**ASO – partea a 3 a**

Cerințele de rezolvat

In aceasta etapa presupunem ca site-ul nostru este funct, ional si il pregatim de lansare ın

piata. Pentru un deploy cat mai portabil vom utiliza containere Docker.

• (1p) Instalații Docker si creați un Dockerfile pentru aplicația voastra .

• (1p) In aplicatia Django, modificati setarile astfel incat baza de date sa nu mai fie un fisier

SQLite, ci o baza de date reala, Postgres sau MySQL. Serverul bazei de date va rula si el

ıntr-un container Docker separat. Cele doua servicii (site-ul web si baza de date) vor fi

definite ıntr-un fișier docker-compose.yml si vor rula folosind docker-compose.

• (1p) Pana acum, aplicat, ia voastra a rulat ın mod debug, folosind server-ul implicit de la

Django. In continuare, vom folosi un server WSGI real, numit gunicorn si un reverse proxy

(puteti alege nginx sau apache2 ) care fie va redirecta request-urile catre site-ul web, fie va

servi direct fișierele statice. Creati fisiere docker-compose.dev.yml pentru

development/debug (ın care site-ul ruleaza ca si pana acum, cu server-ul de la Django) si

dockercompose.prod.yml pentru product, ie, ın care reverse proxy-ul ruleaza într-un

container separat.

• (1p) Site-ul nostru trebuie sa poata rula s, i ın mod development s, i ın mod production. Din

acest motiv, ın settings.py nu vom mai avea toate setarile hardcodate, ci unele dintre ele vor

fi citite din variabile de mediu. Vom pastra variabilele de mediu separat in fis, iere .env.dev s,

i .env.prod, care vor fi utilizate de docker-compose pentru development, respectiv productie.

• (1p bonus) Site-ul nostru trebuie sa fie accesibil si prin HTTPS. Pentru aceasta etapa, va

trebui sa generati un certificat self-signed (nu puteți obtine unul semnat de o autoritate,

decat daca detineti un domeniu) si sa configurati reverse proxy-ul sa il foloseasca. Puteti

urma acest tutorial.

Modul de rezolvare

Am urmărit tutorialul pentru instalarea si utilizarea Docker. Am generat requirements.txt si

am rulat comenzile de creare a imaginii si a containerului.

De asemenea, pentru setarea unei baze de date in locul fisierului sqlite3 am urmarit pasii de

la sectiunea de PostgreSQL de la acelasi tutorial.

https://testdriven.io/blog/dockerizing-django-with-postgres-gunicorn-and-nginx/

Din moment ce aplicatia a fost realizata pe un sistem Windows a trebuit sa inlocuiesc

gunicorn cu waitress. Pentru setup-ul necesar am urmarit acest tutorial:

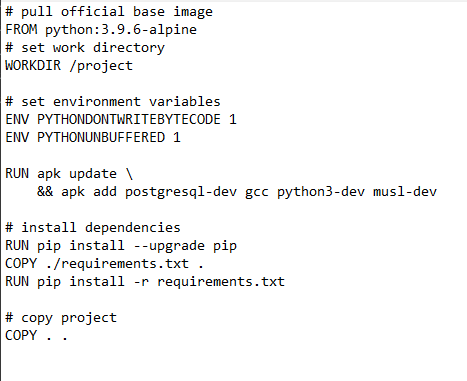
https://www.youtube.com/watch?v=BBKq6H9Rm5g

De asemenea am creat fisiere noi pentru productie (.env.prod & docker-compose.prod.yml).

Diferentele dintre docker-ul de productie si cel de development sunt folosirea librariei

waitress cu nginx pentru a rula server-ul fata de serverul default Django si rularea acestuia

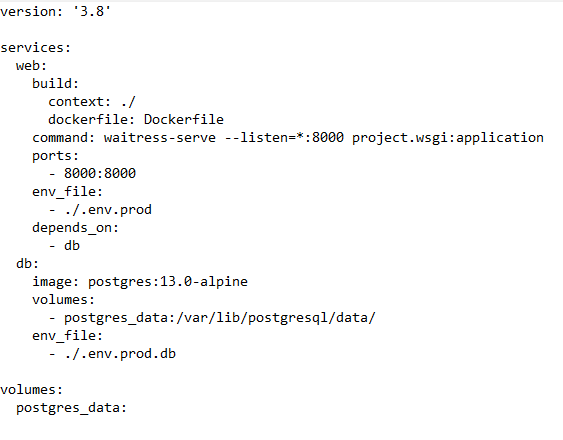
in mod de productie (DEBUG = 0).



Fisierul **Dockerfile**



docker-compose.yml



docker-compose.pro.yml

Concluzii

Urmarind tutorialul si cautand solutii pentru Windows pentru diversele probleme intampinate am invatat cum sa introduc procesele de WSGI si reverse proxy intr-o aplicatie atat Django dar si in alte tipuri de aplicatii. Din moment ce HTTPS a fost un punct bonus si am mai facut procesul de transformare din http in https intr-o aplicatie Django intr-un proiect de anul trecut, nu am mai considerat necesara parcurgerea acestui punct dar am urmarit tutorialul oferit.