OKIEM EKSPERTA

Django 5

Praktyczne tworzenie aplikacji internetowych w Pythonie

Wydanie V

Antonio Melé

Helion

packt

Tytuł oryginału: Django 5 By Example: Build powerful and reliable Python web applications from scratch, 5th Edition

Tłumaczenie: Radosław Meryk

ISBN: 978-83-289-1882-5

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/dj5pr5
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

Utworzenie witryny społecznościowej

Rozdział 4

W poprzednim rozdziale nauczyłeś się, jak zaimplementować system tagowania i polecać podobne posty. Zaimplementowałeś niestandardowe tagi szablonów i filtry. Nauczyłeś się również, jak tworzyć mapy witryn i kanały wiadomości, a także zbudowałeś — za pomocą PostgreSQL — pełnotekstową wyszukiwarkę.

W tym rozdziale dowiesz się, jak opracować funkcjonalności związane z obsługą kont — w tym rejestrację użytkownika, zarządzanie hasłami, edycję profilu i uwierzytelnianie — w celu stworzenia serwisu społecznościowego. W następnych kilku rozdziałach zaimplementujemy w tej witrynie funkcje społecznościowe, pozwalające użytkownikom współdzielenie zdjęć i interakcje między sobą. Użytkownicy będą mogli oznaczyć w internecie dowolne zdjęcie i udostępnić je innym. Będą również mogli zobaczyć aktywność w internecie użytkowników, których obserwują, oraz polubić udostępniane przez nich zdjęcia (lub wyrazić dezaprobatę na ich temat).

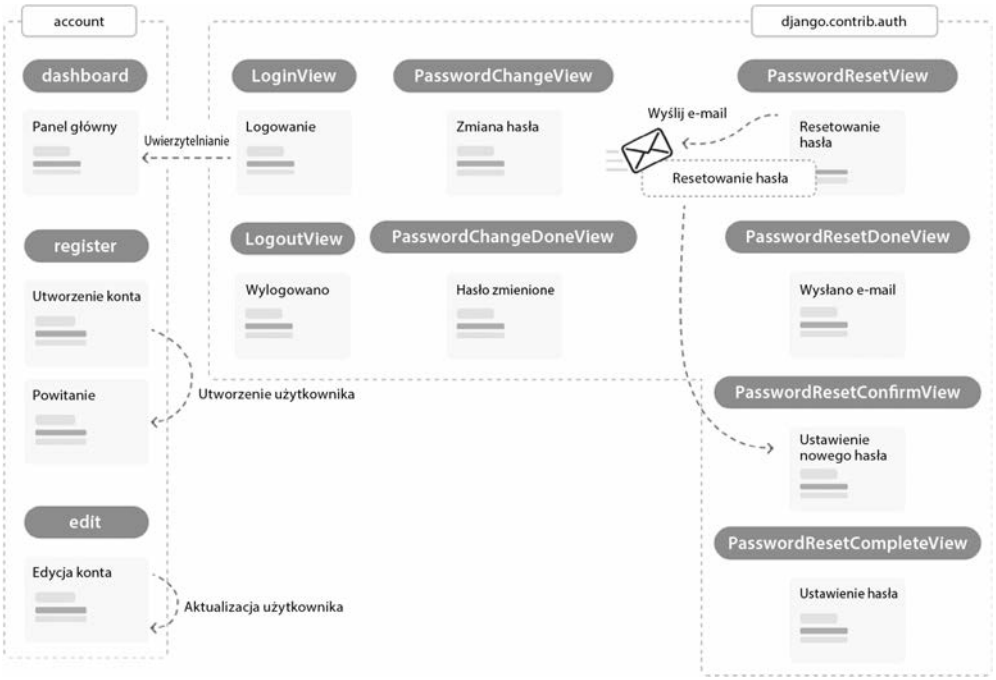
Oto zagadnienia, na których skoncentruję się w tym rozdziale:

- Utworzenie widoku logowania
- Użycie frameworka uwierzytelniania w Django
- Tworzenie w Django szablonów widoków logowania, wylogowania, zmiany hasła i resetowania hasła
- Utworzenie widoków pozwalających na rejestrację użytkowników
- Rozbudowa modelu `User` o obsługę niestandardowego profilu
- Konfigurowanie projektu do przesyłania plików multimedialnych

Przegląd funkcjonalności

Reprezentację widoków, szablonów i funkcjonalności, które zbudujemy w tym rozdziale, przedstawiono na rysunku 4.1.

W tym rozdziale stworzysz nowy projekt i użyjesz widoków logowania, wylogowania, zmiany hasła i odzyskiwania hasła dostarczanych przez Django w pakiecie `django.contrib.auth`. Utworzysz szablony widoków uwierzytelniania, a także widok pulpitu nawigacyjnego, do którego użytkownicy będą mieli dostęp po pomyślnym uwierzytelnieniu. Zaimplementujesz również mechanizm rejestracji użytkowników za pomocą widoku rejestracji (`register`). Na koniec rozszerzysz model użytkownika o niestandardowy model `Profile` i utworzysz widok edycji, by umożliwić użytkownikom edytowanie ich profili.



Rysunek 4.1. Schemat funkcjonalności zbudowanych w rozdziale 4.

Kod źródłowy przykładów z tego rozdziału można znaleźć pod adresem *https://github.com/PacktPublishing/Django-5-by-example/tree/main/Chapter04*.

Wszystkie pakiety Pythona użyte w tym rozdziale są zawarte w pliku *requirements.txt* dołączonym do plików z kodem źródłowym przykładów z tego rozdziału. Aby zainstalować każdy pakiet Pythona, możesz postępować zgodnie z instrukcjami zamieszczonymi w poniższych podrozdziałach. Możesz także zainstalować wszystkie wymagania jednocześnie za pomocą polecenia `pip install -r requirements.txt`.

Utworzenie projektu witryny społecznościowej

Przystępujemy teraz do budowy aplikacji społecznościowej umożliwiającej użytkownikom udostępnianie zdjęć znalezionych w internecie. Ten projekt jest istotny. Wykonanie go pomoże Ci zrozumieć, jak wbudować w witrynę funkcje społecznościowe, a także jak implementować zaawansowane funkcjonalności za pomocą frameworka Django i Java Script.

W ramach witryny do udostępniania zdjęć trzeba będzie zbudować komponenty:

- System uwierzytelniania pozwalający użytkownikowi na rejestrowanie, logowanie, edycję profilu oraz zmianę i zerowanie hasła.
- Uwierzytelnianie społecznościowe do logowania się za pomocą takich usług jak Google.

- Funkcjonalność pozwalającą na wyświetlanie udostępnianych zdjęć oraz udostępnianie obrazów z niemalże każdej witryny internetowej.
- Strumień aktywności dla każdego użytkownika pozwalający użytkownikom śledzić treść dodawaną przez obserwowanych użytkowników.
- System obserwacji pozwalający użytkownikom na śledzenie swoich poczynań.

W tym rozdziale zajmiemy się realizacją pierwszego z wymienionych punktów. Pozostałe punkty omówimy w rozdziałach od 5. do 7.

Rozpoczęcie pracy nad aplikacją społecznościową

Zaczniemy od skonfigurowania środowiska wirtualnego dla projektu i utworzenia początkowej struktury projektu.

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

```
mkdir env
python -m venv env/bookmarks
```

Jeśli używasz systemu Linux lub macOS, aby aktywować środowisko wirtualne, uruchom polecenie:

```
source env/bookmarks/bin/activate
```

Jeśli używasz systemu Windows, skorzystaj z polecenia:

```
.\env\bookmarks\Scripts\activate
```

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

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

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

```
python -m pip install Django~=5.0.4
```

Aby utworzyć nowy projekt, wydaj poniższe polecenie.

```
django-admin startproject bookmarks
```

W ten sposób zbudujemy 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, by dodać aplikację do projektu; zrobisz to, wpisując ją na listę `INSTALLED_APPS` w pliku *settings.py*.

Przeprowadź edycję pliku *settings.py* i dodaj do listy `INSTALLED_APPS` przed jakąkolwiek inną zainstalowaną aplikacją następujący wiersz (wyróżniony na poniższym listingu pogrubioną czcionką).

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

Django szuka szablonów według kolejności występowania aplikacji w ustawieniu `INSTALLED_APPS`. Aplikacja `django.contrib.admin` zawiera standardowe szablony uwierzytelniania, które zastąpimy w aplikacji `account`. Własne aplikacje zazwyczaj umieszczamy na końcu listy. W tym przypadku umieściliśmy aplikację na pierwszym miejscu w ustawieniu `INSTALLED_APPS`, aby zapewnić domyślne wykorzystywanie zdefiniowanych szablonów uwierzytelniania, a nie szablonów uwierzytelniania zawartych w aplikacji `django.contrib.admin`.

Uruchom poniższe polecenie, aby przeprowadzić synchronizację bazy danych z modelami aplikacji domyślnych wskazanymi na liście `INSTALLED_APPS`.

```
python manage.py migrate
```

Zobaczysz, że zostaną zastosowane wszystkie początkowe migracje bazy danych Django. W wyniku migracji zostały utworzone tabele bazy danych odpowiadające modelom Django zainstalowanych aplikacji. Teraz przystępujemy do budowy systemu uwierzytelniania w projekcie z wykorzystaniem frameworka uwierzytelniania Django.

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 aplikacji `django.contrib.auth` i jest używany także przez inne pakiety Django typu `contrib`. Framework uwierzytelniania wykorzystaliśmy już w rozdziale 1., „Utworzenie aplikacji bloga” 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 projektu.

- `AuthenticationMiddleware`: wiąże użytkowników z żądaniami za pomocą mechanizmu sesji.

- `SessionMiddleware`: zapewnia obsługę bieżącej sesji między poszczególnymi żądaniami.

Oprogramowanie middleware to klasy z metodami, które są uruchamiane globalnie podczas faz żądania lub odpowiedzi. W tej książce klasy oprogramowania pośredniczącego będziemy wykorzystywać w wielu sytuacjach. Temat tworzenia oprogramowania pośredniczącego zostanie dokładnie omówiony w rozdziale 17., „Wdrożenie".

Framework uwierzytelniania obejmuje również poniższe modele, które są zdefiniowane w modelach aplikacji `django.contrib.auth.models`.

- `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. Stworzymy widok, który w celu zalogowania użytkownika wykona następujące czynności.

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

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 Password Input do wygenerowania elementu HTML `password`. Zawiera on atrybut `type="password"`, po to aby przeglądarka interpretowała wprowadzane dane jako hasło.

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

```
from django.contrib.auth import authenticate, login
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse
from django.shortcuts import render

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

Oto jakie działania realizuje podstawowy widok logowania:

Po wywołaniu widoku `user_login` przez żądanie `GET` za pomocą wywołania `form = Login ↪Form()` tworzymy nowy egzemplarz formularza logowania. Następnie formularz jest przekazywany do szablonu.

Kiedy użytkownik wyśle formularz przy użyciu żądania `POST`, przeprowadzane są następujące działania.

Utworzenie egzemplarza formularza wraz z wysłanymi danymi. Do tego celu służy polecenie `form = LoginForm(request.POST)`.

Sprawdzenie za pomocą wywołania `form.is_valid()`, czy formularz jest prawidłowy. Jeżeli formularz jest nieprawidłowy, w szablonie wyświetlamy błędy wykryte podczas weryfikacji formularza (np. użytkownik nie wypełnił jednego z pól).

Jeżeli wysłane dane są prawidłowe, uwierzytelniamy użytkownika na podstawie informacji przechowywanych w bazie danych za pomocą metody `authenticate()`. 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 obiekt `HttpResponse` wraz z komunikatem „Nieprawidłowe dane uwierzytelniające”.

W przypadku pomyślnego uwierzytelnienia użytkownika sprawdzamy za pomocą atrybutu `is_active`, czy jego konto jest aktywne. Wymieniony atrybut pochodzi z modelu `User`

dostarczanego przez Django. Gdy konto użytkownika jest nieaktywne, zwracamy obiekt `HttpResponse` wraz z komunikatem „Konto jest zablokowane”.

Gdy konto użytkownika jest aktywne, logujemy go w witrynie internetowej. Rozpoczynamy także sesję dla użytkownika: wywoływana jest metoda `login()` i zwracany komunikat „Uwierzytelnienie zakończyło się sukcesem”.

Uwaga

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.urls import path
from . import views

urlpatterns = [
    path('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. Nowy kod został wyróżniony pogrubioną czcionką.

```
from django.conf import settings
from django.urls import include, path

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

W katalogu aplikacji `account` utwórz wymienioną poniżej strukturę plików i katalogów.

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

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

```
{% load static %}
<!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* aplikacji `account` ze wspomnianych materiałów i umieścić go w tej samej lokalizacji w budowanym projekcie. Zawartość katalogu można znaleźć pod adresem *https://github.com/PacktPublishing/Django-5-by-Example/tree/master/Chapter04/bookmarks/account/static*.

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 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., „Usprawnianie bloga i dodanie funkcjonalności społecznościowych”.

W bazie danych nie ma jeszcze żadnych kont użytkowników. Aby zapewnić sobie dostęp do witryny administracyjnej i zarządzać pozostałymi użytkownikami, trzeba najpierw utworzyć superużytkownika.

W wierszu polecenia powłoki uruchom polecenie:

```
python manage.py createsuperuser
```

Wyświetli się wynik pokazany poniżej. Wprowadź żądaną nazwę użytkownika, adres e-mail i hasło w następujący sposób.

```
Username (leave blank to use 'admin'): admin
Email address: admin@admin.com
Password: ********
Password (again): ********
```

Następnie wyświetli się komunikat o powodzeniu:

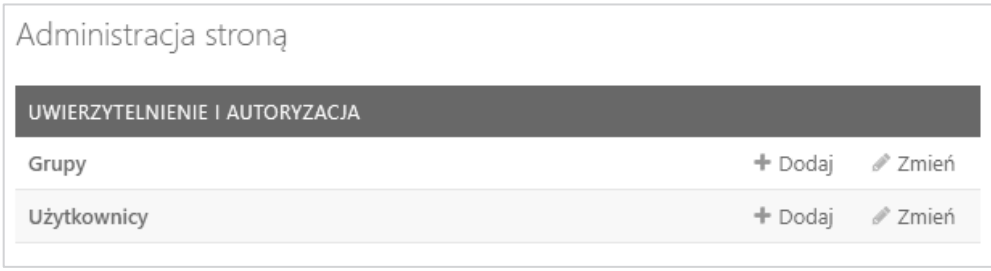
```
Superuser created successfully.
```

Uruchom serwer programistyczny za pomocą polecenia:

```
python manage.py runserver
```

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 `Users` (link *Użytkownicy*) i `Group` (link *Grupy*) dla wbudowanego w Django frameworka uwierzytelniania.

Strona wygląda jak na rysunku 4.2.



Rysunek 4.2. Główna strona witryny administracyjnej Django włącznie z modelami Users i Group

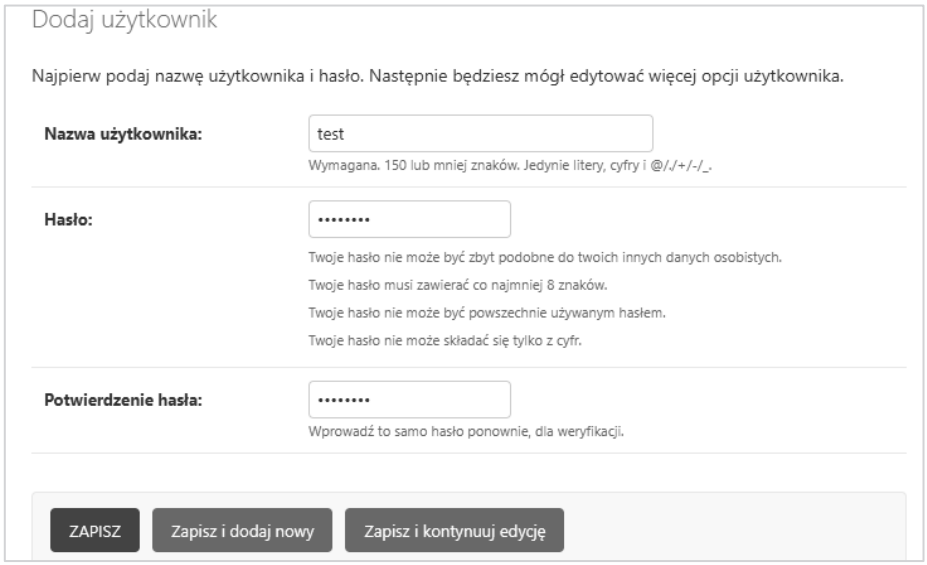
W wierszu *Użytkownicy* kliknij *Dodaj*.

W witrynie administracyjnej utwórz nowego użytkownika (rysunek 4.3).

Wprowadź dane użytkownika i aby zapisać nowego użytkownika w bazie danych, kliknij przycisk *ZAPISZ*.

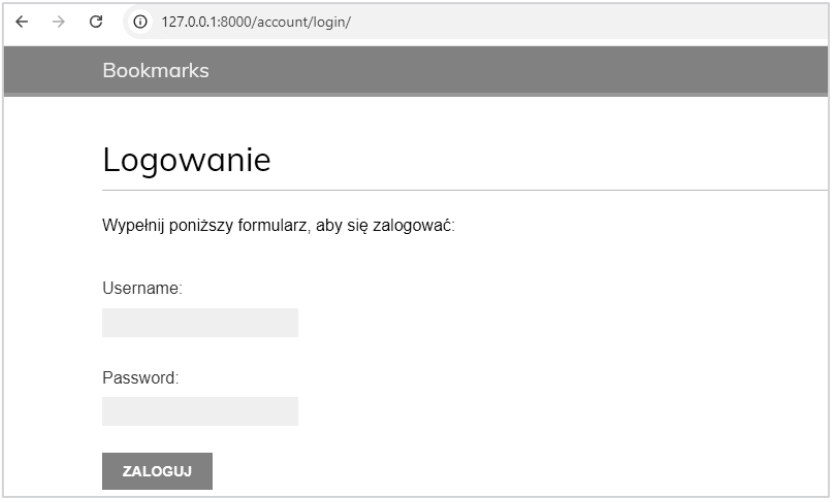
Następnie w obszarze *Informacje osobiste* wypełnij pola *Imię*, *Nazwisko* i *Adres e-mail* w sposób pokazany na rysunku 4.4, po czym kliknij przycisk *Zapisz*, aby zapisać zmiany.

W przeglądarce internetowej przejdź pod adres `http://127.0.0.1:8000/account/login/`. Powinieneś zobaczyć wyrenderowany szablon wraz z formularzem logowania (rysunek 4.5).



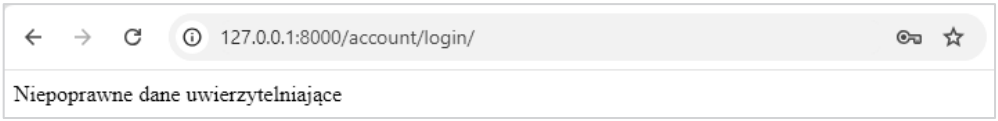
Rysunek 4.3. Strona logowania użytkownika

Rysunek 4.4. Formularz edycji danych użytkownika w witrynie administracyjnej Django



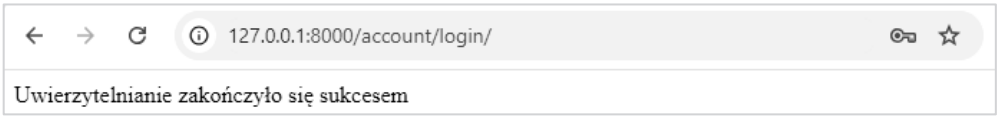
Rysunek 4.5. Strona logowania użytkownika

Wprowadź błędne dane uwierzytelniające i wyślij formularz. Powinieneś otrzymać komunikat *Niepoprawne dane uwierzytelniające* (rysunek 4.6).



Rysunek 4.6. Komunikat tekstowy informujący o niepoprawnym logowaniu

Wprowadź poprawne poświadczenia. Wyświetli się komunikat *Uwierzytelnianie zakończyło się sukcesem* (rysunek 4.7).



Rysunek 4.7. Komunikat tekstowy z informacją o pomyślnym uwierzytelnianiu

Nauczyłeś się uwierzytelniać użytkowników i tworzyć własny widok uwierzytelniania. Co prawda możesz tworzyć własne widoki uwierzytelniania, ale Django zapewnia gotowe do użycia widoki uwierzytelniania, z których możesz skorzystać.

Użycie wbudowanych 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. Wszystkie są dostępne w module `django.contrib.auth.views`.

- `LoginView`: obsługa formularza logowania oraz proces zalogowania użytkownika.
- `LogoutView`: obsługa wylogowania użytkownika.

Do obsługi zmiany hasła Django oferuje wymienione poniżej widoki.

- `PasswordChangeView`: obsługa formularza pozwalającego użytkownikowi na zmianę hasła.
- `PasswordChangeDoneView`: strona informująca o sukcesie operacji; zostanie wyświetlona użytkownikowi, gdy zmiana hasła zakończy się powodzeniem.

Do obsługi operacji resetowania hasła Django oferuje zaś następujące widoki.

- `PasswordResetView`: Umożliwienie użytkownikowi ponownego ustawienia hasła. Django generuje link wraz z tokenem przeznaczonym do jednokrotnego użycia, który jest wysyłany na adres e-mail danego użytkownika.
- `PasswordResetDoneView`: Wyświetlenie użytkownikowi strony z informacją o wysłaniu wiadomości e-mail wraz z linkiem pozwalającym na ponowne ustawienie hasła.
- `PasswordResetConfirmView`: widok umożliwiający użytkownikowi zdefiniowanie nowego hasła.
- `PasswordResetCompleteView`: strona informująca o sukcesie operacji; zostanie wyświetlona użytkownikowi, gdy zmiana 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 można znaleźć pod adresem *https://docs.djangoproject.com/en/5.0/topics/auth/default/#all-authentication-views*.

Widoki logowania i wylogowania

Aby nauczyć się korzystać z widoków uwierzytelniania Django, zastąpimy niestandardowy widok logowania wbudowanym odpowiednikiem Django, a także zintegrujemy z aplikacją widok wylogowania.

Przeprowadź edycję pliku *urls.py* aplikacji `account` i dodaj kod wyróżniony pogrubioną czcionką.

```
from django.contrib.auth import views as auth_views
from django.urls import path
from . import views


urlpatterns = [
    # Poprzedni adres URL strony logowania
    # path('login/', views.user_login, name='login'),
    #  adresy url logowania / wylogowania
    path('login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
]
```

W powyższym kodzie oznaczyliśmy jako komentarz wzorzec adresu URL dla utworzonego wcześniej widoku `user_login`. Teraz wykorzystamy widok `LoginView` oferowany przez wbudowany w Django framework uwierzytelniania. Dodaliśmy także wzorzec adresu URL dla widoku `LogoutView`.

Utwórz nowy podkatalog w katalogu *templates/* aplikacji `account` i nadaj mu nazwę *registration*. Ten podkatalog to domyślna lokalizacja, w której widoki uwierzytelniania Django spodziewają się znaleźć szablony.

Moduł `django.contrib.admin` zawiera kilka szablonów uwierzytelniania, które są używane w witrynie administracyjnej. Podczas konfigurowania projektu aplikację `account` umieściliśmy na początku ustawienia `INSTALLED_APPS`, aby Django domyślnie korzystał z niestandardowych szablonów zamiast szablonów uwierzytelniania zdefiniowanych w innych aplikacjach.

W katalogu *templates/registration* 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 niepoprawne.
      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 modułu `django.contrib.auth.forms`. Formularz próbuje uwierzytelnić użytkownika i zgłasza błąd walidacji, gdy logowanie zakończy się niepowodzeniem. W takim przypadku, aby sprawdzić, czy podane zostały niepoprawne dane uwierzytelniające, można przeanalizować te błędy za pomocą znacznika szablonu `{% if form.errors %}`.

Zwróć uwagę na dodanie ukrytego elementu HTML `<input>`, przeznaczonego do wysłania wartości zmiennej o nazwie `next`. Ta zmienna będzie dostarczana do widoku logowania, jeśli przekażesz do żądania parametr o nazwie `next`, na przykład przez otwarcie w przeglądarce strony `http://127.0.0.1:8000/account/login/?next=/account/`.

Parametr `next` musi reprezentować adres URL. Jeżeli podamy ten parametr, to po pomyślnym zalogowaniu w widoku logowania Django przekieruje użytkownika pod wskazany adres URL.

Teraz utwórz w katalogu *registration* szablon *logged_out.html* 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 %}
```

To jest szablon, który Django wyświetli po wylogowaniu się użytkownika.

Dodaliśmy wzorce i szablony adresów URL dla widoków logowania i wylogowania. Użytkownicy mogą teraz logować się i wylogowywać za pomocą oferowanych przez Django widoków uwierzytelniania.

Przystępujemy teraz do utworzenia nowego widoku. Będzie on przeznaczony do wyświetlenia użytkownikowi, po pomyślnym zalogowaniu się w aplikacji, panelu głównego (ang. *dashboard*).

Przeprowadź edycję pliku *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'}
    )
```

Utworzyliśmy widok `dashboard` i zastosowaliśmy do niego dekorator `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 HTML `<input>`.

Zdefiniowaliśmy również zmienną section. Użyjemy tej zmiennej do zaznaczenia bieżącej sekcji w menu głównym witryny.

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 %}
```

Przeprowadź edycję pliku *urls.py* aplikacji `account` i dodaj w nim poniższy wzorzec adresu URL oznaczony pogrubieniem.

```
urlpatterns = [
    # Poprzedni adres URL strony logowania
    #  path('login/', views.user_login, name='login'),

    # Adresy url logowania / wylogowania
    path('login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),

    path('', views.dashboard, name='dashboard'),
]
```

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

```
LOGIN_REDIRECT_URL = 'dashboard'
LOGIN_URL = 'login'
LOGOUT_URL = 'logout'
```

Zdefiniowaliśmy następujące ustawienia.

- `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 (np. za pomocą dekoratora `login_required`).
- `LOGOUT_URL`: Adres URL, do którego ma nastąpić przekierowanie po wylogowaniu użytkownika.

Użyliśmy nazw adresów URL, które wcześniej zdefiniowaliśmy we wzorcach adresów URL za pomocą atrybutu `name` funkcji `path()`. Do tych ustawień zamiast nazw adresów URL można również użyć zakodowanych „na sztywno” adresów URL.

Oto krótkie podsumowanie przeprowadzonych dotąd działań:

- Do projektu dodaliśmy wbudowane we frameworku uwierzytelniania Django widoki logowania (`login`) i wylogowania (`logout`).
- 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 linki logowania i wylogowania. Aby to zrobić, trzeba ustalić, czy bieżący użytkownik jest zalogowany. Na tej podstawie zostanie wyświetlony właściwy link (logowania lub wylogowania). Warstwa middleware odpowiedzialna za uwierzytelnianie ustawia bieżącego użytkownika w obiekcie `HttpRequest`. Dostęp do niego

można uzyskać za pomocą `request.user`. Obiekt `request` zawiera obiekt `User` nawet wtedy, kiedy użytkownik nie został uwierzytelniony. W takim przypadku użytkownik będzie zdefiniowany w postaci egzemplarza obiektu `AnonymousUser`. Najlepszym sposobem zweryfikowania, czy użytkownik został uwierzytelniony, jest sprawdzenie wartości jego atrybutu „tylko do odczytu" `is_authenticated`.

Przeprowadź edycję pliku *templates/base.html* i dodaj poniższe wiersze wyróżnione pogrubioną czcionką.

```
{% load static %}
<!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>
    {% 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|default:request.user.username }},
        <form action="{% url "logout" %}" method="post">
          <button type="submit">Wyloguj</button>
          {% csrf_token %}
        </form>
      {% else %}
        <a href="{% url "login" %}">Zaloguj </a>
      {% endif %}
    </span>
  </div>
  <div id="content">
    {% block content %}
    {% endblock %}
  </div>
</body>
</html>
```

Menu witryny internetowej będzie wyświetlane jedynie uwierzytelnionym użytkownikom. Sprawdzana jest także zmienna `section`, reprezentująca sekcję witryny, w celu dodania klasy atrybutu `selected` do odpowiedniego elementu `<li>` i tym samym zaznaczenia za

pomocą CSS nazwy bieżącej sekcji. Jeśli użytkownik jest zalogowany, wyświetlane jest imię uwierzytelnionego użytkownika i link pozwalający mu na wylogowanie. Jeżeli użytkownik nie jest uwierzytelniony, wyświetlony będzie link pozwalający mu na zalogowanie. Jeśli nazwa użytkownika jest pusta, wyświetlana jest wartość pola `username` za pomocą wywołania `request.user.first_name|default:request.user.username`. Zauważ, że dla akcji wylogowania używamy formularza z metodą `POST` i przycisku do przesłania formularza. To dlatego, że widok `LogoutView` wymaga żądań `POST`.

W przeglądarce internetowej przejdź pod adres `http://127.0.0.1:8000/account/login/`. Powinieneś zobaczyć stronę logowania. Podaj poprawne dane uwierzytelniające i kliknij przycisk *Zaloguj*. Wyświetli się ekran pokazany na rysunku 4.8.

Rysunek 4.8. Strona panelu głównego

Dzięki zastosowaniu stylów CSS nazwa sekcji *Panel główny* została wyświetlona innym kolorem czcionki, ponieważ odpowiadającemu jej elementowi `<li>` przypisaliśmy klasę `selected`. Skoro użytkownik jest uwierzytelniony, jego imię wyświetlamy po prawej stronie nagłówka. Kliknij *Wyloguj*. Powinieneś zobaczyć stronę podobną do pokazanej na rysunku 4.9.

Rysunek 4.9. Strona wyświetlana po wylogowaniu użytkownika

Na tej stronie widać, że użytkownik jest wylogowany, w związku z czym menu witryny nie jest wyświetlane. Link znajdujący się po prawej stronie nagłówka zmienia się na *Zaloguj*.

Uwaga

Jeżeli zamiast przygotowanej wcześniej strony wylogowania zostanie wyświetlona strona wylogowania witryny administracyjnej Django, sprawdź listę `INSTALLED_APPS` projektu i zadbaj o to, aby wpis dotyczący aplikacji `django.contrib.admin` został wymieniony po `account`. Obie aplikacje zawierają szablony wylogowania znajdujące się w tej samej ścieżce względnej. Z tego powodu mechanizm ładujący szablony Django przejrzy różne aplikacje na liście `INSTALLED_APPS` i użyje pierwszego znalezionego szablonu.

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.

```
urlpatterns = [
    # poprzedni widok logowania
    # path('login/', views.user_login, name='login'),

    # adresy url logowania / wylogowania
    path('login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),

    # Adresy URL przeznaczone do obsługi zmiany hasła
    path(
        'password-change/',
        auth_views.PasswordChangeView.as_view(),
        name='password_change'
    ),
    path(
        'password-change/done/',
        auth_views.PasswordChangeDoneView.as_view(),
        name='password_change_done'
    ),

    path('', views.dashboard, name='dashboard'),
]
```

Widok `PasswordChangeView` zapewnia obsługę formularza pozwalającego na zmianę hasła, natomiast `PasswordChangeDoneView` 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 method="post">
    {{ form.as_p }}
    <p><input type="submit" value="Zmień"></p>
    {% csrf_token %}
  </form>
{% endblock %}
```

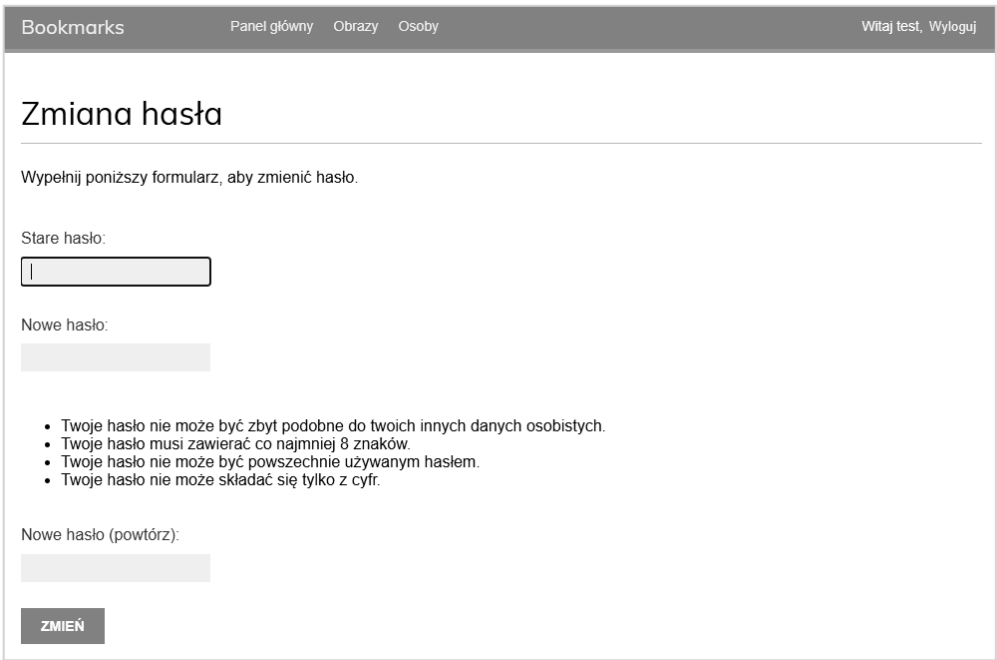
Przedstawiony szablon zawiera formularz przeznaczony do obsługi procedury zmiany hasła.

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 o sukcesie operacji 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 formularz pozwalający na zmianę hasła, pokazany na rysunku 4.10.



Rysunek 4.10. Formularz zmiany hasła

W wyświetlonym formularzu należy podać dotychczasowe hasło oraz nowe, a następnie kliknąć przycisk *ZMIEŃ*. Jeżeli operacja przebiegnie bez problemów, zostanie wyświetlona pokazana poniżej strona wraz z komunikatem informującym o sukcesie (rysunek 4.11).

Rysunek 4.11. Strona z komunikatem o pomyślnej zmianie hasła

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

Widoki odzyskiwania hasła

Jeśli użytkownik zapomni hasła, powinien mieć możliwość odzyskania swojego konta. W tym punkcie zaimplementujemy funkcjonalność resetowania hasła. Umożliwi to użytkownikom odzyskanie dostępu do konta poprzez otrzymanie wiadomości e-mail zawierającej bezpieczny link, wygenerowany za pomocą unikatowego tokena, umożliwiający utworzenie nowego hasła.

Otwórz plik *urls*.py aplikacji `account` i umieść w nim poniższe wzorce adresów URL wyróżnione pogrubioną czcionką.

```
urlpatterns = [
    # Poprzedni adres URL strony logowania
    # path('login/', views.user_login, name='login'),

    # adresy url logowania / wylogowania
    path('login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),

    # Adresy URL przeznaczone do obsługi zmiany hasła
    path(
        'password-change/',
        auth_views.PasswordChangeView.as_view(),
        name='password_change'
    ),
    path(
        'password-change/done/',
        auth_views.PasswordChangeDoneView.as_view(),
        name='password_change_done'
    ),

    # Adresy URL przeznaczone do obsługi procedury odzyskiwania hasła
    path(
        'password-reset/',
        auth_views.PasswordResetView.as_view(),
        name='password_reset'
    ),
    path(
        'password-reset/done/',
```

```
            auth_views.PasswordResetDoneView.as_view(),
            name='password_reset_done'
        ),
        path(
            'password-reset/<uidb64>/<token>/',
            auth_views.PasswordResetConfirmView.as_view(),
            name='password_reset_confirm'
        ),
        path(
            'password-reset/complete/',
            auth_views.PasswordResetCompleteView.as_view(),
            name='password_reset_complete'
        ),
        path('', views.dashboard, name='dashboard'),
    ]
```

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

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

{% block title %}Odzyskiwanie hasła{% endblock %}

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

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

```
Otrzymaliśmy żądanie odzyskania hasła dla użytkownika używającego adresu e-mail
➥{{email }}. Kliknij poniższy link:
{{ protocol }}://{{ domain }}{% url "password_reset_confirm" uidb64=uid token=token %}
Twoja nazwa użytkownika, na wypadek gdybyś zapomniał: {{ user.get_username }}
```

Szablon *password_reset_email.html* zostanie użyty do wygenerowania wiadomości e-mail wysyłanej użytkownikowi, który chce przeprowadzić operację odzyskania hasła. Wiadomość ta zawiera wygenerowany przez widok token, który jest potrzebny do ustawienia nowego hasła.

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

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

{% block title %}Odzyskiwanie hasła{% endblock %}

{% block content %}
  <h1>Odzyskiwania 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 wpisałeś
  ↪adres e-mail podany podczas zakładania konta użytkownika.</p>
{% endblock %}
```

Utwórz kolejny plik szablonu w tym samym katalogu i nadaj mu nazwę *password_reset_confirm.html*. Następnie w nowym pliku umieść poniższy fragment kodu.

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

{% block title %}Odzyskiwanie hasła{% endblock %}

{% block content %}
  <h1>Odzyskiwanie hasła</h1>
  {% if validlink %}
    <p>Dwukrotnie podaj nowe hasło:</p>
    <form method="post">
      {{ form.as_p }}
      {% csrf_token %}
      <p><input type="submit" value="Zmień hasło" /></p>
    </form>
  {% else %}
    <p>Link pozwalający na odzyskanie hasła jest niepoprawny, ponieważ
    ↪prawdopodobnie został już wcześniej użyty. Musisz ponownie rozpocząć procedurę
    ↪odzyskiwania hasła.</p>
   {% endif %}
{% endblock %}
```

W tym szablonie sprawdzamy, czy podany link jest poprawny. W tym celu weryfikowana jest zmienna `validlink`. Oferowany przez Django widok `PasswordResertConfirmView` ustawia zmienną i umieszcza ją w kontekście szablonu. Jeżeli link jest poprawny, wtedy wyświetlamy użytkownikowi formularz odzyskiwania hasła. Użytkownicy mogą ustawić nowe hasło tylko wtedy, gdy dysponują poprawnym linkiem odzyskiwania hasła.

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

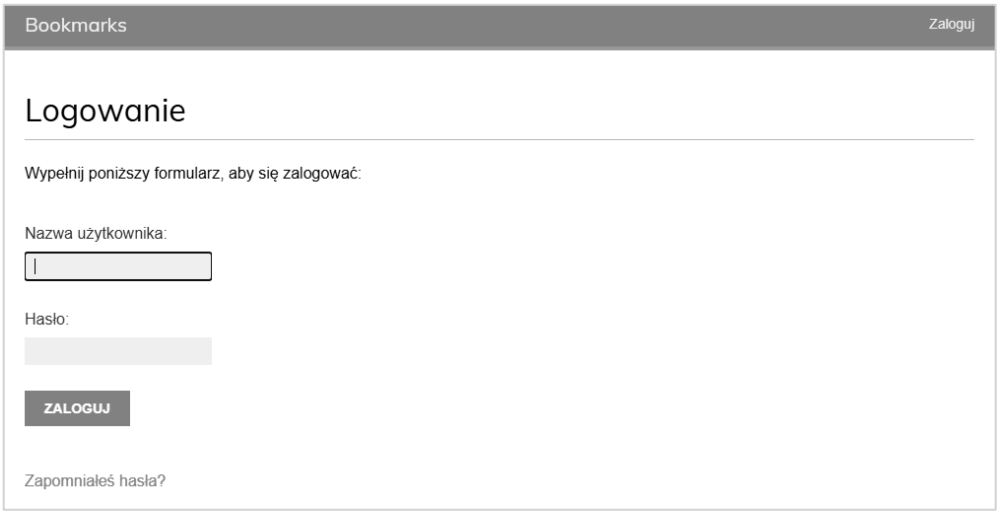
```
{% extends "base.html" %}

{% block title %}Odzyskiwanie hasła{% endblock %}

{% block content %}
  <h1>Ustawianie 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 poniższe wiersze wyróżnione pogrubioną czcionką.

```
{% extends "base.html" %}
{% block title %}Logowanie{% endblock %}
{% block content %}
  <h1>Logowanie</h1>
  {% if form.errors %}
```

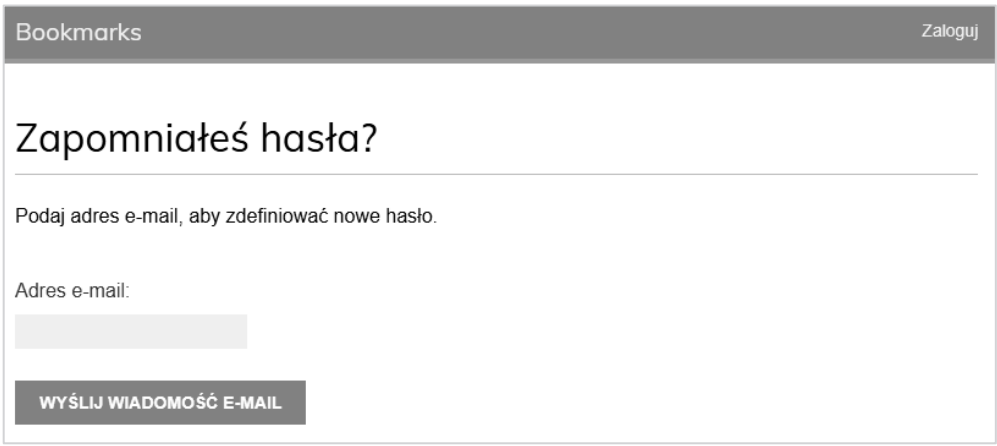
```
    <p>
      Nazwa użytkownika lub hasło są niepoprawne.
      Spróbuj ponownie.
    </p>
  {% else %}
    <p>Wypełnij poniższy formularz, aby się zalogować:</p>
  {% endif %}
  <div class="login-form">
    <form action="{% url 'login' %}" method="post">
      {{ form.as_p }}
      {% csrf_token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

Teraz w przeglądarce internetowej przejdź pod adres `http://127.0.0.1:8000/account/login/`. Strona logowania powinna teraz zawierać link do strony odzyskiwania hasła (rysunek 4.12).



Rysunek 4.12. Strona logowania zawierająca link do strony odzyskiwania hasła

Kliknij link *Zapomniałeś hasła?*. Powinieneś zobaczyć stronę podobną do pokazanej na rysunku 4.13.



Rysunek 4.13. Formularz przywracania hasła

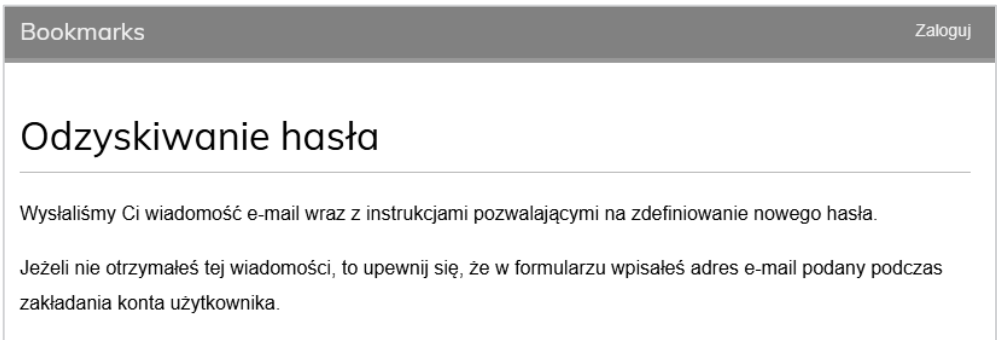
Na tym etapie, aby umożliwić Django wysyłanie wiadomości e-mail, w pliku *settings.py* projektu trzeba umieścić konfigurację serwera **SMTP**. Procedurę dodawania tego rodzaju konfiguracji do projektu omówiłem w rozdziale 2., „Usprawnianie bloga i dodanie funkcjonalności społecznościowych". Jednak podczas 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 backend do wyświetlania wiadomości e-mail w konsoli.

W pliku *settings.py* swojego projektu dodaj poniższy fragment kodu.

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

Ustawienie `EMAIL_BACKEND` wskazuje klasę, która będzie używana do wysyłania wiadomości e-mail.

Wróć do przeglądarki internetowej, podaj adres e-mail istniejącego użytkownika i kliknij przycisk *WYŚLIJ WIADOMOŚĆ E-MAIL*. Powinieneś zobaczyć stronę podobną do pokazanej na rysunku 4.14.



Rysunek 4.14. Strona z informacją o wysłaniu wiadomości e-mail dotyczącej odzyskiwania hasła

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

```
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 8bit
Subject:
 =?utf-8?q?Reset_has=C5=82a_dla_konta_na_stronie_127=2E0=2E0=2E1=3A8000?=
From: webmaster@localhost
To: test@test.pl
Date: Sun, 22 Sep 2024 04:44:14 -0000
Message-ID: <172698025475.19188.14083776009677761263@DESKTOP-3ADGNTT>
Otrzymaliśmy żądanie odzyskania hasła dla użytkownika używającego adresu e-mail
↪test@test.pl. Kliknij poniższy link:
http://127.0.0.1:8000/account/password-reset/Mg/cdqtdq-76b2060878e0441fc8d4f60133257372/
Twoja nazwa użytkownika, na wypadek gdybyś zapomniał: test
```

Ta wiadomość e-mail została wygenerowana za pomocą utworzonego wcześniej szablonu *password_reset_email.html*. Adres URL pozwalający na przejście do strony odzyskiwania hasła zawiera token dynamicznie wygenerowany przez Django.

Skopiuj z wiadomości e-mail adres URL, który powinien wyglądać podobnie do następującego: `http://127.0.0.1:8000/account/password-reset/MQ/ardx0u-b4973cfa2c70d652a1↪90e79054bc479a/`, i otwórz go w przeglądarce. Powinieneś zobaczyć stronę podobną do pokazanej na rysunku 4.15.

Rysunek 4.15. Formularz odzyskiwania 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 nowy skrót hasła i zapisze go w bazie danych. Jeżeli operacja przebiegnie bez problemów, zostanie wyświetlona strona pokazana na rysunku 4.16 wraz z komunikatem informującym o sukcesie.

Rysunek 4.16. Strona z komunikatem o pomyślnym odzyskaniu hasła

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 otrzymany link, 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.

Wzorce adresów URL uwierzytelniania, które właśnie utworzyliśmy, udostępnia również framework Django. Teraz zastąpimy wzorce adresów URL uwierzytelniania wzorcami dostarczonymi przez Django.

Ujmij w komentarz wzorce adresów URL uwierzytelniania, które dodaliśmy do pliku *urls.py* aplikacji `account`, i zamiast tego dodaj aplikację `django.contrib.auth.urls` tak, jak pokazałem poniżej. Nowy kod został wyróżniony pogrubioną czcionką.

```
from django.urls import include, path
from django.contrib.auth import views as auth_views
from . import views

urlpatterns = [
    # poprzedni widok logowania
    # path('login/', views.user_login, name='login'),

    # path('login/', auth_views.LoginView.as_view(), name='login'),
    # path('logout/', auth_views.LogoutView.as_view(), name='logout'),

    # Adresy URL przeznaczone do obsługi zmiany hasła
    # path(
    #   'password-change/',
    #   auth_views.PasswordChangeView.as_view(),
```

```
        #    name='password_change'
        # ),
        # path(
        #    'password-change/done/',
        #    auth_views.PasswordChangeDoneView.as_view(),
        #    name='password_change_done'
        # ),
        # Adresy URL przeznaczone do obsługi procedury odzyskiwania hasła
        # path(
        #    'password-reset/',
        #    auth_views.PasswordResetView.as_view(),
        #    name='password_reset'
        # ),
        # path(
        #    'password-reset/done/',
        #    auth_views.PasswordResetDoneView.as_view(),
        #    name='password_reset_done'
        # ),
        # path(
        #    'password-reset/<uidb64>/<token>/',
        #    auth_views.PasswordResetConfirmView.as_view(),
        #     name='password_reset_confirm'
        # ),
        # path(
        #    'password-reset/complete/',
        #    auth_views.PasswordResetCompleteView.as_view(),
        #    name='password_reset_complete'
        # ),

        path('', include('django.contrib.auth.urls')),
        path('', views.dashboard, name='dashboard'),
    ]
```

Wzorce URL uwierzytelniania można znaleźć pod adresem *https://github.com/django/django/blob/stable/5.0.x/django/contrib/auth/urls.py*.

Dodaliśmy teraz do naszego projektu wszystkie niezbędne widoki uwierzytelniania. Następnie zaimplementujemy mechanizmy rejestracji użytkowników.

Rejestracja użytkownika i profile użytkownika

Istniejący użytkownicy mogą się zalogować, wylogować, zmienić hasło lub je odzyskać, jeśli go zapomną. Musimy teraz przygotować widok pozwalający nowym odwiedzającym witrynę na założenie w niej konta użytkownika. Powinni oni być w stanie zarejestrować się na stronie i utworzyć profil. Po zarejestrowaniu użytkownicy będą mogli zalogować się w aplikacji przy użyciu swoich poświadczeń.

Rejestracja użytkownika

Przystępujemy do utworzenia prostego widoku pozwalającego odwiedzającemu na zarejestrowanie się w 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 import forms
from django.contrib.auth import get_user_model


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


class UserRegistrationForm(forms.ModelForm):
    password = forms.CharField(
        label='Password',
        widget=forms.PasswordInput
    )
    password2 = forms.CharField(
        label='Repeat password',
        widget=forms.PasswordInput
    )

    class Meta:
        model = get_user_model()
        fields = ['username', 'first_name', 'email']
```

Utworzyliśmy formularz modelu (klasa `ModelForm`) dla modelu `User`. W przygotowanym formularzu będą uwzględnione jedynie pola `username`, `first_name` i `email`. Model użytkownika uzyskujemy dynamicznie, za pomocą funkcji `get_user_model()` dostarczanej przez aplikację `auth`. Framework Django pozwala na definiowanie niestandardowych modeli użytkowników, dlatego uruchomienie tej funkcji powoduje pobranie modelu użytkownika, który może być niestandardowym modelem, zamiast domyślnego modelu `User` aplikacji `auth`. Wartości wymienionych pól będą weryfikowane na podstawie odpowiadających im pól modelu. Jeśli na przykład użytkownik wybierze już istniejącą nazwę użytkownika, otrzyma komunikat o błędzie w trakcie walidacji formularza, ponieważ pole `username` jest zdefiniowane z ustawieniem `unique=True`.

Uwaga

Aby kod był generyczny, do pobrania modelu użytkownika używamy funkcji `get_user_model()`. Następnie, podczas definiowania relacji z tym modelem zamiast bezpośredniego odwoływania się do modelu `User` można korzystać z opcji `AUTH_USER_MODEL`. Więcej informacji na ten temat można przeczytać na stronie *https://docs.djangoproject.com/en/5.0/topics/auth/customizing/#django.contrib.auth.get_user_model*.

Dodaliśmy dwa nowe pola `password` i `password2` przeznaczone do zdefiniowania hasła i jego potwierdzenia. Dodajmy kod walidacji pola, aby sprawdzić, czy oba hasła są takie same.

Zmodyfikuj plik *forms.py* w aplikacji `account` i dodaj do klasy `UserRegistrationForm` zamieszczoną poniżej metodę `clean_password2()`. Nowy kod został wyróżniony pogrubioną czcionką.

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

    class Meta:
        model = get_user_model()
        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']
```

Zdefiniowaliśmy metodę `clean_password2()` w celu porównywania drugiego hasła z pierwszym i zgłoszenia błędu sprawdzania poprawności, jeśli hasła nie pasują. Metoda ta jest wywoływana podczas walidacji 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 walidacji formularza dla określonego pola. Formularze zawierają także ogólną metodę `clean()` przeznaczoną do sprawdzenia całego formularza, co okazuje się użyteczne podczas walidacji pól zależnych wzajemnie od siebie. W tym przypadku zamiast zastępowania metody `clean()` formularza użyliśmy walidacji specyficznej dla pola `clean_password2()`. Pozwala to uniknąć zastępowania innych, specyficznych dla pól mechanizmów walidacji, które klasa `ModelForm` pobiera na podstawie mechanizmów kontroli (ang. *constraints*) ustawionych w modelu (na przykład sprawdzanie, czy nazwa użytkownika jest unikatowa).

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

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

```
from django.contrib.auth import authenticate, login
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse
from django.shortcuts import render
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; jednak 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. Ze względów bezpieczeństwa zamiast zapisywać wprowadzone przez użytkownika hasło w postaci zwykłego tekstu, wykorzystujemy metodę `set_password()` modelu `User`. Metoda ta obsługuje haszowanie hasła przed zapisaniem go w bazie danych.

Django nie przechowuje haseł w postaci zwykłego tekstu; zamiast tego przechowuje skróty (ang. *hashes*) haseł. Haszowanie to proces przekształcania podanego klucza w inną wartość (tzw. skrót). Funkcja skrótu służy do generowania wartości o stałej długości zgodnie z algorytmem matematycznym. Dzięki haszowaniu haseł za pomocą bezpiecznych algorytmów Django znacząco utrudnia złamanie haseł przechowywanych w bazie danych.

Domyślnie do przechowywania wszystkich haseł użytkowników Django używa algorytmu haszowania `PBKDF2` ze skrótem SHA256. Framework obsługuje nie tylko sprawdzanie istniejących skrótów haseł obliczonych za pomocą algorytmu `PBKDF2`, ale także sprawdzanie przechowywanych skrótów haseł obliczonych za pomocą innych algorytmów, takich jak `PBKDF2SHA1`, `argon2`, `bcrypt` i `scrypt`.

Skróty haseł obsługiwane przez projekt Django są zdefiniowane za pomocą ustawienia `PASSWORD_HASHERS`. Oto domyślna lista `PASSWORD_HASHERS`:

```
PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
    'django.contrib.auth.hashers.Argon2PasswordHasher',
```

```
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
    'django.contrib.auth.hashers.ScryptPasswordHasher',
]
```

Do haszowania wszystkich haseł Django używa pierwszego wpisu na liście, w tym przypadku `PBKDF2PasswordHasher`. Reszty hasherów z listy Django może używać do sprawdzania istniejących haseł.

Uwaga

Hasher `scrypt` wprowadzono w Django 4.0. Jest bezpieczniejszy i bardziej zalecany niż `PBKDF2`. `PBKDF2` nadal jest jednak domyślnym hasherem, ponieważ `scrypt` wymaga zainstalowania OpenSSL 1.1+ i więcej pamięci.

Więcej informacji o sposobach przechowywania haseł w Django i o hasherach możesz znaleźć na stronie *https://docs.djangoproject.com/en/5.0/topics/auth/passwords/*.

Otwórz plik *urls.py* aplikacji `account` i umieść w nim poniższe wzorce adresów URL oznaczone pogrubioną czcionką.

```
urlpatterns = [

    # ...

    path('', include('django.contrib.auth.urls')),
    path('', views.dashboard, name='dashboard'),
    path('register/', views.register, name='register'),
]
```

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

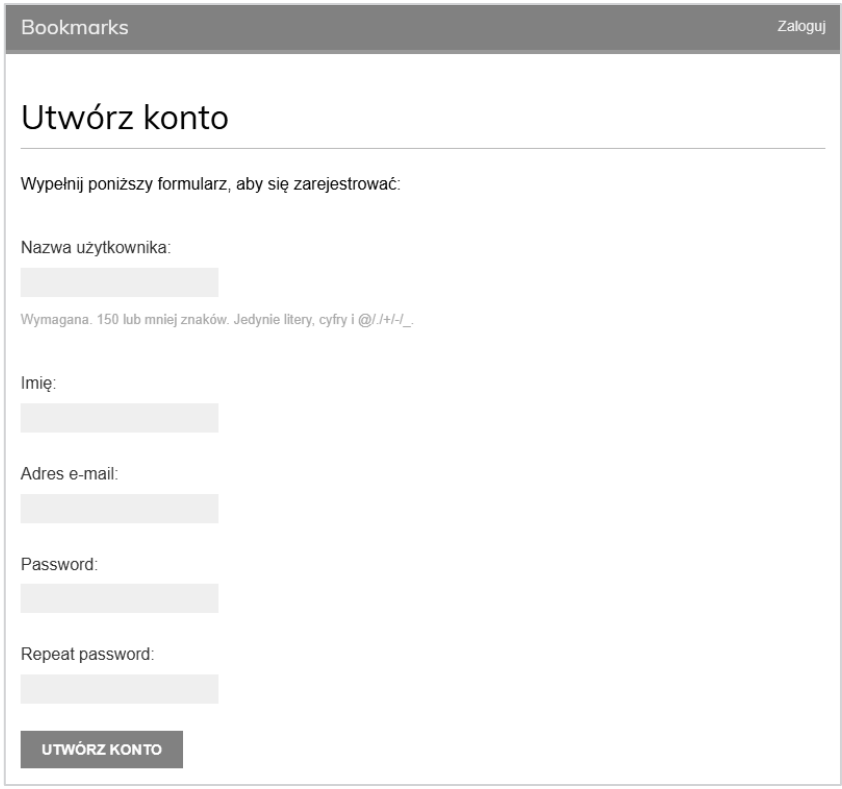
```
{% extends "base.html" %}

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

{% block content %}
  <h1>Utwórz konto</h1>
  <p> Wypełnij poniższy formularz, aby się zarejestrować:</p>
  <form 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 w nowym pliku umieść 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 %}
```

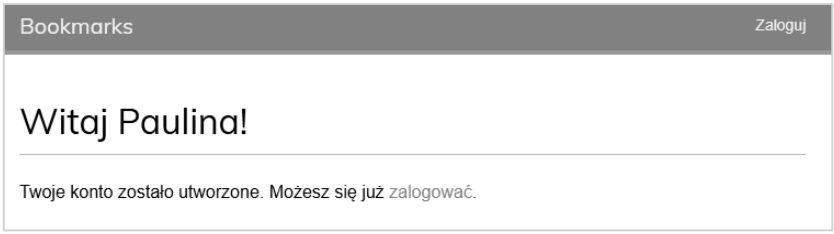
W przeglądarce internetowej przejdź pod adres `http://127.0.0.1:8000/account/register/`. Powinieneś zobaczyć wyświetloną stronę rejestracji nowego użytkownika (rysunek 4.17).



Rysunek 4.17. Formularz tworzenia konta

Podaj informacje potrzebne do utworzenia nowego konta użytkownika i kliknij przycisk *UTWÓRZ KONTO*.

Jeżeli wszystkie pola wypełnisz poprawnie, zostanie wyświetlona strona wraz z komunikatem informującym o pomyślnym utworzeniu nowego konta użytkownika (rysunek 4.18).



Rysunek 4.18. Strona z komunikatem o pomyślnym utworzeniu konta

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

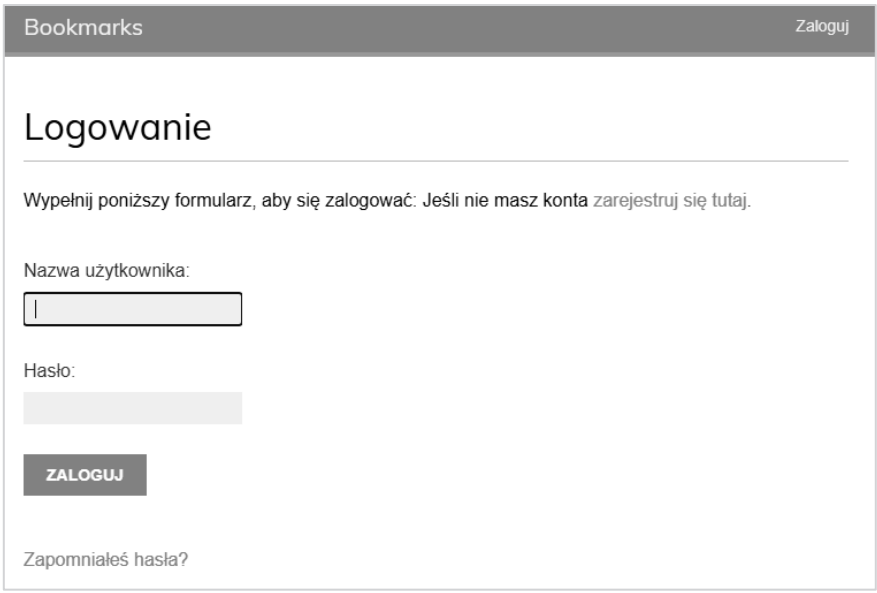
Teraz do szablonu logowania trzeba dodać link pozwalający na rejestrację użytkownika. Przeprowadź edycję szablonu *registration/login.html* i poniższy wiersz kodu:

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

zastąp wierszami:

```
<p>
  Wypełnij poniższy formularz, aby się zalogować:
  Jeśli nie masz konta <a href="{% url "register" %}">zarejestruj się tutaj</a>.
</p>
```

W przeglądarce internetowej przejdź pod adres `http://127.0.0.1:8000/account/login/`. Wynik powinien być podobny do pokazanego na rysunku 4.19.



Rysunek 4.19. Strona logowania zawierająca link do rejestracji

Stworzyliśmy stronę rejestracji dostępną ze strony logowania.

Rozbudowa modelu User

O ile model `User` dostarczany przez framework uwierzytelniania Django jest wystarczający do większości typowych scenariuszy, o tyle ma on ograniczony zestaw predefiniowanych pól. Jeśli chcesz uwzględnić dodatkowe szczegóły istotne dla Twojej aplikacji, możesz rozszerzyć domyślny model użytkownika. Na przykład domyślny model użytkownika zawiera pola `first_name` i `last_name`. Taka struktura może nie być zgodna z konwencjami pisowni nazwisk w niektórych krajach. Dodatkowo czasami trzeba przechowywać inne dane użytkownika lub skonstruować bardziej kompleksowy profil użytkownika.

Prostym sposobem na rozszerzenie modelu `User` jest utworzenie modelu profilu, który zawiera relację „jeden do jednego” z modelem `User` frameworka Django oraz wszelkie dodatkowe pola. Relacja „jeden do jednego” jest podobna do stosowania pola `ForeignKey` z parametrem `unique = True`. Druga strona relacji to niejawna relacja „jeden do jednego” z powiązanym modelem zamiast menedżera wielu elementów. Z każdej strony relacji pobieramy jeden powiązany obiekt.

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

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


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

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

Utworzony profil użytkownika będzie zawierał datę urodzenia użytkownika oraz jego zdjęcie.

Do powiązania profili z użytkownikami służy pole relacji „jeden do jednego” `user`. Aby odwoływać się do modelu użytkownika, zamiast wskazywać bezpośrednio na model `auth.User`, używamy ustawienia `AUTH_USER_MODEL`. Dzięki temu kod jest bardziej generyczny, ponieważ można go obsługiwać za pomocą niestandardowych modeli użytkowników. Użyliśmy klauzuli `CASCADE` dla parametru `on_delete`, aby po usunięciu użytkownika został również usunięty powiązany z nim profil.

Pole `date_of_birth` jest typu `DateField`. Ustawiliśmy to pole jako opcjonalne za pomocą opcji `blank=True`, a dzięki opcji `null=True` dopuściliśmy wprowadzanie wartości `null`.

Zdjęcie użytkownika jest przechowywane w polu `photo` typu `ImageField`. Ustawiliśmy to pole jako opcjonalne za pomocą opcji `blank=True`. Pole `ImageField` służy do przechowywania plików ze zdjęciami. Django sprawdza, czy podany plik jest prawidłowym zdjęciem, zapisuje plik ze zdjęciem w katalogu wskazanym przez parametr `upload_to` i zapisuje względną ścieżkę do pliku w odpowiednim polu bazy danych. Pole `ImageField` w bazie danych jest domyślnie tłumaczone na kolumnę `VARHAR(100)`. Jeśli wartość pola jest pusta, w bazie danych zostanie zapisany pusty ciąg.

Instalowanie modułu Pillow i udostępnianie plików multimedialnych

W celu zarządzania obrazami konieczne będzie zainstalowanie pakietu Pythona `Pillow`. Moduł `Pillow` jest standardem de facto przetwarzania zdjęć w Pythonie. Obsługuje wiele formatów obrazów i zapewnia zaawansowane funkcje przetwarzania obrazu. Moduł `Pillow` jest wymagany przez Django do obsługi zdjęć zapisanych w polach typu `ImageField`.

Zainstaluj pakiet `Pillow` przez wydanie w powłoce poniższego polecenia.

```
python -m pip install Pillow==10.3.0
```

Przeprowadź edycję pliku *settings.py* projektu i dodaj w nim poniższe wiersze kodu.

```
MEDIA_URL = 'media/'
MEDIA_ROOT = BASE_DIR / 'media'
```

Wprowadzenie tych ustawień umożliwi frameworkowi Django zarządzanie przesyłaniem plików i udostępnianiem plików multimedialnych. 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. Ścieżki i adresy URL plików są budowane dynamicznie przez dodawanie do nich ścieżki projektu lub adresu URL multimediów, co ułatwia przenośność plików.

Teraz w głównym pliku *urls.py* projektu `bookmarks` zmodyfikuj znajdujący się w nim kod w pokazany niżej sposób. Nowe wiersze zostały wyróżnione pogrubioną czcionką.

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

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

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

Do serwowania plików multimedialnych przez serwer programistyczny Django podczas programowania (to znaczy gdy opcja `DEBUG` jest ustawiona na `True`) dodaliśmy funkcję pomocniczą `static()`.

Uwaga

Funkcja pomocnicza `static()` jest odpowiednia do stosowania w środowisku programistycznym, ale na pewno nie w produkcyjnym. Serwowanie statycznych plików przez Django jest bardzo niewydajne. Pamiętaj, aby w środowisku produkcyjnym nigdy nie udostępniać plików statycznych za pomocą Django. Sposób serwowania statycznych plików w środowisku produkcyjnym omówiłem w rozdziale 17., „Wdrożenie”.

Tworzenie migracji dla modelu Profile

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

```
python manage.py makemigrations
```

Powinieneś otrzymać taki wynik:

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

Kolejnym krokiem jest zsynchronizowanie bazy danych za pomocą poniższego polecenia:

```
python manage.py migrate
```

Wygenerowany wynik będzie zawierał m.in. następujący wiersz.

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

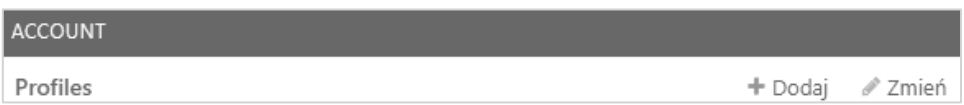
Przeprowadź edycję pliku *admin.py* aplikacji `account` i zarejestruj model `Profile` w witrynie administracyjnej, co pokazałem poniżej.

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

@admin.register(Profile)
class ProfileAdmin(admin.ModelAdmin):
    list_display = ['user', 'date_of_birth', 'photo']
    raw_id_fields = ['user']
```

Uruchom serwer programistyczny za pomocą polecenia:

```
python manage.py runserver
```

W przeglądarce internetowej przejdź pod adres `http://127.0.0.1:8000/admin/`. Teraz będziesz mógł zobaczyć model `Profile` w witrynie administracyjnej projektu, co pokazałem na rysunku 4.20.

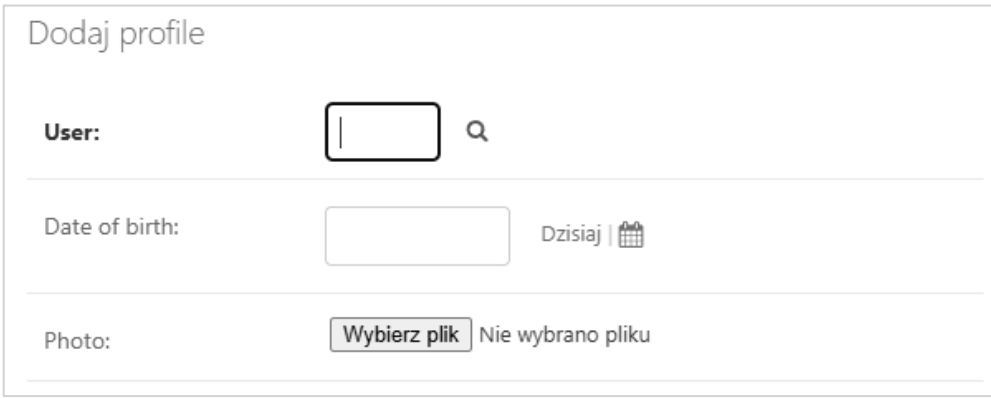


Rysunek 4.20. Blok ACCOUNT głównej strony witryny administracyjnej

W wierszu *Profiles* kliknij *Dodaj*. Wyświetli się formularz służący do dodawania nowego profilu, pokazany na rysunku 4.21.

Utwórz ręcznie obiekt `Profile` dla każdego istniejącego użytkownika w bazie danych.

Teraz zajmiemy się umożliwieniem użytkownikom edycji profilu w witrynie internetowej.



Rysunek 4.21. Formularz dodawania profilu

Przeprowadź edycję pliku *forms.py* w aplikacji `account` i umieść w nim poniższe wiersze oznaczone pogrubioną czcionką.

```
# ...
from .models import Profile

# ...

class UserEditForm(forms.ModelForm):
    class Meta:
        model = get_user_model()
        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 (tzw. awatar) dla swojego profilu.

Przeprowadź edycję pliku *views.py* w aplikacji `account` i umieść w nim poniższe wiersze oznaczone pogrubioną czcionką.

```
# ...
from .models import Profile

# ...

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

```
            user_form = UserRegistrationForm(request.POST)
            if user_form.is_valid():
                # Utworzenie nowego obiektu użytkownika; jednak 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()
                # Utworzenie profilu użytkownika
                Profile.objects.create(user=new_user)
                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}
        )
```

Gdy użytkownicy rejestrują się w witrynie, tworzony jest obiekt `Profile`, który jest powiązany z utworzonym obiektem `User`. Jednak dla użytkowników utworzonych za pośrednictwem witryny administracyjnej nie zostanie automatycznie utworzony powiązany obiekt `Profile`. W bazie danych mogą współistnieć użytkownicy z profilem, jak i bez niego (np. użytkownicy uprzywilejowani — ang. *staff users*).

Aby wymusić tworzenie profili dla wszystkich użytkowników, możesz użyć sygnałów Django. W ten sposób można zainicjować utworzenie obiektu `Profile` za każdym razem, gdy tworzony jest użytkownik. O sygnałach dowiesz się w rozdziale 7., „Śledzenie działań użytkownika", w którym w podrozdziale „Rozszerzanie projektu przy użyciu sztucznej inteligencji" wykonamy ćwiczenie implementujące tę funkcjonalność.

Teraz umożliwimy użytkownikowi edycję profilu.

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

```
from django.contrib.auth import authenticate, login
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse
from django.shortcuts import render
from .forms import (
    LoginForm,
    UserRegistrationForm,
    UserEditForm,
    ProfileEditForm
)
from .models import Profile
```

```
    # ...
    @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
            }
        )
```

Aby umożliwić użytkownikom modyfikowanie swoich danych osobowych, dodaliśmy nowy widok `edit`. Dodaliśmy do widoku dekorator `login_required`, ponieważ tylko uwierzytelnieni użytkownicy będą mogli edytować swoje profile.

W tym widoku używamy dwóch formularzy opartych na modelu `UserEditForm`, przeznaczonym do przechowywania danych wbudowanego modelu User i `ProfileEditForm`, przeznaczonym do przechowywania dodatkowych danych profilu. Aby zweryfikować poprawność danych wysłanych w formularzu, sprawdzamy, czy wartością zwrotną metody `is_valid()` w obu wymienionych formularzach jest `True`. Jeżeli oba formularze zawierają prawidłowe dane, zapisujemy zawartość obu formularzy. W tym celu wywołujemy metodę `save()`, która uaktualnia odpowiedni obiekt w bazie danych.

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

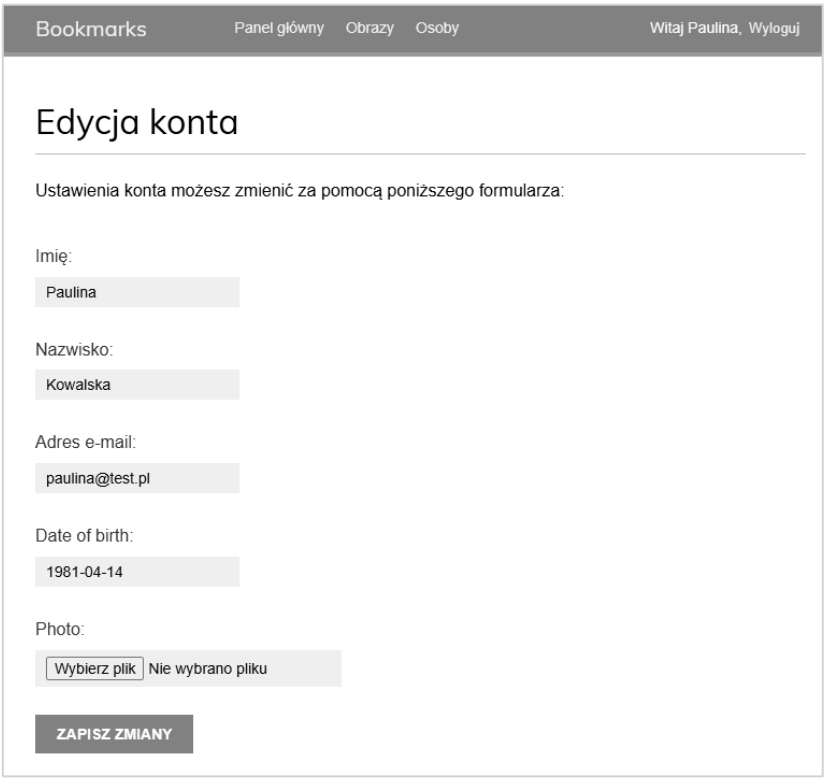
```
urlpatterns = [
    #...
    path('', include('django.contrib.auth.urls')),
    path('', views.dashboard, name='dashboard'),
    path('register/', views.register, name='register'),
    path('edit/', views.edit, name='edit'),
]
```

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 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 powyższym kodzie dodaliśmy do elementu HTML `<form>` opcję `enctype="multipart/form-data"`. Wykorzystujemy tylko jeden formularz HTML do wysłania obu formularzy Django, czyli `user_form` i `profile_form`.

Otwórz w przeglądarce stronę `http://127.0.0.1:8000/account/register/` i zarejestruj nowego użytkownika. Następnie zaloguj się jako nowy użytkownik i otwórz stronę `http://127.0.0.1:8000/account/edit/`. Powinieneś zobaczyć stronę podobną do pokazanej na rysunku 4.22.



Rysunek 4.22. Formularz edycji profilu

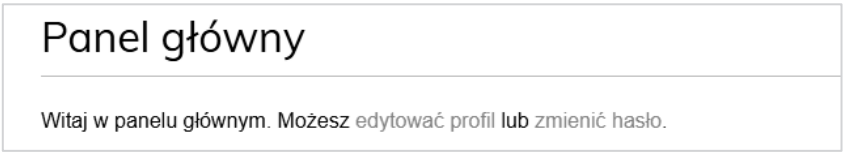
Możesz teraz dodać informacje o profilu i zapisać zmiany.

Teraz możemy zmodyfikować stronę panelu głównego i umieścić na niej linki prowadzące do stron pozwalających na edycję profilu i zmianę hasła.

Otwórz w przeglądarce szablon *templates/account/dashboard.html* i dodaj następujące wiersze wyróżnione pogrubioną czcionką.

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

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

{% block content %}
  <h1>Panel główny</h1>
  <p>
    Witaj w panelu głównym. Możesz <a href="{% url "edit" %}">edytować profil</a>
    ↪lub <a href="{% url "password_change" %}">zmienić hasło</a>.
   </p>
{% endblock %}
```

Po wprowadzonych zmianach użytkownik będzie miał z poziomu panelu głównego dostęp do formularza umożliwiającego edycję profilu. Otwórz w przeglądarce stronę `http://127.0.0.1:8000/account/` i przetestuj nowy link edycji profilu użytkownika (rysunek 4.23). Panel główny powinien teraz wyglądać jak na rysunku 4.23.



Rysunek 4.23. Zawartość strony panelu głównego razem z linkami do edycji profilu i zmiany hasła

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 o tej metodzie można przeczytać na stronie *https://docs.djangoproject.com/en/5.0/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`.

Podsumowanie

W tym rozdziale dowiedziałeś się, jak zaimplementować w budowanej witrynie internetowej system uwierzytelniania. Zaimplementowałeś wszystkie widoki potrzebne do rejestracji użytkownika, logowania i wylogowania oraz edycji i odzyskiwania hasła. Stworzyłeś również model do przechowywania niestandardowych profili użytkowników.

W następnym rozdziale poprawisz komfort użytkownika przez zaimplementowanie dostarczania informacji zwrotnych na temat jego działań za pomocą frameworka `messages` platformy Django. Rozszerzysz również zakres metod uwierzytelniania przez umożliwienie użytkownikom uwierzytelniania za pomocą adresu e-mail i zintegrowanie uwierzytelniania społecznościowego za pośrednictwem konta Google. Dowiesz się także, jak za pomocą rozszerzenia *Django Extensions* obsługiwać serwer programistyczny przez HTTPS oraz jak dostosować potok uwierzytelniania, aby automatycznie tworzyć profile użytkowników.

Dodatkowe zasoby

Oto lista zasobów zawierających dodatkowe informacje związane z tematami omawianymi w tym rozdziale:

- Kod źródłowy przykładów zamieszczonych w tym rozdziale — *https://github.com/PacktPublishing/Django-5-by-example/tree/main/Chapter04*.
- Wbudowane widoki uwierzytelniania — *https://docs.djangoproject.com/en/5.0/topics/auth/default/#all-authentication-views*.
- Wzorce adresów URL uwierzytelniania — *https://github.com/django/django/blob/stable/3.0.x/ django/contrib/auth/urls.py*.
- W jaki sposób Django zarządza hasłami i dostępnymi mechanizmami haszowania — *https://docs.djangoproject.com/en/5.0/topics/auth/passwords/*.
- Generyczny model użytkownika i metoda `get_user_model()` — *https://docs.djangoproject.com/en/5.0/topics/auth/customizing/#django.contrib.auth.get_user_model*.
- Korzystanie z niestandardowego modelu użytkownika — *https://docs.djangoproject.com/en/5.0/topics/auth/customizing/#substituting-a-custom-user-model*.


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

Django 5
Ciesz się doskonałością swoich aplikacji!

Django pozwala na pełne wykorzystanie zalet Pythona, a przy tym jest bardzo przyjemny w pracy. Dzięki niemu programiści na różnym poziomie zaawansowania mogą efektywnie tworzyć aplikacje internetowe. W corocznej ankiecie dla programistów serwisu Stack Overflow od kilku lat Django jest wybierany jako jeden z najbardziej lubianych frameworków webowych.

To piąte, zaktualizowane i uzupełnione wydanie bestsellerowego przewodnika po tworzeniu aplikacji internetowych za pomocą Django. Pokazano tu proces planowania i budowy atrakcyjnych aplikacji, rozwiązywania typowych problemów i implementacji najlepszych praktyk programistycznych. Podczas tworzenia aplikacji, takich jak blog, serwis społecznościowy, aplikacja e-commerce i platforma e-learningowa, zapoznasz się z szerokim zakresem zagadnień związanych z tworzeniem złożonych aplikacji internetowych w Pythonie. Krok po kroku, dzięki szczegółowym planom projektów, dowiesz się, jakie korzyści niesie ze sobą praca z Django 5, i zrozumiesz zasady tworzenia aplikacji przy użyciu tego frameworka.

W książce:

- podstawy Django, w tym modele, ORM, widoki, szablony, adresy URL, formularze, uwierzytelnianie, sygnały i warstwy middleware
- integracja projektu aplikacji Django z zewnętrznym oprogramowaniem
- praca z Redis, PostgreSQL, Celery, RabbitMQ i Memcached
- konfiguracja środowiska produkcyjnego za pomocą Docker Compose
- budowa RESTful API za pomocą Django Rest Framework
- implementacja zaawansowanych funkcji i tworzenie ASGI

Antonio Melé jest dyrektorem do spraw informatyki w firmie Backbase, przodującej w dziedzinie transformacji cyfrowej instytucji finansowych. Rozwija projekty Django dla klientów z różnych branż od 2006 roku. Jest uznanym mentorem startupów na wczesnym etapie rozwoju.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-1882-5
9 788328 918825
Cena: 149,00 zł
packt