**1 Szkolenie z Flaska**

**Utworzenie pierwszego projektu**

**O Flasku słów kilka**

Flask jest na tyle specyficznym frameworkiem, że często nazywa się go prima-aprillisowym żartem. I to wcale nie dlatego, że jest nieużyteczny i niepraktyczny. Flask bowiem umożliwia tworzenie całej funkcjonalności serwisu w jednym pliku. Nie ma on określonego szkieletu, a liczne biblioteki i rozszerzenia umożliwiają łatwe korzystanie z takich komponentów jak ORM (sposób odwzorowania systemu informatycznego na bazę danych) czy panelu admina.

Framework ten jest również aplikacją wykorzystującą WSGI (Web Server Gateway Interface). Jest to interfejs, który umożliwia komunikację między serwerami a web-wymi aplikacjami napisanymi w Pythonie.

Flask-ową biblioteką wykorzystującą WSGI jest Werkzeug, a jinja2 jest silnikiem szablonowym (umożliwiającym generowanie kodu HTML i CSS).

Efektywność i przyjazność Flaska dla użytkownika wynika z faktu, iż wraz z nim dostarczane są różne narzędzia i biblioteki ułatwiające pracę developera (np. łatwa integracja z bazą danych). Ponadto programiści tworzący własne web-owe rozwiązania nie są zmuszeni do zaciągania żadnych zewnętrznych dependencji ani trzymać się ścisłej struktury projektu. Dlatego też Flask jest idealnym do wykorzystania rozwiązaniem, gdy chcesz szybko budować własne API, elastyczne i proste aplikacje web-owe, mieć pełną kontrolę nad tworzeniem i obsługą podłączonej bazy danych.

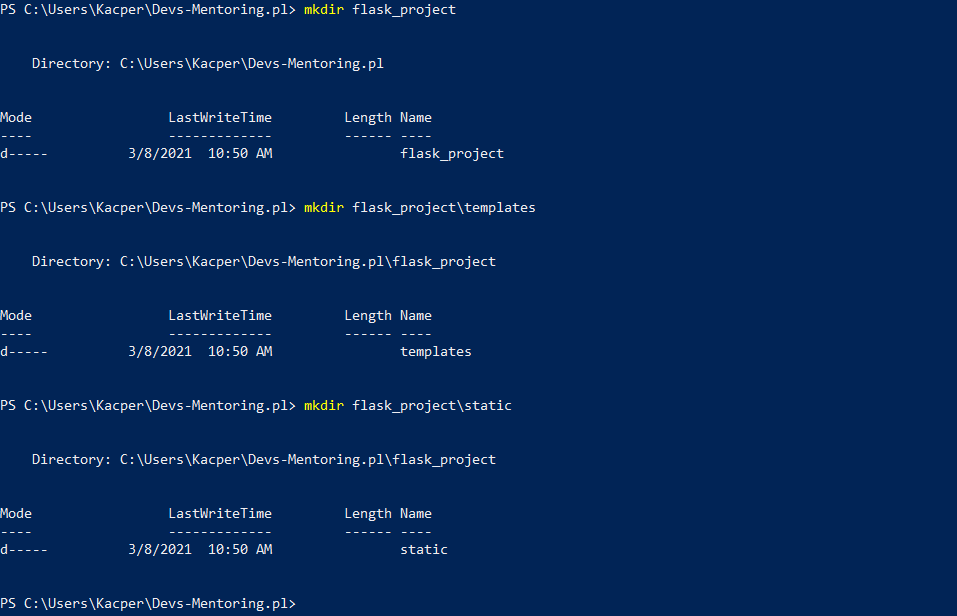
Flask bardzo dobrze sprawdza się w momencie, gdy tworzymy rozwiązanie podzielone na mniejsze serwisy. Wówczas każda pojedyncza projektu część nie będzie wymagała dużej ilości kodu do zaimplementowania. Proces maintanance’u i wdrożenia będzie prosty i skuteczny.

**Pierwszy Projekt**

Pierwszą aplikacją web-ową, która stworzymy z wykorzystaniem Flaska będzie projekt Hello World.

1. **Utworzenie projektu.**

Projekt utworzę w Powershell-owym CLI. Aczkolwiek, jeżeli korzystasz z dowolnego IDE (np. PyCharm), przez który łatwo tworzyć środowisko wirtualne i doinstalowywać do niego paczki - śmiało, utwórz projekt i zainstaluj zależności na własną rękę : )



Utworzenie projektu wraz z folderami templates i static



Drzewo katalogu

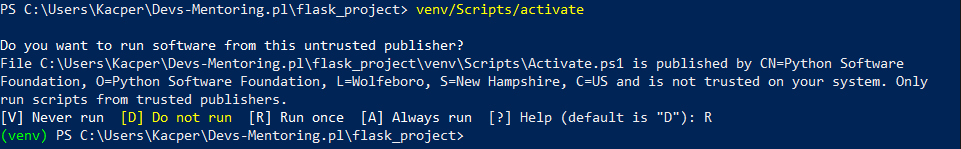
Folder **templates** znajdujący się w katalogu projektu flask\_project posłuży Nam do umieszczania **HTML-owych szablonów**.   
Katalog **static** przyda się natomiast do przechowywania wszystkich plików **statycznych projektu**, tj. zdjęcia, kod CSS, pliki .js niezbędne do prawidłowego działania tworzonej aplikacji.

1. **Utworzenie środowiska wirtualnego i instalacja Flask-a**

Aby zachować hermetyzację wszystkich doinstalowywanych zależności, ręcznie venv-a utworzymy do powstającego projektu flask\_project.

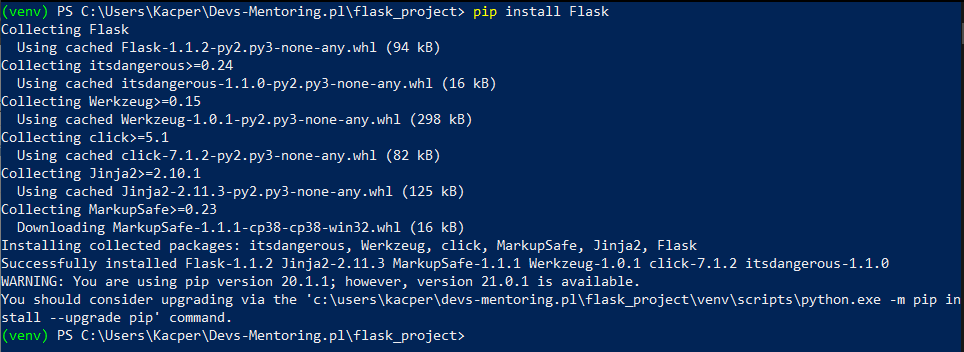


Dodanie środowiska wirtualnego



Aktywacja środowiska wirtualnego

Mając już teraz aktywowane środowisko, możemy zainstalować Flask-a.



1. **Utworzenie pliku głównego aplikacji**

W tym szkoleniu pokażę Ci dwa podejścia do tworzenia Flask-owej aplikacji. Pierwsze będzie dość podstawowe - sprawdzające się przy tworzeniu małych aplikacji.   
Drugie natomiast zaprezentuje Ci, jak powinno się prawidłowo i czytelnie rozmieszczać pliki pod stworzenie bardziej rozbudowanych projektów.

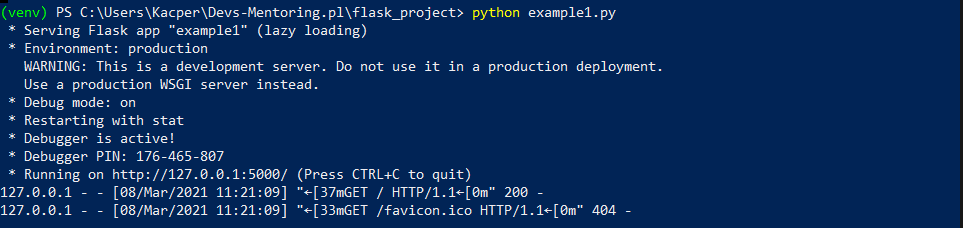
**Sposób I Prosta aplikacja Hello World:**

Po utworzeniu powyższej struktury projektu, możesz już z poziomu własnego edytora tekstu przejść do katalogu głównego projektu i utworzyć pierwszy główny plik aplikacji, którego nazwiemy **example1.py**.

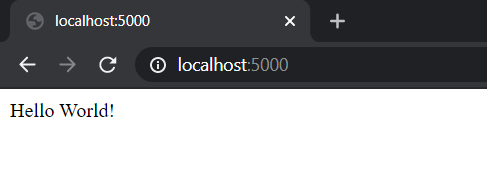
To będzie przykład Naszej pierwszej aplikacji Flask-owej generującej napis Hello World w przeglądarce internetowej!

| from flask import Flask  APP = Flask(\_\_name\_\_)   @APP.route('/') def index():  return "Hello World!"  if \_\_name\_\_ == "\_\_main\_\_":  APP.debug = True  APP.run() |
| --- |

Uruchomienie:



Efekt:



Jeżeli po wpisaniu w pasek wyszukiwania przeglądarki adresu: localhost:5000 lub 127.0.0.1:5000 otrzymałeś taki wynik jak powyżej, to moje gratulacje! Właśnie stworzyłeś własny działający projekt!

Jednak niewątpliwie musimy nieco objaśnić wyżej napisany kod.

Aby zapewnić poprawne działanie Naszego Flask-owego projektu, w pierwszej kolejności konieczne jest utworzenie obiektu reprezentującego aplikację.

Z tego powodu konieczny był zapis:

| APP = Flask(\_\_name\_\_) |
| --- |

Następnie Nasza aplikacja musi wiedzieć, jak powinna się zachować, gdy użytkownik wejdzie na określony **URI (URL) aplikacji**.

Zachowanie definiujemy przez dowolne funkcje i umieszczenie nad nimi dekoratora

| @APP.route(URI) |
| --- |

W Naszym przypadku chcieliśmy, aby napis Hello World był wyświetlany po wejściu do tzw. ścieżki głównej wyrażonej przez forward-slash: /

Jeżeli natomiast, chciałbyś, aby napis Hello World! generowany i wyświetlany był po przejściu na adres: localhost:5000/home, to wówczas dekorator musiałby przyjąć postać:

| @APP.route('/home') |
| --- |

Zwróć również uwagę na tę część kodu:

| if \_\_name\_\_ == "\_\_main\_\_":  APP.debug = True  APP.run() |
| --- |

Objaśnienie:

* APP.debug = Truezapewnia automatyczne odświeżanie aplikacji po wprowadzonych w projekcie zmianach.

Czyli w momencie, gdybyśmy napis Hello World! chcieli zastąpić pozycją Hello Devs-Mentoring.pl!, wystarczyłoby dokonać modyfikacji w kodzie i po prostu zapisać zmiany.   
Flask po ich wykryciu, oszczędziłby Nam czasu oraz zachodu i automatycznie zrestartowałby uruchomioną aplikację. Nie byłoby wówczas konieczne jej ręczne zamykanie i uruchamianie na nowo.

* APP.run()natomiast to linia uruchamiająca aplikację. Nic dodać nic ująć.

Uważam, że warto jeszcze co nieco wspomnieć o uruchomieniu Naszej aplikacji.

Zauważ, że aby tego dokonać wystarczyło po prostu uruchomić plik z kodem źródłowym (example1.py).

Następnie, aby móc obserwować lokalne działanie aplikacji, konieczne było wpisanie w przeglądarce adresu w postaci: **localhost:5000**.

To, co musisz bowiem wiedzieć, to fakt, iż aplikacja uruchamiana jest na Twojej lokalnej maszynie (wyrażonej przez **localhost** lub równoważnie przez **127.0.0.1**) na **porcie 5000**. Dopóki nie dokonasz jej deploymentu na zewnętrznym serwerze, nikt poza Tobą nie ma wglądu do tego, co tworzysz.

**Ciekawostka:**

Port jest logicznym adresem każdej aplikacji umożliwiającym jej komunikację z sieciowymi zasobami.

**Sposób II Utworzenie Hello World w innej strukturze projektu**

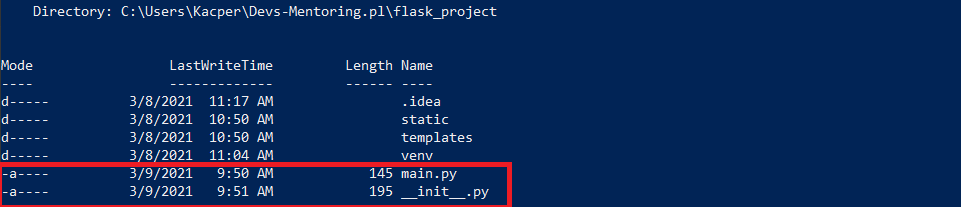
A teraz chciałbym w formie małej dygresji zaprezentować Ci, jak można utworzyć tę samą aplikację, ale z wykorzystaniem innej struktury projektu.

Struktury **bardziej zaawansowanej**, którą będziesz mógł bez problemu wykorzystywać przy większych projektach.

Idea bowiem, jaką wprowadzimy będzie polegała na rozdzieleniu aplikacji na **kilka mniejszych plików w taki sposób**, aby jej ewentualna późniejsza rozbudowa była jak najłatwiejsza.

W nowym podejściu chodzi bowiem o to, żeby stworzyć plik **\_\_init\_\_.py**, który będziemy zawsze uruchamiali jako pierwszy i będzie on zaciągał i uruchamiał wszystkie zależności i funkcjonalności z aplikacji.

Nowa struktura katalogu:



Tak jak wspomniałem, do katalogu dodaliśmy nowy plik **\_\_init\_\_.py**, którego zawartość za chwilę zobaczysz. Funkcjonalność pliku example1.py przenieśliśmy natomiast do **main.py**.

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

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

**Plik main.py:**

| from flask import Blueprint  index\_blueprint = Blueprint('index', \_\_name\_\_)   @index\_blueprint.route('/') def index():  return "Hello World!" |
| --- |

Zwróć uwagę na nowo wykorzystany obiekt - **Blueprint**. Blueprinty ułatwiają Nam zarządzanie aplikacją i jasne wydzielanie projektu na różne moduły.

Stosowanie przedstawionej wyżej organizacji plików we Flask-owym projekcie wprowadza do aplikacji jej łatwą rozszerzalność oraz ewentualną reorganizację.

Z drugiej strony powiesz - ale musimy przecież pisać więcej kodu! Zgodzę się, ale tak wydzieloną strukturę projektu docenisz dopiero w momencie, gdy będziesz tworzył bardziej rozbudowaną aplikację składającą się, np. z modułów uwierzytelniających, zewnętrznych API, modułów odpowiadających za funkcjonalność (sklep, strona główna itd).

**Uruchomienie aplikacji:**

Na koniec musimy się jeszcze pochylić dotyczy sposobu uruchomienia tak wydzielonego projektu:

1. Przejdź do katalogu nadrzędnego dla projektu flask\_project (w Naszym przypadku jest to katalog Devs-Mentoring.pl)
2. Wykonaj jedno z poniższych poleceń:

**Unix Bash (Linux, Mac, etc.):**$ export FLASK\_DEBUG=1

$ export FLASK\_APP=flask\_project

$ flask run

**Windows CMD:**

> set FLASK\_DEBUG=1

> set FLASK\_APP=flask\_project

> flask run

**Windows PowerShell:**

> $env:FLASK\_DEBUG=1

> $env:FLASK\_APP="flask\_project"

> flask run

1. Po wykonaniu powyższych kroków, aplikacja zostanie uruchomiona wraz z włączonym trybem debugowania.

**Zakończenie**

To tyle, jeśli chodzi o podstawowy workflow przy budowaniu Flask-owych aplikacji. Omawiając dalsze tematy, będę wykorzystywał drugi sposób organizacji plików w projekcie. W ten sposób będę chciał jak najefektywniej wdrożyć Cię w pisanie coraz to większych projektów i już od samego początku budować w Tobie jak najlepsze nawyki.

A w następnej sekcji - m.in generowanie szablonów z wykorzystaniem HTML-a i CSS-a i przekazywanie własnych parametrów!