Ministerul Educației, Culturii şi Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare Informatică şi Microelectronică

Departamentul Inginerie Software şi Automatică

Disciplina: Tehnologii Web

Lucrarea de Laborator nr.3

Tema: Modele de proiectare. Pattern BusinessLogic

A efectuat: Balaur Dorina gr. TI-184

A verificat: asist. univ. Cristian Rusu

Chişinău 2020

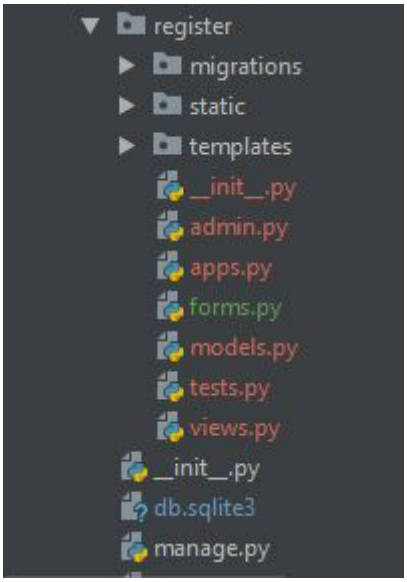
Scopul lucrarii: Asigurarea functionarii backend-ului si a formularelor de inregistrare sign up, log in si log out si a formularului de contact.

**Am utilizat Django si nu Visual Studio intrucat cunosc mai bine acest limbaj

Ce este BusinessLogic?

BusinessLogic este partea din program care determina cum este creata, schimbata si salvata data.

Pas 1. Am creat o noua aplicatie in interiorul proiectului cu numele Register unde se vor afla ulterior paginile de inregistrare si logare a utilizatorilor. Directoriul Templates e responsabil de fisierele .html iar Static este pentru .css.



Pas 2. In view.py Adaugam urmatorul cod care va forma un formular de inregistrare

```python
from django.shortcuts import render,redirect
from .forms import RegisterForm


def register(response):
    if response.method=="POST":
        form= RegisterForm(response.POST)
        if form.is_valid():
            form.save()

        return redirect("/header")
    else:
        form = RegisterForm()
    return render(response,'signup.html', {"form": form})
```

Pas 3. Pentru a face legatura acestui nou app cu celelalte templaturi fost nevoie ca sa le aducem o referinte in urls.py din app-ul de baza bubbleyou. In path facem legatura.

```
from django.contrib import admin
from django.contrib.auth import views as auth_views
from django.urls import include, path
from . import views
from register import views as v
```

```
path("signup/", v.register, name="register"),
```

Pas 4. In folderul Register adaugam un file numit forms.py care va contine urmatorul cod.

```
from django.contrib.auth import login, authenticate
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User
class RegisterForm(UserCreationForm):
    email = forms.EmailField()


    class Meta:
        model = User
        fields=["username","email","password1","password2"]
```

Codul contine elementel care vor fi cerute de la user si anume username, email, parola si confirmarea parolei.

Pas 5. In Register adaugam un directoriu Templates iar in el adaugam un fisier signup.html unde adaugam codul acestuia:

```
{%load crispy_forms_tags%}
<div class="container">
  <div class="row">
    <div class= "col-md-6 order-md-2 " >
      <h2>Sign Up</h2>
      <form method="POST" class="form-group" >
          {%csrf_token%}
          {{form|crispy}}
        <button class="btn btn-warning">Submit</button>
      </form>
    </div>
    <div class="col-md-6 order-md-1" >
      <img src="https://www.ivyboarding.com/w3template/w3images/cherries.jpg" height="729">
    </div>
  </div>
</div>
```

```
{%load crispy_forms_tags%}
```

Aceatsa linie de cod reprezinta adaugarea unei noi biblioteci care va formata formularul intr-un mod mai dragut. Se datoreaza adaugarea acestei biblioteci in settings.py in modul urmator:

```
INSTALLED_APPS = [
    'poll.apps.PollConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    "crispy_forms",
    'register.apps.RegisterConfig',
]
```

si la sfarsitul codului adaugam asta:

```
CRISPY_TEMPLATE_PACK="bootstrap4"
```

Acestea acum activeaza `{{form|crispy}}` care permite acum sa-i folosim biblioteca.

Pas 6. Acum e momentul pentru paginile de logare pentru care facem cam acelasi lucru:

```
urlpatterns = [

    path('admin/', admin.site.urls),
    path('', views.header, name='header'),
    path('header/', views.home, name='home'),
    path('contacts/', views.contacts, name='contacts'),
    path('posts/', views.posts, name='posts'),
    path("signup/", v.register, name="register"),
    path('', include("django.contrib.auth.urls")),
]
```

In urls.py adaugam randul subliniat care va crea un sistem de autorizare. Django face asta singur deci nu e nevoie de alte coduri adaugatoare.

Pas 7. In Register-> Templates adaugam un alt directoriu numit registration in care vom adauga file-rile login.html si logout.html cu aceleasi moderari de cod ca la signup.html:

```
{%load crispy_forms_tags%}
<div class="container " >
    <div class="row">
        <div class="col-md-6">
            <h2>Log in</h2>
      <form method="POST" class="form-group" >
          {%csrf_token%}
          {{form|crispy}}
        <button class="btn btn-success">Submit</button>
      </form>
        </div>
        <div class="col-md-6">
            <img src="https://www.w3schools.com/w3images/rocks.jpg" alt="img" height:50%>

        </div>
    </div>

</div>
```

Pas 8. In setting.py adaugam urmatoarele linii de cod la sfarsitul codului existent deja:

```
LOGIN_REDIRECT_URL = "/header"
LOGOUT_REDIRECT_URL = "/logout"
```

Acestea sunt responsabile de directionarea userului in dependenta de ce acesta alege, fie log in care il va duce pe pagina principala, fie logout care il va redirectiona spre o pagina creata aparte logout.html.

Pas 9. Pentru crearea unui admin introducem in terminal urmatorul cod:

```
python manage.py createsuperuser
```

iar in continuare se vor cere username si parola de introdus de doua ori.

Pas 10. Odata ce acestea sunt facute, putem da manage.py runserver iar acesta ne va genera un link care ne va directiona spre site-ul nostru. Adaugand /admin, ar trebui sa fim directionati spre baza de date care contine datele introduse si pe a celorlati useri care se vor conecta.

Pas 11. In directoriul proiectului, in cazul meu bubbleyou, adaugam un fisier forms.py cu urmatorul cod:

```python
from django import forms

def should_be_empty(value):
    if value:
        raise forms.ValidationError('Field is not empty')

class ContactForm(forms.Form):
    name=forms.CharField(max_length=80)
    message=forms.CharField(widget=forms.Textarea)
    email=forms.EmailField()
    forcefield=forms.CharField(
        required=False,widget=forms.HiddenInput, label="Leave empty", validators=[should_be_empty])
```

Acest cod va forma campurile formularului urmand logica: nume, mesaj, email.

Pas 12. In views.py adaugam un nou path pentru contacts care va directiona utilizatorul spre pagina html corespunzatoare formularului de contacte. `path('contacts/', views.contacts, name='contacts'),`

Pas 13. In pagina de html home, completam href=”/contacts”

Pas 14. Realizam un fisier cu numele contacts.html in directoriul Templates si introducem urmatorul cod care va realiza formularul in formatul html necesar:

```
<form action="" method="POST">
    {% csrf_token %}
    {%for hidden in form.hidden_fields%}
    {{field}}
    {% endfor %}

    {% for field in form.visible_fields%}
    <div class="form-group">
        {{field.errors}}
        <label>{{field.label_tag}}</label>
        <p>{{field}}</p>
    </div>
    {% endfor %}
```

Pas 15. In views.py adaugam urmatoarea functie:

```python
def contacts(request):
    form=ContactForm()
    if request.method=='POST':
        form=ContactForm(request.POST)
        if form.is_valid():
            subject=f'Message from {form.cleaned_data[" name"]}'
            message=form.cleaned_data["message"]
            sender=form.cleaned_data["email"]
            recipients=['balaurdorina@gmail.com']
            try:
                send_mail(subject, message,sender,recipients,fail_silently=False)
            except BadHeaderError:
                return HttpResponse('Invalidule')
            return HttpResponse('woohoo you did it...')

    return render(request, 'contacts.html',{'form':form})
```

Acest cod inregistreaza data necesara in locul campurilor subject, message, sender, recipient si verifica sa fie introdusa corect si corespunzator. In caz ca campurile nu sunt valide, utilizatorul va primi un mesaj care-l va anunta de acest lucru, iar in caz ca este valida completarea, acesta va primi, din nou, un mesaj. Deasemenea este adaugata o adresa de email spre care va pleca mesajul scris in formular si directiunea formularului aflata la contacts.html.

Pas 15. Pentru ca procesul transmiterii sa funtioneze, am folosit SendGrid - o platforma de comunicare si tranzictie de email-uri. Pentru a ne conecta a fost nevoie de instalarea acesteia prin pip install django-sendgrid. Si adaugarea in setting.py urmatorul cod:

```python
EMAIL_BACKEND="sendgrid_backend.SendgridBackend"
SENDGRID_API_KEY='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'
SENDGRID_SANDBOX_MODE_IN_DEBUG=False
```

de unde am format o cheie API si i-am introdus linkul acesteia aici.

Rezultatul:

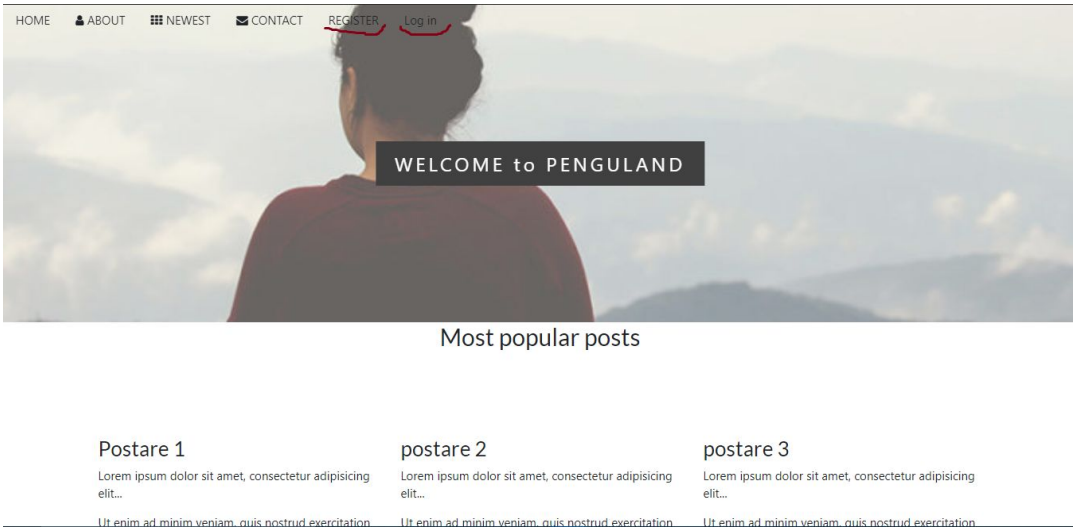


fig 1.Interfata site-ului cu butoanele Register si Log in

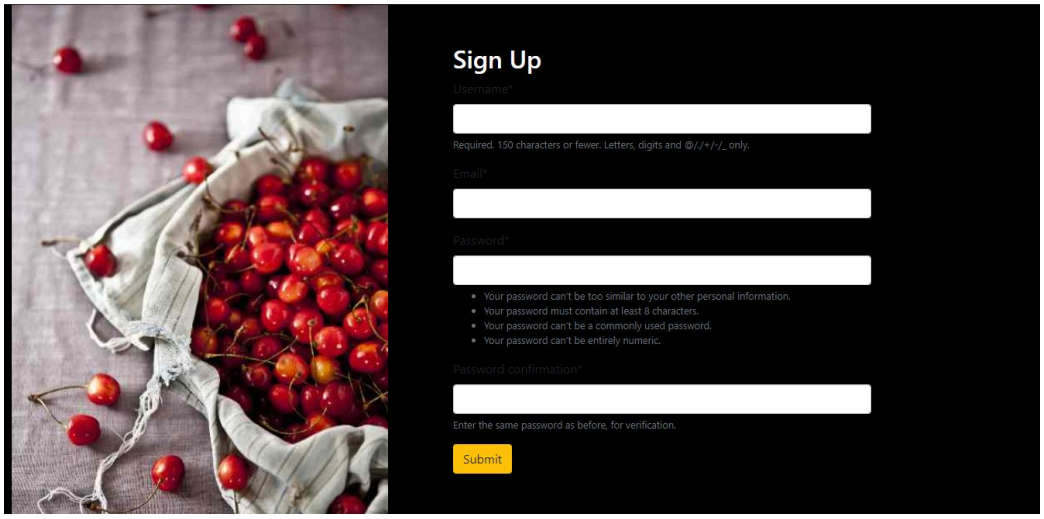


fig 2. Interfata formularului de inregistrare

fig 3. Interfata formularului de logare

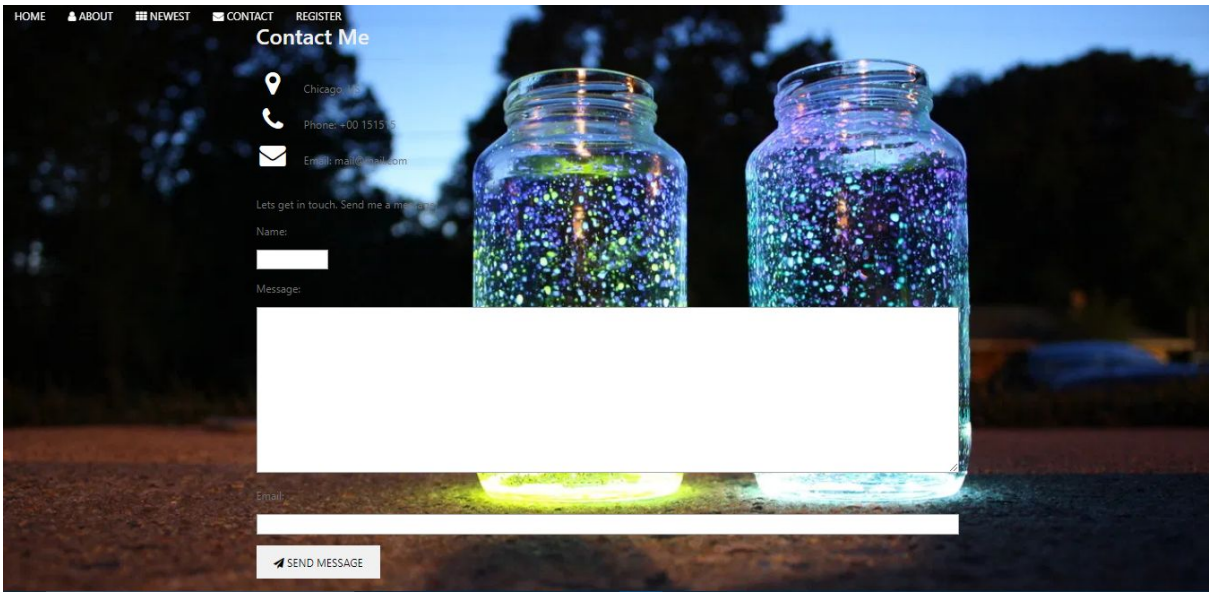


fig.4 Interfata formularului de contactare

Concluzie: In urma efectuarii acestei lucrari de laborator am efectuat backendul formularelor de inregistrare si logare si dezlogare si am facut functional bara de menu. Am insusit cum acestea depind una de alta si ce legatura se formeaza cand le legam cu aplicatia initiala care contine paginile home. Deasemenea am delimitat utilizatorii admin si userii obisnuiti pe care in urmatorul laborator ii vom introduce si afisa in baza de date creata in Django la fel si expedierea si salvarea mesajelor trimise prin contact form.

link la Github-->https://github.com/Dorinautm/laboratoareTW