# **---------------- Django ----------------**

# **Основы**

**python -m venv name\_venv** создание виртуального окружения python

**pip list** показывает все модули установление в вирутальном окружении (или просто на пк)

**pip freeze или pip freeze > req.txt** показывает все сторонние модули, которые мы сами установили в вирутальное окружение

**pip install -r req.txt** скачиваение всех зависимостей из указаногго файла

**pip uninstall -y -r req.txt** удаление всех модулей указаных в текстовом файле

**python -m pip install --upgrade pip** обновление покатного менеджера pip до последней версии

**django-admin** выводит все команды django

**django-admin check** проверка всего проекта на ошибки

**python manage.py shell** оболочка проекта

**django-admin startproject my\_project** создание проекта

**python manage.py startapp my\_app** создание приложения внутри проекта django

**python manage.py createsuperuser** создание одного пользователя-администратора

**python manage.py collectstatic** собирате все статические файлы , кеширует их и помещает в путь STATIC\_ROOT

# **Базы Данных ( Модели )**

**python manage.py makemigrations** создание файла миграции(определяет структура базы данных) Храниться в файле 0001\_initial.py

**python manage.py migrate** применение изменений в базе данных

**python manage.py sqlmigrate** отображает в терминале инструкции SQL, которые мы зделали

# **Django Extensions**

**python manage.py shell\_plus --print-sql** в етой оболочке будут показываться SQL инструкции, которые мы зделали в командной строке

# **Тестирование (Unittest)**

# **python manage.py test my\_app** провди тестирование указаного одного приложения

**python manage.py my\_app.my\_folder.my\_file** проводим тестирование только одного указаного файла в нашем приложении

# **Ipython**

**ipython** вызов улучшеного интепритатора

# **Coverage**

**coverage run --source="." manage.py my\_folder .** проверка написали на покрытие тестами наших функций

**coverage report** выводим анализ в терминал

**coverage html** создает папку с результатом теста, который можна открыть через index.html

# **Отношения Таблиц друг к другу**

**my\_column = models.ForeignKey(“Other\_Table”, on\_delete = models.CASCAD , related\_name = “name\_manager”)** отношение многие к одному

**my\_column = models.ManyToMany(“Other\_Table”, on\_delete = models.PROTECT , related\_name = “name\_manager” )** отношение многи ко многим

**my\_column = models.OneToOne(“Other\_Table” , on\_delete.SETNULL , related\_name = “name\_manager”)** отношение один к одному

**Other\_Table** название другой таблици в которую будут вставляться значения в текущую таблицу

**on\_delete** указывает что будет на место значение в текущей таблице, если удаляем значение во второй таблице, на которую ссылаеться текущая

**related\_name** указывает названия менеджера для обращения в другой таблице из текущей

# **CRUD**

**ClassBase.objects.create(my\_col1 = “my\_value1” , my\_col2=”my\_value2”)** создание новой ячейки данных

**ClassBase.objects.get(id = my\_value)** получаем данные ячейки из базы

**ClassBase.objects.get(id = my\_value).delete( )** удаление указаной ячейки из базы

# **Одна Запись в Модели**

**my\_var -> Моя Запись в Базе Данных (Модели)**

**my\_var = ClassBase (my\_property = “text”)** создание записи в таблице (не сохраняеться)

**my\_var.save( )** сохранение записи в таблице

**my\_var.update(my\_column= 1)** изменение значений колонки в записи в указаной колонке

**my\_var.delete( )** удаление записи из таблици

**my\_var = \_** нижняя черта содержит последний созданый обьект

**my\_var.pk** или **my\_var.id** выводит значение primary key (id) записи в базе

**my\_var.my\_column1.all( )** выводит значение указаной колонки указаной ячейки в модели

**ClassBase.objects.all()** выводит все записи из указаной базы данных

**ClassBase.objects.all()[ 0 ]** или **ClassBase.objects.all([ : 3 ])** возвращает записи по индексу или по срезу

**ClassBase.objects.filter(my\_column= “text”)** возвращает запись из базы данных только если виражение возвращает True

**my\_var.set ( my\_var2 )** или **my\_var.set ( [my\_var1 , my\_var2] )** изменение пустых строк на значение в третьей таблице, отношение многие к многим

**my\_var.add(my\_var2)** добавление значения в третьей таблице, отношения многие ко многим

**my\_var1.remove(my\_var2)** удаление значение в третьей таблице, отношения многие к многим

# **Основные Операторы**

**ClassBase -> Контекстный Менеджер для обращения к Базе Данных (Модели)**

**ClassBase.objects.filter( Q(pk\_id = 5) | Q(value = True))** обьект Q позволяет работат с условиями указывая свой оператор | (OR) &(AND) . ~Q(pk\_id = 3 ) означает NOT

**ClassBase.objects.first**(last)**()** возвращает первый или последнеий запись из базы данных

**ClassBase.objects.earliest(latest)(“my\_column”)** возвращает первую или последнюю запись в базе по указаному столбцу

**ClassBase.objects.get\_previous\_by\_my\_column ()** (get\_next\_by\_my\_column ()) возвращает предидущий или следующий леемент в базе по указаному столбцу

**my\_var.my\_manager** обращение от значения из текущей таблици в таблицу, где может находиться

**my\_var.my\_manager.exists()** проверят на существование в другой таблице

**my\_var.my\_manager.count()** проверят запись на количество в другой таблице

**ClassBase.objects.order\_by(“my\_column”)** вывод отсортированых записей в базе данных

**ClassBase.objects.filter(pk\_\_gt = F(“my\_column”))** позволяет обращаться к значения колонки по ее названию (from django.db.models import F )

**my\_var.my\_column = F(“my\_column”) + 1** увеличение текущего значения колонки записи на 1 (можно также уменьшать)

**my\_var = ClassBase.objects.all().annotate(my\_column = Value(my\_value))** возвращает такую же таблице, но с добавленой колонкой и указаными внутри ее значениями

**ClassBase.objects.all().annotate(new\_column=True)**  временно добавляет новую колонку в указаную таблицу из указаным значение True

**ClassBase.objects.aggregate(Min(“my\_column”))** позволяет работать с агрегирующими функциями такими как Count , Min , Max , Avg , Sum . Count – возвращает количество записей в указаной колонке. **values(“my\_column1” , “my\_column2”)** будет возвращать в выборке только указанные колонки

**ClassBase.objects.values(“name1\_\_name2”)** выводиться значения из таблици на которую ссылаеться текущая. name1 – название колонки в текущей таблице , name2 – название колонки в другой таблице на которую ссылаеться текущая таблица

**ClassBase.objects.annotate(Count(“name\_manager”))** групировка GROUP BY по всем колонкам текущей таблици

# **Import Django**

from . берем из текущей директории

from django.shortcuts import **render , redirect**

from django.contrib import **admin**

from django.urls import **path** , **re\_path** , **include**

from django.urls import **reverse** , **reverse\_lazy**

from django.db import **models**

from django.views.generic.base import **TemplateView ,**

from django.views.generic.list import **ListView**

from django.views.generic.detail import **DetailView**

from django.views.generic.edit import **FormView , CreateView , UpdateView , DeleteView**

from django.contrib.auth.decorators import **login\_required , permission\_required**

from django.views.decorators.http import **require\_http\_methods**

from django.contrib.auth.mixins import **LoginRequiredMixin , PermissionRequiredMixin**

from django.contrib.auth.views import **LoginView** , **LogoutView , PasswordChangeView , PasswordChangeDoneView**

**Измение Пароля Через Email -> PasswordResetView -> PasswordResetDoneView -> PasswordResetConfirmView -> PasswordResetCompleteView**

from django.contrib.auth import **authenticate, login, logout , get\_user\_model**

from django.contrib.auth.forms import **AuthenticationForm , UserCreationForm**

from django.contrib.auth.backends import **BaseBackend**

from django.core.validators import **MinLengthValidator, MaxLengthValidator , BaseValidator**

from django.contrib.auth.models import **User**

from django.contrib.auth.models import **AbstractUser**

from django.forms import **model\_to\_dict**

# **-------------- Django REST -------------**

# **Views**

**APIView** базовый класс представления, от которого наследуються все остальные

**CreateAPIView**  для создания записей по post запросу

**ListAPIView** для чтения списка записей по get запросу

**RetrieveAPIView** чтение одной записи по get запросу

**DestoyAPIView** удаление одной записи по delete запросу

**UpdateAPIView** изменение одной записи по put или patch запросу

**ListCreateAPIView**  для чтения списка записей по get запросу и создания етого списка по post запросу

**RetrieveUpdateAPIView** чтение одной записи по get запросу и изменение ее по put запросу

**RetrieveDestroyAPIView** чтение одной записи по get запросу и удалениее ее по delete запросу

**RetrieveUpdateDestroyAPIView**  чтение одной записи по get запросу, изменение ее по put или patch запросу и удалениее ее по delete запросу

# **Import Django REST**

from rest\_framework import **generics**

from rest\_framework import **serializers**

from rest\_framework.views import **APIView**

from rest\_framework.response import **Response**

from rest\_framework.renderers import **JSONRenderer**

from rest\_framework.parsers import **JSONParser**

from rest\_framework.generics import **CreateAPIView , ListAPIView , RetrieveAPIView , DestroyAPIView , UpdateAPIView , ListCreateAPIView , RetrieveUpdateAPIView , RetrieveDestroyAPIView , RetrieveUpdateDestroyAPIView**

from rest\_framework import **viewsets**

from rest\_framework import **routers**

from rest\_framework.decorators import **action**

# **----------------- Celery -----------------**

**pip install celery** установка самого celery внутрь проекта

**celery -A config worker -l info** запуск celery, config береться из config.celery.py

# **----------------- Redis ------------------**

**pip install redis** установка redis внутри проекта

**docker run --name my\_redis -d -p 6379:6379 redis --> docker exec it my\_redis /bin/sh** установка redis в докер контейнере и его запуск (по причине того что redis не поддерживаеться на windows)

**pip install redis** установка самого redis в проект

**redis-cli** запуск самого redis внутри контейнера

**ping** проеверка redis на работу, должны принять ответ PONG

# **Основы**

**keys \*** выводит все названия ячеек

**rename old\_name new\_name** изменение названий ячеек

# **Cтроки**

**set my\_key my\_value** строки , ключ : значение

**get my\_key** получение значения ключа по его названию

**type my\_key** возвращает тип ячейки

**del my\_key** удаление ячейки

**setex my\_key my\_value 10** укызваем время жизни для ячейки 10 секунд

# **Списки**

**lpush my\_list my\_value** создание списка и добавление слева елемента

**rpush my\_list my\_value** создание списка и добалвение елемента справа

**lrange my\_list 0 -1** выводим значения списка по срезу (можно выводить только по серзам)

**lpop my\_list** удаляет первый елемент списка

**llen my\_list** выводит длину списка

**lmove my\_list1 my\_list2 LEFT LEFT** вырезаем первый елемент из первого списка и вставляем во второй, при етом етот елемент из первого списка удаляеться. Первый LEFT берем самы левы елемент из первого списка , а второй LEFT вставляем в левую часть второго списка

# **Хеш-Таблици**

**hset my\_set my\_col1 my\_value1 my\_col2 my\_value2** создание хеш-таблици

**hget my\_set my\_col1** возвращает все значения указаной одной ячейки указаной хеш-таблици

**hmget my\_set my\_col1 my\_col2** возвращает значения указаных ячеек из хеш-таблици

**hgetall my\_set** выводит все названия ячеек и все их значения

**hincrby my\_set my\_col 10** добавляем к указаной ячейке +10 к значению

**ttl my\_set** возвращает время жизни хеш-таблици, -1 ето неограниченое количество времени

**expire my\_set 10** указываем время жизни хеш-таблици 10 секунд