Almaty Management University

Site development

Discipline “Basics of IS”

The Student: Saken Mukhtarbek

Almaty

2019

1. **Introduction**

Overview

This report depicts the outcomes of the work done in development of “Fan Site”. It is a part of the Basics of IS course, according to which, student should show on the example what he learnt. Student had a choice to choose any topic and use any tools wanted. On this project I will develop a website with primarily options of creating, updating and deleting posts with some features that can be included. The website will run locally for the testing, so no expanses will be during the work. All the used instruments are free. This work will be done by a student without the interference of university professors. The subject was chosen so that the student will be able to show his skills and knowledge in full-stack development. Overall a good-looking and functioning website could be a potentially good project. A timeline of the project is provided to have a vision of how things were built.(figure 1.)

Background and Motivation

As technology advances internet is becoming ever popular. Website has become an essential part of any organization and social group. Often website is used as a source of information. However, most platforms provide information about almost everything. That is why it is hard to track the interesting things with all the advertisements and unnecessary news. Project is done within the university so it is absolutely non-profit oriented. The main aim is to see how well student knows the chosen topic.

Objectives

The objectives of the project are below:

1. Provide a full-functioning website.
2. Proper time management.
3. Provide platform to write about interesting.
4. Website should be relative and functioning for both – PC and Phone users.

Resources

This project will be planned with following resources:

**Time:**

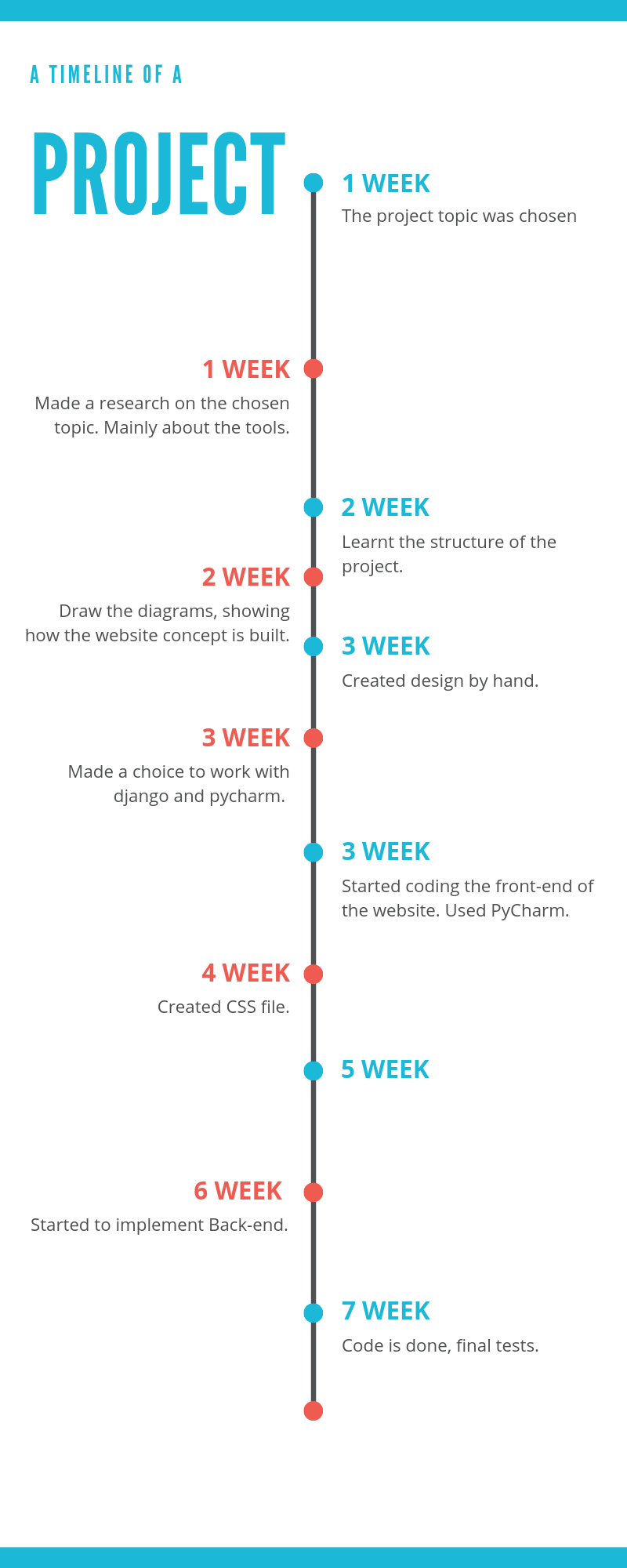
8 weeks to finish the whole project. Deadline is 27th of April 2019.

**Budget:**

No budget.

**Human Resources:**

All the professors of the IT department of ALMAU.

 figure 1. Timeline

Methodology

To implement the above goals, following methodology should be used:

1. Specifying the friendly, comfortable designs for non-professional users.

2. Implement the action functions.

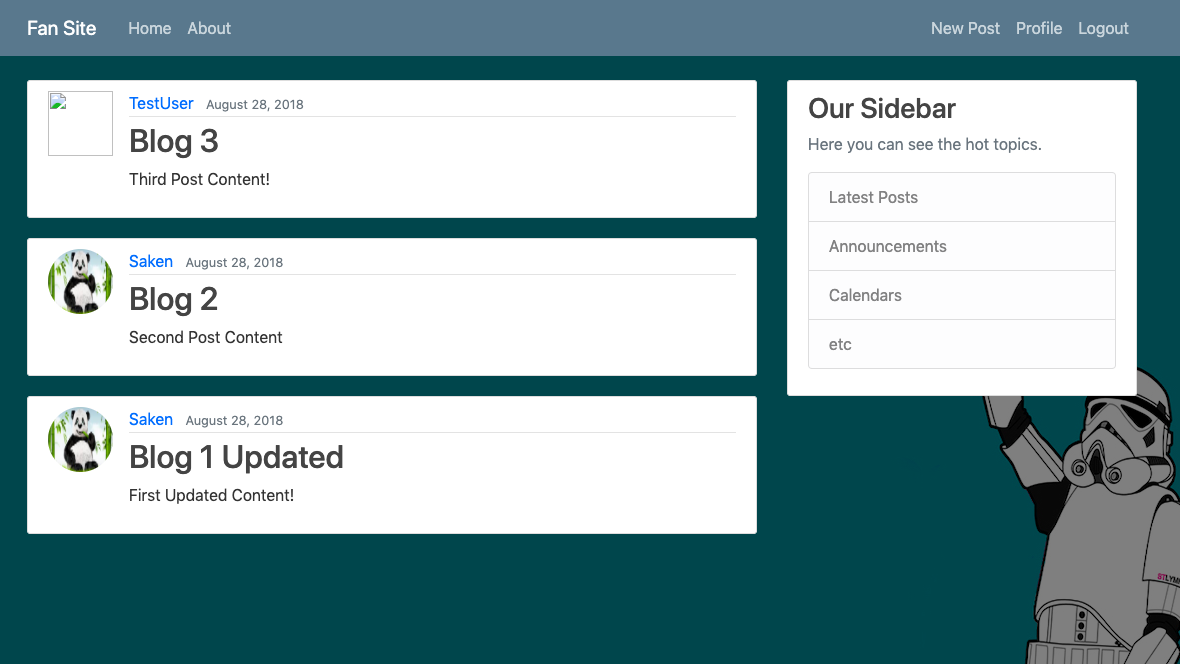
3. Make the action functions possible only for the authors.

4. Author status available only after logging in.

1. **Tool Description**

**User Interface**

The interface is very user-friendly and intuitive. It uses graphical-user interface, so any user will be capable of using it. Interface implemented in html, css.(figure 2.) Navigation bar is written on the top corner of the website to make it comfortable to switch between pages. It is converted to a dropdown menu for small screens, so the size of the screen will not be a problem. All the forms are written so they are easy to manipulate with. Colors are not to bright and too dark, so it will not hurt user’s eyesight.

figure 2. Website frontpage

**Features**

1. *Writing access*
   1. Provide a function of writing posts
   2. Anybody can write posts
2. Account access
   1. People will have their own accounts where all the information will be storaged in Django ORM.
   2. It would be impossible to create two identical accounts, so errors are minimized.
3. Security for online registrations
   1. All the data will be transformed using tokens, so nobody will be able to see the private information of users, including passwords.
4. Preference
   1. It is possible to choose the interesting post and read it in a full screen
5. Sort
   1. All the posts are sorted from the newest at the top to the oldest in the bottom.
6. Editing
   1. While being logged in, user can update his posts or even delete. However, the date of the post will be changed to the date of the update automatically.

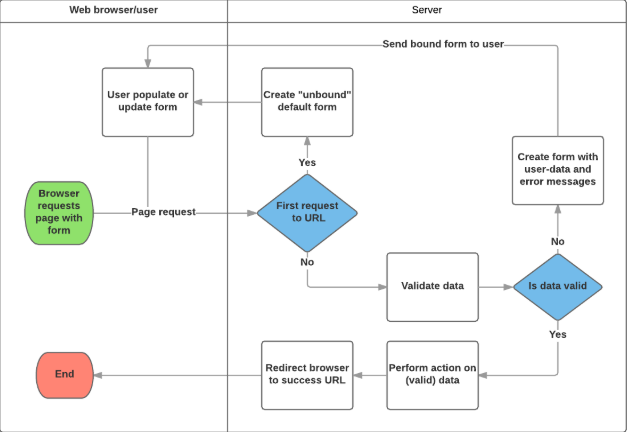


Figure 3. Process of using Forms in Django.

**Use Case Diagram**

My project is a post-writing-reading website. The main principle is to create account of fan pages and then collect posts by the author. (see figure 4.)

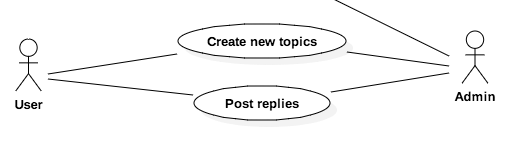


Figure 4. Use Case Diagram

From the Use Case Diagram we may draw the entities and their relations. Using this figure 4, we can assume that our project will have only 3 models: Topics, Posts, Users. Now, it is also important to understand how this models relate to each other. In Topic we want to see what posts are in there. Or better saying, to which topics every post is linked. Also, we will need to have the place in Topic and Post to know who wrote them. The topic model will have fields “subject” and “last update”. They will depict the actual name of the topic and the date of last update.

The Post field will have “message” which will contain the description or any text written. “created at” date will be used to sort posts, while “updated at” will be used to see if the post was edited and when.

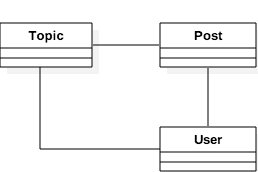


Figure 5. Class Diagram(draft)

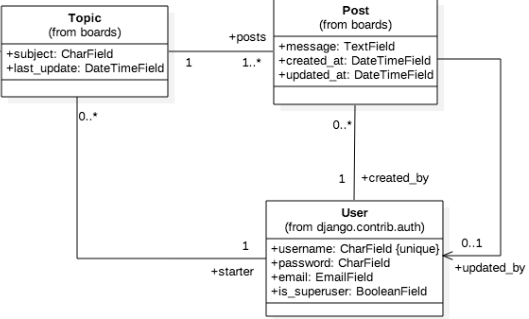


Figure 6. Class Diagram

Class of the User will be automatically created by Django, so I just wrote basic info that will be stored there.

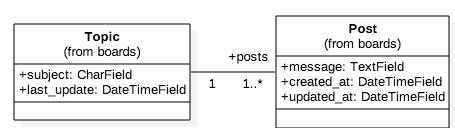


Figure 7. Topic to Post relationship

Topic should have at least one post, while every post will have only one topic.

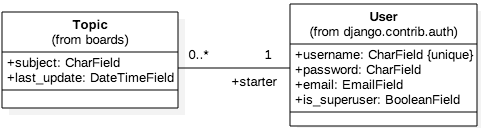


Figure 8. Topic to User relationship

The Topic may have only one User. A User may have a lot or none of Topics.

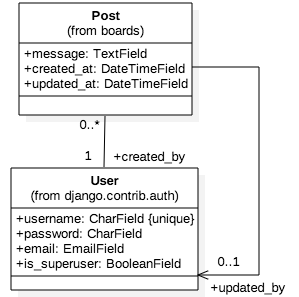


Figure 9. Post to User relationship.

The same with Posts. Each post have only one User, while User may have zero or more posts. Secondary relationship is direct, it is saying that posts could be updated by User. It could be edited(state 1) or could stay null (state 0).

**Wireframes**

To see how everything comes up together I like to show some screens of the final product.

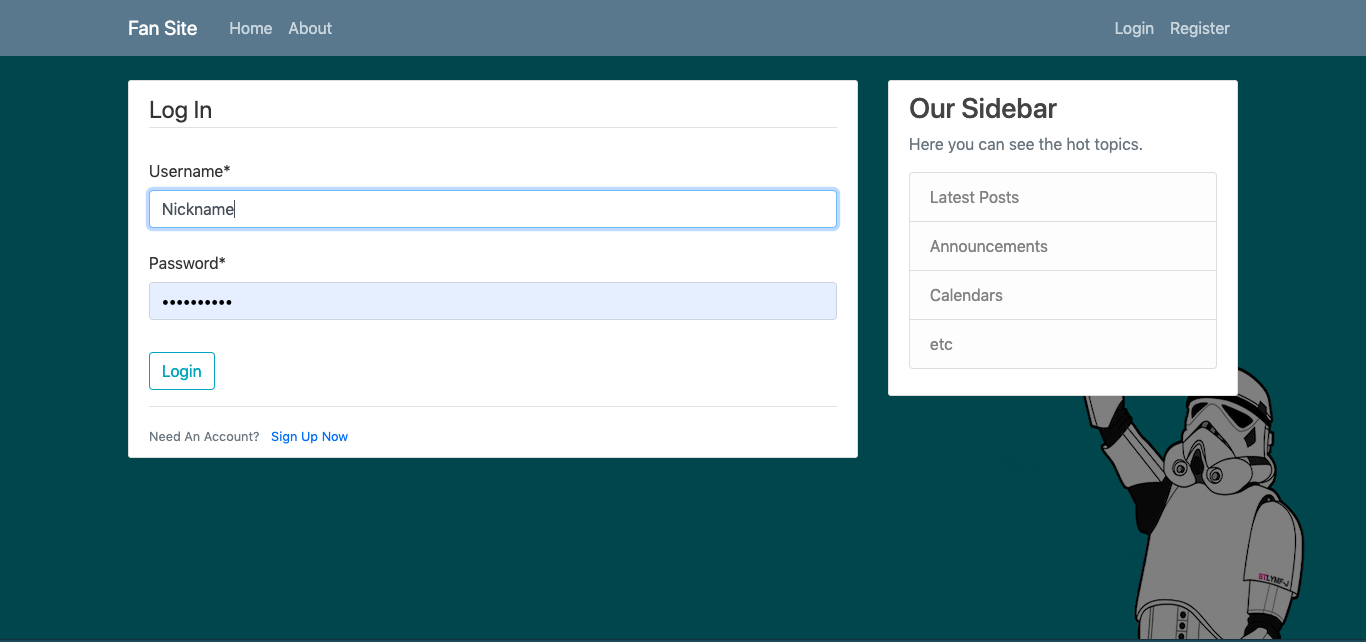


Figure 10. Log In Page.

Here we can see the LogIn Form that is asking for a username and password. It also has a hyperlink saying “Sign Up Now” in case if user don’t have an account.

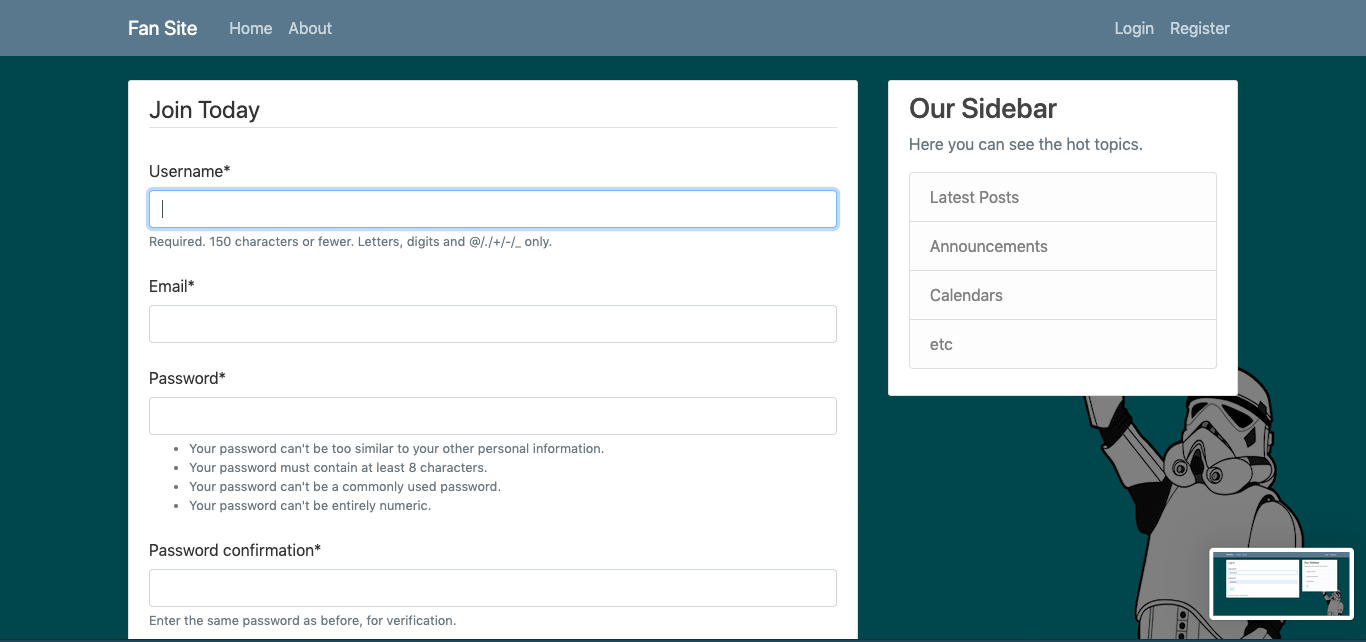


Figure 11. Registration Form

On figure 11 we see the registration form, where new users will create their account. The password is asking to be at least 8 characters and be strong.

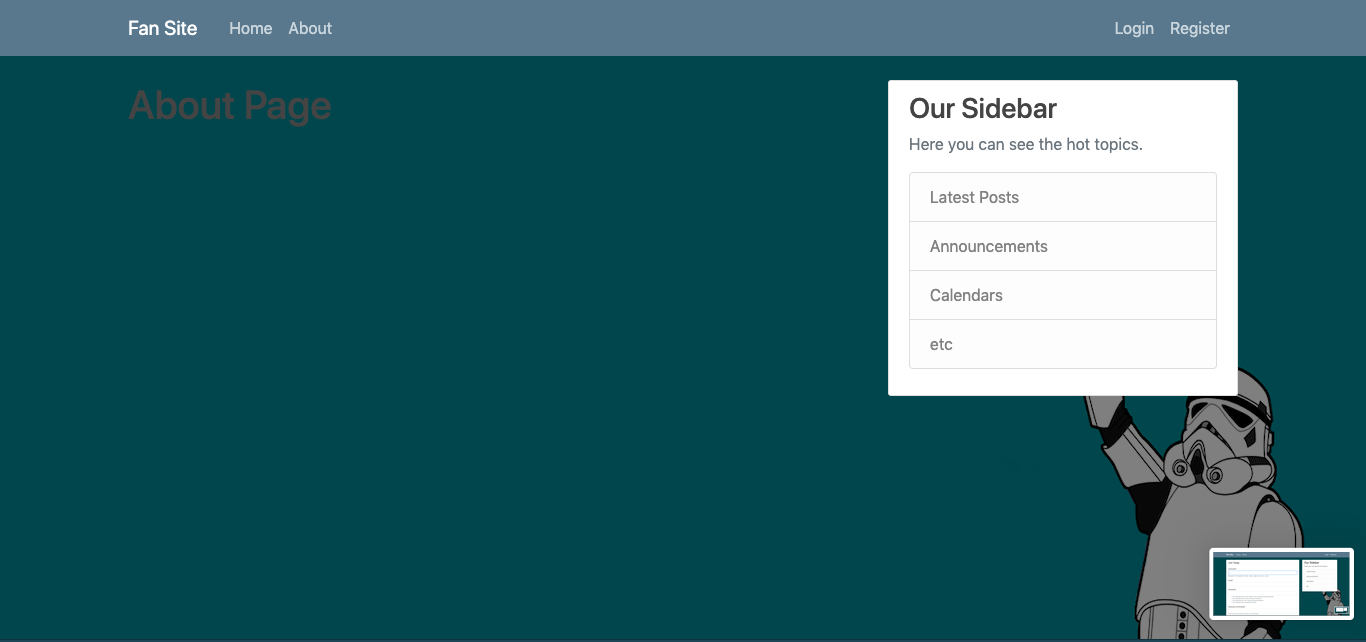


Figure 12. About Page

The about page will contain the contact information of the creator/me. It is not necessary to fill it on developing stages.

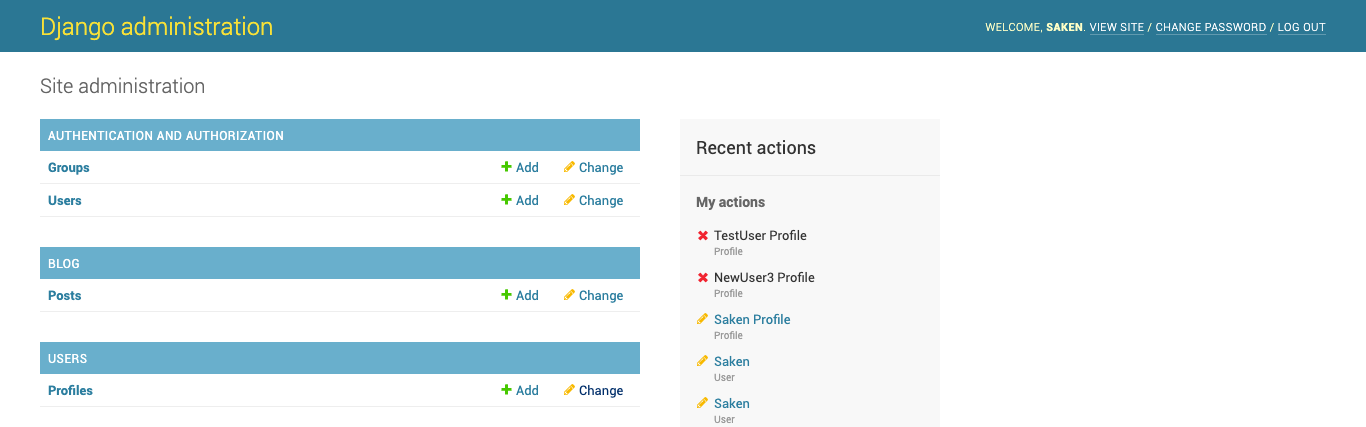


Figure 13. Admin Page

Admin page(figure 13.) will provide a superusers an opportunity to stalk all actions, users and all entities within the website.

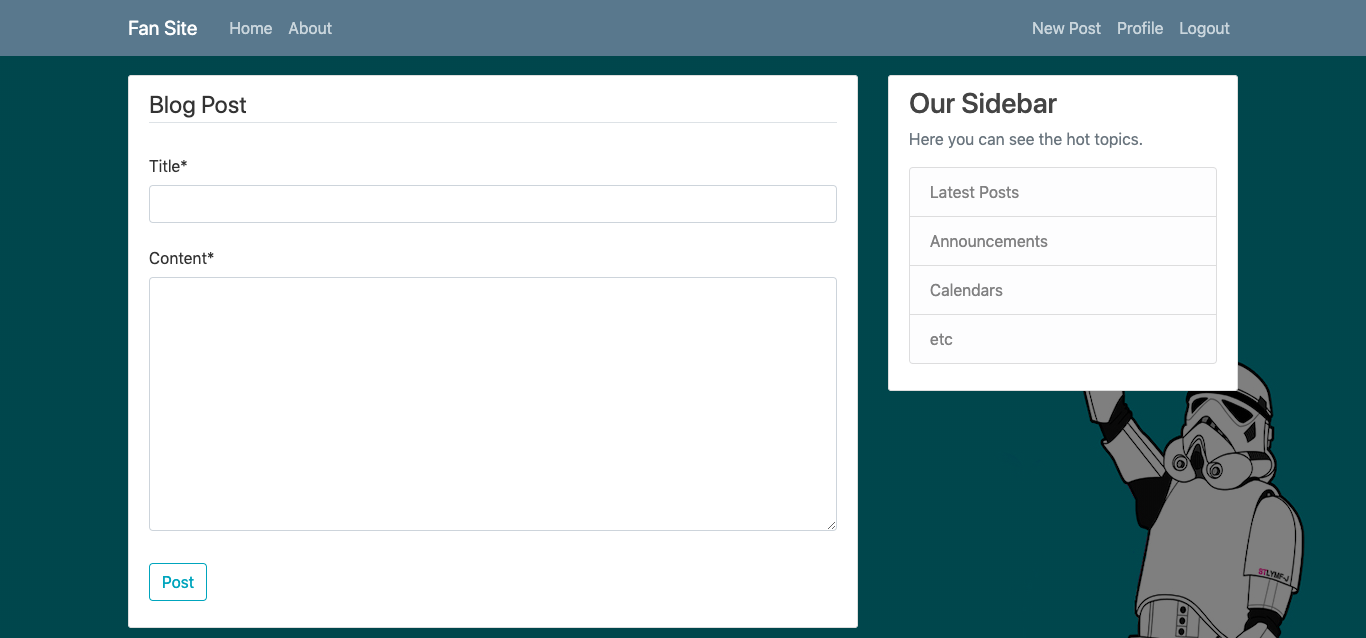


Figure 14. New Post Page.

The ability to create a post will occur only when user will log in to the system. The link will be on the right navigation bar, saying “New Post”. Here user will be able to write the title and the content of the post.

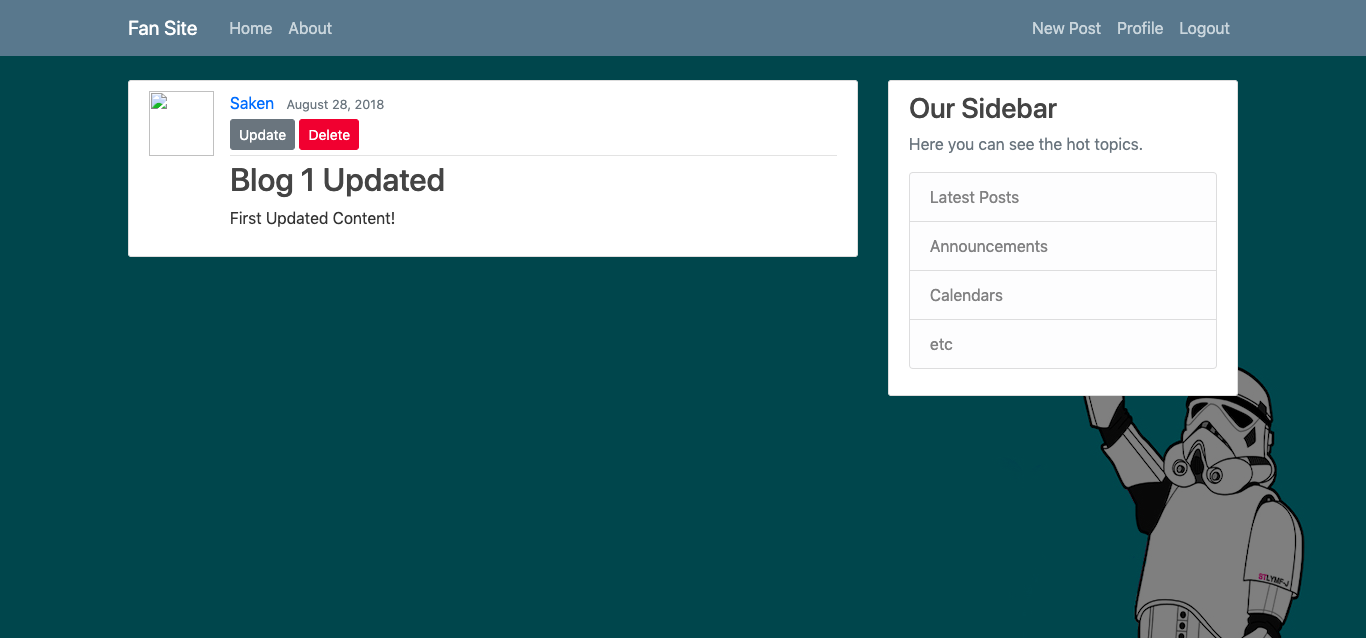


Figure 15. Post Editing.

If user will click on some post that he wrote, there will be two additional buttons, saying “Delete” and “Update”. They are created for the editing purposes. Only the author could see the buttons and edit post.

1. **Conclusion.**

The project us fully working and code is perfectly executed. However, there a lot of developments that came to mind during coding the website, that could improve the interface. If I had more time, approximately a one month more, there would be much functions. The main difficulty was in learning the Django while working on code. I had no experience working with it before, so it took some time to master some skills. There are a lot of things that could be done to the current website to improve it, however main objectives are met. The website is fully-working, user friendly and we can manipulate with entities in our front pages.

**Appendix 1.**

Here is the raw code of the projects front-end and back end. I separated it by files names to make it easier to follow.

The “Blog” folder:

0001\_initial.py

class Migration(migrations.Migration):  
  
 initial = True  
  
 dependencies = [  
 migrations.swappable\_dependency(settings.AUTH\_USER\_MODEL),  
 ]  
  
 operations = [  
 migrations.CreateModel(  
 name='Post',  
 fields=[  
 ('id', models.AutoField(auto\_created=True, primary\_key=True, serialize=False, verbose\_name='ID')),  
 ('title', models.CharField(max\_length=100)),  
 ('content', models.TextField()),  
 ('date\_posted', models.DateTimeField(default=django.utils.timezone.now)),  
 ('author', models.ForeignKey(on\_delete=django.db.models.deletion.CASCADE, to=settings.AUTH\_USER\_MODEL)),  
 ],  
 ),  
 ]

main.css

body {  
 background: #fafafa;  
 color: #333333;  
 margin-top: 5rem;  
}  
  
h1, h2, h3, h4, h5, h6 {  
 color: #444444;  
}  
  
ul {  
 margin: 0;  
}  
  
.bg-steel {  
 background-color: #5f788a;  
}  
  
.site-header .navbar-nav .nav-link {  
 color: #cbd5db;  
}  
  
.site-header .navbar-nav .nav-link:hover {  
 color: #ffffff;  
}  
  
.site-header .navbar-nav .nav-link.active {  
 font-weight: 500;  
}  
  
.content-section {  
 background: #ffffff;  
 padding: 10px 20px;  
 border: 1px solid #dddddd;  
 border-radius: 3px;  
 margin-bottom: 20px;  
}  
  
.article-title {  
 color: #444444;  
}  
  
a.article-title:hover {  
 color: #428bca;  
 text-decoration: none;  
}  
  
.article-content {  
 white-space: pre-line;  
}  
  
.article-img {  
 height: 65px;  
 width: 65px;  
 margin-right: 16px;  
}  
  
.article-metadata {  
 padding-bottom: 1px;  
 margin-bottom: 4px;  
 border-bottom: 1px solid #e3e3e3  
}  
  
.article-metadata a:hover {  
 color: #333;  
 text-decoration: none;  
}  
  
.article-svg {  
 width: 25px;  
 height: 25px;  
 vertical-align: middle;  
}  
  
.account-img {  
 height: 125px;  
 width: 125px;  
 margin-right: 20px;  
 margin-bottom: 16px;  
}  
  
.account-heading {  
 font-size: 2.5rem;  
}

about.html

{% extends "blog/base.html" %}  
{% block content %}  
 <h1>About Page</h1>  
  
{% endblock content %}

Base.html

{% load static %}  
<!DOCTYPE html>  
<html>  
<head>  
  
 <!-- Required meta tags -->  
 <meta charset="utf-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  
  
 <!-- Bootstrap CSS -->  
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">  
  
 <link rel="stylesheet" type="text/css" href="{% static 'blog/main.css' %}">  
  
 {% if title %}  
 <title>Fan Site - {{ title }}</title>  
 {% else %}  
 <title>Fan Site</title>  
 {% endif %}  
</head>  
<body>  
 <header class="site-header">  
 <nav class="navbar navbar-expand-md navbar-dark bg-steel fixed-top">  
 <div class="container">  
 <a class="navbar-brand mr-4" href="{% url 'blog-home' %}">Fan Site</a>  
 <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarToggle" aria-controls="navbarToggle" aria-expanded="false" aria-label="Toggle navigation">  
 <span class="navbar-toggler-icon"></span>  
 </button>  
 <div class="collapse navbar-collapse" id="navbarToggle">  
 <div class="navbar-nav mr-auto">  
 <a class="nav-item nav-link" href="{% url 'blog-home' %}">Home</a>  
 <a class="nav-item nav-link" href="{% url 'blog-about' %}">About</a>  
 </div>  
 <!-- Navbar Right Side -->  
 <div class="navbar-nav">  
 {% if user.is\_authenticated %}  
 <a class="nav-item nav-link" href="{% url 'post-create' %}">New Post</a>  
 <a class="nav-item nav-link" href="{% url 'profile' %}">Profile</a>  
 <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>  
 {% else %}  
 <a class="nav-item nav-link" href="{% url 'login' %}">Login</a>  
 <a class="nav-item nav-link" href="{% url 'register' %}">Register</a>  
 {% endif %}  
 </div>  
 </div>  
 </div>  
 </nav>  
 </header>  
 <main role="main" class="container">  
 <div class="row">  
 <div class="col-md-8">  
 {% if messages %}  
 {% for message in messages %}  
 <div class="alert alert-{{ message.tags }}">  
 {{ message }}  
 </div>  
 {% endfor %}  
 {% endif %}  
 {% block content %}{% endblock %}  
 </div>  
 <div class="col-md-4">  
 <div class="content-section">  
 <h3>Our Sidebar</h3>  
 <p class='text-muted'>Here you can see the hot topics.  
 <ul class="list-group">  
 <li class="list-group-item list-group-item-light">Latest Posts</li>  
 <li class="list-group-item list-group-item-light">Announcements</li>  
 <li class="list-group-item list-group-item-light">Calendars</li>  
 <li class="list-group-item list-group-item-light">etc</li>  
 </ul>  
 </p>  
 </div>  
 </div>  
 </div>  
 </main>  
  
 <!-- Optional JavaScript -->  
 <!-- jQuery first, then Popper.js, then Bootstrap JS -->  
 <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>  
 <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>  
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>  
</body>  
</html>

Home.html

{% extends "blog/base.html" %}  
{% block content %}  
 {% for post in posts %}  
 <article class="media content-section">  
 <img class="rounded-circle article-img" src="{{ post.author.profile.image.url }}">  
 <div class="media-body">  
 <div class="article-metadata">  
 <a class="mr-2" href="#">{{ post.author }}</a>  
 <small class="text-muted">{{ post.date\_posted|date:"F d, Y" }}</small>  
 </div>  
 <h2><a class="article-title" href="{% url 'post-detail' post.id %}">{{ post.title }}</a></h2>  
 <p class="article-content">{{ post.content }}</p>  
 </div>  
 </article>  
 {% endfor %}  
{% endblock content %}

Post\_confirm\_delete.html

{% extends "blog/base.html" %}  
{% block content %}  
 <div class="content-section">  
 <form method="POST">  
 {% csrf\_token %}  
 <fieldset class="form-group">  
 <legend class="border-bottom mb-4">Delete Post</legend>  
 <h2>Are you sure you want to delete post "{{ object.title }}</h2>  
 </fieldset>  
 <div class="form-group">  
 <button class="btn btn-outline-danger" type="submit">Delete!</button>  
 <a class="btn btn-outline-secondary" href="{% url 'post-detail' object.id %}">Cancel</a>  
 </div>  
 </form>  
 </div>  
{% endblock content %}

Post\_detail.html

{% extends "blog/base.html" %}  
{% block content %}  
 <article class="media content-section">  
 <img class="rounded-circle article-img" src="{{ object.author.profile }}">  
 <div class="media-body">  
 <div class="article-metadata">  
 <a class="mr-2" href="#">{{ object.author }}</a>  
 <small class="text-muted">{{ object.date\_posted|date:"F d, Y" }}</small>  
 {% if object.author == user %}  
 <div>  
 <a class="btn btn-secondary btn-sm mt-1 mb-1" href="{% url 'post-update' object.id %}">Update</a>  
 <a class="btn btn-danger btn-sm mt-1 mb-1" href="{% url 'post-delete' object.id %}">Delete</a>  
 </div>  
 {% endif %}  
  
 </div>  
 <h2><a class="article-title" href="{% url 'post-detail' post.id %}">{{ object.title }}</a></h2>  
  
 <p class="article-content">{{ object.content }}</p>  
 </div>  
 </article>  
{% endblock content %}

Post\_form.html

{% extends "blog/base.html" %}  
{% load crispy\_forms\_tags %}  
{% block content %}  
 <div class="content-section">  
 <form method="POST">  
 {% csrf\_token %}  
 <fieldset class="form-group">  
 <legend class="border-bottom mb-4">Blog Post</legend>  
 {{ form|crispy }}  
 </fieldset>  
 <div class="form-group">  
 <button class="btn btn-outline-info" type="submit">Post</button>  
 </div>  
 </form>  
 </div>  
{% endblock content %}

Admin.py

from django.contrib import admin  
from .models import Post  
  
admin.site.register(Post)

apps.py

from django.apps import AppConfig  
  
  
class BlogConfig(AppConfig):  
 name = 'blog'

models.py

from django.db import models  
from django.utils import timezone  
from django.contrib.auth.models import User  
from django.urls import reverse  
  
class Post(models.Model):  
 title = models.CharField(max\_length=100)  
 content = models.TextField()  
 date\_posted = models.DateTimeField(default=timezone.now)  
 author = models.ForeignKey(User, on\_delete=models.CASCADE)  
  
 def \_\_str\_\_(self):  
 return self.title  
  
  
 def get\_absolute\_url(self):  
 return reverse('post-detail', kwargs={'pk': self.pk})

urls.py

from django.urls import path  
from .views import PostListView, PostDetailView, PostCreateView, PostUpdateView, PostDeleteView  
from . import views  
  
urlpatterns = [  
 path('', PostListView.as\_view(), name='blog-home'),  
 path('post/<int:pk>/', PostDetailView.as\_view(), name='post-detail'),  
 path('post/new/', PostCreateView.as\_view(), name='post-create'),  
 path('post/<int:pk>/update/', PostUpdateView.as\_view(), name='post-update'),  
 path('post/<int:pk>/delete/', PostDeleteView.as\_view(), name='post-delete'),  
 path('about/', views.about, name='blog-about'),  
]

views.py

from django.shortcuts import render  
from django.contrib.auth.mixins import LoginRequiredMixin, UserPassesTestMixin  
from django.views.generic import(  
ListView,  
DetailView,  
CreateView,  
UpdateView,  
DeleteView  
)  
from .models import Post  
  
  
def home(request):  
 context = {  
 'posts': Post.objects.all()  
 }  
 return render(request, 'blog/home.html', context)  
  
  
class PostListView(ListView):  
 model = Post  
 template\_name = 'blog/home.html'  
 context\_object\_name = 'posts'  
 ordering = ['-date\_posted']  
  
class PostDetailView(DetailView):  
 model = Post  
  
class PostCreateView(LoginRequiredMixin,CreateView):  
 model = Post  
 fields = ['title', 'content']  
  
 def form\_valid(self, form):  
 form.instance.author = self.request.user  
 return super().form\_valid(form)  
  
  
  
class PostUpdateView(LoginRequiredMixin, UserPassesTestMixin, UpdateView):  
 model = Post  
 fields = ['title', 'content']  
  
 def form\_valid(self, form):  
 form.instance.author = self.request.user  
 return super().form\_valid(form)  
  
 def test\_func(self):  
 post = self.get\_object()  
 if self.request.user == post.author:  
 return True  
 return False  
  
class PostDeleteView(LoginRequiredMixin, UserPassesTestMixin, DeleteView):  
 model = Post  
 success\_url = '/'  
 def test\_func(self):  
 post = self.get\_object()  
 if self.request.user == post.author:  
 return True  
 return False  
  
def about(request):  
 return render(request, 'blog/about.html', {'title': 'About'})

the “Django project” folder:

settings.py

import os  
  
# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)  
BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
  
  
# Quick-start development settings - unsuitable for production  
# See https://docs.djangoproject.com/en/2.1/howto/deployment/checklist/  
  
# SECURITY WARNING: keep the secret key used in production secret!  
SECRET\_KEY = 'exhlfdat&vfum(-34\*c2uroi(($ww(yo$9pv98=e6p^gl(-eoj'  
  
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = True  
  
ALLOWED\_HOSTS = []  
  
  
# Application definition  
  
INSTALLED\_APPS = [  
 'blog.apps.BlogConfig',  
 'users.apps.UsersConfig',  
 'crispy\_forms',  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
]  
  
MIDDLEWARE = [  
 'django.middleware.security.SecurityMiddleware',  
 'django.contrib.sessions.middleware.SessionMiddleware',  
 'django.middleware.common.CommonMiddleware',  
 'django.middleware.csrf.CsrfViewMiddleware',  
 'django.contrib.auth.middleware.AuthenticationMiddleware',  
 'django.contrib.messages.middleware.MessageMiddleware',  
 'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]  
  
ROOT\_URLCONF = 'django\_project.urls'  
  
TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': [],  
 'APP\_DIRS': True,  
 'OPTIONS': {  
 'context\_processors': [  
 'django.template.context\_processors.debug',  
 'django.template.context\_processors.request',  
 'django.contrib.auth.context\_processors.auth',  
 'django.contrib.messages.context\_processors.messages',  
 ],  
 },  
 },  
]  
  
WSGI\_APPLICATION = 'django\_project.wsgi.application'  
  
  
# Database  
# https://docs.djangoproject.com/en/2.1/ref/settings/#databases  
  
DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.sqlite3',  
 'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),  
 }  
}  
  
  
# Password validation  
# https://docs.djangoproject.com/en/2.1/ref/settings/#auth-password-validators  
  
AUTH\_PASSWORD\_VALIDATORS = [  
 {  
 'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',  
 },  
]  
  
  
# Internationalization  
# https://docs.djangoproject.com/en/2.1/topics/i18n/  
  
LANGUAGE\_CODE = 'en-us'  
  
TIME\_ZONE = 'UTC'  
  
USE\_I18N = True  
  
USE\_L10N = True  
  
USE\_TZ = True  
  
  
# Static files (CSS, JavaScript, Images)  
# https://docs.djangoproject.com/en/2.1/howto/static-files/  
  
STATIC\_URL = '/static/'  
  
MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')  
MEDIA\_URL = '/media/'  
  
CRISPY\_TEMPLATE\_PACK = 'bootstrap4'  
  
LOGIN\_REDIRECT\_URL = 'blog-home'  
LOGIN\_URL = 'login'

urls.py

from django.contrib import admin  
from django.contrib.auth import views as auth\_views  
from django.urls import path, include  
from django.conf import settings  
from django.conf.urls.static import static  
from users import views as user\_views  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('register/', user\_views.register, name='register'),  
 path('profile/', user\_views.profile, name='profile'),  
 path('login/', auth\_views.LoginView.as\_view(template\_name='users/login.html'), name='login'),  
 path('logout/', auth\_views.LogoutView.as\_view(template\_name='users/logout.html'), name='logout'),  
 path('', include('blog.urls')),  
]  
  
  
if settings.DEBUG:  
 urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

The “Users” folder:

Login.html

{% extends "blog/base.html" %}  
{% load crispy\_forms\_tags %}  
{% block content %}  
 <div class="content-section">  
 <form method="POST">  
 {% csrf\_token %}  
 <fieldset class="form-group">  
 <legend class="border-bottom mb-4">Log In</legend>  
 {{ form|crispy }}  
 </fieldset>  
 <div class="form-group">  
 <button class="btn btn-outline-info" type="submit">Login</button>  
 </div>  
 </form>  
 <div class="border-top pt-3">  
 <small class="text-muted">  
 Need An Account? <a class="ml-2" href="{% url 'register' %}">Sign Up Now</a>  
 </small>  
 </div>  
 </div>  
{% endblock content %}

Logout.html

{% extends "blog/base.html" %}  
{% block content %}  
 <h2>You have been logged out</h2>  
 <div class="border-top pt-3">  
 <small class="text-muted">  
 <a href="{% url 'login' %}">Log In Again</a>  
 </small>  
 </div>  
{% endblock content %}

Profile.html

{% extends "blog/base.html" %}  
{% load crispy\_forms\_tags %}  
{% block content %}  
 <div class="content-section">  
 <div class="media">  
 <img class="rounded-circle account-img" src="{{ user.profile.image.url }}">  
 <div class="media-body">  
 <h2 class="account-heading">{{ user.username }}</h2>  
 <p class="text-secondary">{{ user.email }}</p>  
 </div>  
 </div>  
 <form method="POST" enctype="multipart/form-data">  
 {% csrf\_token %}  
 <fieldset class="form-group">  
 <legend class="border-bottom mb-4">Profile Info</legend>  
 {{ u\_form|crispy }}  
 {{ p\_form|crispy }}  
 </fieldset>  
 <div class="form-group">  
 <button class="btn btn-outline-info" type="submit">Update</button>  
 </div>  
 </form>  
 </div>  
{% endblock content %}

Register.html

{% extends "blog/base.html" %}  
{% load crispy\_forms\_tags %}  
{% block content %}  
 <div class="content-section">  
 <form method="POST">  
 {% csrf\_token %}  
 <fieldset class="form-group">  
 <legend class="border-bottom mb-4">Join Today</legend>  
 {{ form|crispy }}  
 </fieldset>  
 <div class="form-group">  
 <button class="btn btn-outline-info" type="submit">Sign Up</button>  
 </div>  
 </form>  
 <div class="border-top pt-3">  
 <small class="text-muted">  
 Already Have An Account? <a class="ml-2" href="{% url 'login' %}">Sign In</a>  
 </small>  
 </div>  
 </div>  
{% endblock content %}

Admin.py

from django.contrib import admin  
from .models import Profile  
  
admin.site.register(Profile)

apps.py

from django.apps import AppConfig  
  
  
class UsersConfig(AppConfig):  
 name = 'users'  
  
 def ready(self):  
 import users.signals

forms.py

from django import forms  
from django.contrib.auth.models import User  
from django.contrib.auth.forms import UserCreationForm  
from .models import Profile  
  
  
  
class UserRegisterForm(UserCreationForm):  
 email = forms.EmailField()  
  
 class Meta:  
 model = User  
 fields = ['username', 'email', 'password1', 'password2']  
  
class UserUpdateForm(forms.ModelForm):  
 email = forms.EmailField()  
  
 class Meta:  
 model = User  
 fields = ['username', 'email']  
  
  
class ProfileUpdateForm(forms.ModelForm):  
 class Meta:  
 model = Profile  
 fields = ['image']

models.py

from django.db import models  
from django.contrib.auth.models import User  
from PIL import Image  
  
  
class Profile(models.Model):  
 user = models.OneToOneField(User, on\_delete=models.CASCADE)  
 image = models.ImageField(default='default.jpg', upload\_to='profile\_pics')  
  
 def \_\_str\_\_(self):  
 return f'{self.user.username} Profile'  
  
 def save(self):  
 super().save()  
  
 img = Image.open(self.image.path)  
  
 if img.height > 300 or img.width > 300:  
 output\_size = (300,300)  
 img.thumbnail(output\_size)  
 img.save(self.image.path)

signals.py

from django.db.models.signals import post\_save  
from django.contrib.auth.models import User  
from django.dispatch import receiver  
from .models import Profile  
  
  
@receiver(post\_save, sender=User)  
def create\_profile(sender, instance, created, \*\*kwargs):  
 if created:  
 Profile.objects.create(user=instance)  
  
  
@receiver(post\_save, sender=User)  
def save\_profile(sender, instance, \*\*kwargs):  
 instance.profile.save()

views.py

from django.shortcuts import render, redirect  
from django.contrib import messages  
from django.contrib.auth.decorators import login\_required  
from .forms import UserRegisterForm, UserUpdateForm, ProfileUpdateForm  
  
def register(request):  
 if request.method == 'POST':  
 form = UserRegisterForm(request.POST)  
 if form.is\_valid():  
 form.save()  
 username = form.cleaned\_data.get('username')  
 messages.success(request, f'Your account has been created! You are now able to log in')  
 return redirect('login')  
 else:  
 form = UserRegisterForm()  
 return render(request, 'users/register.html', {'form': form})  
  
  
@login\_required  
def profile(request):  
 if request.method =='POST':  
 u\_form = UserUpdateForm(request.POST, instance = request.user)  
 p\_form = ProfileUpdateForm(request.POST,  
 request.FILES,  
 instance=request.user.profile)  
 if u\_form.is\_valid() and p\_form.is\_valid():  
 u\_form.save()  
 p\_form.save()  
 messages.success(request, f'Your account has been updated!')  
 return redirect('profile')  
 else:  
 u\_form = UserUpdateForm(instance=request.user)  
 p\_form = ProfileUpdateForm(instance=request.user.profile)  
  
 context = {  
 'u\_form': u\_form,  
 'p\_form': p\_form  
 }  
 return render(request, 'users/profile.html', context)

manage.py

if \_\_name\_\_ == '\_\_main\_\_':  
 os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'django\_project.settings')  
 try:  
 from django.core.management import execute\_from\_command\_line  
 except ImportError as exc:  
 raise ImportError(  
 "Couldn't import Django. Are you sure it's installed and "  
 "available on your PYTHONPATH environment variable? Did you "  
 "forget to activate a virtual environment?"  
 ) from exc  
 execute\_from\_command\_line(sys.argv)

**References.**

1. Unknown.(unknown). Django views and URLconfs. Retrieved April 12, 2019, https://djangobook.com/views-urlconfs/
2. Big-nige.(May 12). Django overview. Retrieved April 12, 2019, https://djangobook.com/django-tutorials/django-overview/
3. Big-nige.(May 12). Django overview. Retrieved April 12, 2019, https://djangobook.com/django-tutorials/django-overview/
4. Big-nige(June 10). Python for Django programmers part-1. Retrieved April 12 2019, <https://djangobook.com/django-tutorials/python-django-programmers-part-1/>
5. Big-nige(June 14). Python for Django programmers part-2. Retrieved April 12 2019, <https://djangobook.com/django-tutorials/python-django-programmers-part-2/>
6. Big-nige(June 18). Python for Django programmers part-3. Retrieved April 12 2019, <https://djangobook.com/django-tutorials/python-django-programmers-part-3/>
7. Unknown(unknown). Django Documentation. Retrieved April 15. 2019. <https://docs.djangoproject.com/en/2.1/howto/static-files/>
8. Unknown(unknown). Using the Django authentication system. Retrieved April 15. 2019. <https://docs.djangoproject.com/en/2.2/topics/auth/default/>
9. Unknown(unknown). Bootstrap Pagination. Retrieved April 15, 2019. <https://www.w3schools.com/bootstrap/bootstrap_pagination.asp>
10. Unknown(unknown). Bootstrap Get Started. Retrieved April 15, 2019. <https://www.w3schools.com/bootstrap/bootstrap_get_started.asp>
11. Unknown(unknown). Bootstrap buttons. Retrieved April 15, 2019. <https://www.w3schools.com/bootstrap/bootstrap_buttons.asp>
12. Unknown(unknown). Bootstrap dropdowns. Retrieved April 15, 2019. <https://www.w3schools.com/bootstrap/bootstrap_dropdowns.asp>
13. Unknown(unknown). Django display views. Retrieved April 15, 2019. <https://docs.djangoproject.com/en/2.2/ref/class-based-views/generic-display/>
14. Johnsharpe(June 16, 2016). AttributeError: module ‘html.parser’has no attribute ‘HTMLParseError’. Retrieved April 15, 2019. <https://stackoverflow.com/questions/34827566/attributeerror-module-html-parser-has-no-attribute-htmlparseerror>
15. Canina M. (April 1, 2019). Generate UML class diagrams from Django models Retrieved April 15, 2019. <https://simpleit.rocks/python/django/generate-uml-class-diagrams-from-django-models/>
16. Unknown(unknown). Creating entity relationship diagram. Retrieved April 19, 2019. <https://cyphon.readthedocs.io/en/latest/dev-guide/graphing-models.html>
17. CodingMedved(June 1, 2017). Codingmedved/shop. Retrieved April 12, 2019. <https://github.com/codingmedved/shop>
18. Finer J.(April 1, 2019). Get started with Django part 1: Build a portfolio app. Retrieved April 12, 2019. <https://realpython.com/get-started-with-django-1/>
19. Unknown(unknown). Working with Forms. Retrieved April 15, 2019. <https://docs.djangoproject.com/en/2.2/topics/forms/>
20. Unknown(unknown). How to use Sessions. Retrieved April 15, 2019. <https://docs.djangoproject.com/en/2.2/topics/http/sessions/>
21. Unknown(unknown). Models. Retrieved April 15, 2019. <https://docs.djangoproject.com/en/2.2/ref/models/>
22. Unknown(unknown). Request and Response objects. Retrieved April 15, 2019. <https://docs.djangoproject.com/en/2.2/ref/request-response/>
23. Unknown(unknown). Signals. Retrieved April 17, 2019. <https://docs.djangoproject.com/en/2.2/ref/signals/>
24. Unknown(unknown). Templates. Retrieved April 17, 2019. <https://docs.djangoproject.com/en/2.2/ref/templates/>
25. Unknown(unknown). Django.urls functions for use in URLConfs. Retrieved April 17, 2019. <https://docs.djangoproject.com/en/2.2/ref/urls/>
26. Unknown(unknown). Django.urls utility functions. Retrieved April 17, 2019. <https://docs.djangoproject.com/en/2.2/ref/urlresolvers/>
27. Unknown(unknown). Migration operations. Retrieved April 17, 2019. <https://docs.djangoproject.com/en/2.2/ref/migration-operations/>
28. Unknown(unknown). Databases. Retrieved April 17, 2019. <https://docs.djangoproject.com/en/2.2/ref/databases/>
29. Unknown(unknown). Contrib packages. Retrieved April 17, 2019. <https://docs.djangoproject.com/en/2.2/ref/contrib/>
30. Unknown(unknown). Middleware. Retrieved April 17, 2019. <https://docs.djangoproject.com/en/2.2/ref/middleware/>
31. Unknown(unknown). Javascript. Retrieved April 17, 2019. <https://getbootstrap.com/docs/4.3/getting-started/javascript/>
32. Unknown(unknown). Browsers and Devices. Retrieved April 17, 2019. <https://getbootstrap.com/docs/4.3/getting-started/browsers-devices/>
33. SAFWANA(August 18, 2017). How to use Bootstrap with HTML. Retrieved April 17, 2019. <https://www.bootstrapdash.com/use-bootstrap-with-html/>
34. Unknown(unknown). Bootstrap images. Retrieved April 17, 2019. <https://www.w3schools.com/bootstrap/bootstrap_images.asp>
35. Unknown(unknown). Bootstrap navigation bar. Retrieved April 13, 2019. <https://www.w3schools.com/bootstrap/bootstrap_navbar.asp>
36. Unknown(unknown). The syndication feed framework. Retrieved April 23, 2019. <https://docs.djangoproject.com/en/2.2/ref/contrib/syndication/>