Aplicatie de manageriere a unui spital

2. Dezvoltarea aplicatiei

2.1. Setarea mediului de lucru

Inainte de a ma apuca de dezvoltarea efectiva a aplicatiei trebuiei sa imi setez un mediu de lucru si sa descarcam si instalam tehnologiile si aplicatiile folosite. Voi incepe cu descarcarea tehnologiilor si aplicatiilor.

Desi pentru a dezvolta aceasta aplicatie web putem folosi orice editor de text, este utila folosirea unui editor de text mai avansat care sa poata fi configurat astfel incat sa se potriveasca cat mai mult preferintelor noastre.

Pentru a descarca si instala editorul de text *Sublime Text Editor* se va accesa versiunea dupa sistemul de operare, in cazul acestei aplicatii *Windows*, accesand acest link catre pagina oficiala a editorului.

Inainte de a descarca si instala platforma de lucru Django trebuie sa ma asigur mai intai ca am limbajul de programare Python instalat. Pentru aceasta pot rula comanda “python –version” intr-o fereastra de command window sau intr-un terminal.

Daca Python este instalat, comanda de mai sus ar trebui sa ne afiseze ce versiune de Python avem instalata ca in figura de mai jos.

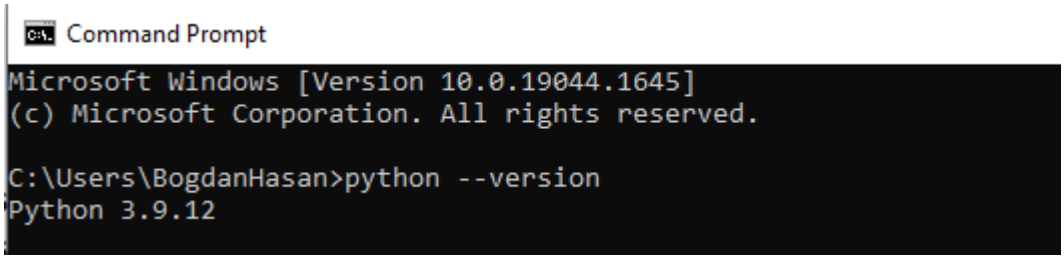

figura 1 Verificare instalare Python

Daca mesajul este unul de eroare atunci Python se poate instala de pe site-ul oficial al acestuia accesand acest link.

Un ultim pas mai este necesar inainte de a putea incepea dezvoltarea propriu-zisa a aplicatiei, si anume descarcarea si instalarea platformei Django. Aceasta se face prin utilizarea administratorului de pachete al Python, si anume pip. Se ruleaza intr-o fereastra de *Command Window* sau *terminal* comanda:

pip install django.

Pentru a verifica ca instalarea a functionat, putem executa comanda: *python -m Django –version* care afiseaza versiunea de Django instalata.

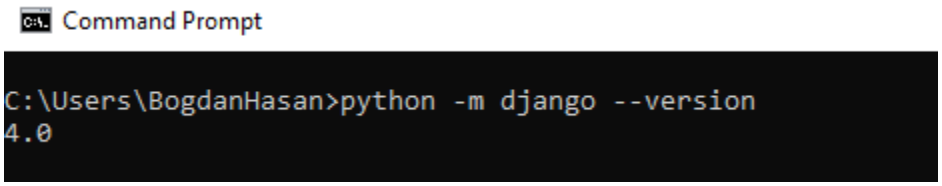

figura 2Verificare instalare versiune Django

2.2. Crearea unui proiect Django nou

Pentru a putea incepe sa dezvolt aplicatia, trebuie sa imi creez un proiect Django nou. Acest proiect ne va permite sa creem aplicatii in interioriul lui. Pentru a crea un proiect nou, intr-un director gol deschid o fereastra de comanda si execut comanda: *django-admin startproject hospital_mngt.* Aceasta va crea un proiect nou cu numele *“hospital_mngt”.*

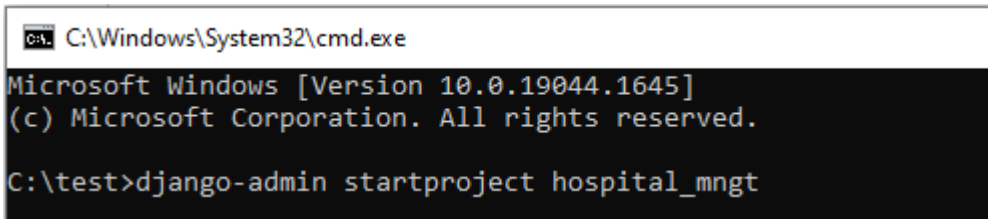

figura 3 Crearea unui nou Proiect Django

Se poate observa ca in directorul in care am proiectul s-a creat un director nou cu numele proeictului meu. Prin *drag & drop* se aduce proiectul nou in Sublime.

In bara din stanga a editorului de text putem acum naviga prin structura proiectului ca in figura de mai jos.

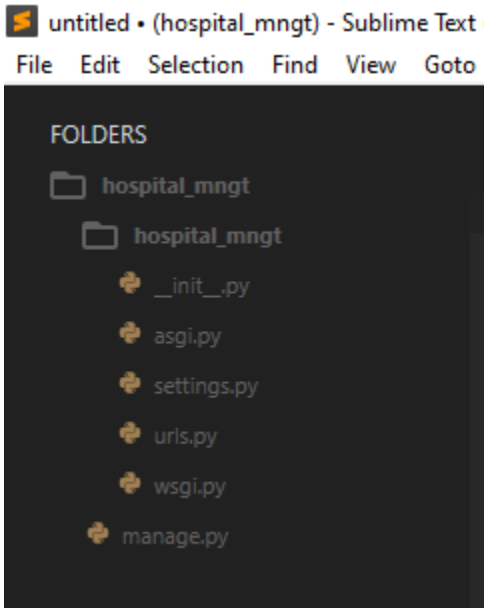

figura 4 Listare structura proiect Django

Django foloseste o structura de directoare pentru a aranja diferite parti ale aplicatiei web. Pentru aceasta, Django creeaza un director nou pentru fiecare proiect sau aplicatie. Atunci cand creem un proiect nou in Django, Django insusi creeaza un director radacina cu numele proiectului dat de noi si care contine fisierele necesare de baza ale functionarii unei aplicatii web.

Structura proiectului Django este explicata in subcapitolele care urmeaza.

2.2.3. Fisierul manage.py

Fisierul *manage.py* contine codul necesar pornirii serverului dar si al unor comenzi pe care le vom folosi pentru a porni serverul, depana, lansa, crea migrari, salvari migrari dar si multe altele.

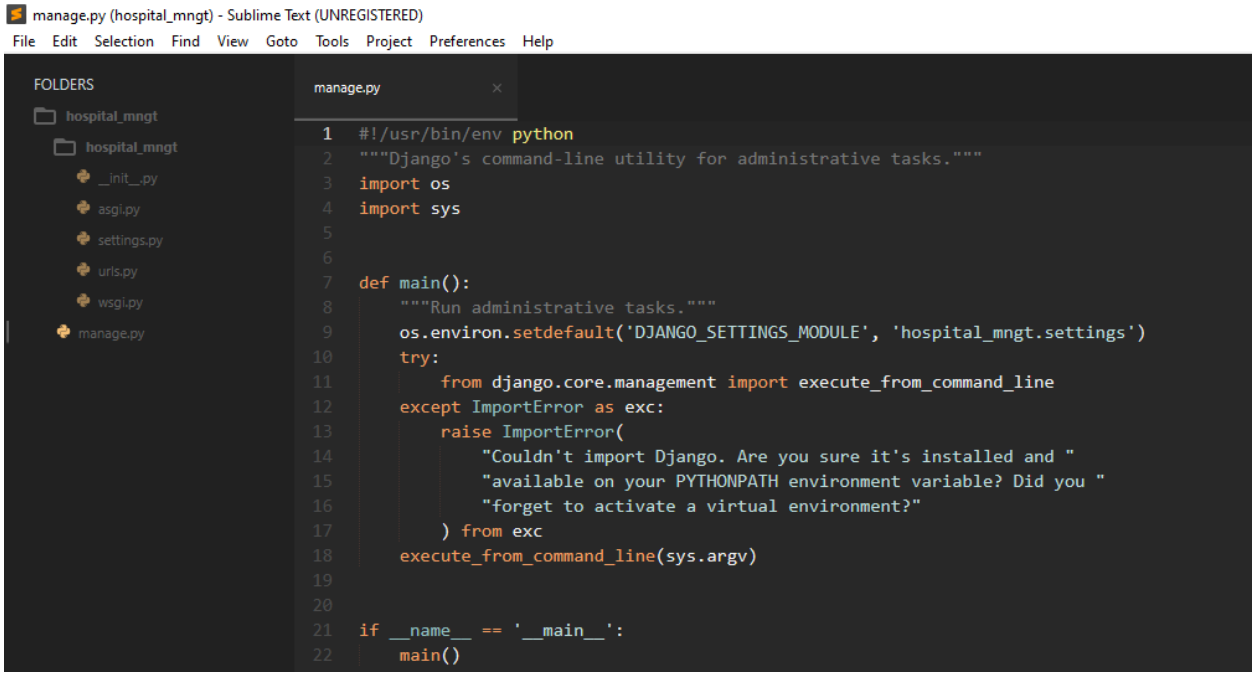

figura 5 Fisierul manage.py

2.2.4. Fisierul __init.py__

Acesta este un fisier gol. Utilitatea acestui fisier este aceea ca interpretorul de Python va stii prin prezenta acestui fisier ca in interiorul acelui director este un pachet si asadar prezenta lui face ca directorul in care se afla sa fie tratat ca un proiect.

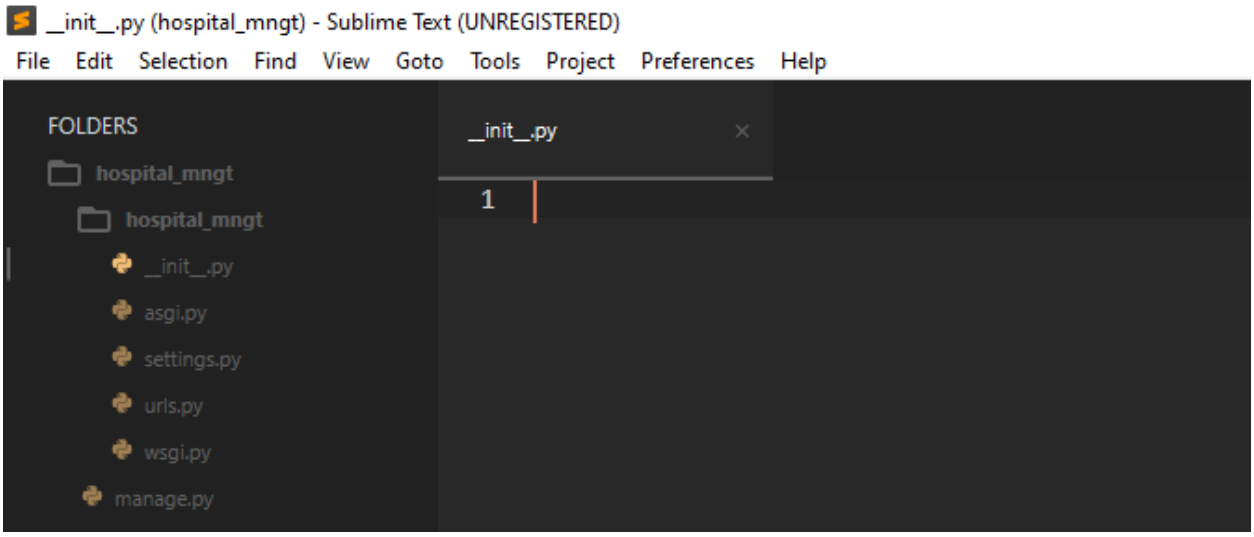

figura 6 Fisierul __init.py__

2.2.5. Fisierul settings.py

Acesta este cel mai important fisier, deorece contine toate configuratiile proiectului si este folosit ca sa lege aplicatiile de proiect.

Acesta contine cateva nume de variabile, *sqlite3* ca baza de date in mod implicit, dar care poate fi schimbata in *Mysql*, *PostgreSQL* sau *MongoDB* in functie de aplicatia web pe care dorim sa o dezvoltam.

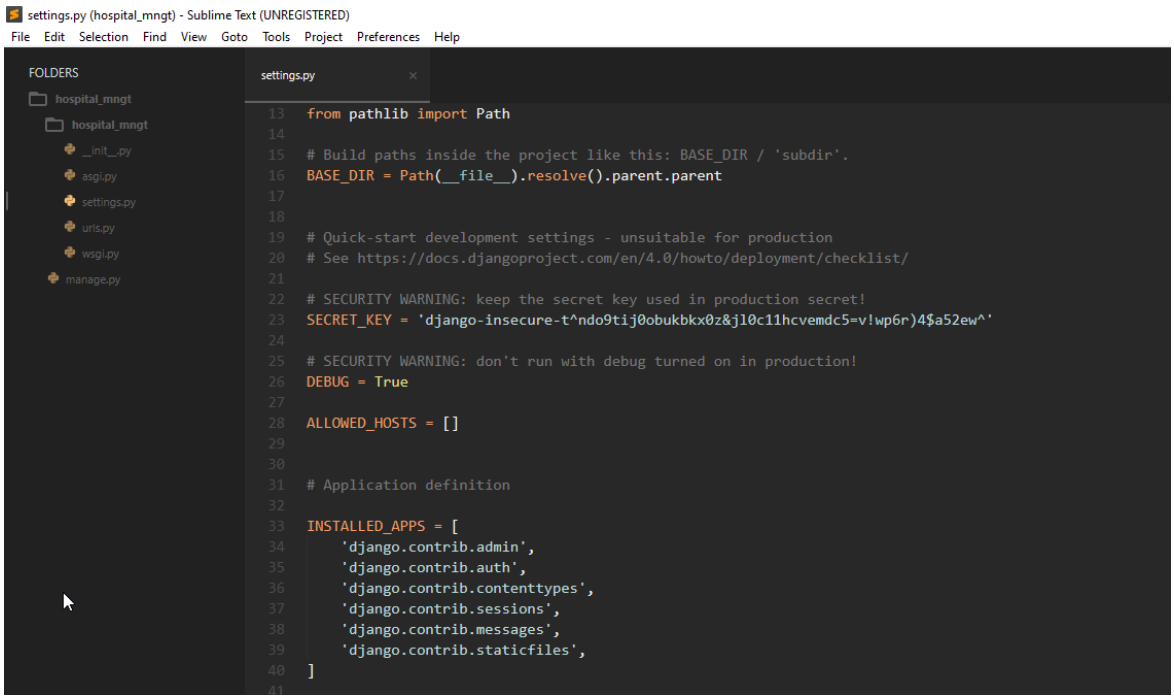

figura 7 Fisierul settings.py

2.2.6. Fisierul urls.py

Acesta contine toate destinatiile folosite in aplicatia noastra web. In cuvinte mai simple, acest fisier este folosit pentru a redirectiona utilizatorii care acceseaza un anumit link catre pagina respectiva.

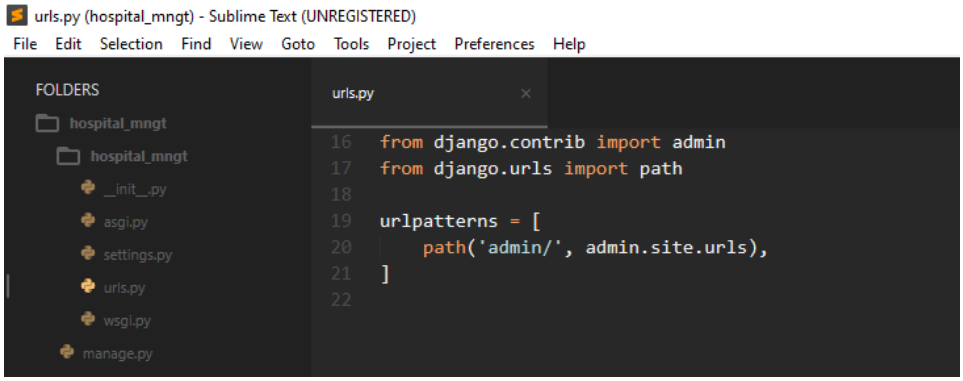

figura 8 Fisierul urls.py

2.2.7. Fisierul wsgi.py

WSGI vine de la Web Server Gateway Interface, si descrie modul in care serverul interactioneaza cu aplicatia. Vom folosi acest fisier in momentul in care terminam de dezvoltat aplicatia si urmeaza sa o hostam. Este un pas simplu, trebuie doar sa importam componentele de mijloc in concordanta cu serverul pe care il vom folosi. Pentru fiecare server, exista o componenta Django de mijloc care rezolva toate problemele de integrare si conectivitate pentru noi.

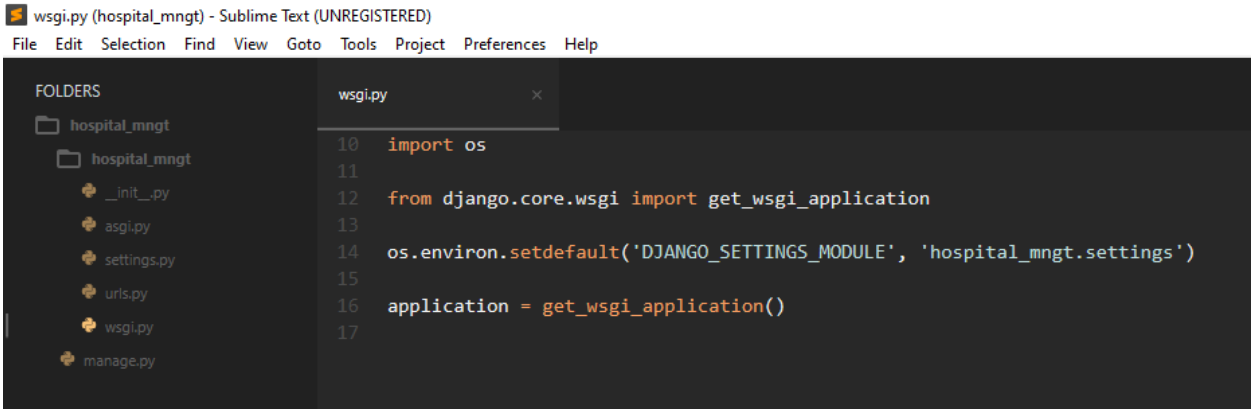

figura 9 Fisierul wsgi.py

2.2.8. Fisierul asgi.py

Fisierul *asgi.py* functioneaza in mod similar cu fisierul *wsgi.py* descris in capitolul anterior dar care vine cu cateva functionalitati in plus. *ASGI* vine de la *Asynchronous Server Gateway Interface* si inlocuieste mai nou fisierul *wsgi.py.*

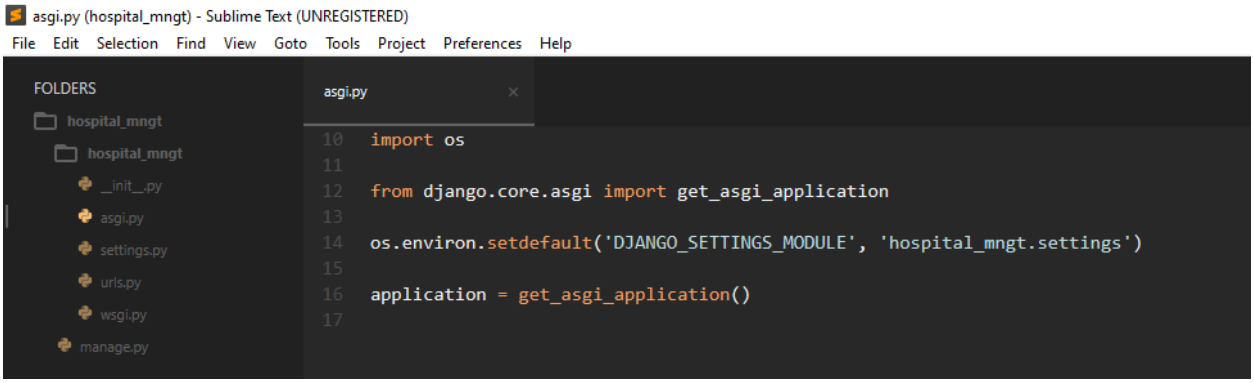

figura 10 Fisierul asgi.py

2.3. Crearea unei aplicatii Django noi

Django foloseste conceptul de proiecte si aplicatii pentru a administra codurile scrise si pentru a le prezenta intr-un mod cat mai curat si organizat. Un proiect Django contine una sau mai multe aplicatii, care executa o munca simultan pentru a asigura un mediu de lucru cat mail in.

Pentru a crea o aplicatie Django, in interiorul directorului unde am creat proiectul, intr-un *command window* sau *terminal* se va executa urmatoarea comanda *python manage.py startapp hospital* care va crea o aplicatie noua cu numele *hospital* ca in figura de mai jos.

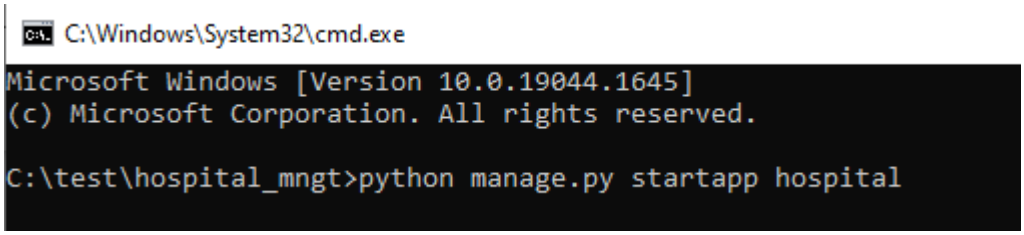

figura 11 Crearea unei aplicatii Django noi

In urma executarii comenzii de mai sus, in directorul proiectului s-a creat un director nou cu numele “*hospital*’. De asemenea se mai poate observa ca in momentul creeri aplicatiei, Django a creat in mod automat cateva fisiere Python

pentru noi. Structura unei aplicatii Django si fisierele create vor fi descrise in subcapitolele urmatoare.

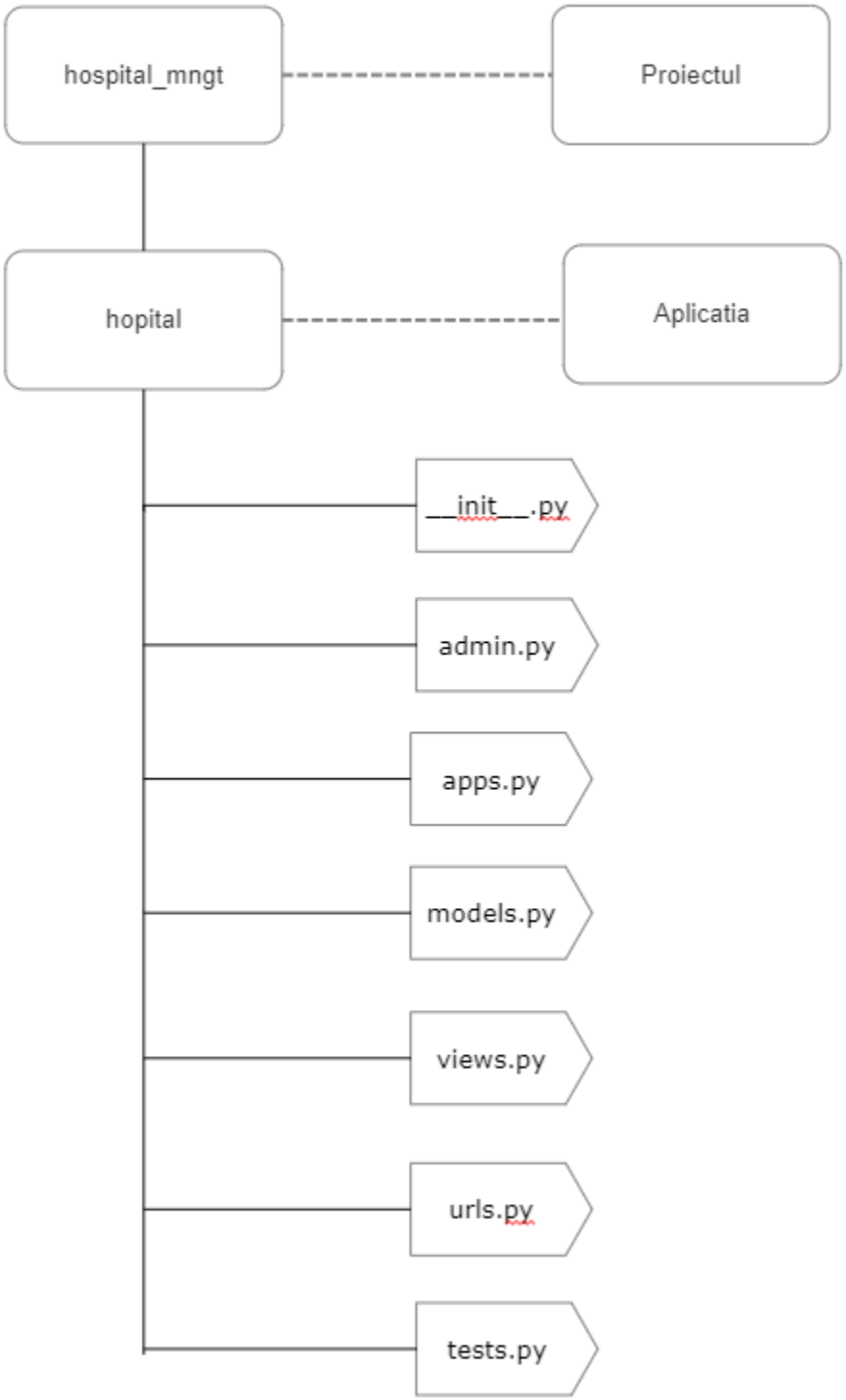

figura 12 Structura unei aplicatii Django

2.3.1 Fisierul hospital.__init__.py

Ca si in cazul fisierului “__init__.py” descris anterior in capitolul 2.2.4 Fisierul __init__.py, si aceste este un fisier gol. Singurul rol al acestui fisier este acela de a-I spune interpretului de Python ca acel director trebuie tratat ca un proiect.

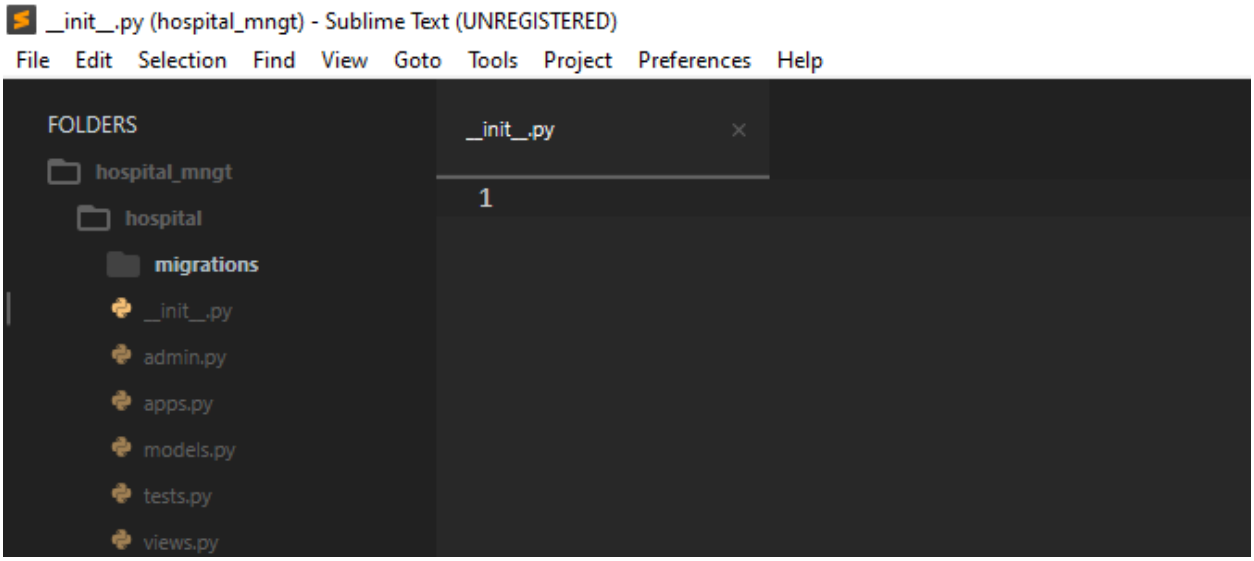

figura 13 Fisierul hospital.__init__.py

2.3.2. Fisierul admin.py

Fisierul *admin.py* este folosit pentru inregistrarea modelelor Django in administrarea aplicatiei. Este folosit pentru afisarea modelelor in panoul de administrare. Acest fisier are trei intrebuintari majore si anume:

- Inregistrarea modelelor;
- Crearea de superusers;
- Autentificarea si utilizarea aplicatiei;

2.3.3. Fisierul apps.py

Apps.py este un fisier folosit pentru a ajuta utilizatorul sa includa configuratiile aplicatiei. Cu toate acestea, includerea configurarilor de catre utilizator nu este un obicei foarte comun deoarece in marea majoritate a cazurilor configuratiile standard sunt suficiente.

2.3.4. Fisierul models.py

Fisierul *models.py* reprezinta modelele de aplicatii web sub forma de clase. Este considerat a fi cel mai important aspect al structurii fisierelor aplicatiei. Modelele defines structura bazei de date si descrie design-ul real, relatiile dintre seturile de date si constrangerile atributelor acestora.

2.3.5. Fisierul views.py

Un alt fisier foarte important atunci cand vorbim despre strctura aplicatiei Django este fisierul *views.py.* Acest fisier ofera o interfata prin care un utilizator interactioneaza cu o aplicatie web Django. Ca si in cazul fisierului *models.py*, *views.py* contine toate view-urile sub forma de clase.

2.3.6. Fisierul urls.py

Urls.py functioneaza in mod asemantor cu fisierul *urls.py* discutat anterior in capitolul *2.2.6. Fisierul urls.py* avand ca scop principal de a lega solicitarile *URL* ale utilizatorului catre paginile corespunzatoare pe care acesta le indica.

2.3.7. Fisierul test.py

Acest fisier permite utilizatorului sa scrie cod de testare pentru aplicatiile web. Este folosit pentru testarea aplicatiei.

3. Dezvoltarea aplicatiei

3.1. Adaugarea url-ului aplicatiei

Odata creata aplicatia, primul lucru pe care trebuie sa il facem este sa modificam fisierul de *urls.py* din interiorul proiectului si sa legam url-urile proiectului cu cele ale aplicatiei. Pentru aceasta, se va deschide fisierul *urls.py* din interiorul proiectului si se va adauga un element nou in lista de *urlpatterns.* Practic, ii vom spune utilitarului Django ca in momentul accesarii url-ului respectiv, utilizatorul sa fie redirectionat catre aplicatia creata de noi. Pentru aceasta, avem nevoie intai sa importam o functie noua numita "*include*".

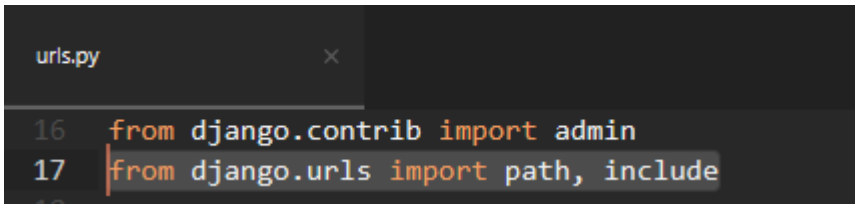

figura 14 includerea functiei "include"

Apoi, in lista de *urlpatterns* vom adauga url-ul pe care dorim sa il redirectionam si calea unde dorim sa redirectionam utilizatorul.

figura 15 Adaugarea unui nou url in proiect

In cazul nostru, cand utilizatorul acceseaza url-ul aplicatiei noastre, acesta va fi redirectionat catre url-urile din interiorul aplcatiei create de noi, pe care le vom implementa ulterior.

3.2. Fisierele statice

Urmatorul pas este sa cream doua noi directoare in interiorul aplicatiei noastre. Acestea sunt directorul *static* care va contine toate fisierele statice precum imaginile pe care le vom afisa in aplicatia noastra web si fisierele ce tin de *css,* adica limbajul pe care il vom folosi pentru a stiliza documentele *HTML,* si inca un director numit *templates* unde vom crea toate fisierele *HTML,* practice, in acest director vom crea paginile pe care dorim sa le afisam in aplicatia noastra web. Directorul *static* este la randul sau compus din alte doua directoare, unul *css* care va contine fisierele de stilizare discutate anterior, si unul *images* care va contine imaginile aplicatiei noastre. La sfarsit noua structura a aplicatiei noastre va arata ca in figura de mai jos.

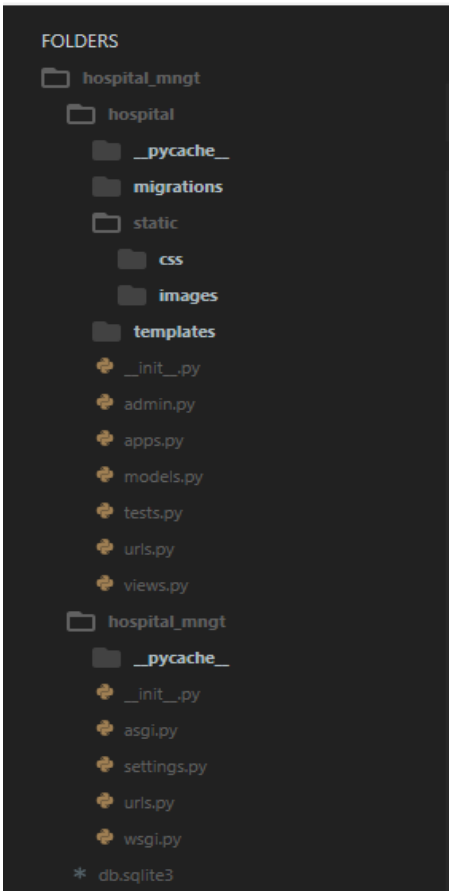

figura 16 Noua structura a proiectului

Totusi, pentru a putea vedea fisierele statice in browser mai este nevoie de cateva setari aditionale. Aceste setari se vor face in fisierul *settings.py* din interiorul proiectului nostru. Prima setare de care avem nevoie este sa adaugam aplicatia noastra in lista de aplicatii instalate ale proiectului. Deoarece in Django un proiect poate avea mai multe aplicatii, trebuie sa ii spunem lui Django ce aplicatii foloseste proiectul nostru. Desi aceasta setare nu are o legatura directa cu fisierele statice, fara ea tot ce vom dezvolta in interiorul aplicatiei noastre nu va fi luat in considerare, inclusive fisierele statice.

Adaugarea unei aplicatii noi in lista de aplicatii instalate se face ca in figura de mai jos, unde in lista de *INSTALLED_APPS,* am adaugat aplicatia *hospital.*

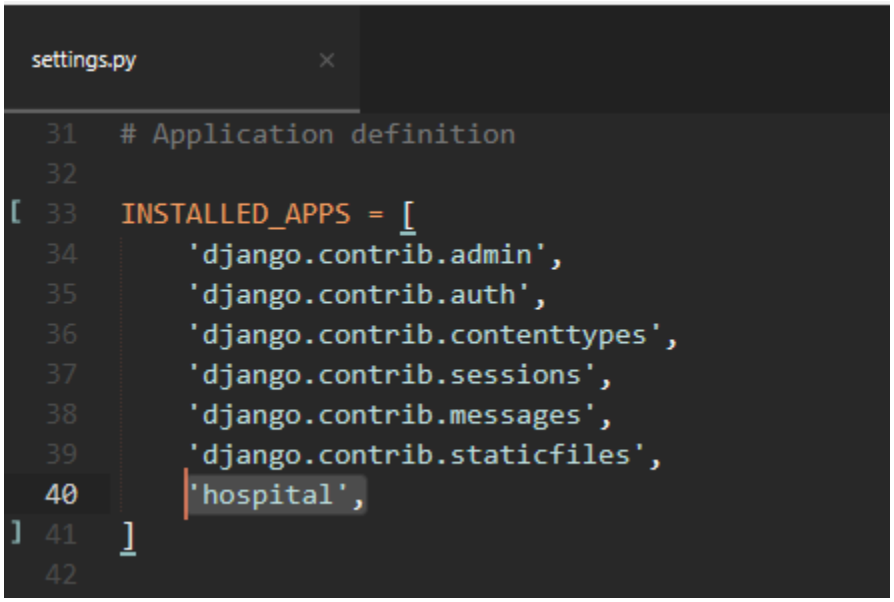

figura 17 Adaugarea unei aplicatii noi in aplicatiile instalate

Urmatoarele doua setari au legatura directa cu fisierele statice, si anume aduagarea campurilor *STATIC_URL* si *STATICFILES_DIRS* pe care le voi prezenta in subcapitolele care urmeaza.

3.2.1. Campul STATIC_URL

Cu ajutorul acestui camp vom defini URL-ul pe care il vom folosi cand vom face referire la fisierele statice situate in *STATIC_ROOT* despre care vom discuta pe scurt in subcapitolul 3.2.3. STATIC_ROOT. In mod implicit, acest camp este setat cu valoarea "*None*", insemnand ca in mod implicit nu avem setat niciun url pentru aceste fisiere. Daca acest camp este diferit de "*None",* valoarea url-ului pe care o vom da trebuie sa se termine cu "*/*".

Setarea acestui camp se face ca in figura de mai jos.

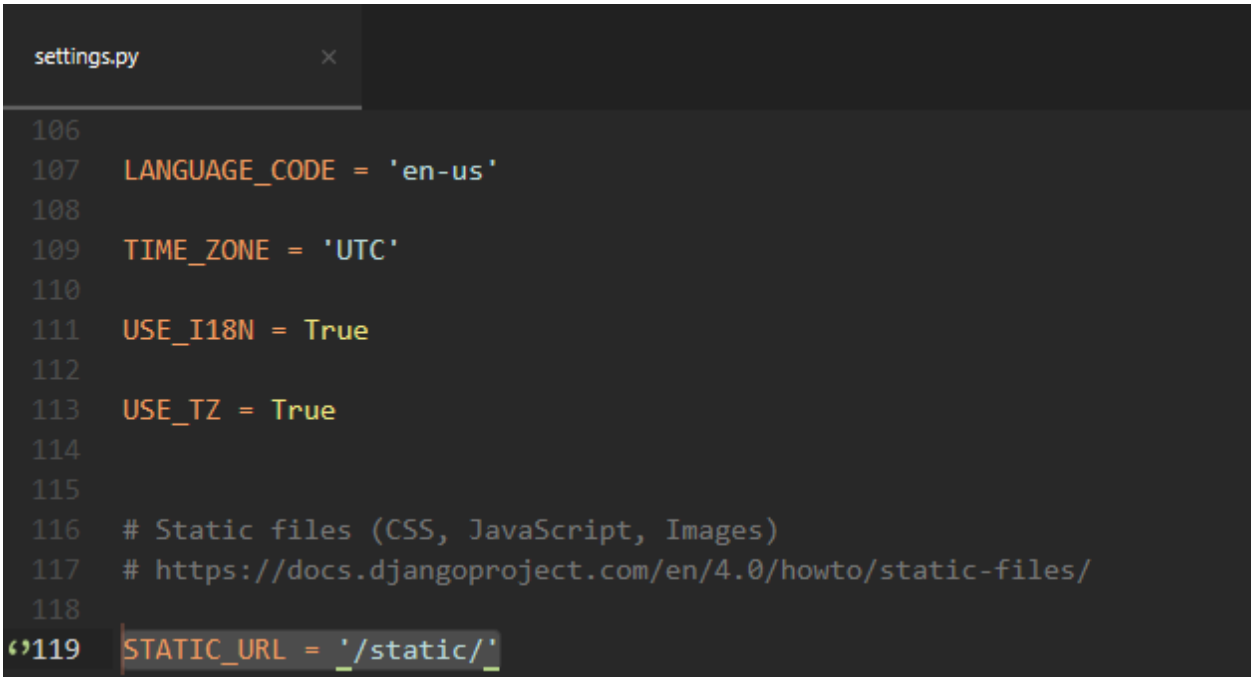

figura 18 Setarea campului STATIC_URL

3.2.2. Campul STATICFILES_DIRS

In mod implicit, acest camp este o lista goala. Aceasta setare ne ajuta sa definim locatia fisierelor noastre statice. Setarea acestui camp se face ca in figura de mai jos.

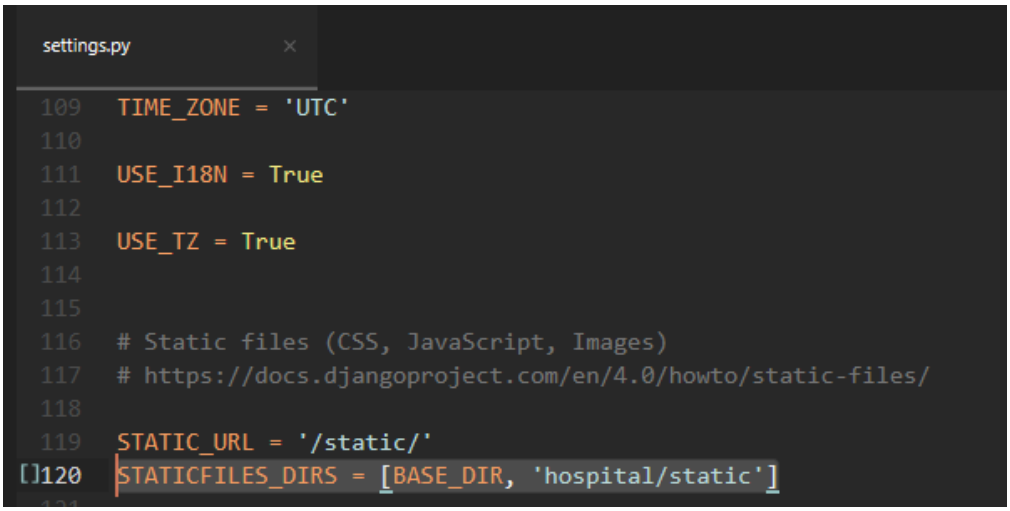

figura 19 Setarea campului STATICFILES_DIRS

Aceasta setare semnaleaza ca fisierele statice ale aplicatiei sunt situate in directorul proiectului nostru Django (*BASE_DIR*) in directorul *static* din directorul aplicatiei noastre *hospital* (*hospital/static*).

3.2.3. STATIC_ROOT

In mod implicit, acest camp este setat cu valoarea "*None".* Acesta reprezinta calea absoluta a directorului de unde *collectstatic* va colecta fisierele statice pentru implementarea aplicatiei. *collectstatic* este o comanda de administrare a aplicatiei care colecteaza fisierele statice in *STATIC_ROOT.*

3.2.4. Directorul static

Asa cum am mentionat si anterior, acest director este la randul sau compus din alte doua directoare "*css"* si "*images".* In directorul "*css"* am creat fisierul de

stilizare “*mystyle.css”.* Acest fisier este un fisier simplu care doar seteaza culoarea header-ului 1 documentului *HTML* cu albastru. Acesta va fi folosit ulterior cand voi discuta despre fisierele *HTML*.

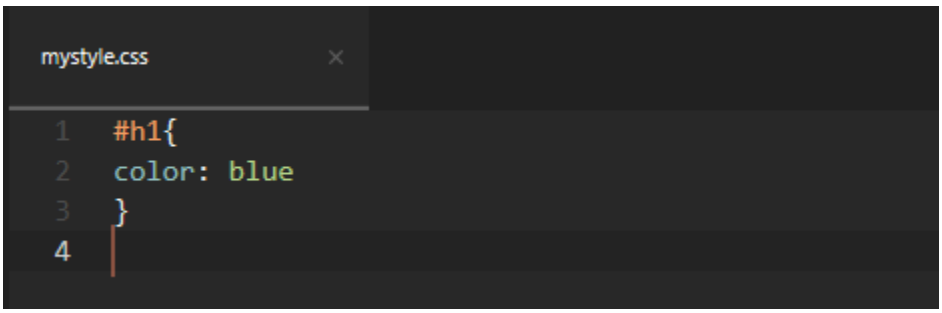

figura 20 Fisierul mystyle.css

Directorul “*images”* contine imaginile pe care dorim sa le afisam in aplicatia noastra web. Aceste imagini sunt gratuite si pot fi folosite in interiorul aplicatiei noastre in mod legal. Sursa acestor imagini se poate gasi in capitolul 5. Referinte.

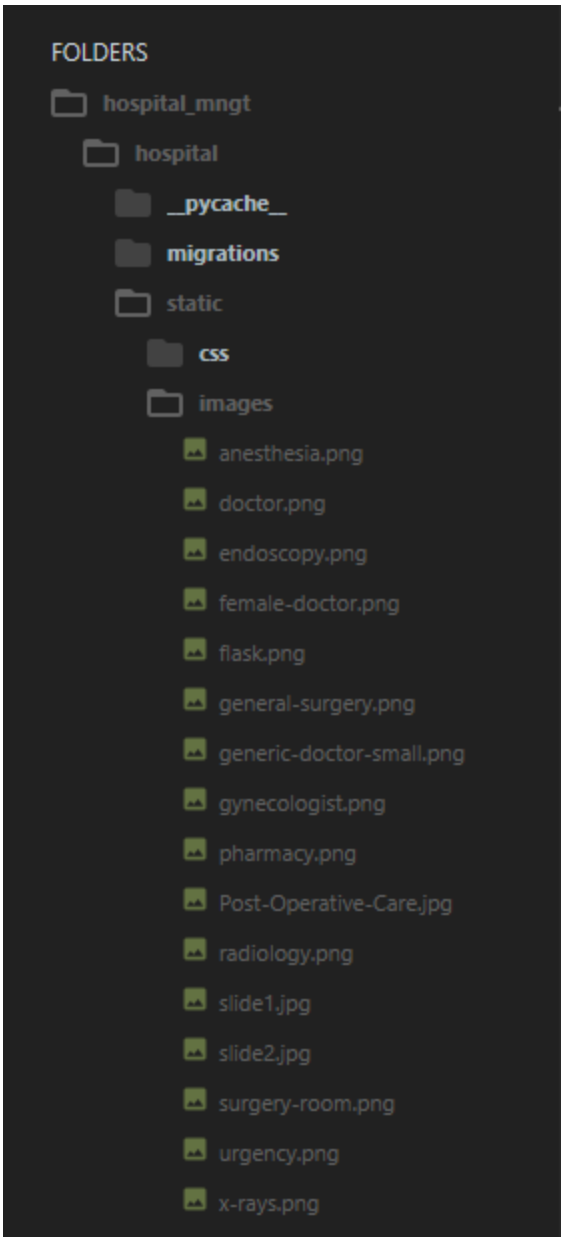

figura 21 Directorul images

3.3. Migrarile si crearea de superusers

Migrarile reprezinta modalitatea Django de a propaga modificarile pe care le facem asupra modelelor noastre (adaugarea unui camp, stergerea unui model s.a.m.d) in schema bazei de date.

Exista mai multe comenzi pentru a interactiona cu migrarile si pentru a gestiona schema bazei de date. Acestea sunt:

- migrate: este responsabil pentru aplicarea si anularea migrarilor;
- makemigrations: este responsabil pentru crearea de noi migrari bazate pe modificarile facute asupra modelelor aplicatiei;
- sqlmigrate: afiseaza instructiunile SQL pentru o migrare;
- showmigrations: listeaza migratiile unui proiect si starea acestora.

Putem considera *migrate* ca pe un sistem de control al versiunilor pentru schema bazei de date iar *makemigrations* este responsabil pentru impachetarea modificarilor modelului nostrum in fisiere de migrare individuale, iar *migrate* este responsabil pentru aplicarea acestora in baza de date.

Django, in mod implicit creeaza pentru noi si o pagina de admin de unde putem administra operatiile asupra modelelor noastre si ne ofera si o interfata grafica pentru aceasta. Cu toate acestea, in cadrul aplicatiei vom crea toate functionalitatile de care dispune aceasta pagina fara a interactiona prea mult cu pagina implicita de admin a Django, insa in cadrul aplicatiei vom lucre cu *superusers,* acei utilizatori care au drepturi de administrator, aceasta fiind o aplicatie de administrare.

Pentru a crea un *superuser* trebuie mai intai sa rulam comanda “*makemigrations”* ca in figura de mai jos.

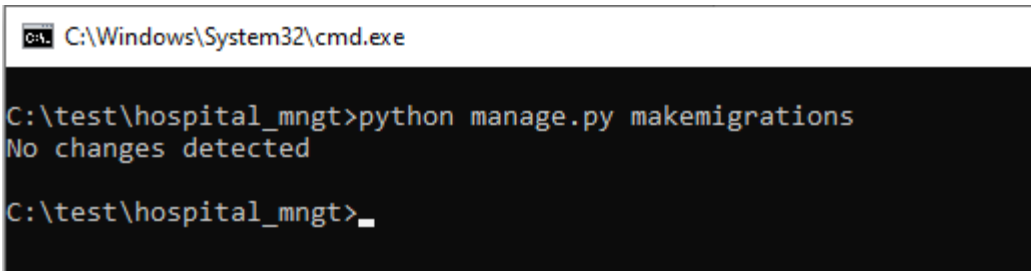

figura 22 Executarea comenzii makemigrations

In urma executarii acestei comenzi observam ca primim raspunsul ca nu au fost detectate schimbari deoarece nu am facut inca modificari in schema de bazei si nici nu am creat inca modelele aplicatiei noastre.

Urmatorul pas este sa executam comanda *"migrate"* care va aplica modificarile noastre in schema bazei de date. Chiar daca nu am facut inca modificari in schema bazei de date, aceasta va aplica anumite setari implicite si va crea si tabela aferenta *superuser-ilor* de care avem nevoie pentru a crea un utilizator cu drepturi de administrator. Comanda se executa ca in figura de mai jos.

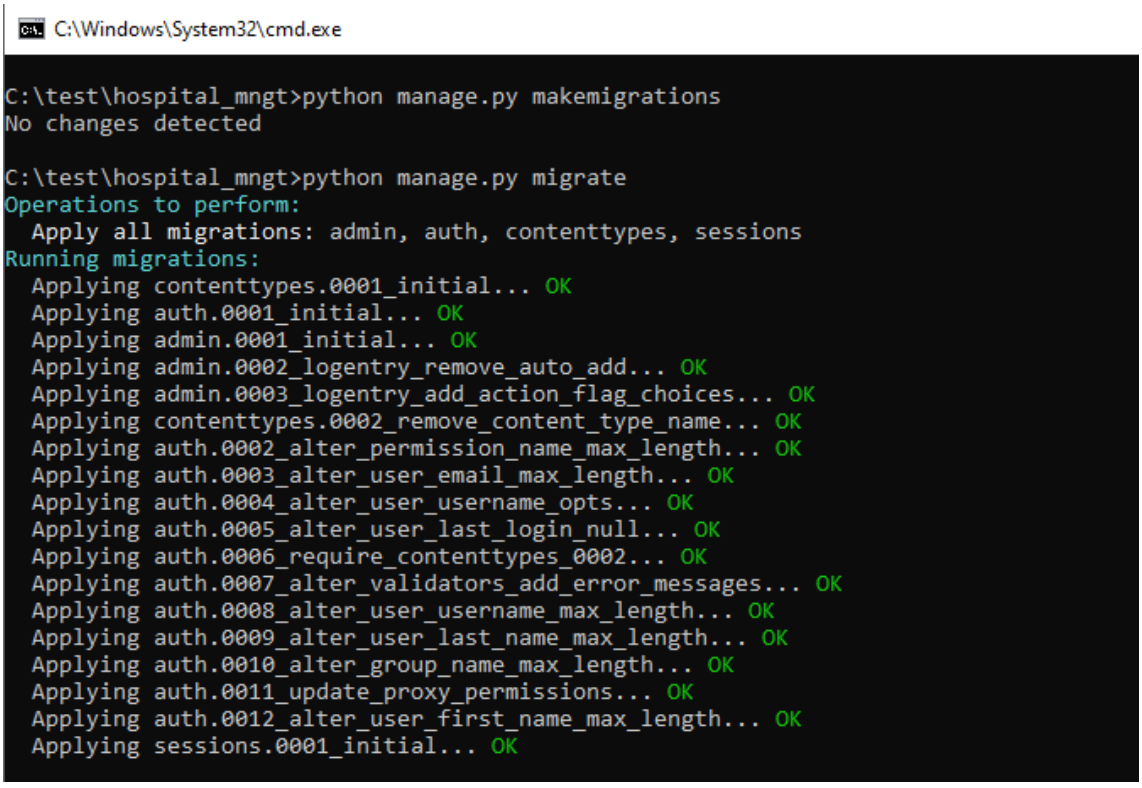

figura 23 Executarea comenzii migrate

Acum putem crea un *superuser* executand comanda “*python manage.py createsuperuser”* ca in figura de mai jos.

```
C:\test\hospital_mngt>python manage.py createsuperuser
Username (leave blank to use 'bogdanhasan'): admin
Email address: admin@yahoo.com
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.

C:\test\hospital_mngt>
```

figura 24 Crearea unui superuser

Pentru aceasta aplicatie am ales ca utilizatorul sa fie *admin* cu parola *admin*. Se poate observa ca Django ne antentioneaza ca parola este prea slaba si similara cu numele utilizatorului, insa putem ocoli aceste avertismente, insa pentru aplicatiile care ruleaza in medii de productie este recomandat sa alegem parole mai greu de ghicit.

3.4. Pagina initiala Django de administrare

Acum ca avem un *superuser* create, putem porni server-ul si naviga catre pagina de administrator. Pentru a porni server-ul se executa comanda ca mai jos.

C:\Windows\System32\cmd.exe - python manage.py runserver

```
C:\test\hospital_mngt>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 03, 2022 - 19:46:35
Django version 4.0, using settings 'hospital_mngt.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

figura 25 Pornirea server-ului

In aceasta fereastra putem observa ca server-ul a pornit la adresa “127.0.0.1:8000” reprezentand adresa la care poate fi gasita aplicatia(127.0.0.1 sau localhost) si port-ul la care asculta aplicatia noastra (8000). De asemenea, in cazul in care apar erori de cod, mesajele de eroare vor fi afisate in aceasta fereastra.

Odata pornit server-ul, putem accesa aplicatia din browser ca in figura de mai jos.

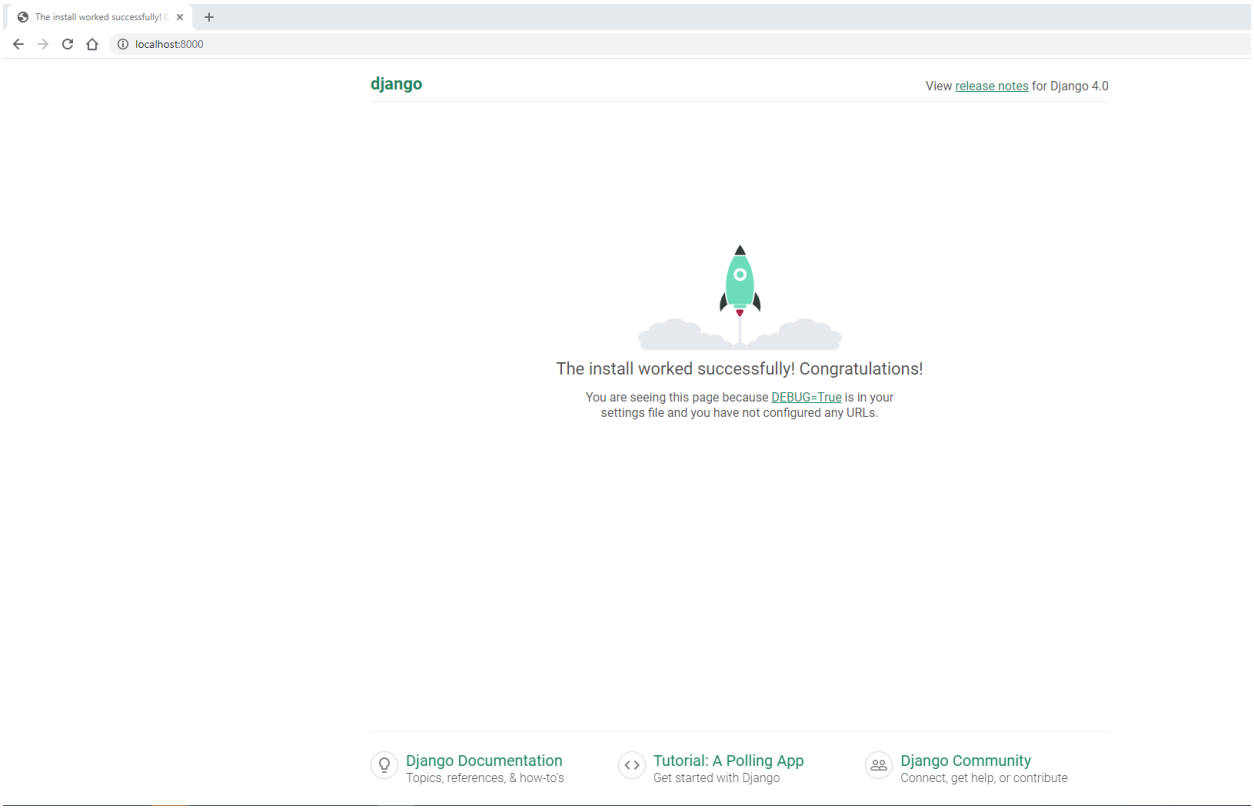

figura 26 Accesarea aplicatiei

Putem observa si mesajul standard care ne confirma ca instalarea aplicatiei a fost cu succes, insa nu avem inca paginile noastre create asa ca ne este afisat un mesaj standard.

Pentru a accesa pagina de admin discutata anterior, la adresa url-ului se va adauga “/admin” conform cu fisierul *urls.py* din cadrul aplicatiei.

```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('hospital.urls'))
]
```

figura 27 URL-ul admin din urls.py din cadrul proiectului

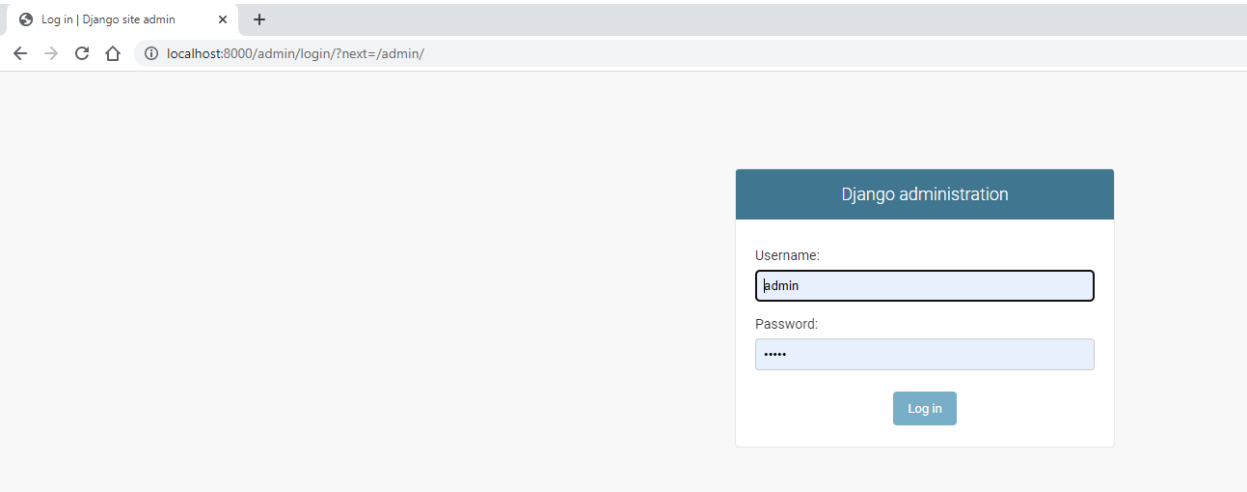

figura 28 URL-ul admin din browser

Se poate observa ca atunci cand accesam pagina de admin ni se va cere sa ne autentificam ca utilizatorul si parola unui *superuser.*

Odata accesat, din interiorul paginii putem face modificari asupra grupurilor si a utilizatorilor, insa nu vom lucra cu aceasta pagina, acesta fiind mai degraba o prezentare asupra functionalitatilor Django.

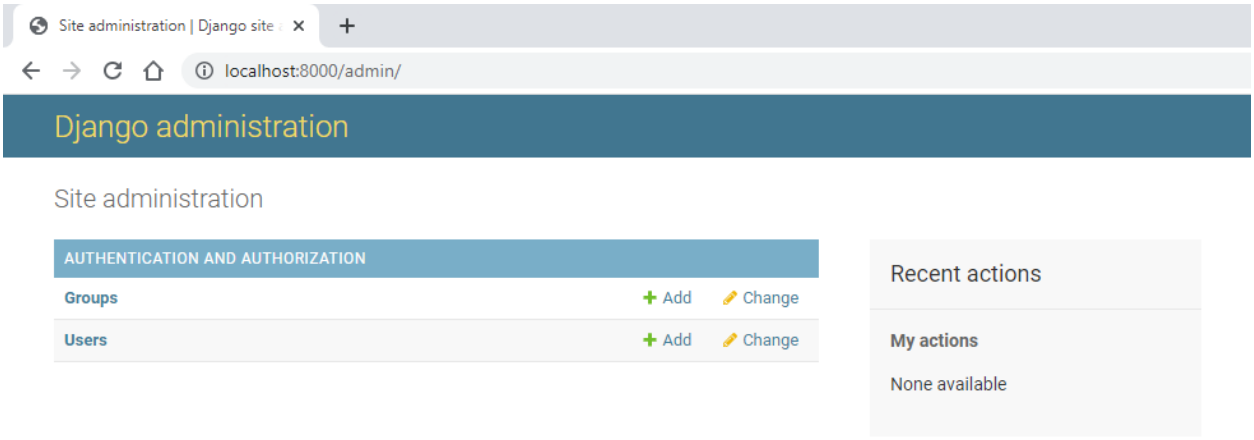

figura 29 Pagina de administrator Django

3.5. Directorul templates

Asa cum am mentionat anterior, acest director contine toate paginile HTML ale aplicatiei noastre. Folosim aceasta structura deoarece ne ajuta sa avem un proiect bine organizat si curat si ne ajuta atunci cand depanam diveresele probleme care apar in dezvoltarea aplicatiei. In cazul in care primim o eroare vom sti exact care este fisierul care a cauzat eroarea, unde este situat acesta, si in functie de locatia acestuia vom sti si care este rolul acestuia.

Primul fisier *HTML* pe care il vom crea este fisierul *home.html* descris in subcapitolul urmator.

3.5.1. Fisierul home.html

Acest fisier reprezinta pagina *HTML* pe care utilizatorul o va vedea prima oara cand va accesa aplicatia dezvoltata.

Incepem dezvoltarea acestei pagini prin incarcarea fisierelor statice (*{% load static %}*) si includerea altor pagini care au legatura directa cu pagina curenta (*{% include 'base.html' %}* si *{% include 'navigation.html' %}*, pagini pe care le voi descrie in subcapitolele ce urmeaza. Important de precizat aici sunt semnele de acolada si de procentaj "*{%%}*". Acestea ne ajuta sa scriem cod de Python in interiorul acestora.

Fisierul *home.html* este compus din mai multe sectiuni, prima sectiune find un diapozitiv de tip carusel cu trei imagini, fiecare imagine avand si un paragraf care descrie ce reprezinta fiecare din acele imagini, dar si doua butoane de inainte si inapoi pe care le vom folosi pentru a trece la urmatoarea imagine sau la imaginea precedenta.

Putem observa in inceputul sectiunii crearea diapozituvului cu trei imagini

```
<div id="carousel-slider" class="carousel slide" data-ride="carousel">
    <ol class="carousel-indicators">
        <li data-target="#carousel-slider" data-slide-to="0" class="active"></li>
        <li data-target="#carousel-slider" data-slide-to="1"></li>
        <li data-target="#carousel-slider" data-slide-to="2"></li>
    </ol>
```

figura 30 Diapozitivul de tip carusel

Acesta este urmat de elementele diapozitivului, primul fiind pentru ingrijirea posoperatorie, unde este inserata o imagine reprezentativa "*Post-Operative-Care.jpg*" din directorul *images*, urmat de un paragraf cu definitia ingrijirii post operatorii.

```
    <div class="carousel-inner">
        <div class="carousel-item active">
            <img src="{% static 'images/Post-Operative-Care.jpg' %}" class="d-block w-100" alt="...">
            <div class="carousel-caption text-white">
                <div class="background-1">
                    <h2>Îngrijire postoperatorie</h2>
                    <p>Îngrijirea postoperatorie este îngrijirea pe care o primiți după o intervenție chirurgicală.
                        îngrijirea de care aveți nevoie depinde de tipul de intervenție chirurgicală pe care o aveți,
                        precum și de istoricul dumneavoastră de sănătate. Adesea
                        include gestionarea durerii și îngrijirea rănilor.
                    </p>
                    <button class="btn btn-light"><a href="#departments" class="nav-link">Vezi toate serviciile</a></button>
                </div>
            </div>
        </div>
```

figura 31 Elementul Ingrijire postoperatorie

Se mai poate observa de asemenea si un buton “*Vezi toate serviciile*”, care este de fapt un link catre sectiunea de departamente din interiorul aceleasi pagini pe care o voi prezenta ulterior.

In mod similar sunt create si celelalte doua elemente ale diapozitivului, Interventia Chirugicala si Unitatea de terapie intensive neonatala.

```
<div class="carousel-item">
    <img src="{% static 'images/slide2.jpg' %}" class="d-block w-100" alt="...">
    <div class="carousel-caption text-white">
        <div class="background-1">
            <h2>Intervenția Chirurgicală</h2>
            <p>Chirurgia este o specialitate medicală care utilizează tehnici operative manuale și instrumentale asupra unei persoane
                pentru a investiga sau trata o afecțiune patologică, cum ar fi o boală sau o leziune, pentru a ajuta la îmbunătățirea
                funcția sau aspectul corporal sau pentru a repara zonele rupte nedorite.
            </p>
            <button class="btn btn-light"><a href="#departments" class="nav-link">Vezi toate serviciile</a></button>
        </div>
    </div>
</div>
```

figura 32 Elementul Interventia chirurgicala

```
<div class="carousel-item">
    <img src="{% static 'images/slide1.jpg' %}" class="d-block w-100" alt="...">
    <div class="carousel-caption text-white">
        <div class="background-1">
            <h2>Unitatea de terapie intensivă neonatală</h2>
            <p>O unitate de terapie intensivă neonatală, cunoscută și sub numele de creșă de terapie intensivă, este o terapie intensivă
                unitate specializată în îngrijirea nou-născuților bolnavi sau prematuri. Neonatal se referă la primul
                28 de zile de viață. Îngrijirea neonatală, cunoscută sub numele de creșe specializate sau terapie intensivă, a fost
                în jur din anii 1960.
            </p>
            <button class="btn btn-light"><a href="#departments" class="nav-link">Vezi toate serviciile</a></button>
        </div>
    </div>
</div>
```

figura 33 Unitatea de terapie intensiva neonatala

Sectiunea se incheie cu cele doua butoane de “Inainte” si “Inapoi”.

```
<a class="carousel-control-prev" href="#carousel-slider" role="button" data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Înapoi</span>
</a>
<a class="carousel-control-next" href="#carousel-slider" role="button" data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Înainte</span>
</a>
```

figura 34 Butoanele Inainte si Inapoi

Rezultatul final este prezentat in figurile ce urmeaza.

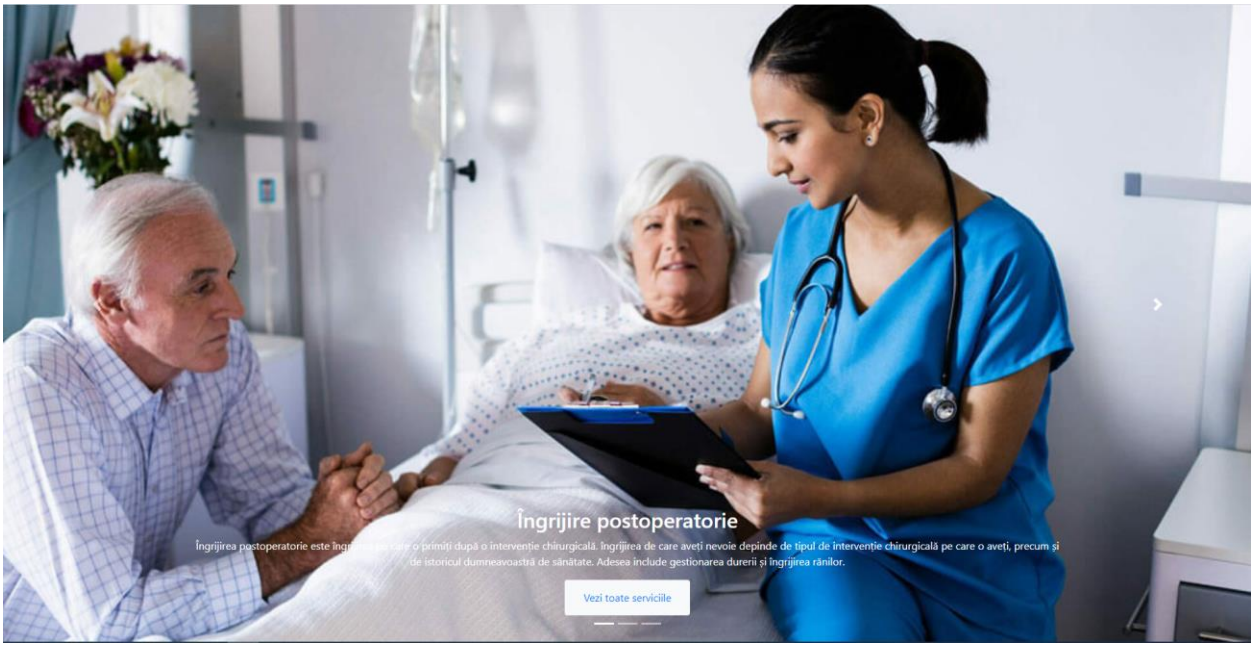

figura 35 Ingrijire postoperatorie

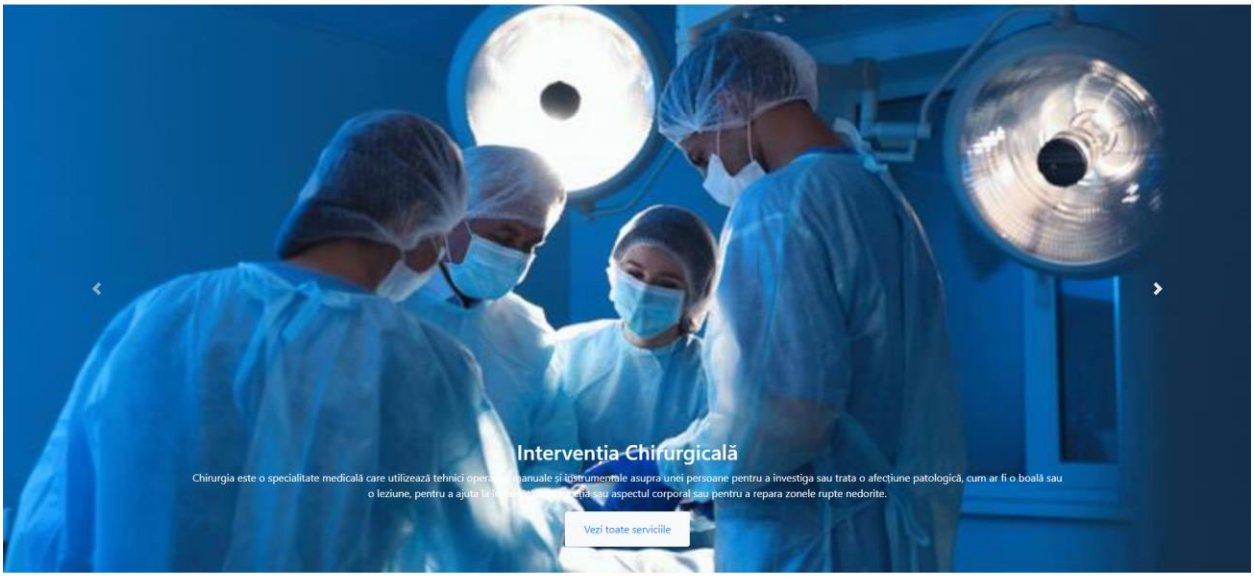

figura 36 Interventie Chirurgicala

figura 37 Unitatea de terapie intensiva neonatala

Sectiunea urmatoare este sectiunea de introducere care prezinta un mesaj descriptiv al aplicatiei si functionalitatile acesteia.

```
<div class="container-fluid">
    <div class="row">
        <div class="col-md-4">
            <img src=" ">
        </div>
        <div class="col-md-4" id="intro-text">
            <h1 class="text-center">
                <h1 class="text-center">Bun venit la sistemul de management al spitalului </h1>
            </h1>
            <p class="text-justify">
            <h5>Sistemul de management al spitalelor este un sistem în care poți avea orice tip de servicii. Este localizat în
                 București. Oferă diferite tipuri de servicii precum radiografie, diferite tipuri de intervenții chirurgicale cu
                 medici de mare renume.</h5>
            </p>
        </div>
        <div class="col-md-4">
            <div class="container-fluid">
                <br>
                <button class="btn btn-lg btn-block"><a href="#doctors" class="nav-link"><i
                            class="fa fa-user-md"></i>    Găsește Doctori</a></button>
                <br>
                <button class="btn btn-lg btn-block"><a href="http://maps.google.com/maps?q='Bucuresti'" class="nav-link"><i
                            class="fa fa-map-marker"></i>    Localizare</a></button>
                <br>
                <button class="btn btn-lg btn-block"><a href="tel:+40 000 000 000" class="nav-link"><i
                            class="fa fa-phone"></i>    Contact de urgență</a></button>
            </div>
        </div>
    </div>
</div>
```

figura 38 Sectiunea de introducere

Notabil in aceasta sectiune sunt cele trei butoane din dreapta paginii “*Gaseste Doctori”,* care la fel ca si in cazul butonului “*Vezi toate serviciile”* reprezinta un link catre o sectiune din interiorul paginii pe care o voi descrie ulterior, “*Localizare”* care este un link catre o pagina de *Google Maps* cu localizarea spitalului, in cazul nostru Bucuresti, si butonul de “*Contact de urgenta”* care reprezinta un numar de telefon la care utilizatorul poate apela.

```
<div class="col-md-4">
    <div class="container-fluid">
        <br>
        <button class="btn btn-lg btn-block"><a href="#doctors" class="nav-link"><i
                    class="fa fa-user-md"></i>    Găsește Doctori</a></button>
        <br>
        <button class="btn btn-lg btn-block"><a href="http://maps.google.com/maps?q='Bucuresti'" class="nav-link"><i
                    class="fa fa-map-marker"></i>    Localizare</a></button>
        <br>
        <button class="btn btn-lg btn-block"><a href="tel:+40 000 000 000" class="nav-link"><i
                    class="fa fa-phone"></i>    Contact de urgență</a></button>
    </div>
</div>
```

figura 39 Butoanele din sectiunea de introducere

Rezultatul final este prezentat in figura de mai jos.

figura 40 Sectiunea de introducere

In urma apasarii butonului “*Contact de urgenta”* se deschide o fereastra de tip *pop-up* prin care putem apela la numarul respectiv.

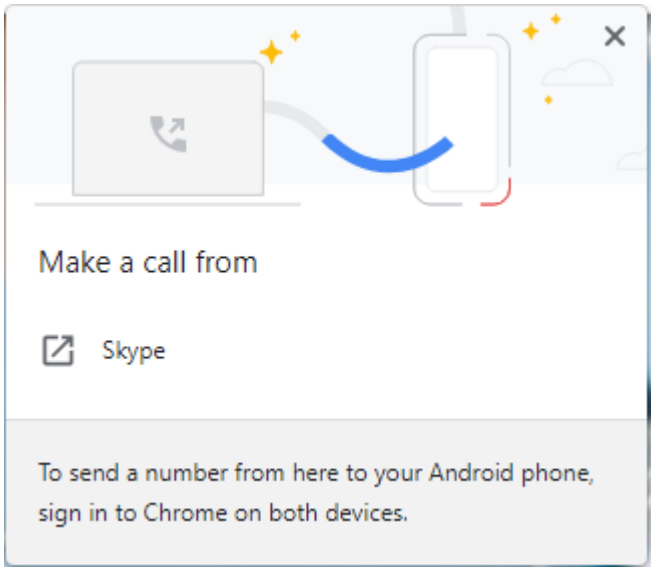

figura 41 Butonul contact de urgenta

Urmatoarea sectiune, este sectiunea de servicii medicale, care reprezinta cateva blocuri cu imagini reprezentative pentru fiecare serviciu medical pe care spitalul il ofera. Incep aceasta sectiune prin unui identificator “*departments*” catre care butonul “*Vezi toate serviciile*” va redirectiona.

```
<div class="container-fluid" id="departments">
    <div class="row">
        <div class="col-md-12">
            <h2 class="text-center text-primary">Departamente</h2>
            <br>
            <h1 class="text-center">Servicii Medicale</h1>
        </div>
    </div>
</div>
```

figura 42 Identificatorul departments

Acesta este urmat de sase blocuri cu imagini reprezentative departamentului respectiv. Toate blocurile sunt create in mod similar, diferenta facand doar imaginea inserata.

```
<div class="container-fluid">
    <div class="row owl-carousel owl-theme">
        <div class="card shadow text-center items" style="height: 159px;">
            <div class="card-img">
                <img src="{% static 'images/gynecologist.png' %}"
                    style="margin-left: auto; margin-right: auto; width: 25%; padding-top: 5px;">
            </div>

            <div class="card-body">
                <p class="card-text">Ginecolog si Obstetrician</p>
            </div>
        </div>
```

figura 43 Blocul de imagini al departamentelor

Rezultatul final este prezentat in figura urmatoare.

figura 44 Sectiunea servicii medicale

Urmeaza o sectiune asemanatoare celei prezentate anterior, numita sectiunea de “*Alte servicii*”.

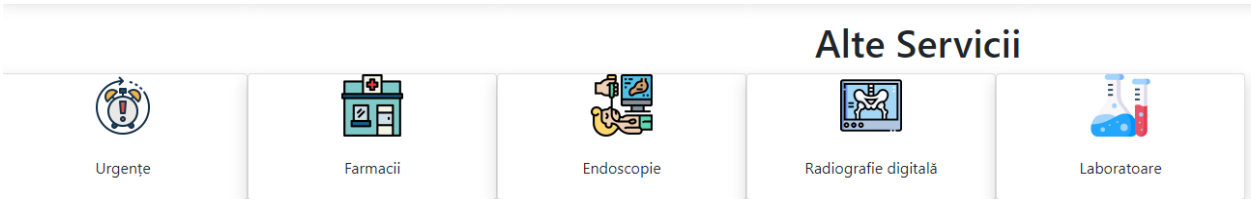

figura 45 Sectiunea alte servicii

In final, vom incheia cu sectiunea in care se prezinta doctorii. De asemenea este prezent si aici un identificator “doctors” la care butonul “*Gaseste doctori”* face referire.

```
<div class="container-fluid" id="doctors">
    <div class="row">
        <div class="col-md-12">
            <h4 class="text-center text-primary">Întâlnește-ne echipa</h4>
            <h1 class="text-center">Doctorii cu care ne mândrim</h1>
        </div>
    </div>
</div>
```

figura 46 Identificatorul doctors

Voi continua cu cate un bloc pentru fiecare doctor cu o scurta prezentare a specializarii acestuia si cu cate o imagine reprezentativa. Fiecare bloc ce descrie un doctor este implementat in mod similar.

```
<div class="row owl-carousel owl-theme">
    <div class="card shadow text-center items">
        <div class="card-body profile-card">
            <img class="img-thumbnail rounded-circle border-0" src="{% static 'images/female-doctor.png' %}">
            <h4 class="card-title" style="font-size: 18px;">Dr. Maria Grigorescu</h4>
            <h6 class="card-subtitle">Chirurg Consultant Senior</h6>
            <p class="card-text">MBBS, MS(TU), FMAS, DMAS (WLH)</p>
            <a href="#" class="card-link"><i class="fa fa-facebook"></i></a>
            <a href="#" class="card-link"><i class="fa fa-instagram"></i></a>
            <a href="#" class="card-link"><i class="fa fa-linkedin"></i></a>
        </div>
    </div>
```

figura 47 Blocul doctorilor

Rezultatul final este prezentat mai jos.

figura 48 Sectiunea doctorilor

3.5.2. Pagina about.html

Aceasta este pagina “Despre noi” care contine o scurta prezentare a spitalului.

```
{% include 'base.html' %}
{% include 'navigation.html' %}

{% block content %}
<h1 class="text-center" style="margin-top: 50px;">Despre noi</h1>
<div class="container" style="margin-top: 100px;">
    <div class="row">
        <div class="col-md-6">
            <h1 class="strong">Cine suntem<br>Și ce facem</h1>
            <p class="lead">Acesta este portalul lider mondial pentru<br>managerierea spitalelor </p>
        </div>
        <div class="col-md-6">
            <p>Cu ajutorul serviciilor noastre, vă puteți asigura că veți beneficia de servicii medicale de calitate în confortul casei dumneavoastră.</p>
            <p>Oferim specialiști, însoțitori și kinetoterapeuți pentru vizite la domiciliu care au trecut de principiile noastre aprofundate de înrolare
               și au avut experiențele și învățarea restaurativă verificate de specialiști seniori.</p>
        </div>
        <div class="row">
            <div class="site-heading text-center">
                <h3>Echipa noastră</h3>
                <p>specialiștilor noștri sunt persoane din organisme de acreditare terapeutică universală. Datorită utilizării administrațiilor noastre,
                   pacienții noștri pot rămâne în casele lor mai mult timp, pot pune deoparte bani în plus și pot avea sentimente autentice de liniște.</p>
                <div class="border"></div>
            </div>
        </div>
    </div>
</div>
{% endblock content %}
{% include 'footer.html' %}
```

figura 49 about.html

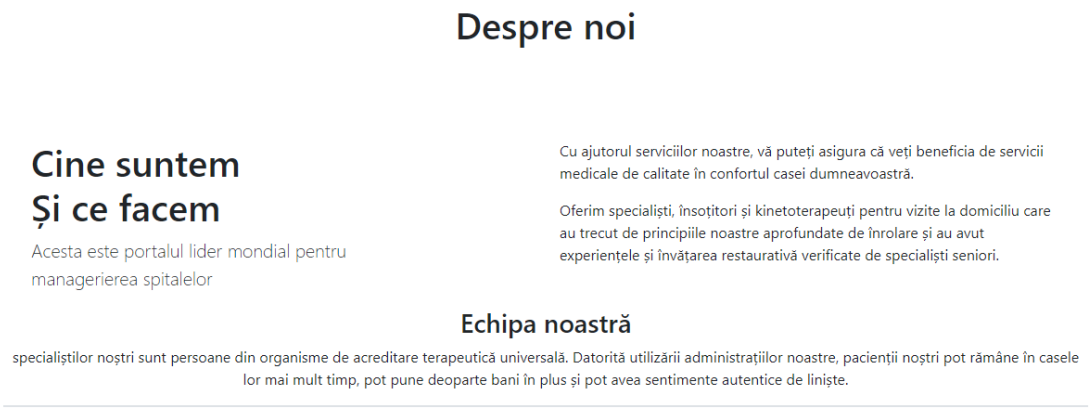

figura 50 Pagina despre noi

3.5.3. Pagina contact.html

Aceasta este pagina prin care utilizatorul poate vedea metodele prin care poate lua legatura cu spitalul. Aceasta este formata din patru blocuri, unul pentru contactul prin telefon, unul pentru contactul prin e-mail, pentru adresa sediului si inca un bloc pentru programul spitalului. Ca si in cazul paginii *home.html* blocul pentru contact prin telefon va deschide o fereastra de tip pop-up cu optiunea de a apela la numarul respectiv, blocul "*Trimite-ne un e-mail*" va deschide o fereastra de tip pop-up cu posibilitatea de a trimite un e-mail la adresa respective, iar blocul cu adresa va redirectiona catre o pagina de *Google Maps* cu adresa spitalului. De notat la aceasta pagina este stilizarea de la inceputul acesteia care face referire si la fisierul *mystyle.css* si care creeaza un effect de planare atunci cand trecem cu mouse-ul peste unul din blocuri.

figura 51 Efectul de planare al blocurilor din contact.html

```
{% include 'base.html' %}
{% include 'navigation.html' %}
<style>
    .mydivstyle:hover {
        box-shadow: 0 0 20px 0 rgba(0, 0, 0, 0.3);
        transform: translateY(-20px);
    }
</style>


{% block content %}
<div class="container" style="margin-top: 50px;">
    <h1 class="text-center">Ia legătura cu noi</h1>
    <div class="row mt-5 text-center">
        <div class="col-lg-3 col-sm-6 mydivstyle">
            <h4><i class="fas fa-phone-square"> Contactează-ne prin telefon</i></h4>
            <p class="text-danger font-weight-bold mt-3"><a href="tel:+40 000 000 000">+40 000 000 000</a></p>
        </div>
        <div class="col-lg-3 col-sm-6 mydivstyle">
            <h4><i class="fa fa-envelope"></i> Trimite-ne un e-mail</h4>
            <p class="text-danger font-weight-bold mt-3"><a href="mailto:abc@gmail.com">abc@gmail.com</a></p>
        </div>
        <div class="col-lg-3 col-sm-6 mydivstyle">
            <h4><i class="fa fa-address-card"></i> Adresa sediului </h4>
            <p class="text-danger font-weight-bold mt-3"><a href="http://maps.google.com/maps?q='Bucuresti'">Bucuresti</a></p>
        </div>
        <div class="col-lg-3 col-sm-6 mydivstyle">
            <h4><i class="fa fa-clock"></i> Programul Spitalului </h4>
            <p class="font-weight-bold mt-3">Deschis Nonstop</p>
        </div>
    </div>
</div>
{% endblock content %}
{% include 'footer.html' %}
```

figura 52 Pagina contact.html

3.5.4. Pagina login.html

Urmatoarea pagina despre care voi vorbi este pagina *login.html.* In aceasta pagina utilizatorul se poate autentifica in aplicatie pentru a putea beneficia de functiile de administrator ale aplicatiei. Incep dezvoltarea paginii prin doua clauze de “if”. Acestea sunt necesare pentru a verifica daca utilizatorul a introdus datele de autentificare corect.

```
{} 1   {% include 'base.html' %}
   2   {% include 'navigation.html' %}
   3   {% block content %}
   4
   5   {% if error == "no" %}
   6   <script>
   7       alert('Autentificare cu succes!');
   8       window.location = "{% url 'dashboard' %}"
   9   </script>
  10   {% endif %}
  11   {% if error == "yes" %}
  12   <script>
  13       alert('Autentificare invalidă! Mai încearcă o dată.');
  14   </script>
  15   {% endif %}
```

figura 53 Clauzele "if" din pagina login.html

Fiecare din aceasta clauza va genera un mesaj de alerta in functie daca datele de autentificare introduse au fost corecte, caz in care in pagina va aparea un mesaj “Autentificare cu succes”, sau incorecte, caz in care se va afisa mesajul “Autentificare invalida! Mai incearca o data.”. Important de notat in aceasta pagina este faptul ca cele doua clauze “if” se folosesc de o variabila “error” pentru a putea verifica daca datele introduse sunt corecte sau nu. Mai multe despre aceasta variabila si despre modul in care este asignata valoarea acesteia voi discuta in capitolul *views.py.*

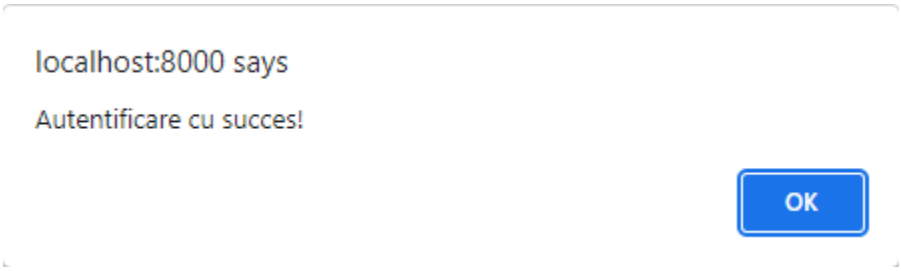

figura 54 Autentificare cu succes

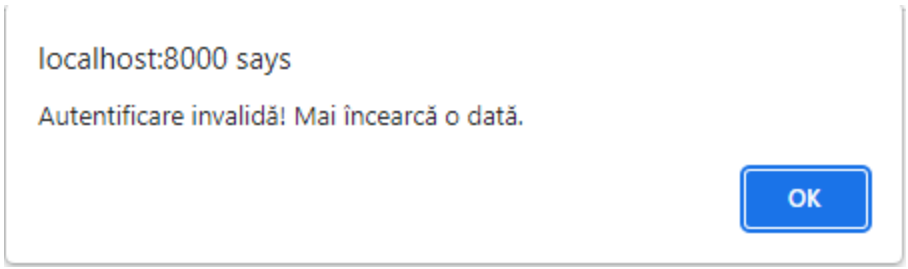

figura 55 Autentificare invalida

In continuare, voi crea campurile necesare pentru utilizator si parola.

```
<div class="container">
    <h2 class="text-center">Autentificare Admin</h2>
    <form method="POST">
        {% csrf_token %}
        <label>Numele de utilizator</label>
        <input type="text" class="form-control" name="uname">
        <label>Parola</label>
        <input type="password" class="form-control" name="pwd">

        <input type="submit" value="Autentificare" class="form-control btn btn-primary">
    </form>
</div>
```

figura 56 Campurile pentru utilizator si parola

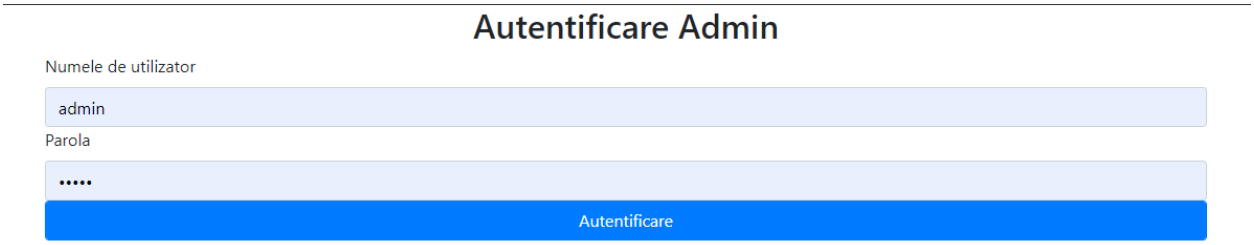

figura 57 Pagina de autentificare

3.5.5. Pagina add_docotor.html

Aceasta este pagina prin care administratorul poate adauga un doctor in aplicatie. Insa, pentru a putea face acest lucru trebuie mai intai sa creem modelul pentru doctori. In subcapitolele ce urmeaza voi crea modelele pentru doctori, pacienti si pentru programari.

3.5.5.1. Modelul doctorilor

Modelul doctorilor reprezinta datele de care avem nevoie pentru a crea un doctor in baza de date. Acest model este de fapt o clasa cu variabilele care definesc un obiect de tip doctor. Pentru aceasta, se va deschide fisierul *models.py* si vom crea clasa *Doctor* cu campurile de *name* pentru numele doctorului, acest camp fiind de tip Char cu o lungime maxima de 50 de caractere, *mobile,* camp de tip Integer si *special,* pentru specialitatea doctorului, acest camp fiind tot un camp de tip Char cu lungime maxima de 50 de caractere.

```python
from django.db import models

# Create your models here.
class Doctor(models.Model):
    name = models.CharField(max_length=50)
    mobile = models.IntegerField()
    special = models.CharField(max_length=50)

    def __str__(self):
        return self.name
```

figura 58 Modelul Doctor

De mentionat este faptul ca la finalul clasei, am definit si o functie "__str__(self)". Scopul acestei functii este de a intoarce campul care ni se pare relevant atunci cand dorim sa facem o interogare in baza de date. In cazul de fata, atunci cand dorim sa interogam un doctor din baza de date, acesta ne va intoarce numele doctorului respectiv.

Pentru a putea face interogari in baza de date, trebuie mai intai sa migram modificarile facute asupra modelelor create. Acest lucru se realizeaza prin executarea comenzile de *makemigrations* si *migrate* descrise in capitolul 3.3. Migrarile si crearea de superusers.

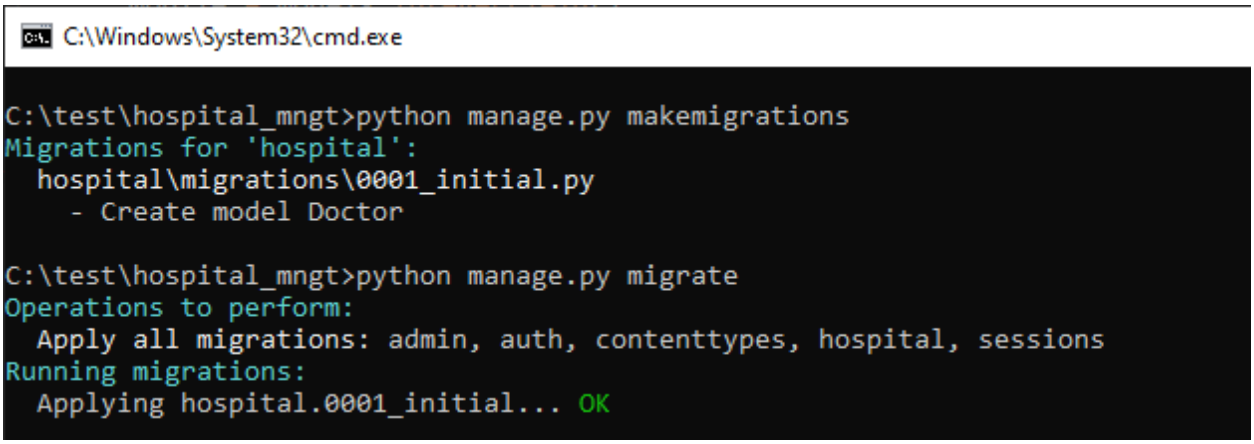

figura 59 Migrarea modelelor

3.5.5.2. Modelul pacientilor

Similar cu modelul doctorilor descries anterior, se vac rea modelul *Pacient,* cu campurile *name,* de tip Char cu lungime maxima de 50 de caractere, *gender,* tot de tip Char, cu lungime maxima de 10 caractere reprezentand sexul pacientului, *mobile,* de tip Integer, in acest caz, spre deosebire de campul cu acelasi nume din modelul *Doctor* poate fi *null,* si campul *address*, de tip Text, reprezentand adresa pacientului. Ca si in cazul doctorilor, acest model are implemantata functia "*__str__(self)"* care va intoarce numele pacientului.

```
class Patient(models.Model):
    name = models.CharField(max_length=50)
    gender = models.CharField(max_length=10)
    mobile = models.IntegerField(null=True)
    address = models.TextField()

    def __str__(self):
        return self.name
```

figura 60 Modelul Patient

3.5.5.3. Modelul programarilor

Pentru creearea modelului programarilor, ma folosesc de celelalte doua modele create anterior, *Doctor* si *Patient,* avand in vedere faptul ca un pacient isi poate face programare la un doctor. In acest caz, in clasa *Appointment* voi crea doua campuri de tip *ForeignKey,* unul pentru doctor si unul pentru pacient. De notat la aceste doua campuril este faptul ca atunci cand unul din campuri este sters, se vor sterge toate obiectele care referentiaza obiectul sters. Acest lucru se realizeaza prin "*on_delete=models.CASCADE".* Pe langa aceste doua campuri, o programare mai are nevoie si de o data si o ora la care se face programarea, acestea realizandu-se prin campurile *date* de tip *Date* si *time* de tip *Time.*

```
class Appointment(models.Model):
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE)
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE)
    date = models.DateField()
    time = models.TimeField()

    def __str__(self):
        return self.Doctor.name + "_" + self.Patient.name
```

figura 61 Modelul programarilor

Acum ca toate modelele sunt create voi discuta despre pagina *add_doctor.html.* Ca si in cazul paginii *login.html* aceasta pagina incepe prin doua clauze "if" care verifica daca datele introduse sunt corecte, caz in care se va afisa mesajul "*Record salvat cu success!",* sau daca datele introduse sunt invalide, caz in care se va afisa mesajul "*Recordul nu a putut fi salvat! Va rugam mai incercati o data.".*

```
{% include 'base.html' %}
{% include 'navigation.html' %}

{% block content %}
{% if error == "no" %}
<script>
    alert('Record salvat cu succes!');
    window.location = "{% url 'view_doctor' %}"
</script>
{% endif %}
{% if error == "yes" %}
<script>
    alert('Recordul nu a putut fi salvat! Vă rugam mai încercați o data.');
</script>
{% endif %}
```

figura 62 Clauzele if pentru adaugarea doctorului

In continuare, voi crea campurile specifice pentru numele doctorului, numarului de telefon si pentru specialitatea acestuia.

```
<div class="container">
    <h2 class="text-center">Adaugă un doctor</h2>
    <form method="POST">
        {% csrf_token %}
        <label>Numele Doctorului</label>
        <input type="text" class="form-control" name="name">
        <label>Telefonul Mobil</label>
        <input type="text" class="form-control" name="mobile">
        <label>Specialitatea</label>
        <input type="text" class="form-control" name="special">

        <input type="submit" value="Trimitere" class="form-control btn btn-primary">
    </form>
</div>
```

figura 63 Campurile pentru adaugarea unui doctor

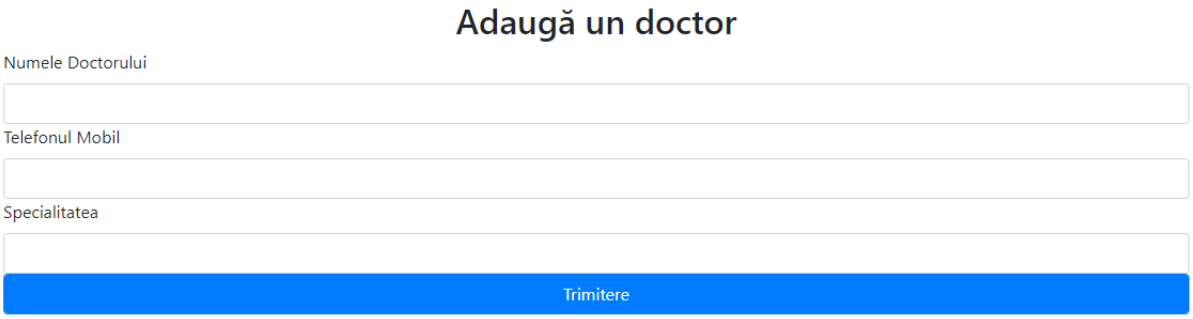

figura 64 Pagina add_doctor.html

3.5.6. Pagina add_patient.html

Aceasta pagina a fost implementata similar celei descrise anterior, diferenta facand-o doar campurile specifice unui model de tip *Patient.*

```
{% include 'base.html' %}
{% include 'navigation.html' %}

{% block content %}
{% if error == "no" %}
<script>
    alert('Record salvat cu succes!');
    window.location = "{% url 'view_patient' %}"
</script>
{% endif %}
{% if error == "yes" %}
<script>
    alert('Recordul nu a putut fi salvat! Vă rugam mai încercați o data.');
</script>
{% endif %}
<div class="container">
    <h2 class="text-center">Adaugă un pacient</h2>
    <form method="POST">
        {% csrf_token %}
        <label>Numele Pacientului</label>
        <input type="text" class="form-control" name="name">
        <label>Sexul</label>
        <input type="text" class="form-control" name="gender">
        <label>Telefonul Mobil</label>
        <input type="text" class="form-control" name="mobile">
        <label>Adresa</label>
        <input type="text" class="form-control" name="address">
        <input type="submit" value="submit" class="form-control btn btn-primary">
    </form>
</div>
{% endblock content %}
{% include 'footer.html' %}
```

figura 65 Pagina add_patient.html

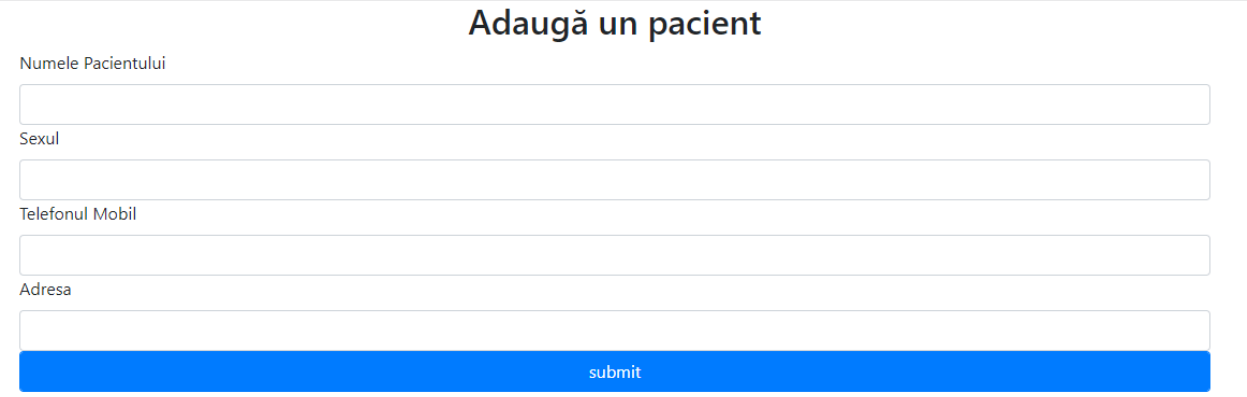

figura 66 Afisarea paginii add_patient.html

3.5.7. Pagina add_appointment.html

Aceasta este pagina in care se pot face programari. De notat la aceasta pagina este faptul ca pentru doctori si pacienti se va face o itererare prin doctorii si pacientii din baza de date. Astfel, administratorul poate alege unul din pacientii din baza de date caruia sa I se faca o programare la unul din doctorii din baza de date.

```
{% include 'base.html' %}
{% include 'navigation.html' %}

{% block content %}
{% if error == "no" %}
<script>
    alert('Record salvat cu succes!');
    window.location = "{% url 'view_appointment' %}"
</script>
{% endif %}
{% if error == "yes" %}
<script>
    alert('Recordul nu a putut fi salvat! Vă rugam mai încercați o data.');
</script>
{% endif %}
<div class="container">
    <h2 class="text-center">Fă o programare</h2>
    <form method="POST">
        {% csrf_token %}
        <label>Numele Doctorului</label>
        <select name="doctor" class="form-control">
            {% for i in doctor %}
            <option value="{{ i.name }}">{{i.name}}</option>
            {% endfor %}
        </select>
        <label>Numele Pacientului</label>
        <select name="patient" class="form-control">
            {% for i in patient %}
            <option value="{{ i.name }}">{{i.name}}</option>
            {% endfor %}
        </select>
        <label>Data</label>
        <input type="date" class="form-control" name="date">
        <label>Ora</label>
        <input type="time" class="form-control" name="time">

        <input type="submit" value="submit" class="form-control btn btn-primary">
    </form>
</div>
{% endblock content %}
{% include 'footer.html' %}
```

figura 67 Pagina add_appointment.html

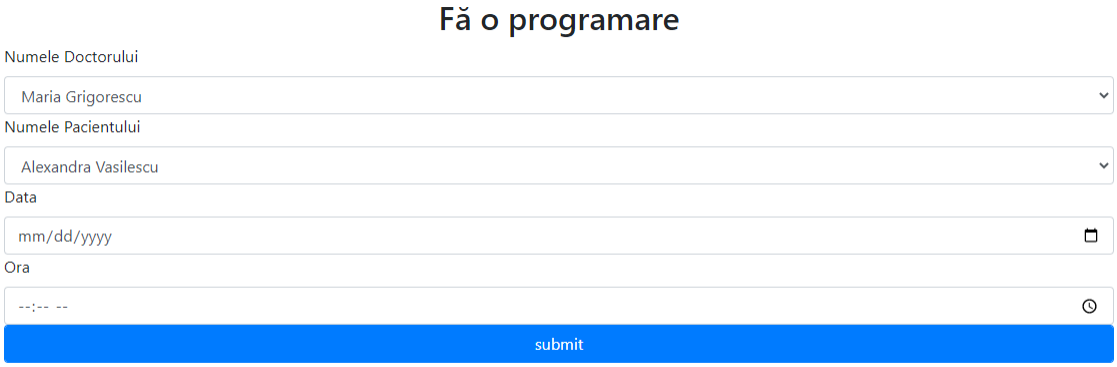

figura 68 Afisarea paginii add_appointment.html

3.5.8. Paginile view_doctor.html, view_patient.html si view_appointment.html

Aceste trei pagini sunt create in mod similar si reprezinta paginile pentru afisarea doctorilor, pacientilor si respectiv programarilor. De notat la aceste pagini este faptul ca pentru fiecare pagina se face o iterare asupra datelor din baza de date, spre exemplu, toti doctorii vor fi salvati in variabila *doc* si apoi se face o iterare prin aceasta variabila *"for I in doc",* apoi pentru fiecare record se va afisa id-ul respective "i.id", aceasta valoarea fiind incrementata in mod automat de Django, apoi si vor afisa numele, telefonul mobil si specialitatea doctorului. "*i.name, i.mobile, i.special".* Acelasi lucru se intampla si pentru programari, unde toate programarile sunt salvate in variabila *apt,* si pentru pacienti, *pat,* pentru fiecare afisandu-se campurile specifice modelului corespunzator. De notat mai este si faptul ca fiecare pagina ofera posibilitatea de a sterge un record. Odata apasat butonul de *Sterge* se va afisa un mesaj de confirmare "*Sunteti siguri ca vreti sa stergeti acest record?".*

```
{% include 'base.html' %}
{% include 'navigation.html' %}
{% block content %}
<div>
    <h2>Afișare Doctori</h2>
    <table id="example" class="table table-hover">
        <thead>
            <tr>
                <th>Id-ul Doctorului</th>
                <th>Numele</th>
                <th>Telefonul Mobil</th>
                <th>Specialitatea</th>
                <th>Șterge</th>
            </tr>
        </thead>
        <tbody>
            {% for i in doc %}
            <tr>
                <td>{{i.id}}</td>
                <td>{{i.name}}</td>
                <td>{{i.mobile}}</td>
                <td>{{i.special}}</td>
                <td><a href="{% url 'delete_doctor' i.id %}"
                    onclick="return confirm('Sunteți siguri că vreți să ștergeți acest record?')" class="btn btn-danger">Șterge</a>
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
{% endblock content %}
{% include 'footer.html' %}
```

figura 69 Pagina view_doctor.html

```
{% include 'base.html' %}
{% include 'navigation.html' %}
{% block content %}
<div>
    <h2>Afișare Pacienți</h2>
    <table id="example" class="table table-hover">
        <thead>
            <tr>
                <th>Id-ul Pacientului</th>
                <th>Numele</th>
                <th>Sexul</th>
                <th>Telefonul Mobil</th>
                <th>Adresa</th>
                <th>Șterge</th>
            </tr>
        </thead>
        <tbody>
            {% for i in pat %}
            <tr>
                <td>{{i.id}}</td>
                <td>{{i.name}}</td>
                <td>{{i.gender}}</td>
                <td>{{i.mobile}}</td>
                <td>{{i.address}}</td>
                <td><a href="{% url 'delete_patient' i.id %}"
                    onclick="return confirm('Sunteți siguri că vreți să ștergeți acest record?')" class="btn btn-danger">Șterge</a>
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
{% endblock content %}
{% include 'footer.html' %}
```

figura 70 Pagina view_patient.html

```
{% include 'base.html' %}
{% include 'navigation.html' %}
{% block content %}
<div>
    <h2>Afișare Programări</h2>
    <table id="example" class="table table-hover">
        <thead>
            <tr>
                <th>Id-ul Programării</th>
                <th>Numele Doctorului</th>
                <th>Numele Pacientului</th>
                <th>Data</th>
                <th>Ora</th>
                <th>Șterge</th>
            </tr>
        </thead>
        <tbody>
            {% for i in apt %}
            <tr>
                <td>{{i.id}}</td>
                <td>{{i.doctor.name}}</td>
                <td>{{i.patient.name}}</td>
                <td>{{i.date}}</td>
                <td>{{i.time}}</td>
                <td><a href="{% url 'delete_appointment' i.id %}"
                    onclick="return confirm('Sunteți siguri că vreți să ștergeți acest record?')" class="btn btn-danger">Șterge</a>
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
{% endblock content %}
{% include 'footer.html' %}
```

figura 71 Pagina view_appointment.html

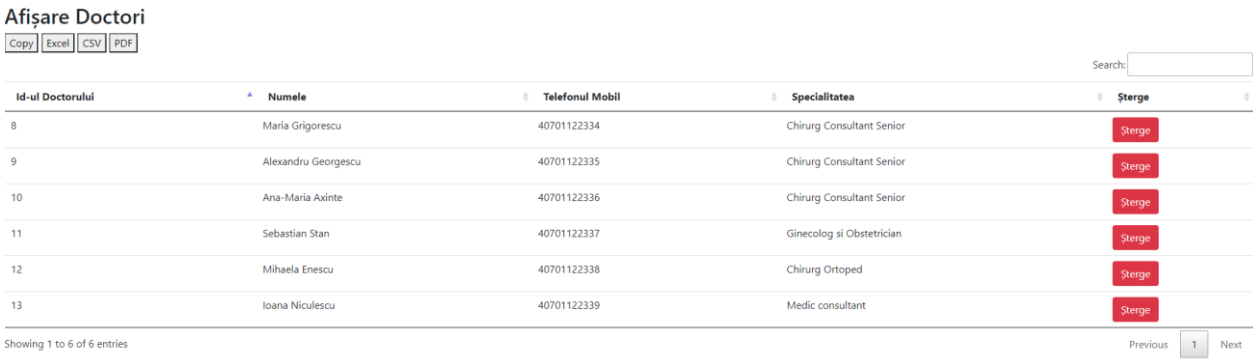

figura 72 Afisarea paginii view_doctor.html

figura 73 Afisarea paginii view_patient.html

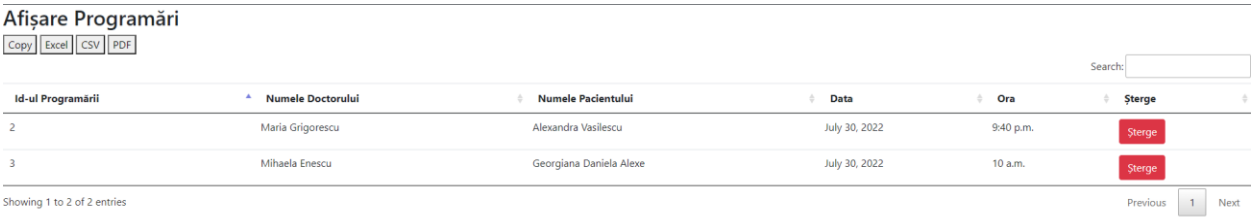

figura 74 Afisarea paginii view_appointment.html

Voi discuta despre modul in care au fost create variabilele *doc, pat* si *apt* in capitolul *views.py,* iar despre bara de search si generarea de fisiere *Excel, CSV* si *PDF* in capitolul *3.5.10. base.html.*

3.5.9. Pagina navigation.html

Aceasta pagina este bara de navigatie a aplicatiei web. Aceasta ofera posibilitatea de a naviga prin aplicatie deoarece contine butoanele care duc catre paginile *Acasa, Contact, Despre noi* si *Autentificare* pentru utilizatorii care nu sunt autentificati in aplicatie, si paginile *Bord,* cate doua butoate extensibile pentru *Doctori* si *Pacienti,* dupa apasarea fiecaruie dintre aceste butoane apar butoanele pentru inca doua pagini, *Afisare Doctor, Adaugare Doctor*, respectiv *Afisare Pacienti, Adaugare Pacienti*, inca un buton extensibil pentru *Programari* si butonul de *Deconectare* pentru utilizatorii autentificati si cu drepturi de administrator. De asemenea se mai poate observa si numele temei in bara de navigiatie, acesta fiind tot un buton care va duce catre pagina de *Acasa*.

figura 75 Bara de navigatie utilizatori neautentificati

figura 76 Bara de navigatie utilizatori autentificati

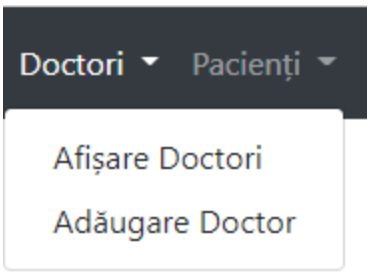

figura 77 Butonul extensibil pentru Doctori

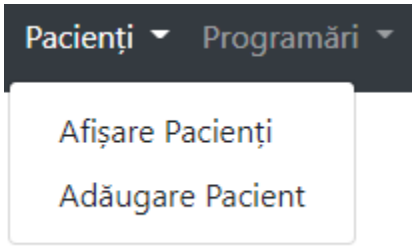

figura 78 Butonul extensibil pentru Pacienti

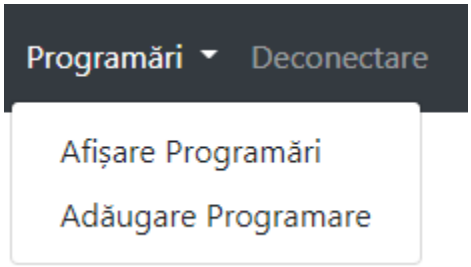

figura 79 Butonul extensibil pentru Progrmari

Voi incepe dezvoltarea paginii prin crearea barii de navigatie, a butonului extensibil si a butonului cu denumirea aplicatiei.

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <a class="navbar-brand" href="{% url 'home' %}">Sistemul de Management al Spitalului</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false"
    <span class="navbar-toggler-icon"></span>
  </button>

  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav ml-auto">
```

figura 80 Dezvoltarea barii de navigatie

In continuare pun o conditie ca daca utilizatorul nu este autentificat in aplicatie si implicit sa afiseze butoanele cu link-urile catre paginile pentru *Acasa, Contact, Despre noi* si *Autentificare.*

```
{% if not request.user.is_staff %}
<li class="nav-item active">
  <a class="nav-link" href="{% url 'home' %}">Acasă <span class="sr-only">(current)</span></a>
</li>
<li class="nav-item">
  <a class="nav-link" href="{% url 'contact' %}">Contact</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="{% url 'about' %}">Despre noi</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="{% url 'admin_login' %}">Autentificare</a>
</li>
{% endif %}
```

figura 81 butoanele pentru paginile Acasa, Contact, Despre noi si Autentificare

In continuare pun conditia pentru afisarea butoanelor in cazul in care utilizatorul este autentificat in aplicatie si are drepturi de administrator.

```
{% if request.user.is_staff %}
<li class="nav-item">
  <a class="nav-link" href="{% url 'dashboard' %}">Bord</a>
</li>
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
    Doctori
  </a>
  <div class="dropdown-menu" aria-labelledby="navbarDropdown">
    <a class="dropdown-item" href="{% url 'view_doctor' %}">Afișare Doctori</a>
    <a class="dropdown-item" href="{% url 'add_doctor' %}">Adăugare Doctor</a>
</li>
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
    Pacienți
  </a>
  <div class="dropdown-menu" aria-labelledby="navbarDropdown">
    <a class="dropdown-item" href="{% url 'view_patient' %}">Afișare Pacienți</a>
    <a class="dropdown-item" href="{% url 'add_patient' %}">Adăugare Pacient</a>
</li>
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
    Programări
  </a>
  <div class="dropdown-menu" aria-labelledby="navbarDropdown">
    <a class="dropdown-item" href="{% url 'view_appointment' %}">Afișare Programări</a>
    <a class="dropdown-item" href="{% url 'add_appointment' %}">Adăugare Programare</a>
</li>
  <li class="nav-item">
    <a class="nav-link" href="{% url 'admin_logout' %}">Deconectare</a>
  </li>
{% endif %}
```

figura 82 butoanele pentru paginile utilizatorilor autentificati si cu drepturi de administrator

3.5.10. Pagina base.html

Aceasta pagina este responsabila cu tema aplicatiei. Aceasta contine link-uri catre paginile sursa de unde importam clasele folosite in aplicatie. Spre exemplu, clasele pentru coloane au fost importate chiar de pe site-ul oficial al Bootstrap.

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!- Columns ->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css">
    <script src="https://cdn.jsdelivr.net/npm/jquery@3.6.0/dist/jquery.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.js"></script>
```

figura 83 Clasele pentru coloane

Deoarece Bootstrap 4 nu are propria sa librarie pentru *icons,* a trebuit sa ma folosesc de una din multele librarii gratuite, spre exemplu fontawesome.

```
<!- Icons ->
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.0/css/all.css">
<link rel="stylesheet" type="text/css" href="{% static 'css/mystyle.css' %}">
```

figura 84 Clasele pentru Icons

De asemenea se observa ca aici este folosit si fisierul css *mystyle.css* care coloreaza butoanele cu culoarea albastra.

In continuare avem clasa pentru butoanele de export *csv, pdf* si in format excel si script-ul pentru crearea acestora.

```
<!- Export buttons ->
<link rel="stylesheet" href="https://cdn.datatables.net/1.11.5/css/jquery.dataTables.min.css">
<link rel="stylesheet" href=https://cdn.datatables.net/buttons/2.2.2/css/buttons.dataTables.min.css">
<script src="https://code.jquery.com/jquery-3.5.1.js"></script>
<script src="https://cdn.datatables.net/1.11.5/js/jquery.dataTables.min.js"></script>
<script src="https://cdn.datatables.net/buttons/2.2.2/js/dataTables.buttons.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jszip/3.1.3/jszip.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/pdfmake/0.1.53/pdfmake.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/pdfmake/0.1.53/vfs_fonts.js"></script>
<script src="https://cdn.datatables.net/buttons/2.2.2/js/buttons.html5.min.js"></script>
<title>Sistemul de Management al Spitalului</title>
<script>
    $(document).ready(function() {
        $('#example').DataTable( {
            dom: 'Bfrtip',
            buttons: [
                'copyHtml5',
                'excelHtml5',
                'csvHtml5',
                'pdfHtml5'
            ]
        } );
    } );
</script>
```

figura 85 Butoanele de export

Tot codul mentionat pana acum in aceasta pagina trebuie sa fie inserat in campul *head* al paginii, iar in campul *body* se va crea un *block* denumit *content,* astfel, paginile care folosesc acest *block* se vor putea folosi de clase importate mai sus.

```
<body>

    {% block content %}

    {% endblock content %}

</body>
</html>
```

figura 86 Crearea block-ului content

3.5.11. Pagina footer.html

In incheierea fiecarei pagini, mai includem inca o pagina "*footer.html*" care reprezinta blocul cu drepturile de autor.

```
{% include 'footer.html' %}
```

figura 87 Includerea paginii footer.html

```
<nav class="navbar navbar-expand-sm bg-dark navbar-dark fixed-bottom"></class>
    <div class="footer-bottom">
        <div class="container">
            <div class="col-sm-12 text-center ">
                <p class="text-white">CopyRight © 2022 Aplicație dezvoltată de Ana</p>
            </div>
        </div>
    </div>
</nav>
```

figura 88 footer.html

Efectul acestei pagini este prezentat mai jos.

figura 89 Efectul paginii footer.html

3.5.12. Pagina index.html

Aceasta pagina este un bord in care este prezentata situatia doctorilor, pacientilor si a programarilor. Este pagina pe care administratorul o vede prima data cand se autentifica in pagina aplicatie. Aceasta are o stilizare ca atunci cand trecem cu mouse-ul peste unul din blocurile pentru doctor, pacient si respective programari sa planeze. De asemenea, fiecare bloc are cate un link aferent, pentru doctori, acesta va trimite catre pagina de afisare doctori, pentru pacienti va trimite catre pagina afisare pacienti si pentru programari va trimite catre pagina afisare programari.

```
<style>
    .mydivstyle:hover {
        box-shadow: 0 0 20px 0 rgba(0, 0, 0, 0.3);
        transform: translateY(-20px);
    }

    hr.rounded {
        border-top: 10px solid lightblue;
        border-radius: 5px;
    }
</style>

{% block content %}
<h1 class="text-center">Bordul Administratorului</h1>
<br><br><br><br><br>

<div class="container">
    <div class="row">
        <div class="col-sm-4 mydivstyle">
            <h2 style="color: red;">{{ d }}</h2>
            <a href="{% url 'view_doctor' %}">
                <h4 style="color: blue">Doctori</h4>
                <hr class="rounded">
            </a>
        </div>
        <div class="col-sm-4 mydivstyle">
            <h2 style="color: red;">{{ p }}</h2>
            <a href="{% url 'view_patient' %}">
                <h4 style="color: blue">Pacienți</h4>
                <hr class="rounded">
            </a>
        </div>
        <div class="col-sm-4 mydivstyle">
            <h2 style="color: red;">{{ a }}</h2>
            <a href="{% url 'view_appointment' %}">
                <h4 style="color: blue">Programări</h4>
                <hr class="rounded">
            </a>
        </div>
    </div>
</div>
```

figura 90 Pagina index.html

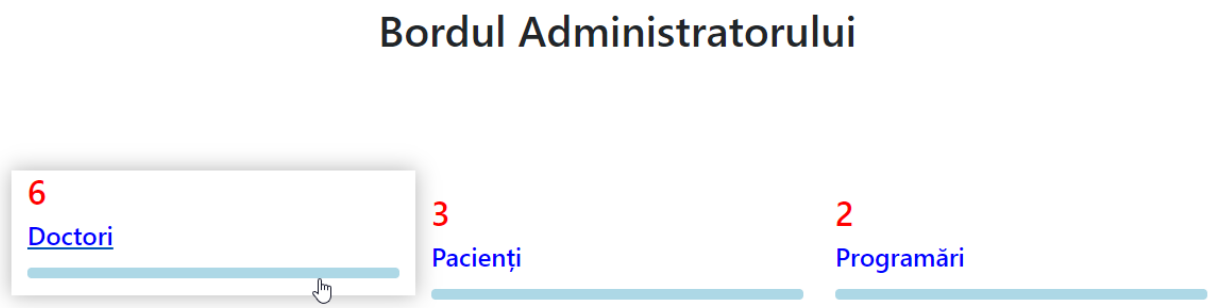

figura 91 Afisarea paginii index.html

Modul in care au fost implementate contoarele pentru doctori, pacienti si programari este descris in capitolul 3.6. Adaugarea view-urilor.

3.6. Adaugarea view-urilor

In Django, un *view* este o functie care primeste o cerere web si intoarce un raspuns web. Acest raspuns web poate fi continutul *HTML* al unei pagini, sa fie un raspuns de redirectionare catre alta pagina, poate fi o eroare de tip 404, un fisier *XML* sau multe altele.

In continuare voi discuta despre pagina *views.py* din cadrul aplicatiei de administrare a unui spital.

Incep prin a importa cateva functii si clase pe care le voi folosi in cadrul acestei pagini.

```
from django.shortcuts import render, redirect
from django.contrib.auth.models import User
from django.contrib.auth import authenticate, login, logout
from .models import Doctor, Patient, Appointment
```

figura 92 Importurile din views.py

Se poate observa ca din pachetul *Django.shortcuts,* un pachet care colecteaza functii de ajutor sic lase, importam doua fuctii si anume *render* si *redirect,* pe care le voi prezenta in subcapitolele ce urmeaza. De altfel, din pachetul *Django.contrib.auth* importam functiile *authenticate,* care va fi folosita pentru verificarea credentialelelor utilizatorului, *login* si *logout,* functii ajutatatoare pentru mecanisemele de autentificare si deconectare. In plus, importam si modelele *Doctor, Patient si Appointment* pe care le vom folosi in crearea *view-urilor* pentru listarea si adaugarea dcotorilor, pacientilor si a programarilor.

3.6.1. Functia render

Aceasta functie primeste doua argumente obligatorii, un *request,* sau o cerere, si un *template_name,* adica numele complet al unui sablon care va fi utilizat, sau o secventa de nume de sabloane, caz in care va fi folosit primul sablon existent. Aceasta functie mai poate primi si cateva alte argumente optionale si anume:

- *context:* Un dictionar de valori de adaugat contextului sablonului. In mod implicit, acesta este un dictionar gol. Daca o valoare din dictionar poate fi apelata, vizualizarea o va apela chiar inainte de redarea sablonului.
- *content_type:* Tipul *MIME* de utilizat pentru documentul rezultat. Setarea implicita este “text/html”. Abrevierea *MIME* vine de la “Multipupose Internet Mail Extension” si se refera la un tip de continut media sau de pe internet.
- *status:* Codul de stare pentru raspuns. Acesta este in mod implicit 200.
- *using:* Numele unui motor de sablon de utilizat pentru incarcarea sablonului.

Rolul acestei functii este de a combina un sablon dat cu un dictionar de context si intoarce un obiect de tip *HttpResponse* cu acel text randat.

3.6.2. Functia redirect

Aceasta functie intoarce un *HttpResponseRedirect* la adresa *URL* corespunzatoare argumentelor transmise. Aceste argumente pot fi:

- Un model: in acest caz functia get_absolute_url() a modelului va fi apelata
- Un nume de vizualizare
- O adresa *URL* absoluta sau relativa care va fi folosita pentru locatia de redirectionare.

3.6.3. View-urile About, Home si Contact

In continuare voi discuta despre modul in care au fost implementate view-urile aplicatiei. Voi incepe prin descrierea view-urilor About, Home si Contact deoarece acestea sunt implementate in mod similar, acestea folosindu-se de functia *render* descrisa anterior. Aceste functii primesc ca argument un *request,* o cerere, si intoarce pagina randata respectiva cererii.

```python
# Create your views here.
def About(request):
    return render(request, 'about.html')

def Home(request):
    return render(request, 'home.html')

def Contact(request):
    return render(request, 'contact.html')
```

figura 93 View-urile About, Home si Contact

3.6.4. View-ul Index

Ca si in cazul celorlalte view-uri, acesta primeste ca argument o cerere. Incep acest view prin a verifica daca utilizatorul din cererea primita are drepturi de administrator. Daca utilizatorul nu are aceste drepturi, atunci se va folosi functia *redirect* pentru a fi redirectionat catre pagina de login. In cazul in care utilizatorul are drepturile de administrator, se vor crea trei liste *doctors, patients* si *appointments.* Aceste liste vor stoca toate datele din baza de date pentru doctori, pacienti si programari. In continuare voi crea trei variabile, si pentru fiecare din aceasta variabila voi apela functia *Python len,* pentru a afla numarul dcotorilor, pacientilor si programarilor. In final, mai creez un dictionar cu aceste trei variabile pe care le voi pasa fuctiei *render* pentru a randa pagina *index.html* cu aceste valori.

```
def Index(request):
    if not (request.user.is_staff):
        return redirect('login')
    doctors = Doctor.objects.all()
    patients = Patient.objects.all()
    appointments = Appointment.objects.all()
    d = len(doctors)
    p = len(patients)
    a = len(appointments)

    entry = {'d': d, 'p': p, 'a': a}
    return render(request, 'index.html', entry)
```

figura 94 View-ul Index

3.6.5. View-ul Login

Incep acest *view* prin declararea unei variabile *error* pe care o voi lasa gooala de moment. Acest variabila imi va spune daca exista vreo eroare sau nu atunci cand utilizatorul incearca sa se autentifice in aplicatie. In continuare verific daca metoda primita in *request* este tip *POST*, caz in care voi mai declara doua variabile care vor stoca numele utilizatorului si parola pe care utilizatorul le-a introdus in aplicatie, apoi voi folosi metoda *authenticate* pentru a verifica aceste date. Continui cu o declaratie de tip *try – except,* prin care se va incerca apelarea functiei *login.* Daca totul a functionat in mod corect, atunci variabila *error* declarata la inceputul view-ului va stoca valoarea "*no",* insemnand ca nu exista erori. In caz contrar va stoca valoarea "*yes".* Pe baza acestei variabile se va crea un dictionar care va fi pasat functiei *render* pentru a randa pagina de *login.*

```
def Login(request):
    error = ""
    if(request.method == "POST"):
        u = request.POST['uname']
        p = request.POST['pwd']
        user = authenticate(username=u, password=p)
        try:
            if(user.is_staff):
                login(request, user)
                error = 'no'
            else:
                error = "yes"
        except:
            error = "yes"
    d = {'error': error}
    return render(request, 'login.html', d)
```

figura 95 View-ul Login

3.6.6. View-ul Logout_admin

Incep acest *view* prin a verifica daca utilizatorul are drepturi de administrator, in caz contrar va fi redirectionat catre pagina de Login, altfel se va apela functia *logout* si va fi redirectionat catre pagina de Login.

```
def Logout_admin(request):
    if not (request.user.is_staff):
        return redirect('login')

    logout(request)
    return redirect('admin_login')
```

figura 96 View-ul Logout_admin

3.6.7. View-urile View_Doctor, View_Patient si View_Appointment

Aceste trei view-uri sunt implementate similar, acestea au rolul de a lista obiectele din baza de date. Incep dezvoltarea acestor pagini prin verificarea drepturilor de administrator ale utilizatorului. Daca utilizatorul nu are drepturi de administrator aesta va fi redirectionat catre pagina de *login*. Pentru fiecare *view* se creeaza o variabila care va stoca datele din baza de date care vor fi adaugate intr-un dictionar pentru a randa paginile de *view_doctor.html, view_patient.html* si *view_appointment.html* cu datele din baza de date.

```
def View_Doctor(request):
    if not request.user.is_staff:
        return redirect('login')
    doc = Doctor.objects.all()
    d = {'doc': doc}
    return render(request, 'view_doctor.html', d)

def View_Patient(request):
    if not request.user.is_staff:
        return redirect('login')
    pat = Patient.objects.all()
    p = {'pat': pat}
    return render(request, 'view_patient.html', p)

def View_Appointment(request):
    if not request.user.is_staff:
        return redirect('login')
    apt = Appointment.objects.all()
    a = {'apt': apt}
    return render(request, 'view_appointment.html', a)
```

figura 97 View-urile View_Doctor, View_Patient si View_Appointment

3.7. Adaugarea url-urilor

Adaugarea url-urilor este necesara deoarece aceasta ii va spune lui Django ca odata introdus un url in browser, acesta va stii ce pagina trebuie afisata. Important de mentionat aici este faptul ca trebuie importate toate view-urile folosite. Astfel, din fisierul *views.py* prezentat in capitolul anterior vom importa toate functiile create. Apoi vom adauga in lista *urlpatterns* url-ul pe care dorim ca utilizatorul sa il acceseze pentru a afisa pagina respectiva. De asmenea mai este de notat ca pentru fiecare intrare in lista avem si un nume, acest lucru usureaza munca de dezvoltare fiind mai usor sa facem referire la o anumita pagina printr-un nume reprezentativ.

```
from django.urls import path
from .views import About, Home, Contact, Login, Logout_admin, Index, View_Doctor, Delete_Doctor,

urlpatterns = [
    path('', Home, name='home'),
    path('about/', About, name='about'),
    path('contact/', Contact, name='contact'),
    path('admin_login/', Login, name='admin_login'),
    path('admin_logout/', Logout_admin, name='admin_logout'),
    path('index/', Index, name='dashboard'),
    path('view_doctor/', View_Doctor, name='view_doctor'),
    path('add_doctor/', Add_Doctor, name='add_doctor'),
    path('delete_doctor(?P<int:pid>)/', Delete_Doctor, name='delete_doctor'),
    path('view_patient/', View_Patient, name='view_patient'),
    path('delete_patient(?P<int:pid>)/', Delete_Patient, name='delete_patient'),
    path('add_patient/', Add_Patient, name='add_patient'),
    path('view_appointment/', View_Appointment, name='view_appointment'),
    path('add_appointment', Add_Appointment, name='add_appointment'),
    path('delete_appointment(?P<int:pid>)/', Delete_Appointment, name='delete_appointment'),
]
```

figura 98 Adaugarea url-urilor

Important de notat mai este si faptul ca pentru url-urile de “*delete_patient”, “delete_doctor”* si “*delete_appointment”* avem acest “*(?P<int:pid>)”,* acesta fiind necesar deoarece aceste url-uri contin metode de tip POST, adica cereri ale utilizatorului de a trimite inormatii catre server. In cazul

aplicatiei noastre, prin apasarea butonului de stergere, ii cerem server-ului sa stearga intrarea cu id-ul respectiv.

4. Lista figurilor

5. Referinte

https://docs.djangoproject.com/en/4.0/

https://ro.wikipedia.org/wiki/Python

https://packagecontrol.io/

https://techvidvan.com/tutorials/django-project-structure-layout/

https://docs.djangoproject.com/en/4.0/ref/contrib/staticfiles/#module-django.contrib.staticfiles

https://docs.djangoproject.com/en/4.0/ref/settings/#std:setting-STATIC_URL

https://www.flaticon.com/

https://docs.djangoproject.com/en/4.0/topics/migrations/

https://docs.djangoproject.com/en/1.8/intro/tutorial02/

https://docs.djangoproject.com/en/4.0/topics/http/views/

https://docs.djangoproject.com/en/4.0/topics/http/shortcuts/

https://docs.djangoproject.com/en/4.0/topics/auth/default/

6. Cuprins