# Documentatie proiect DJANGOCHAT

Student: Buzila-Garda Andra-Maria

Profesor indrumator: Andrei Bogdan Leucuta

1. **Descrierea aplicatiei**

Proiectul consta intr-un chat, iar pentru realizarea lui am folosit Python3. Am creat atat parte de register cat si parte de login. Utilizatorul isi poate face cont in aplicatie iar apoi va fi redirectionat pe pagina lui. Aplicatia mai contine o pagina allRooms unde se pot vizualiza toate camerele de chat create. Pagina de chat propriu-zisa contine 2 parti: prima parte, cea in care utilizatorul introduce camera in care vrea sa intre pentru a conversa. Dupa ce a introdus camera (a creat-o sau a intrat pe o camera deja existenta) va fi redirectionat spre pagina de conversatie. Pe aceasta camera pot intra oricat de multi utilizatori care au cont pe site-ul creat. Mai jos se poate observa pagina de start a utilizatorului, dar si pagina de chat.

Graphical user interface, application

Description automatically generated

Fig. 1.a. Pagina de start a utilizatorului

Graphical user interface, application

Description automatically generated

Fig. 1.b. Pagina de chat pentru o camera numita Work

1. **Descrierea implementarii**

Cum am precizat si la punctul 1., limbajul folosit pentru crearea aplicatiei este Python3.

Pentru partea de style am folosit tailwind, in tag-ul “Class” am adaugat diferite culori dimensiuni etc.. Pentru temele 1 si 2 am folosit baza de date db.sqlite3, iar la tema 3 am folosit postgres in loc de db.sqlite3 (am adaptat codul a.i. conexiunea sa se faca la posgres).

In fisierele “urls” din folderele “djangochat”, “room” si “core”, in variabilele urlpatterns (din fiecare fisier) am adaugat rutele catre paginile web create. In folderele “templates” din folderele mentionate anterior se afla fisierele .html in care se afla codul HTML pentru fiecare pagina. In fisierele views am definit fiecare pagina ca si o functie si in respectivele metode am efectuat diferite comenzi (de exemplu redirect care o anumita pagina, sau am creat camera de conversatie).

Pentru chat am folosit ajax pentru a face call-urile de GET si POST ale mesajelor trimise respectiv interceptate. Am folosit setInterval, cu un interval de 1 secunda si in interiorul acestui interval am facut call-ul de GET. Asta inseamna ca in fiecare secunda se va faceun call catre backend pentru a prelua mesajele (atat pe cele noi cat si pe cele vechi). Asta face sa para ca tot chat-ul functioneaza in real-time.

Text

Description automatically generated

Fig. 2.a. Script-ul care contine call-ul de Get catre db

Cum am specificat mai sus, pentru tema 3 am folosit baza de date postgres in loc de sqlite. Setarile necesare pentru a se realiza conexiunea la baza de date au fost facute in fisierul “settings.py” din folderul principal: “djangochat”. Am setat numele bazei de date, numele user-ului, parola si portul de pe care se citesc date. La host, pana sa se realizeze si partea de docker, am pus “localhost”, iar ulterior, am folosit numele container-ului creat.

Tot la tema 3 am facut si partea de deploy a aplicatiei. Am folosit docker. Prima data am creat fisierul Dockerfile unde am specificat niste comenzi care vor fi rulate. Pentru a se rula Dockerfile am folosit comanda “docker build -t <nume\_imagine> .”. In urma rularii acestei comenzi am obtinut o imagine de docker in aplicatia “docker desktop”.

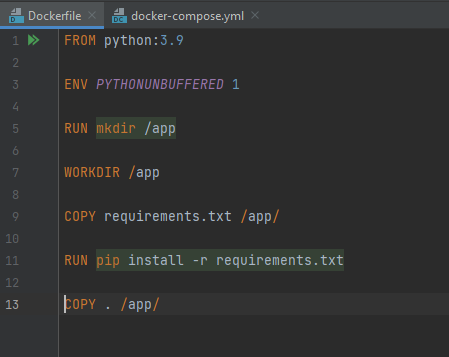


Fig. 2.b. Fisierul Dockerfile

Dupa ce am creat fisierul Dockerfile, am mai creat unul, de care avem nevoie pentru a se realiza deploy, intitulat “docker\_compose.yml”. In acest fisier am introdus datele necesare pentru realizarea container-elor si anume:

* **Pentru db**: am adaugat un nume pentru container, am selectat imaginea, am adaugat, de asemenea, am adaugat variabile de environment si anume: numele bazei de date, numele userului si parola. Tot la db am adaugat si portul bazei de date si un volum.
* **Pentru backend**: am adaugat un nume pentru container, am selectat imaginea, iar, in plus fata de db, am adaugat un string cu comenzi care sa se ruleze in momentul in care se face compose (comenzi pentru crearea migrarilor, aplicarea lor asupra bazei de date si pentru pornirea aplicatiei). Tot in plus fata de db, pentru backend mai avem si un atribut “depends\_on” caruia i-am dat valoarea db. De asemenea, am adaugat portul pentru backend.

Inainte de a rula fisierul am introdus comanda: “docker pull postgres” pentru a extrage

cea mai noua versiune a bazei de date postgres. Pe urma am rulat folosind comanda: “docker-compose up”.

Mai jos se poate observa intreg fisierul “docker-compose.yml”.

Text

Description automatically generated

Fig. 2.c. fisierul “docker-compose.yml”

1. **Concluzie**

In concluzie, acest proiect a fost realizat folosind tehnologii complexe cu toate ca este o aplicatie de chatting usor de folosit de catre orice utilizator. In urma implementarii acestui proiect am invatat sa lucrez foarte usor cu Python, am invatat sa fac migrari si sa creez un chat cu posibilitatea mai multor utilizatori de a intra intr-un canal conversational.