**DJANGO TUTORIAL**

**TUTORIAL #1 GENERAL**

* General Explaining what Django is
* Web development python framework that includes features such as:
* User authentication
* And more…

**TUTORIAL #2 CREATING A BLOCK**

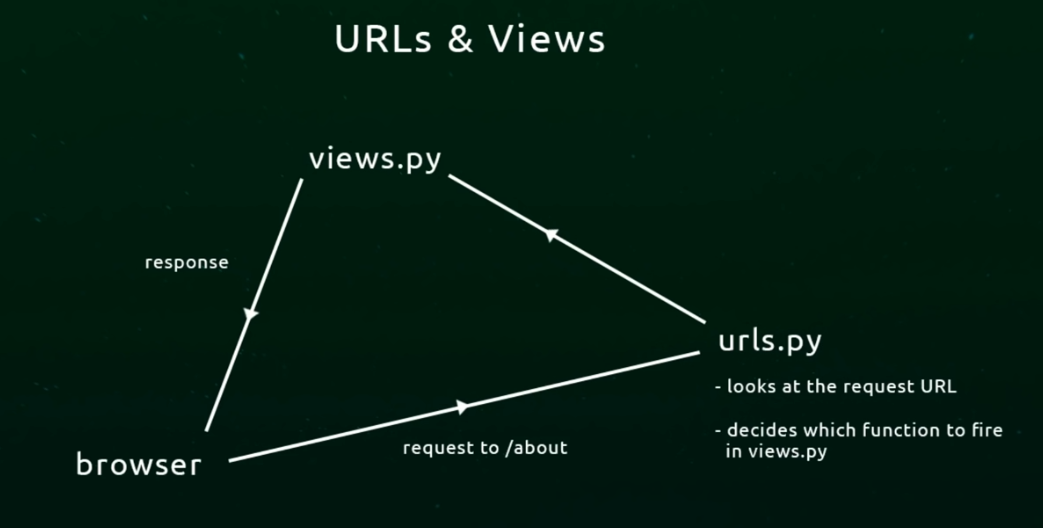
* You have to have Python installed, if not, do it
* Here a parenthesis to indicate that you can create a virtual environment to run django.
* Create a folder where you want to run your virtual environment
* From outside run: *python -m venv folder\_name*
* Activate the virtual environment running folder\_name/Scripts/activate.bat You may need to actually move inside those folders to reach *activate.bat*
* Now you can install django moving into the virtual environment you just created.
* Execute: *python* *-m pip install django* or just run the next bullet.
* We need Django installed. If not, we can use *pip install django*
* Now navigate to a folder (using terminal) where we going to create our django project. With django installed run: *django-admin startproject project-name*
* That creates the project and if you see in the editor, other folder with the same name is created under our **project folder (root folder)**, and the file manage.py That folder is the **base app (root app, first app, main app)** of the project, inside we have the following files:
  + - * + \_ini\_.py
        + settings.py
        + url.py
        + wsgi.py
        + *manage.py* allows to communicate with the server and to perform a series of action to control our website.
* So now move into the project folder to run the following: *python manage.py runserver* This is going to activate the local server where our application is going to run.
* Copy the https address provided and paste it in a browser.

Every time we want to see our app running, we have to active our serve with:

*python manage.py runserver*

When the server is running, we also can see db.sqlite3 file which is database automatically configure to store our data.

**TUTORIAL #3 URLs & VIEWS**



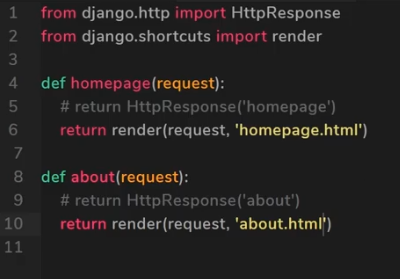
The graphic shows how django use urls to display through functions

path(path, function, name)

**TUTORIAL #4 HTML TEMPLATES**

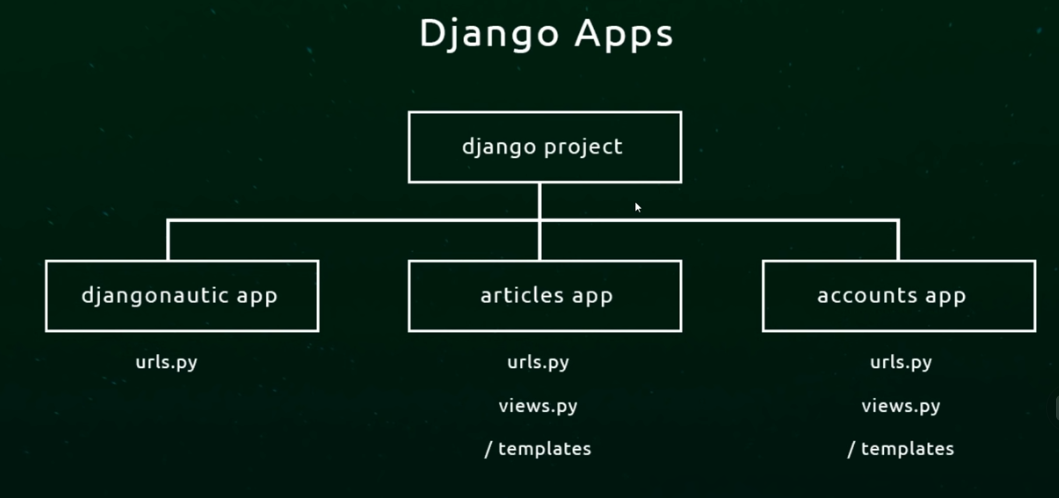
At the same level of your **base app,** you should create a folder named “**template**” to store **.html** files

In the views files import the shown libraries and modify the functions to render the .html file.



Don’t forget to go to settings.py and include the ‘templates’ directory in the DIRS parameters of the TEMPLATES section.

**TUTORIAL #5 DJANGO APPS**



In Django we tend to create separate app or object which perform different aspects of or projects.

When we create a Django project a root folder is created together with a base app. In the base app we created our url’s in the urls.py file and a views.py file.

We split up our application into separated mini apps and each app is going to control a certain section of the web application so each application will have: template folder, urls.py and views.py files. The views.py and the templates folder will be only for that particular app.

To create a new app, go to the root folder of the app, from there run:

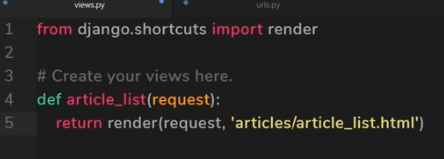
*Python manage.py startapp name\_app*

That will create an directory inside the root folder containing the folder <migrations> , \_\_ini\_\_.py, admin.py, apps.py, models.py, tests.py, views.py .

With ***views.py*** already created Django wants you to create views for this app. Opening this file you can see the ***render*** function already imported.

We will have to create new urls file to control all view for that app which basically will be a copy of urls file from root app so we can just copy & paste.

Inside the templates create a folder named as base app so we have : app\_name>templates>app\_name to name spacing. So when we render the page we do:



Whenever we create an app we have to register that app inside the project or it won’t work.

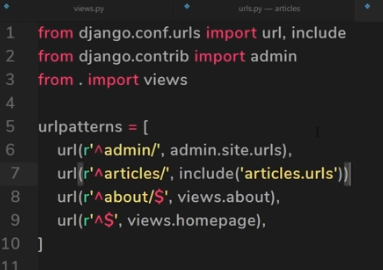
Go to setting.py in that app and scroll down to INSTALLED\_APPS so insert the name of that app at the end: *‘app\_name’,*

Finally, and very important we have to register the new app’s URLs inside the main app’s URLs.

In the main urls file import include function so

*From django.conf.urls import url, include*

And in the urlpattern add the line: url(r’^articles/’, include(‘articles/urls’).



Don’t forget to register the new app’s urls.py in the base app’s url using the *include* function (need to be imported) do: from django.conf.urls import include

**TUTORIAL #6 MODELS**

Allow us to work with databases and store information.

Models in python are a class which represent a table in a database.

Each data can be represented by a model (users, articles, sales, books)

Each model maps to a single table in a database.



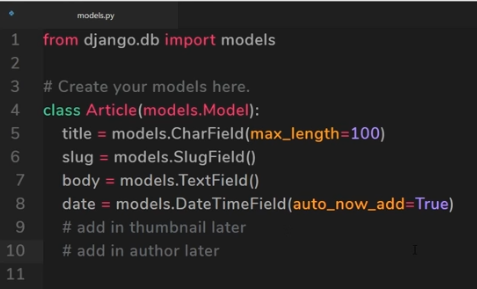
Every line in the table represents a unique record.

In Django we can use ORM to interact with models.

So, in our app, we have our models.py automatically generated, so, there is where we store our models. If we open it, we see something is already imported. So, creating models is creating class.

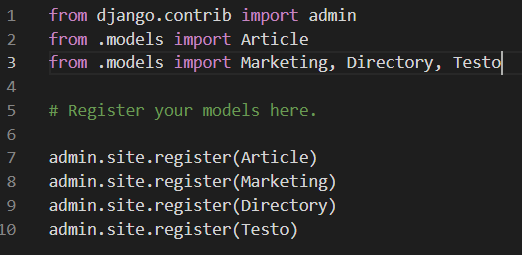
When create models name starts with uppercase. Article.

Fields of models could be text, thumbnail, file, image, number, etc as long as we indicate its type during the creation.



Next thing to do is make a migration so this model is mapped to a table.

Very important is to register our new model in the *admin.py* section of the app where the model was created importing the model required at the beginning.



**TUTORIAL #7 MIGRATIONS**

When we make a model, we must migrate that model to the database, so a table is created.

To migrate, get out of server (go to terminal) and run: *python manage.py migrate*

Before *migrate*, we have to make a migration file then migrate the changes, that for every change.

From the **console** we do *python manage.py makemigrations*

A migrations file is created and stored in the migrations folder inside that app. We can check it any time.

Id field is automatically added to that table.

Then: *python manage.py migrate* to create the table in the database.

**TUTORIAL #8 DJANGO ORM**

* ORM bridges the gap between the model and the database,
* To do that we go to the terminal and open an interactive shell so do: *python manage.py shell*
* Once shell is open we need to import a model so:

*from base\_app.models import Class\_name*

* So now we can use it. if we type Class\_name enter

We get base\_app\_name.models.Class\_name

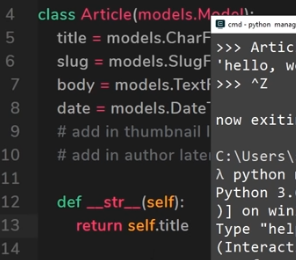
* Retrieving data from table: Article.objects.all()
* To enter info: first create an instance of the article: article = Article()

Now we can do: article.title = “Hellooo”

To save: article.save()

To see the first record: Article.objects.all()[0].title

* Crtl+Z to get out of the shell
* If we include the following function in the models.py to define how the class is gonna look when we use the shell to check its records.



**TUTORIAL #9 THE DJANGO ADMIN**

* The admin area es the area where we control the content of the website or the content of our database or models, users, articles, etc.
* We need to create a super user. To do that we exit the server and from the terminal run the following command: *python manage.py createsuperuser*
* Now type the information required and go back to the server.
* Login to get access to Django administration area.
* Remember to register all models you created so you will be able to see them all in the admin area (tables in the admin ares).
* To register the models, we need to open the admin.py and type:

A screen shot of a computer

Description automatically generated

* Use the method \_\_str\_\_ (in models.py) to show every record’s information.

**TUTORIAL #10 TEMPLATE TAGS**

User to insert python logic data into our templates, could be information from database.

To call that information on the template we use {{ }}. Inside we reference the info using object notation. Example: *{{ articles.title }}.*

To bring python logic (statements) into templates use: {% %} and if we want to put data we user {{ }}.

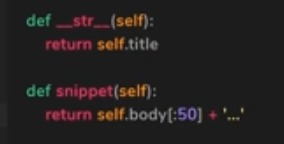
The data to be displayed on the template must be transferred as a *parameters* when defining the functions that respond to urls.

*return render(request, "solutions/solution\_list.html", {'dataFromsolutions': solutions})*

**TUTORIAL #11 MODEL METHODS**

We can include more methods to affect our models as we did with \_\_str\_\_.

Another example is:



We can reference that function in the template as {{ articles.snippet }} and obtain a reduced version of .body content.

**TUTORIAL #12 STATIC FILES & IMAGES**

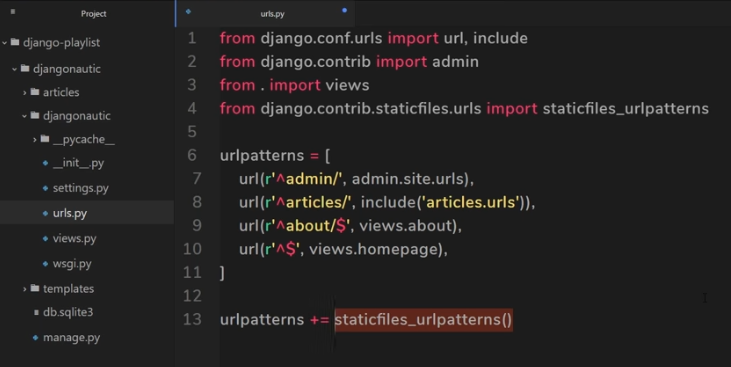
Static files are css, js, images that we can serve to the client, the browser.

Django does not handle those as we know normally is done, that won’t work. We have to explicitly say it we have those in our project and where are they.

In production these files are served by the server which is more efficient.

For this project we have to make Django to handle the serving of url static files.

In the main urls.py of the base app is where we need to tell Django to serve our static files for us adding a line to import *staticfiles\_urlpatterns* and also creatin one line for: *urlpatterns += staticfiles\_urlpatterns()*



Next thing is to jump into the *settings.py* file got to the line *STATIC\_URL = ‘/static/*’ because that’s where we say to Django to look for static files. That name could be any.

To tell Django where to look for that static folder we create another line with a new property so: *STATICFILES\_DIRS=(*

*os.path.join(BASE\_DIR, ‘assets’),*

*)*

If *os* is not present, import it with a line at the top of the file. *import os*

Hard coding is not advisable so we can say at the beginning of the .html. we use *template tags* and say: *{% load static from staticfiles %}* so, now we can use static to replace the path to find the static files in every mention to those we can put: *{% static ‘styles.css’ %}*

Or {% static ‘static/Logo.png’ %}

So, if we do change the STATIC\_DIR from settings.py we do not need to change every single appearance of it through the templates.

**TUTORIAL #13 EXTENDING TEMPLATES**

If we have many templates that repeat in some way, some parts of it, the best thing to do is to keep a base template and only change other parts.

To create a base template use *{% block name\_block %}* to indicate the beginning of the block to be inserted into the base template.

Mark the end with: *{% endblock %}*

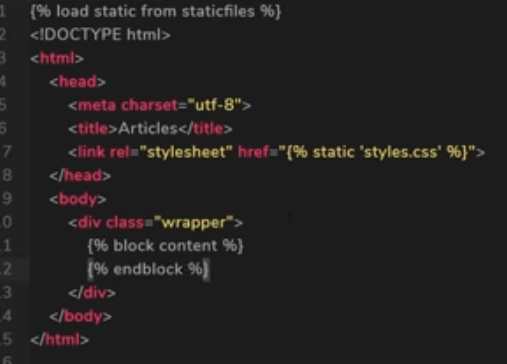
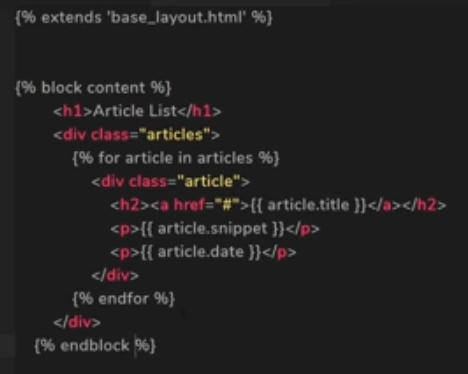
Now, in the file we need to display (using the base template) at the top of the file:

*{% extends “base\_template.html” %}*

Next, we have to indicate where the “block” is providing the name of the block created.

*{% block content %} and*

*{% endblock %}*

**TUTORIAL #14 URL PARAMETERS**

In the urls.py we can assign names to the urls that call views. URL Parameters and Regex.

It is important to put a trailing slash to the URL pattern to ensure consistent URL resolution so, it t= would be: *path('<str:clase>'*

We need to use a name capturing group.

TUTORIAL #15 NAMED URLs

To display specific content depending on what we click on the ref tag on the Front End we can name the URLs.

TUTORIAL #16 ARTICLE DETAILS TEMPLATE

TUTORIAL #17 UPLOADING MEDIA

TUTORIAL #18 ACCOUNTS APP

TUTORIAL #19 USER CREATION FORM

TUTORIAL #20 SAVING USERS

TUTORIAL #21 LOGIN FORM

TUTORIAL #22 LOGGING USERS IN

TUTORIAL #23 LOGGING USERS OUT

TUTORIAL #24 REQUIRING LOGGING

TUTORIAL #25 REDIRECTING AFTER LOGGING

TUTORIAL #26 MODEL FORMS

TUTORIAL #27 FOREIGN KEYS

TUTORIAL #28 CHECKING LOGGING STATUS

TUTORIAL #29 REDIRECTING THE HOMEPAGE

TUTORIAL #30 STYLING THE APP (P1)

TUTORIAL #31 STYLING THE APP (P2)

TUTORIAL #32 SLUGIFYING THE TITLE

TUTORIAL #33

**CONSEJOS**

* When trying to handle static files and images make sure you *import os* in the settings.py file to be possible to use*: STATICFILES\_DIRS=(*

*os.path.join(BASE\_DIR, ‘assets’),*

*)*

* Al the end I created *assets* directory and under it *static* directory where I keep my .css, .js and images.
* The assets directory just under *project folder* at the same level of the *base app.*
* Sometime you can get error pylint(no member) when you importing a model; also says that the object has no records. You can ignore it or deactivate that tool. Using flake8 tool is better.
* When trying to use \_\_str\_\_ in the models .py to be able to see the object in the admin area or entering a record from the admin area, is showing problem. Maybe because now I am using a *ImageField* and *BooleanField* also to store a picture for every article. From StackOverflow I found:

Instead of just using the method:

def \_\_str\_\_(self):

         return self.title, self.body,

we use:

def \_\_str\_\_(self):

return '{} {} {}'.format(self.title, self.slug, self.body, self.picture, self.front, self.date)

Apparently is question of format and worked perfectly and entering new data was ok.

* When having a large text in a file and want to present on a template. Line breaks, formats, etc are not recognised by HTLM. Use the built -in template tag *{{value | linebreaks }}.* At least the text will look better with paragraphs and line breaks.