Zarządzanie kontem użytkownika

Uprawnienia użytkowników (powtórzenie)

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

Zarządzanie kontem

- Określenie "zarządzanie kontem" będzie dalej oznaczało operacje dotyczące konta użytkownika aplikacji Django,
  które może wykonać sam użytkownik. Należą do nich:
  - Rejestracja – samodzielne założenie konta użytkownika aplikacji prze wypełnienie formularza rejestracji
  - Logowanie i wylogowanie
  - Zmiana hasła
  - Resetowanie hasła – podanie adresu e-mail w formularzu, odebranie wiadomości z odnośnikiem do formularza, w którym można wpisać nowe hasło, wpisanie tam nowego hasła
- Niektóre operacje zarządzania kontem są dostępne przez panel administracyjny (logowania, wylogowanie, zmiana hasła), ale tylko dla użytkowników posiadających uprawnienie "w zespole"

Możliwości zarządzania kontem w Django

- Django umożliwia wyposażenie aplikacji w operacje zarządzania kontem na kilka różnych sposobów – daje to dużą elastyczność dopasowania aplikacji do oczekiwań użytkowników;
- Możliwe warianty działania:
  1. Wykorzystanie formularzy, funkcji widoku i szablonów Django
  2. Wykorzystanie formularzy i funkcji widoku Django;
     własne szablony
  3. Wykorzystanie formularzy Django;
     własne funkcje widoku i szablony
  4. Własne formularze, funkcje widoku i szablony
- Inne warianty (np. widoki Django, formularze własne)
  nie mają sensu – formularz i tak musi mieć pola, których oczekuje widok, nie ma uzasadnienia tworzenie własnego

Wariant 1: wykorzystanie formularzy, funkcji widoku i szablonów Django

- Panel administracyjny Django udostępnia wybrane (ważniejsze) elementy zarządzania kontem:
    - Logowanie
    - Wylogowanie
    - Zmiana hasła
- Zalety:
    - Dodawane do aplikacji razem z panelem podczas tworzenia aplikacji przez kreator Django – zerowy nakład pracy
    - Stabilne i pewne działanie
    - Obsługa błędów
- Wady
    - Nie wszystkie operacje zarządzania kontem są dostępne (nie rejestracji użytkowników i resetowania hasła)
    - Szablon panelu Django (układ, styl, kolory, …) – wygląd może znacznie różnić się od wyglądu aplikacji;
      Można to zmienić, ale jest to trudne, metoda 2 jest prostsza
    - Przełączanie pomiędzy aplikacją a panelem np. w czasie logowania – niekoniecznie dobre wrażenie
    - Użytkownik musi mieć prawo "w zespole" – inaczej nie ma dostępu do panelu i nawet przy poprawnym logowaniu pojawia się komunikat o błędzie
- Zastosowanie
    - Nieliczni użytkownicy, wszyscy mają dostęp do panelu;
    - Użytkownicy spoza zespołu nie mają możliwości rejestracji, nie uczestniczą w tworzeniu strony (np. przez komentarze) albo robią to anonimowo
    - Przykłady: strony prywatne, proste strony typu "portfolio" prowadzone na zlecenie
- Implementacja – wystarczy dodać odnośniki do istniejących stron panelu; Ważne: nie należy nic dodawać do plików urls.py; Bezpośrednio po zalogowaniu lub wylogowaniu wyświetlana jest strona panelu (trzeba mieć prawo "w zespole")

```
<a href='/admin/login'>Login</a>
<a href='/admin/logout'>Logout</a>
```

- Można dodać przekierowanie po skutecznym zalogowaniu lub wylogowaniu, dopisując w adresie ?next=<url-podstrony>
  Pomimo przekierowania trzeba mieć prawo "w zespole"

```
<a href='/admin/login?next=/main'>Login</a>
<a href='/admin/logout?next=/'>Logout</a>
```

Wariant 2: wykorzystanie formularzy i funkcji widoku Django; własne szablony

- Moduł django.contrib.auth definiuje funkcje widoku dla operacji zarządzania kontem, które mają też własne formularze – wystarczy dostarczyć szablon, aby z nich skorzystać:
    - Logowanie
    - Wylogowanie
    - Zmiana hasła
      (dwie funkcje: PasswordChange i PasswordChangeDone)
    - Resetowanie hasła
      (cztery funkcje: PasswordReset, PasswordResetDone, PasswordResetConfirm i PasswordResetComplete)
- Zalety:
    - Wystarczy dostarczyć szablon – niewielki nakład pracy
    - Stabilne i pewne działanie
    - Obsługa błędów
    - Wygląd zgodny z wyglądem aplikacji
- Wady
    - Nie wszystkie operacje zarządzania kontem są dostępne
      (nie ma rejestracji użytkowników)
    - Resetowanie hasła wymaga skonfigurowania serwera pocztowego i konta dla Django
- Zastosowanie
    - Niezbyt liczni użytkownicy – osoby spoza zespołu nie mają możliwości samodzielnej rejestracji, administrator musi ich dopisać
    - Tylko wybrani mają dostęp do panelu – korzystanie z panelu nie jest niezbędne, o ile aplikacja dostarcza wymagane funkcjonalności (np. dodawanie i edycja treści)
    - Przykład: proste strony "portfolio", aktualizowane przez firmę we własnym zakresie, z niewielkim wsparciem z zewnątrz
- Implementacja – są dwa możliwe podejścia:
    1. Wykorzystanie domyślnych lokalizacji szablonów
       Podejście nieco łatwiejsze, ale ma wadę: ukrywa wbudowane strony logowania i zmiany hasła panelu
    2. Jawne wskazanie lokalizacji szablonów
       Kilka linijek kodu więcej, ale pozwala zachować oryginalne strony logowania i zmiany hasła panelu

Wariant 2, Podejście 1 implementacji

- W pliku settings.py projektu trzeba zmienić kolejność aplikacji
  (Django szuka szablonów w kolejnych aplikacjach – jeżeli własna będzie na końcu, Django użyje wbudowanego szablonu)

```
INSTALLED_APPS = [
    'blog',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

- W pliku settings.py projektu trzeba dopisać URL przekierowania po pomyślnym zalogowaniu – brak tej informacji spowoduje błąd, ponieważ Django będzie próbować wyświetlić widok, accounts/profile, którego Django nie ma (można go dodać…);
- URL można dopisać gdziekolwiek, najlepiej na końcu pliku, gdzie powinien już być URL lokalizacji plików statycznych:

```
STATIC_URL = '/static/'
LOGIN_REDIRECT_URL = '/'
```

- W pliku urls.py projektu trzeba dyrektywę importu modułu django.contrib.auth i ścieżki logowania, wylogowania oraz zmiany hasła:

```
urlpatterns = [
  path('admin/', admin.site.urls),
  path('accounts/login/',
    views.LoginView.as_view(),
    name='login'),
  path('accounts/logout/',
    views.LogoutView.as_view(),
    name='logout'),
  path('', include('blog.urls')),
]
```

- Można zrezygnować z własnego szablonu strony wylogowania, dodając do wywołania widoku nazwany argument next_page:

```
urlpatterns = [
  path('admin/', admin.site.urls),
  path('accounts/login/',
    views.LoginView.as_view(),
    name='login'),
  path('accounts/logout/',
    views.LogoutView.as_view(next_page='/'),
    name='logout'),
  path('', include('blog.urls')),
]
```

- W folderze templates aplikacji utworzyć folder registration
  i umieścić w nim szablony strony logowania oraz (opcjonalnie) wylogowania; Należy zachować domyślne nazwy folderów i plików

```
blog
-- templates
   -- registration
      -- login.html
      -- logged_out.html
```

- Szablon **login.html** powinien być skonstruowany podobnie
  jak szablon strony formularza dodawania/edycji wpisu;
  Kontekst szablonu zawiera formularz (o nazwie "form"), który trzeba wyświetlić analogicznie jak formularz wpisu;

```
<form method="post" action="{% url 'login' %}">
  {% csrf_token %}
  {{ form.as_p }}
  <p>
    <label></label>
    <button type="submit" >Zaloguj</button>
  </p>
  <input type="hidden" name="next" value="{{ next }}">
</form>
```

- Dane formularza powinny być wysłane do tej samego widoku, która go wyświetla (pod ten sam adres URL), zatem atrybut action formularza może zawierać pusty łańcuch (action="") albo jawnie podany adres strony logowania:

```
<form method="post" action="{% url 'login' %}">
  {% csrf_token %}
  {{ form.as_p }}
  <p>
    <label></label>
    <button type="submit" >Zaloguj</button>
  </p>
  <input type="hidden" name="next" value="{{ next }}">
</form>
```

- Dodatkowe ukryte pole (hidden) zawiera adres przekierowania, wykorzystywane np. kiedy formularz logowania zostanie wyświetlony np. w efekcie użycia dekoratora @login_required
  (zmienną next dodaje do kontekstu funkcja widoku)

```
<form method="post" action="{% url 'login' %}">
  {% csrf_token %}
  {{ form.as_p }}
  <p>
    <label></label>
    <button type="submit" >Zaloguj</button>
  </p>
  <input type="hidden" name="next" value="{{ next }}">
</form>
```

- Szablon **logged_out.html** jest wyświetlany po udanym wylogowaniu; Może zawierać dodatkowe informacje, przekazywane przez argument nazwany extra_context, zazwyczaj zawiera odnośnik do ponownego zalogowania się:

```
<h2>Pomyślnie wylogowano z serwisu</h2>
<p>
  Kliknij <a href="{% url 'login' %}">tutaj</a>
  aby zalogować się ponownie
</p>
```

- Wbudowana aplikacja accounts dostarcza funkcje widoków oraz formularze również do zmiany i resetowania hasła:
    - Widok i formularz zmiany hasła
    - Widok potwierdzenia zmiany hasła
    - Widok i formularz resetowania hasła
    - Widok potwierdzenia resetowania hasła
    - Widok i formularz ustawienia nowego hasła
      (odnośnik do tego formularza Django wysyła mailem)
    - Widok potwierdzenia ustawienia nowego hasła

- Nazwy funkcji widoków, adresy URL i nazwy szablonów można znaleźć w dokumentacji Django:
  https://docs.djangoproject.com/en/3.1/topics/auth/default/#module-django.contrib.auth.forms
- Opisy implementacji tych funkcjonalności można znaleźć również w poradnikach, np.:
  Django Girls Tutorial: Extensions, https://tutorial-extensions.djangogirls.org/en/authentication_authorization,
  Mozilla Developer Network, https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Authentication,
  Ultimate Django, https://ultimatedjango.com/learn-django/lessons/create-the-login-template/

- Formularz zmiany hasła
  (szablon wyświetla formularz "form", jak przy logowaniu):
    - URL: accounts/password_change/
    - widok: views.PasswordChangeView.as_view()
    - szablon: password_change_form.html
    - Nazwa URL: password_change
- Strona potwierdzenia zmiany hasła
  (szablon wyświetla tylko komunikat, jak przy wylogowaniu):
    - URL: accounts/password_change/done/
    - widok: views.PasswordChangeDoneView.as_view()
    - szablon: password_change_done.html
    - Nazwa URL: password_change_done

- Dla domyślnych nazw i lokalizacji szablonów, odpowiedni fragment pliku urls.py projektu powinien zawierać kod jak poniżej. Podobnie realizuje się resetowanie hasła, ale do tego niezbędny jest serwer poczty…

```
urlpatterns = [
  #...
  path('accounts/password_change/',
    views.PasswordChangeView.as_view(),
    name='password_change'),
  path('accounts/password_change/done/',
    views.PasswordChangeDoneView.as_view(),
    name='password_change_done'),
  #...
]
```

- Wadą tego podejścia jest "przysłonięcie" wbudowanych stron logowania, wylogowania i zmiany hasła, które normalnie są dostępne w panelu administracyjnym; Kilka drobnym zmian w kodzie pozwala zachować wbudowane strony panelu administracyjnego, a jednocześnie udostępnić własne strony w aplikacji

Wariant 2, Podejście 2 implementacji

- W pliku settings.py projektu trzeba zmienić kolejność aplikacji
  – tym razem własna aplikacja na końcu, aby nie przesłoniła elementów wbudowanych Django:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog',
]
```

- W pliku urls.py projektu trzeba dodatkowo użyć opcjonalnego argumentu template_name i wskazać lokalizację szablonu;

```
urlpatterns = [
  #...
  path('accounts/login/',
    views.LoginView.as_view(
      template_name='accounts/login.html'
    ),
    name='login'),
```

- Ważne! (1) Lokalizacja szablonu musi być inna, niż domyślna, inaczej Django weźmie szablony panelu administracyjnego; (2) Ścieżki szablonów są względne (od <aplikacja>/templates/)
- Można też dodać stronę "profilu" użytkownika, wyświetlaną tuż po zalogowaniu
- Django definiuje tylko URL strony profilu oraz realizuje przekierowanie na ten URL po zalogowaniu, natomiast funkcję widoku i szablon należy zdefiniować we własnym zakresie;
- W pliku settings.py projektu trzeba usunąć URL przekierowania po pomyślnym zalogowaniu, inaczej strona profilu nie zostanie wyświetlona:

```
STATIC_URL = '/static/'
#LOGIN_REDIRECT_URL = '/'
```

- Funkcję widoku można dodać do własnej aplikacji, jednak aby była widoczna z pliku urls.py projektu, trzeba ją zaimportować:

```
from blog.views import profile

urlpatterns = [
  #...
  path('accounts/profile/',
    profile,
    name='profile'),
  #...
]
```

Wariant 3: wykorzystanie formularzy Django, własne funkcje widoku i szablony

- Moduł django.contrib.auth.forms definiuje formularze wykorzystywane przez wbudowane funkcje widoku zarządzania kontem, ale można napisać do nich własne funkcje widoku:
    - AuthenticationForm
    - PasswordChangeForm
    - PasswordResetForm
    - SetPasswordForm
    - UserCreationForm – formularz dodawania/rejestracji użytkownika; niestety nie ma wbudowanej funkcji widoku
- Django dostarcza funkcje związane z zarządzaniem kontami, trzeba tylko wywołać funkcję i ewentualnie sprawdzić rezultat
- Zalety:
    - Możliwość zmiany domyślnych działań Django,
      np. blokowanie konta po kilku nieudanych próbach logowania, logowanie dwuetapowe (potwierdzenie kodem, email lub SMS)
    - Możliwość samodzielnego rejestrowania się użytkowników
    - Wygląd zgodny z wyglądem aplikacji
- Wady
    - Większy nakład pracy
    - Możliwe błędy i niestabilne działanie – zależy od implementacji funkcji widoku
- Zastosowanie
    - Liczni użytkownicy – osoby spoza zespołu mają możliwość samodzielnej rejestracji, administrator nie musi ich dopisać ręcznie; Domyślne uprawnienia użytkowników (nadawane przy rejestracji) mogą być później zmienione przez administratora
    - Tylko wybrani mają dostęp do panelu – korzystanie z panelu nie jest niezbędne, o ile aplikacja dostarcza wymagane funkcjonalności (np. dodawanie i edycja treści)
    - Przykład: strony "portfolio" z możliwością rejestracji klientów, publiczne fora internetowe

Wariant 4: Własne formularze, funkcje widoku i szablony

- Można zdefiniować własne formularze od podstaw, ale można wykorzystać formularze modułu django.contrib.auth.forms jako klasy bazowe i na ich podstawie zdefiniować klasy potomne
- Własne formularze mogą zawierać dowolne dodatkowe pola (inaczej nie ma to sensu) – np. adres dostawy i dane do faktury w przypadku rejestracji użytkownika sklepu internetowego, zdjęcie profilowe na forum itp.
- Zalety:
    - Możliwość zmiany domyślnych formularzy Django oraz dodania nowych, np. dodanie captcha do rejestracji użytkowników
    - Dowolna zmiana funkcjonalności zarządzania kontem,
    - Wygląd zgodny z wyglądem aplikacji
- Wady
    - Znacznie większy nakład pracy
    - Możliwe błędy i niestabilne działanie – zależy od implementacji funkcji widoku
- Zastosowanie
    - Wszystkie możliwości z wariantu 3,
    - Pełna kontrola procesów zarządzania kontem, możliwość implementowania niestandardowych zabezpieczeń
    - Strony o podwyższonych wymaganiach funkcjonalnych lub bezpieczeństwa
    - Przykład: publiczne fora internetowe, sklepy internetowe, serwisy informacyjne

Zadania

Proszę uzupełnić projekt Django z poprzedniego ćwiczenia o elementy zarządzania kontem użytkownika – logowanie i wylogowanie z aplikacji oraz zmianę hasła, a w tym celu:

1. Zapisać kopię folderu projektu – na wypadek poważniejszych błędów i konieczności cofnięcia zmian
2. Dodać funkcjonalności logowania i wylogowania w sposób przedstawiony na wykładzie jako wariant 2 – wykorzystanie formularzy i funkcji widoku Django oraz własnych szablonów; podejście 1 implementacji, wg poradnika Django Girls Tutorial, Extensions: https://tutorial-extensions.djangogirls.org/en/authentication_authorization
3. Dodać funkcjonalność zmiany hasła użytkownika, a w tym odnośnik do zmiany hasła w menu, wyświetlany kiedy użytkownik jest zalogowany
4. Sprawdzić, czy oprócz własnych formularzy logowania i zmiany hasła dostępne są formularze panelu administracyjnego Django (np. po wpisaniu adresu /admin)
5. Zmienić funkcjonalności logowania i wylogowania w sposób przedstawiony na wykładzie jako wariant 2 – wykorzystanie formularzy i funkcji widoku Django oraz własnych szablonów, podejście 2 implementacji
6. Powtórzyć testy z punktu 4 – powinny być dostępne oba formularze, Django i własny
7. * Dodać szablon strony wylogowania (powinien zawierać tylko komunikat o pomyślnym wylogowaniu i odnośnik do ponownego zalogowania
8. * Dodać funkcję widoku i szablon strony profilu użytkownika, wyświetlanej przez Django po udanym logowaniu użytkownika (accounts/profile) – na początek wystarczy komunikat o udanym logowaniu
9. ** Rozszerzyć stronę profilu, przez wyświetlenie informacji dotyczących użytkownika (im więcej, tym lepiej), np. liczba wpisów i komentarzy, lista wpisów (np. tytuł, data dodania, liczba komentarzy, odnośnik do wyświetlenia, odnośnik do edycji), uprawnienia użytkownika (np. czy ma uprawnienia "w zespole" i "superuser", jakie ma uprawnienia CRUD dla modeli wpisów i komentarzy) itp.