**sli.do** channel (**#python-web**) **during the exam**.

**python\_web\_basics**

**Create New Django Project + Application**

**INSTALL psycopg2, Pillow, black**

**pip freeze > requirements.txt**

**python manage.py startapp app,app\_auth**

* Add app’s names in INSTALLED\_APPS
* make urls.py in app directory
* path(**''**,include(**'app.urls'**))in urls.py of the project
* add settings for database

from django.urls import path, include  
from django.conf.urls.static import static

from Petstagram2024 import settings

STATIC\_URL = "static/"

STATICFILES\_DIRS = (

BASE\_DIR / 'static',

)

MEDIA\_URL = 'media/'

MEDIA\_ROOT = BASE\_DIR / 'media’

if settings.Debug:

urlpatterns+= static(settings.MEDIA\_URL,document\_root=settings.MEDIA\_ROOT)

**Migrate before models.**

**Dababase for exam – db\_exam /postgres/123456**

DATABASES = {  
 **'default'**: {  
 **'ENGINE'**: **'django.db.backends.postgresql'**,  
 **'NAME'**: **'db\_exam'**,  
 **'USER'**: **'postgres'**,  
 **'PASSWORD'**: **'123456'**,  
 **'HOST'**: **'127.0.0.1'**,  
 **'PORT'**: **'5433'**,  
 }  
}

**python manage.py migrate ->** create tables to keep track of administrators and sessions.

* **create paths is urlpatterns[]**
* **path(‘’,view\_name,name=***’path name for {%url ….%} in template’***)**
* urlpatterns = [  
   path(**''**, home, name=**'home'**),  
   path(**'add/'**, add\_note, name=**'add note'**),  
   path(**'edit/<int:pk>'**, edit\_note, name=**"edit note"**),  
   path(**'delete/<int:pk>'**, delete\_note, name=**"delete note"**),  
   path(**'details/<int:pk>'**, details\_note, name=**"details note"**),  
   path(**'profile/'**, show\_profile, name=**"profile"**),  
  ]
* **! be carefull about the the names of views end name**
* **Create empty views functions, according to urls.py in app.**
* def home(request):  
   return render(request,”home.html”,context)  
    
    
  def add\_note(request):  
   pass  
    
    
  def edit\_note(request,pk):  
   pass  
    
    
  def delete\_note(request,pk):  
   pass  
    
    
  def details\_note(request,pk):  
   pass  
    
    
  def show\_profile(request):  
   pass  
    
    
  def profile(request):  
   pass

**Make MODELS**

class Pet(models.Model):

**name = models.CharField(max\_length=6,choices= …,**

**verbose\_name=’Client name’)**

class Like(models.Model):

**pet = models.ForeignKey(Pet, on\_delete=models.CASCADE)**

def \_\_str\_\_(self):  
 return **f'**{self.name}’

[**https://docs.djangoproject.com/en/3.2/ref/models/fields/**](https://docs.djangoproject.com/en/3.2/ref/models/fields/)

**python manage.py makemigrations**

**python manage.py migrate**

**Django admin site**

**python manage.py createsuperuser - user/123456**

**app1/admin.py - admin.site.register**(model-Pet[, PetAdmin])

class PetAdmin(admin.ModelAdmin):

list\_display=(‘name’,’type’,…..)

list\_filter = (‘name’)

**or** @admin.register(Pet)

class PetAdmin …..

*# https://docs.djangoproject.com/en/4.0/howto/static-files/*

**create directory /static**

**/css**

**/js**

**/img**

**copy files from resources.zip in templates and static – look in html files where should be static files.**

-----------------------------------------------------------

**in .html ->** {% load static %} **– в началото**

**<head>**

**<title>…..**

<link rel="stylesheet" href="{% static 'css/style.css' %}" type="text/css">

**\*\*\*\* Connect html to pictures from task. Print html files.**

**Start index view. -> edit index.html**

**Make forms.py with ModelForm about the model.**

**edit create.html -> {% csrf\_token %}**

*{# !!!add action - default is current url#}* <form method="post" class="furniture-form" action="{% url 'create pet' %}">  
*{# !!! add csrf\_token #}* {% csrf\_token %}  
 {{ form }}  
 <button class="btn btn-primary mt-2" type="submit">Create</button> </form>

if form.is\_valid():  
*# Return True if the form has no errors, or False otherwise.  
#Clean all of self.data and populate self.\_errors and self.cleaned\_data*

**------------------------**

**create Views - using QuerySets** [**https://docs.djangoproject.com/en/3.2/ref/models/querysets/#all**](https://docs.djangoproject.com/en/3.2/ref/models/querysets/#all)

* **create views with corresponding templates – look how to use context in template**

**def pet\_all(request):**

context = {  
 **'pets'**: Pet.objects.all()  
 }  
return render(request, **'pets/pet\_list.html'**, context)

* **insert in urlpatterns according url’s paths**

app1/urls -> path(**''**, pet\_all, name=**'list pets'**),

* **can use this path to view in .html**

<li><a class="nav-link" href**="{% url 'list pets' %}">**All pets</a></li>

**view with id**

path(**'details/<int:pk>'**, pet\_detail, name=**'pet details’) -> /pets/details/2**

def pet\_detail(request, **pk**):  
 pet = Pet.objects.get(pk=**pk**)

**pet.likes\_count = pet.like\_set.count() -> Related objects reference**

context = {  
 **'pet'**: pet,  
 }  
  
 return render(request, **'pets/pet\_detail.html'**, context)

class Like(models.Model):  
 pet = models.ForeignKey(Pet, on\_delete=models.CASCADE)

**https://docs.djangoproject.com/en/3.2/topics/http/shortcuts/**

render,redirect

**-----------------------------**

**edit templates with data from view’s context -> object’s attributes**

**“pet\_url” -> {{pet.image\_url}}**

**“pet\_detail\_with\_pet\_id” -> {% url ‘pet details’ pet.id %}**

{% if not todos %} {% for todo in todos %}

{% else %} {{todo. … }}

{% endif %} {% endfor %}

**Create forms**

[**https://www.w3schools.com/tags/tag\_form.asp**](https://www.w3schools.com/tags/tag_form.asp)

**widgets -** [**https://docs.djangoproject.com/en/3.2/ref/forms/widgets/**](https://docs.djangoproject.com/en/3.2/ref/forms/widgets/)

**document your fields in models**

**Преди архивиране на проекта:**

**pip freeze > requirements.txt**

**pip install -r requirements.txt**

Once you have completed the project, you must **archive it** (zip format) and **upload** it in the contest. You do not need to include in the zip file your **venv**, **.idea**, **pycache**), so you do not exceed the **maximum allowed memory** of **19.53 MB**. You do not need to **export your database either**.

Add requirements.txt + superuser.txt

**Python dot notation:**

**Django classes – package.module.class.method**

**Python function – package.module.function.attribute**

**Make requirements.txt – pip freeze > requirements.txt**

**/pip install -r requirements.txt/**

**Make zip file – without venv, include requirements.txt + user.txt**

**render(request, template\_name, context=None, content\_type=None, status=None, using=None)**

**redirect(to, \*args, permanent=False, \*\*kwargs)**

**# {{variable | filter: args }}**

**https://docs.djangoproject.com/en/4.0/ref/templates/builtins/#built-in-filter-reference**

**Basic model data types and fields list**

| Field Name | Description |
| --- | --- |
| [AutoField](https://www.geeksforgeeks.org/autofield-django-models/) | It An IntegerField that automatically increments. |
| [BigAutoField](https://www.geeksforgeeks.org/bigautofield-django-models/) | It is a 64-bit integer, much like an AutoField except that it is guaranteed to fit numbers from 1 to 9223372036854775807. |
| [BigIntegerField](https://www.geeksforgeeks.org/bigintegerfield-django-models/) | It is a 64-bit integer, much like an IntegerField except that it is guaranteed to fit numbers from -9223372036854775808 to 9223372036854775807. |
| [BinaryField](https://www.geeksforgeeks.org/binaryfield-django-models/) | A field to store raw binary data. |
| [BooleanField](https://www.geeksforgeeks.org/booleanfield-django-models/) | A true/false field.  The default form widget for this field is a CheckboxInput. |
| [CharField](https://www.geeksforgeeks.org/charfield-django-models/) | A field to store text based values. |
| [DateField](https://www.geeksforgeeks.org/datefield-django-models/) | A date, represented in Python by a datetime.date instance |
|  | It is used for date and time, represented in Python by a datetime.datetime instance. |
| [DecimalField](https://www.geeksforgeeks.org/decimalfield-django-models/) | It is a fixed-precision decimal number, represented in Python by a Decimal instance. |
| [DurationField](https://www.geeksforgeeks.org/durationfield-django-models/) | A field for storing periods of time. |
| [EmailField](https://www.geeksforgeeks.org/emailfield-django-models/) | It is a CharField that checks that the value is a valid email address. |
| [FileField](https://www.geeksforgeeks.org/filefield-django-models/) | It is a file-upload field. |
| [FloatField](https://www.geeksforgeeks.org/floatfield-django-models/) | It is a floating-point number represented in Python by a float instance. |
| [ImageField](https://www.geeksforgeeks.org/imagefield-django-models/) | It inherits all attributes and methods from FileField, but also validates that the uploaded object is a valid image. |
| [IntegerField](https://www.geeksforgeeks.org/integerfield-django-models/) | It is an integer field. Values from -2147483648 to 2147483647 are safe in all databases supported by Django. |
| [GenericIPAddressField](https://www.geeksforgeeks.org/genericipaddressfield-django-models/) | An IPv4 or IPv6 address, in string format (e.g. 192.0.2.30 or 2a02:42fe::4). |
| [NullBooleanField](https://www.geeksforgeeks.org/nullbooleanfield-django-forms/) | Like a BooleanField, but allows NULL as one of the options. |
| [PositiveIntegerField](https://www.geeksforgeeks.org/positiveintegerfield-django-models/) | Like an IntegerField, but must be either positive or zero (0). |
| [PositiveSmallIntegerField](https://www.geeksforgeeks.org/positivesmallintegerfield-django-models/) | Like a PositiveIntegerField, but only allows values under a certain (database-dependent) point. |
| [SlugField](https://www.geeksforgeeks.org/slugfield-django-models/) | Slug is a newspaper term. A slug is a short label for something, containing only letters, numbers, underscores or hyphens. They’re generally used in URLs. |
| [SmallIntegerField](https://www.geeksforgeeks.org/smallintegerfield-django-models/) | It is like an IntegerField, but only allows values under a certain (database-dependent) point. |
| [TextField](https://www.geeksforgeeks.org/textfield-django-models/) | A large text field. The default form widget for this field is a Textarea. |
| [TimeField](https://www.geeksforgeeks.org/timefield-django-models/) | A time, represented in Python by a datetime.time instance. |
| [URLField](https://www.geeksforgeeks.org/urlfield-django-models/) | A CharField for a URL, validated by URLValidator. |
| [UUIDField](https://www.geeksforgeeks.org/uuidfield-django-models/) | A field for storing universally unique identifiers. Uses Python’s UUID class. When used on PostgreSQL, this stores in a uuid datatype, otherwise in a char(32). |

**Relationship Fields**

Django also defines a set of fields that represent relations.

| Field Name | Description |
| --- | --- |
| [ForeignKey](https://www.geeksforgeeks.org/python-relational-fields-in-django-models/) | A many-to-one relationship. Requires two positional arguments: the class to which the model is related and the on\_delete option. |
| [ManyToManyField](https://www.geeksforgeeks.org/python-relational-fields-in-django-models/) | A many-to-many relationship. Requires a positional argument: the class to which the model is related, which works exactly the same as it does for ForeignKey, including recursive and lazy relationships. |
| [OneToOneField](https://www.geeksforgeeks.org/python-relational-fields-in-django-models/) | A one-to-one relationship. Conceptually, this is similar to a ForeignKey with unique=True, but the “reverse” side of the relation will directly return a single object. |