Administrarea Sistemelor de Operare
- Django project -

Birlutiu Claudiu-Andrei

27/10/2022

1 Jurnal Faza 1

1.1 Cerințele rezolvate

Proiectul dezvoltat în cadrul laboratorului de Administrarea Sistemelor de Operare presupune realizarea unui web-site minimal cu simularea lansării acestuia în producție. Principalul scop al acestui proiect este acela de familiarizare cu tehnologiile folosite și lansarea lui in producție.

Dezvoltarea site-ului se va realiza cu ajutorul framework-ului Django (bazat pe limbajul de programare Python), iar în primă fază cerințele s-au concetrat asupra instalării și preagtirii mediului de dezvoltare a aplicației. De asemenea, la finalul acestei etape s-a dorit crearea unui site minimalist pe baza comenzilor oferite de biblioteca Django instalată pentru Python și crearea interfeței de administrotor (un superuser).

1.2 Modul de rezolvare

În această subsecțiune sunt cuprinși pașii efectuați pentru rezolvarea cerințelor propuse în această fază a proiectului. Aceștia vor fi descriși în ordinea efectuării lor și se va prezenta necesitatea acestora în finalizarea cerințelor cerute

1.2.1 Mediul de dezvoltare a aplicației

Mediul local de dezvoltare a aplicatiei este pe un sistem de operare linux. Sistemul de operare al calculatorului personal este Linux Mint 20.3 Cinnamon, versiunea Cinnamon 5.2.7. Principalele avantaje ale unui sitem de operare linux se referă la faptul că e un program de tip open-source ce are o securitate performantă, e stabil, suportă majoritatea limbajelor de programare și are un management al fisierelor performant, suportând fișiere de aproape orice format.

Versiunea de python pe care o am este 3.9.12. Pentru crearea site-ului minimalist a fost nevoie de instalarea bibliotecii Django. S-a instalat un official release pentru Django cu ajutorul package installer-ului **pip**. Aceasta este printre principalele metode recomandate pe site-ul oficial al framework-ului

```
$ python -m pip install Django
```

Django este un framework folosit pentru dezvoltarea site-urilor web ce urmează un model arhitectural de tipul Model-View-Controller. La crearea unui proiect Django se creează principala structură a site-ului cu separarea functionalităților pe pachete specializate. Un alt avantaj al framework-ului este acela că oferă un panou administrativ, astfel un user cu rol de superuser in aplicație poate accesa acest panou de administrare a aplicatiei și poate crea, citi sau actualiza informațiile cuprinse în baza de date. O cerință din cadrul acestei faze a proiectului se refera la activarea acestui panou administrativ și crearea unui super user cu un username și parolă care poată să acceseze ulterior pagina de admin. Versiunea de Django actuală este 4.1.2 .

1.2.2 Crearea structurii initiale a site-ului

Pentru crearea structurii initiale a site-ului s-a rulat următoarea comandă. În urma rulării comenzii s-a creat un fișier *manage.py* și un folder în care se regăsesc principalele fișiere de configurare ale aplicației web.

```
$ django-admin startproject claudiu_first_site
```

Fișierele create conțin informații despre configurarile și setăriile proiectului in ceea ce privește aplicațiile pe care le conține și conexiunea la baza de date (**settings.py**). De asemenea, cuprinsul cu toate rutele url ale aplicației sunt incluse intr-un fișier denumit **urls.py**. De asemenea, se mai creează un fișier important (*manage.py*) ce reprezintă un utilitar pentru linia de comanda la pornirea server-ului local al aplicației.

Pornirea server-ului local de test al site-ului se realizează prin rularea comenzii de mai jos si se poate accesa la portul default 8000. Acest port se poate de asemenea modifica din fisierul de settings.

```
$ python manage.py runserver
```

1.2.3 Crearea unei applicatii pentru proiect

Am creat prima aplicatie din cadrul proiectului pe care am denumit-o conform tutorialului *aso*. Pentru această aplicație s-a creat un view de *"Hello world"* care e accesibil la URL-ul *localhost:8000/aso/*. Am creat in interiorul folder-ului specific aplicației nou create un fisier denumit *urls.py*, unde am definit pentru un urlpattern ce functie din cadrul view-urilor definite să se apeleze în momentul accesării lui. Asftel, pentru un context " în cadrul aplicatiei **aso** se va apela view-ul de Hello World (vezi imagine 1).

```
-crearea aplicatiei aso---
$ python manage.py startapp aso
```

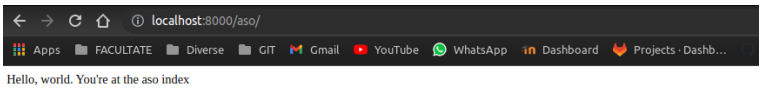

Figure 1: Hello world aso app

1.2.4 Crearea unui super user

Am creat prin intermediul comenzii de mai jos un super user și am accesat panoul administrativ prin intermediul logarii cu acest user.

```
$ python manage.py createsuperuser
```

1.3 Probleme întâlnite și modul de rezolvare

Nu am întâmpinat probleme, iar procesul de creare a minisite-ului a fost ușor de urmărit.

1.4 Concluzii

Ceea ce apreciez la un proiect creat prin intermediul framework-ului Django este faptul că îți impune de la început să îți structurezi proiectul după modelul MVC. Respectarea acestui template va oferi un grad ridicat de reutilizare a codului.

1.5 References

https://docs.djangoproject.com/en/3.2/intro/tutorial01/

2 Jurnal Faza 2

2.1 Cerințele rezolvate

În cea de a doua pate a proiectului s-a cerut implementarea unei aplicatii de chat in care utilizatorii pot sa creeze **room-uri** și să trimită mesaje text simple sau să atașeze imagini. În cazul în care un utilizator incepe să scrie, ceilalti participanti din room vor vedea in timp real ca utilizatorul respectiv scrie. De asemenea, un utilizator care este inscris intr-un room, dar nu e online, va primi mail-uri cu mesajele ce se vor scrie in room in absenta lui.

Aplicatia va randa mesajele în timp real, fără a fi nevoie ca utilizatorul sa dea refresh paginii de chat. Un utilizator poate crea un room și apoi să invite alți participanți care pot sa accepte sau sa respinga cererea.

2.2 Modul de rezolvare

Pentru implementarea logicii de real time s-a recurs la folosirea *websocket-urilor*. Astfel, s-a setat tipul ASGI pentru standardul Python de aplicatii si webservere asincrone. In acest caz, pentru fiecare room s-a creat un topic și s-a atasat o metoda pe partea de server cu rol de chat consumer, responsabila sa primeasca asincron de pe aplicatia client mesajele si sa faca broadcast cu mesajele respective pe toate aplicatiile client conectate la topicul room-ului pe care s-a trimis mesajul.

2.2.1 Modelele aplicației

S-a creat o noua aplicatie in cadrul proiectului django descris in faza 1, denumita **aso-chat**, si pentru care s-au definit mai multe url-uri. În cadrul acestei aplicații s-au definit 3 modele : **Room, ChatMessage și RoomMessage**.

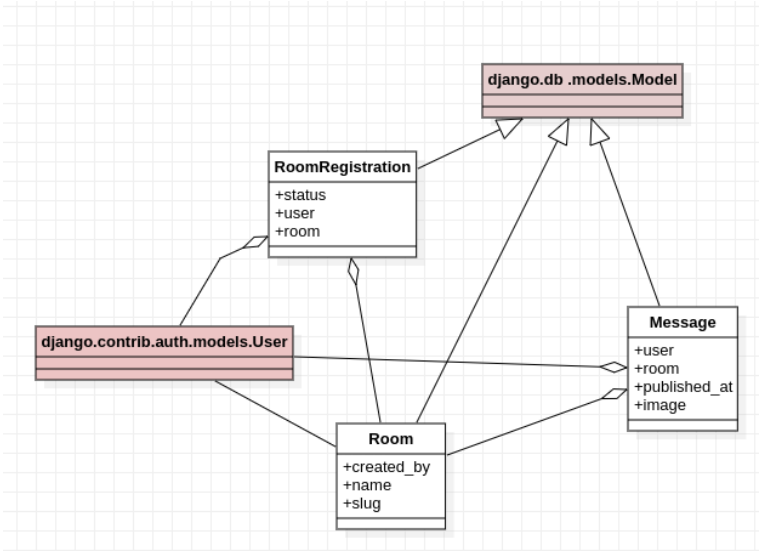

Figure 2: Class diagram

In cadrul unui room sunt mai multe inregistrari de useri care pot fi in starea de pending (userul nu a acceptat inca invitatia) sau in starea de JOINED, userul a acceptat sau este cel care a creat room-ul. Implicit, s-a setat printr-un listener de post_save pe modelul Room, sa se inscrie in tabela de registration userul care a creat room-ul respectiv.

Pentru partea de user nu s-a creat un profil, ci s-a ales sa se foloseasca userul din contrib.auth.models. Partea de login si logout s-a realizat pe baza unor functionalități oferite de pachetele oferite de framework-ul django, iar formulare de login si register au fost realizate cu ajutorul template-urilor. Pentru verificarea

userilor ce sunt online s-a folosit o aplicatie numita **online_users** pe care am instalat-o cu următoarea comanda:

```
---crearea aplicatiei django-online-users---
$ pip install django-online-users
```

2.2.2 Views

S-au creat mai multe view-uri in aplicatia aso-chat care s-a rapsunda request-urilor din partea aplciatiei de client. In cadrul acestor view-uri s-a definit logica de salvare si filtrare a mesajelor, s-a implementat logica de invitatie inntr-un room si partea de accept si approve pentru cererea de intrare intr-un anumit room.

2.2.3 Templates

S-au folosit template-urile django cu formularele aferente, iar pentru partea de aranjare a componentelor UI s-a folosit css si html.

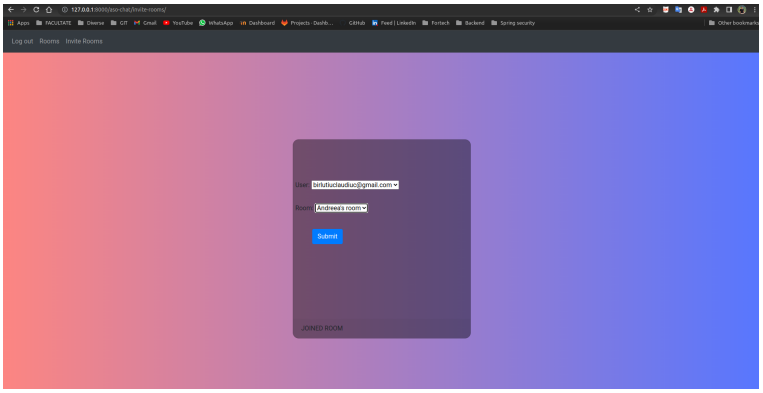

Figure 3: Invite

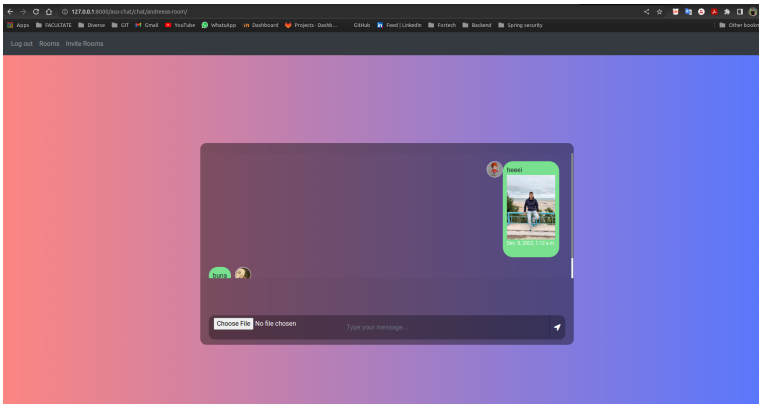

Figure 4: Chat

2.2.4 Email

S-a folosit metota **send_email**:

```
from django.core.mail import send_mail
.....
send_mail( subject, message, email_from, recipient_list )
```

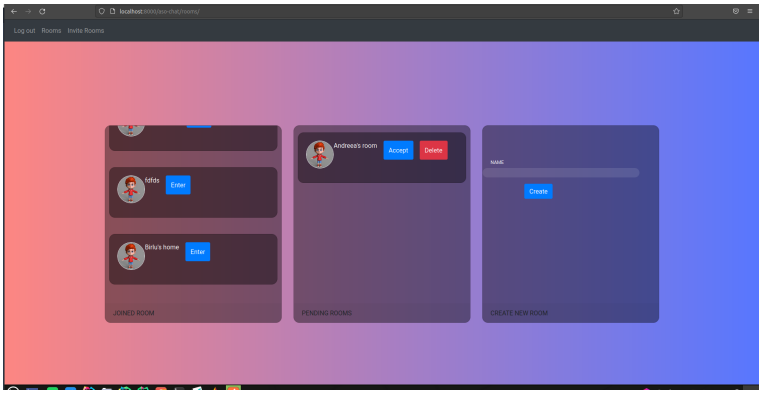

Figure 5: Rooms

2.3 Probleme întâlnite și modul de rezolvare

Am intampinat probleme la integrarea chanbel-urilor pentru websocket din cauza versiunilor instalate, dar s-a rezolvat prin schimbarea versiuninii si a celei de channels.

2.4 Concluzii

Ceea ce apreciez la un proiect creat prin intermediul framework-ului Django este faptul că îți impune de la început să îți structurezi proiectul după modelul MVC. Respectarea acestui template va oferi un grad ridicat de reutilizare a codului.

2.5 References

https://docs.djangoproject.com/en/3.2/intro/tutorial01/
https://www.honeybadger.io/blog/django-channels-websockets-chat/
https://www.youtube.com/watch?v=SF1k$_T$$wr9cg$

3 Jurnal Faza 3

3.1 Cerințele rezolvate

În cea de a treia pate a proiectului s-a cerut in mod principal pregatirea aplicatiei pentru productie și crearea containerelor **docker** corespunzătoare deployment-ului local. Baza de date locală va fi inlocuită cu un postgres SQL, server ce ruleaza de asemenea într-un container docker. Se va crea fișierul docker-compose care include ambele servicii, aplicația și serverul de baze de date. Site-ul va rula fie in mod *dev*, fie in mod *production* utilizând variabile de mediu diferite pentru fișierul **settings.py**.

In modul production, se va folosi un server WSGI real, numit **gunicorns**, și un reverse proxy (**nginx**) care fie va redirecta request-urile către site-ul web, fie va servi direct fișierele statice.

3.2 Modul de rezolvare

Prima modificare pe care am realizat-o a fost trecerea de la un fișier local ce reprezinta baza de date (SQLlite) la conectarea la o baza de date PostgreSQL al cărui server rulează într-un container docker. Trecerea la o baza de date reală s-a făcut cu ajutorul dependeței **psycopg2**. S-a modificat fișierul settings.py, astfel incat sa se conecteze la o baza de date de tipul postgresql, iar variabilele folosite se citesc din fisierele de environment (dev sau prod).

S-a creat un fisier requirements.txt unde au fost definite dependențele necesare rulării aplicației.

```
Django==4.0
channels==3.0.4
Pillow==4.3.0
django-online-users==0.3
psycopg2-binary==2.9.1
gunicorn==20.1.0
```

3.2.1 Development

Steps:

1. S-a creat fișierul **.env.dev** unde s-au definit varibilele de environment pentru fișierul *settings.py*. Modul Debug a fost setat.
2. S-a creat fișierul **Dockerfile** unde s-au definit principalele componente necesare pentru construcția imaginii aplicației.
3. S-a definit fișierul docker-compose.yml unde au fost definite containerele pentru baza de date și aplicația web, definindu-se porturile pe care rulează și volumele necesare.
4. S-au construit containerele prin rularea comenzii de docker-compose.

3.2.2 Production

Steps:

1. S-a creat fișierul **.env.prod** unde s-au definit variabilele de environment pentru fișierul *settings.py*.
2. S-a creat fișierul **Dockerfile.prod** unde s-au definit principalele componente necesare pentru construcția imaginii aplicației pentru modul production.
3. S-a generat certitificatul self-signed și cheia privată pentru securizarea aplicației.

```
$ openssl req -newkey rsa:2048 -nodes -keyout chat.com.key
-x509 -days 365 -out chat.com.crt
```

4. S-a definit fisierul de configurare pentru reverse proxy-ul de tip **nginx**.

5. S-a definit fișierul docker-compose.yml unde au fost definite containerele pentru baza de date, aplicația web și reverse-proxyul, definindu-se porturile pe care rulează și volumele necesare.

6. S-au construit containerele

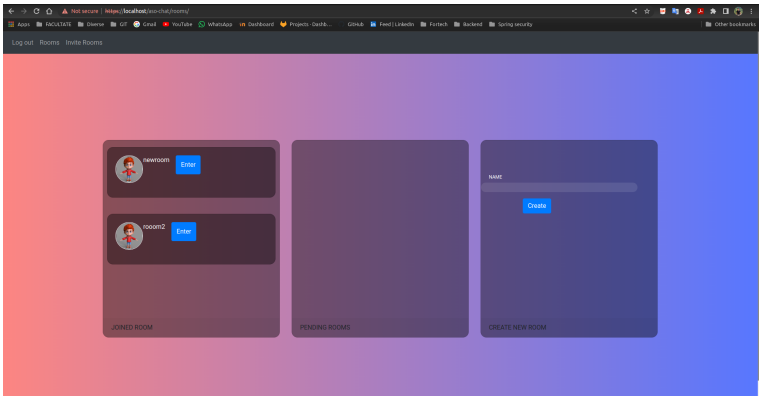

Figure 6: Https self signed certificate

3.3 Probleme întâlnite și modul de rezolvare

Nu am intampinat probleme mari in ceea ce priveste configurarea fisierul de docker deoarece am mai lucrat cu acest tip de containerizare pentru aplicatii de java. Tutorialele oferite ca suport au fost de ajutor.

3.4 Concluzii

Ceea ce apreciez la un proiect creat prin intermediul framework-ului Django este faptul că îți impune de la început să îți structurezi proiectul după modelul MVC. Respectarea acestui template va oferi un grad ridicat de reutilizare a codului.

3.5 References

https://testdriven.io/blog/dockerizing-django-with-postgres-gunicorn-and-nginx/
8https://www.johnmackenzie.co.uk/posts/using-self-signed-ssl-certificates-with-docker-and-nginx/