

Imię i nazwisko: Kamiński Mateusz

Nr albumu: 47087

Informatyka V semestr

Grupa 1 niestacjonarna

**Wielowarstwowa aplikacja internetowa
stworzona z wykorzystaniem HTML, CSS,
JavaScript oraz Python (framework Django)**

1. Model MVT wykorzystywany przez framework Django

Django posługuje się modelem MVT (Model, View, Template).

MVT to wzorzec projektowy lub architektura projektowa, którą Django stosuje do tworzenia aplikacji internetowych. Różni się nieco od powszechnie znanego wzorca projektowego MVC (Model-View-Controller). MVT określa całkowitą strukturę i przepływ pracy aplikacji Django. W architekturze MVT — Model zarządza danymi i jest reprezentowany przez bazę danych. Model jest w zasadzie tabelą bazy danych. Widok odbiera żądania HTTP i wysyła odpowiedzi HTTP. Widok wchodzi w interakcję z modelem i szablonem w celu uzupełnienia odpowiedzi. Szablon jest w zasadzie warstwą frontendową i dynamicznym komponentem HTML aplikacji Django.

2. Opis działania aplikacji

Aplikacja pełni rolę chatu z podziałem na tematyczne pokoje. Do udziału w konwersacji niezbędne jest założenie konta. Użytkownik może edytować swoje dane, tworzyć nowe pokoje chatu oraz przeglądać profile innych użytkowników. Do uruchomienia aplikacji w środowisku lokalnym niezbędne jest zainstalowanie: Python, Django, Django-Rest-Framework, Pillow, Django-Cors-Headers. Będąc w katalogu aplikacji, komenda „python manage.py runserver” uruchamia ją.

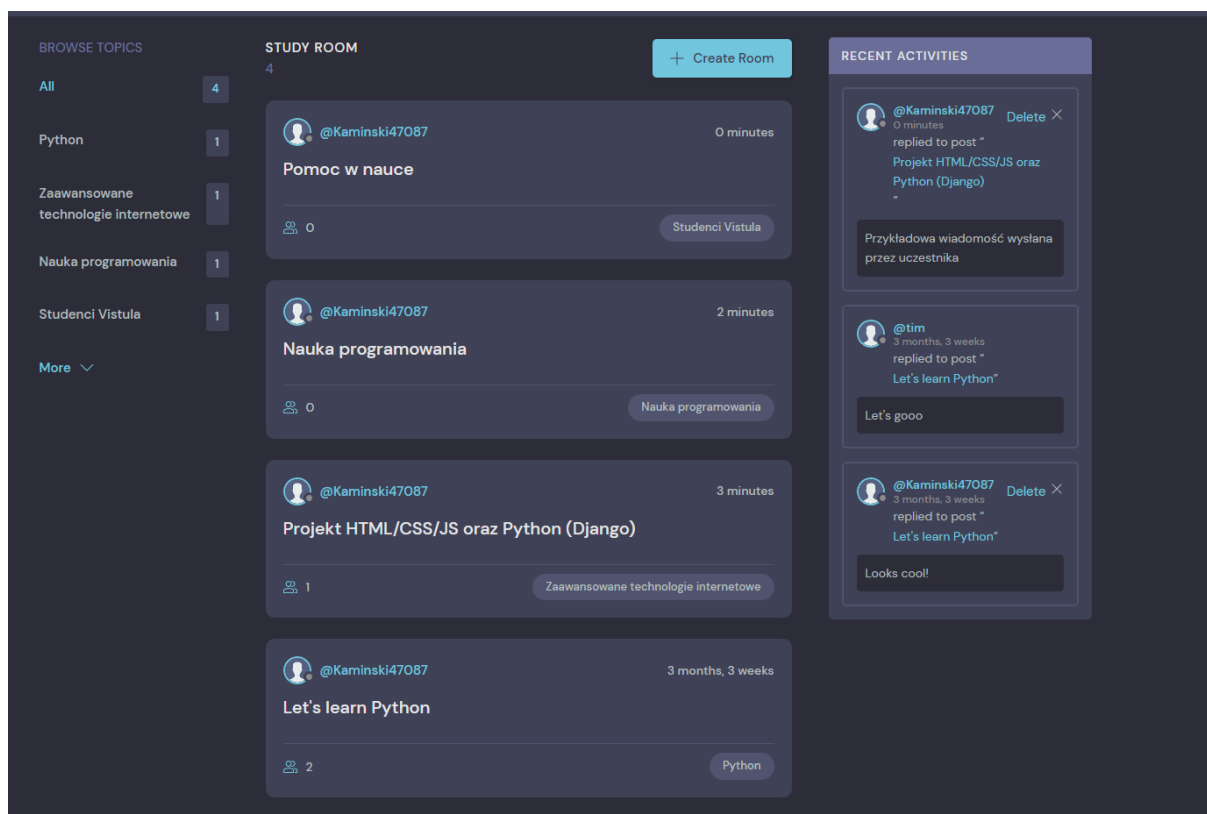
Konto administratora (logowanie):

Nazwa użytkownika: kaminski47087@mail.com

Hasło: kaminskivistula

3. Widoki aplikacji oraz odpowiadająca im logika biznesowa

Widok strony głównej z poziomu zalogowanego użytkownika.



W lewej części strony „Browse topics” wyświetlane są stworzone do tej pory przez użytkowników tematy pokoi. Centralna część przedstawia ostatnio utworzone pokoje, ich nazwy, ilość uczestników chatu oraz datę utworzenia.

Sekcja „Recent activities” przedstawia ostatnie wpisy dodane przez użytkowników.

Widok tworzenia nowego pokoju. Formularz wymaga podania tematu pokoju, który wykorzystywany jest do segregowania go z innymi pokojami o tej samej tematyce, oraz nazwy, pod którą będzie wyświetlał się chat. Użytkownik może także podać dokładniejszy opis celu istnienia chatu. Framework Django wymusza użycie tzw. CSRF Token podczas tworzenia formularza. Zabezpiecza on użytkowników przed podatnością CSRF.

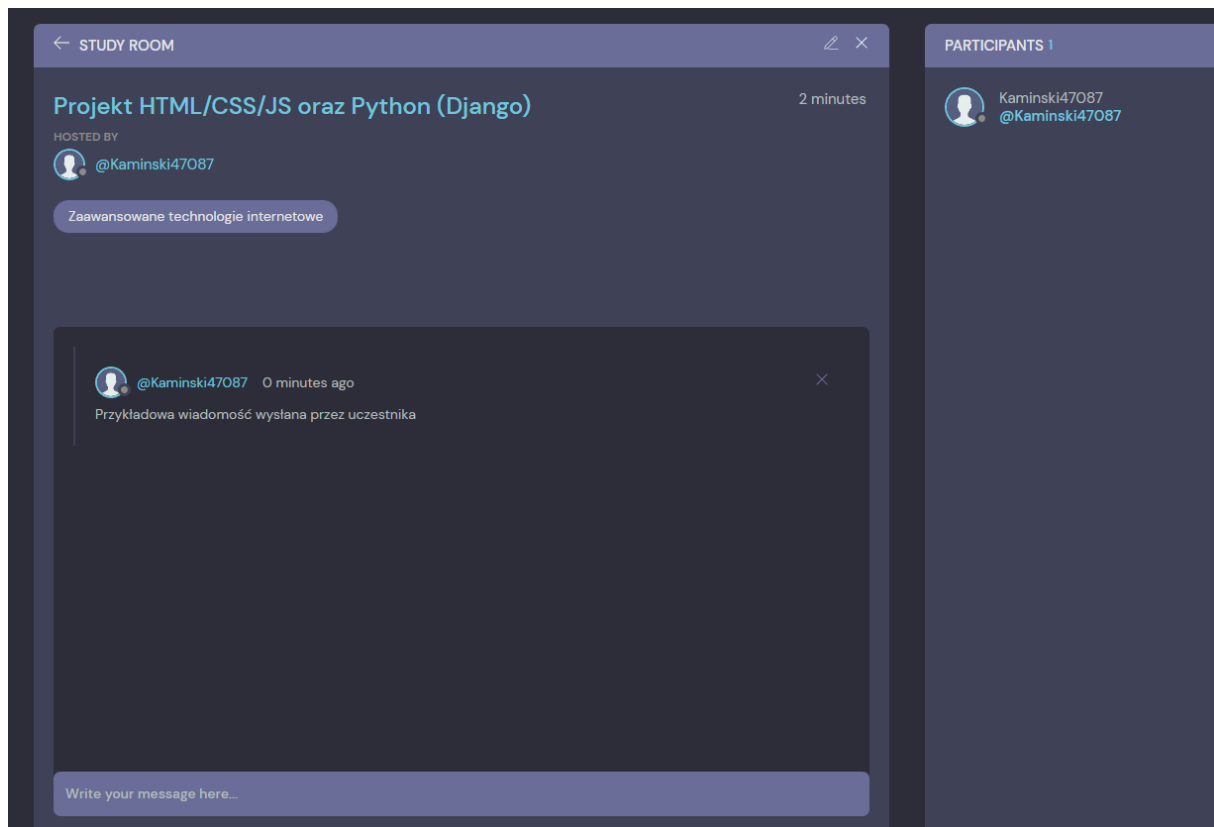
Widok funkcyjny `create_room` opatrzony jest w dekorator, który wymaga od użytkownika, aby był zalogowany przed dokonaniem operacji utworzenia pokoju. Z bazy danych pobieram tematy, w celu sprawdzenia, czy ten utworzony przez użytkownika powinien być do niej dopisany lub pobrany. Używam tu metody `get_or_create`. Formularz tworzony jest z wykorzystaniem mechanizmu dostarczanego przez framework Django, który znacząco przyspiesza ich tworzenie. Kod formularzy pokazany jest na późniejszych stronach. Odwołuję się do modelu `Room`, który jest tabelą w bazie danych o takiej samej nazwie i tworzę nowy wpis. Po pomyślnej akcji POST przekierowuję użytkownika za pomocą „`redirect`” na stronę główną.

```
@login_required(login_url='login')
def create_room(request):
    form = RoomForm()
    topics = Topic.objects.all()
    if request.method == 'POST':
        topic_name = request.POST.get('topic')
        topic, created = Topic.objects.get_or_create(name=topic_name)
        Room.objects.create(
            host=request.user,
            topic=topic,
            name=request.POST.get('name'),
            description=request.POST.get('description')
        )
    return redirect('home')

context = {'form': form, 'topics': topics}
return render(request, 'base/room_form.html', context)
```

Widok po dołączeniu do pokoju.

Z tego miejsca użytkownik może rozpocząć rozmowę. Widoczny jest temat chatu, jego założyciel oraz tag, z którym jest powiązany. Po prawej stronie w części „Participants” wyświetlają się użytkownicy, którzy dołączyli do chatu.



W widoku funkcyjnym room odwołuję się do modelu Room i pobieram z niego rekord o id=pk. Następnie pobieram z niego wiadomości oraz sortuję je według daty utworzenia. Odwołuję się także do „participants”, aby pobrać uczestników chatu. Obsługa „POST” w tym kontekście polega na odwołaniu się do modelu Message oraz utworzeniu nowego wpisu, z danymi użytkownika, pokojem oraz wiadomością. Na koniec dodaję nowego użytkownika do „participants”.

```
def room(request, pk):
    room = Room.objects.get(id=pk)
    room_messages = room.message_set.all().order_by('-created')
    participants = room.participants.all()

    if request.method == 'POST':
        message = Message.objects.create(
            user=request.user,
            room=room,
            body=request.POST.get('body')
        )
        room.participants.add(request.user)
        return redirect('room', pk=room.id)
    context = {'room': room, 'room_messages': room_messages, "participants": participants}
    return render(request, 'base/room.html', context)
```

Widok delete_room wykorzystuje dekorator login_required. Pobieram parametr pk i odwołujemy się do tabeli w bazy danych Room w celu pobrania informacji. Sprawdzam, czy użytkownik jest

założycielem pokoju. Jeśli nie, wyświetlam informację o braku uprawnień. W przypadku spełnienia wymagań usuwam rekord z tabeli Room i przekierowuję użytkownika na stronę główną.

```
@login_required(login_url='login')
def delete_room(request, pk):
    room = Room.objects.get(id=pk)

    if request.user != room.host:
        return HttpResponseRedirect('You are not allowed here.')

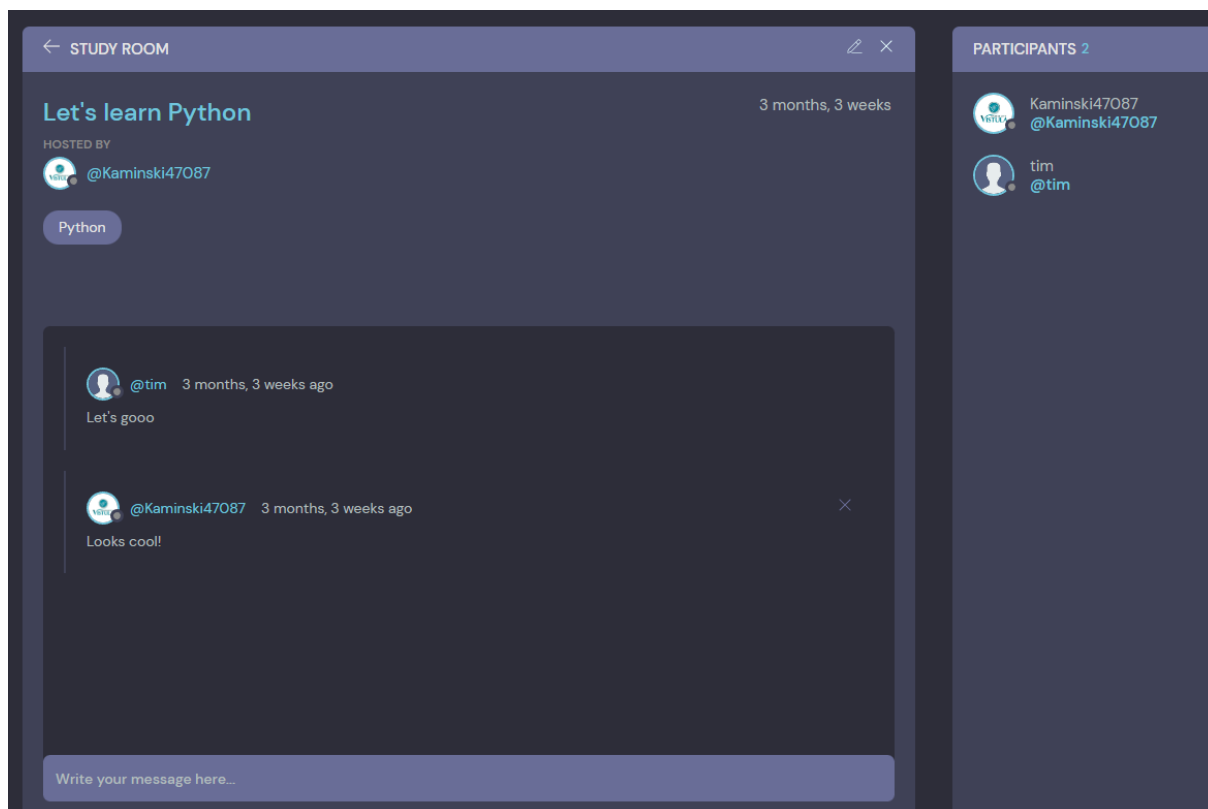
    if request.method == 'POST':
        room.delete()
        return redirect('home')
    return render(request, 'base/delete.html', {'obj': room})
```

Widok delete_message działa podobnie jak widok opisany powyżej. Różnicą jest odwołanie się do innej tabeli, tym razem Message. Sprawdzam, czy użytkownik jest „właścicielem” wpisu. Jeśli nie, wyświetlam komunikat o braku uprawnień. Jeśli tak usuwam rekord z tabeli Message i przekierowuję użytkownika na stronę główną.

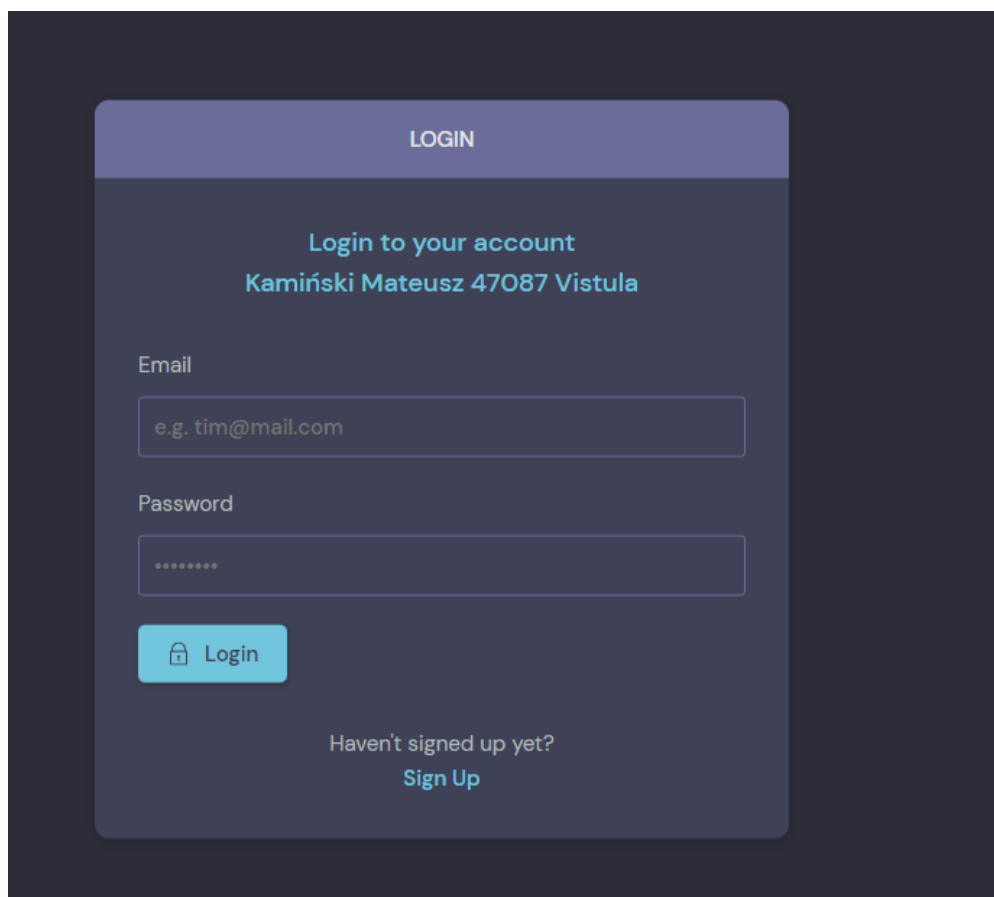
```
@login_required(login_url='login')
def delete_message(request, pk):
    message = Message.objects.get(id=pk)

    if request.user != message.user:
        return HttpResponseRedirect('You are not allowed here.')

    if request.method == 'POST':
        message.delete()
        return redirect('home')
    return render(request, 'base/delete.html', {'obj': message})
```



Widok logowania użytkownika (email oraz hasło), możliwość przejścia do strony rejestracji.



Na początku sprawdzam, czy użytkownik jest już zalogowany. Jeśli tak to zamiast wyświetlać stronę logowania przekierowuję go na stronę główną. W przypadku metody „POST” pobieram parametry „email” oraz „password”. Sprawdzam w bazie danych, wykorzystując model User, czy użytkownik o podanym emailu istnieje. Jeśli nie wyświetlam error. Jeśli istnieje, używam wbudowanej metody we framework Django „authenticate”. Gdy uzyskam prawidłowy wynik zwrotny, dokonuję logowania za pomocą metody login i przekierowuję użytkownika na stronę główną. Jeśli metoda authenticate zwróci błąd, wyświetlam error o nieprawidłowym emailu lub hasle.

```
def login_page(request):
    page = 'login'
    if request.user.is_authenticated:
        return redirect('home')

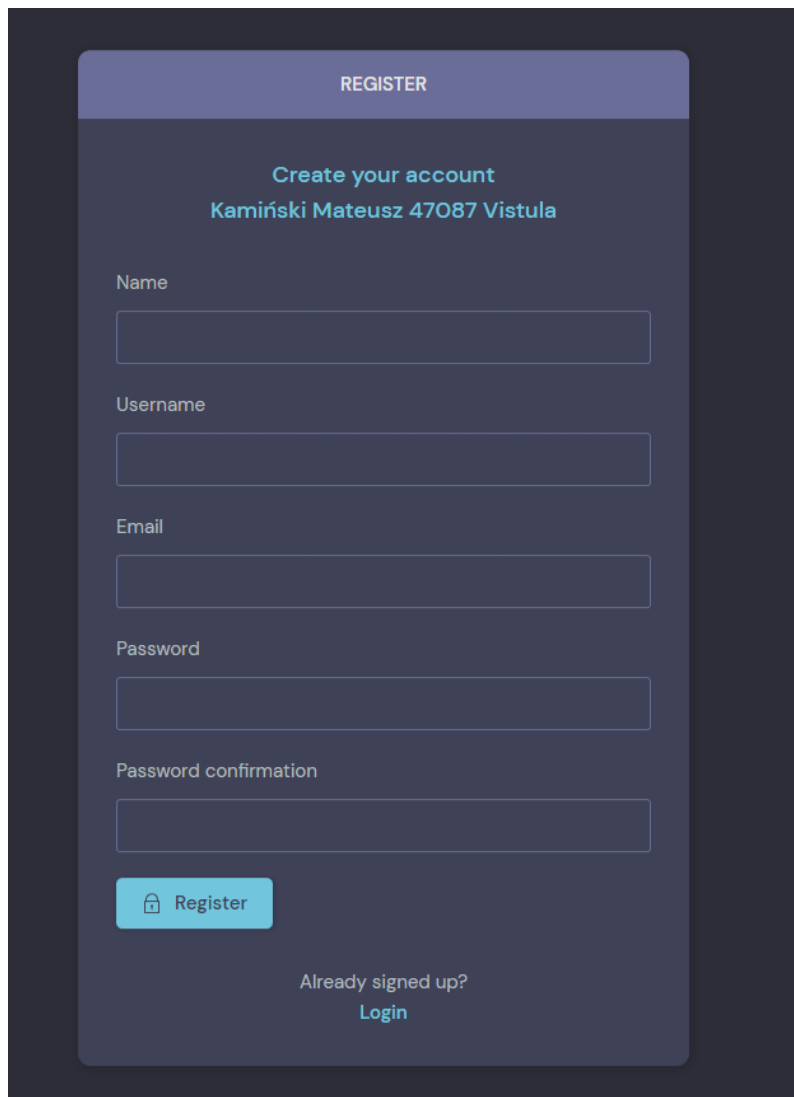
    if request.method == 'POST':
        email = request.POST.get('email')
        password = request.POST.get('password')

        try:
            user = User.objects.get(email=email)
        except:
            messages.error(request, 'Email does not exist')

        user = authenticate(request, email=email, password=password)
        if user is not None:
            login(request, user)
            return redirect('home')
        else:
            messages.error(request, "Email or password does not exist.")

    context = {'page': page}
    return render(request, 'base/login_register.html', context)
```


Widok strony rejestracji



REGISTER

Create your account
Kamiński Mateusz 47087 Vistula


Name

Username

Email

Password

Password confirmation

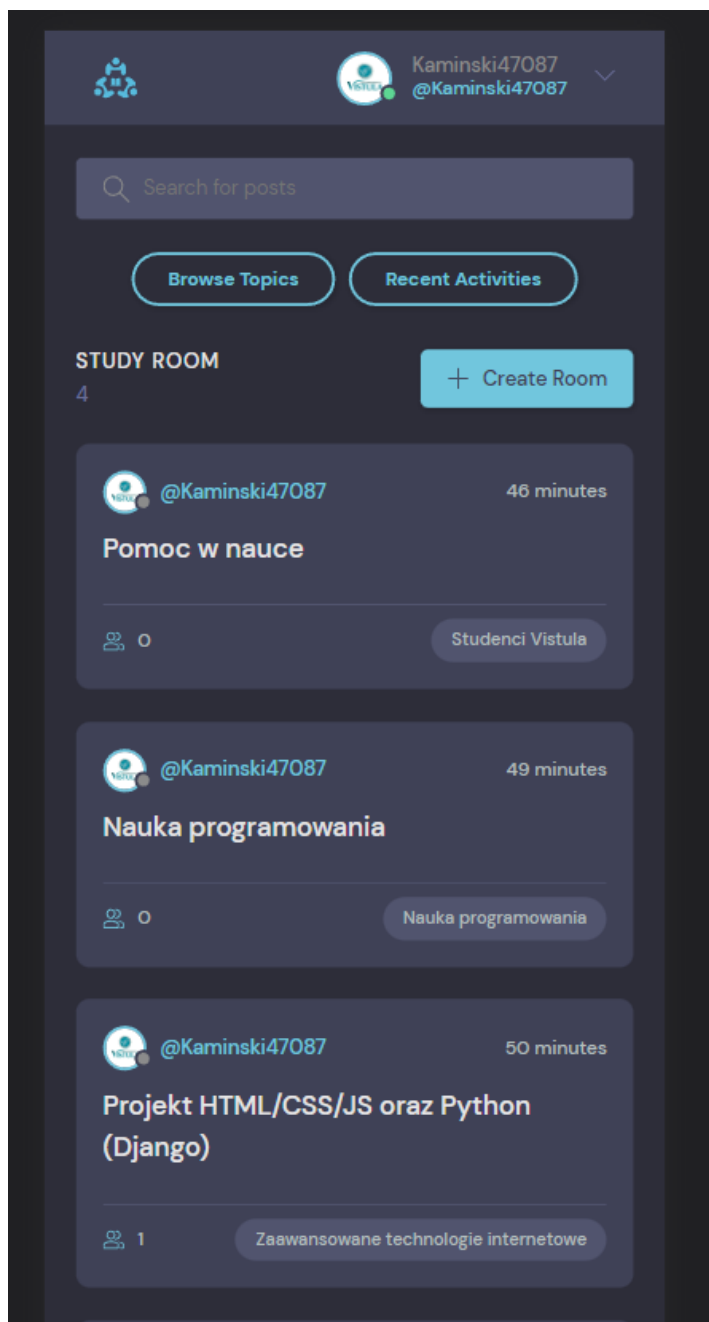
 Register

Already signed up?
[Login](#)

Tworzę formularz z wykorzystaniem mechanizmu upraszczającego dostarczonego przez framework Django. Sprawdzam poprawność danych wprowadzonych przez użytkownika. Jeśli są prawidłowe, zapisuję użytkownika w bazie danych oraz dokonuję automatycznego zalogowania konto oraz przekierowania na stronę główną.

```
def register_page(request):
    form = MyUserCreationForm()
    if request.method == 'POST':
        form = MyUserCreationForm(request.POST)
        if form.is_valid():
            user = form.save(commit=False)
            user.username = user.username.lower()
            user.save()
            login(request, user)
            return redirect('home')
        else:
            messages.error(request, 'An error has occurred during registration.')
    return render(request, 'base/login_register.html', {'form': form})
```

Strona jest w pełni responsywna. Przykładowe widoki strony głównej oraz przeglądu utworzonych tematów dla urządzeń mobilnych.





Kaminski47087
@Kaminski47087

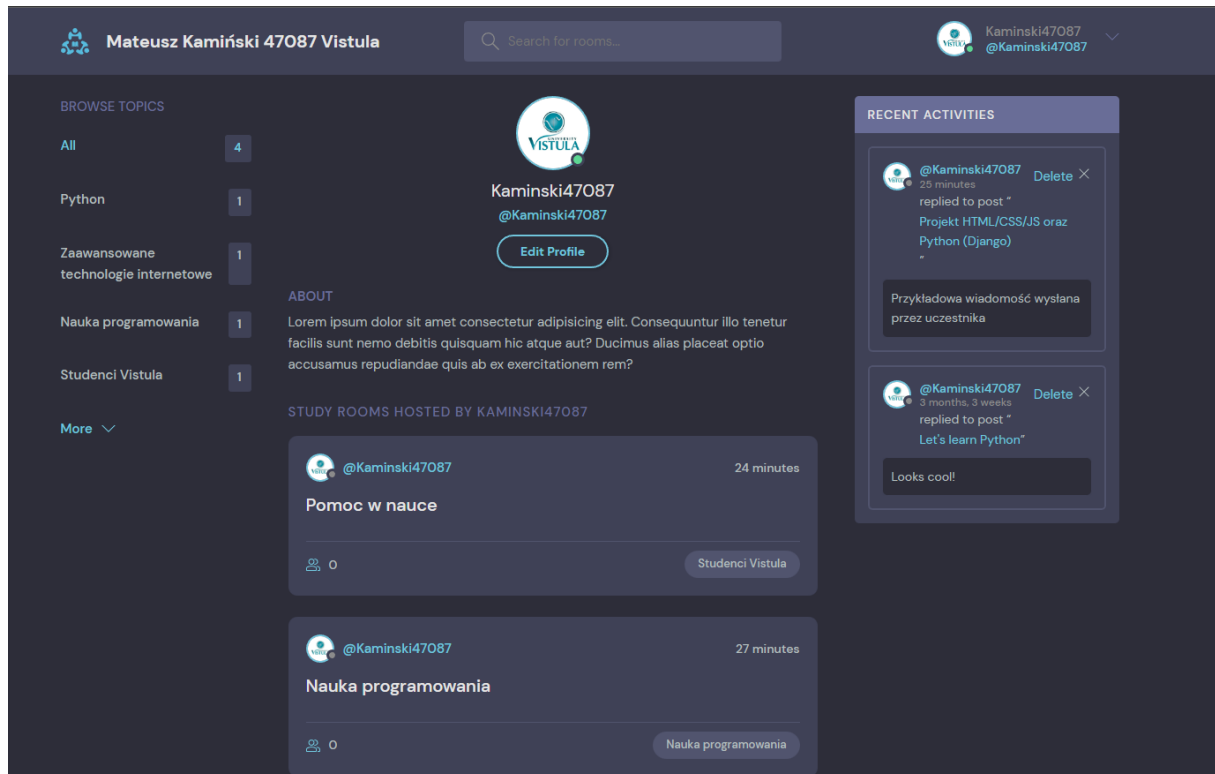


BROWSE TOPICS

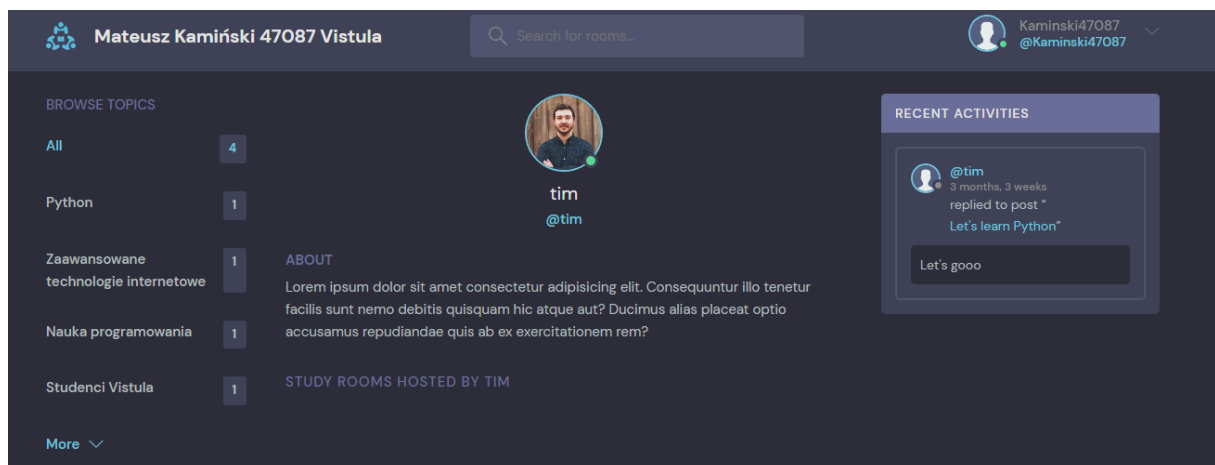


All	4
Python	1
Zaawansowane technologie internetowe	1
Nauka programowania	1
Studenci Vistula	1

Widok podglądu konta użytkownika. Przedstawia jego nazwę, opis, utworzone pokoju, oraz tablicę ostatnich aktywności. Zalogowany użytkownik może także zmienić swoje dane, co przedstawione jest na następnym zrzucie ekranu.



Widok przeglądu konta innego z użytkowników.



W celu wyświetlenia informacji o użytkowniku stworzyłem widok funkcyjny `user_profile`. Pobieram parametr `pk` i wykonuję zapytanie do tabeli `User` w bazie danych. Pobieram informacje o wszystkich pokojach oraz wiadomościach użytkownika oraz tematach z tabeli `Topics`.

```
def user_profile(request, pk):
    user = User.objects.get(id=pk)
    rooms = user.room_set.all()
    room_messages = user.message_set.all()
    topics = Topic.objects.all()
    context = {'user': user, 'rooms': rooms, 'room_messages': room_messages, 'topics': topics}
    return render(request, 'base/profile.html', context)

@login_required(login_url='login')
```

Widok edycji danych konta przez zalogowanego użytkownika. Formularz daje możliwość zmiany awatara, nazwy, email, oraz opisu konta.

Widok funkcyjny `update_user` wykorzystuje dekorator `login_required` dostarczony przez framework Django. Pobieram dane o użytkowniku i wypełniam nimi formularz `UserForm`. W tym kontekście obsługa metody „POST” polega na sprawdzeniu poprawności formularza za pomocą metody wbudowanej w Django „`is_valid`”. Jeśli jest on poprawny następuje aktualizacja danych oraz przekierowanie na profil użytkownika.

```

@login_required(login_url='login')
✓ def update_user(request):
    user = request.user
    form = UserForm(instance=user)

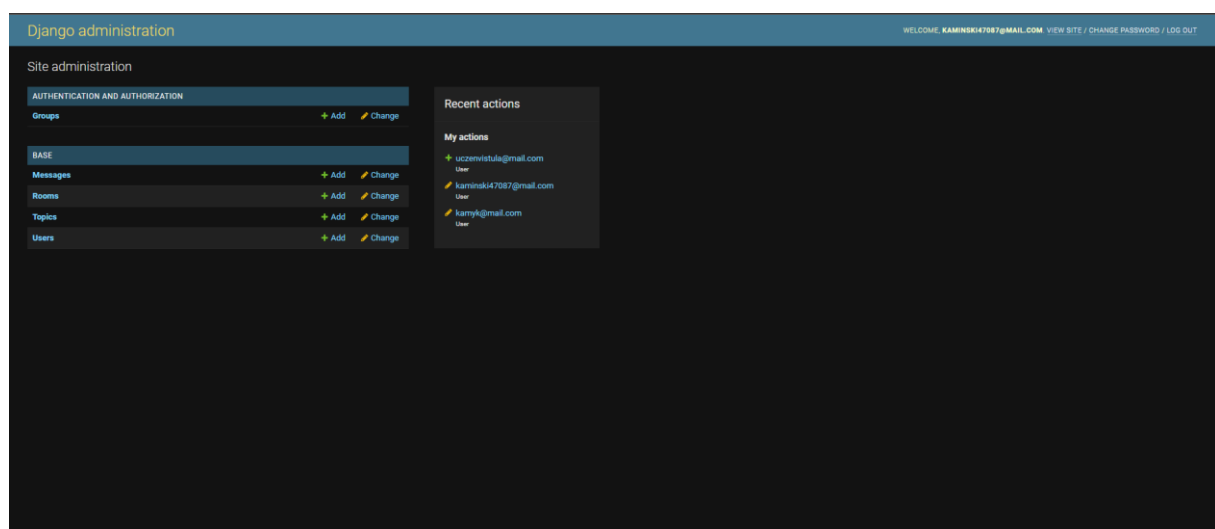
    if request.method == 'POST':
        form = UserForm(request.POST, request.FILES, instance=user)
        if form.is_valid():
            form.save()
            return redirect('user_profile', pk=user.id)

    return render(request, 'base/update_user.html', {'form': form})

```

Widok panelu administracyjnego.

ORM dostarczany przez Django. Widoczne są utworzone modele wykorzystywane przez bazę danych.



Z panelu administracyjnego istnieje także możliwość dodania użytkowników lub edycja danych.

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

BASE

Messages + Add

Rooms + Add

Topics + Add

Users + Add

Add user

Password:

Last login:

Date: Today | 📅

Time: Now | ⌚

Note: You are 2 hours ahead of server time.

☒ Superuser status

Designates that this user has all permissions without explicitly assigning them.

Groups:

+
-

The groups this user belongs to. A user will get all permissions granted to each of their groups. Hold down "Control", or "Command" on a Mac, to select more than one.

User permissions:

admin | log entry | Can add log entry
admin | log entry | Can change log entry
admin | log entry | Can delete log entry
admin | log entry | Can view log entry
auth | group | Can add group
auth | group | Can change group
auth | group | Can delete group
auth | group | Can view group
auth | permission | Can add permission

+
-

Specific permissions for this user. Hold down "Control", or "Command" on a Mac, to select more than one.

Username:

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

First name:

Last name:

☒ Staff status

Designates whether the user can log into this admin site.

☒ Active

Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

Active

Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

Date joined:

Date: 2022-10-04Today | 📅

Time: 09:33:20Now | ⌚

Note: You are 2 hours ahead of server time.

Name:

Email:

Bio:

Avatar:

Wybierz plik

Nie wybrano pliku

Save and add another

Save and continue editing

SAVE

Tworzenie oraz przegląd postów z poziomu panelu administracyjnego.

Django administration

WELCOME_KAMINSKI470877@MAIL.COM VIEW SITE / CHANGE PASSWORD / LOG OUT

Home - Base - Rooms

Start typing to filter:

AUTHENTICATION AND AUTHORIZATION

Groups + Add

BASE

Messages + Add

Rooms + Add

Topics + Add

Users + Add

Select room to change

Action: Go 0 of 4 selected

☐ ROOM

☐ Pomoc w nauce

☒ Nauka programowania
Python

☒ Projekt HTML/CSS/JS oraz Python (Django)

☐ Let's learn Python

4 rooms

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups

+ Add

BASE

Messages

+ Add

Rooms

+ Add

Topics

+ Add

Users

+ Add

Change user

uczenvistula@mail.com

HISTORY

Password:vistula

Last login:

Date:Today

📅

Time:Now

🕒

Note: You are 2 hours ahead of server time.

☐ Superuser status

Designates that this user has all permissions without explicitly assigning them.

Groups:

+

The groups this user belongs to. A user will get all permissions granted to each of their groups. Hold down "Control", or "Command" on a Mac, to select more than one.

User permissions:

admin | log entry | Can add log entry

admin | log entry | Can change log entry

admin | log entry | Can delete log entry

admin | log entry | Can view log entry

auth | group | Can add group

auth | group | Can change group

auth | group | Can delete group

auth | group | Can view group

auth | permissions | Can add permissions

Specific permissions for this user. Hold down "Control", or "Command" on a Mac, to select more than one.

Username:uczen_vistula

Required: 150 characters or fewer. Letters, digits and @/./+/-/_ only.

First name:Uczen

Last name:Vistula

☐ Staff status

Designates whether the user can log into this admin site.

☒ Active

Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

Date joined:

Date:2022-10-04Today

📅

Time:09:04:55Now

🕒

Note: You are 2 hours ahead of server time.

Name:uczenvistula

Email:uczenvistula@mail.com

Bio:

Testowy uczen Vistula

Avatar:

Currently:avatar.svg

Change:Wybierz plikNie wybrano pliku

Delete

Save and add another

Save and continue editing

SAVE

Django administration

WELCOME, KAMINSKI47087@MAIL.COM. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > Base > Users

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups

+ Add

BASE

Messages

+ Add

Rooms

+ Add

Topics

+ Add

Users

+ Add

Select user to change

ADD USER +

Action:Go0 of 3 selected

☐ user

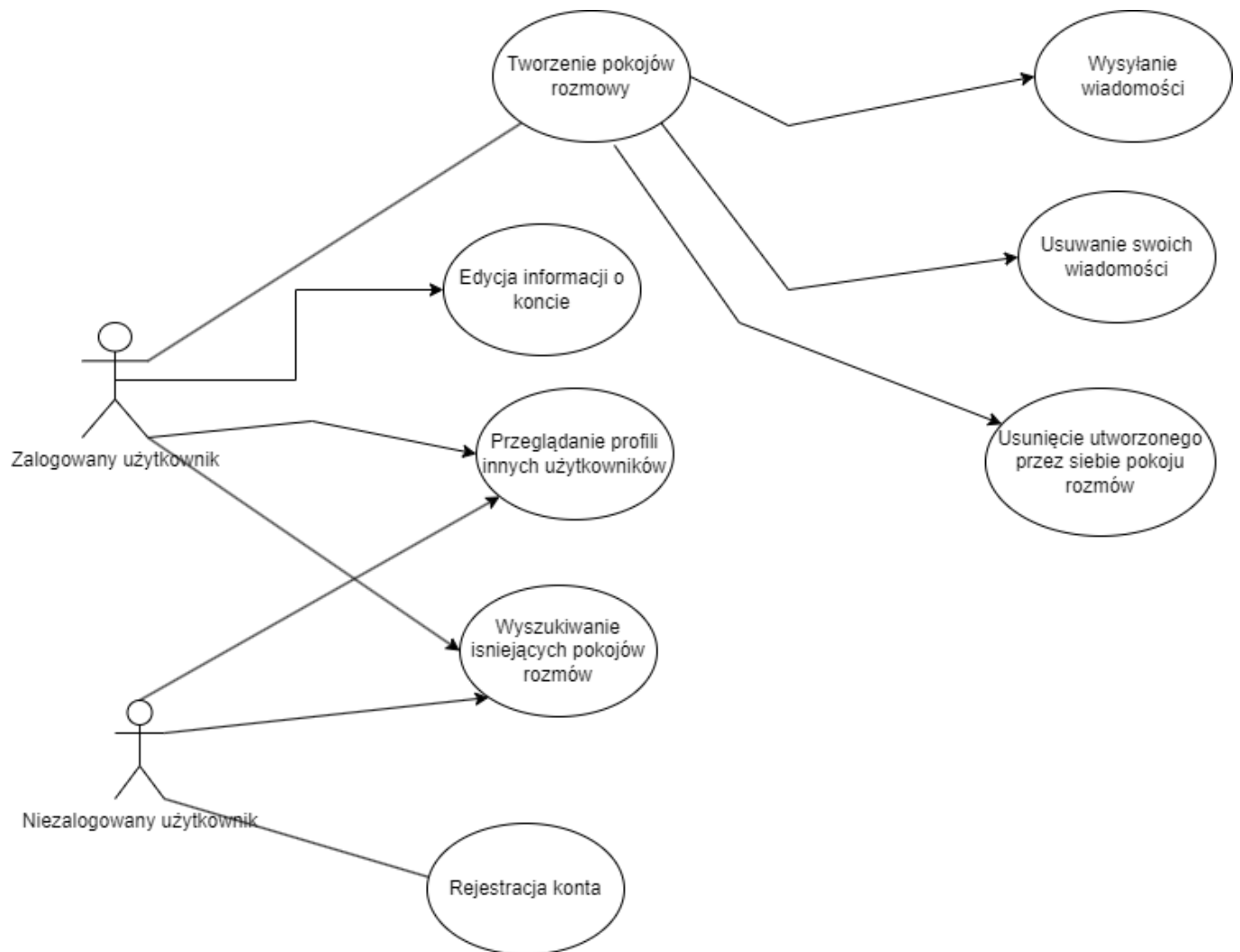
☒ uczenvistula@mail.com

☐ tm@mail.com

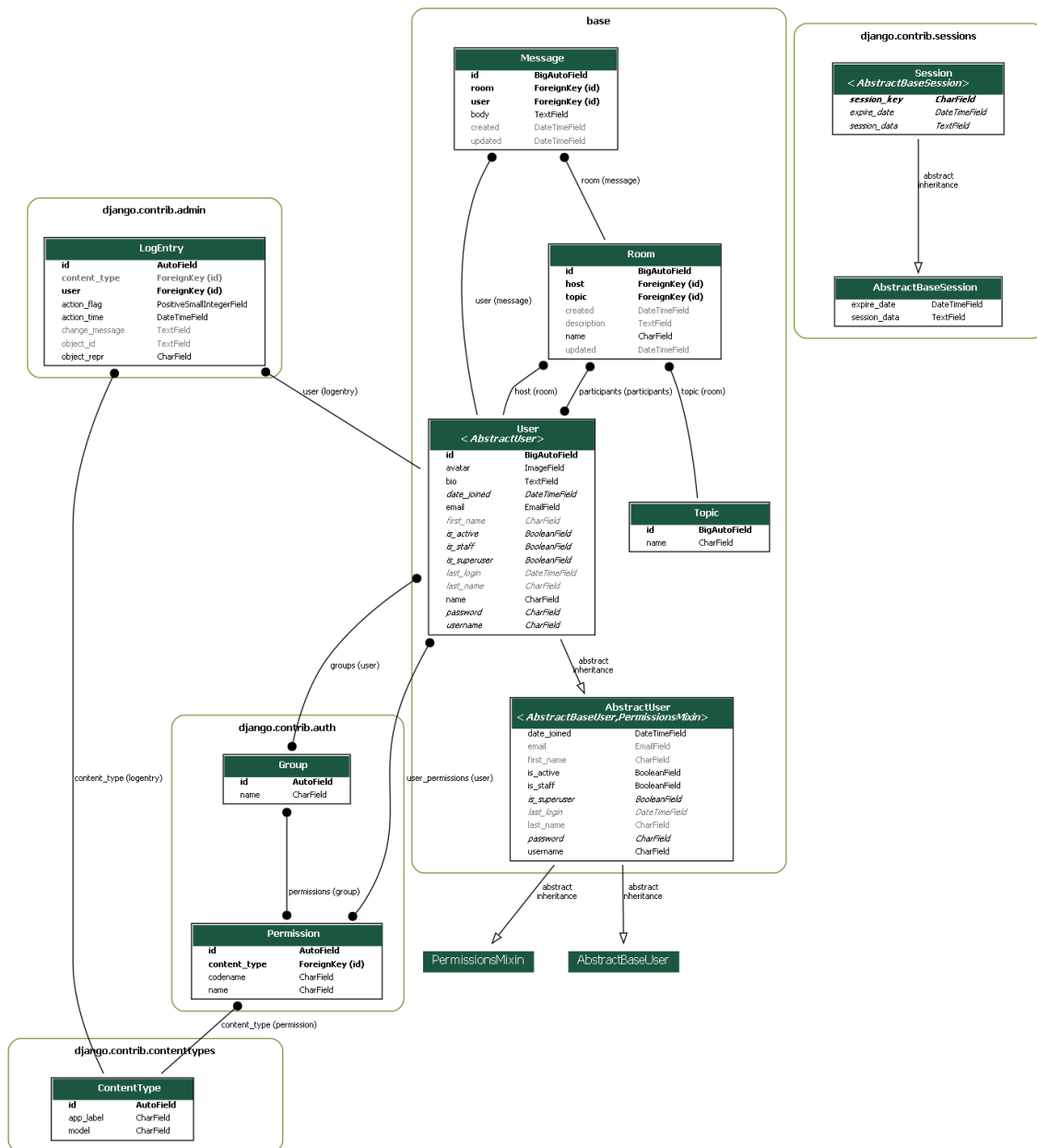
☐ kaminski47087@mail.com

3 users

4. Diagram przypadków użycia



5. Diagram bazy danych



Utworzone modele wykorzystywane przez bazę danych.

Model User, opisuje informacje przechowywane o użytkownikach. Posiada pola takie jak name (CharField), email (EmailField), bio (TextField), avatar (ImageField – w tym przypadku skorzystałem z zewnętrznej biblioteki o nazwie Pillow).

Kolejny model Topic, przechowuje informacje o tematach, posiada jedno pole name (CharField). Najbardziej rozbudowany model Room posiada pola host (służy jako klucz obcy do modelu User), topic (klucz obcy do modelu Topic), name (CharField), description (TextField), participants (relacja wiele do wielu z modelem User), updated (DateTimeField – czas aktualizacji rekordu), created (DateTimeField – czas utworzenia rekordu).

```

base > models.py > Message
1  from django.db import models
2  from django.contrib.auth.models import AbstractUser
3
4
5  class User(AbstractUser):
6      name = models.CharField(max_length=200, null=True)
7      email = models.EmailField(unique=True, null=True)
8      bio = models.TextField(null=True)
9      avatar = models.ImageField(null=True, default="avatar.svg")
10     USERNAME_FIELD = 'email'
11     REQUIRED_FIELDS = []
12
13     class Topic(models.Model):
14         name = models.CharField(max_length=200)
15
16         def __str__(self):
17             return self.name
18
19
20     class Room(models.Model):
21         host = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
22         topic = models.ForeignKey(Topic, on_delete=models.SET_NULL, null=True)
23         name = models.CharField(max_length=200)
24         description = models.TextField(null=True, blank=True)
25         participants = models.ManyToManyField(User, related_name='participants', blank=True)
26         updated = models.DateTimeField(auto_now=True)
27         created = models.DateTimeField(auto_now_add=True)
28
29         class Meta:
30             ordering = ['-updated', '-created']
31
32         def __str__(self):
33             return self.name

```

Model Message zawiera pola user(klucz obcy do modelu User), room (klucz obcy do modelu Room), body (TextField), updated (DateTimeField), created (DateTimeField).

```

class Message(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    room = models.ForeignKey(Room, on_delete=models.CASCADE, )
    body = models.TextField()
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-updated', '-created']

    def __str__(self):
        return self.body[:50]

```

Utworzone ścieżki aplikacji oraz odpowiadające im widoki funkcyjne Django.

```

from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

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

urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

```

```

from django.urls import path
from . import views

urlpatterns = [
    path('', views.get_routes),
    path('rooms', views.get_rooms),
    path('rooms/<str:pk>', views.get_room),
]

```

```

base > 🐞 urls.py > ...
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path('login/', views.login_page, name='login'),
6      path('logout/', views.logout_user, name='logout'),
7      path('register/', views.register_page, name='register'),
8      path('', views.home, name='home'),
9      path('room/<str:pk>', views.room, name='room'),
10     path('profile/<str:pk>', views.user_profile, name='user_profile'),
11
12     path('create_room/', views.create_room, name='create_room'),
13     path('update_room/<str:pk>', views.update_room, name='update_room'),
14     path('delete_room/<str:pk>', views.delete_room, name='delete_room'),
15     path('delete_message/<str:pk>', views.delete_message, name='delete_message'),
16     path('update-user', views.update_user, name='update_user'),
17     path('topics', views.topics_page, name='topics'),
18     path('activity', views.activity_page, name='activity'),
19
20 ]

```

Jeden z plików HTML. Dla przykładu wybrałem stronę główną, która dobrze obrazuje logikę Django, która ułatwia tworzenie stron. Wykorzystuję wcześniej utworzony template „main.html” (zrzut ekranu poniżej opisywanego obecnie pliku), który zawiera „statyczną” część strony pojawiającą się w wielu miejscach, np. strona główna, profil, aktualizacja danych. W bloku content wstawiamy naszą „modyfikowalną” część strony, która rozbudowuje „main.html”.

```
{% extends 'main.html' %}
{% block content %}
<main class="layout layout--3">
  <div class="container">
    <!-- Topics Start -->
    {% include 'base/topic_component.html' %}

    <!-- Topics End -->

    <!-- Room List Start -->
    <div class="roomList">
      <div class="mobile-menu">
        <form action="{% url 'home' %}" class="header__search">
          <label>
            <svg version="1.1" xmlns="http://www.w3.org/2000/svg" width="32" height="32"
              viewBox="0 0 32 32">
              <title>search</title>
              <path
                d="M32 30.5861-10.845-10.845c1.771-2.092 2.845-4.791 2.845-7.741 0-6.617-5.383-12-12-12s-12 5.383-12
              >>/path>
            </svg>
            <input name="q" placeholder="Search for posts"/>
          </label>
        </form>
        <div class="mobile-menuItems">
          <a class="btn btn--main btn--pill" href="{% url 'topics' %}">Browse Topics</a>
          <a class="btn btn--main btn--pill" href="{% url 'activity' %}">Recent Activities</a>
        </div>
      </div>
      <div class="roomList__header">
        <div>
          <h2>Study Room</h2>
          <p>{{ rooms.count }}</p>
        </div>
        <a class="btn btn--main" href="{% url 'create_room' %}">
          <svg version="1.1" xmlns="http://www.w3.org/2000/svg" width="32" height="32" viewBox="0 0 32 32">
            <title>add</title>
            <path
              d="M16.943 0.943h-1.885v14.115h-14.115v1.885h14.115v14.115h1.885v-14.115h14.115v1.885h-14.115v-14.115z"
            >>/path>
          </svg>
          Create Room
        </a>
      </div>
      {% include 'base/feed_component.html' %}
    </div>
    <!-- Room List End -->

    <!-- Activities Start -->
    {% include 'base/activity_component.html' %}

    <!-- Activities End -->
  </div>
</main>
{% endblock content %}
```

Template „main.html”

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="shortcut icon" href="assets/favicon.ico" type="image/x-icon" />
  <link rel="stylesheet" href="{% static 'styles/style.css' %}" />
  <title>Kamiński 47087</title>
</head>

<body>
  {% include 'navbar.html' %} {% if messages %}
  <ul class="messages">
    {% for message in messages %}
    <li>{{ message }}</li>
    {% endfor %}
  </ul>
  {% endif %} {% block content %} {% endblock %}
  <script src="{% static 'js/script.js' %}"></script>
</body>
</html>
```

Formularze utworzone za pomocą klas wbudowanych we framework Django.

```
base > forms.py > ...
1  from django.forms import ModelForm
2  from .models import Room, User
3  from django.contrib.auth.forms import UserCreationForm
4
5
6  class MyUserCreationForm(UserCreationForm):
7      class Meta:
8          model = User
9          fields = ['name', 'username', 'email', 'password1', 'password2']
10
11
12  class RoomForm(ModelForm):
13      class Meta:
14          model = Room
15          fields = '__all__'
16          exclude = ['host', 'participants']
17
18
19  class UserForm(ModelForm):
20      class Meta:
21          model = User
22          fields = ['avatar', 'name', 'username', 'email', 'bio']
23
```