Logo

Description automatically generated

Administrarea Sistemelor de Operare

Jurnal Faza 2

Zelenszky Bianca, 30643

Contents

[1. Cerințele rezolvate 2](#_Toc121305432)

[2. Modul de rezolvare 2](#_Toc121305433)

[3. Probleme întâlnite și modul de rezolvare 3](#_Toc121305434)

[4. Concluzii 3](#_Toc121305435)

# Cerințele rezolvate

Pentru faza 2 a proiectului, am realizat un chat minimalist, în care:

* Utilizatorii se pot loga
* Utilizatorii pot posta mesaje și vizualiza mesajele postate
* Utilizatorii pot posta imagini și vizualiza imaginile postate
* Atunci când un utilizator începe să tasteze, pe chat apare informația că ”User X is typing...”

Un chat cu aceste funcționalități este folositor în situații reale, pentru a oferi posibilitatea de comunicare la distanță în timp real prin mesaje și imagini, asemenea unor aplicații precum Whatsapp și Messenger.

# Modul de rezolvare

Pentru realizarea proiectului, folosind framework-ul Django, am urmărit următorii pași:

* Pentru implementarea funcționalității de chat, am avut nevoie de pachetul *channels*, care ajută la crearea de websocket-uri, precum și de *daphne*, care este un server de protocol websocket pentru ASGI, înlocuind serverul de development de la Django la rularea comenzii *runserver.*
* Am adăugat suport pentru websocket în fișierul *asgi.py*, și la setările proiectului am adăugat constanta ASGI\_APPLICATION.

1. **Utilizatorii se pot loga**

* Am creat un template, *LoginPage.html*, care este pagina de logare a utilizatorului, și am adăugat path-ul ("*auth/login*/") în *urls.py*. Am importat *LoginView* din *django.contrib.auth.views* pentru a crea un formular cu username și parolă pentru ca să se poată loga utilizatorul în pagină.
* Am introdus doi utilizatori în aplicație prin comanda **python manage.py create superuser**

1. **Utilizatorii pot posta mesaje și vizualiza mesajele postate**

* Pentru a posta și vizualiza mesaje, am creat clasa consumers.py care conține metode legate de funcționalitățile de bază ale websocket-ului (*connect*, *disconnect*, *receive*, *sendMesage*).
* Am creat template-ul *ChatPage.html*, care e pagina web a chat-ului, l-am adăugat cu path-ul "" în *urls.py*, și am creat un view pentru acesta pentru a face render la pagină. În interiorul template-ului, am creat un textfield și un buton pentru a trimite mesajele, și am deschis conexiunea cu webserver-ul. Astfel am putut realiza trimiterea și primirea mesajelor și afișarea lor în chat.
* Pentru a persista datele, am folosit baza de date SQLite, și am creat în models.py tabelul pentru mesaje. În *consumer.py*, la primirea mesajului, se salvează în baza de date informații precum conținutul, a cui îi aparține, timestamp-ul și dacă este o imagine sau nu. În *views.py* se trimit în variabila context către template lista cu cele mai recente 10 mesaje din baza de date pentru a fi afișat în chat.

1. **Utilizatorii pot posta imagini și vizualiza imaginile postate**

* Pentru a lucra cu imagini, la setările proiectului am introdus MEDIA\_URL și MEDIA\_ROOT pentru ca salvarea imaginii să aibă loc în folder-ul media în proiect, iar în urls.py am utilizat aceste setări.
* Am creat un form în *forms.py* pentru selecția fișierului și l-am afișat în template, unde am mai adăugat un buton de *Upload* (salvarea imaginii) și *Send Image* (transmiterea propriu-zisă a ei).
* La apăsarea butonului de *Upload*, în *views.py* se scrie în folderul media conținutul imaginii (metoda *handle\_uploaded\_file*). De asemenea, în variabila context se mai ține minte și url-ul fișierului pentru a putea fi utilizat în template.
* La apăsarea butonului *Send Image*, se trimite url-ul imaginii către websocket, care va fi salvat în baza de date pentru a asigura persistența.

1. **Atunci când un utilizator începe să tasteze, pe chat apare informația că ”User X is typing...”**

* Pentru a implementa funcționalitatea ca utilizatorul să vadă dacă celălalt user scrie în acel moment sau nu, de fiecare dată când user-ul scrie în textbox-ul din chat, se va trimite către websocket o variabilă care ține cont dacă utilizatorul a făcut această acțiune. La primirea mesajului din websocket, se verifică dacă variabila e adevărată, și atunci pentru 1.5 ms se afișează textul ”User X is typing...”

# Probleme întâlnite și modul de rezolvare

Cu toate că nu am avut probleme majore în rezolvarea cerințelor, am făcut câteva observații în rezolvarea lor care ar putea fi de folos în viitor:

* Inițial nu am știut cum să realizez salvarea imaginii fără a salva în baza de date. După mai multe tutoriale, am observat că imaginile se pot salva în baza de date cu un form de tip ModelForm, iar în models.py să creez un model cu un field numit ImageField. Această metodă de rezolvare, chiar dacă nu este relevantă pentru cerința curentă, ar putea fi totuși folositoare la un moment dat.
* În *consumers.py*, dacă funcțiile de websocket rămân *async*, nu se pot salva în baza de date mesajele. Astfel, a trebuit să folosesc metoda *async\_to\_sync()* din *asgiref.sync* pentru comunicarea prin socket.

# Concluzii

Prin acest proiect mi-am dezvoltat capabilitățile de a scrie cod în python3, m-am familiarizat cu framework-ul Django, și am învățat mai multe despre utilizarea websocket-urilor.