

# Użytkownicy

Zajęcia Prymus 2020

Agnieszka Rudnicka, [rudnicka@agh.edu.pl](mailto:rudnicka@agh.edu.pl)

## Użytkownicy

- Django - wbudowane mechanizmy uwierzytelnienia
- Sprawdzenie konfiguracji projektu
- Podstawowe widoki użytkowników
  - Otwórzmy przeglądarkę
  - Szablon strony logowania
  - Logujemy się
  - Strona profilu zalogowanego użytkownika
- Rejestracja użytkowników
  - URL
  - Widok (logika)
    - Dlaczego sprawdzamy `request.method == "POST"`?
    - Szablony HTML
    - Zapisywanie danych z formularza
    - Finalny widok rejestracji
- Dalsze kroki
- Rozszerzamy interfejs użytkownika
- Źródła i materiały pomocnicze

Nasza dotychczasowa wersja strony posiada prosty model książek. Niestety, żeby dodać nową pozycję lub edytować istniejącą trzeba się zalogować do panelu administracyjnego. Gdybyśmy chcieli dodać możliwość oceniania książek i pisania recenzji, panel administracyjny to za mało.

Dodajmy więc możliwość rejestracji i logowania się użytkowników, którzy nie są administratorami.

## Django - wbudowane mechanizmy uwierzytelnienia

Zgodnie z dokumentacją Django: <https://docs.djangoproject.com/en/3.0/topics/auth/> framework dostarcza nam potrzebny szkielet kodu na start.

## Sprawdzenie konfiguracji projektu

Na początek sprawdzimy, czy mamy dodaną aplikację `django.contrib.auth` oraz `django.contrib.contenttypes` do `INSTALLED_APPS` w ustawieniach naszego projektu (`settings.py`).

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth', # <-- tu jest  
    'django.contrib.contenttypes', # <-- i tu  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'books',  
]
```

Następnie sprawdzimy czy mamy `SessionMiddleware` oraz `AuthenticationMiddleware` w naszych ustawieniach `MIDDLEWARE`. Znajdziemy je również w ustawieniach naszego projektu (`settings.py`)

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware', # <-- tu  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware', # <-- i tu  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]
```

Następnie zalecane jest zaaplikowanie migracji, które stworzą w bazie danych tabele na użytkowników (`manage.py migrate`). Dla nas jednak nic nowego się nie powinno wykonać, bo mieliśmy już zainstalowaną aplikację użytkowników od początku - i tabele użytkowników zostały stworzone razem z tabelami książek i innymi.

Gdybyśmy mieli źle skonfigurowaną aplikację użytkowników nie moglibyśmy się zalogować do panelu administracyjnego.

## Podstawowe widoki użytkowników

Teraz kiedy upewniliśmy się, że mamy poprawnie skonfigurowaną aplikację do obsługi użytkowników pora zobaczyć co potrafi! Będziemy się opierać głównie na przykładach z [oficjalnej dokumentacji](#), nie ma tu żadnej "magii" :)

Otwórzmy nasz plik `urls.py`, gdzie znajdują się wszystkie obsługiwane ścieżki.

Django dostarcza kilka widoków, które zajmują się podstawowym zarządzaniem sesją użytkownika. Mamy tutaj:

- logowanie
- wylogowanie
- zmianę hasła
- reset hasła (wymaga dodatkowej konfiguracji m.in. wysyłki email)

Wszystkie te widoki znajdują się w paczce `django.contrib.auth.urls`. Dołączmy je do naszej aplikacji dodając następujący kod do pliku `urls.py` do listy `urlpatterns`:

```
urlpatterns = [  
    path('accounts/', include('django.contrib.auth.urls')), # <-- nowe  
  
    path("admin/", admin.site.urls),  
    ... # pozostałe nasze widoki  
]
```

Musimy jeszcze zamieścić import funkcji `include`. Dodajmy więc w pierwszych liniach pliku:

```
from django.urls import include
```

Funkcja `include` pozwala nam zaimportować i dołączyć do routingu naszej aplikacji wszystkie widoki zdefiniowane w `django.contrib.auth.urls`. Dzięki temu nasz projekt użyje już raz zaprogramowanych przez twórców Django widoków.

W podobny sposób możemy dołączać widoki innych aplikacji/bibliotek dodanych do projektu lub innych naszych aplikacji. Czasem bowiem plik `urls.py` dzieli się na kilka mniejszych, żeby nie były zbyt długie/przytłaczające :)

## Otwórzmy przeglądarkę

Aktywujemy nasze środowisko wirtualne, jeśli jeszcze tego nie zrobiliśmy lub nasze IDE nas nie wyręczyło. Uruchommy naszą aplikację poleceniem `python manage.py runserver`. Następnie otworzymy przeglądarkę na standardowym adresie <http://127.0.0.1:8000/>.

Jeśli działa, to sprawdzamy dalej. Dodaliśmy widoki logowania pod adresem `/accounts/`, więc przejdźmy w przeglądarce pod <http://127.0.0.1:8000/accounts/>.

### Page not found (404)

Request Method: GET

Request URL: <http://127.0.0.1:8000/accounts/>

Using the URLconf defined in `libraryproj.urls`, Django tried these URL patterns, in this order:

1. admin/
2. accounts/ login/ [name='login']
3. accounts/ logout/ [name='logout']
4. accounts/ password\_change/ [name='password\_change']
5. accounts/ password\_change/done/ [name='password\_change\_done']
6. accounts/ password\_reset/ [name='password\_reset']
7. accounts/ password\_reset/done/ [name='password\_reset\_done']
8. accounts/ reset/<uidb64>/<token>/ [name='password\_reset\_confirm']
9. accounts/ reset/done/ [name='password\_reset\_complete']
10. [name='index']
11. ksiazki [name='book\_list']
12. books
13. ksiazki/<int:book\_id> [name='book\_details']

The current path, `accounts/`, didn't match any of these.

Blisko, pod tym adresem nic nie ma, ale widzimy, że na liście "dostępnych" URLi znajdują się nowe widoki, w tym `accounts/login/`.

Przejdźmy pod <http://127.0.0.1:8000/accounts/login/>

# TemplateDoesNotExist at /accounts/login/ registration/login.html

Request Method: GET  
Request URL: http://127.0.0.1:8000/accounts/login/  
Django Version: 3.0.5  
Exception Type: TemplateDoesNotExist  
Exception Value: registration/login.html

To już znamy 😊 O ile wbudowana appka Django dostarcza nam widoków/logiki, to nie narzuca nam szablonów HTML. Jednak w [dokumentacji](#) można znaleźć startowy kawałek kodu.

## Szablon strony logowania

Utwórzmy zatem plik `templates/registration/login.html`. Ponieważ to szablon HTML - musi się on znajdować w katalogu `templates` naszej aplikacji. Django zakłada, że HTML będzie w podkatalogu `registration`.

Oto snippet kodu, którego użyjemy:

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

{% block content %}

{% if form.errors %}
<p>Your username and password didn't match. Please try again.</p>
{% endif %}

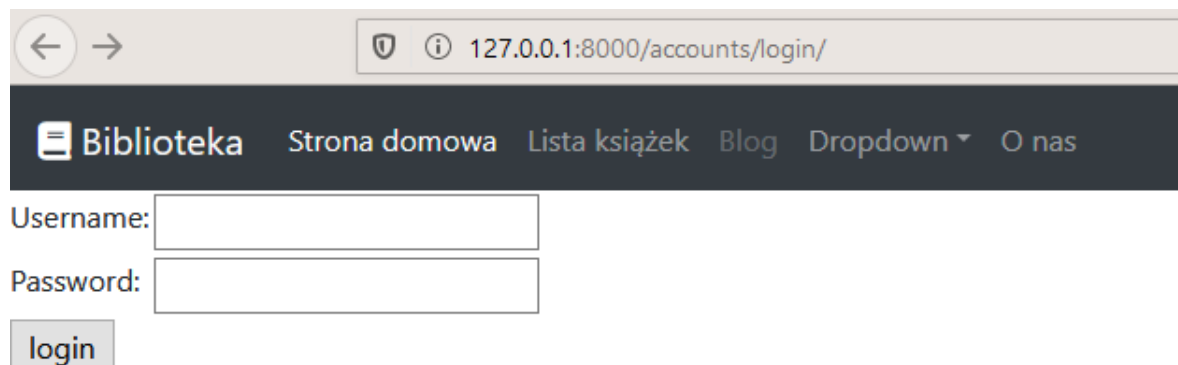
{% if next %}
    {% if user.is_authenticated %}
    <p>Your account doesn't have access to this page. To proceed,
    please login with an account that has access.</p>
    {% else %}
    <p>Please login to see this page.</p>
    {% endif %}
{% endif %}

<form method="post" action="{% url 'login' %}">
{% csrf_token %}
<table>
<tr>
    <td>{{ form.username.label_tag }}</td>
    <td>{{ form.username }}</td>
</tr>
<tr>
    <td>{{ form.password.label_tag }}</td>
    <td>{{ form.password }}</td>
</tr>
</table>

<input type="submit" value="login">
<input type="hidden" name="next" value="{{ next }}">
</form>

{% endblock %}
```

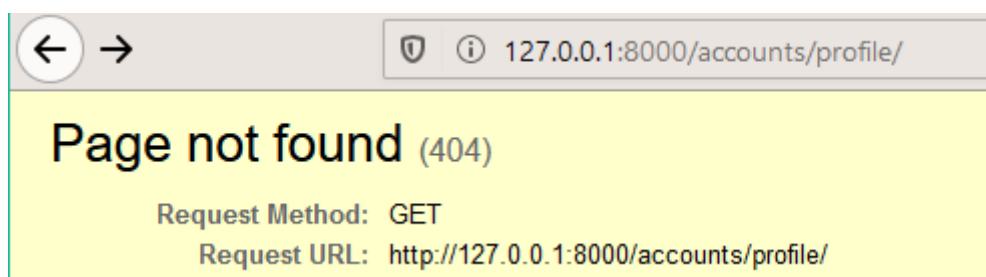
Jeśli plik został dobrze nazwany, to po odświeżeniu strony w przeglądarce powinniśmy ujrzeć formularz logowania, mój wygląda jak poniżej:



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8000/accounts/login/`. The page has a dark navigation bar with the following links: **Biblioteka**, **Strona domowa**, **Lista książek**, **Blog**, **Dropdown ▾**, and **O nas**. Below the navigation bar, there is a login form with two input fields: **Username:** and **Password:**. A **login** button is positioned below the password field.

## *Logujemy się*

Możemy oczywiście skorzystać z naszych danych logowania do panelu administracyjnego (jeśli ktoś nie ma konta python `manage.py createsuperuser`).



Dostaniemy jednak błąd 404 po poprawnym logowaniu. Dzieje się tak ponieważ domyślnie zostajemy przekierowani na stronę profilu zalogowanego użytkownika.

## *Strona profilu zalogowanego użytkownika*

Jednym z rozwiązań jest dodanie widoku profilu aktualnie zalogowanego użytkownika.

Inne to zmiana adresu przekierowania [https://docs.djangoproject.com/en/3.0/ref/settings/#std:setting-LOGIN\\_REDIRECT\\_URL](https://docs.djangoproject.com/en/3.0/ref/settings/#std:setting-LOGIN_REDIRECT_URL) Czyli np ustawienie `LOGIN_REDIRECT_URL="/"` w naszym pliku `settings.py`.

Skupmy się jednak na tym pierwszym, w naszym pliku `urls.py` potrzebujemy dodać nowy widok, który obsłuży zapytania kierowane na adres `accounts/profile/` naszej aplikacji. Dodajmy więc odpowiednią ścieżkę:

```
path('accounts/profile/', views.profile_view, name='user_profile'),
```

do istniejących już w liście `urlpatterns`.

Taka funkcja `profile_view` oczywiście jeszcze nie istnieje, musimy ją napisać :)

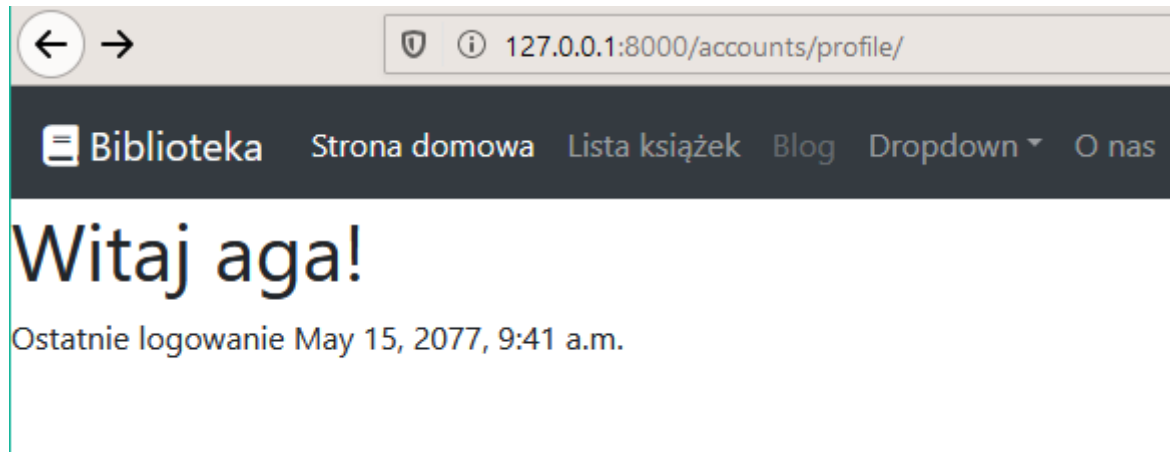
Spróbujcie napisać widok i podpiąć do niego szablon, który wyświetla nazwę zalogowanego użytkownika oraz datę ostatniego logowania. Podpowiedź:

Datę ostatniego logowania wyświetlimy w HTML:

```
<p>Ostatnie logowanie {{ request.user.last_login }}</p>
```

Spis wszystkich pól modelu użytkownika znajdziecie w dokumentacji: <https://docs.djangoproject.com/en/3.0/ref/contrib/auth/#django.contrib.auth.models.User>

Przykładowy wygląd strony profilowej:



## Rejestracja użytkowników

Skoro możemy się już zalogować pora na dodanie rejestracji. Potrzebujemy kilku rzeczy:

1. szablonu HTML z formularzem rejestracji
2. widoku, który przetworzy dane nowego użytkownika i stworzy mu konto
3. szablonu HTML potwierdzającego poprawne założenie konta

Oczywiście metod na zaimplementowanie przebiegu rejestracji jest mnóstwo, to tylko jedna z nich :)

### *URL*

Standardowo potrzebujemy dodać nową ścieżkę do `urls.py`, tym razem niech to będzie:

```
path('accounts/signup/', views.user_signup, name="user_signup"),
```

### *Widok (logika)*

Następnym krokiem jest zaimplementowanie funkcji, która obsłuży dane rejestrowanego użytkownika. Możemy w tym przypadku wykorzystać `UserCreationForm` czyli przygotowany przez autorów Django formularz rejestracji.

Zaimportujmy go na początku pliku `views.py` dodając:

```
from django.contrib.auth.forms import UserCreationForm
```

A następnie przejdźmy do tworzenia widoku:

```
def user_signup(request):
    if request.method == 'POST':
        ... # tu trzeba przetworzyć dane z formularza
    else:
        # tutaj obsługujemy przypadek kiedy użytkownik pierwszy raz wyświetlił stronę
        form = UserCreationForm()

    # na końcu zwracamy wyrenderowanego HTMLa
    return render(request, template_name="registration/signup_form.html", context=
{'form': form})
```

Dzieje się tutaj kilka rzeczy.

Jeśli zapytanie ma metodę "POST" oznacza to zwykle, że dostaliśmy jakieś dane zakodowane w zapytanie HTTP (ogromne uproszczenie). Zwykle zapytań HTTP-POST używa się do przesłania formularzy, które to mają w efekcie coś nowego stworzyć w bazie danych.

W tym przypadku tworzony będzie nowy użytkownik z danych formularza (`UserCreationForm`) przesłanych w zapytanie HTTP z przeglądarki. `UserCreationForm` to coś co pozwala na wyświetlenie formularza w HTML oraz zmapowanie danych od użytkownika na login i hasło przyszłego użytkownika.

## Dlaczego sprawdzamy `request.method == "POST"`?

Użytkownikowi, który pierwszy raz wszedł na stronę rejestracji chcemy wyświetlić pusty, "świeży" formularz rejestracji. Przeglądarki domyślnie wysyłają zapytanie metodą [GET](#). Jest to typowe zapytanie przeglądarki "podaj mi zawartość tej strony, chcę ją wyświetlić mojemu użytkownikowi".

Jeżeli użytkownik kliknął "zarejestruj się" po wypełnieniu formularza - przeglądarka wyśle zapytanie metodą POST - Django odbierze dane formularza zakodowane w zapytaniu POST a następnie odpowiednio je przetworzy. Z tych przetworzonych danych będziemy mogli "stworzyć" nowego użytkownika w naszej bazie danych.

## Szablony HTML

Dla porządku stwórzmy sobie dwa szablony HTML, jeden na formularz rejestracji i drugi na wyświetlenie podziękowania za rejestrację na naszej stronie.

Potrzebujemy więc:

```
registration/signup_form.html
```

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

{% block content %}
    <form method="post">
        {% csrf_token %}

        {{ form }}

        <hr>

        <button class="btn btn-primary">Zarejestruj się</button>
    </form>
{% endblock %}
```

Ten szablon HTML wyświetli nam formularz rejestracji (`{{ form }}`), który przekażemy w kontekście z widoku, który właśnie piszemy. Formularz potrzebuje być otoczony specyficznym tagiem HTML `<form>` `</form>`.

Ważne jest też dodanie metody wysyłki danych z formularza `<form method="post">` poprzez atrybut `method` oraz oczywiście zamieszczenie przycisku, żeby można było zatwierdzić wpisane dane :)

`{% csrf_token %}` wygeneruje nam token CSRF. Jest to popularne zabezpieczenie przed atakami *Cross-site request forgery*. Dla zainteresowanych polecam np <https://sekurak.pl/czym-jest-podatnosc-csrf-cross-site-request-forgery/>

The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8000/accounts/signup/`. The page has a dark navigation bar with links: Biblioteka, Strona domowa, Lista książek, Blog, Dropdown, O nas, and Zaloguj się. The main content area contains a registration form. It starts with a 'Username:' label and an input field. Below it is a 'Password:' label with an input field. To the right of the password field, there is a list of requirements: 'Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only.' and a bulleted list: 'Your password can't be too similar to your other personal information.', 'Your password must contain at least 8 characters.', 'Your password can't be a commonly used password.', and 'Your password can't be entirely numeric.' Below the password field is a 'Password confirmation:' label and an input field. Underneath that is the text 'Enter the same password as before, for verification.' At the bottom of the form is a blue button labeled 'Zarejestruj się'.

Drugi szablon `registration/signup_complete.html` będzie znacznie prostszy:

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

{% block content %}
    <h1>Dziękujemy za rejestrację na naszej stronie!</h1>

    <a href="{% url 'login' %}" class="btn btn-success">Zaloguj się</a>
{% endblock %}
```



# Dziękujemy za rejestrację na naszej stronie!

Zaloguj się

## Zapisywanie danych z formularza

Mamy dwa szablony oraz szkielet widoku. Jednak jeśli wypełnimy formularz i zatwierdzimy to użytkownik się nie tworzy. Dla kodu `def user_signup(request):` z powyżej nawet dostajemy błąd. Dopiszmy więc brakujące przetwarzanie formularza.

```
if request.method == 'POST':
    # tu trzeba przetworzyć dane z formularza
    form = UserCreationForm(request.POST)
    if form.is_valid():
        form.save()
        return render(request, template_name="registration/signup_complete.html")
```

Uwaga na wcięcia! To wszystko ma się wykonać dla metody POST.

Przeanalizujmy. Wczytajmy do naszego formularza (`UserCreationForm`) dane z zapytania HTTP (`request.POST`) od użytkownika:

```
form = UserCreationForm(request.POST)
```

Następnie sprawdzimy, czy dane w formularzu są poprawne. W tym przypadku jest to porównanie hasła 1 i hasła 2, sprawdzenie czy są wypełnione wszystkie pola itp. Następnie zapiszmy formularz tym samym tworząc nowego użytkownika w bazie danych.

Tak działają formularze w Django - pozwalają nam wczytać dane od użytkownika i na ich bazie stworzyć nowy obiekt w bazie danych. Wszystkim zajmuje się za kulisami Django. My tylko deklarujemy co i z czego.

## Finalny widok rejestracji

Po wszystkich modyfikacjach widok rejestracji powinien wyglądać następująco:

```
def user_signup(request):
    if request.method == 'POST':
        # tu trzeba przetworzyć dane z formularza
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return render(request, template_name="registration/signup_complete.html")
    else:
        # tutaj obsługujemy przypadek kiedy użytkownik pierwszy raz wyświetlił stronę
        form = UserCreationForm()

    # na końcu zwracamy wyrenderowanego HTMLa
    return render(request, template_name="registration/signup_form.html", context=
        {'form': form})
```

Spróbujmy się teraz zarejestrować, a następnie zalogować na nowe konto :)

## Dalsze kroki

Warto również dodać swój szablon dla strony wylogowania tworząc szablon `registration/logged_out.html`.

Uwaga: żeby pomyślnie nadpisać szablon dostarczany przez bibliotekę, taki jak widok wylogowania się, trzeba upewnić się, że nasza aplikacja jest przez panelem administracyjnym w `INSTALLED_APPS`.

```
INSTALLED_APPS = [  
    'books', # <-- PRZED django.contrib.admin  
    'django.contrib.admin', # django.contrib.admin  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]
```

Dzięki temu szablony z naszej aplikacji `books` będą miały priorytet nad wszystkimi dostarczonymi przez pozostałe aplikacje :) W szczególności pozwoli nam to na dostosowanie wyglądu panelu administracyjnego i inne.

## Rozszerzamy interfejs użytkownika

W tej chwili linki do logowania i wylogowania użytkownika nie są umieszczone nigdzie w interfejsie. Poprawmy więc UX (user-experience) naszej strony dodając odpowiednie linki w pasku nawigacji.

Tutaj przyda nam się tag `{% url 'nazwa_widoku' %}` oraz lista dostępnych: <https://docs.djangoproject.com/en/3.0/topics/auth/default/#module-django.contrib.auth.views>

## Źródła i materiały pomocnicze

- <https://docs.djangoproject.com/en/3.0/topics/auth/>
- <https://docs.djangoproject.com/en/3.0/ref/contrib/auth/#django.contrib.auth.models.User>
- <https://docs.djangoproject.com/en/3.0/ref/templates/builtins/>
- <https://learndjango.com/tutorials/django-signup-tutorial>