Panel administracyjny i uprawnienia użytkowników

Panel administracyjny

- Jest jedną ze standardowych aplikacji Django;
  Zostaje dodany do projektu podczas jego tworzenia:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    # …
    'blog',
]
```

- Adres panelu jest zapisany w pliku urls.py projektu
  (domyślnie "admin", ale można zmienić, np. na "zaplecze"):

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

Funkcje

- Zarządzanie użytkownikami i grupami
  – operacje CRUD na grupach
  – operacje CRUD na użytkownikach
  (w tym przypisanie użytkownika do grupy)
- Zarządzanie treścią
  – operacje CRUD na wszystkich modelach

Zalety

- Ułatwia nadawanie uprawnień użytkowników przez grupy
  - Wystarczy dodać grupę i nadać uprawnienia grupie, a potem przypisać użytkownika do grupy – użytkownik otrzyma wszystkie jej uprawnienia
  - Użytkownik może być przypisany do wielu grup, dzięki czemu łatwiej jest budować złożone systemy uprawnień
  - Istnieje możliwość nadania wybranym użytkownikom dodatkowych uprawnień, oprócz tych, które wynikają z przynależności do grup
  - W bardzo prostych serwisach z niewielką liczbą użytkowników można nadawać uprawnienia z pominięciem grup
- Umożliwia zarządzanie całym serwisem bez pisania kodu
  - Umożliwia operacje CRUD na wszystkich modelach wszystkich aplikacji dodanych panelu przez plik admin.py:

```
from django.contrib import admin
from .models import Post, Comment

admin.site.register(Post)
admin.site.register(Comment)
```

- Uwzględnia uprawnienia użytkowników
  Użytkownik mający np. tylko uprawnienia create i read będzie miał możliwości dodawania i przeglądania danych modelu, ale edycji i usuwania już nie (wskazówka: do edycji i usuwania trzeba też mieć uprawnienie do odczytu danych)

Ograniczenia

- Nie ma możliwości nadania uprawnień edycji tylko do wybranych pól modelu lub wybranych rekordów danych:
  – uprawnienie do edycji użytkownika umożliwia nadanie sobie większych uprawnień, np. superusera
  – uprawnienie do edycji np. wpisów umożliwia edycję każdego wpisu, również wpisu innego użytkownika
- Nie zna kontekstu modeli, dlatego umożliwia również i takie zmiany danych modeli, które z punktu widzenia funkcjonalności serwisu nie mają sensu, np. zmiana autora wpisu/komentarza
- Wniosek: uprawnienia do korzystania z panelu powinno być ograniczone dla administratora, pozostali użytkownicy powinni korzystać wyłącznie z aplikacji

Uprawnienia użytkowników

Uprawnienia ogólne (niezwiązane z modelami):

- Zalogowany
- Aktywny – tylko użytkownik aktywny może się zalogować;
  umożliwia zablokowanie dostępu do serwisu bez konieczności usuwania konta użytkownika, co mogło by pociągnąć za sobą konieczność np. usunięcia wpisów/komentarzy, których jest autorem
- W zespole – ma prawo korzystania z panelu administracyjnego; tylko dla najbardziej zaufanych użytkowników – przez panel można dokonać dowolnych zmian w treści, np. zmienić autora wpisu
- Superużytkownik – ma wszystkie uprawnienia bez ich jawnego nadawania

Uprawnienia związane z modelami:

- Uprawnienia CRUD do modeli, nadawane dla poszczególnych modeli niezależnie; trzy z nich mają nazwy inne niż w CRUD:
  - Add (w CRUD: create)
  - View (w CRUD: read)
  - Change (w CRUD: update)
  - Delete

Uprawnienia w widokach i szablonach:

- Django w żaden sposób nie sprawdza uprawnień użytkowników w funkcjach widoku i szablonach – to już zadanie programisty; Jeżeli aplikacja nie zostanie wyposażona w odpowiednie zabezpieczenia, każdy użytkownik (również niezalogowany) będzie mógł dowolnie korzystać z formularzy aplikacji – np. edycji wpisów – i w efekcie modyfikować treść serwisu
- Django udostępnia mechanizmy do sprawdzania uprawnień; Można i trzeba z nich korzystać w dwóch miejscach:
    - szablony – powinny wyświetlać odnośniki tylko do tych części serwisu, do których użytkownik ma uprawnienia
    - Widoki – powinno się sprawdzać uprawnienia, na wypadek gdyby użytkownik próbował obejść zabezpieczenia szablonów, np. wpisał ręcznie adres niedostępnej dla niego części serwisu

Sprawdzanie uprawnień w szablonach:

- Kontekst wszystkich szablonów zawiera: obiekt żądania (request), dane przekazane przez funkcję widoku oraz dane dodawane przez tzw. procesory kontekstu (ich lista jest w konfiguracji projektu, w pliku settings.py):
    - user – obiekt typu auth.User, z danymi aktualnie zalogowanego użytkownika, dane jak w panelu administracyjnym Django,
      albo obiekt AnonymousUser, jeżeli nikt nie jest zalogowany;
    - perms – obiekt PermWrapper, z uprawnieniami użytkownika,
      jak w panelu administracyjnym Django, wg. schematu
      <aplikacja>.<add/view/change/delete>_<model),
      np. blog.add_post, blog.view_post, …, blog.add_comment, …
- Uprawnienia ogólne:
    - user.is_authenticated – zalogowany
    - user.is_staff – w zespole (dostęp do panelu)
    - user.is_superuser – superużytkownik
- Uprawnienia związane z modelami
    - perms. <aplikacja>.<add/view/change/delete>_<model>,
      np. perms.blog.add_comment
- Uprawnienia zależne od aplikacji, np. możliwość edycji wpisu tylko dla jego autora – można wykorzystać wszystkie dane przekazane w kontekście oraz zalogowanego użytkownika, np.:
    - user.is_authenticated and post.author.id == user.id
- Do sprawdzania uprawnień należy wykorzystać znacznik if:

```
{% if <uprawnienie> %}
  treść dla uprawnionego użytkownika
{% else %}
  treść dla nieuprawnionego użytkownika
{% endif %}
```

Sprawdzanie uprawnień w widokach:

- Widok otrzymuje jako argument obiekt żądania (request);
  Jest to słownik Pythona, zawierający szczegóły żądania otrzymane z serwera WWW, do którego Django dodaje m.in. przetworzone dane formularzy i ciasteczka oraz obiekt user:
  - user – obiekt typu auth.User, z danymi aktualnie zalogowanego użytkownika, dane jak w panelu administracyjnym Django,
    albo obiekt AnonymousUser, jeżeli nikt nie jest zalogowany;
- Uprawnienia ogólne:
  - request.user.is_authenticated – zalogowany
  - request.user.is_staff – w zespole (dostęp do panelu)
  - request.user.is_superuser – superużytkownik
- Uprawnienia związane z modelami
  - request.user.has_perm("<aplikacja>.<prawo>_<model>"),
    np. request.user.has_perm("blog.add_comment")
- Uprawnienia zależne od aplikacji, np. możliwość edycji wpisu tylko dla jego autora – można wykorzystać wszystkie dane pobierane z modeli oraz dane użytkownika, np.:
  - post = Post.objects.get(id=id)
    if post.author.id == request.user.id:
- Do sprawdzania uprawnień należy wykorzystać – co dość oczywiste – instrukcję warunkową, a dodatkowo można posłużyć się przekierowaniem – jeżeli użytkownik nie posiada wymaganych uprawnień, np.:

```
from django.shortcuts import render, redirect

def post_edit(request, id):
  if not request.user.has_perm('blog.change_post'):
    return redirect('/')
```

- Django ma dekoratory służące do sprawdzania uprawnień:
  - @login_required – sprawdza czy użytkownik jest zalogowany, jeżeli nie jest, to przekierowuje do formularza logowania
  - @permission_required("<aplikacja>.<prawo>_<model>") sprawdza czy użytkownik jest zalogowany i czy ma wskazane uprawnienie, jeżeli nie, to przekierowuje do formularza logowania – co nie zawsze ma sens
- Dekorator @login_required może być użyteczny, jednak uprawnienia modeli lepiej sprawdzać we własnym zakresie

```
from django.contrib.auth.decorators import login_required

@login_required
def post_new(request):
    if not request.user.has_perm('blog.add_post'):
        return redirect('/')
```

Zadania

Proszę uzupełnić projekt Django z poprzedniego ćwiczenia o sprawdzanie uprawnień użytkowników w szablonach i widokach, a w tym celu:

1. Zalogować się w panelu administracyjnym (http://127.0.0.1:8000/admin/login) jako administrator i nadać innym użytkownikom odpowiednie uprawnienia:
   (1) uprawnienie ogólne "w zespole", wszystkie uprawnienia do modeli `post` i `comment`
   (2) uprawnienie ogólne "w zespole", uprawnienie add (ale nie edit) do modelu `post` i wszystkie uprawnienia do modelu `comment`
   (3) uprawnienie ogólne "w zespole", brak uprawnień do modelu `post` i wszystkie uprawnienia do modelu `comment`
   (4) brak uprawnienia "w zespole", brak uprawnień do modeli `post` i `comment`
2. Sprawdzić możliwości logowania się i zmiany danych modeli `post` i `comment` w panelu administracyjnym dla użytkowników mających różne uprawnienia
3. Ponownie zalogować się jako administrator, zanotować lub zapisać w ulubionych przeglądarki internetowej adresy formularzy
   (1) dodawania wpisu
   (2) edycji wpisu
   (3) dodawania komentarza
4. Sprawdzić możliwość dodawania i edycji wpisów i komentarzy dla użytkowników niezalogowanych oraz zalogowanych, ale mających różne uprawnienia

   Po każdym z kolejnych zadań należy powtórzyć testy z punktu 4, korzystając przy tym z adresów zanotowanych w punkcie 3

5. Uzupełnić szablony aplikacji o sprawdzanie uprawnień w taki sposób, aby odnośniki do formularzy z punktu 3 były widoczne tylko dla uprawnionych użytkowników;
6. Uzupełnić funkcje widoków dodawania wpisu, edycji wpisu i dodawania komentarza o dekorator `@login_required`;
7. W funkcjach widoku z punktu 7 dodać sprawdzania uprawnień użytkownika dotyczących odnośnych modeli – np. dodawanie komentarza – uprawnienie blog.add_comment
8. W wybranej funkcji widoku zastąpić dekorator `@login_required` przez sprawdzenie uprawnienia użytkownika `is_authenticated`,
9. * Rozszerzyć zabezpieczenie z punktów 5-8 (w szablonie i w widoku), aby edycji wpisu mógł dokonać tylko jego autor, a nie dowolny zalogowany użytkownik;
10. ** Rozszerzyć zabezpieczenie z punktów 5-8 (w szablonie i w widoku), aby autor wpisu nie mógł dodać pierwszego komentarza do wpisu, ale drugi i kolejne już tak

Wskazówki:

1. Można przyspieszyć testowanie zabezpieczeń, mając jednocześnie zalogowanych kilku użytkowników, z różnymi uprawnieniami – wystarczy otworzyć przeglądarkę w trybie normalnym oraz w trybie prywatnym (aka incognito) lub dwie różne przeglądarki
2. Będąc zalogowanym jako superuser w jednym z okien przeglądarki, można zmieniać uprawnienia użytkownika zalogowanego w drugim oknie (zamiast tworzyć kilka kont)