**2 Szkolenie z Flaska**

**Routing, zmienne, szablony HTML**

**Nowe funkcjonalności**

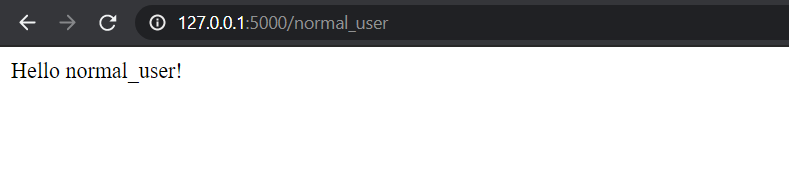
Rozszerzmy przykład ze szkolenia nr 1 o kilka dodatkowych podstron i funkcjonalności. Pokażę Ci również, jak przekazywać wartości pewnych zmiennych po adresie URL strony.

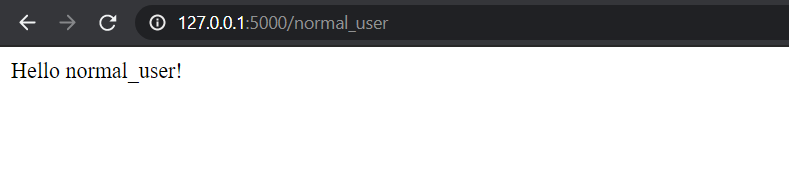
**Plik main.py:**

| from flask import Blueprint, redirect, url\_for #### 0  index\_blueprint = Blueprint('index', \_\_name\_\_) user\_blueprint = Blueprint('user', \_\_name\_\_) admin\_blueprint = Blueprint('admin', \_\_name\_\_)   @index\_blueprint.route('/') def index():  return "Hello World!"   @user\_blueprint.route('/<name>') #### 1 def user(name):  return f"Hello {name}!"   @admin\_blueprint.route('/admin') def admin():  return redirect(url\_for("user.user", name='Admin')) #### 2 |
| --- |

**Plik \_\_init\_\_.py**

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

Po wpisaniu adresu 127.0.0.1:5000/normal\_user

Po wpisaniu adresu 127.0.0.1:5000/admin

Zwróć szczególną uwagę na linie kodu, do których dodaliśmy komentarze #### 0, #### 1, #### 2.

**Parametry**

Technika, którą zastosowaliśmy w dekoratorze funkcji user() t**o przekazywanie parametru w adresie URL.**

Zmienne umieszczane w adresie zapisujemy między **ostrymi klamrami: <>.** Ich nazwy musimy podać również **jako argumenty** funkcji:

| @user\_blueprint.route('/<name>')  def user(name):  ... |
| --- |

Opcjonalnie, do tak dodanego parametru możemy również dodać jego statyczne typowanie, zapisując:

**<type:variable\_name>**

| @user\_blueprint.route('/<string:name>')  def user(name):  ... |
| --- |

Możliwe do użycia typy:

| **string** | domyślny - dowolny tekst bez znaku slash (/) |
| --- | --- |
| **int** | dowolna dodatnia liczba całkowita |
| **float** | dowolna dodatnia liczba typu float |
| **path** | tak jak string, ale również z slash-em |

**Redirect(), url\_for()**

Funkcja admin() z początkowego przykładu przekierowuje użytkownika za pomocą metody **redirect** do funkcji user() przez adres **localhost:5000/Admin**.

O ile nazwa **redirect()** jasno mówi, za co odpowiada (przekierowywanie), to tak **url\_for** może budzić kontrowersje.

Otóż metoda ta umożliwia łatwe budowanie adresów URL na podstawie podawanych argumentów.

Url\_for przyjmuje jako pierwszy argument **nazwę funkcji**, do której chcemy się odnieść oraz kolejne argumenty jako **wartości parametrów funkcji**.

Następnie metoda ta zwraca utworzony adres.

**Przykłady:**

url\_for("user.user", name='Admin') zwraca **“/user/Admin”**

url\_for(“index.index”) zwraca **“/”**

**A co z hard-typingiem?**

Oczywiście zamiast korzystać z metody url\_for, moglibyśmy ręcznie zapisywać adresy i pisać coś w postaci: redirect(“/Admin”) jednak byłoby to kompletnie niepraktyczne i generowałoby niepotrzebny effort.

**Szablony HTML**

Jak zapewne się domyślasz, rzadko kiedy będziemy chcieli generować pojedynczy niesformatowany tekst. Chcąc upiększać Naszą aplikacją będziemy dążyli do tego, by używać szablonów HTML i CSS.

Tak więc ulepszmy dotychczas stworzoną aplikację i dodajmy do niej szablon HTML, za pomocą którego będziemy **generować sformatowany napis**.

W tym celu musimy:

1. W katalogu **templates** utworzyć plik o rozszerzeniu **.html**, który będzie odpowiadał za rozmieszczenie elementów na dowolnej stronie.   
   Na rzecz przykładu wrócimy do podstawowej funkcjonalności aplikacji, kiedy to mieliśmy tylko jedną stronę główną wyświetlającą spersonalizowany komunikat. Będzie ona o adresie **localhost:5000/<string:name>**.

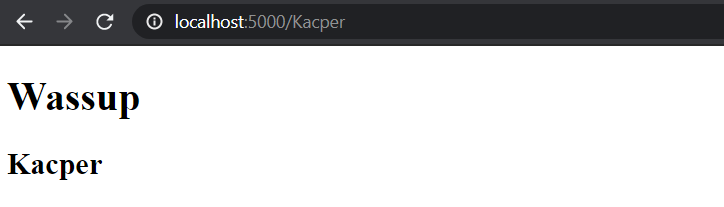
**Plik main.py**

| from flask import Blueprint, render\_template  index\_blueprint = Blueprint('index', \_\_name\_\_)   @index\_blueprint.route('/<string:name>') def index(name):  return render\_template("index.html", welcome\_text="Wassup", user\_name=name) |
| --- |

**Plik templates/index.html:**

| <!DOCTYPE html> <**html** lang="en"> <**head**>  <**meta** charset="UTF-8">  <**title**>Home Page!</**title**> </**head**> <**body**>  <**h1**>{{welcome\_text}}</**h1**>  <**h2**>{{user\_name}}</**h2**> </**body**> </**html**> |
| --- |

**Efekt**:



Jak widzisz, cała magia drzemie w wyniku zwracanym przez metodę render\_template("index.html", welcome\_text="Wassup", user\_name=name) . Przekazujemy do niej nazwę pliku, **z którego ma być generowany kod .html**, a w formie kolejnych keyword argumentów **nazwy zmiennych wykorzystywanych** w kodzie HTML.

Zwróć również uwagę, jak odwoływać się do przekazanych wartości.

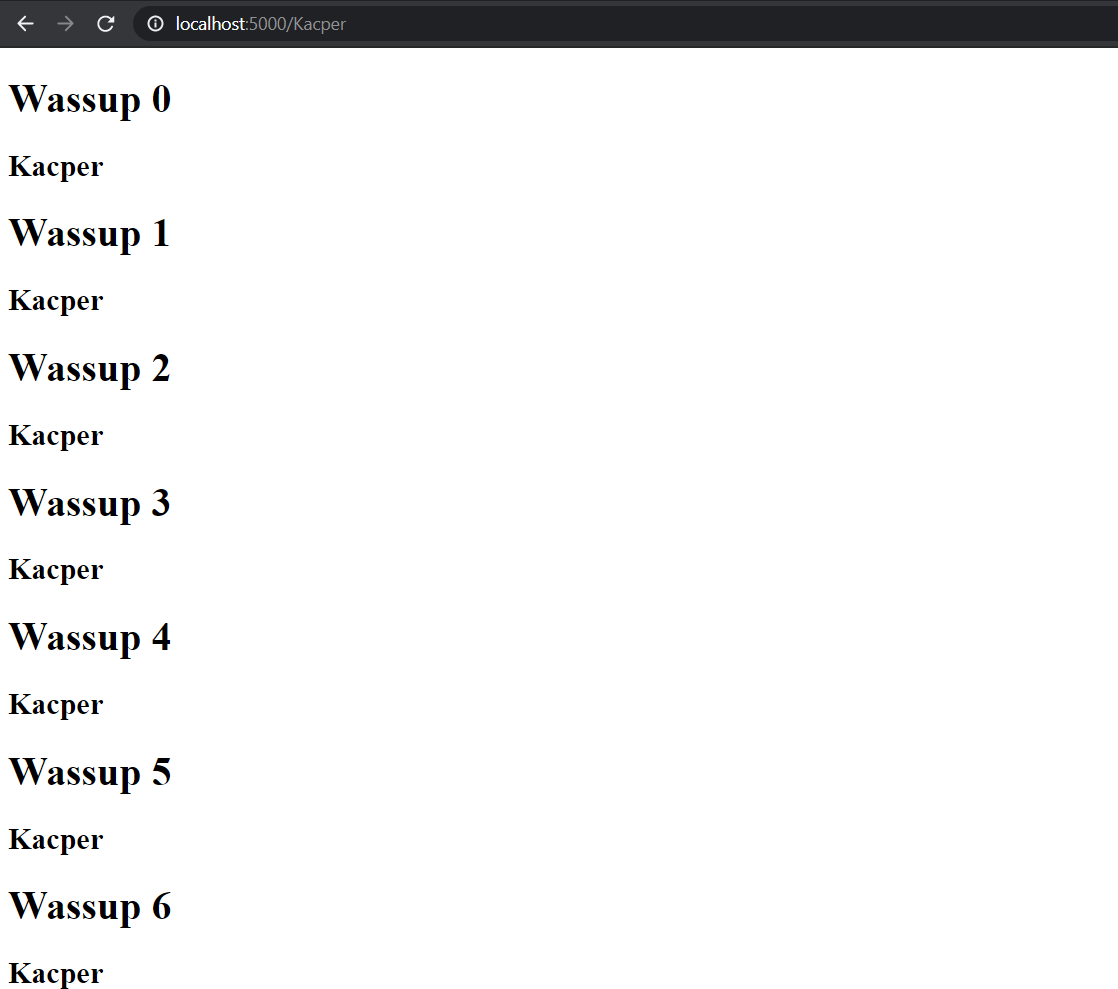
W pliku .html. Wykorzystujemy zapis **{{nazwa\_zmiennej}}**. Podwójne nawiasy sześcienne umożliwiają poprawne interpretowanie kodu przez silnik Flask’a.   
Konieczne będzie ich wykorzystywanie za każdym razem, gdy będziemy chcieli odwoływać się do **zmiennych przekazanych z części logicznej** programu.

Powiem Ci więcej, silnik Flaska pozwala na interpretowanie bardziej złożonych poleceń z poziomu plików HTML.   
Wiesz, że możemy dodawać do składni HTML takie twory jak choćby **pętle**, **instrukcje warunkowe**?

**Plik index.html**

| <!DOCTYPE html> <html lang="en"> <head>  <meta charset="UTF-8">  <title>Home Page!</title> </head> <body>  {% for i in range(10) %}  <h1>{{welcome\_text}} {{i}}</h1>  <h2>{{user\_name}}</h2>  {% endfor %} </body> </html> |
| --- |

**Efekt:**



**Dodanie pliku CSS, dziedziczenie szablonów**

Kolejnym zagadnieniem, który powinniśmy poruszyć po zapoznaniu się z możliwością generowania kodu HTML z poziomu Flaska, jest **załączanie plików CSS** oraz **dziedziczenie szablonów**.

Jeżeli kodowałeś już wcześniej front-end swoich stron internetowych, to zapewne domyślasz się, w jaki sposób dołączyć swój plik CSS-owy.

Wystarczy dodać do katalogu **static** plik o rozszerzeniu **.css**, który **formatować będzie dodawane do strony elementy** z poziomu HTML-a.   
Następnie należy załączyć go do pliku .html **znacznikiem <link>**.

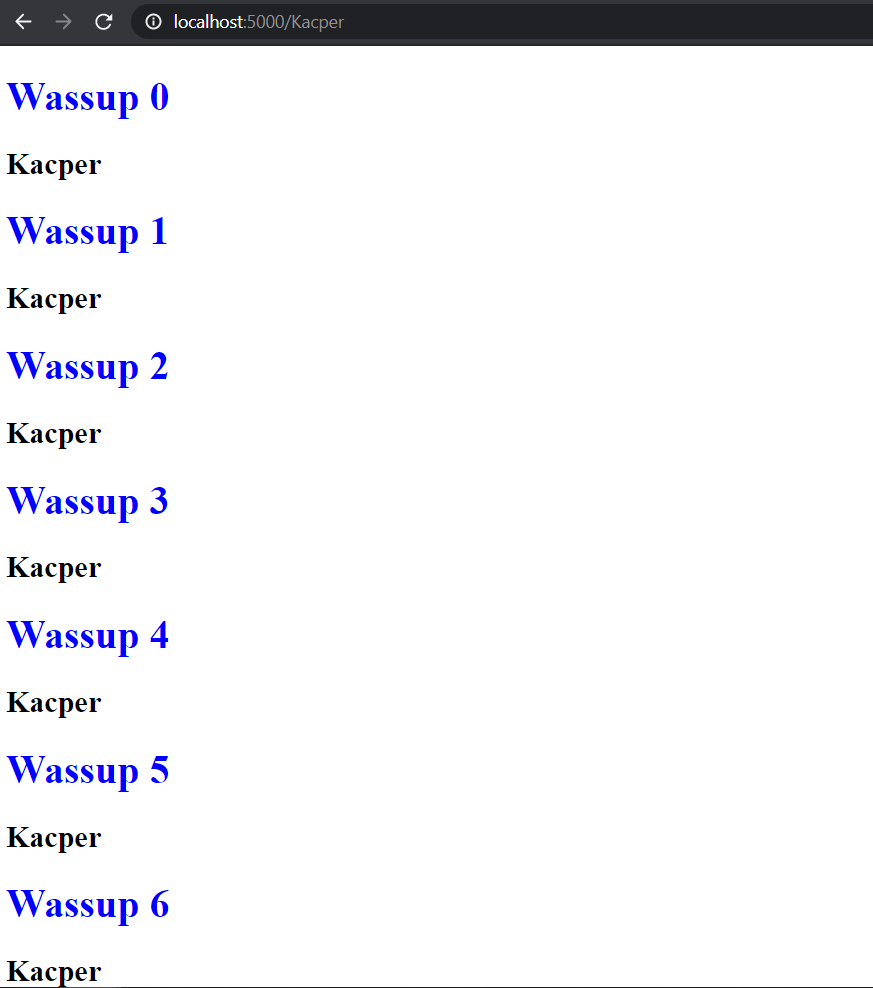
Przykład: *Utworzenie i dołączenie do pliku .html, odpowiedniego pliku .css*

1. Utwórz plik .css w folderze static.
2. Dodaj do pliku .html, do sekcji <head> linię:  
   <link rel="stylesheet" type="text/css" href="static/index.css" />

| <!DOCTYPE html> <**html** lang="en"> <**head**>  <**meta** charset="UTF-8">  <**link** rel="stylesheet" type="text/css" href="static/index.css" />  <**title**>Home Page!</**title**> </**head**> <**body**>  {% for i in range(10) %}  <**h1**>{{welcome\_text}} {{i}}</**h1**>  <**h2**>{{user\_name}}</**h2**>  {% endfor %} </**body**> </**html**> |
| --- |

1. Z poziomu pliku .css sformatujemy każdy nagłowek <h1>, aby wyświetlał się w kolorze niebieskim.

| h1 {  color: blue; } |
| --- |



**Dziedziczenie szablonów**

Aby maksymalnie podnieść efektywność Naszej pracy, warto zapoznać się z możliwością wprowadzenia, tzw. **dziedziczenia szablonów**.

Dzięki takiemu podejściu, możemy tworzyć **szablon bazowy**, jasno definiujący wygląd strony, a następnie dziedziczyć po nim z poziomu kolejnych podstron.

Dzięki temu, ustalimy **jeden wspólny szablon dla każdej podstrony**, a Nasz serwis nabierze jednolitości i usystematyzowania.

Dziedziczenie po danym szablonie - rodzicu - jest bardzo podobne do procesu, jaki zachodzi w programowaniu obiektowym. Otóż tworzymy plik **base.html**, w którym zdefiniujemy cechy wspólne dla każdego pliku dziedziczącego wraz z możliwością ich ewentualnego modyfikowania.

Na przykład: na każdej podstronie dziedziczyć będziemy to samo menu pozycyjne, design strony, a obszar do content’u wraz tytułem również będzie dziedziczony, ale z możliwością modyfikowania przez dzieci.

Przykład: *Utworzenie pliku bazowego dla podstrony index.html*

1. Do katalogu templates dodajemy plik base.html, który będzie rodzicem dla pozostałych podstron. Zauważ, w jaki sposób definiujemy bloki, których zawartość będziemy chcieli zmieniać z poziomu dzieci (**blok title oraz blok content**).

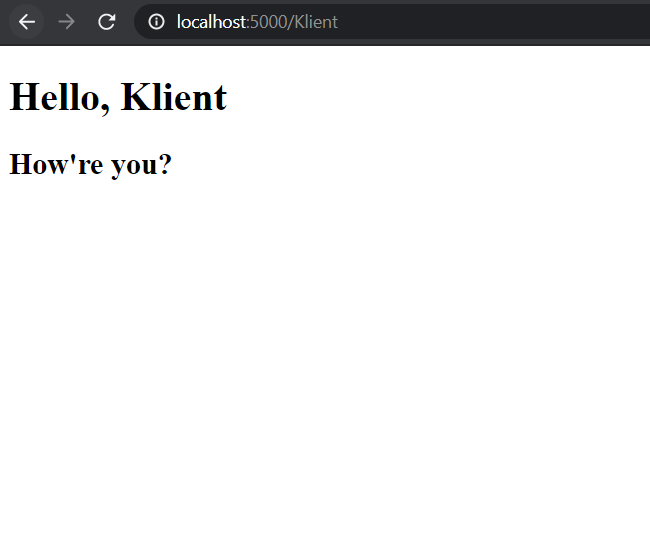
| <!DOCTYPE html> <**html** lang="en"> <**head**>  <**meta** charset="UTF-8">  <**title**>{% block title %}{% endblock %}</**title**> </**head**> <**body**>  <**h1**>Hello, {{user\_name}}</**h1**>  {% block content %}  {% endblock %} </**body**> </**html**> |
| --- |

Plik base.html

1. Następnie możliwe jest zdefiniowanie hierarchii dziedziczenia w pozostałych plikach .html, które mają posiadać cechy base.html.   
     
   Zgodnie z tym zmodyfikujmy plik index.html.

| {% extends "base.html" %} {% block title %}Home{% endblock %}  {% block content %} <**h2**>How’re you?</**h2**> {% endblock %} |
| --- |

Plik index.html



Uruchomiona aplikacja

**Wyjaśnienie:**

Kluczowe linie w powyższym przykładzie to na pewno:

{% extends "base.html" %} określa dziedziczenie po konkretnej klasie

Modyfikowanie poszczególnych odziedziczonych cech:

| {% block title %}Home{% endblock %} |
| --- |

| {% block content %} <**h2**>How're you?</**h2**> {% endblock %} |
| --- |

**Bootstrap i upiększanie strony**

Na zakończenie jeszcze, chciałbym przedstawić Ci łatwy sposób na definiowanie przejrzystego i dobrze wyglądającego layoutu stron.

Jako front-endowi developerzy, mamy możliwość korzystania z wielu bibliotek CSS-owych, w tym popularnego **Bootstrapa**.   
  
Bootstrap to niejako gotowy szkielet do budowania wizualnej części stron internetowych. Autorami Bootstrapa są programiści Twittera i jest on wydawany na licencji MIT. Oznacza to tyle, że można go wykorzystywać bezpłatnie także w komercyjnych projektach.

Przy wykorzystaniu Bootstrap-a, możemy przede wszystkim łatwo tworzyć responsywne i dobrze wyglądające strony. Przy tym w ogóle się nie trudząc!

Przykład: *Stworzenie prostej aplikacji z trzema podstronami i szkieletem Bootstrap: localhost:5000/welcome/<name>, localhost:5000/goodbye/<name>.*

**Plik \_\_init\_\_.py**

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

**Plik main.py**

| from flask import Blueprint, render\_template  welcome\_blueprint = Blueprint('welcome', \_\_name\_\_) goodbye\_blueprint = Blueprint('goodbye', \_\_name\_\_)   @welcome\_blueprint.route('/welcome/<string:name>') def welcome(name):  return render\_template("welcome.html", additional\_text="Wassup", user\_name=name)   @goodbye\_blueprint.route('/bye/<string:name>') def goodbye(name):  return render\_template("goodbye.html", user\_name=name) |
| --- |

**Plik goodbye.html**

| {% extends "base.html" %} {% block title %} Goodbye {% endblock %}  {% block content %} <**h2**>Take care,</**h2**> {% endblock %} |
| --- |

**Plik welcome.html**

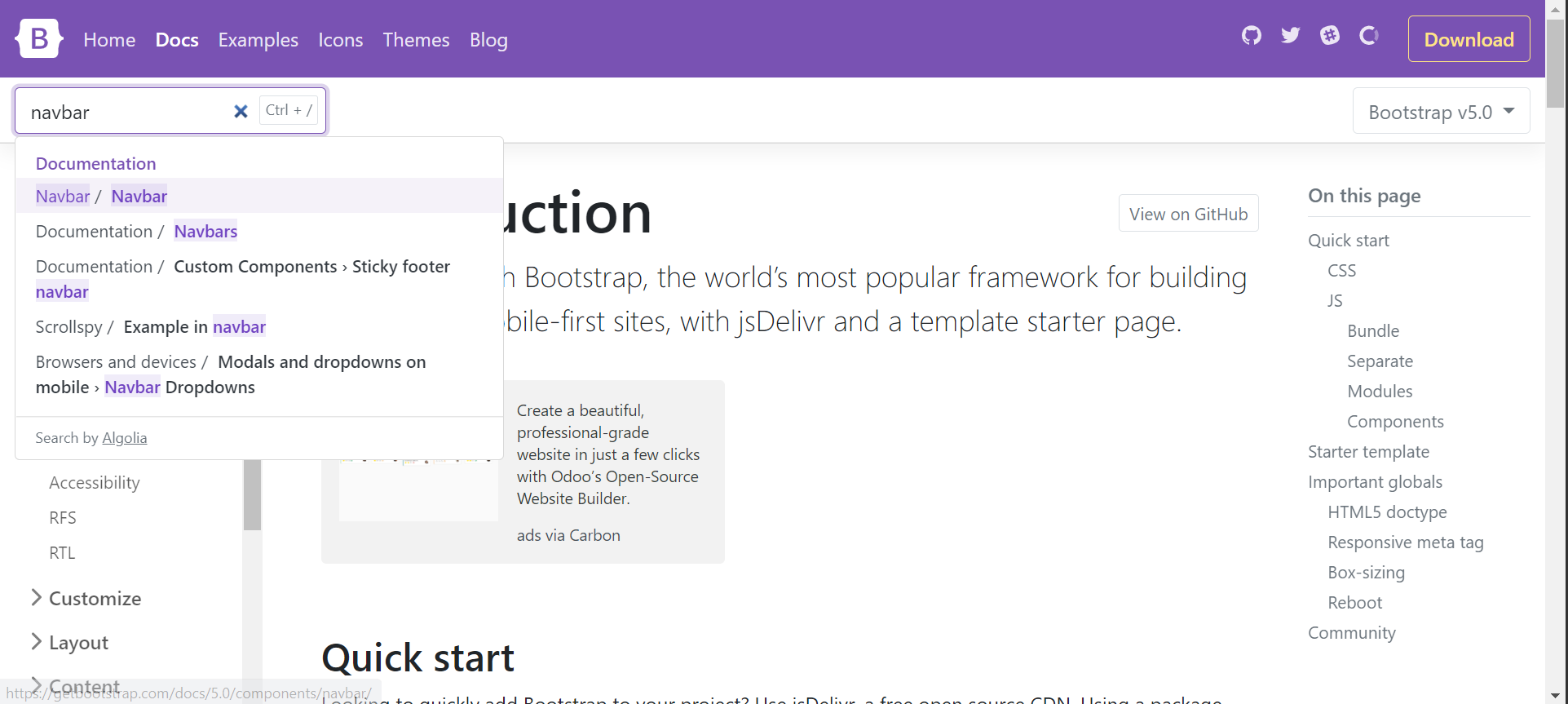
| {% extends "base.html" %} {% block title %} Welcome {% endblock %}  {% block content %} <**h2**>Hello, </**h2**> {% endblock %} |
| --- |

**Plik base.html** z wykorzystaniem Bootstrap’a.

| <!DOCTYPE html> <**html** lang="en"> <**head**>  <**meta** charset="UTF-8">  <**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="/welcome/Default">Welcome</**a**>  </**li**>  <**li** class="nav-item">  <**a** class="nav-link" href="/bye/Default">Goodbye</**a**>  </**li**>  </**ul**>  </**div**>  </**div**>  </**nav**>  {% block content %}  {% endblock %}  <**h1**>{{user\_name}}</**h1**> </**body**> </**html**> |
| --- |

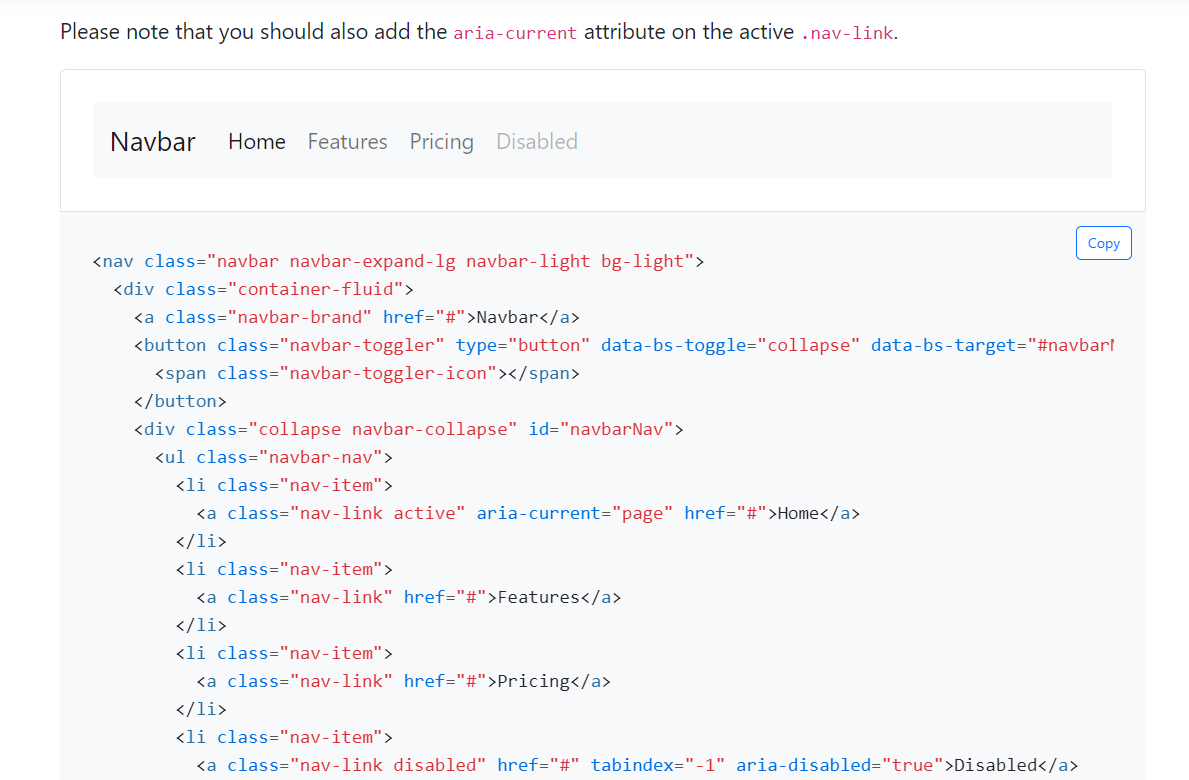
**Wyjaśnienie**:

Lwią część kodu z pliku base.html pobraliśmy z oficjalnej dokumentacji Bootstrapa spod zakładki Navbar znajdującej się w Docs.

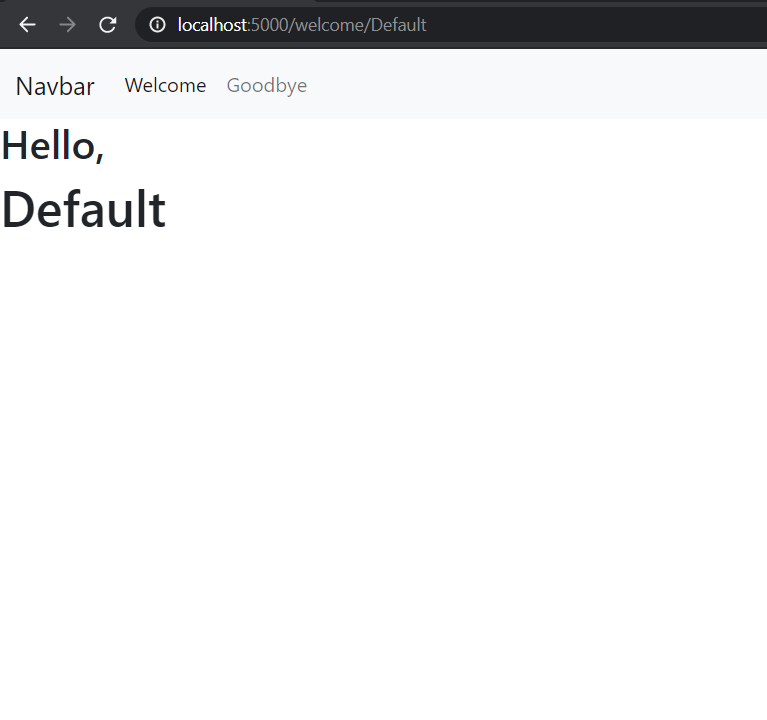


Dokumentacja Bootstrap

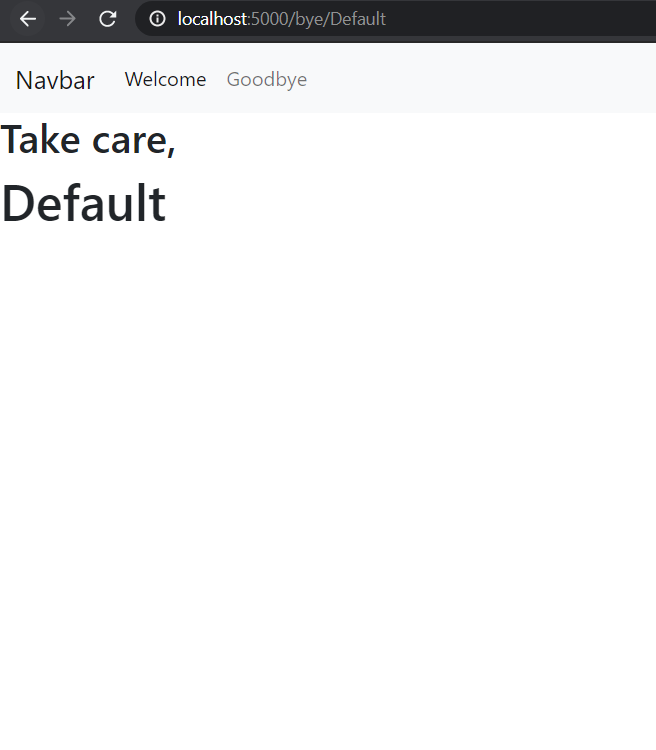
Jedyne zmiany, jakich dokonaliśmy, to lekkie dostosowanie szablonu pod Nasze preferencje, czyli ograniczenie zakładek w menu nawigacyjnym i zmiana parametrów znacznika <a>, aby możliwe było przekierowywania do danych podstron.



**Efekt:**



Zakładka Welcome



Zakładka Goodbye