flex-wrap -m-4">        {% for lead in leads %}        <div class="p-4 lg:w-1/2 md:w-full">          <div            class="flex border-2 rounded-lg border-gray-200 border-opacity-50 p-8 sm:flex-row flex-col"          >            <div              class="w-16 h-16 sm:mr-8 sm:mb-0 mb-4 inline-flex items-center justify-center rounded-full bg-indigo-100 text-indigo-500 flex-shrink-0"            >              <svg                fill="none"                stroke="currentColor"                stroke-linecap="round"                stroke-linejoin="round"                stroke-width="2"                class="w-8 h-8"                viewBox="0 0 24 24"              >                <path d="M22 12h-4l-3 9L9 3l-3 9H2"></path>              </svg>            </div>            <div class="flex-grow">              <h2 class="text-gray-900 text-lg title-font font-medium mb-3">                {{ lead.first\_name }} {{ lead.last\_name }}              </h2>              <p class="leading-relaxed text-base">Age: {{ lead.age }}</p>              <a                href="{% url 'leads:lead-detail' lead.pk %}"                class="mt-3 text-indigo-500 inline-flex items-center"                >View this Lead                <svg                  fill="none"                  stroke="currentColor"                  stroke-linecap="round"                  stroke-linejoin="round"                  stroke-width="2"                  class="w-4 h-4 ml-2"                  viewBox="0 0 24 24"                >                  <path d="M5 12h14M12 5l7 7-7 7"></path>                </svg>              </a>            </div>          </div>        </div>        {% endfor %}      </div>    </div>  </section>  {% endblock content %} |

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Now we will style all the other pages as well. First, we will style the lead detail page. We will be using the ecommerce style from the Tailblock website.

|  |
| --- |
|  |

|  |
| --- |
| {% extends 'base.html' %} {% block content %}  <section class="text-gray-600 body-font overflow-hidden">    <div class="container px-5 py-24 mx-auto">      <div class="lg:w-4/5 mx-auto flex flex-wrap">        <div class="lg:w-1/2 w-full lg:pr-10 lg:py-6 mb-6 lg:mb-0">          <h2 class="text-sm title-font text-gray-500 tracking-widest">LEAD</h2>          <h1 class="text-gray-900 text-3xl title-font font-medium mb-4">            {{ lead.first\_name }} {{ lead.last\_name }}          </h1>          <div class="flex mb-4">            <a              class="flex-grow text-indigo-500 border-b-2 border-indigo-500 py-2 text-lg px-1"              >Description</a            >            <a class="flex-grow border-b-2 border-gray-300 py-2 text-lg px-1"              >Reviews</a            >            <a              href="{% url 'leads:lead-update' lead.pk %}"              class="flex-grow border-b-2 border-gray-300 py-2 text-lg px-1"              >Details</a            >          </div>          <p class="leading-relaxed mb-4">            Fam locavore kickstarter distillery. Mixtape chillwave tumeric            sriracha taximy chia microdosing tilde DIY. XOXO fam inxigo juiceramps            cornhole raw denim forage brooklyn. Everyday carry +1 seitan poutine            tumeric. Gastropub blue bottle austin listicle pour-over, neutra jean.          </p>          <div class="flex border-t border-gray-200 py-2">            <span class="text-gray-500">Age</span>            <span class="ml-auto text-gray-900">{{ lead.age }}</span>          </div>          <div class="flex border-t border-gray-200 py-2">            <span class="text-gray-500">Location</span>            <span class="ml-auto text-gray-900">test</span>          </div>          <div class="flex border-t border-b mb-6 border-gray-200 py-2">            <span class="text-gray-500">Phone no.</span>            <span class="ml-auto text-gray-900">444</span>          </div>          <div class="flex">            <button              class="flex ml-auto text-white bg-indigo-500 border-0 py-2 px-6 focus:outline-none hover:bg-indigo-600 rounded"            >              Button            </button>            <button              class="rounded-full w-10 h-10 bg-gray-200 p-0 border-0 inline-flex items-center justify-center text-gray-500 ml-4"            >              <svg                fill="currentColor"                stroke-linecap="round"                stroke-linejoin="round"                stroke-width="2"                class="w-5 h-5"                viewBox="0 0 24 24"              >                <path                  d="M20.84 4.61a5.5 5.5 0 00-7.78 0L12 5.67l-1.06-1.06a5.5 5.5 0 00-7.78 7.78l1.06 1.06L12 21.23l7.78-7.78 1.06-1.06a5.5 5.5 0 000-7.78z"                ></path>              </svg>            </button>          </div>        </div>        <img          alt="ecommerce"          class="lg:w-1/2 w-full lg:h-auto h-64 object-cover object-center rounded"          src="https://dummyimage.com/400x400"        />      </div>    </div>  </section>  <a href="{% url 'leads:lead-list' %}">Go back to leads</a>  <hr />  <h1>This is details of {{ lead.first\_name }}</h1>  <p>{{ lead.first\_name }}'s age is {{ lead.age }}</p>  <p>The agent responsible for this lead is : {{ lead.agent }}</p>  <hr />  <a href="{% url 'leads:lead-update' lead.pk %} ">Update</a>  <a href="{% url 'leads:lead-delete' lead.pk %}">Delete</a>  {% endblock content %} |

I have made a few changes here (go through it). Now while clicking on the detail link, I want the page to lead us to the update page.

|  |
| --- |
|  |

We will then style lead update page. We will copy all the code from the ‘lead-detail’ page to ‘lead-update’ page and then we will make some changes.

|  |
| --- |
| {% extends 'base.html' %}{% block content %}  <section class="text-gray-600 body-font overflow-hidden">    <div class="container px-5 py-24 mx-auto">      <div class="lg:w-4/5 mx-auto flex flex-wrap">        <div class="lg:w-1/2 w-full lg:pr-10 lg:py-6 mb-6 lg:mb-0">          <h2 class="text-sm title-font text-gray-500 tracking-widest">LEAD</h2>          <h1 class="text-gray-900 text-3xl title-font font-medium mb-4">            {{ lead.first\_name }} {{ lead.last\_name }}          </h1>          <div class="flex mb-4">            <a              href="{% url 'leads:lead-detail' lead.pk %}"              class="flex-grow border-b-2 border-gray-300 py-2 text-lg px-1"              >Description</a            >            <a class="flex-grow border-b-2 border-gray-300 py-2 text-lg px-1"              >Reviews</a            >            <a              href="{% url 'leads:lead-update' lead.pk %}"              class="flex-grow text-indigo-500 border-b-2 border-indigo-500 py-2 text-lg px-1"              >Update Details</a            >          </div>          <a href="{% url 'leads:lead-detail' lead.pk %}"            >Go back to {{ lead.first\_name}} {{ lead.last\_name }}</a          >          <form method="post">            {% csrf\_token %} {{ form.as\_p }}            <button type="submit">Submit</button>          </form>          <a            href="{% url 'leads:lead-delete' lead.pk %}"            class="w-1/2 mt-3 flex ml-auto text-white bg-indigo-500 border-0 py-2 px-6 focus:outline-none hover:bg-indigo-600 rounded"          >            Delete          </a>        </div>        <img          alt="ecommerce"          class="lg:w-1/2 w-full lg:h-auto h-64 object-cover object-center rounded"          src="https://dummyimage.com/400x400"        />      </div>    </div>  </section>  {% endblock content %} |

|  |
| --- |
|  |

**Class based views**

Till now we have used function-based views in ‘views.py’ file but this is not the only way we can create views.

“A view is a callable which takes a request and returns a response. This can be more than just a function, and Django provides an example of some classes which can be used as views. These allow you to structure your views and reuse code by harnessing inheritance and mixins.” – Django Documentation.

Class base views in used simply to avoid the repetition of the code in each function-based view.

First we will start by import the default Django Template view and then we will create a class as picture below.

|  |
| --- |
|  |

Then we will import the class name in ‘urls.py’ as such.

|  |
| --- |
|  |

Now try refresh the URL ‘http://127.0.0.1:8000/’. It will work.

Before we move any further, it is important to understand the concept of ‘CRUD’. It simply stands for ‘Create Retrieve Update Delete’. Django has built-in CRUD functionality, and we will now utilize ListView for listing records. Previously, we created functions for create, retrieve, update, and delete operations, but now we will use Django's built-in views and we will see how much code we are going remove.

We will start list view class.

|  |
| --- |
|  |

|  |
| --- |
| A screen shot of a computer program  Description automatically generated |

First, we import ‘ListView’ which we can see in line 6 in the picture above and then we will write the class for the Lead List View. Like before, we will mention the template name and then we will be required to the query set just like while writing a function class view.

Then in ‘urls.py’ inside leads app folder, we will change the function name to class name with ‘as\_view’ method. We will also have to import the class name for this step.

Now the one final change we will be required to make is in ‘lead\_list.html’ file. Since we are not using the context dictionary in the class-based view, we will have to change the ‘for loop’.

|  |
| --- |
|  |

We can see in line 19, we have changed the ‘for loop’ a bit. We have replaced ‘leads’ to ‘object\_list’ and it will work exactly the same. ‘object\_list’ is the default settings which can also be customized to our likings.

|  |
| --- |
|  |

|  |
| --- |
| A screenshot of a computer program  Description automatically generated |

This way, we do not need to change the ‘for loop’.

Similarly, we will create a ‘detail-view’. The steps are the same. We will first import the ‘DetailView’ and the create a class. We will put the context object name as ‘lead’ here just like we did in function-based view.

|  |
| --- |
|  |

Now we will have to change the ‘urls.py’ codes.

|  |
| --- |
|  |

Now, let’s do it for create. For create we do not need the query set as we are working on forms for this page. We will need to import ‘create view’ and then write the following code.

|  |
| --- |
|  |

|  |
| --- |
|  |

Now for update page.

|  |
| --- |
|  |

|  |
| --- |
|  |

Lastly, delete-view.

|  |
| --- |
|  |

We need to create ‘lead\_delete.html’ for this step. We will copy the code form ‘lead\_create.html’ file.

|  |
| --- |
|  |

|  |
| --- |
|  |

From this lesson we can see how little code we have to write while using class-based view.

**Static Files**

Now we are going to set up static files. Files like ‘.css’ and ‘.js’ are static files which will stay the same for all the html pages.

Let us first create a static folder in the base directory and then create CSS and JS files. We will then mention it in ‘settings.py’ file. We also need to specify another setting called static root.

|  |
| --- |
|  |

Now we need to import the setting in root URL. We will also create a separate folder for JS and CSS files.

|  |
| --- |
|  |

We will then link the CSS file to ‘base.html’ file.

|  |
| --- |
|  |

For JS, we will create a new html file.

|  |
| --- |
|  |

This is how static files work.

**Sending Emails**

Although Python provides a mail sending interface via the smtplib module, Django provides a couple of light wrappers over it. These wrappers are provided to make sending email extra quick, to help test email sending during development, and to provide support for platforms that can’t use SMTP.

In ‘views.py’, we will first import ‘send\_mail’ and then we will create a function inside ‘LeadCreateView’ class called ‘form\_valid’ as in the picture below.

|  |
| --- |
|  |

|  |
| --- |
|  |

Authentication

We will be using built in authentication by Django. First, we need to go to the file location which is:

|  |
| --- |
| Env>lib>site-packages>django>contrib>auth>views.py |

|  |
| --- |
|  |

In this ‘views.py’ file, we have got all sorts of functions like login, logout, password reset, etc. which we can use for our own purposes.

Now if we go to:

|  |
| --- |
| templates>registration |
|  |

Here we will need to create out own templates here for authentication purposes.  
We will create a new folder inside templates folder of base app called ‘login.html’.

|  |
| --- |
|  |

Now we need to import ‘LoginView’ function we saw in env folder’s ‘views.py’ in our base ‘urls.py’.

|  |
| --- |
|  |

Now let’s go to the URL ‘http://127.0.0.1:8000/login/’.

|  |
| --- |
|  |

If we login using our super user login credentials, we will be sent to the URL ‘http://127.0.0.1:8000/accounts/profile/’ and we will be shown an error of 404.

|  |
| --- |
|  |

Now after logging in, the login and sign-up button will not make sense so, let’s remove it using some python codes in ‘navbar.html’.

|  |
| --- |
|  |

|  |
| --- |
|  |

Now for a ‘logout’ button whilst logged in we will add a new ‘a tag’ in ‘if statement’.

|  |
| --- |
|  |

For logout, we need to call a ‘logoutview’ function in ‘urls.py’.

|  |
| --- |
|  |

Let’s change the URL href for logout as well.

|  |
| --- |
|  |

Now as we click the logout button, we will be redirected to logout page of default Django admin.

|  |
| --- |
|  |

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

If we create a new user, this will look a bit different. Let’s create a link for signup as well. First, we will create a class view for signup in ‘views.py’ in leads app.

|  |
| --- |
|  |

|  |
| --- |
|  |

Let’s import it to ‘urls.py’ from base app.

|  |
| --- |
|  |

Now set the href for the anchor tag for the signup button.

|  |
| --- |
|  |

Now let’s click the signup button.

|  |
| --- |
|  |

If we sign up a new user, we will get an error saying “Manager isn't available; 'auth.User' has been swapped for 'leads.User'”. For this we need to make a few adjustments. If we go to the ‘UserCreationForm’, we can see that the model is specified particularly to be the default Django user model, so we are going to create our own custom ‘UserCreationForm’.

|  |
| --- |
|  |

|  |
| --- |
|  |

Now let’s import the new ‘CustomerUserCreationForm’ to ‘views.py’ file.

Now, we will be able to signup new user.

This authentication in a nutshell.

Test

Now we are going to see how to write test in Django. We will write a test in Leads app.

|  |
| --- |
|  |

We will write test in ‘test.py’ file in leads app.

|  |
| --- |
|  |

Now for testing, we will do the following in the terminal.

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py test  Creating test database for alias 'default'...  System check identified no issues (0 silenced).  b'\n\n<!DOCTYPE html>\n<html lang="en">\n <head>\n <meta charset="UTF-8" />\n <meta name="viewport" content="width=device-width, initial-scale=1.0" />\n <title>DJCRM</title>\n <link rel="stylesheet" href="/static/css/styles.css" />\n <link\n rel="stylesheet"\n href="https://unpkg.com/tailwindcss@^2/dist/tailwind.min.css"\n />\n </head>\n <body>\n <div class="max-w-7xl mx-auto">\n <header class="text-gray-600 body-font">\n <div\n class="container mx-auto flex flex-wrap p-5 flex-col md:flex-row items-center"\n >\n <a\n class="flex title-font font-medium items-center text-gray-900 mb-4 md:mb-0"\n >\n <svg\n xmlns="http://www.w3.org/2000/svg"\n fill="none"\n stroke="currentColor"\n stroke-linecap="round"\n stroke-linejoin="round"\n stroke-width="2"\n class="w-10 h-10 text-white p-2 bg-indigo-500 rounded-full"\n viewBox="0 0 24 24"\n >\n <path\n d="M12 2L2 7l10 5 10-5-10-5zM2 17l10 5 10-5M2 12l10 5 10-5"\n ></path>\n </svg>\n <span class="ml-3 text-xl">DJ CRM</span>\n </a>\n <nav\n class="md:ml-auto flex flex-wrap items-center text-base justify-center"\n >\n <a href="/leads/" class="mr-5 hover:text-gray-900"\n >Leads</a\n >\n \n <a href="/signup/" class="mr-5 hover:text-gray-900">Sign up</a>\n \n </nav>\n \n <a\n href="/login/"\n class="inline-flex items-center bg-gray-100 border-0 py-1 px-3 focus:outline-none hover:bg-gray-200 rounded text-base mt-4 md:mt-0"\n >\n Login\n <svg\n fill="none"\n stroke="currentColor"\n stroke-linecap="round"\n stroke-linejoin="round"\n stroke-width="2"\n class="w-4 h-4 ml-1"\n viewBox="0 0 24 24"\n >\n <path d="M5 12h14M12 5l7 7-7 7"></path>\n </svg>\n </a>\n \n </div>\n</header>\n \n<section class="text-gray-600 body-font">\n <div\n class="container mx-auto flex px-5 py-24 items-center justify-center flex-col"\n >\n <img\n class="lg:w-2/6 md:w-3/6 w-5/6 mb-10 object-cover object-center rounded"\n alt="hero"\n src="https://dummyimage.com/720x600"\n />\n <div class="text-center lg:w-2/3 w-full">\n <h1\n class="title-font sm:text-4xl text-3xl mb-4 font-medium text-gray-900"\n >\n This CRM built with Django\n </h1>\n <p class="mb-8 leading-relaxed">This CRM helps you manage your leads.</p>\n <div class="flex justify-center">\n <button\n class="inline-flex text-white bg-indigo-500 border-0 py-2 px-6 focus:outline-none hover:bg-indigo-600 rounded text-lg"\n >\n Button\n </button>\n <button\n class="ml-4 inline-flex text-gray-700 bg-gray-100 border-0 py-2 px-6 focus:outline-none hover:bg-gray-200 rounded text-lg"\n >\n Button\n </button>\n </div>\n </div>\n </div>\n</section>\n\n </div>\n \n\n<script src="/static/js/main.js"></script>\n\n </body>\n</html>\n'  .  ----------------------------------------------------------------------  Ran 1 test in 0.044s  OK  Destroying test database for alias 'default'... |

It basically runs the test and then deletes itself in the end.

We can do other things as well. The following code for checking whether the status code is 200 or not.

|  |
| --- |
|  |

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py test  Creating test database for alias 'default'...  System check identified no issues (0 silenced).  .  ----------------------------------------------------------------------  Ran 1 test in 0.020s  OK  Destroying test database for alias 'default'... |

Let’s test template used.

|  |
| --- |
|  |

We can run two tests as well but it not an optimized way to do this.

|  |
| --- |
|  |

|  |
| --- |
| A screenshot of a computer program  Description automatically generated |

Now we can keep the tests in a better way.

We will create a folder called “test” in leads app and then we will first create “\_\_init\_\_.py” and then we will put the tests files as in the picture.

**Auth Permission**

Now we will make leads visible only when we are logged in. We will learn about mixins. First, we will import ‘LoginRequiredMixin’ and then put it in the parameters of the class ‘LeadListView’. This will require us to login to check the leads.

|  |
| --- |
|  |
|  |

Now we will do the same for all the view.

|  |
| --- |
|  |

We will also change the redirect URL. We will add this code in our ‘setting.py’ file.

|  |
| --- |
|  |

Now to create a lead page which shows leads of only specific agents, we will create a new model inside ‘models.py’ file.

|  |
| --- |
|  |

Now always remember that we need to do migrations after creating a model. We will get an error while we make migrations. We will be asked whether to do something which we will do by hitting 1 or 2. So, to avoid the error, we will need to delete our ‘dbsqlite’ file and then run our migrations.

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py migrate  Operations to perform:  Apply all migrations: admin, auth, contenttypes, leads, sessions  Running migrations:  Applying contenttypes.0001\_initial... OK  Applying contenttypes.0002\_remove\_content\_type\_name... OK  Applying auth.0001\_initial... OK  Applying auth.0002\_alter\_permission\_name\_max\_length... OK  Applying auth.0003\_alter\_user\_email\_max\_length... OK  Applying auth.0004\_alter\_user\_username\_opts... OK  Applying auth.0005\_alter\_user\_last\_login\_null... OK  Applying auth.0006\_require\_contenttypes\_0002... OK  Applying auth.0007\_alter\_validators\_add\_error\_messages... OK  Applying auth.0008\_alter\_user\_username\_max\_length... OK  Applying auth.0009\_alter\_user\_last\_name\_max\_length... OK  Applying auth.0010\_alter\_group\_name\_max\_length... OK  Applying auth.0011\_update\_proxy\_permissions... OK  Applying auth.0012\_alter\_user\_first\_name\_max\_length... OK  Applying leads.0001\_initial... OK  Applying admin.0001\_initial... OK  Applying admin.0002\_logentry\_remove\_auto\_add... OK  Applying admin.0003\_logentry\_add\_action\_flag\_choices... OK  Applying sessions.0001\_initial... OK  (env) C:\Users\Hp\Documents\projects\django\getting started with django> |

Now let’s create our super user again.

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py createsuperuser  Username: sam  Email address: raisambidh@gmail.com  Password:  Password (again):  Superuser created successfully. |

**Signals**

Creating user profile manually is not ideal, so now we will try to create the user profile automatically.

In ‘models.py’ file, we are going to create a signal that is going to listen here the event when a user is created. When we create a user, a corresponding user profile needs to be created.

First, we need to import the following.

|  |
| --- |
|  |

Here we have imported ‘post\_save’ from signals but there are a lot of things we could have imported instead. In the instance, ‘post\_save’ basically would help us to save the user profile after we save the user. We could have imported ‘pre\_save’ as well which saves the user profile before the user is saved.

Now we will create a function called ‘post\_user\_created\_signal’.

|  |
| --- |
|  |

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Here in line 37, we have called the signal and use the connect and in the parameter as the function we will create and then we will put User function as the sender (exact model that will send an event). In line 34, we create a function where we will call sender, instance, created and ‘\*\*kwargs’ for future possible arguments. If we now save the existing user from Django admin, we will see instance in the terminal.

|  |
| --- |
|  |

Now we can see the printed instance i.e. User in this case, in the terminal.

|  |
| --- |
|  |

Now let’s print ‘created’ also.

|  |
| --- |
|  |

Now if we save again, we get to see the following in the terminal.

|  |
| --- |
|  |

If we try to create a new user, then we get to see the following. Here ‘test’ is the username I used.

|  |
| --- |
|  |

Now we only want the user profile to be created if the new user is being created. So, we write the following code.

|  |
| --- |
|  |

Now, try creating a new user.

Agent List and Create View

Creating agents is one of the things we missed, so now we will try and create agents. First, we will do the following in the terminal.

|  |
| --- |
| (env) C:\Users\Hp\Documents\projects\django\getting started with django>python manage.py startapp agents |

Then in the explorer, we can see a new app has been created.

|  |
| --- |
|  |

Now go to ‘settings.py’ of ‘djcrm’ folder and add the agents in installed apps as shown below.

|  |
| --- |
|  |

Now in ‘urls.py’ file of ‘djcrm’ folder, we will add ‘agents’ path as shown below in line 13.

|  |
| --- |
|  |

Then we will create a new ‘urls.py’ file for ‘agents’ app.

|  |
| --- |
|  |

For normal list view, we are going to import ‘generic’, and we will create a class-based view. We will also make it login required so we will import the following. For the list view, we will need to specify query set as well.

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |