Technologia i rozwiązania

Django

Praktyczne tworzenie aplikacji sieciowych

Django — framework dla perfekcjonistów, którzy muszą przestrzegać terminów!

Antonio Melé

Helion

[PACKT] PUBLISHING open source community experience distilled

Tytuł oryginału: Django By Example

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-2587-6

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/djptas
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 4

Utworzenie witryny społecznościowej

W poprzednim rozdziale dowiedziałeś się, jak opracować mapę witryny i kanał wiadomości dla postów bloga oraz zaimplementować silnik wyszukiwania w naszej aplikacji bloga. W tym rozdziale przechodzimy do opracowania aplikacji społecznościowej. Przygotujemy funkcjonalność pozwalającą użytkownikom na logowanie, wylogowanie oraz edytowanie i zerowanie hasła. Zobaczysz, jak można utworzyć niestandardowe profile dla użytkowników i jak zaimplementować uwierzytelnianie za pomocą innej witryny społecznościowej.

Oto zagadnienia, na których skoncentruję się w tym rozdziale.

- Użycie frameworka uwierzytelniania.
- Utworzenie widoków pozwalających na rejestrację użytkowników.
- Rozbudowa modelu `User` o obsługę niestandardowego profilu.
- Implementacja uwierzytelnienia społecznościowego za pomocą modułu `python-social-auth`.

Pracę rozpoczynamy od utworzenia nowego projektu.

Utworzenie projektu witryny społecznościowej

Przystępujemy teraz do budowy aplikacji społecznościowej umożliwiającej użytkownikom udostępnianie obrazów znalezionych w internecie. Na potrzeby tego projektu konieczne jest opracowanie pewnych komponentów. Oto one.

- System uwierzytelniania pozwalający użytkownikowi na rejestrowanie, logowanie, edycję profilu oraz zmianę i zerowanie hasła.
- System obserwacji pozwalający użytkownikom na śledzenie swoich poczynań.
- Funkcjonalność pozwalająca na wyświetlanie udostępnianych obrazów oraz implementacja bookmarkletu umożliwiającego użytkownikowi pobieranie obrazów z praktycznie każdej witryny internetowej.
- Strumień aktywności dla każdego użytkownika pozwalający użytkownikom śledzić treść dodawaną przez obserwowanych użytkowników.

W tym rozdziale zajmiemy się realizacją pierwszego z wymienionych punktów.

Rozpoczęcie pracy nad aplikacją społecznościową

Przejdź do powłoki i wydaj poniższe polecenia w celu utworzenia środowiska wirtualnego dla projektu, a następnie jego aktywacji.

```
$ mkdir env
$ virtualenv env
$ source env/bin/activate
```

Znak zachęty w powłoce wyświetla nazwę aktywnego środowiska wirtualnego, co pokazałem poniżej.

```
(env)laptop:~ zenx$
```

W przygotowanym środowisku wirtualnym zainstaluj framework Django, wydając poniższe polecenie.

```
$ pip install Django==1.8.6
```

Kolejnym krokiem jest utworzenie projektu, którego będziemy używać podczas prac nad aplikacją społecznościową. Przejdź do powłoki i wydaj poniższe polecenie.

```
$ django-admin startproject bookmarks
```

W ten sposób utworzymy nowy projekt Django o nazwie `bookmarks` wraz z początkową strukturą plików i katalogów. Teraz przejdź do nowego katalogu projektu i utwórz nową aplikację o nazwie `account`, wydając poniższe polecenia.

```
$ cd bookmarks/
$ django-admin startapp account
```

Pamiętaj, aby aktywować nową aplikację w projekcie poprzez dodanie jej do elementów wymienionych na liście `INSTALLED_APPS` w pliku *settings.py*. Naszą aplikację umieść na początku listy, przed pozostałymi zainstalowanymi aplikacjami, tak jak pokazałem poniżej.

```
INSTALLED_APPS = (
    'account',
    # ...
)
```

Nie zapomnij o wydaniu poniższego polecenia, aby przeprowadzić synchronizację bazy danych z modelami aplikacji domyślnych wskazanymi na liście `INSTALLED_APPS`.

```
$ python manage.py migrate
```

Teraz możemy już przystąpić do budowy systemu uwierzytelniania w projekcie, używając frameworka uwierzytelniania.

Użycie frameworka uwierzytelniania w Django

Django jest dostarczany wraz z wbudowanym frameworkiem uwierzytelniania, który może obsługiwać uwierzytelnianie użytkowników, sesje, uprawnienia i grupy użytkowników. System uwierzytelniania oferuje widoki dla działań najczęściej podejmowanych przez użytkowników, takich jak logowanie, wylogowanie, zmiana hasła i zerowanie hasła.

Wspomniany framework uwierzytelniania znajduje się w `django.contrib.auth` i jest używany także przez inne pakiety Django, typu `contrib`. Framework uwierzytelniania wykorzystaliśmy już w rozdziale 1. do utworzenia superużytkownika dla aplikacji bloga, aby mieć dostęp do witryny administracyjnej.

Kiedy tworzysz nowy projekt Django za pomocą polecenia `startproject`, framework uwierzytelniania zostaje wymieniony w domyślnych ustawieniach projektu. Składa się z aplikacji `django.contrib.auth` oraz przedstawionych poniżej dwóch klas wymienionych w opcji `MIDDLEWARE_CLASSES` projektu.

- `AuthenticationMiddleware`. Wiąże użytkowników z żądaniami za pomocą mechanizmu sesji.
- `SessionMiddleware`. Zapewnia obsługę bieżącej sesji między poszczególnymi żądaniami.

Oprogramowanie pośredniczące to klasa wraz z metodami wykonywanymi globalnie w trakcie fazy przetwarzania żądania lub udzielania odpowiedzi na nie. W tej książce klasy oprogramowania pośredniczącego będziemy wykorzystywać w wielu sytuacjach. Temat tworzenia oprogramowania pośredniczącego zostanie dokładnie omówiony w rozdziale 13.

Framework uwierzytelniania zawiera również wymienione poniżej modele.

- `User`. Model użytkownika wraz z podstawowymi kolumnami, takimi jak `username`, `password`, `email`, `first_name`, `last_name` i `is_active`.
- `Group`. Model grupy do nadawania kategorii użytkownikom.
- `Permission`. Uprawnienia pozwalające na wykonywanie określonych operacji.

Opisywany framework zawiera także domyślne widoki uwierzytelniania i formularze, z których będziemy korzystać nieco później.

Utworzenie widoku logowania

Rozpoczynamy od użycia wbudowanego w Django frameworka uwierzytelniania w celu umożliwienia użytkownikom zalogowania się w witrynie. Aby zalogować użytkownika, widok powinien wykonywać poniższe akcje.

1. Pobranie nazwy użytkownika i hasła z wysłanego formularza.
2. Uwierzytelnienie użytkownika na podstawie danych przechowywanych w bazie danych.
3. Sprawdzenie, czy konto użytkownika jest aktywne.
4. Zalogowanie użytkownika w witrynie i rozpoczęcie uwierzytelnionej sekcji.

Najpierw musimy przygotować formularz logowania. Utwórz nowy plik *forms.py* w katalogu aplikacji `account` i umieść w nim poniższy fragment kodu.

```
from django import forms

class LoginForm(forms.Form):
    username = forms.CharField()
    password = forms.CharField(widget=forms.PasswordInput)
```

Formularz będzie używany do uwierzytelnienia użytkownika na podstawie informacji przechowywanych w bazie danych. Zwróć uwagę na wykorzystanie widżetu `PasswordInput` do wygenerowania elementu HTML `<input>` wraz z atrybutem `type="password"`. Przeprowadź edycję pliku *views.py* aplikacji `account` i umieść w nim poniższy fragment kodu.

```
from django.http import HttpResponse
from django.shortcuts import render
from django.contrib.auth import authenticate, login
from .forms import LoginForm

def user_login(request):
    if request.method == 'POST':
```

```
            form = LoginForm(request.POST)
            if form.is_valid():
                cd = form.cleaned_data
                user = authenticate(username=cd['username'],
                                    password=cd['password'])
                if user is not None:
                    if user.is_active:
                        login(request, user)
                        return HttpResponse('Uwierzytelnienie zakończyło się sukcesem.')
                    else:
                        return HttpResponse('Konto jest zablokowane.')
                else:
                    return HttpResponse('Nieprawidłowe dane uwierzytelniające.')
        else:
            form = LoginForm()
        return render(request, 'account/login.html', {'form': form})
```

To jest kod podstawowego widoku logowania użytkownika. Po wywołaniu widoku `user_login` przez żądanie `GET` za pomocą wywołania `form = LoginForm()` tworzymy nowy egzemplarz formularza logowania i wyświetlamy go w szablonie. Kiedy użytkownik wyśle formularz przy użyciu żądania `POST`, przeprowadzane są następujące akcje.

1. Utworzenie egzemplarza formularza wraz z wysłanymi danymi. Do tego celu służy polecenie `form = LoginForm(request.POST)`.
2. Sprawdzenie, czy formularz jest prawidłowy. Jeżeli formularz jest nieprawidłowy, w szablonie wyświetlamy błędy wykryte podczas weryfikacji formularza (na przykład użytkownik nie wypełnił jednego z pól).
3. Jeżeli wysłane dane są prawidłowe, za pomocą metody `authenticate()` uwierzytelniamy użytkownika na podstawie informacji przechowywanych w bazie danych. Wymieniona metoda pobiera `username` i `password`, a zwraca obiekt `User`, gdy użytkownik zostanie uwierzytelniony, lub `None` w przeciwnym przypadku. Ponadto jeśli użytkownik nie będzie uwierzytelniony, zwracamy także obiekt `HttpResponse` wraz z odpowiednim komunikatem.
4. W przypadku pomyślnego uwierzytelnienia użytkownika za pomocą atrybutu `is_active` sprawdzamy, czy jego konto użytkownika jest aktywne. Wymieniony atrybut pochodzi z modelu `User` dostarczanego przez Django. Gdy konto użytkownika jest nieaktywne, zwracamy obiekt `HttpResponse` wraz z odpowiednim komunikatem.
5. Gdy konto użytkownika jest aktywne, logujemy go w witrynie internetowej. Rozpoczynamy także sesję dla użytkownika przez wywołanie metody `login()` i zwracamy odpowiedni komunikat informujący o powodzeniu operacji logowania.

> Zwróć uwagę na różnice między metodami `authenticate()` i `login()`. Metoda `authenticate()` sprawdza dane uwierzytelniające użytkownika i jeśli są prawidłowe, zwraca obiekt użytkownika. Natomiast metoda `login()` umieszcza użytkownika w bieżącej sesji.

Teraz musimy opracować wzorzec adresu URL dla nowo zdefiniowanego widoku. Utwórz nowy plik *urls.py* w katalogu aplikacji `account` i umieść w nim poniższy fragment kodu.

```
from django.conf.urls import url
from . import views

urlpatterns = [
    # Widoki logowania.
    url(r'^login/$', views.user_login, name='login'),
]
```

Przeprowadź edycję głównego pliku *urls.py* znajdującego się katalogu projektu `bookmarks` i dodaj wzorzec adresu URL aplikacji `account`, co przedstawiłem poniżej.

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^account/', include('account.urls')),
]
```

Widok logowania jest teraz dostępny za pomocą adresu URL. Przechodzimy więc do przygotowania szablonu dla tego widoku. Ponieważ w projekcie nie mamy jeszcze żadnych szablonów, najpierw musimy utworzyć szablon bazowy, który następnie będzie mógł być rozszerzony przez szablon logowania. Wymienioną poniżej strukturę plików i katalogów utwórz w katalogu aplikacji `account`.

```
templates/
    account/
        login.html
    base.html
```

Przeprowadź edycję pliku *base.html* i umieść w nim poniższy fragment kodu.

```
{% load staticfiles %}
<!DOCTYPE html>
<html>
<head>
  <title>{% block title %}{% endblock %}</title>
  <link href="{% static "css/base.css" %}" rel="stylesheet">
</head>
<body>
  <div id="header">
    <span class="logo">Bookmarks</span>
  </div>
  <div id="content">
    {% block content %}
    {% endblock %}
  </div>
</body>
</html>
```

W ten sposób przygotowaliśmy szablon bazowy dla budowanej witryny internetowej. Podobnie jak w poprzednim projekcie, także w tym style CSS dołączamy w szablonie głównym. Niezbędne pliki statyczne znajdziesz w materiałach przygotowanych dla książki. Wystarczy skopiować podkatalog *static* z katalogu *account* we wspomnianych materiałach i umieścić go w tym samym położeniu budowanego projektu.

Szablon bazowy definiuje bloki `title` i `content`, które mogą być wypełniane przez treść szablonów rozszerzających szablon bazowy.

Przechodzimy do utworzenia szablonu dla formularza logowania. W tym celu otwórz plik *account/login.html* i umieść w nim poniższy fragment kodu.

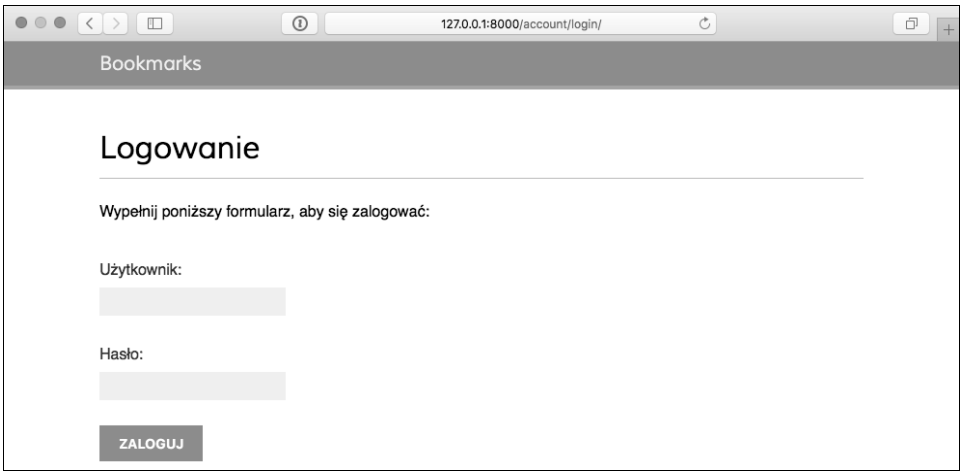
```
{% extends "base.html" %}

{% block title %}Logowanie{% endblock %}

{% block content %}
  <h1>Logowanie</h1>
  <p>Wypełnij poniższy formularz, aby się zalogować:</p>
  <form action="." method="post">
    {{ form.as_p }}
    {% csrf_token %}
    <p><input type="submit" value="Zaloguj"></p>
  </form>
{% endblock %}
```

Ten szablon zawiera formularz, którego egzemplarz jest tworzony w widoku. Ponieważ formularz zostanie wysłany za pomocą metody `POST`, dołączamy znacznik szablonu `{% csrf_token %}` w celu zapewnienia ochrony przed atakami typu CSRF. Więcej informacji na temat ataków CSRF przedstawiłem w rozdziale 2.

W bazie danych nie ma jeszcze żadnych kont użytkowników. Konieczne jest utworzenie najpierw superużytkownika, aby zapewnić sobie dostęp do witryny administracyjnej i zarządzać pozostałymi użytkownikami. Przejdź do powłoki i wydaj polecenie `python manage.py createsuperuser`. Podaj wybraną nazwę użytkownika, adres e-mail i hasło. Następnie uruchom serwer programistyczny przez wydanie polecenia `python manage.py runserver` i w przeglądarce internetowej przejdź pod adres *http://127.0.0.1:8000/admin/*. Dostęp do witryny administracyjnej uzyskasz po podaniu ustalonej przed chwilą nazwy użytkownika i hasła. Gdy znajdziesz się już w witrynie administracyjnej Django, zobaczysz modele `User` (łącze *Użytkownicy*) i `Group` (łącze *Grupy*) dla wbudowanego w Django frameworka uwierzytelniania. Stronę przeznaczoną do zarządzania użytkownikami i grupami pokazałem na rysunku 4.1.

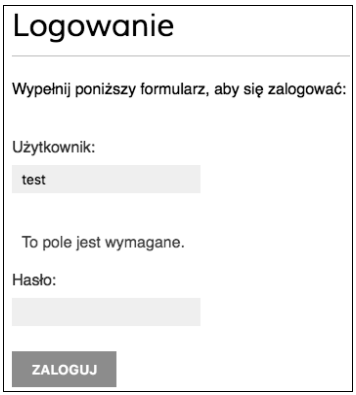
Utwórz nowego użytkownika, używając do tego witryny administracyjnej, a następnie w przeglądarce internetowej przejdź pod adres *http://127.0.0.1:8000/account/login/*. Powinieneś zobaczyć wygenerowany szablon wraz z formularzem logowania (patrz rysunek 4.2).

Rysunek 4.1. Strona przeznaczona do zarządzania użytkownikami i grupami



Rysunek 4.2. Szablon wraz z wyświetlonym formularzem logowania

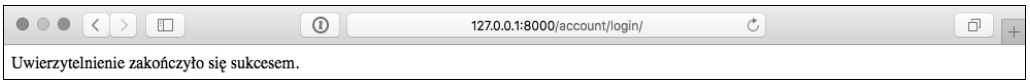
Spróbuj teraz wysłać formularz, pozostawiając niewypełnione jedno z pól. W takim przypadku formularz jest uznawany za nieprawidłowy i zostanie wyświetlony komunikat błędu, co pokazałem na rysunku 4.3.



Rysunek 4.3. Komunikat błędu wyświetlany po niewypełnieniu wymaganego pola formularza

Jeżeli podasz dane nieistniejącego użytkownika lub błędne hasło, Django wygeneruje komunikat o nieudanym logowaniu.

Natomiast po podaniu prawidłowych danych uwierzytelniających Django wyświetli komunikat o zakończonym sukcesem logowaniu, co pokazałem na rysunku 4.4.



Rysunek 4.4. Logowanie zakończone powodzeniem

Użycie widoków uwierzytelniania w Django

Framework uwierzytelniania w Django zawiera wiele formularzy i widoków gotowych do natychmiastowego użycia. Utworzony przed chwilą widok logowania to dobre ćwiczenie pomagające w zrozumieniu procesu uwierzytelniania użytkowników w Django. Jednak w większości przypadków możesz wykorzystać wspomniane domyślne widoki uwierzytelniania.

Do obsługi uwierzytelniania Django oferuje wymienione poniżej widoki.

- `login`. Obsługa formularza logowania oraz proces zalogowania użytkownika.
- `logout`. Obsługa wylogowania użytkownika.
- `logout_then_login`. Wylogowanie użytkownika, a następnie przeniesienie go na stronę logowania.

Do obsługi zmiany hasła Django oferuje wymienione poniżej widoki.

- `password_change`. Obsługa formularza pozwalającego użytkownikowi na zmianę hasła.
- `password_change_done`. Strona informująca o sukcesie operacji; zostanie wyświetlona użytkownikowi, gdy zmiana hasła zakończy się powodzeniem.

Natomiast do obsługi operacji zerowania hasła Django oferuje następujące widoki.

- `password_reset`. Umożliwienie użytkownikowi wyzerowania hasła. Generowane jest przeznaczone tylko do jednokrotnego użycia łącze wraz z tokenem, które następnie będzie wysłane na adres e-mail danego użytkownika.
- `password_reset_done`. Wyświetlenie użytkownikowi strony z informacją o wysłaniu wiadomości e-mail wraz z łączem pozwalającym na wyzerowanie hasła.
- `password_reset_confirm`. Widok umożliwiający użytkownikowi zdefiniowanie nowego hasła.
- `password_reset_complete`. Strona informująca o sukcesie operacji; zostanie wyświetlona użytkownikowi, gdy wyzerowanie hasła zakończy się powodzeniem.

Zastosowanie wymienionych wyżej widoków może zaoszczędzić sporą ilość czasu podczas tworzenia witryny internetowej obsługującej konta użytkowników. W widokach tych używane są wartości domyślne, które oczywiście można nadpisać. Przykładem może być wskazanie położenia szablonu przeznaczonego do wygenerowania lub formularza wyświetlanego przez widok.

Więcej informacji na temat wbudowanych widoków uwierzytelniania znajdziesz na stronie *https://docs.djangoproject.com/en/1.8/topics/auth/default/#module-django.contrib.auth.views*.

Widoki logowania i wylogowania

Przeprowadź edycję pliku *urls.py* aplikacji `account` i dodaj kolejne wzorce adresów URL. Po wprowadzeniu zmian zawartość wymienionego pliku powinna przedstawiać się następująco.

```
from django.conf.urls import url
from . import views

urlpatterns = [
    # Poprzedni widok logowania.
    # url(r'^login/$', views.user_login, name='login'),

    # Wzorce adresów URL dla widoków logowania i wylogowania.
    url(r'^login/$',
        'django.contrib.auth.views.login',
        name='login'),
    url(r'^logout/$',
        'django.contrib.auth.views.logout',
        name='logout'),
    url(r'^logout-then-login/$',
        'django.contrib.auth.views.logout_then_login',
        name='logout_then_login'),
]
```

Umieściliśmy znak komentarza na początku wiersza wzorca adresu URL dla utworzonego wcześniej widoku `user_login`. Teraz wykorzystamy widok `login` oferowany przez wbudowany w Django framework uwierzytelniania.

Utwórz nowy podkatalog w katalogu szablonów aplikacji `account` i nadaj mu nazwę *registration*. Podkatalog ten to domyślna lokalizacja, w której widoki uwierzytelniania Django spodziewają się znaleźć szablony. Teraz w nowym podkatalogu utwórz plik *login.html* i umieść w nim poniższy fragment kodu.

```
{% extends "base.html" %}

{% block title %}Logowanie{% endblock %}

{% block content %}
  <h1>Logowanie</h1>
  {% if form.errors %}
```

```
    <p>
      Nazwa użytkownika lub hasło są nieprawidłowe.
      Spróbuj ponownie.
    </p>
  {% else %}
    <p>Wypełnij poniższy formularz, aby się zalogować:</p>
  {% endif %}
  <div class="login-form">
    <form action="{% url 'login' %}" method="post">
      {{ form.as_p }}
      {% csrf_token %}
      <input type="hidden" name="next" value="{{ next }}" />
      <p><input type="submit" value="Zaloguj"></p>
    </form>
  </div>
{% endblock %}
```

Ten szablon logowania jest bardzo podobny do utworzonego wcześniej. Domyślnie Django używa formularza `AuthenticationForm` pochodzącego z `django.contrib.auth.forms`. Formularz próbuje uwierzytelnić użytkownika i zgłasza błąd weryfikacji, gdy logowanie zakończy się niepowodzeniem. W takim przypadku za pomocą znacznika szablonu `{% if form.errors %}` można przeanalizować te błędy, aby sprawdzić, czy podane zostały nieprawidłowe dane uwierzytelniające. Zwróć uwagę na dodanie ukrytego elementu HTML `<input>` przeznaczonego do wysłania wartości zmiennej o nazwie `next`. Zmienna jest ustawiania przez widok logowania, gdy w żądaniu będzie przekazany parametr `next` (na przykład *http://127.0.0.1:8000/account/login/?next=/account/*).

Wartością parametru `next` musi być adres URL. Jeżeli ten parametr zostanie podany, widok logowania w Django przekieruje użytkownika po zalogowaniu do podanego adresu URL.

Teraz utwórz szablon *logged_out.html* w katalogu *registration* i umieść w nim następujący fragment kodu.

```
{% extends "base.html" %}

{% block title %}Wylogowanie{% endblock %}

{% block content %}
  <h1>Wylogowanie</h1>
  <p>Zostałeś pomyślnie wylogowany. Możesz <a href="{% url
"login" %}">zalogować się ponownie</a>.</p>
{% endblock %}
```

Ten szablon zostanie przez Django wyświetlony po wylogowaniu użytkownika.

Po dodaniu wzorców adresu URL oraz szablonów dla widoków logowania i wylogowania budowana tutaj witryna internetowa jest gotowa na obsługę logowania użytkowników za pomocą oferowanych przez Django widoków uwierzytelniania.

> Zwróć uwagę, że widok `logout_then_login` podany w `urlconf` nie wymaga użycia żadnego szablonu, ponieważ przekierowuje użytkownika do widoku logowania.

Przystępujemy teraz do utworzenia nowego widoku przeznaczonego do wyświetlenia użytkownikowi panelu głównego (ang. *dashboard*) po tym, jak już zaloguje się w aplikacji. Otwórz plik *views.py* aplikacji `account` i umieść w nim poniższy fragment kodu.

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

@login_required
def dashboard(request):
    return render(request,
                  'account/dashboard.html',
                  {'section': 'dashboard'})
```

Widok został oznaczony dekoratorem `login_required` frameworka uwierzytelniania. Zadanie dekoratora `login_required` polega na sprawdzeniu, czy bieżący użytkownik został uwierzytelniony. Jeżeli użytkownik jest uwierzytelniony, następuje wykonanie udekorowanego widoku. Gdy natomiast użytkownik nie jest uwierzytelniony, zostaje przekierowany na stronę logowania, a adres URL, do którego próbował uzyskać dostęp, będzie podany jako wartość parametru `next` żądania `GET`. Tym samym po udanym logowaniu użytkownik powróci na stronę, do której wcześniej próbował uzyskać dostęp. Pamiętaj, że do obsługi tego rodzaju sytuacji dodaliśmy w szablonie logowania ukryty element `<input>`.

Zdefiniowaliśmy również zmienną `section`. Wykorzystamy ją do ustalenia, którą sekcję witryny obserwuje użytkownik. Wiele widoków może odpowiadać tej samej sekcji. To jest prosty sposób na zdefiniowanie, której sekcji odpowiadają poszczególne widoki.

Teraz należy utworzyć szablon dla widoku panelu głównego. Utwórz nowy plik w katalogu *templates/account/*, nadaj mu nazwę *dashboard.html* i umieść w nim przedstawiony poniżej kod.

```
{% extends "base.html" %}

{% block title %}Panel główny{% endblock %}

{% block content %}
  <h1>Panel główny</h1>
  <p>Witaj w panelu głównym.</p>
{% endblock %}
```

Kolejnym krokiem jest dodanie poniższego wzorca adresu URL dla nowego widoku. To zadanie przeprowadzamy w pliku *urls.py* aplikacji `account`.

```
urlpatterns = [
    # ...
    url(r'^$', views.dashboard, name='dashboard'),
]
```

Teraz przeprowadź edycję pliku *settings.py* projektu `bookmarks` i dodaj poniższy fragment kodu.

```
from django.core.urlresolvers import reverse_lazy

LOGIN_REDIRECT_URL = reverse_lazy('dashboard')
LOGIN_URL = reverse_lazy('login')
LOGOUT_URL = reverse_lazy('logout')
```

Oto wyjaśnienie działania poszczególnych opcji.

- `LOGIN_REDIRECT_URL`. Wskazujemy Django adres URL, do którego ma nastąpić przekierowanie, gdy widok `contrib.auth.views.login` nie otrzymuje parametru `next`.
- `LOGIN_URL`. Adres URL, do którego ma nastąpić przekierowanie po zalogowaniu użytkownika (na przykład za pomocą dekoratora `login_required`).
- `LOGOUT_URL`. Adres URL, do którego ma nastąpić przekierowanie po wylogowaniu użytkownika.

Do dynamicznego utworzenia adresów URL na podstawie ich nazw używamy funkcji `reverse_lazy()`. Wymieniona funkcja odwraca adres URL, podobnie jak `reverse()`, ale można ją wykorzystać, gdy zachodzi potrzeba odwrócenia adresu URL przed wczytaniem konfiguracji projektu.

Oto krótkie podsumowanie przeprowadzonych dotąd działań.

- Do naszego projektu dodaliśmy wbudowane we frameworku uwierzytelniania Django widoki logowania i wylogowania.
- Przygotowaliśmy własne szablony dla obu widoków i zdefiniowaliśmy prosty widok, do którego użytkownik zostanie przekierowany po zalogowaniu.
- Na koniec skonfigurowaliśmy ustawienia Django, aby wspomniane adresy URL były używane domyślnie.

Teraz do szablonu bazowego dodamy łącza logowania i wylogowania, co pozwoli na zebranie wszystkiego w całość.

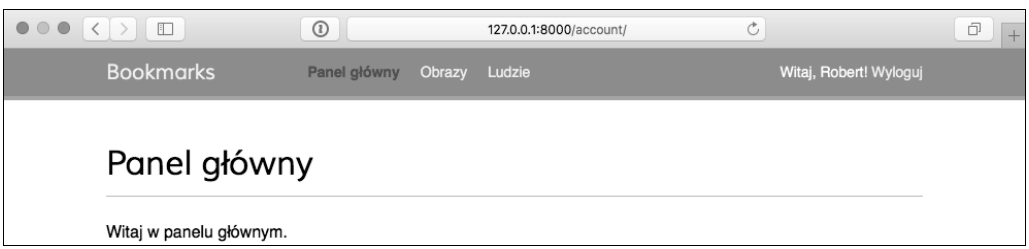
Konieczne jest ustalenie, czy bieżący użytkownik jest zalogowany, aby wyświetlić prawidłowe łącze (logowania lub wylogowania). Bieżący użytkownik jest przez oprogramowanie pośredniczące ustawiony w obiekcie `HttpRequest`. Dostęp do niego uzyskujesz za pomocą `request.user`. Użytkownika znajdziesz w wymienionym obiekcie nawet wtedy, gdy nie został uwierzytelniony. W takim przypadku użytkownik będzie zdefiniowany w postaci egzemplarza obiektu `AnonymousUser`. Najlepszym sposobem sprawdzenia, czy użytkownik został uwierzytelniony, jest wywołanie metody `request.user.is_authenticated()`.

Przeprowadź edycję pliku *base.html* i zmodyfikuj element `<div>` o identyfikatorze `header`, tak jak przedstawiłem poniżej.

```
<div id="header">
  <span class="logo">Bookmarks</span>
  {% if request.user.is_authenticated %}
    <ul class="menu">
      <li {% if section == "dashboard" %}class="selected"{% endif %}>
        <a href="{% url "dashboard" %}">Panel główny</a>
      </li>
      <li {% if section == "images" %}class="selected"{% endif %}>
        <a href="#">Obrazy</a>
      </li>
      <li {% if section == "people" %}class="selected"{% endif %}>
        <a href="#">Osoby</a>
      </li>
    </ul>
  {% endif %}

  <span class="user">
    {% if request.user.is_authenticated %}
      Witaj, {{ request.user.first_name }}!
      <a href="{% url "logout" %}">Wyloguj</a>
    {% else %}
      <a href="{% url "login" %}">Zaloguj</a>
    {% endif %}
  </span>
</div>
```

Jak możesz zobaczyć, menu witryny internetowej będzie wyświetlane jedynie uwierzytelnionym użytkownikom. Sprawdzana jest także bieżąca sekcja witryny, aby dodać klasę atrybutu `selected` do odpowiedniego elementu `<li>` i tym samym za pomocą CSS podświetlić nazwę aktualnej sekcji. Wyświetlane jest również imię uwierzytelnionego użytkownika i łącze pozwalające mu na wylogowanie. Jeżeli użytkownik nie jest uwierzytelniony, wyświetlone będzie jedynie łącze pozwalające mu na zalogowanie.

Teraz w przeglądarce internetowej przejdź pod adres *http://127.0.0.1:8000/account/login/*. Powinieneś zobaczyć stronę logowania. Podaj prawidłowe dane uwierzytelniające i kliknij przycisk *Zaloguj*. Po udanym logowaniu znajdziesz się na stronie pokazanej na rysunku 4.5.



Rysunek 4.5. Strona wyświetlana użytkownikowi po udanym logowaniu

Jak możesz zobaczyć, nazwa sekcji *Panel główny* została za pomocą stylów CSS wyświetlona innym kolorem czcionki, ponieważ odpowiadającemu jej elementowi `<li>` przypisaliśmy klasę `selected`. Skoro użytkownik jest uwierzytelniony, jego imię wyświetlamy po prawej stronie nagłówka. Kliknij łącze *Wyloguj*, powinieneś zobaczyć stronę pokazaną na rysunku 4.6.

Rysunek 4.6. Strona wyświetlana użytkownikowi po udanym wylogowaniu

Na tej stronie został wyświetlony komunikat informujący o udanym wylogowaniu i dlatego nie jest dłużej wyświetlane menu witryny internetowej. Łącze znajdujące się po prawej stronie nagłówka zmienia się na *Zaloguj*.

Jeżeli zamiast przygotowanej wcześniej strony wylogowania zostanie wyświetlona strona wylogowania witryny administracyjnej Django, sprawdź listę `INSTALLED_APPS` projektu i upewnij się, że wpis dotyczący aplikacji `django.contrib.admin` znajduje się po `account`. Oba wymienione szablony są umieszczone na tej samej względnej ścieżce dostępu i mechanizm wczytywania szablonów w Django po prostu użyje pierwszego znalezionego.

Widoki zmiany hasła

Użytkownikom witryny musimy zapewnić możliwość zmiany hasła po zalogowaniu się. Zintegrujemy więc oferowane przez framework uwierzytelniania Django widoki przeznaczone do obsługi procedury zmiany hasła. Otwórz plik *urls.py* aplikacji `account` i umieść w nim poniższe wzorce adresów URL.

```
    # Adresy URL przeznaczone do obsługi zmiany hasła.
    url(r'^password-change/$',
        'django.contrib.auth.views.password_change',
        name='password_change'),
    url(r'^password-change/done/$',
        'django.contrib.auth.views.password_change_done',
        name='password_change_done'),
```

Widok `password_change` zapewnia obsługę formularza pozwalającego na zmianę hasła, natomiast `password_change_done` wyświetla komunikat informujący o sukcesie po udanej operacji zmiany hasła przez użytkownika. Przystępujemy więc do przygotowania szablonu dla wymienionych widoków.

Dodaj nowy plik w katalogu *templates/registration* aplikacji `account` i nadaj mu nazwę *password_change_form.html*. Następnie w nowym pliku umieść poniższy fragment kodu.

```
{% extends "base.html" %}

{% block title %}Zmiana hasła{% endblock %}

{% block content %}
  <h1>Zmiana hasła</h1>
  <p>Wypełnij poniższy formularz, aby zmienić hasło.</p>
  <form action="." method="post">
    {{ form.as_p }}
    <p><input type="submit" value="Zmień"></p>
    {% csrf_token %}
  </form>
{% endblock %}
```

Przedstawiony szablon zawiera formularz przeznaczony do obsługi procedury zmiany hasła. Teraz w tym samym katalogu utwórz kolejny plik i nadaj mu nazwę *password_change_done.html*. Następnie w nowym pliku umieść poniższy fragment kodu.

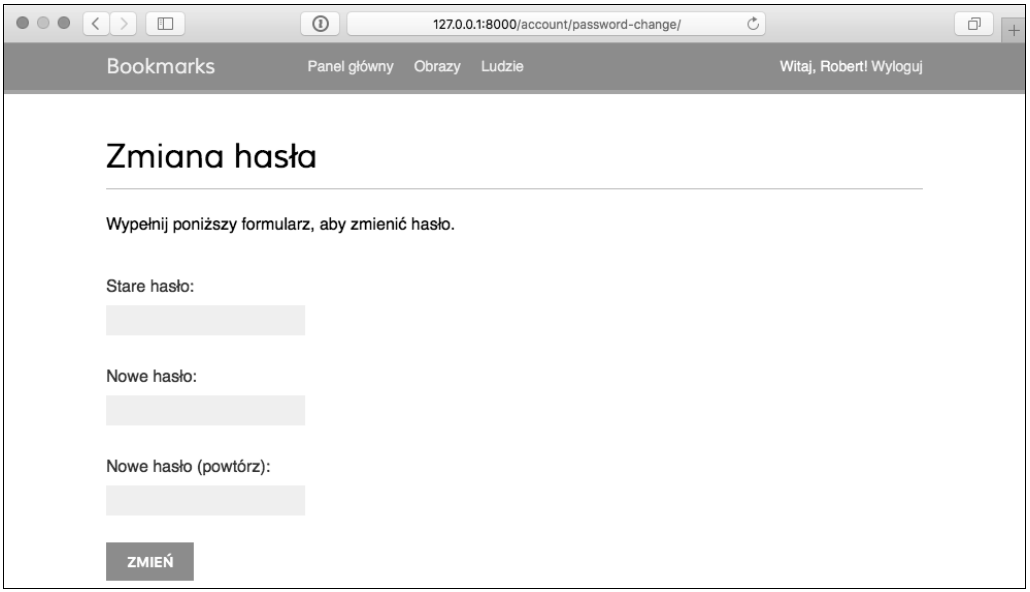
```
{% extends "base.html" %}

{% block title %}Hasło zostało zmienione{% endblock %}

{% block content %}
  <h1>Hasło zostało zmienione</h1>
  <p>Zmiana hasła zakończyła się powodzeniem.</p>
{% endblock %}
```

Ten szablon zawiera jedynie komunikat sukcesu wyświetlany, gdy przeprowadzona przez użytkownika operacja zmiany hasła zakończy się powodzeniem.

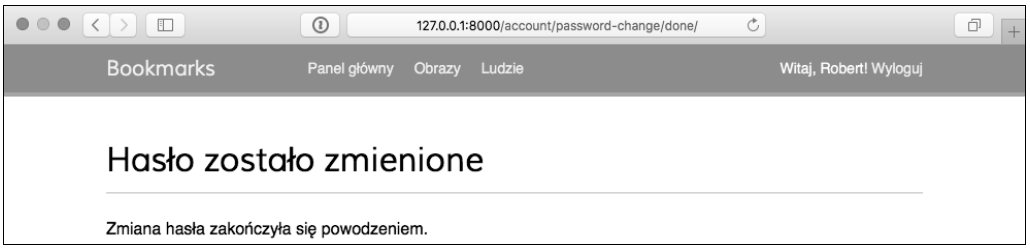
W przeglądarce internetowej przejdź pod adres *http://127.0.0.1:8000/account/password-change/*. Jeżeli użytkownik nie jest zalogowany, nastąpi przekierowanie na stronę logowania. Po udanym uwierzytelnieniu zobaczysz pokazany na rysunku 4.7 formularz pozwalający na zmianę hasła.

W wyświetlonym formularzu należy podać dotychczasowe hasło oraz dwukrotnie nowe, a następnie kliknąć przycisk *Zmień*. Jeżeli operacja przebiegnie bez problemów, zostanie wyświetlona strona wraz komunikatem informującym o sukcesie (patrz rysunek 4.8).

Wyloguj się i zaloguj ponownie za pomocą nowego hasła, aby sprawdzić, że wszystko działa zgodnie z oczekiwaniami.



Rysunek 4.7. Formularz pozwalający użytkownikowi na zmianę hasła



Rysunek 4.8. Komunikat informujący o udanej zmianie hasła

Widoki zerowania hasła

W pliku *urls.py* aplikacji `account` dodaj poniższe wzorce adresów URL dla widoków przeznaczonych do obsługi procedury zerowania hasła.

```
# Adresy URL przeznaczone do obsługi procedury zerowania hasła.
url(r'^password-reset/$',
    'django.contrib.auth.views.password_reset',
    name='password_reset'),
url(r'^password-reset/done/$',
    'django.contrib.auth.views.password_reset_done',
    name='password_reset_done'),
```

```
url(r'^password-reset/confirm/(?P<uidb64>[-\w]+)/(?P<token>[-\w]+)/$',
    'django.contrib.auth.views.password_reset_confirm',
    name='password_reset_confirm'),
url(r'^password-reset/complete/$',
    'django.contrib.auth.views.password_reset_complete',
    name='password_reset_complete'),
```

Dodaj nowy plik w katalogu *templates/registration/* aplikacji `account` i nadaj mu nazwę *password_reset_form.html*. Następnie w utworzonym pliku umieść poniższy fragment kodu.

```
{% extends "base.html" %}

{% block title %}Zerowanie hasła{% endblock %}

{% block content %}
  <h1>Zapomniałeś hasła?</h1>
  <p>Podaj adres e-mail, aby zdefiniować nowe hasło.</p>
  <form action="." method="post">
    {{ form.as_p }}
    <p><input type="submit" value="Wyślij e-mail"></p>
    {% csrf_token %}
  </form>
{% endblock %}
```

Teraz utwórz w tym samym katalogu kolejny plik, tym razem o nazwie *password_reset_email.html*. Następnie umieść w nim poniższy fragment kodu.

```
Otrzymaliśmy żądanie wyzerowania hasła dla użytkownika używającego adresu e-mail {{
↪email }}. Kliknij poniższe łącze:
{{ protocol }}://{{ domain }}{% url "password_reset_confirm"
uidb64=uid token=token %}
Twoja nazwa użytkownika: {{ user.get_username }}
```

Szablon ten zostanie użyty do wygenerowania wiadomości e-mail wysyłanej użytkownikowi, który chce przeprowadzić operację wyzerowania hasła.

Utwórz w tym samym katalogu kolejny plik i nadaj mu nazwę *password_reset_done.html*. Następnie umieść w nim poniższy fragment kodu.

```
{% extends "base.html" %}

{% block title %}Zerowanie hasła{% endblock %}

{% block content %}
  <h1>Zerowanie hasła</h1>
  <p>Wysłaliśmy Ci wiadomość e-mail wraz z instrukcjami pozwalającymi na
  ↪zdefiniowanie nowego hasła.</p>
  <p>Jeżeli nie otrzymałeś tej wiadomości, to upewnij się, że w formularzu zerowania
  ↪hasła wpisałeś adres e-mail podany podczas zakładania konta użytkownika.</p>
{% endblock %}
```

Utwórz kolejny plik szablonu, nadaj mu nazwę *password_reset_confirm.html*, a następnie umieść w nim poniższy fragment kodu.

```
{% extends "base.html" %}

{% block title %}Zerowanie hasła{% endblock %}

{% block content %}
  <h1>Zerowanie hasła</h1>
  {% if validlink %}
    <p>Dwukrotnie podaj nowe hasło:</p>
    <form action="." method="post">
      {{ form.as_p }}
      {% csrf_token %}
      <p><input type="submit" value="Zmień hasło" /></p>
    </form>
  {% else %}
    <p>Łącze pozwalające na wyzerowanie hasła jest nieprawidłowe, ponieważ
    ↪prawdopodobnie zostało już wcześniej użyte. Musisz ponownie rozpocząć
    ↪procedurę zerowania hasła.</p>
  {% endif %}
{% endblock %}
```

W kodzie sprawdzamy, czy podane łącze jest prawidłowe. Oferowany przez Django widok zerowania hasła ustawia zmienną i umieszcza ją w kontekście szablonu. Jeżeli łącze jest prawidłowe, wtedy wyświetlamy użytkownikowi formularz wyzerowania hasła.

Utwórz kolejny plik szablonu i nadaj mu nazwę *password_reset_complete.html*. Następnie umieść w nim poniższy fragment kodu.

```
{% extends "base.html" %}

{% block title %}Zerowanie hasła{% endblock %}

{% block content %}
  <h1>Zerowanie hasła</h1>
  <p>Hasło zostało zdefiniowane. Możesz się już <a href="{% url "login"
%}">zalogować</a>.</p>
{% endblock %}
```

Na koniec przeprowadź edycję szablonu *registration/login.html* aplikacji `account` i dodaj poniższy fragment kodu po elemencie `<form>`.

```
<p><a href="{% url "password_reset" %}">Zapomniałeś hasła?</a></p>
```

Teraz w przeglądarce internetowej przejdź pod adres *http://127.0.0.1:8000/account/login/* i kliknij łącze *Zapomniałeś hasła?*. Powinieneś zobaczyć stronę pokazaną na rysunku 4.9.

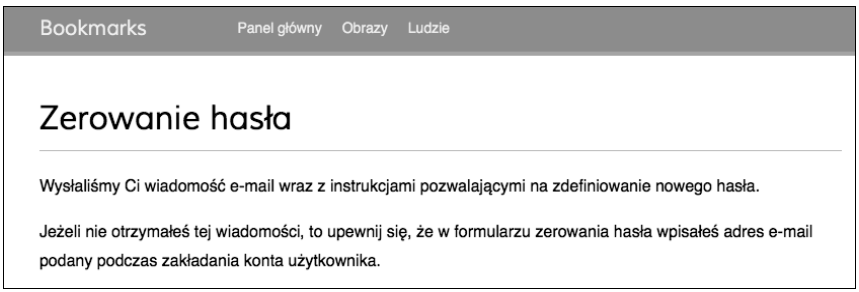
Rysunek 4.9. Strona pozwalająca na rozpoczęcie procedury wyzerowania hasła

Na tym etapie w pliku *settings.py* projektu trzeba umieścić konfigurację serwera SMTP, aby umożliwić Django wysyłanie wiadomości e-mail. Procedura dodania tego rodzaju konfiguracji do projektu została omówiona w rozdziale 2. Jednak podczas pracy nad aplikacją można skonfigurować Django do przekazywania wiadomości e-mail na standardowe wyjście zamiast ich faktycznego wysyłania za pomocą serwera SMTP. Framework Django oferuje mechanizm wyświetlania wiadomości e-mail w powłoce. Przeprowadź edycję pliku *settings.py* projektu i dodaj w nim poniższy wiersz kodu.

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

Opcja `EMAIL_BACKEDN` wskazuje na użycie klasy przeznaczonej do wysyłania wiadomości e-mail.

Wróć do przeglądarki internetowej, podaj adres e-mail istniejącego użytkownika i kliknij przycisk *Wyślij e-mail*. Powinieneś zobaczyć stronę pokazaną na rysunku 4.10.



Rysunek 4.10. Komunikat potwierdzający wysłanie wiadomości e-mail wraz z opisem procedury wyzerowania hasła

Spójrz na powłokę, w której został uruchomiony serwer programistyczny. Powinieneś w niej zobaczyć wygenerowaną wiadomość e-mail.

```
IME-Version: 1.0
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: 7bit
Subject: Password reset on 127.0.0.1:8000
From: webmaster@localhost
To: user@domain.com
Date: Thu, 24 Sep 2015 14:35:08 -0000
Message-ID: <20150924143508.62996.55653@zenx.local>

Otrzymaliśmy żądanie wyzerowania hasła dla użytkownika używającego adresu e-mail
nazwa_użytkownika@nazwa_domeny.pl. Kliknij poniższe łącze:
http://127.0.0.1:8000/account/password-reset/confirm/MQ/45f9c3f30caafd523055fcc/
Twoja nazwa użytkownika: zenx
```

Ta wiadomość e-mail jest generowana za pomocą utworzonego wcześniej szablonu *password_reset_email.html*. Adres URL pozwalający na przejście do strony zerowania hasła zawiera token dynamicznie wygenerowany przez Django. Po otworzeniu w przeglądarce internetowej otrzymanego łącza przejdziesz na stronę pokazaną na rysunku 4.11.

Rysunek 4.11. Strona pozwalająca na wyzerowanie hasła

To jest strona umożliwiająca użytkownikowi podanie nowego hasła; odpowiada ona szablonowi *password_reset_confirm.html*. W obu polach formularza wpisz nowe hasło, a następnie kliknij przycisk *Zmień hasło*. Django utworzy nowe zaszyfrowane hasło i zapisze je w bazie danych. Następnie zostanie wyświetlona pokazana na rysunku 4.12 strona wraz z komunikatem informującym o sukcesie operacji.

Rysunek 4.12. Operacja wyzerowania hasła zakończyła się powodzeniem

Teraz użytkownik może zalogować się na swoje konto, podając nowe hasło. Każdy token przeznaczony do ustawienia nowego hasła może być użyty tylko jednokrotnie. Jeżeli ponownie otworzysz w przeglądarce internetowej otrzymane łącze, zostanie wyświetlony komunikat informujący o nieprawidłowym tokenie.

W ten sposób w projekcie zintegrowałeś widoki oferowane przez framework uwierzytelniania w Django. Wspomniane widoki są odpowiednie do użycia w większości sytuacji. Jednak zawsze możesz utworzyć własne widoki, jeśli potrzebna jest obsługa niestandardowego zachowania.

Rejestracja użytkownika i profil użytkownika

Istniejący użytkownicy mogą się zalogować, wylogować, zmienić hasło lub je wyzerować, jeśli zapomnieli, jakie było. Musimy teraz przygotować widok pozwalający nowym odwiedzającym witrynę na założenie w niej konta użytkownika.

Rejestracja użytkownika

Przystępujemy do utworzenia prostego widoku pozwalającego odwiedzającemu na zarejestrowanie się w naszej witrynie internetowej. Zaczniemy od formularza, w którym nowy użytkownik wprowadzi nazwę użytkownika, swoje imię i nazwisko oraz hasło. Przeprowadź edycję pliku *forms.py* w katalogu aplikacji `account` i umieść w nim poniższy fragment kodu.

```
from django.contrib.auth.models import User

class UserRegistrationForm(forms.ModelForm):
    password = forms.CharField(label='Hasło',
                               widget=forms.PasswordInput)
    password2 = forms.CharField(label='Powtórz hasło',
                                widget=forms.PasswordInput)

    class Meta:
        model = User
        fields = ('username', 'first_name', 'email')
```

```
    def clean_password2(self):
        cd = self.cleaned_data
        if cd['password'] != cd['password2']:
            raise forms.ValidationError('Hasła nie są identyczne.')
        return cd['password2']
```

Utworzyliśmy formularz modelu (klasa `ModelForm`) dla modelu `User`. W przygotowanym formularzu będą uwzględnione jedynie pola `username`, `first_name` i `email`. Wartości wymienionych pól będą weryfikowane na podstawie odpowiadających im kolumn modelu. Jeśli na przykład użytkownik wybierze już istniejącą nazwę użytkownika, otrzyma błąd w trakcie weryfikacji formularza. Dodaliśmy dwa dodatkowe pola `password` i `password2` przeznaczone do zdefiniowania hasła i jego potwierdzenia. Ponadto zdefiniowaliśmy metodę `clean_password2()` odpowiedzialną za porównanie obu wpisanych haseł. Jeżeli nie są takie same, formularz będzie uznany za nieprawidłowy. Ta operacja sprawdzenia nowego hasła jest przeprowadzana podczas weryfikacji formularza za pomocą jego metody `is_valid()`. Istnieje możliwość dostarczenia metody `clean_<nazwa_pola>()` dla dowolnego pola formularza w celu wyczyszczenia jego wartości lub zgłoszenia błędu weryfikacji formularza dla określonego pola. Formularze zawierają także ogólną metodę `clean()` przeznaczoną do sprawdzenia całego formularza, co okazuje się użyteczne podczas weryfikacji pól zależnych wzajemnie od siebie.

Django oferuje również formularz `UserCreationForm` gotowy do natychmiastowego użycia. Znajdziesz go w `django.contrib.auth.forms`, a sam formularz jest bardzo podobny do utworzonego przez nas wcześniej.

Przeprowadź edycję pliku *views.py* aplikacji `account` i umieść w nim poniższy fragment kodu.

```
from .forms import LoginForm, UserRegistrationForm

def register(request):
    if request.method == 'POST':
        user_form = UserRegistrationForm(request.POST)
        if user_form.is_valid():
            # Utworzenie nowego obiektu użytkownika, ale jeszcze nie zapisujemy go w bazie danych.
            new_user = user_form.save(commit=False)
            # Ustawienie wybranego hasła.
            new_user.set_password(
                user_form.cleaned_data['password'])
            # Zapisanie obiektu User.
            new_user.save()
            return render(request,
                          'account/register_done.html',
                          {'new_user': new_user})
    else:
        user_form = UserRegistrationForm()
    return render(request,
                  'account/register.html',
                  {'user_form': user_form})
```

Widok pozwalający na utworzenie nowego konta użytkownika jest całkiem prosty. Zamiast zapisywać wprowadzone przez użytkownika hasło w postaci zwykłego tekstu, wykorzystujemy metodę `set_password()` modelu `User`, która ze względów bezpieczeństwa szyfruje hasło.

Teraz przeprowadź edycję pliku *urls.py* aplikacji `account` i dodaj w nim poniższy wzorzec adresu URL.

```
url(r'^register/$', views.register, name='register'),
```

Na koniec utwórz nowy szablon w katalogu szablonów aplikacji `account`, nadaj mu nazwę *register.html*, a następnie umieść w nim poniższy kod.

```
{% extends "base.html" %}

{% block title %}Utwórz konto{% endblock %}

{% block content %}
  <h1>Utwórz konto</h1>
  <p>Wypełnij poniższy formularz, aby się zarejestrować:</p>
<form action="." method="post">
    {{ user_form.as_p }}
    {% csrf_token %}
    <p><input type="submit" value="Utwórz konto"></p>
  </form>
{% endblock %}
```

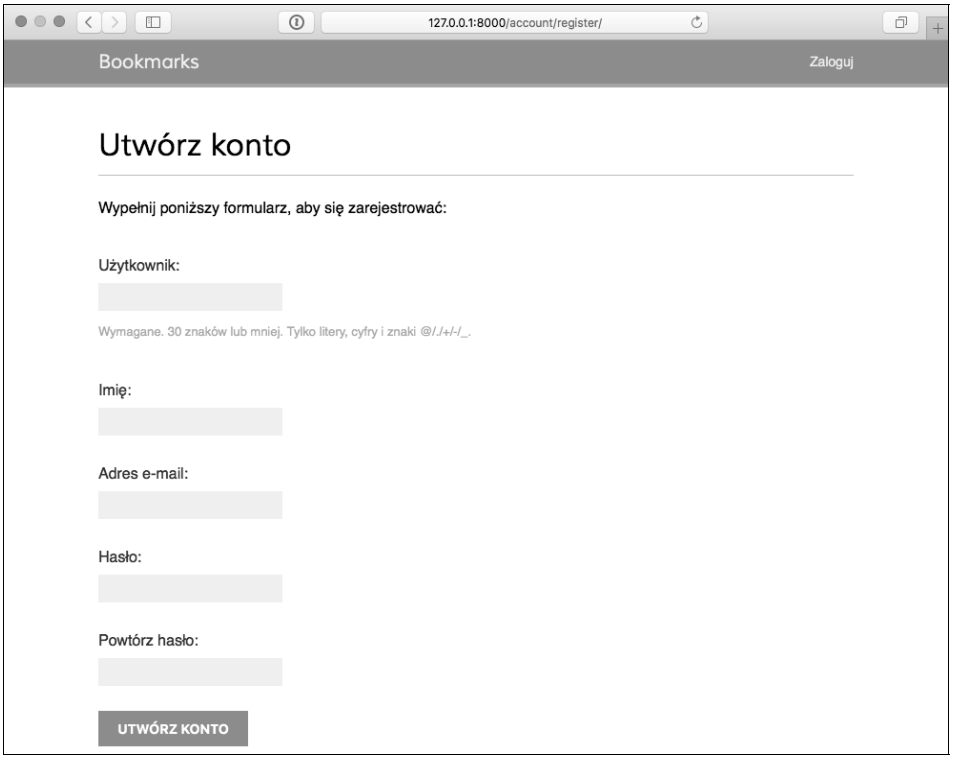
Do tego samego katalogu dodaj nowy plik szablonu o nazwie *register_done.html*. Następnie umieść w nim poniższy fragment kodu.

```
{% extends "base.html" %}

{% block title %}Witaj{% endblock %}

{% block content %}
  <h1>Witaj, {{ new_user.first_name }}!</h1>
  <p>Twoje konto zostało utworzone. Możesz się już <a
href="{% url "login" %}">zalogować</a>.</p>
{% endblock %}
```

Teraz w przeglądarce internetowej przejdź pod adres *http://127.0.0.1:8000/account/register/*. Powinieneś zobaczyć wyświetloną stronę rejestracji nowego użytkownika (patrz rysunek 4.13).

Podaj informacje potrzebne do utworzenia nowego konta użytkownika i kliknij przycisk *Utwórz konto*. Jeżeli wszystkie pola zostały wypełnione prawidłowo, zostanie wyświetlona strona wraz z komunikatem informującym o pomyślnym utworzeniu nowego konta użytkownika (patrz rysunek 4.14).

Kliknij łącze *Zaloguj*, a następnie podaj dane uwierzytelniające utworzonego przed chwilą użytkownika, aby potwierdzić, że możesz uzyskać dostęp do nowego konta.



Rysunek 4.13. Strona wyświetlająca formularz pozwalający na utworzenie nowego konta użytkownika

Rysunek 4.14. Utworzenie nowego konta użytkownika zakończyło się powodzeniem

Teraz do szablonu logowania musimy dodać łącze pozwalające na rejestrację użytkownika. Przeprowadź edycję szablonu *registration/login.html* i poniższy wiersz kodu

```
<p>Wypełnij poniższy formularz, aby się zalogować:</p>
```

zastąp następującym.

```
<p>Wypełnij poniższy formularz, aby się zalogować. Jeżeli nie masz jeszcze
konta, możesz je utworzyć <a href="{% url "register" %}">tutaj</a>.</p>
```

W ten sposób do strony rejestracji nowego konta użytkownika można przejść bezpośrednio ze strony logowania.

Rozbudowa modelu User

Podczas pracy z kontami użytkowników przekonasz się, że model `User` oferowany przez framework uwierzytelniania Django sprawdza się w większości przypadków. Jednak model `User` jest dostarczany wraz z jedynie najbardziej podstawowymi kolumnami. Dlatego też prawdopodobnie będzie trzeba niekiedy rozbudować model o możliwość przechowywania dodatkowych danych. Najlepszym sposobem będzie przygotowanie modelu profilu zawierającego wszystkie dodatkowe kolumny oraz związek typu „jeden do jednego" z dostarczanym przez Django modelem `User`.

Przeprowadź edycję pliku *models.py* aplikacji `account` i umieść w nim poniższy fragment kodu.

```
from django.db import models
from django.conf import settings

class Profile(models.Model):
    user = models.OneToOneField(settings.AUTH_USER_MODEL)
    date_of_birth = models.DateField(blank=True, null=True)
    photo = models.ImageField(upload_to='users/%Y/%m/%d',
                              blank=True)

    def __str__(self):
        return 'Profil użytkownika {}.'.format(self.user.username)
```

> Aby zapewnić możliwość ogólnego działania kodu, do pobrania modelu użytkownika używamy funkcji `get_user_model()`. Następnie opcja `AUTH_USER_MODEL` pozwala na odwołanie do tego modelu podczas definiowania związku z modelem `User` zamiast konieczności użycia bezpośredniego odwołania do modelu `User`.

Kolumna `user` definiuje związek typu „jeden do jednego" i pozwala na powiązanie profilu z użytkownikiem. Zdjęcie użytkownika jest przechowywane w kolumnie `ImageField`. W celu zarządzania obrazami konieczne będzie zainstalowanie jednego z podanych pakietów Pythona — `PIL` (ang. *python imaging library*) lub `Pillow`, który powstał na bazie `PIL`. Instalacja pakietu `Pillow` zostanie przeprowadzona po wydaniu w powłoce poniższego polecenia.

```
$ pip install Pillow==2.9.0
```

Aby w serwerze programistycznym Django umożliwić obsługę plików multimedialnych przekazywanych przez użytkowników, w pliku *settings.py* projektu należy dodać poniższe wiersze kodu.

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

Opcja `MEDIA_URL` wskazuje bazowy adres URL określający lokalizację przeznaczoną do przechowywania plików multimedialnych przekazywanych przez użytkowników. Natomiast opcja `MEDIA_ROOT` określa lokalną ścieżkę dostępu dla tych plików. Ścieżka dostępu jest budowana dynamicznie względem projektu, co zapewnia możliwość ogólnego działania kodu.

Teraz przeprowadź edycję pliku głównego *urls.py* projektu `bookmarks` i w następujący sposób zmodyfikuj znajdujący się w nim kod.

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

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^account/', include('account.urls')),
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
                          document_root=settings.MEDIA_ROOT)
```

W ten sposób serwer programistyczny Django stał się odpowiedzialny za obsługę plików multimedialnych podczas prac nad aplikacją.

Funkcja pomocnicza `static()` jest odpowiednia do stosowania w środowisku programistycznym, ale na pewno nie w produkcyjnym. Pamiętaj, aby w środowisku produkcyjnym nigdy nie udostępniać plików statycznych za pomocą Django.

Przejdź do powłoki i wydaj następujące polecenie, które spowoduje utworzenie migracji bazy danych dla nowego modelu.

```
$ python manage.py makemigrations
```

Powinieneś otrzymać następujące dane wyjściowe.

```
Migrations for 'account':
  0001_initial.py:
    - Create model Profile
```

Kolejnym krokiem jest zsynchronizowanie bazy danych za pomocą poniższego polecenia.

```
$ python manage.py migrate
```

Wygenerowane dane wyjściowe będą zawierały między innymi następujący wiersz.

```
Applying account.0001_initial... OK
```

Przeprowadź edycję pliku *admin.py* aplikacji `account` i zarejestruj model `Profile` w witrynie administracyjnej, co pokazałem poniżej.

```
from django.contrib import admin
from .models import Profile

class ProfileAdmin(admin.ModelAdmin):
    list_display = ['user', 'date_of_birth', 'photo']

admin.site.register(Profile, ProfileAdmin)
```

Ponownie uruchom serwer programistyczny za pomocą polecenia `python manage.py runserver`. Teraz będziesz mógł zobaczyć model `Profile` w witrynie administracyjnej projektu, co pokazałem na rysunku 4.15.

Rysunek 4.15. Model Profile wyświetlony w witrynie administracyjnej Django

Teraz zajmiemy się umożliwieniem użytkownikowi przeprowadzenia edycji profilu w witrynie internetowej. Dodaj poniższe formularze modelu do pliku *forms.py* aplikacji `account`.

```
from .models import Profile

class UserEditForm(forms.ModelForm):
    class Meta:
        model = User
        fields = ('first_name', 'last_name', 'email')

class ProfileEditForm(forms.ModelForm):
    class Meta:
        model = Profile
        fields = ('date_of_birth', 'photo')
```

Oto krótkie omówienie dodanych formularzy.

- `UserEditForm`. Formularz pozwala użytkownikowi na edycję imienia, nazwiska i adresu e-mail. Wymienione informacje są przechowywane we wbudowanym w Django modelu `User`.
- `ProfileEditForm`. Formularz pozwala użytkownikowi na edycję danych dodatkowych, które zostaną zapisane w modelu `Profile`. Użytkownik będzie mógł podać datę urodzenia i wczytać obraz (tak zwany awatar) dla swojego profilu.

Przeprowadź edycję pliku *views.py* aplikacji `account` i zaimportuj model `Profile` w następujący sposób.

```
from .models import Profile
```

Następnie dodaj poniższe wiersze kodu do widoku `register`; umieść je pod funkcją `new_user.save()`.

```
# Utworzenie profilu użytkownika.
profile = Profile.objects.create(user=new_user)
```

Kiedy użytkownik rejestruje się w witrynie, tworzymy powiązany z nim pusty model. Dla istniejących użytkowników obiekty `Profile` musimy utworzyć ręcznie za pomocą witryny administracyjnej Django.

Teraz umożliwimy użytkownikowi edycję profilu. Dodaj poniższy fragment kodu do tego samego pliku (*views.py*).

```
from .forms import LoginForm, UserRegistrationForm, UserEditForm, ProfileEditForm

@login_required
def edit(request):
    if request.method == 'POST':
        user_form = UserEditForm(instance=request.user,
                                 data=request.POST)
        profile_form = ProfileEditForm(
                                    instance=request.user.profile,
                                    data=request.POST,
                                    files=request.FILES)
        if user_form.is_valid() and profile_form.is_valid():
            user_form.save()
            profile_form.save()
    else:
        user_form = UserEditForm(instance=request.user)
        profile_form = ProfileEditForm(instance=request.user.profile)
    return render(request,
                  'account/edit.html',
                  {'user_form': user_form,
                   'profile_form': profile_form})
```

Używamy dekoratora `login_required`, ponieważ w celu przeprowadzenia edycji profilu użytkownik musi być uwierzytelniony. W takim przypadku korzystamy z dwóch modeli formularzy, czyli `UserEditForm` przeznaczonego do przechowywania danych wbudowanego modelu `User` i `ProfileEditForm` przeznaczonego do przechowywania dodatkowych danych profilu. Aby zweryfikować dane wysłane w formularzu, sprawdzamy, czy wartością zwrotną metody `is_valid()` w obu wymienionych formularzach jest `True`. Jeżeli tak, zawartość obu formularzy zapisujemy w celu uaktualnienia odpowiedniego obiektu w bazie danych.

Dodaj poniższy wzorzec adresu URL do pliku *urls.py* aplikacji `account`.

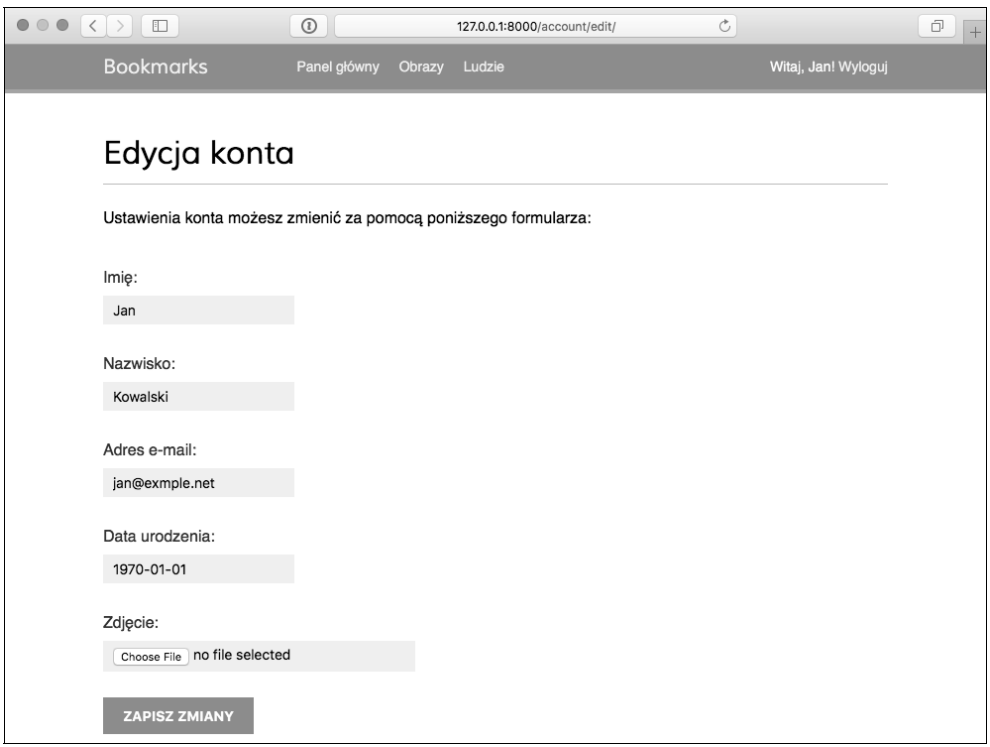
```
url(r'^edit/$', views.edit, name='edit'),
```

Na koniec w katalogu *templates/account/* utwórz nowy szablon dla widoku i nadaj mu nazwę *edit.html*. Następnie w tym pliku umieść poniższy fragment kodu.

```
{% extends "base.html" %}

{% block title %}Edycja konta{% endblock %}

{% block content %}
  <h1>Edycja konta</h1>
  <p>Ustawienia konta możesz zmienić za pomocą poniższego formularza:</p>
  <form action="." method="post" enctype="multipart/form-data">
    {{ user_form.as_p }}
    {{ profile_form.as_p }}
    {% csrf_token %}
    <p><input type="submit" value="Zapisz zmiany"></p>
  </form>
{% endblock %}
```

Aby umożliwić przekazywanie plików, w formularzu musi znaleźć się opcja `enctype="multipart/form-data"`. Wykorzystujemy tylko jeden formularz HTML do wysłania obu formularzy Django, czyli `user_form` i `profile_form`.

Zarejestruj nowego użytkownika i przejdź pod adres *http://127.0.0.1:8000/account/edit/* w przeglądarce internetowej. Powinieneś zobaczyć stronę pokazaną na rysunku 4.16.



Rysunek 4.16. Strona pozwalająca na edycję profilu użytkownika

Teraz możemy zmodyfikować stronę panelu głównego i umieścić na niej łącza prowadzące do stron pozwalających na edycję profilu i zmianę hasła. Otwórz szablon *account/dashboard.html* i poniższy wiersz kodu

```
<p>Witaj w panelu głównym.</p>
```

zastąp następującym.

```
<p>Witaj w panelu głównym. Możesz <a href="{% url "edit" %}">edytować
profil</a> lub <a href="{% url "password_change" %}">zmienić hasło</a>.</p>
```

Po wprowadzonych zmianach użytkownik będzie miał z poziomu panelu głównego dostęp do formularza umożliwiającego edycję profilu.

Użycie własnego modelu User

Django oferuje możliwość całkowitego zastąpienia modelu `User` własnym. Klasa modelu powinna dziedziczyć po klasie `AbstractUser` zapewniającej pełną implementację użytkownika domyślnego jako modelu abstrakcyjnego. Więcej informacji na temat tego rodzaju podejścia znajdziesz na stronie *https://docs.djangoproject.com/en/1.8/topics/auth/customizing/#substituting-a-custom-user-model*.

Zastosowanie własnego modelu użytkownika daje znacznie większą elastyczność, choć może oznaczać również nieco większą trudność podczas integracji z innymi aplikacjami, które współdziałają ze standardowym modelem `User`.

Użycie frameworka komunikatów

Podczas obsługi akcji podejmowanych przez użytkowników może wystąpić konieczność informowania ich o skutkach podjętych przez nich działań. Django oferuje wbudowany framework pozwalający na wyświetlanie użytkownikom jednorazowych powiadomień. Framework jest dostarczany przez aplikację `django.contrib.messages`, która została domyślnie umieszczona na liście `INSTALLED_APPS` w pliku *settings.py* podczas tworzenia nowego projektu za pomocą polecenia `python manage.py startproject`. Zwróć uwagę na fakt, że plik ustawień zawiera oprogramowanie pośredniczące o nazwie `django.contrib.messages.middleware.MessageMiddleware` umieszczone na liście `MIDDLEWARE_CLASSES` ustawień projektu. Framework komunikatów zapewnia prosty sposób dodawania komunikatów przeznaczonych do wyświetlenia użytkownikom. Wspomniane komunikaty są przechowywane w bazie danych i wyświetlane podczas następnego żądania wykonywanego przez danego użytkownika. Framework komunikatów można wykorzystać w widokach — w tym celu należy zaimportować odpowiedni moduł — a następnie można już dodawać nowe komunikaty za pomocą prostych skrótów, co pokazałem poniżej.

```
from django.contrib import messages
messages.error(request, 'Wystąpił pewien problem!')
```

Nowe wiadomości możesz tworzyć z wykorzystaniem metody `add_message()` lub dowolnej z wymienionych poniżej metod skrótów.

- `success()`. Komunikat sukcesu wyświetlany po zakończeniu operacji powodzeniem.
- `info()`. Ogólny komunikat informacyjny.
- `warning()`. Wystąpił pewien problem, ale jeszcze nie mamy do czynienia z niepowodzeniem.
- `error()`. Akcja nie zakończyła się powodzeniem lub doszło do niepowodzenia.
- `debug()`. Komunikaty debugowania, które będą usunięte lub zignorowane w środowisku produkcyjnym.

Przechodzimy teraz do wyświetlania komunikatów użytkownikom. Ponieważ framework komunikatów jest stosowany globalnie w projekcie, komunikaty możemy wyświetlać użytkownikowi za pomocą szablonu bazowego. Otwórz więc plik *base.html* i poniższy fragment kodu umieść pomiędzy elementem `<div>` o identyfikatorze `header` a elementem `<div>` o identyfikatorze `content`.

```
{% if messages %}
  <ul class="messages">
    {% for message in messages %}
      <li class="{{ message.tags }}">
        {{ message|safe }}
          <a href="#" class="close">✖</a>
      </li>
    {% endfor %}
  </ul>
{% endif %}
```

Framework komunikatów zawiera procesor kontekstu dodający zmienną `messages` do kontekstu żądania. Dlatego też wymienionej zmiennej można używać w szablonie do wyświetlania użytkownikowi aktualnych komunikatów.

Teraz zmodyfikujmy widok edycji profilu, aby wykorzystać w nim framework komunikatów. Przeprowadź edycję pliku *views.py* naszej aplikacji i zmień kod widoku `edit` na następujący.

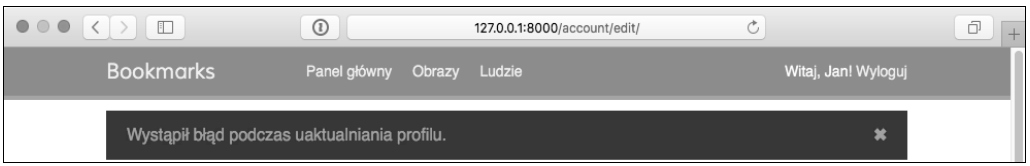
```
from django.contrib import messages

@login_required
def edit(request):
    if request.method == 'POST':
    # ...
        if user_form.is_valid() and profile_form.is_valid():
            user_form.save()
            profile_form.save()
            messages.success(request, 'Uaktualnienie profilu '\
                                      'zakończyło się sukcesem.')
        else:
            messages.error(request, 'Wystąpił błąd podczas uaktualniania profilu.')
    else:
        user_form = UserEditForm(instance=request.user)
        # ...
```

Dodaliśmy komunikat informujący o sukcesie wyświetlany, gdy operacja zmiany profilu zakończy się powodzeniem. Jeżeli którykolwiek formularz będzie wypełniony nieprawidłowo, nastąpi wyświetlenie komunikatu błędu.

W przeglądarce internetowej przejdź pod adres *http://127.0.0.1:8000/account/edit/*, a następnie przeprowadź edycję profilu. Jeśli profil zostanie pomyślnie zaktualizowany, powinieneś zobaczyć komunikat informujący o sukcesie, co pokazałem na rysunku 4.17.

Rysunek 4.17. Komunikat informujący o zakończonej powodzeniem edycji profilu

Gdy formularz jest nieprawidłowy, Django wyświetli komunikat błędu, taki jaki pokazałem na rysunku 4.18.



Rysunek 4.18. Komunikat informujący o błędzie, który wystąpił podczas uaktualniania profilu

Implementacja własnego mechanizmu uwierzytelniania

Django pozwala na uwierzytelnianie względem wielu różnych źródeł. Opcja `AUTHENTICATION_BACKENDS` to lista mechanizmów uwierzytelniania, które mogą być stosowane w projekcie. Domyślnie opcja ta ma następującą wartość.

```
('django.contrib.auth.backends.ModelBackend',)
```

Domyślny model o nazwie `ModelBackend` uwierzytelnia użytkowników na podstawie bazy danych za pomocą modelu `User` dostarczanego przez `django.contrib.auth`. Takie rozwiązanie okazuje się wystarczające w większości projektów. Istnieje jednak możliwość przygotowania własnego mechanizmu uwierzytelniania i wykorzystania innych źródeł, na przykład usług katalogowych (LDAP) bądź też innego systemu.

Więcej informacji dotyczących dostosowania mechanizmu uwierzytelniania do własnych potrzeb znajdziesz na stronie *https://docs.djangoproject.com/en/1.8/topics/auth/customizing/#other-authentication-sources*.

Jeśli tylko użyjesz funkcji `authenticate()` zdefiniowanej w `django.contrib.auth`, Django spróbuje po kolei uwierzytelnić użytkownika względem każdego mechanizmu zdefiniowanego na liście `AUTHENTICATION_BACKENDS` aż do chwili, gdy którakolwiek próba zakończy się powodzeniem. Jeżeli nie uda się uwierzytelnić użytkownika za pomocą żadnego mechanizmu, wtedy pozostanie on niezalogowany w witrynie.

Django pozwala na bardzo łatwe zdefiniowanie własnego mechanizmu uwierzytelniania. Sama procedura opiera się na klasie dostarczającej dwie wymienione poniżej metody.

- `authenticate()`. Metoda pobiera dane uwierzytelniające jako parametry. Wartością zwrotną jest `True`, jeżeli udało się uwierzytelnić użytkownika, i `False` w przypadku niepowodzenia.
- `get_user()`. Metoda pobiera parametr w postaci identyfikatora użytkownika i zwraca odpowiadający mu obiekt `User`.

Przygotowanie własnego mechanizmu uwierzytelnienia jest naprawdę łatwe i sprowadza się do utworzenia klasy Pythona implementującej obie wymienione metody. Zdefiniujemy teraz własny mechanizm uwierzytelniania, aby pozwolić użytkownikom na zalogowanie się do witryny za pomocą adresu e-mail zamiast nazwy użytkownika.

Utwórz nowy plik w katalogu aplikacji `account` i nadaj mu nazwę *authentication.py*. Następnie umieść w nim poniższy fragment kodu.

```
from django.contrib.auth.models import User

class EmailAuthBackend(object):
    """
    Uwierzytelnia użytkownika na podstawie adresu e-mail.
    """
    def authenticate(self, username=None, password=None):
        try:
            user = User.objects.get(email=username)
            if user.check_password(password):
                return user
            return None
        except User.DoesNotExist:
            return None

    def get_user(self, user_id):
        try:
            return User.objects.get(pk=user_id)
        except User.DoesNotExist:
            return None
```

To jest prosty mechanizm uwierzytelniania. Metoda `authenticate()` pobiera parametry opcjonalne `username` i `password`. Wprawdzie można by użyć innych parametrów, ale zdecydowałem się na wymienione, aby ułatwić współpracę tego mechanizmu z widokami frameworka uwierzytelniania. Oto dokładne omówienie sposobu działania powyższego kodu.

- `authenticate()`. Próbujemy pobrać użytkownika na podstawie podanego adresu e-mail oraz sprawdzamy hasło za pomocą wbudowanej metody `check_password()` modelu `User`. Wymieniona metoda dla podanego przez użytkownika hasła generuje wartość hash, którą następnie porównuje z wartością przechowywaną w bazie danych.
- `get_user()`. Metoda pobiera użytkownika na podstawie identyfikatora podanego w parametrze `user_id`. Django wykorzystuje mechanizm, który uwierzytelnił użytkownika, do pobrania odpowiadającego mu obiektu `User` na czas trwania sesji użytkownika.

Przeprowadź edycję pliku *settings.py* projektu i dodaj poniższe ustawienia.

```
AUTHENTICATION_BACKENDS = (
    'django.contrib.auth.backends.ModelBackend',
    'account.authentication.EmailAuthBackend',
)
```

Pozostawiamy domyślny mechanizm `ModelBackend` używany do uwierzytelnienia użytkownika na podstawie podanych przez niego nazwy użytkownika i hasła. Opracowany mechanizm uwierzytelnienia na podstawie adresu e-mail dodajemy jako drugi. Teraz w przeglądarce internetowej przejdź pod adres *http://127.0.0.1:8000/account/login/*. Ponieważ Django będzie próbował uwierzytelnić użytkownika za pomocą każdego mechanizmu, więc do witryny możesz się zalogować, podając nazwę użytkownika lub adres e-mail.

> Kolejność umieszczenia mechanizmów uwierzytelniania na liście `AUTHENTICATION_BACKENDS` ma znaczenie. Jeżeli te same dane uwierzytelniające są prawidłowe dla wielu mechanizmów, Django zakończy próby na pierwszym mechanizmie, za pomocą którego uda się uwierzytelnić użytkownika.

Dodanie do witryny uwierzytelnienia za pomocą innej witryny społecznościowej

Być może do budowanej witryny internetowej zechcesz dodać obsługę uwierzytelnienia za pomocą innego serwisu społecznościowego, takiego jak Facebook, Twitter lub Google. Dostępny jest moduł Pythona o nazwie `Python-social-auth`, który ułatwia proces implementacji tego rodzaju uwierzytelnienia. Dzięki wykorzystaniu wymienionego modułu możesz zezwolić użytkownikowi na logowanie się do witryny internetowej za pomocą konta, które założył w innym serwisie. Kod modułu `Python-social-auth` znajdziesz na stronie *https://github.com/omab/python-social-auth*.

Sam moduł jest dostarczany wraz z różnymi mechanizmami uwierzytelniania dla różnych frameworków Pythona, w tym także dla Django.

Aby zainstalować moduł za pomocą menedżera `pip`, przejdź do powłoki i wydaj poniższe polecenie.

```
$ pip install python-social-auth==0.2.12
```

Następnie umieść `social.apps.django_app.default` na liście `INSTALLED_APPS` w pliku *settings.py* projektu, tak jak pokazałem poniżej.

```
INSTALLED_APPS = (
    #...
    'social.apps.django_app.default',
)
```

W ten sposób dodajemy do projektu Django aplikację `default` modułu `python-social-auth`. Teraz wydaj poniższe polecenie, aby przeprowadzić synchronizację modeli `python-social-auth` z bazą danych.

```
$ python manage.py migrate
```

Powinieneś zobaczyć zastosowanie migracji dla aplikacji `default`.

```
Applying default.0001_initial... OK
Applying default.0002_add_related_name... OK
Applying default.0003_alter_email_max_length... OK
```

Moduł `Python-social-auth` oferuje obsługę mechanizmu uwierzytelniania wielu usług. Pełną listę znajdziesz na stronie *https://python-social-auth.readthedocs.org/en/latest/backends/index.*
↪html#supported-backends.

W rozdziale zaimplementujemy uwierzytelnienie w naszej aplikacji za pomocą konta użytkownika w serwisach Facebook, Twitter i Google.

Na początek konieczne jest dodanie do projektu wzorca adresu URL dla logowania za pomocą serwisu społecznościowego. Otwórz główny plik *urls.py* w projekcie `bookmarks` i umieść w nim przedstawiony poniżej wzorzec.

```
url('social-auth/',
    include('social.apps.django_app.urls', namespace='social')),
```

Aby uwierzytelnianie za pomocą serwisu społecznościowego działało, konieczne jest podanie nazwy hosta, ponieważ wiele tego rodzaju usług nie pozwala na przekierowanie na adres `127.0.0.1` lub `localhost`. W systemach Linux lub OS X rozwiązaniem jest edycja pliku */etc/hosts* polegająca na dodaniu wiersza mapującego dowolnie wybraną domenę na adres komputera lokalnego. W omawianym przykładzie decydujemy się na domenę moja-witryna.pl, więc polecenie ma następującą postać.

```
127.0.0.1 moja-witryna.pl
```

Powyższe polecenie oznacza, że po podaniu adresu *moja-witryna.pl* zostanie wskazany komputer lokalny. Jeżeli używasz Windows, plik *hosts* znajdziesz w *C:\Winwows\System32\Drivers\etc\hosts*.

Aby sprawdzić, czy zdefiniowane przekierowanie działa zgodnie z oczekiwaniami, w przeglądarce internetowej przejdź pod adres *http://moja-witryna.pl:8000/account/login/*. Jeżeli zostanie wyświetlona strona logowania do naszej aplikacji, wszystko działa prawidłowo.

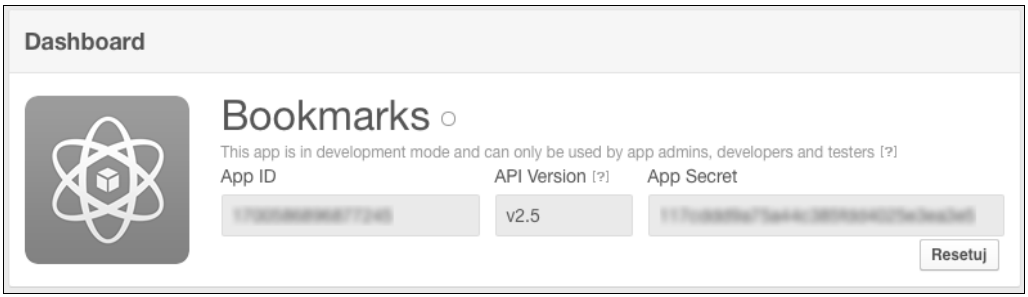
Uwierzytelnienie za pomocą serwisu Facebook

Jeżeli chcesz pozwolić użytkownikowi na zalogowanie się do naszej witryny internetowej za pomocą jego konta w serwisie Facebook, poniższy wiersz kodu umieść na liście `AUTHENTICATION_BACKENDS` w pliku *settings.py* projektu.

```
'social.backends.facebook.Facebook2OAuth2',
```

Dodanie uwierzytelnienia społecznościowego za pomocą serwisu Facebook wymaga konta programistycznego Facebook oraz utworzenia nowej aplikacji Facebook. W przeglądarce internetowej przejdź pod adres *https://developers.facebook.com/apps/?action=create* i kliknij przycisk *Create a New App*. Jako nazwę aplikacji podaj *Bookmarks*, natomiast z rozwijanego menu *Kategoria* wybierz opcję *Aplikacje dla stron* i kliknij przycisk *Create App ID*. Odpowiedz na pytanie mechanizmu zabezpieczającego i kliknij przycisk *Wyślij*. W ustawieniach aplikacji w serwisie Facebook jako platformę wybierz *Strona internetowa*, a gdy padnie pytanie o adres URL witryny, podaj *http://moja-witryna.pl:8000/*.

Teraz przejdź do panelu głównego budowanej witryny internetowej, powinieneś zobaczyć dane pokazane na rysunku 4.19.



Rysunek 4.19. Dane aplikacji Facebook w witrynie administracyjnej Django

Skopiuj wartości *App ID* i *App Secret*, a następnie dodaj je do pliku *settings.py* projektu, tak jak pokazałem poniżej.

```
SOCIAL_AUTH_FACEBOOK_KEY = 'XXX' # Wartość App ID pobrana z serwisu Facebook.
SOCIAL_AUTH_FACEBOOK_SECRET = 'XXX' # Wartość App Secret pobrana z serwisu Facebook.
```

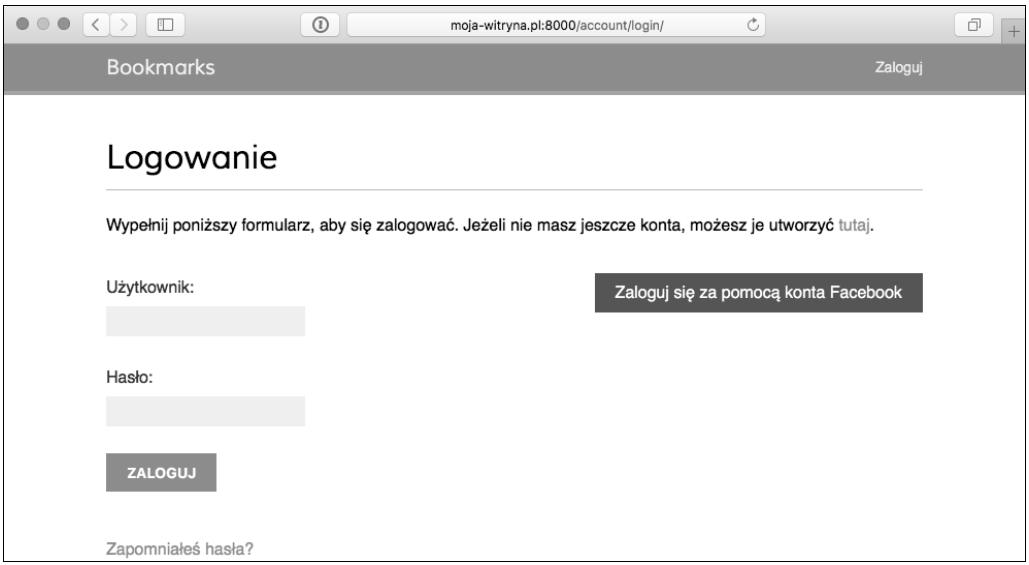
Opcjonalnie możesz dodać opcję `SOCIAL_AUTH_FACEBOOK_SCOPE` i zdefiniować dodatkowe uprawnienia dla użytkowników serwisu Facebook.

```
SOCIAL_AUTH_FACEBOOK_SCOPE = ['email']
```

Na koniec otwórz szablon *registration/login.html* i poniższy fragment kodu umieść w bloku `content`.

```
<div class="social">
  <ul>
    <li class="facebook"><a href="{% url "social:begin" "facebook"
%}">Zaloguj się za pomocą konta Facebook</a></li>
  </ul>
</div>
```

W przeglądarce internetowej przejdź pod adres *http://moja-witryna.pl:8000/account/login/*. Teraz zmodyfikowana strona logowania powinna wyglądać tak, jak pokazałem na rysunku 4.20.



Rysunek 4.20. Zmodyfikowana strona logowania

Kliknij przycisk *Zaloguj się za pomocą konta Facebook*. Zostaniesz przekierowany do serwisu Facebook i zobaczysz okno modalne z pytaniem o udzielenie uprawnień aplikacji `Bookmarks` na uzyskanie dostępu do Twojego publicznego profilu w serwisie Facebook (patrz rysunek 4.21).

Kliknij przycisk *OK*. Moduł `Python-social-auth` zajmie się obsługą uwierzytelnienia. Jeżeli wszystko przebiegnie bez problemów, zostaniesz zalogowany i przeniesiony na stronę panelu głównego w budowanej witrynie internetowej. Pamiętaj, że ten adres URL został podany w opcji `LOGIN_REDIRECT_URL`. Jak możesz zobaczyć, implementacja w aplikacji uwierzytelnienia za pomocą innego serwisu społecznościowego jest naprawdę prosta.

Rysunek 4.21. Pytanie o zgodę na uzyskanie dostępu do profilu w serwisie Facebook

Uwierzytelnienie za pomocą serwisu Twitter

W przypadku uwierzytelnienia z wykorzystaniem serwisu Twitter konieczne jest dodanie poniższego wiersza kodu do listy `AUTHENTICATION_BACKENDS` w pliku *settings.py* aplikacji.

```
'social.backends.twitter.TwitterOAuth',
```

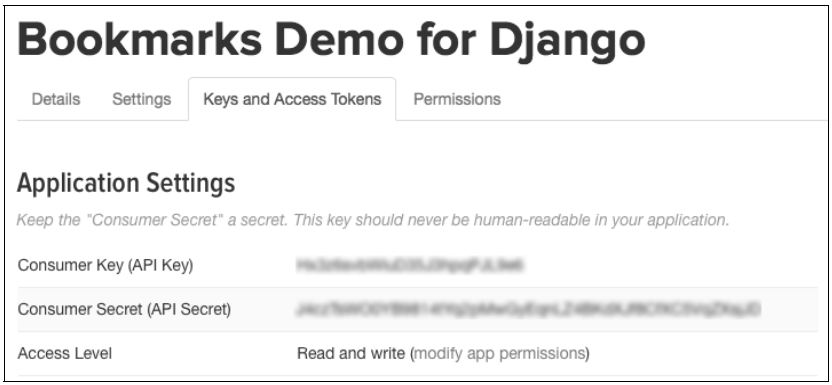
Musisz utworzyć nową aplikację z poziomu konta w serwisie Twitter. W przeglądarce internetowej przejdź pod adres *https://apps.twitter.com/app/new* i podaj informacje szczegółowe dotyczące aplikacji. Oto między innymi one.

- *Website*. `http://moja-witryna.pl:8000/`
- *Callback URL*. `http://moja-witryna.pl:8000/social-auth/complete/twitter/`

Upewnij się, że zaznaczone jest pole wyboru *Allow this application to be used to Sign in with Twitter*. Następnie kliknij kartę *Keys and Access Tokens*. Powinieneś zobaczyć stronę pokazaną na rysunku 4.22.

Dane wymienione w polach *Consumer Key* i *Consumer Secret* skopiuj do przedstawionych poniżej ustawień w pliku *settings.py* projektu.

```
SOCIAL_AUTH_TWITTER_KEY = 'XXX' # Wartość Consumer Key pobrana z serwisu Twitter.
SOCIAL_AUTH_TWITTER_SECRET = 'XXX' # Wartość Consumer Secret pobrana z serwisu Twitter.
```

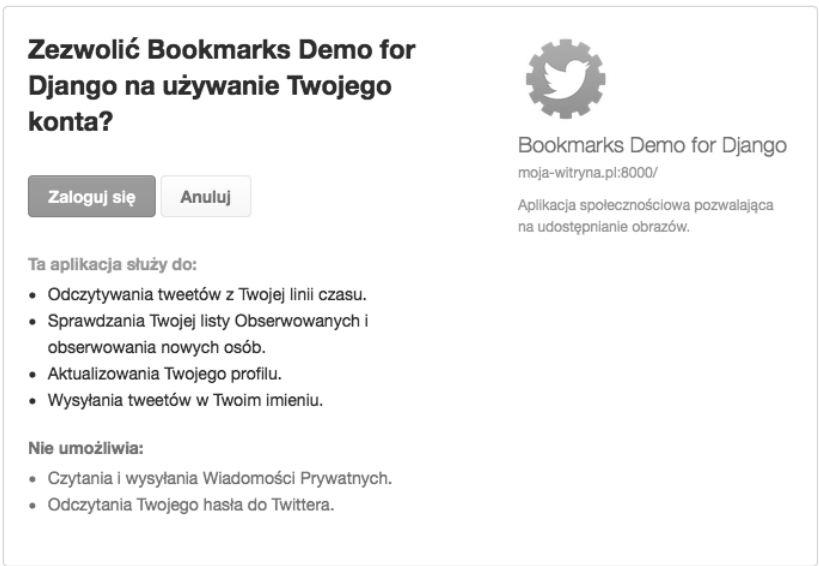


Rysunek 4.22. Szczegóły dotyczące aplikacji w serwisie Twitter

Teraz przeprowadź edycję szablonu *login.html* i poniższy fragment kodu umieść w elemencie `<ul>`.

```
<li class="twitter"><a href="{% url "social:begin" "twitter"
%}">Zaloguj się za pomocą konta Twitter</a></li>
```

W przeglądarce internetowej przejdź pod adres *http://moja-witryna.pl:8000/account/login/* i kliknij łącze *Zaloguj za pomocą konta Twitter*. Zostaniesz przekierowany do serwisu Twitter i poproszony o autoryzowanie aplikacji, co pokazałem na rysunku 4.23.



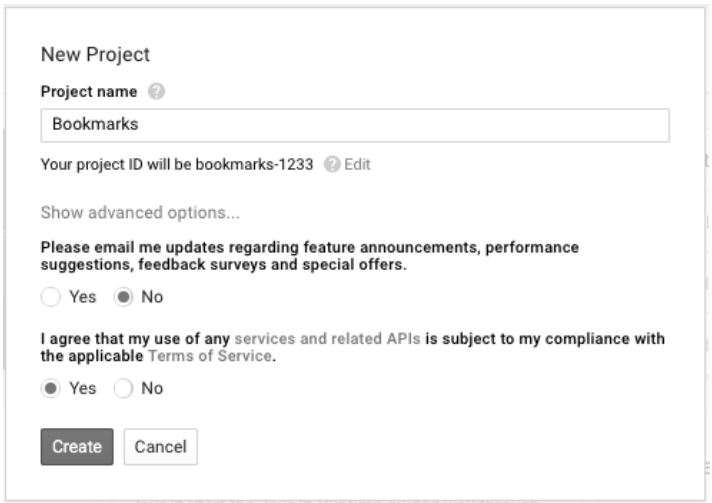
Rysunek 4.23. Autoryzacja aplikacji w serwisie Twitter

Kliknij przycisk *Zaloguj się*. Zostaniesz zalogowany, a następnie przeniesiony na stronę panelu głównego budowanej witryny internetowej.

Uwierzytelnienie za pomocą serwisu Google

Serwis Google oferuje możliwość uwierzytelnienia za pomocą OAuth2. Więcej informacji na temat tej implementacji znajdziesz na stronie *https://developers.google.com/accounts/docs/OAuth2*.

Pracę trzeba zacząć od utworzenia klucza API w Google Developer Console. W przeglądarce internetowej przejdź pod adres *https://console.developers.google.com/project*, a następnie w rozwijanym menu *Select a project* wybierz opcję *Create a project...*. Wpisz nazwę projektu i kliknij przycisk *Create* (patrz rysunek 4.24).



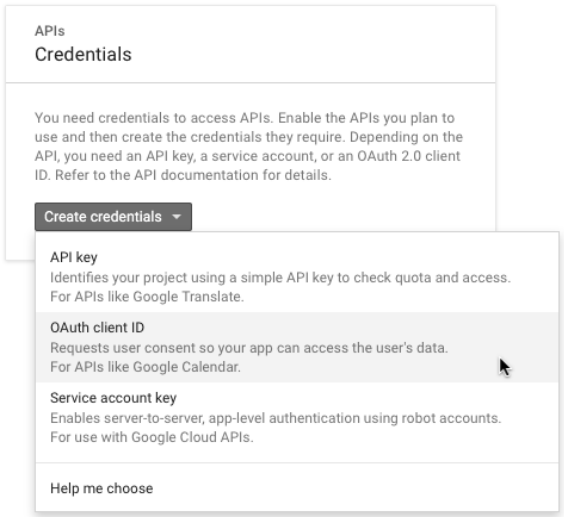
Rysunek 4.24. Utworzenie nowego projektu w Google Developer Console

Po utworzeniu projektu kliknij łącze *Enable and manage APIs* w niebieskim obszarze zatytułowanym Google APIs. W nowym menu po lewej stronie kliknij sekcję *Credentials*. Teraz kliknij przycisk *Create credentials* i wybierz *OAuth client ID*, co pokazałem na rysunku 4.25.

Najpierw trzeba będzie skonfigurować pewne zgody. To jest strona wyświetlana użytkownikowi i zawiera pytanie o zgodę na udzielenie dostępu do Twojej witryny internetowej za pomocą konta Google danego użytkownika. Kliknij przycisk *Configure consent screen*. Wybierz adres e-mail, podaj *Bookmarks* w polu *Product name*, a następnie kliknij przycisk *Save*. W ten sposób dane dla budowanej przez nas aplikacji zostały skonfigurowane i zostaniesz przekierowany na stronę pozwalającą na dokończenie tworzenia identyfikatora klienta.

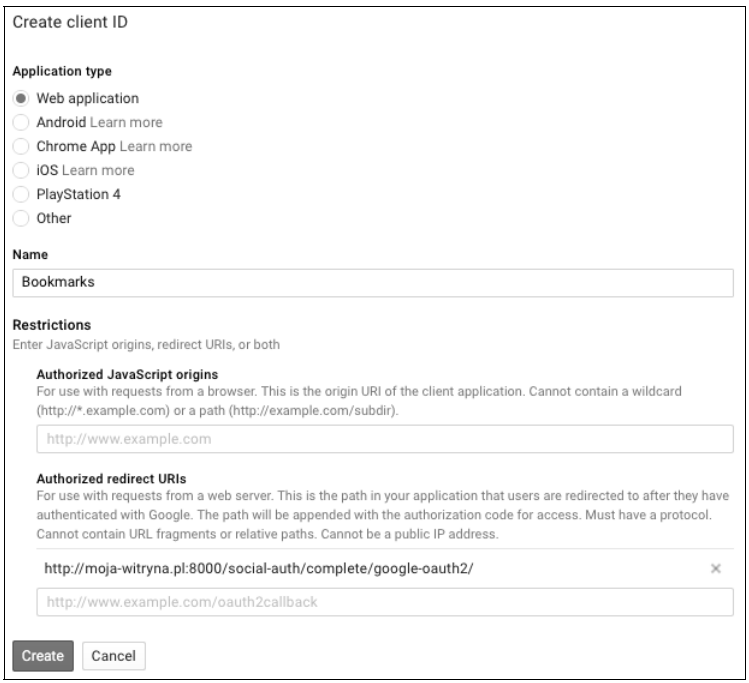
W wyświetlonym formularzu podaj następujące informacje.

- **Application type**. Wybierz opcję *Web application*.
- **Name**. Podaj nazwę *Bookmarks*.
- **Authorized redirect URIs**. Podaj adres *http://moja-witryna.pl:8000/social-auth/complete/google-oauth2/*.



Rysunek 4.25. Przygotowanie uwierzytelnienia za pomocą OAuth2 w Google

Wypełniony formularz powinien wyglądać tak, jak pokazałem na rysunku 4.26.



Rysunek 4.26. Formularz pozwalający na uzyskanie identyfikatora klienta

Kliknij przycisk *Create*. Otrzymasz wartości *Client ID* i *Client Secret*, które musisz umieścić w pliku *settings.py* w następujący sposób.

```
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = ''  # Wartość Consumer Key pobrana z serwisu Google.
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = ''  # Wartość Consumer Secret pobrana z serwisu Google.
```

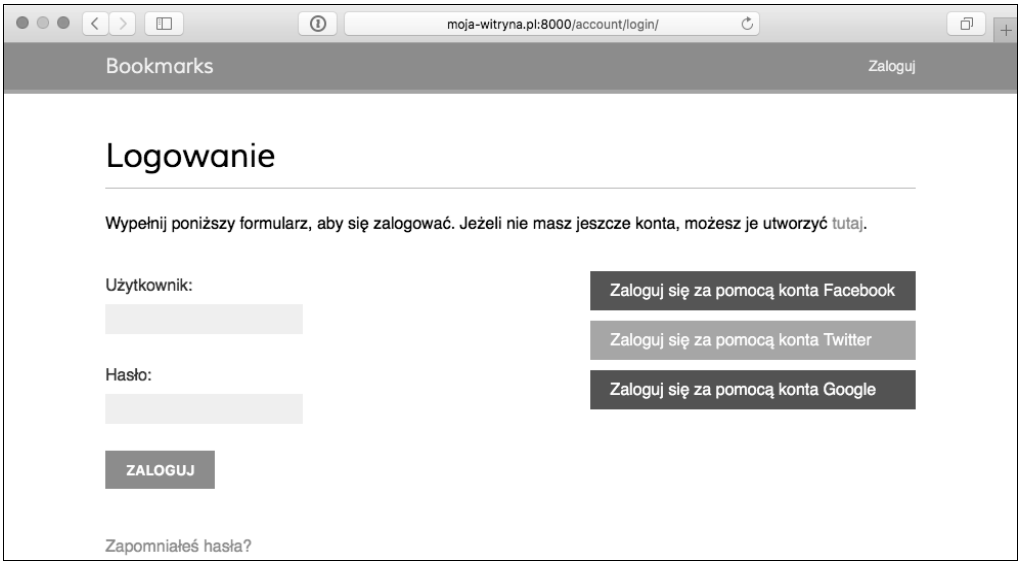
W menu po lewej stronie Google Developers Console przejdź do sekcji *Overview*. Zobaczysz wyświetloną listę wszystkich API udostępnianych przez Google. W sekcji *Social APIs* wybierz *Google+ API*, a następnie kliknij przycisk *Enable* na wyświetlonej stronie, co pokazałem na rysunku 4.27.

Rysunek 4.27. Włączenie API Google+

Przeprowadź edycję szablonu *login.html* i dodaj poniższy fragment kodu do elementu `<ul>`.

```
<li class="google"><a href="{% url "social:begin" "google-oauth2" %}">Zaloguj się
↪za pomocą konta Google</a></li>
```

W przeglądarce internetowej przejdź pod adres *http://moja-witryna.pl:8000/account/login/*. Teraz zmodyfikowana strona logowania powinna wyglądać tak, jak pokazałem na rysunku 4.28.



Rysunek 4.28. Zmodyfikowana strona logowania

Skorowidz

F

G

H

I

J

K

L

M

N

O

P

R

S

Ś

T

U

W

X

Z

Ż

Django
Praktyczne tworzenie aplikacji sieciowych

Django to bardzo przydatne narzędzie ułatwiające pisanie aplikacji sieciowych w języku Python. Jest uważane za framework, który łączy wielkie możliwości z prostotą użytkowania. Pozwala na szybkie tworzenie oprogramowania na podstawie przejrzystych i praktycznych projektów. To atrakcyjne rozwiązanie zarówno dla początkujących, jak i doświadczonych programistów.

Książka, którą trzymasz w rękach, jest znakomitym podręcznikiem pisania aplikacji sieciowych w Django. Krok po kroku pokazano tu pełny proces tworzenia profesjonalnego oprogramowania, a przykładami, na których oparto poszczególne rozdziały, są rzeczywiste projekty aplikacji. Dzięki takiemu podejściu można bardzo szybko zapoznać się z frameworkiem, nauczyć się rozwiązywać najczęstsze problemy i w naturalny sposób wykorzystać najlepsze praktyki programistyczne. Autorzy pokazali również, w jaki sposób w projektach Django stosować kilka popularnych technologii związanych z aplikacjami sieciowymi.

Dowiesz się:

- czym jest Django
- jak przygotować środowisko tego frameworka
- jak stworzyć praktyczny projekt aplikacji umożliwiający dalsze modyfikacje i rozbudowę
- czym charakteryzuje się praca z bazami Redis i innymi technologiami (w tym Celery, Solr i Memcached)
- co to jest API typu RESTful

Antonio Melé — jest informatykiem; projektami Django zajmuje się od 2006 roku. Opiekuje się hiszpańską społecznością użytkowników Django (django.es). Jest założycielem Zenx IT — firmy informatycznej tworzącej aplikacje sieciowe dla klientów z różnych sektorów gospodarki. Brał również udział w tworzeniu wielu startupów informatycznych.

[PACKT] open source PUBLISHING community experience distilled

Helion

45589 numer katalogowy
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-283-2587-6

cena: 77,00 zł