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

Для проектов Flask определим какой-нибудь каталог на жестком диске.

Открываем PyCharm и создаем проект. Используем PIPENV

Указываем папку проекта **Wholesale** (создаем заранее ?)

## **Заходим в Терминал и сетапим Flask та flask-sqlalchemy:**

pipenv install flask

pipenv install flask-sqlalchemy

pipenv install Flask-Migrate

Формуємо основні файли FLASK…

Підключення БД.

Визначаємо налаштування БД –

SQLALCHEMY\_DATABASE\_URI = os.environ.get('DATABASE\_URL') or 'sqlite:///' + str(BASE\_DIR / "data" / "trade.db")

Визначаємо моделі таблиць:

! для ForeignKey назва таблиці з БД, а не назва Класу !

lass Unit(db.Model):  
 *""" --- Одиниці виміру --- """* unit\_code = db.Column(db.String(10), primary\_key=True)  
 unit\_name = db.Column(db.String(15), nullable=False)  
  
 def \_\_repr\_\_(self):  
 return f"Unit {self.unit\_name}"  
  
  
class Item(db.Model):  
 *""" --- Клієнти ---"""* id = db.Column(db.Integer, primary\_key=True)  
 item\_name = db.Column(db.String(50), nullable=False)  
 unit = db.Column(db.String(10), db.ForeignKey('unit.unit\_code'), nullable=False)  
 service = db.Column(db.Boolean, nullable=False, default=False)  
 item\_description = db.Column(db.String(100), nullable=False)  
  
 def \_\_repr\_\_(self):  
 return f"Item {self.id}, {self.item\_name}"

Далі у вікні терміналу:

**flask db init**

Оновлення таблиць при коригування моделей:

**flask db migrate -m "Initial migration."**

**flask db upgrade**

При установке Django в папке виртуальной среды устанавливается утилита django-admin.

А на Windows также исполняемый файл django-admin.exe. Их можно найти в папке виртуальной среды, в которую производилась установка Django

django-admin предоставляет ряд команд для управления проектом Django.

В частности, для создания проекта применяется команда startproject. Этой команде в качестве аргумента передается название проекта.

**## Создаем проекта nem\_portal**

django-admin startproject nem\_portal

И после выполнения этой команды в текущей папке будет создан подкаталог nem\_portal.

manage.py: выполняет различные команды проекта, например, запускает приложение hello

\_\_init\_\_.py: данный файл указывает, что папка, в которой он находится, будет рассматриваться как модуль. Это стандартный файл для Python.

settings.py: содержит настройки конфигурации проекта

urls.py: содержит шаблоны URL-адресов, по сути определяет систему маршрутизации проекта

wsgi.py: содержит свойства конфигурации WSGI (Web Server Gateway Inerface). Он используется при развертывании проекта.

asgi.py: название файла представляет сокращение от Asynchronous Server Gateway Interface и расширяет возможности WSGI, добавляя поддержку для вщавимодействия между асинхронными веб-серверами и приложениями.

**## Создаем приложение, которое будет реализовывать ПО формирования Графика**

python manage.py startapp app\_workingmode

**## устанавливаем модули для работы с MS SQL**

pipenv install pyodbc

pipenv install django-pyodbc-azure

**## Делаем настройки в settings.py**

**## настраиваем запуск сервера из PyCharm: python manage.py runserver 0.0.0.0:9000**

**## Запускаем и получаем стартовую страницу в браузере**

**## Виконуємо міграцію для створення системних таблиць Джанго в БД**

python manage.py migrate

**\*\*\*\*\* створюємо моделі для наших обєктів**

**models.py**

**## створюємо та виконуємо міграцію для створення таблиць в БД**

python manage.py makemigrations

python manage.py migrate

**\*\*\* запуск шела Джанго для розуміння работи функцій в Терминале PyCharm**

python manage.py shell

**\*\*\*\* створюємо суперадміна для роботи з адмін-панеллю**

python manage.py createsuperuser

**\*\*\*\* заходимо в адмін-панель та створюємо користувачів та групи**

**\*\*\* формуємо requirements.txt**

python manage.py shell

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\* GIT**

**## Ініціалізація git-репозиторію**

git init

**Додаємо файл** .gitignore

**Додаємо всі файли в проект**

git add . .

**Видаляємо з репозиторію** git rm --cached .idea/\*.xml

**## \*\*\*\*\*\*\*\* Синхронізація з GitHub:**

**Створюємо на GitHub репозиторій**

Створюємо порожню папку Проекту.

Переходимо в неї

**git clone** [**https://github.com/OlegM-56/NEM\_DJANGO.git**](https://github.com/OlegM-56/NEM_DJANGO.git) **.**

**Копіюємо проект в цю папку**

**git add .**

**git commit**

**git push - завантажити зміни з ПК в GitHub репозиторій**

**git pull - завантажити репозиторій на ПК**

**==== ЗАПУСК в промислову експлуатацію**

12. APPEND\_SLASH = True

Тепер, якщо користувач введе URL-адресу /example, Django автоматично додасть слеш та перенаправить запит на /example/.

Не працює ???

13. gunicorn під віндою не працює !

14. Отримання секретного ключа:

from django.core.management.utils import get\_random\_secret\_key

print(get\_random\_secret\_key())

записуємо в змінну середовища!

Развернуть python проект с docker на примере django + nginx + gunicorn + postgresql

ОПУБЛИКОВАНО 9 ОКТЯБРЯ 2021 · ОБНОВЛЕНО 6 ФЕВРАЛЯ 2022Целью статьи не пояснить, что такое [docker](https://www.docker.com/" \t "_blank" \o "Docker), а показать как быстро собрать проект на примере django + nginx + gunicorn + postgresql

Docker-Compose

Да, именно он помог мне избавиться от боли в мозгу, когда я не мог склеить два контейнера. Именно он поможет нам быстро развернуть проект и запустить. Честно признаюсь, хоть я и не сторонник запуска с docker на production, но один сайт я все таки запустил. И он не просто работает, еще и заказчик доволен ).

Что нам дает docker-compose? С его помощью мы соберем контейнеры, прицепим дисковое пространство, установим необходимые пакеты и запустим все это. Про него можете почитать [здесь](https://docs.docker.com/compose/), а потом продолжим историю.

Для начала же давайте определимся с файловой структурой. Здесь я приведу пример того, как мне удобно работать. Я уверен, что ты сообразительный малый и, если нужно, сам сообразишь что на что поменять и как все это улучшить. Основную директорию, где мы будем создавать все файлы и откуда будем все запускать, я кратко назову **DEV\_PATH**. Это может быть C:/work/site1/ или /home/user/site2 или где то еще. Жирным выделяю названия директорий. Курсивом — файлов.

NEM-IT-Projects – каталог на сервері для розгортання проектів

DEV\_PATH = / NEM-IT-Projects/ NEM\_PORTAL – для проекту

* **DEV\_PATH**
  + **docker**
    - **​nginx**
      * **​***sitename.conf (nem\_portal.conf*
    - **python**
      * **​***Dockerfile*
  + **<project>** (Это директория с python проектом. Название проекта, соответственно, таким, как ты назовешь его. )
    - **logs**
    - **<project>**
      * *settings.py*
      * *wsgi.py*
    - *gunicorn.py*
    - *requirements.txt*
  + *docker-compose.yml*

**<project>** = nem\_portal

В папці розробки все підчищаємо та передаємо в GitHub

Заходимо в порожню папку nem\_portal та клонуємо з GitHub

**git clone** [**https://github.com/OlegM-56/NEM\_DJANGO.git**](https://github.com/OlegM-56/NEM_DJANGO.git) **.**

Завантаження нових/змінених файлів

**git pull**

Dockerfile

Его мы создали для контейнера python. Именно здесь больше работы приходится производить, так как нам нужно установить приложения, пакеты и примонтировать директории. Ниже содержимое файла.

FROM python:3.6

COPY ./<project> /srv/www/<project>

WORKDIR /srv/www/<project>

RUN pip install -r requirements.txt

Построчно:

1. Указываем имя контейнера, на основе кторого будем это все собирать. Я использую официальный контейнер под python 3.6. Ты можешь выбрать свою версию.
2. Указываем что и куда мотировать. Я использую директорию /srv/www/имя\_проекта для лаконичности. Важно понимать, что лучше указывать путь до проекта так, как он будет выглядеть на твоем сервере. Приближаем окружение к боевой системе.
3. Указываем эту же директорию как точку входя при выполнении следующих команд.
4. Запускаем установку пакетов из requirements.txt. После этого твой образ сохранится уже с этими пакетами и тебе не придется каждый раз пересобирать их. главное заранее подготовить файл со всеми нужными пакетами. Каждый раз, меняя requirements.txt, придется перезапускать сборку образа и, соответственно, с нуля устанавливать эти пакеты.

sitename.conf

Обычный файл конфигурации nginx. Он нужен для того, чтобы сразу после запуска nginx «скушал» наш проект.

# portal

server {

listen 8080; # nginx будет слушать этот порт.

server\_name localhost;

charset utf8;

autoindex off;

access\_log /srv/www/<project>/logs/<project>\_access.log;

error\_log /srv/www/<project>/logs/<project>\_error.log error;

set $project\_home /srv/www/<project>;

location / {

root $project\_home;

try\_files $uri @<project>;

}

location @<project> {

proxy\_pass http://python:8000; # gunicorn запускается в контейнере python и слушает порт 8000

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

}

На этом настройка nginx заканчивается.

gunicorn.py

Если нет желания запускать с ним, то, пожалуй, можете через manage.py runserver или иначе. Но мне gunicorn кажется весьма удачным решением. Вот конфигурационный файл.

from multiprocessing import cpu\_count

from os import environ

def max\_workers():

return cpu\_count()

bind = '0.0.0.0:' + environ.get('PORT', '8000')

max\_requests = 1000

worker\_class = 'gevent'

workers = max\_workers()

env = {

'DJANGO\_SETTINGS\_MODULE': '<project>.settings'

}

reload = True

name = 'Project\_name'

не забываем в requirements.txt добавить gevent и gunicorn

docker-compose.yml

Наконец то дошли к самому сладкому. Приведу пример файла, а в нем все пояснения, чтобы мы не потеряли что для чего делалось.

version: '3'

# хранилища

volumes:

pgdata:

driver: local

services:

nginx:

# при падении будет стараться подняться

restart: always

# только свежий nginx

image: nginx:latest

# слушает порт 8080

expose:

- 8080

# мапаем порт 80 на его 8080. Тогда сайт будет доступен по адресу localhost. Убедись, что порт у тебя не занят.

ports:

- "80:8080"

# монтируем только те директории, в которых лежит статика, т.к. nginx с динамикой не будет работать. Также директорию с логами и файл настройки, который мы подготовили.

volumes:

- ./<project>/static:/srv/www/<project>/static

- ./<project>/media:/srv/www/<project>/media

- ./<project>/logs:/srv/www/<project>/logs

- ./docker/nginx:/etc/nginx/conf.d

# и nginx зависит от контейнера python. Т.е. python должен быть запущен первым

depends\_on:

- python

python:

restart: always

# указываем откуда собирать образ

build:

context: .

dockerfile: docker/python/Dockerfile

# монтируем директорию проекта

volumes:

- ./<project>:/srv/www/<project>

expose:

- 8000

ports:

- 8000:8000

# запускаем gunicorn

command: "gunicorn -c gunicorn.py <project>.wsgi"

postgres:

# Ниже даже расписывать не хочу, насколько все просто: логин, пароль, БД, порты и т.д.

image: postgres:9.3.22

ports:

- 5432:5432

environment:

POSTGRES\_USER: username

POSTGRES\_PASSWORD: postgresql\_password

POSTGRES\_DB: database\_name

PGDATA: /var/lib/postgresql/data

volumes:

- pgdata:/var/lib/postgresql/data

Поехали

Ниже привожу только команды. Выполнять их нужно из директории, где лежит yml файл.

Сборка

docker-compose build

Запуск

docker-compose up -d

Остановка

docker

Простий рестарт

docker-compose restart

Рестарт з перезбиранням

**docker-compose up --build**

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

Системні роботи:

lusrmgr.msc – локальні групи та користувачі

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

**Підключення до працюючого контейнера:**

docker ps

docker exec -it «контейнер» /bin/bash

----

**Повернення стилів в АДМІНПАНЕЛЬ**

Что делает команда python manage.py collectstatic

Команда **python manage.py collectstatic** виконує збір статичних файлів усіх додатків, що використовуються в проекті Django, та зберігає їх у папку, вказану в змінній **STATIC\_ROOT** в файлі налаштувань Django. Це допомагає підготувати проект для публікації на продакшен-сервері, де статичні файли зазвичай обслуговуються окремо від динамічного контенту.

Крім того, ця команда дозволяє змінити шлях до статичних файлів у проекті. Наприклад, використовуючи ключ **--clear**, вона видалить всі старі статичні файли, які були зібрані раніше, перед тим як зібрати нові. Також можна використовувати ключ **--noinput** для того, щоб запустити збір статичних файлів без інтерактивного запиту підтвердження.

Як виправити? ERRORS: ?: (staticfiles.E002) The STATICFILES\_DIRS setting should not contain the STATIC\_ROOT setting

Помилка означає, що в налаштуваннях Django STATICFILES\_DIRS містить шлях, який також міститься у STATIC\_ROOT. Це недопустимо, тому що коли виконується команда **collectstatic**, Django копіює файли з **STATICFILES\_DIRS** у **STATIC\_ROOT**, і якщо вони містяться в тому ж самому місці, то це може призвести до перезаписування файлів або їх видалення.

Щоб виправити помилку, переконайтеся, що значення **STATICFILES\_DIRS** не містить шляху, який вже знаходиться в **STATIC\_ROOT**. Наприклад, якщо ви використовуєте папку **static** в кореневій директорії вашого проекту як **STATIC\_ROOT**, то **STATICFILES\_DIRS** повинен містити шлях до інших директорій, які містять статичні файли вашого проекту, але не повинен містити шлях до **static** у вашому кореневому каталозі. Наприклад:

luaCopy code

STATIC\_ROOT = os.path.join(BASE\_DIR, 'static') STATICFILES\_DIRS = [ os.path.join(BASE\_DIR, 'other\_static'), ]

Це дозволить Django знайти статичні файли у папці **other\_static**, яку можна редагувати без переживань, що файли будуть випадково перезаписані або видалені при виконанні **collectstatic**.

Мої файли проекту nem-portal:

docker-compose.yml

version: '2'  
*#створюємо мережу для контейнерів с заданими ip-адресами*networks:  
 network\_local:  
 driver: bridge  
 ipam:  
 driver: default  
 config:  
 - subnet: 192.168.96.0/24  
  
services:  
 nginx:  
 *# при падении будет стараться подняться* restart: always  
 *# только свежий nginx* image: nginx:latest  
 networks:  
 network\_local:  
 ipv4\_address: 192.168.96.2  
 *# слушает порт 8080* expose:  
 - 8080  
 *# мапаем порт 80 на его 8080. Тогда сайт будет доступен по адресу http://nem-portal. Убедись, что порт у тебя не занят.* ports:  
 - 80:8080  
 *# монтируем только те директории, в которых лежит статика, т.к. nginx с динамикой не будет работать.   
 # Также директорию с логами и файл настройки, который мы подготовили.* volumes:  
 - ./NEM\_DJANGO/static\_global:/srv/www/nem\_portal/static  
 - ./NEM\_DJANGO/media:/srv/www/nem\_portal/media  
 - ./NEM\_DJANGO/logs:/srv/www/nem\_portal/logs  
 - ./docker/nginx:/etc/nginx/conf.d  
  
 *# и nginx зависит от контейнера python. Т.е. python должен быть запущен первым* depends\_on:  
 - python  
  
 python:  
 restart: always  
 *# указываем откуда собирать образ* build:  
 context: .  
 dockerfile: docker/python/Dockerfile  
 *# монтируем директорию проекта* volumes:  
 - ./NEM\_DJANGO:/srv/www/nem\_portal  
 networks:  
 network\_local:  
 ipv4\_address: 192.168.96.3  
 expose:  
 - 9000  
 ports:  
 - 9000:9000   
 *# запускаем gunicorn* command: "gunicorn -c gunicorn.py nem\_portal.wsgi"

Dockerfile для контейнера Python

# pull official base image  
FROM python:3.10  
  
# copy project  
COPY ./NEM\_DJANGO /srv/www/nem\_portal  
  
# set work directory  
WORKDIR /srv/www/nem\_portal  
  
# інсталюємо ODBC драйіери для MS SQL  
RUN apt-get update && \  
 apt-get install -y gnupg2 && \  
 curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add - && \  
 curl https://packages.microsoft.com/config/ubuntu/20.04/prod.list > /etc/apt/sources.list.d/mssql-release.list && \  
 apt-get update && \  
 ACCEPT\_EULA=Y apt-get install -y msodbcsql17  
  
# інсталюємо утіліти для роботи з мережею: ip addr show або ip -br a show - перегляд адрес  
RUN apt-get update && apt-get install -y iproute2

# українська локалізація

RUN apt-get install -y locales

RUN sed -i -e 's/# uk\_UA.UTF-8 UTF-8/uk\_UA.UTF-8 UTF-8/' /etc/locale.gen && \

locale-gen

ENV LANG uk\_UA.UTF-8

ENV LANGUAGE uk\_UA:uk

ENV LC\_ALL uk\_UA.UTF-8

# install dependencies  
RUN pip install --upgrade pip  
RUN pip install -r requirements.txt  
  
# set environment variables  
ENV *PYTHONDONTWRITEBYTECODE* 1  
ENV *PYTHONUNBUFFERED* 1  
ENV *DJANGO\_SECRET\_KEY* d3(2)o\_r)1hunf4g35h!a\_\*m4p\_\*t9\*$$*b*%(@f\_r$*ndvj*)x^v3  
ENV *DJANGO\_DEBUG* " "  
ENV *DJANGO\_PARB* "Nem-portal-2023"

nem\_portal.conf

*# ------- налаштування nginx як proxy ---------*server {  
 listen 8080; # nginx будет слушать этот порт.  
 server\_name proxy\_python;  
 charset utf8;  
 autoindex off;  
 access\_log /srv/www/nem\_portal/logs/nem\_portal\_access.log;  
 error\_log /srv/www/nem\_portal/logs/nem\_portal\_error.log error;  
   
 *# обробка статичних файлів* location /static/ {  
 root /srv/www/nem\_portal/;  
 }  
 location /media/ {  
 root /srv/www/nem\_portal/;  
 }  
   
 *# налаштування proxy на gunicorn* location / {  
 proxy\_pass http://192.168.96.3:9000;  
 proxy\_set\_header X-Forwarded-Host $host;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 proxy\_set\_header Origin $http\_origin;  
 proxy\_set\_header Host $proxy\_host;

# ---- proxy\_connect\_timeout 120s;

proxy\_send\_timeout 120s;

proxy\_read\_timeout 120s;  
 }  
}  
  
*# proxy\_set\_header X-Forwarded-Port $server\_port;  
# proxy\_set\_header X-Forwarded-Proto https;  
# proxy\_set\_header Origin $http\_origin;  
# proxy\_set\_header Referer $host;  
# proxy\_set\_header X-Real-IP $remote\_addr;*

gunicorn.py

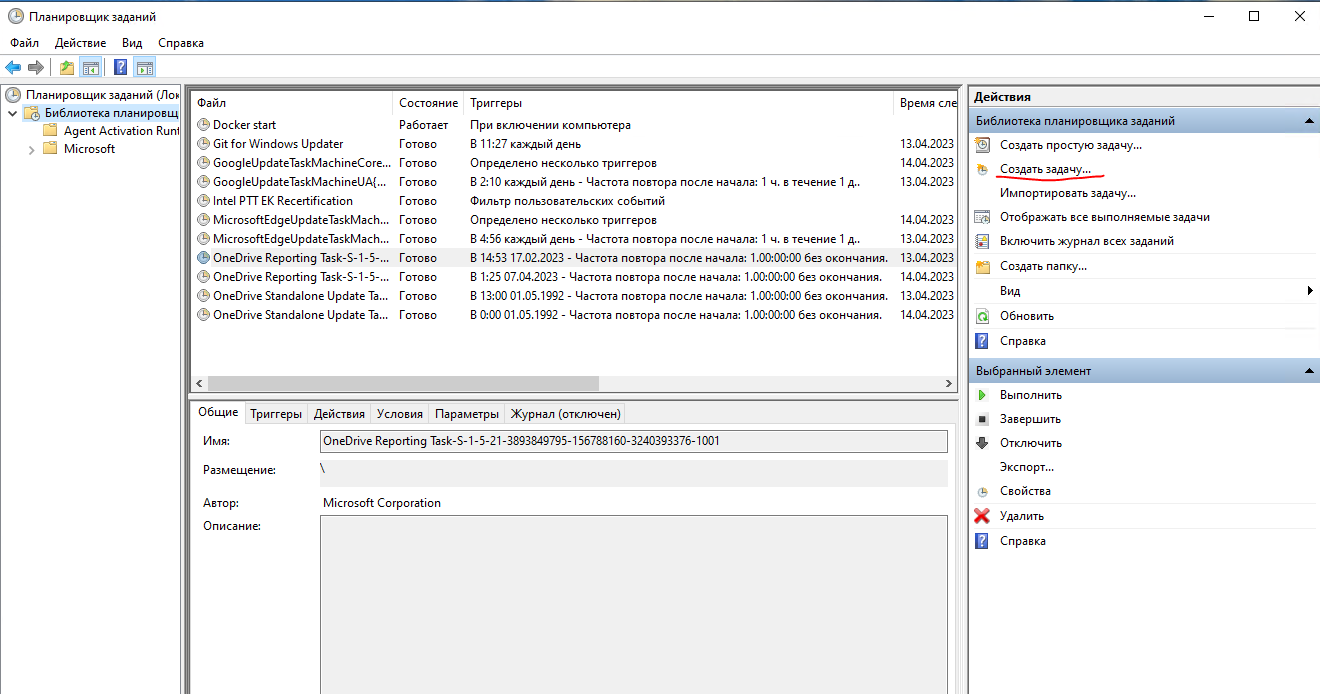
from multiprocessing import cpu\_count  
from os import environ  
  
def max\_workers():  
 return cpu\_count()  
  
bind = '0.0.0.0:' + environ.get('PORT', '9000')  
max\_requests = 1000  
worker\_class = 'gevent'  
workers = max\_workers()  
  
env = {  
 'DJANGO\_SETTINGS\_MODULE': 'nem\_portal.settings'  
}  
reload = True  
name = 'nem\_portal'

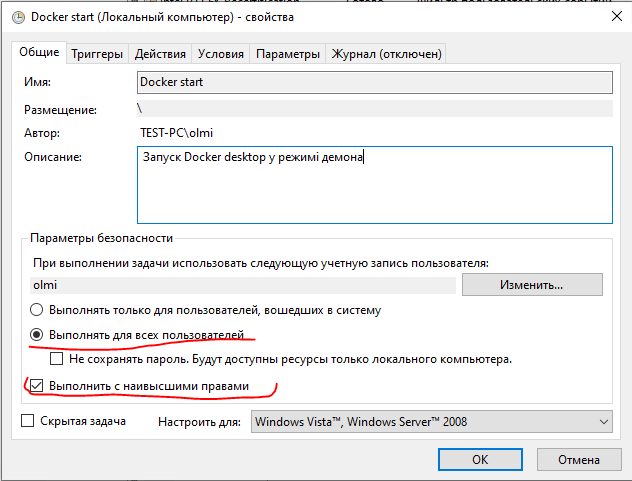
# час очікування відповіді від Питона, важливо! інакше на довгих операціях відвалюється!  
# по дефолту стоїть 30 сек., що замало для Звітів  
timeout = 120

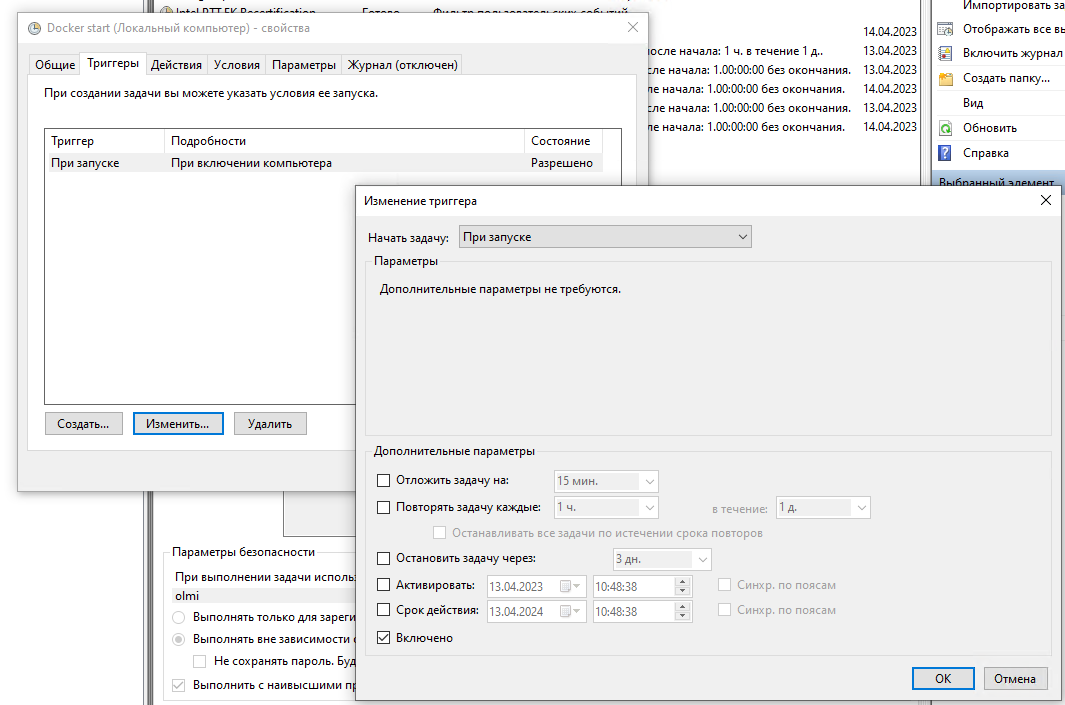
Налаштування запуску DOCKER:

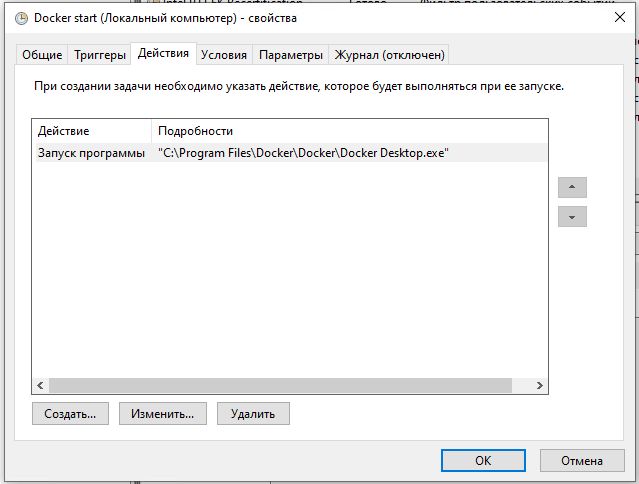
Запустите Планировщик задач Windows (найти в поиске) и выберите *Task Scheduler Library* на левой панели. Затем выберите *Create Task (создать задачу)* на правой панели.

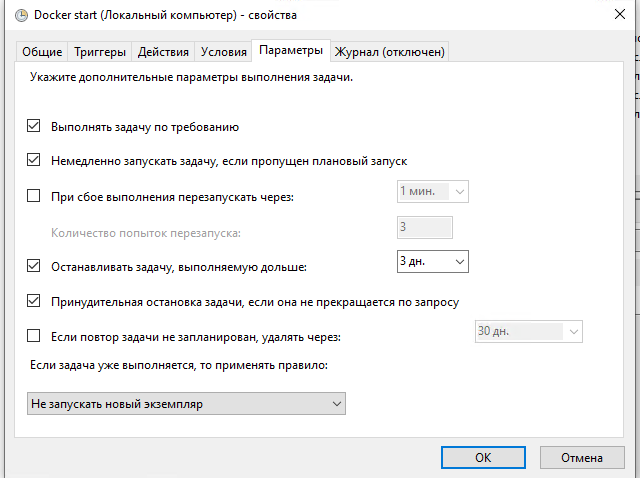
Далее вам предложат отредактировать основные настройки задачи. Дайте вашей задаче понятное название и убедитесь, что остальные настройки соответствуют моим. Важно, чтобы вы выбрали *«Запуск с повышенными привилегиями» (Run with highest privileges)*.











**Створення серверу GIT для проекту**

Заходимо в теку з проектом, запускаємо командний рядок:

git clone --bare . nem\_django.git

Отримуємо теку nem\_django.git

Розміщуємо її на файл-сервері, надаємо права на перегляд та запис для розробників.

**Синхронізація проекту з сервером**

Заходимо в теку, де буде проект, запускаємо командний рядок:

git remote add origin [\\nem-portal\Python\Git\_server\nem\_django.git](file:///\\nem-portal\Python\Git_server\nem_django.git)

git remote show origin

Для першої передачі змін

git push --set-upstream origin master

**Створення клону проекту на ПК розробника**

Заходимо в теку, де буде проект, запускаємо командний рядок:

git clone \\nem-portal\Python\Git\_server\nem\_django.git

Створюємо проект в PyCharm на основі клонованого проекту.

Після установки Pipenv треба додати шлях до нього в PATH:

Запускаємо вікно параметрів:

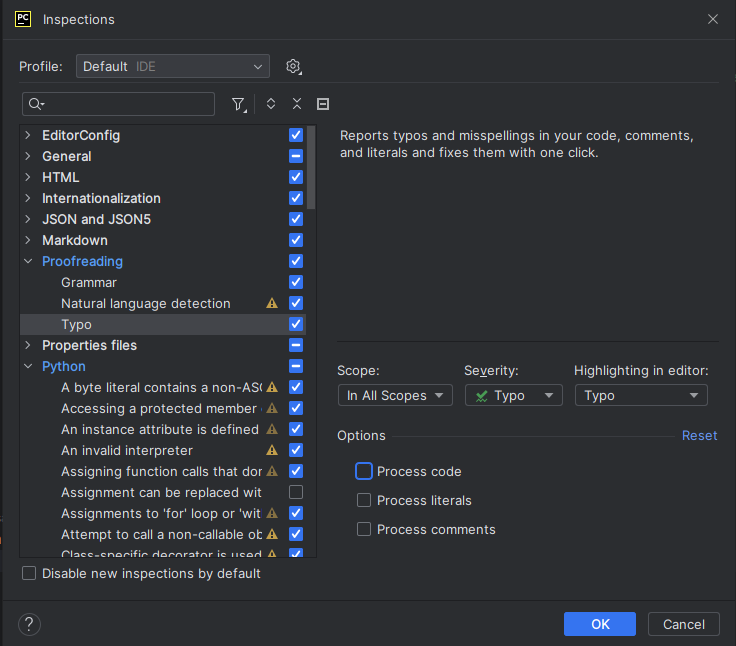
rundll32 sysdm.cpl,EditEnvironmentVariables

Додаємо шлях:

c:\Users\Mihaylik\_OY\AppData\Roaming\Python\Python311\Scripts\

**! PyCharm**

Прибираємо перевірку про друкарські та орфографічні помилки у вашому коді, коментарях і літералах



**! Уважно дивимся на залежності пакетів та версії, які підтримуються**

mssql-django=1.2 працює з django=4.0.10. Більш нові версії викликають проблеми.

Дивитися сумісність тут - <https://pypi.org/project/mssql-django/>