**3 Szkolenie z Flaska**

**Requesty, przesyłanie danych z formularza, sesje**

**Metody HTTP**

W szkoleniu tym omówimy temat związany z podstawowymi **metodami HTTP** - **GET, POST** - które mogą służyć do skutecznego przetwarzania i przekazywania danych i informacji między **klientem** a **serwerem**.

Metody te będziemy więc mogli wykorzystać w celu wprowadzenia, np. **formularzy logowania** w tworzonej aplikacji.

Zapytania HTTP mogą bazować również na wielu innych metodach, które pokrótce opisuje poniższa tabela:

| Nazwa metody: | Zastosowanie: |
| --- | --- |
| **GET** | Pobieranie zasobu lub jego wyświetlanie, np. wyświetlenie szablonu strony. |
| **POST** | Przesyłanie danych zapisanych w postaci **klucz-wartość** do serwera. Kluczem jest nazwa danego pola, a wartością wpisana przez Nas wartość do danego pola. Metoda ta jest również wykorzystywana przy wysyłaniu plików na serwer. POST powoduje uruchomienie określonych operacji ustalonych przez serwer po przejściu na dany endpoint (URI). |
| **PUT** | Służy do przesyłania paczki danych bez powiązania klucz-wartość. Wykorzystywana jest najczęściej przy RESTowych rozwiązaniach (tworzenie i update’owanie danych), gdzie ciałem danych jest formularz w postaci JSONa. |
| **DELETE** | Usuwa zasoby na serwerze. Dlatego jest domyślnie wyłączona i uniemożliwia ingerencję w zasoby “z zewnątrz”. |
| **HEAD** | HEAD często jest wysyłany z tzw. “preflight requestami”. Służy on do sprawdzenia nagłówków zwracanego response z serwera, np. gdy potrzebujemy sprawdzić, jakiego rozmiaru będzie zwrócona odpowiedź bez pobierania jej ciała (przydatne, gdy nie chcemy danych o dużym rozmiarze). |

**Tworzenie formularza. GET & POST**

Zacznijmy od zbudowania prostego formularza, w którym użytkownik będzie mógł podać swoje imię (zalogować się).

Po wypełnieniu danych, zostanie on przekierowany na stronę, w której zostanie powitany odpowiednim komunikatem.

| from flask import Flask   def create\_app():  app = Flask(\_\_name\_\_)  app.debug = True    from .main import login\_blueprint, user\_blueprint  app.register\_blueprint(login\_blueprint)  app.register\_blueprint(user\_blueprint)    return app |
| --- |

Plik \_\_init\_\_.py

| from flask import render\_template, redirect, url\_for, Blueprint, request ### 0  login\_blueprint = Blueprint("login", \_\_name\_\_) user\_blueprint = Blueprint("user", \_\_name\_\_)  @login\_blueprint.route('/login', methods=["POST", "GET"]) ### 0 def login():  if request.method == "POST": ### 1  nickname = request.form['nickname'] ### 2  return redirect(url\_for("user.user", nick=nickname))  else: ### 3  return render\_template("login.html")   @user\_blueprint.route('/<nick>') def user(nick):  return f"<h1>Hello, {nick}</h1>" |
| --- |

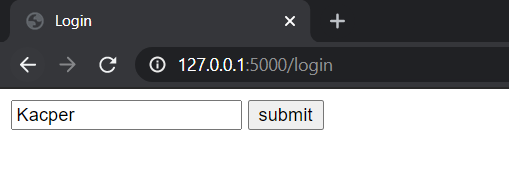
Plik main.py

| {% extends "base.html" %} {% block title %} Login {% endblock %} {% block content %}  <**form** action="#" method="POST">  <**input** type="text" name="nickname" />  <**input** type="submit" value="submit" />  </**form**> {% endblock %} |
| --- |

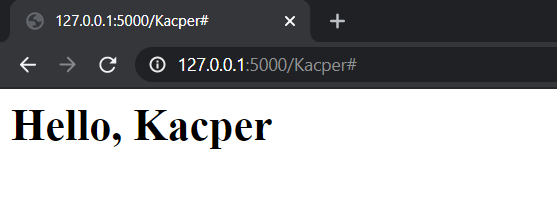
Plik login.html

| <!DOCTYPE html> <**html** lang="en"> <**head**>  <**meta** charset="UTF-8">  <**title**>{% block title %}{% endblock %}</**title**> </**head**> <**body**>  {% block content %} {% endblock %} </**body**> </**html**> |
| --- |

Plik base.html



Efekt 1



Efekt 2

Nowe zagadnienia, których jeszcze nie omówiliśmy, znajdują się w plikach index.html oraz main.py.

**Plik index.html:**

W pliku index.html wprowadziliśmy formularz, który może wypełniać użytkownik i wysyłać przez przycisk submit.   
  
Zwróć uwagę na to, jak został zbudowany:

| <**form** action="#" method="POST">  <**input** type="text" name="nickname" />  <**input** type="submit" value="submit" /> </**form**> |
| --- |

* **action=”#”** określa, dokąd ma zostać przeniesiony użytkownik, gdy zatwierdzi swój wybór klawiszem submit. **#** mówi, iż ma on zostać przekierowany na tę samą stronę, z którego dokonał wyboru (i właśnie z jej poziomu mają być podejmowane dalsze kroki, co ma się zadziać co do działania aplikacji)
* **method=”POST”** określa, jak mają zostać wysłane dane z formularza do serwera
* parametr **name** w znaczniku **<input type=”text”>** określa, z jakim kluczem ma być kojarzona wysyłana wartość z poziomu formularza; określenie takiego klucza jest niezbędne w celu obsługiwania odpowiednich danych z formularza z poziomu logiki programu (plik main.py).

**Plik main.py:**

W pliku main.py zwróć szczególną uwagę na linie, do których dodano komentarze ###1, ###2, ###3.

Dodanie do funkcji login() instrukcji warunkowej if, która bada request.method jest niezbędne z racji tego, iż musimy rozróżniać, kiedy protokół HTTP operuje na metodzie GET (standardowe wygenerowanie szablonu strony) lub POST (po wysłaniu formularza).

Wtedy musimy odpowiednio albo pobrać dane z formularza (###2) bazując właśnie na parametrze name ze znacznika input w index.html, albo wygenerować szablon strony wraz z formularzem.

Tak więc linia nickname = request.form['nickname'] pobiera wartość tekstową wpisaną w pole input z omawianego formularza. Po pobraniu tej informacji, użytkownik przekierowywany jest pod adres /<nick>, gdzie odbywa się wyświetlanie stosownego komunikatu.

Krok spod ###3 wykonuje się w momencie, gdy jeszcze nie wypełnimy formularza, czyli gdy chcemy wygenerować standardowy widok strony o URI /login.

I na zakończenie - standardowo pamiętaj o zaimportowaniu obiektu request (###0).

**Jak przechowywać informacje w sesji?**

Jeżeli stworzyłeś własny program zgodny z powyższym przykładem obsługujący dane z formularza, to na pewno zauważyłeś, że aplikacja **nie jest w stanie zapamiętywać** wpisanych danych do formularza.

To znaczy, że przy każdym przejściu na adres /login, będziemy proszeni o wypełnienie formularza. Może warto by było zastanowić się, jak zasymulować proces logowania się i przechowywać wprowadzone dane do momentu, kiedy użytkownik ręcznie nie wyloguje się z sesji?

Do zrealizowania tego celu posłuży nam właśnie obiekt session, który długofalowo przechowuje określone dane po stronie użytkownika (w plikach cookie).

Przykład: *Budowa aplikacji z formularzem na imię i z zakładkami: login, dashboard, logout.*

| from flask import Flask from hashlib import md5 ### 0  def create\_app():  app = Flask(\_\_name\_\_)  encryptor = md5() ### 0.1  app.debug = True  app.secret\_key = encryptor.digest() ### 1   from .main import login\_blueprint, dashboard\_blueprint, logout\_blueprint  app.register\_blueprint(login\_blueprint)  app.register\_blueprint(dashboard\_blueprint)  app.register\_blueprint(logout\_blueprint)   return app |
| --- |

Plik \_\_init\_\_.py

| from flask import render\_template, redirect, url\_for, Blueprint, request, session ### 0  login\_blueprint = Blueprint("login", \_\_name\_\_) dashboard\_blueprint = Blueprint("dashboard", \_\_name\_\_) logout\_blueprint = Blueprint("logout", \_\_name\_\_)   @login\_blueprint.route('/', methods=["POST", "GET"]) def login():  if request.method == "POST":  session.update({"nick" : request.form['nickname']}) ### 1  elif request.method == "GET" and "nick" not in session:  return render\_template("login.html")   return redirect(url\_for("dashboard.dashboard"))   @logout\_blueprint.route('/logout') def logout():  if "nick" in session:  session.pop("nick") ### 2   return redirect(url\_for("login.login"))   @dashboard\_blueprint.route('/dashboard') def dashboard():  if "nick" in session: ### 3  return render\_template("dashboard.html", nickname=session["nick"])   return redirect(url\_for("login.login")) |
| --- |

Plik main.py

| <!doctype html> <**head**>  <**link** href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl" crossorigin="anonymous">  <**title**>{% block title %}{% endblock %}</**title**> </**head**>  <**body**> <**nav** class="navbar navbar-expand-lg navbar-light bg-light">  <**div** class="container-fluid">  <**a** class="navbar-brand" href="#">Navbar</**a**>  <**button** class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">  <**span** class="navbar-toggler-icon"></**span**>  </**button**>  <**div** class="collapse navbar-collapse" id="navbarNav">  <**ul** class="navbar-nav">  <**li** class="nav-item">  <**a** class="nav-link active" aria-current="page" href="/">Login</**a**>  </**li**>  <**li** class="nav-item">  <**a** class="nav-link" href="/dashboard">Dashboard</**a**>  </**li**>  <**li** class="nav-item">  <**a** class="nav-link" href="/logout">Logout</**a**>  </**li**>  <**li** class="nav-item">  <**a** class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Disabled</**a**>  </**li**>  </**ul**>  </**div**>  </**div**> </**nav**> {% block content %} {% endblock %} <**script** src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.6.0/dist/umd/popper.min.js" integrity="sha384-KsvD1yqQ1/1+IA7gi3P0tyJcT3vR+NdBTt13hSJ2lnve8agRGXTTyNaBYmCR/Nwi" crossorigin="anonymous"></**script**> <**script** src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/js/bootstrap.min.js" integrity="sha384-nsg8ua9HAw1y0W1btsyWgBklPnCUAFLuTMS2G72MMONqmOymq585AcH49TLBQObG" crossorigin="anonymous"></**script**> </**body**> |
| --- |

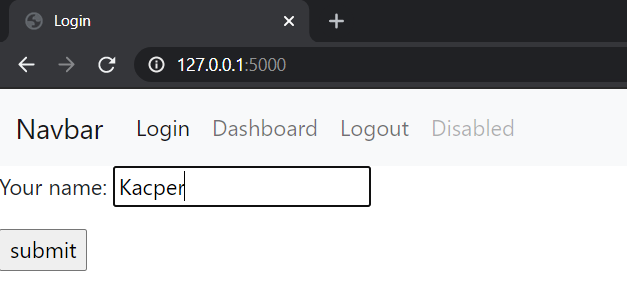
Plik base.html

| {% extends "base.html" %} {% block title %} Login {% endblock %} {% block content %}  <**form** action="#" method="POST">  <**p**>Your name: <**input** type="text" name="nickname" /></**p**>  <**p**><**input** type="submit" value="submit" /></**p**>  </**form**> {% endblock %} |
| --- |

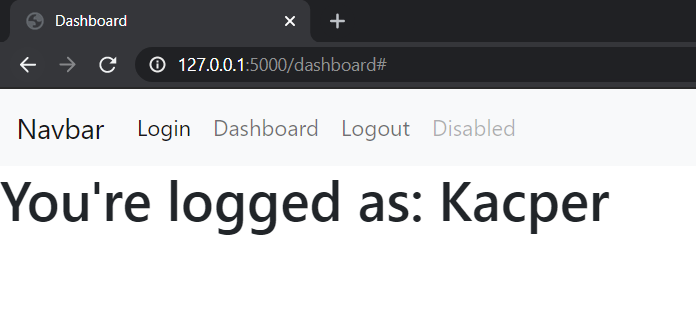
Plik login.html

| {% extends "base.html" %} {% block title %} Dashboard {% endblock %} {% block content %} <**h1**>You're logged as: {{nickname}}</**h1**> {% endblock %} |
| --- |

Plik dashboard.html



Strona główna



Strona /dashboard

**Wyjaśnienie:**

W **pliku \_\_init\_\_.py** możesz zauważyć nowe linie takie jak:

1. from hashlib import md5
2. encryptor = md5()
3. app.secret\_key = encryptor.digest()

Aby dobrze zrozumieć genezę powstania tychże fragmentów kodu, musimy zauważyć, że ich wprowadzanie wyniknęło z dodania sesji przechowującej dane (o której więcej za chwilę) do Naszej aplikacji.

Sesja jest powiązana z plikami cookie. To znaczy, że jeżeli odpowiednio nie zaszyfrujemy danych przechowywanych w sesji, aplikacja będzie narażona na ujawnienie wrażliwych informacji podanych w formularzu.

Stąd właśnie wynika konieczność ustawienia pola **secret\_key** obiektu app na wartość tekstową pełniącą rolę **szyfru/hasła**.

Pole to możemy ustawiać tak naprawdę na dowolną wartość, jednak chcąc zapewnić jak największe bezpieczeństwo, skorzystaliśmy z biblioteki hashlib, która dostarcza funkcjonalność **szyfrowania przez MD5**.

**Plik main.py to serce programu…**

Właśnie w nim odbywa się obsługiwanie (tworzenie i usuwanie) danych sesji. Obiekt sesji importujemy przez linię from flask import … session, a następnie wykonujemy na nim operacje tak jak na normalnym słowniku (session de facto jest typem słownikowym, przechowującym pary klucz : wartość).

Do linii, w których odbywają się bezpośrednio operacje na session, dodałem komentarze ### 1, ### 2, ### 3. Jedyna trudność, jaka może wynikać z powyższego przykładu, to zaprogramowanie logiki aplikacji tak, aby:

1. Przekierowywać użytkownika na stronę logowania w momencie, gdy chce przejść do dashboard, nie podając uprzednio imienia.
2. Przekierowywać użytkownika do dashboard w momencie, gdy jest zalogowany i chce przejść na stronę logowania.
3. Przekierować go do strony logowania po wylogowaniu.

Dlatego zapoznaj się dokładnie z całym modelem rozwiązania, przeanalizuj szczególnie instrukcje warunkowe i logikę aplikacji.

Odnosząc się natomiast do plików .html, korzystamy z rozwiązania, które już poznaliśmy. Zwróć jednak uwagę na zmodyfikowany atrybut href w znaczniku <a> znajdującym się w menu wyboru, np. <a class="nav-link" **href="/dashboard"**>Dashboard</a>.

Dzięki takiemu rozwiązaniu możemy łatwo przechodzić między danymi podstronami (Dashboard, Login, Logout).

**Czas życia danych w sesji?**

Jeżeli zbudowałeś już aplikację podobną do tej powyżej, to na pewno zauważyłeś, iż po zamknięciu przeglądarki i jej ponownym uruchomieniu, dane przechowywane w sesji są czyszczone i użytkownik proszony jest o podanie swojego imienia jeszcze raz.

W celu ustawienia czasu trwania sesji, wystarczy skonfigurować pole obiektu **app** - **permanent\_session\_lifetime** i ustawić je na odpowiednią długość życia.

Zmień więc plik \_\_init\_\_.py w następujący sposób:

| from flask import Flask from hashlib import md5 from datetime import timedelta ### 0  def create\_app():  app = Flask(\_\_name\_\_)  app.permanent\_session\_lifetime = timedelta(minutes=30) ### 1  encryptor = md5()  app.debug = True  app.secret\_key = encryptor.digest()   from .main import login\_blueprint, dashboard\_blueprint, logout\_blueprint  app.register\_blueprint(login\_blueprint)  app.register\_blueprint(dashboard\_blueprint)  app.register\_blueprint(logout\_blueprint)   return app |
| --- |

A w pliku main.py, tuż przed “zalogowaniem” dodaj:  
session.permament = True

| @login\_blueprint.route('/', methods=["POST", "GET"]) def login():  if request.method == "POST":  session.permanent = True  session.update({"nick" : request.form['nickname']}) ### 1  elif request.method == "GET" and "nick" not in session:  return render\_template("login.html")   return redirect(url\_for("dashboard.dashboard")) |
| --- |

Zwróć uwagę na:

1. Konieczność zaimportowania biblioteki, której funkcjonalność umożliwi Nam określenie czasu trwania sesji (timedelta).
2. Ustawienie czasu życia sesji następuje w linii app.permanent\_session\_lifetime = timedelta(minutes=30) i określamy, iż dane mają nie być kasowane przez okres 30 minut od zapamiętania ich w plikach cookie.

Po wprowadzeniu powyższych zmian w kodzie, od tej pory nie będzie potrzeby uzupełnienia na nowo formularza po opuszczeniu budowanej aplikacji! Mega praktyczne rozwiązanie!