Sprawozdanie\_Piac

Repozytorium projektu dostępne pod linkiem: https://github.com/BartoszP37/projektPiac.git

Bartosz Poręba 81733

**Laboratorium 1**



**Flask**- mikro framework aplikacji webowych napisany w języku Python. Aby go zainstalować używamy komendy :

**pip3 install flask**

i importujemy komendą:

**from flask import Flask**

Aby poinformować flaska pod jakim adresem ma wyświetlać treść używamy:

**@app.route(‘/*ścieżka*’)**

**def *nazwa*():**

**return**

Flask ma możliwość zwracania treści tekstowej bądź też stron statycznych. W folderze „templates” możemy dodać strony html i wywołać je tak jak w poprzednim przykładzie ale z użyciem:

**return render\_template(strona.html)**

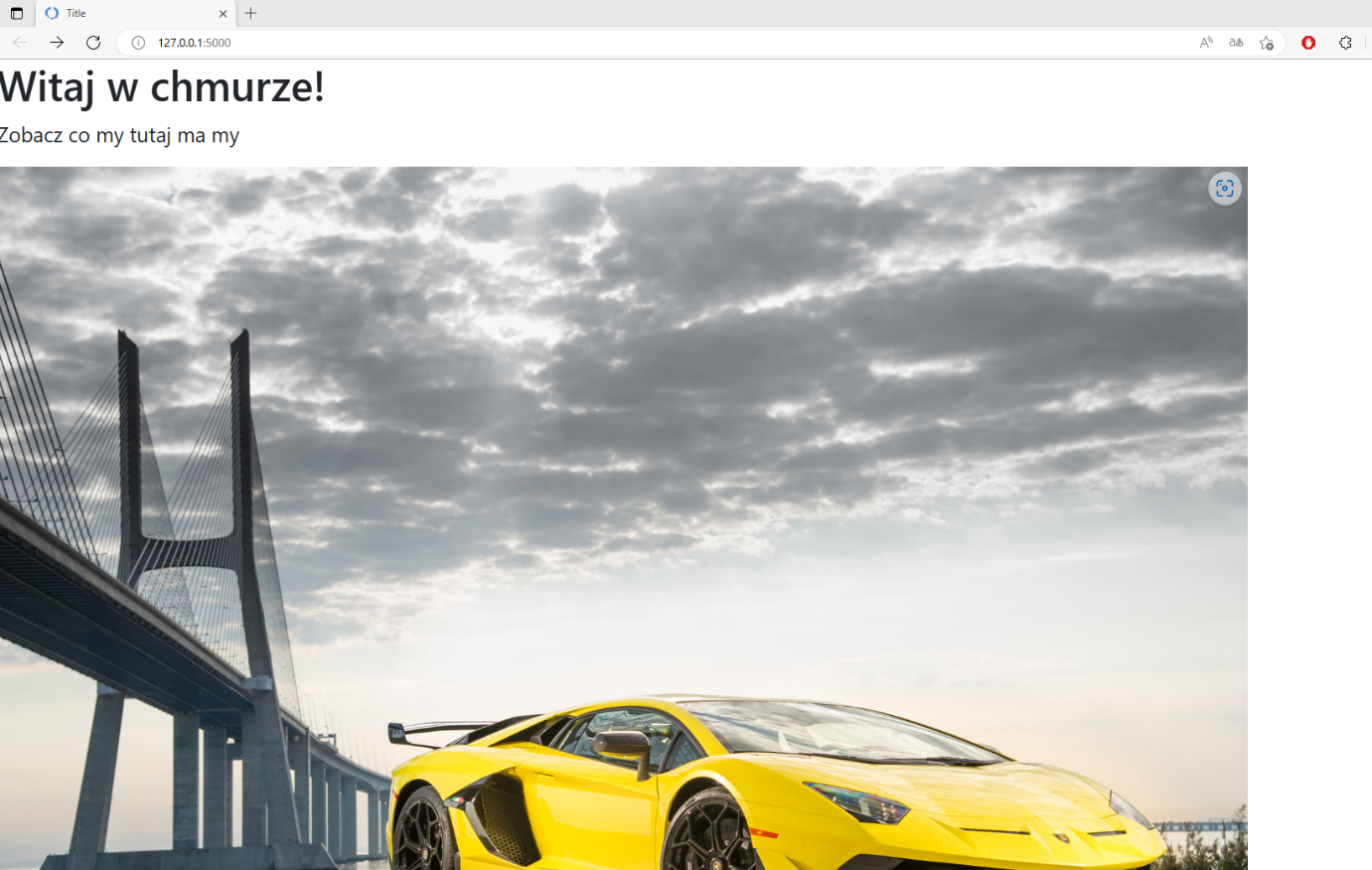
Możemy również stworzyć folder „static” i odwoływać się do statycznych stron znajdujących się w nim kodyfikując return w następujący sposób:

**return app.send\_static\_file(‘nazwa.html’)**

Na końcu aplikacji aby ją uruchamiać musimy zawrzeć:

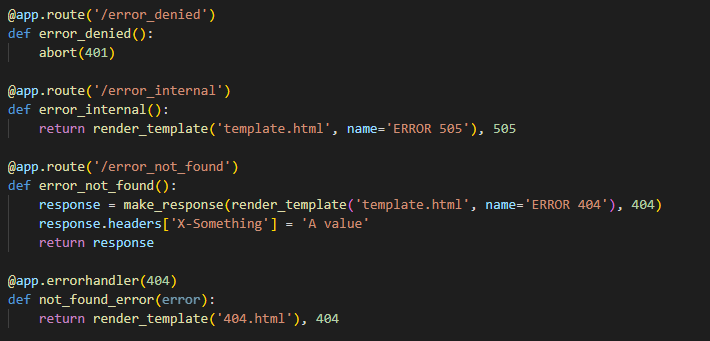
**If\_\_name\_\_==’\_\_main\_\_’:**

**app.run(debug=true, *inne parametry*)**

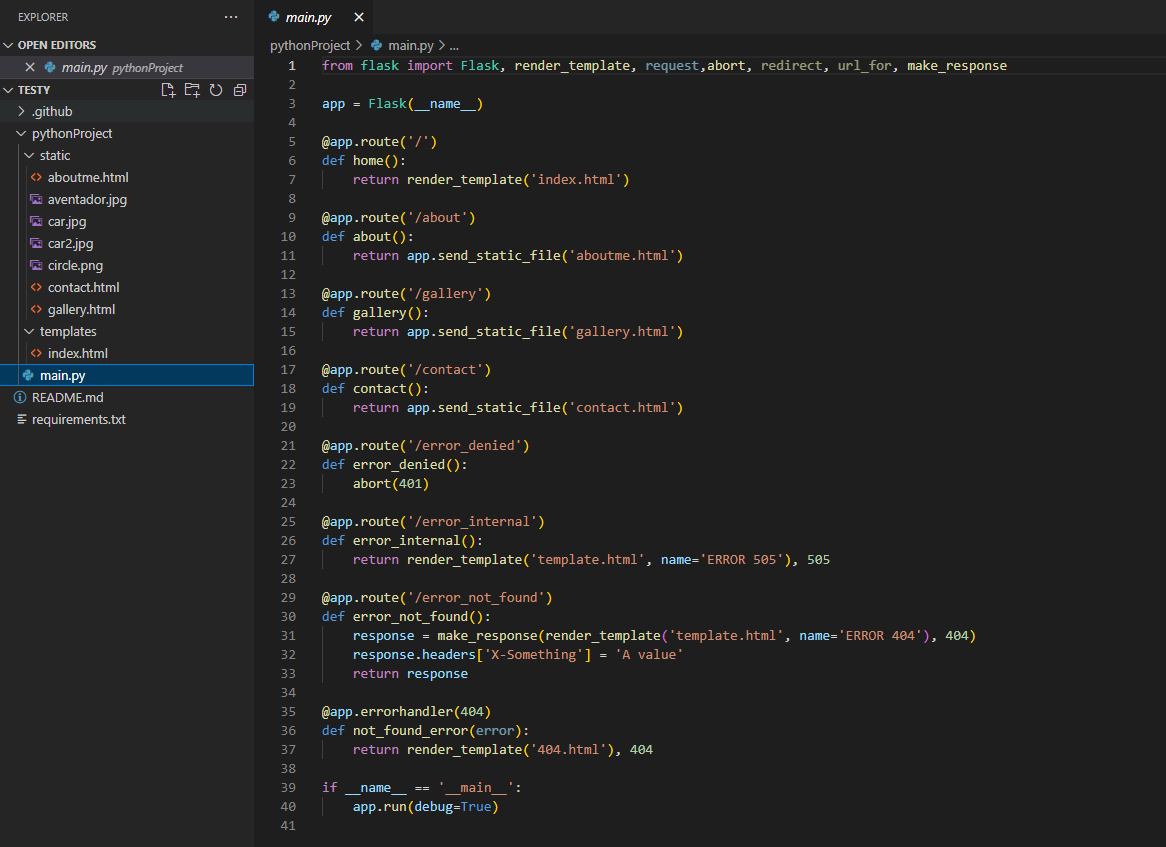
****

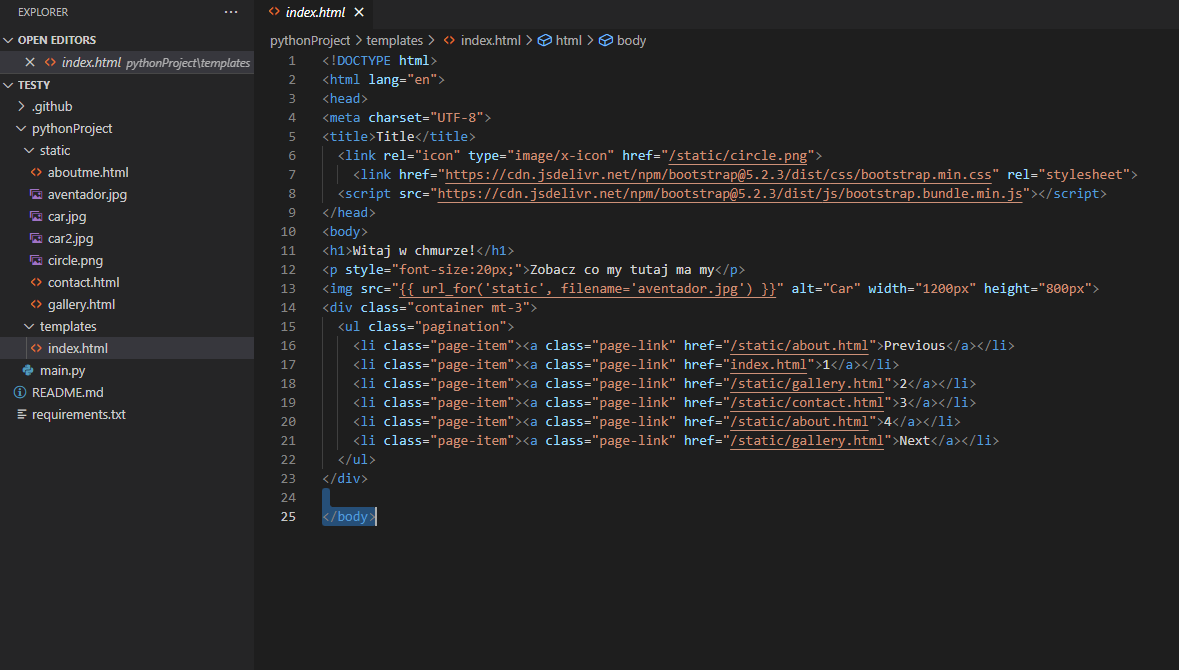
Za pomocą poniższego kodu możemy wyświetlać zdjęcia zamieszczone w folderze statić na stronach.

**<img src="{{ url\_for('static', filename='car.jpg') }}" alt="Car"><br>**

****

Powyższe funkcje takie jak abort() czy errorhandler() zwracają kody błędów wysyłane przez API.

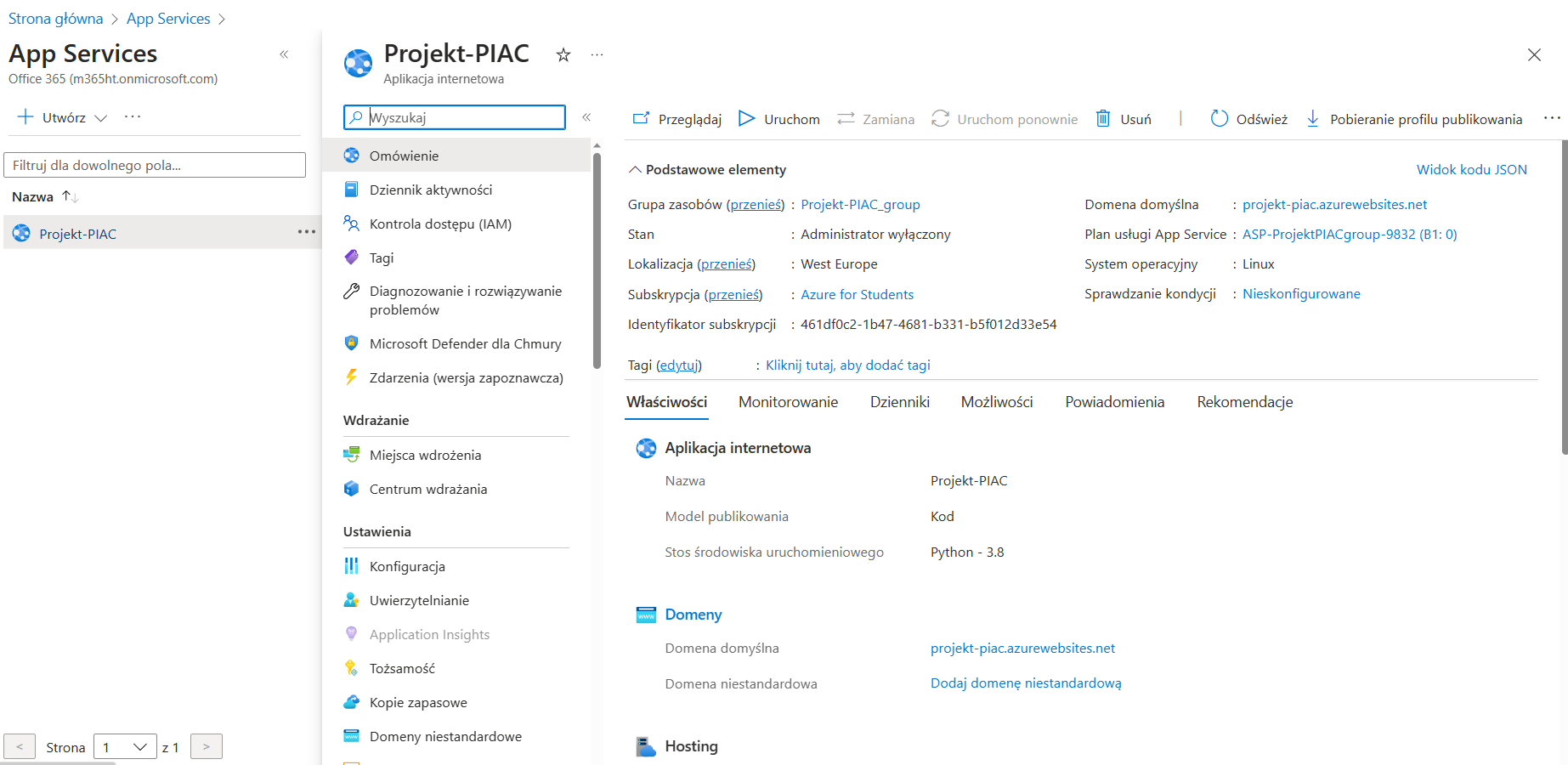


Powyższy kod przedstawia aplikację wykorzystująca flask która zawiera stronę domową renderowaną z szablony oraz podstrony wysyłane jako pliki statyczne z folderu static.

Powyższy kod to przykładowa strona domowa która zawiera przykładową treść. W <head> zawarliśmy favicon oraz dodaliśmy na stronie paginację umożliwiająca przemieszczanie się pomiędzy podstronami.

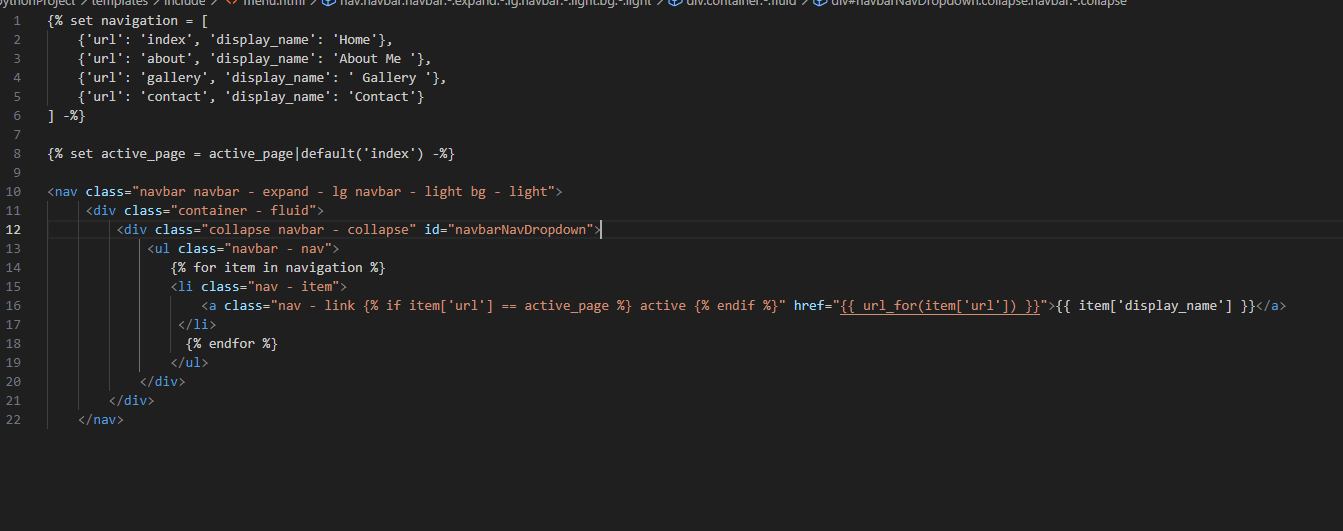
**Laboratorium 2**

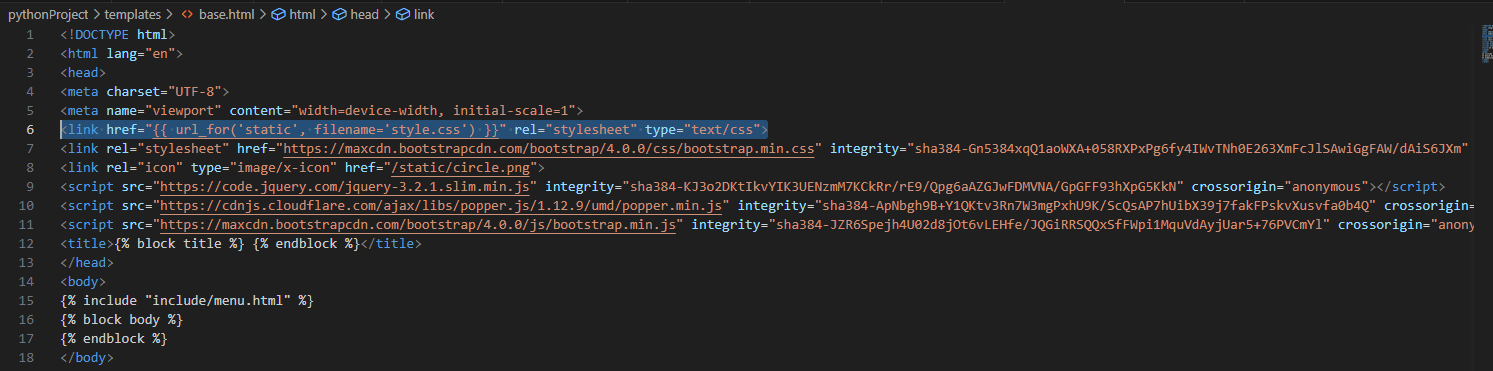
Usługa Azure App Service to usługa chmurowa w modelu PaaS zarządzana przez firmę Microsoft. Bez serwerowa architektura aplikacji w chmurze, to usługa która nie wymaga serwerów, maszyn wirtualnych ani kontenerów do obsługi administracyjnej lub zarządzania.



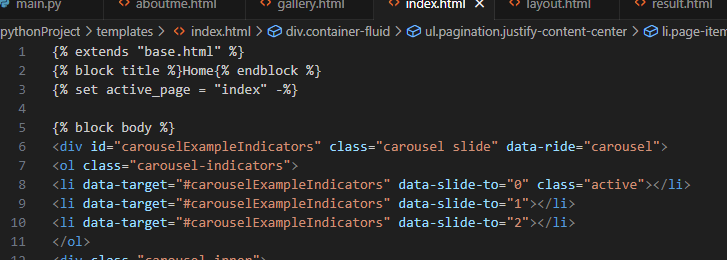
Aby utworzyć aplikację internetową wchodzimy w App Services i wybieramy utwórz. Następnie należy wypełnić wszystkie pola i wybrać interesujące nas opcje. Po pomyślnym stworzeniu aplikacja będzie dostępna pod linkiem https://*nazwa*.azurewebsites.net . Aby wdrożyć projekt z GitHub Actions wchodzimy w zakładkę Centrum wdrażania. Logujemy się do swojego konta GitHub i wybieramy repozytorium oraz gałąź. Jeśli przejdziesz do repozytorium GitHub, zauważysz, że do folderu „.github / workflows” dodano nowy plik. Ostatnim punktem będzie dodanie do projektu pliku requirements.txt i zawarcie w nim formuły „Flask>=1.0,<=1.1.2. Możliwe, że trzeba będzie też wywołać polecenie pip3 install -r requirements.txt. Pod przyciskiem przeglądaj możemy podejrzeć obecny stan naszego projektu.

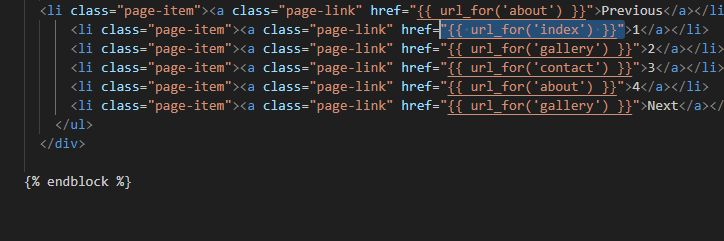
**Laboratorium 3**

Szablon to plik zawierający tekst odpowiedzi ze zmiennymi umieszczonymi w miejscach części dynamicznych, które będą konkretyzowane w kontekście żądania. Proces, który zastępuje zmienne rzeczywistymi wartościami i zwraca końcowy ciąg znaków odpowiedzi, nazywa się rendrowaniem. Dzięki szablonom możemy ułatwić sobie pracę skracając ją o elementy które będziemy chcieli zawrzeć bliźniaczo na każdej stronie.

Możemy na przykład ustawić przełączniki navbaru spisując je w liście navigation i następnie przy użyciu pętli for wyświetlić w liście html.

Następnie możemy to zamieścić w szablonie html używając include i stworzyć blok title oraz body. W bloku body później możemy zamieszczać content strony.



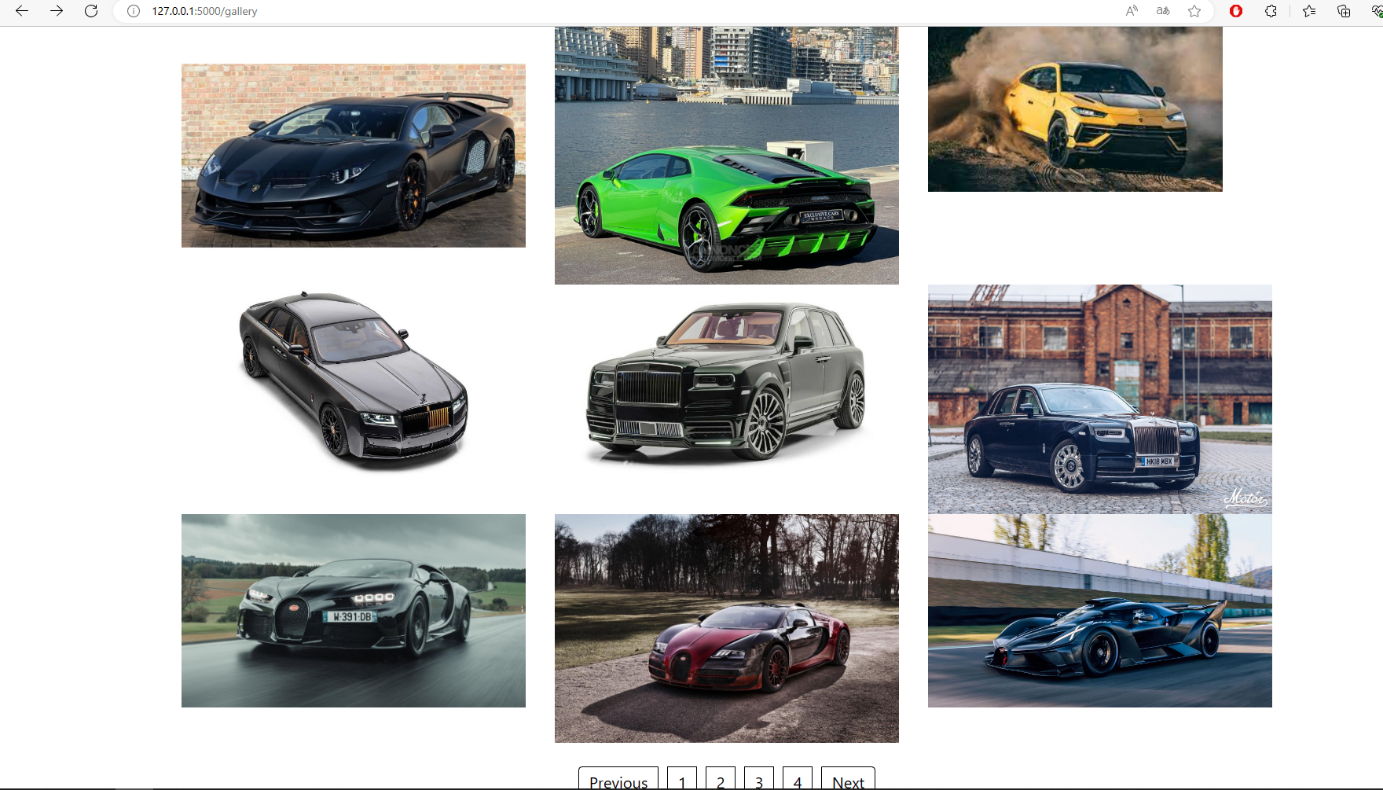
Na stronie html możemy użyć extends aby zaimplementować szablon który stworzyliśmy i dodać blok body w którym zawrzemy treść.

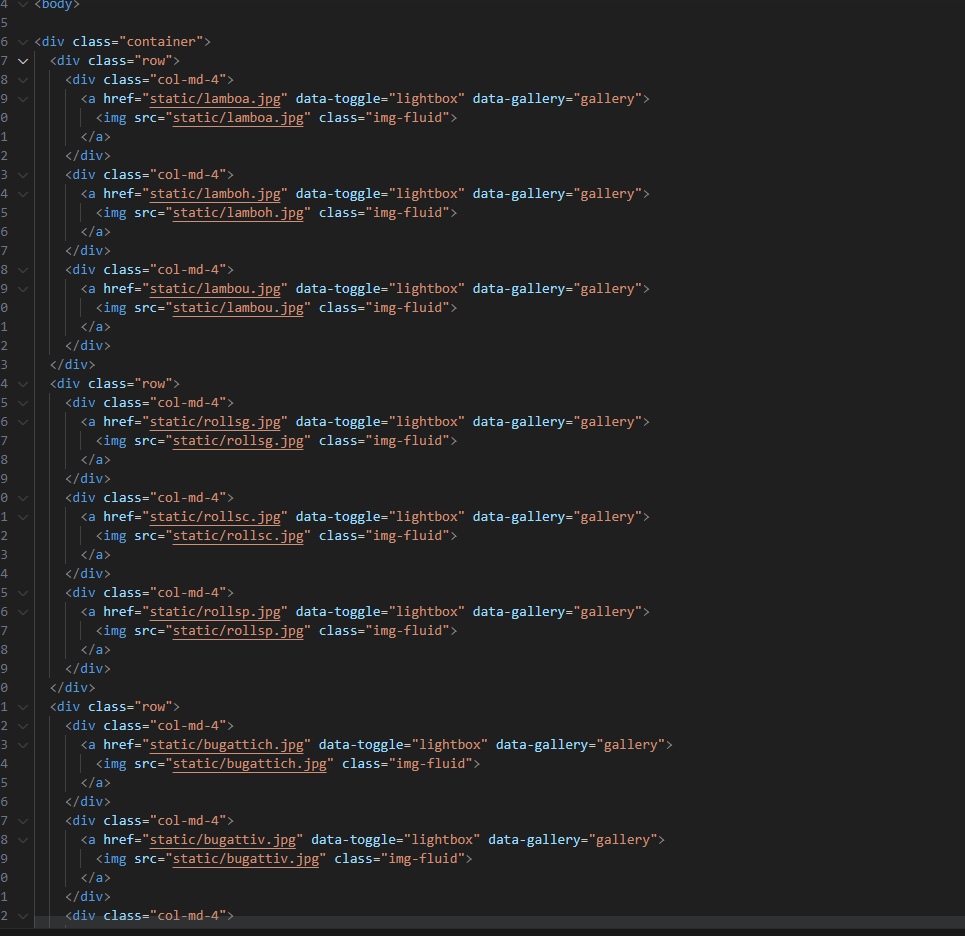
,

Należy pamiętać aby na końcu zamknąć blok.

W flasku mamy również możliwość używania bibliotek bootstrap. Gotowych klas CSS do stylizowania strony. Aby to zrobić najpierw musimy zainstalować bootstrap. Używamy do tego komendy:

**pip3 install flask\_bootstrap**

Następnie musimy podlinkować biblioteki bootstrapowe i już jesteśmy w stanie korzystać w pełni z bootstrapa. Znacznie skróci on nam stylizowanie strony z wykorzystaniem css.



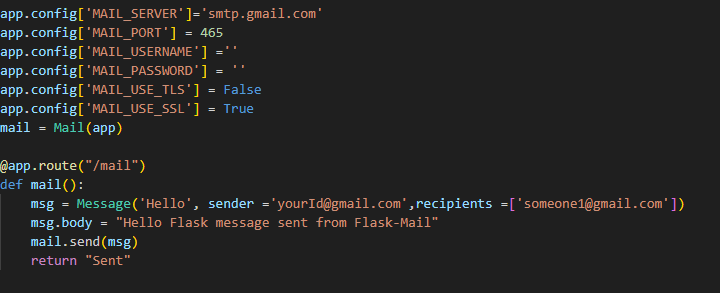
W tym przykładzie użyliśmy bibliotek bootstrap aby stworzyć galerię zdjęć w formacie 3 na 3. Po kliknięciu zdjęcie powiększa się i prezentuje w trybie lightbox.

Dzięki interfejsowi flask mail mamy możliwość wysyłania wiadomości e-mail do użytkowników. Na początek będziemy musieli zainstalować tą funkcjonalność komendą:

**pip3 install flask\_mail**

Następnie importujemy tą funkcjonalność:

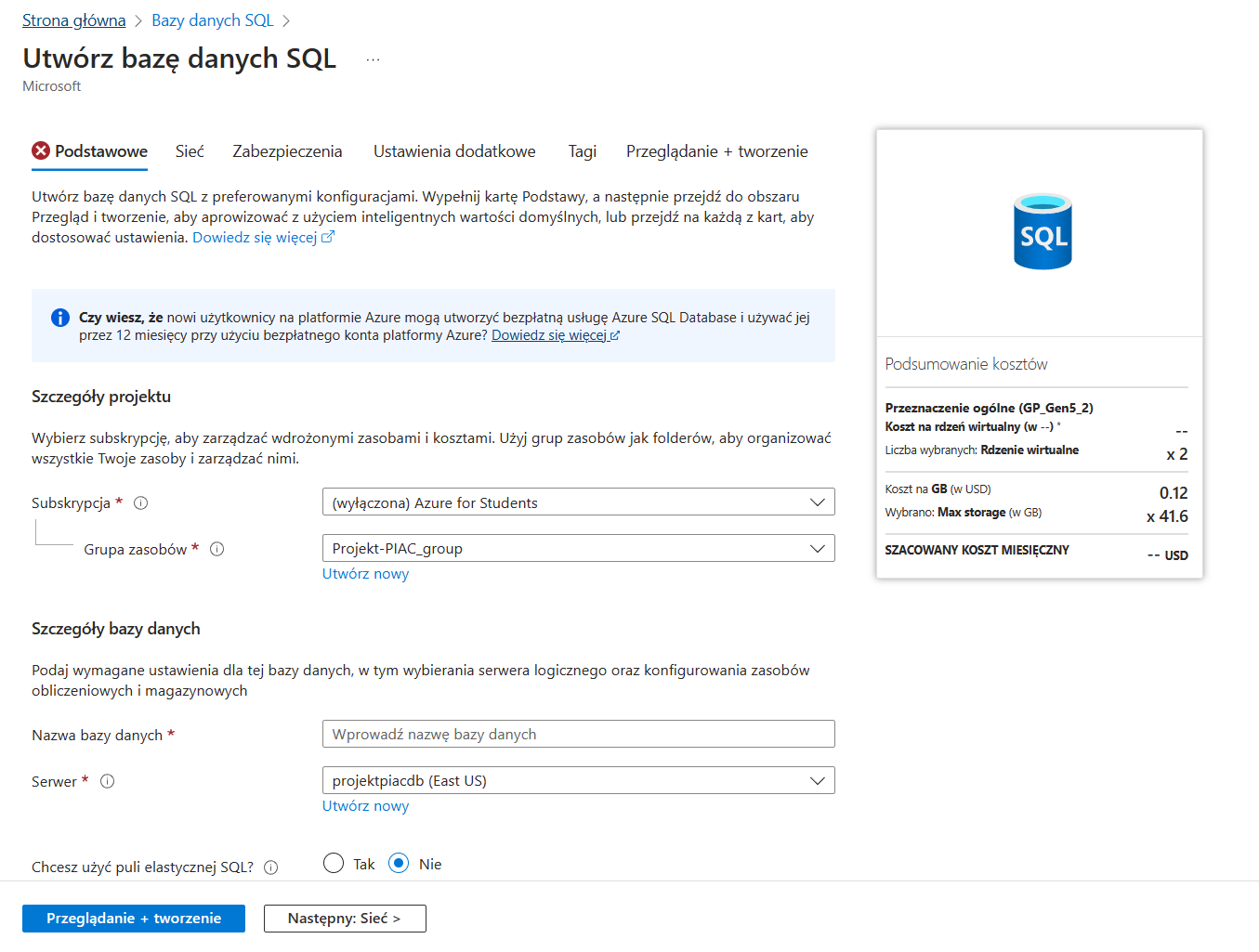
**from flask\_mail import Mail, Message**

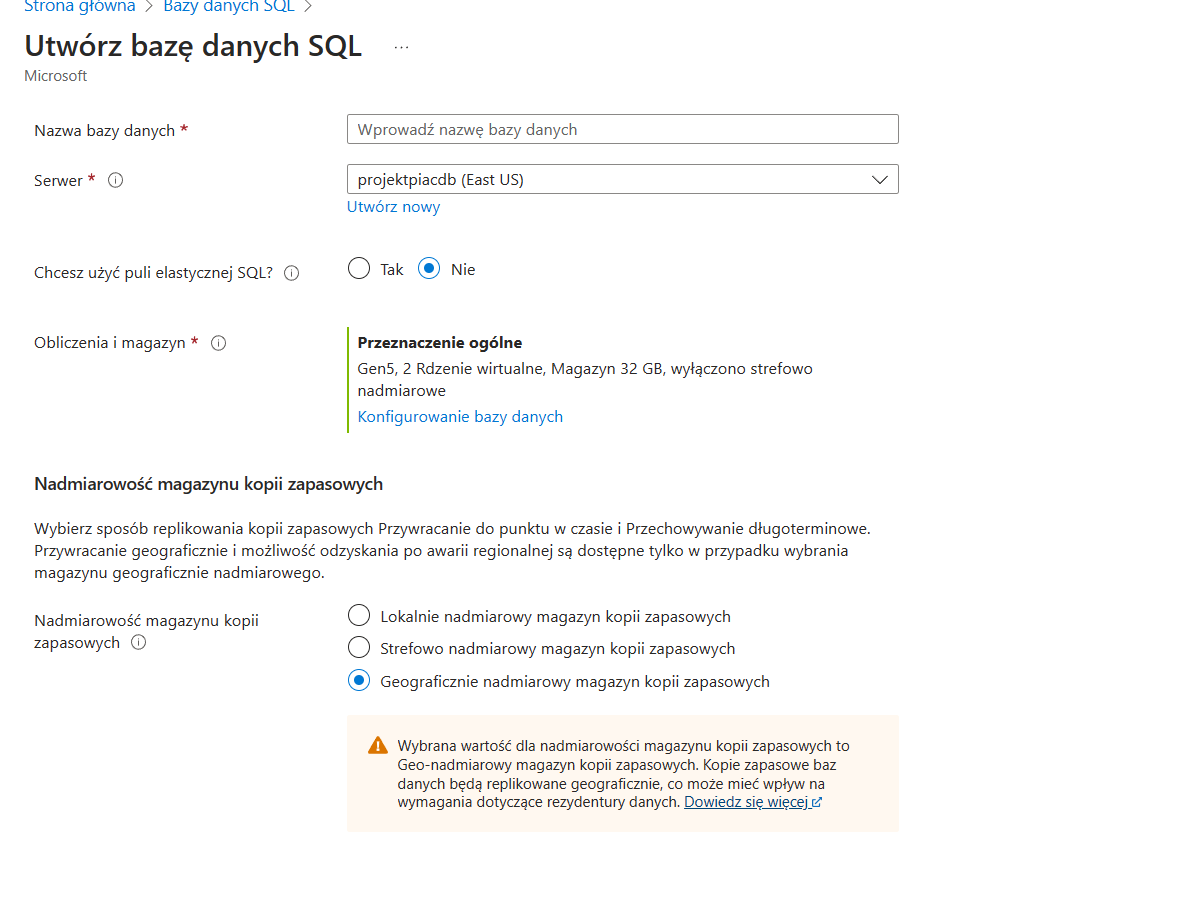


Powyższy kod prezentuje jak zaimplementować możliwość wysyłania maila z flask\_mail. Musimy stworzyć zmienne które będą przechowywały serwer, port po którym wyślemy mail oraz login oraz hasło do maila. W funkcji mail w zmiennej msg przechowujemy tytuł maila oraz dane wysyłającego oraz odbiorcy. W zmiennej msg.body dodajemy treść maila w tym momencie flask\_mail powinien już obsługiwać możliwość wysyłania poczty.

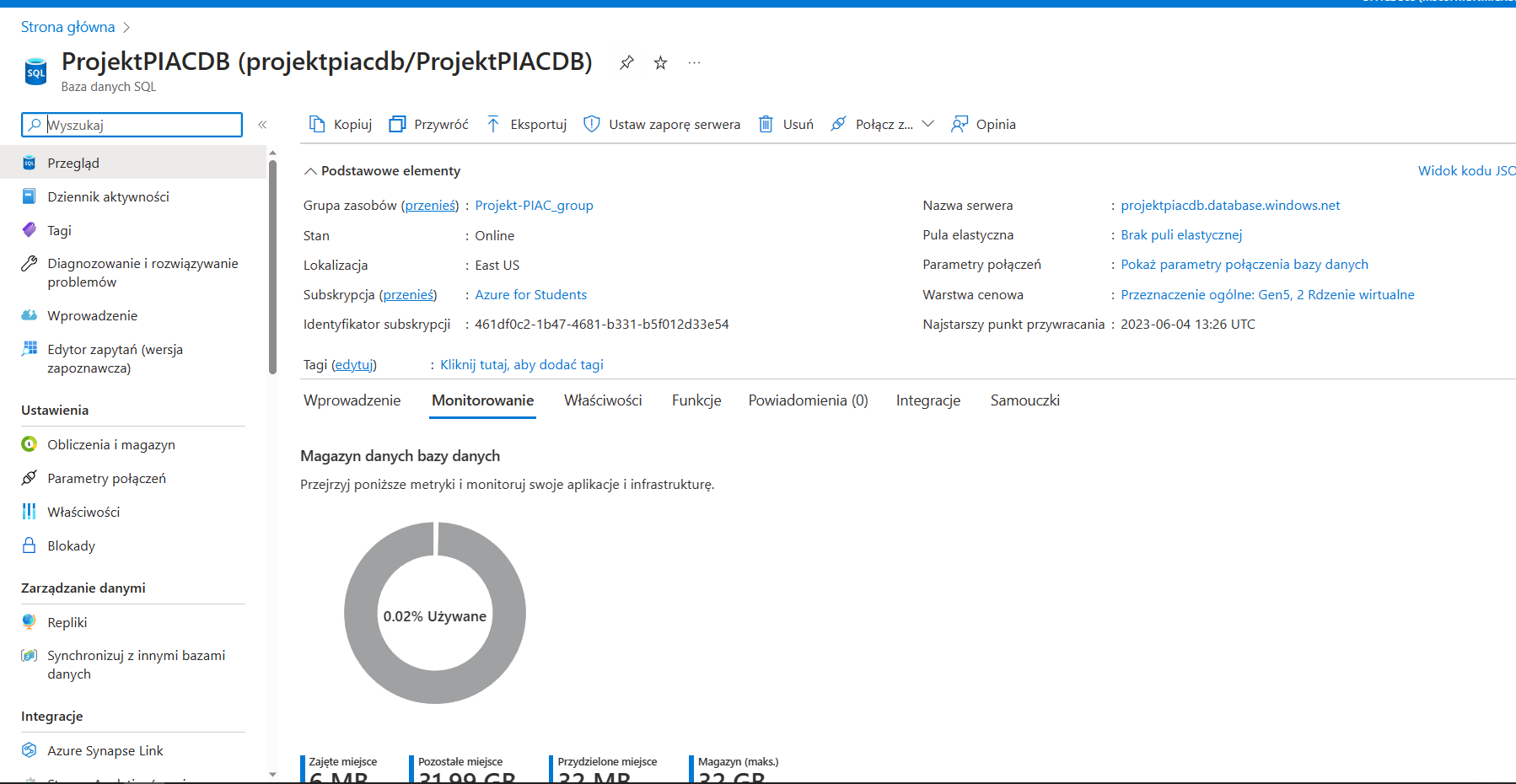


**Laboratorium 4**

****



W usłudze Azure możemy skorzystać z Azure SQL Database czyli usługi PaaS, która jest w pełni zarządzaną bazą danych. Aby stworzyć bazę danych wchodzimy w Bazy danych SQL i wybieramy stwórz. Wybieramy subskrypcję i wybieramy nazwę bazy danych i tworzymy serwer bazodanowy, który będzie dostępny pod linkiem ***nazwa***.database.windows.net. Wybieramy lokalizację serwera i ustawiamy resztę funkcjonalności zgodnie z zapotrzebowaniem. Po utworzeniu musimy poczekać chwilę na utworzenie bazy danych.



Aby dodać tabelę do bazy danych wchodzimy do edytora zapytań. W polu zapytanie możemy wpisać polecenie SQL które stworzy tabelę i doda do niej jakieś rekordy. W naszym przypadku będzie to:

**create table księga\_gosci (**

**nick varchar(50)**

**tekst varchar(250)**

**data\_dodania DATE**

**)**

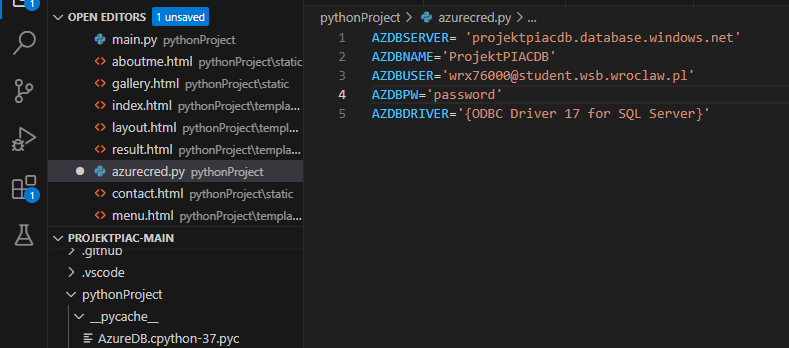
**insert into ksiega\_gosci(nick, text, data\_dodania) values (‘Marek’, ‘Podoba mi się’, GETDATE()),(‘Basia’, ‘Ciekawa stronka’, GETDATE());**

Aby zainstalować funkcjonalności flaska związane z bazą danych musimy wywołać komendę:

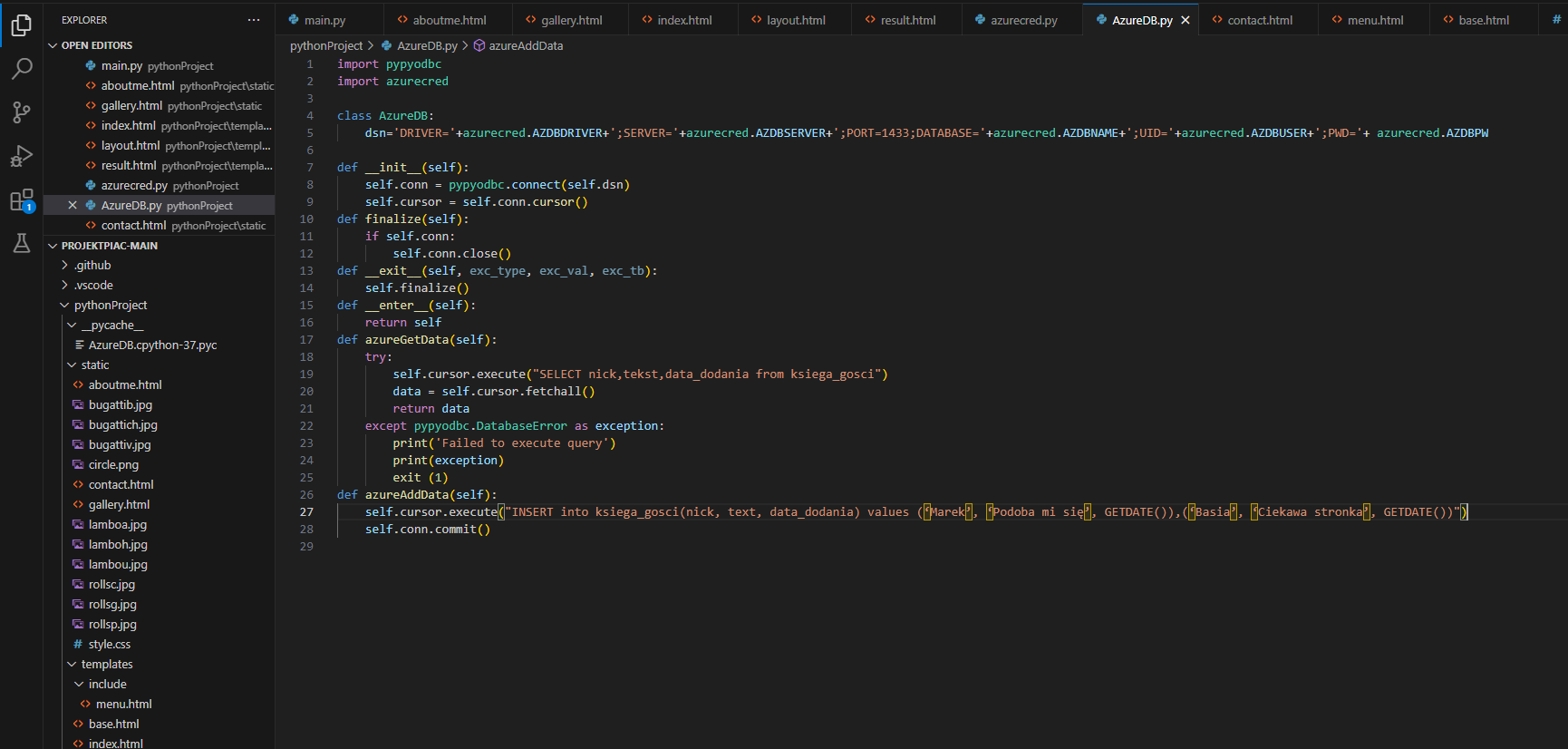
**pip3 install pypyodbc**

i zaimportować

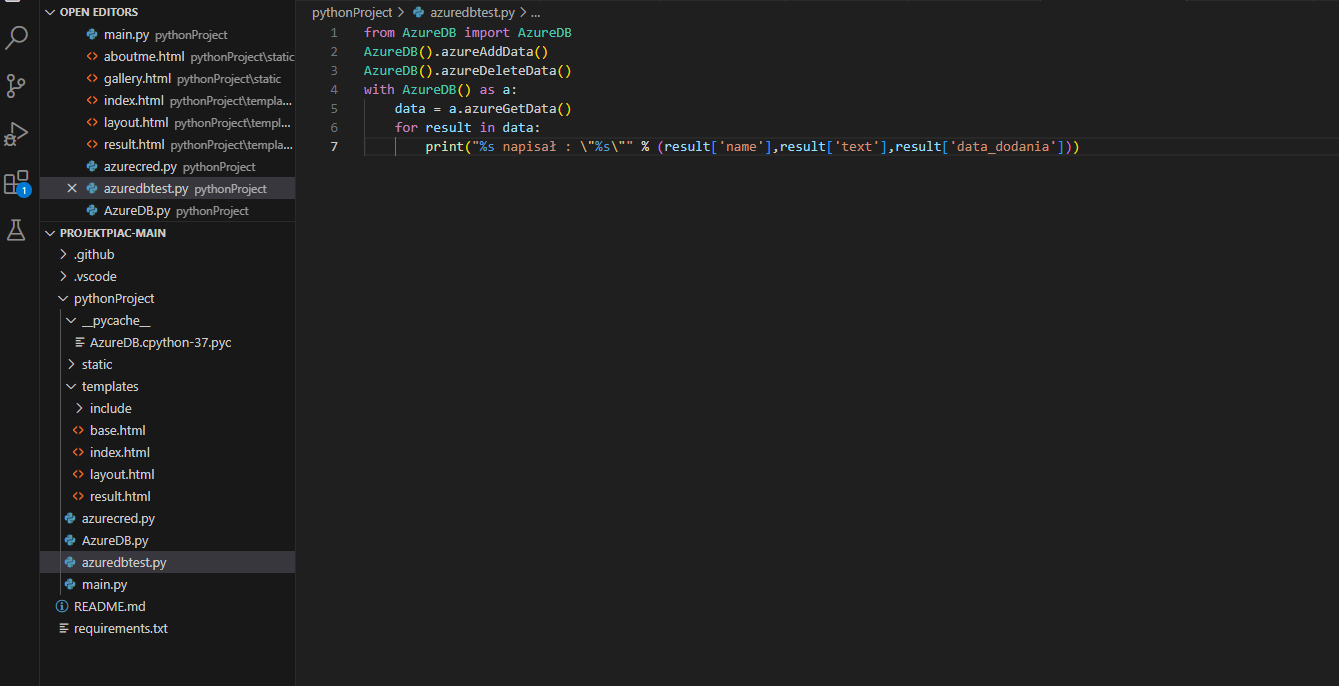
**from AzureDB import AzureDB**

****

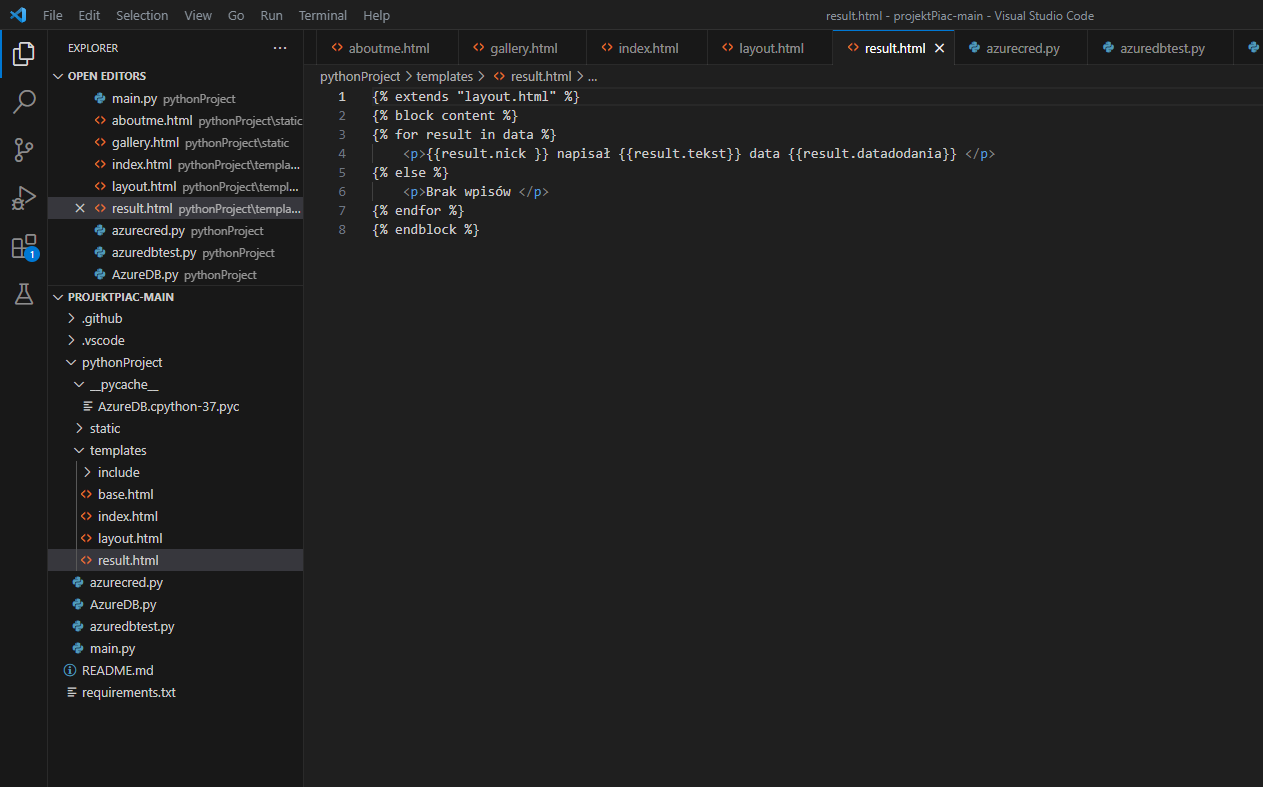
Następnie musimy utworzyć nowy plik python, który będzie zawierał informacje o połączeniu z bazą danych. Następnie musimy zainstalować sterownik ODBC aby móc komunikować się z bazą danych z poziomu naszego systemu operacyjnego.



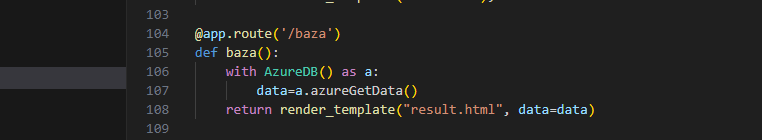
Następnie tworzymy plik AzureDB.py, który będzie się łączył z serwerem Azure SQL. Powyższy kod przedstawia kod który wykonuje to zadanie.

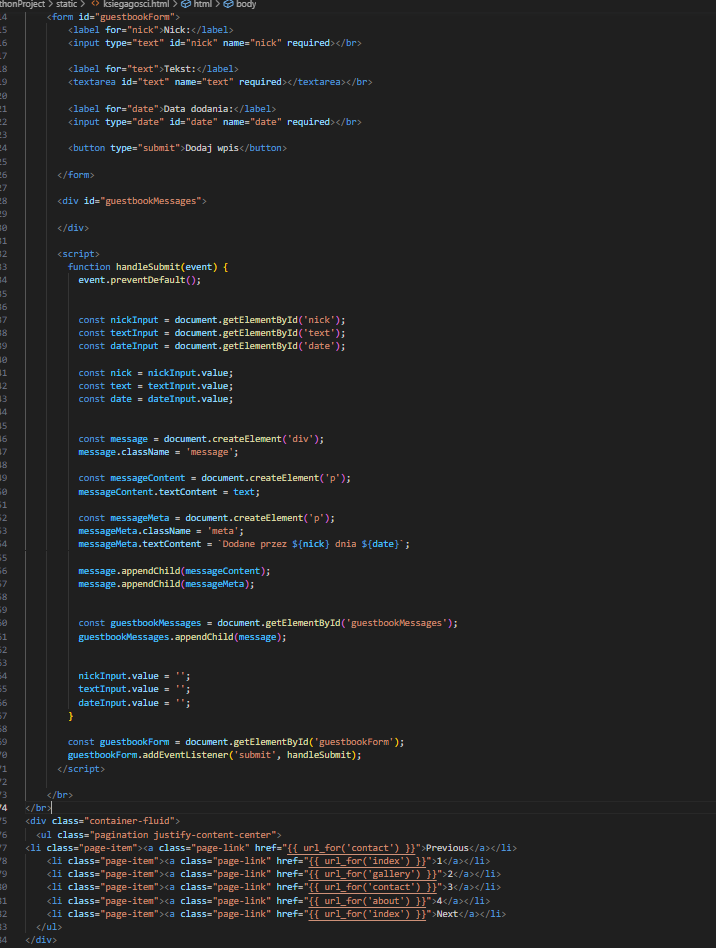


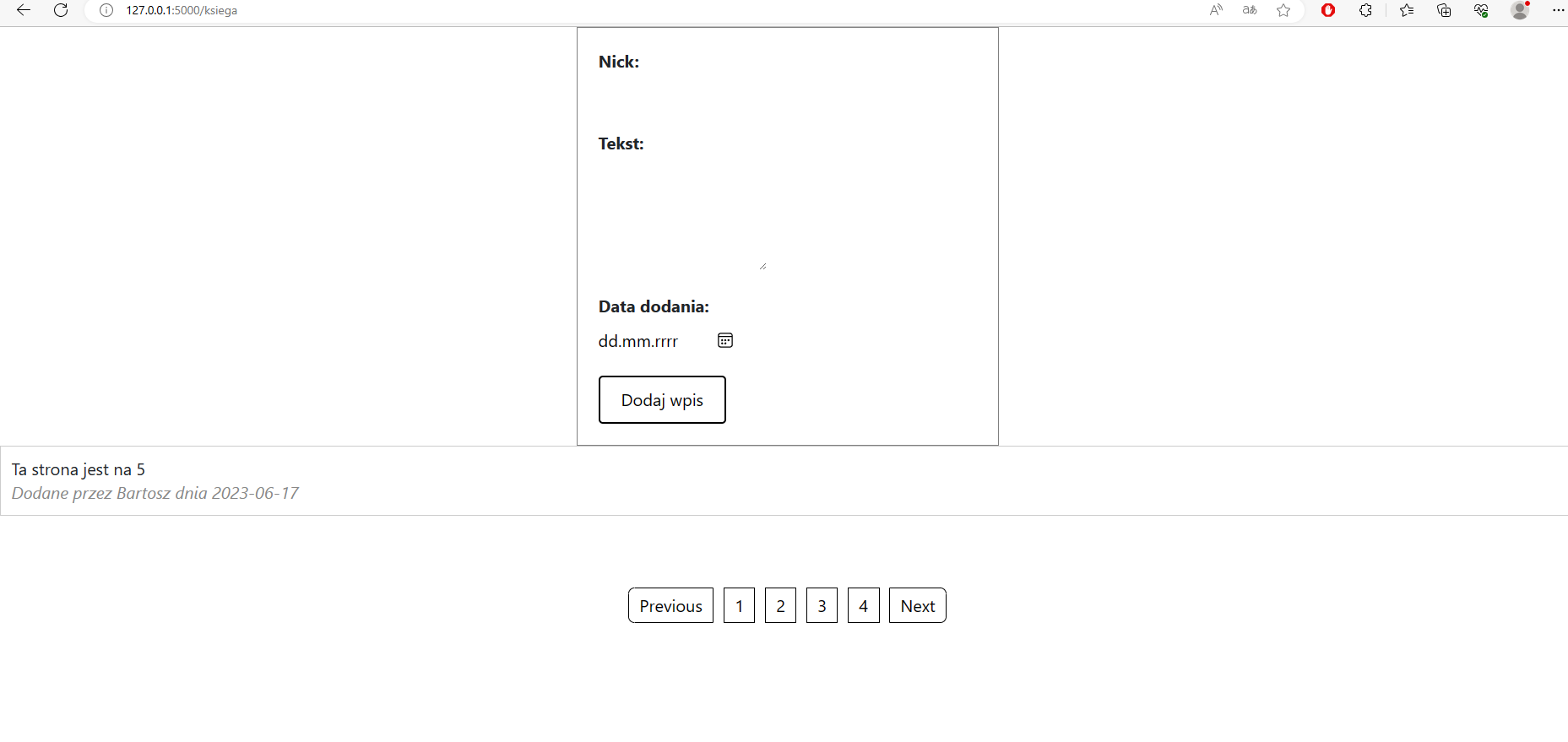
Tworzymy plik, który będzie testował połącznie z bazą danych. Powyższy kod prezentuje jak wykonać to zadanie.



Tworzymy pliki html, których zadaniem będzie wyświetlenie rezultatu naszej bazy danych.



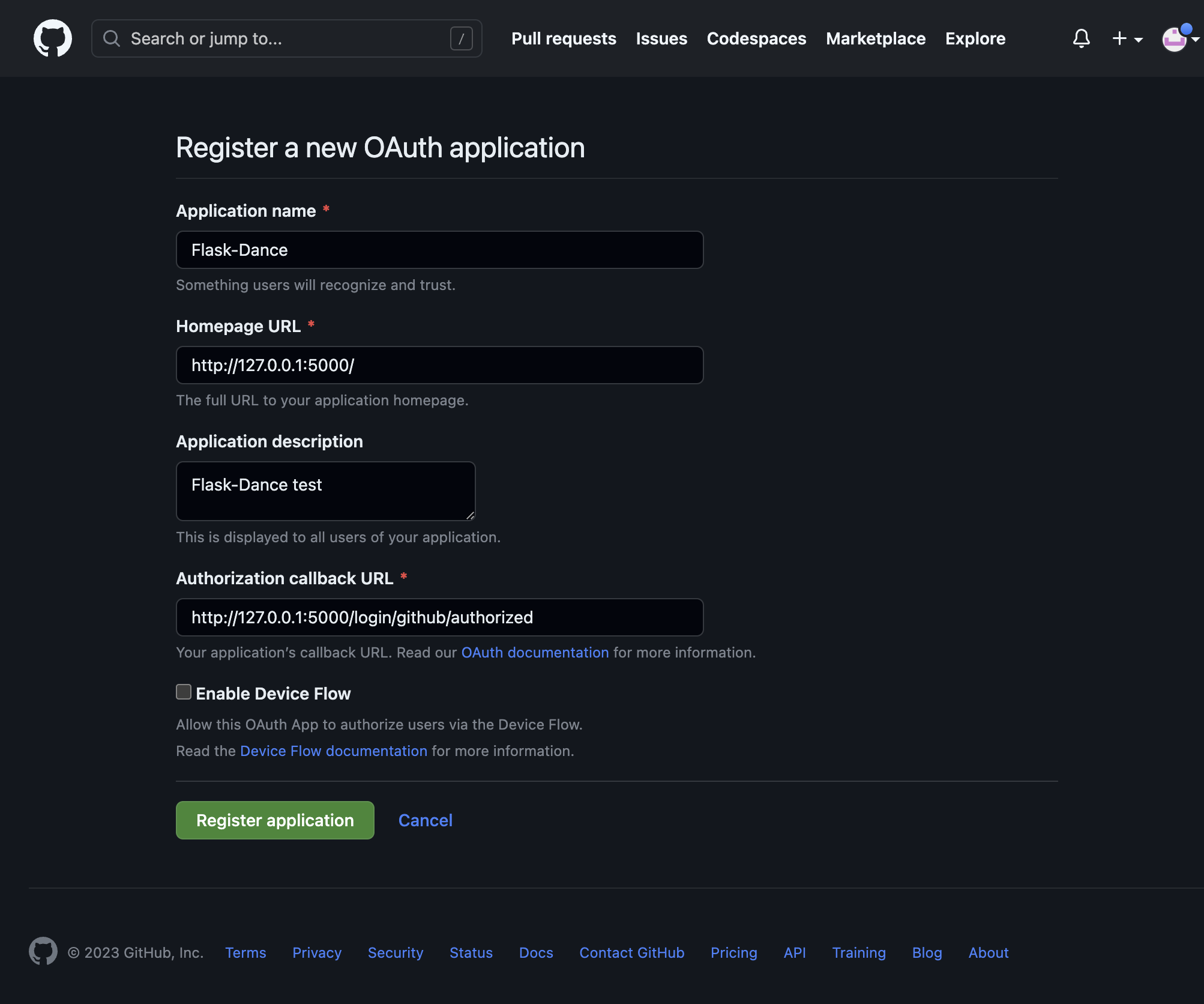
Tworzymy ścieżkę w głównym pliku flaska, który wskaże drogę do strony, która ukazuje rezultat bazy danych. Powyższy kod prezentuje strukturę kodu, który obsłuży to zadanie.



W zakładce Księga Gości tworzymy formularz, za pomocą html, css oraz JS, który daje możliwość dodawania na stronie wpisów. Możemy ustawić swój nick wybrać datę oraz napisać wiadomość, która wyświetli się na naszej stronie. Powyższy kod JS oraz html pokazuje jak stworzyć formularz, który pobiera dane od użytkownika i skrypt JS przetwarza te informacje i zamieszcza komentarz na stornie pod formularzem.

**LABORATORIUM 5**

Biblioteka Flask-Dance to biblioteka do autoryzacji użytkowników. Dzięki niej możemy skorzystać z protokołu OAuth czyli struktury do autoryzacji, która umożliwia dostęp do danych użytkowników. Dzięki OAuth możemy logować się za pośrednictwem Google, Facebook bądź Microsoft do aplikacji webowych. Aby wdrożyć autoryzację z wykorzystaniem GitHub musimy przejść do https://github.com/settings/applications/new. Wypełniamy dane w formularz.

Po zarejestrowaniu aplikacji wygenerujemy sekretny kod oraz identyfikator klienta, które będą nam potrzebne do użycia w kodzie aplikacji.

Następnym krokiem będzie zainstalowanie rozszerzenia FlaskDance komendą:

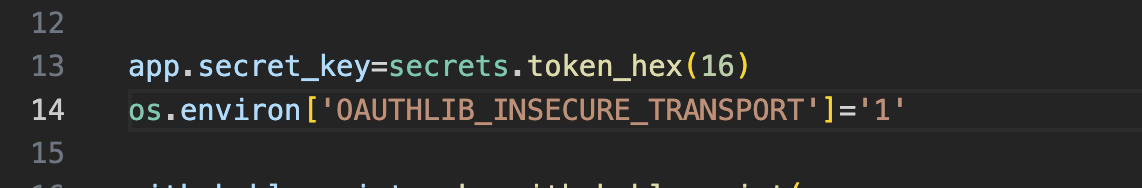
pip3 install flask-dance

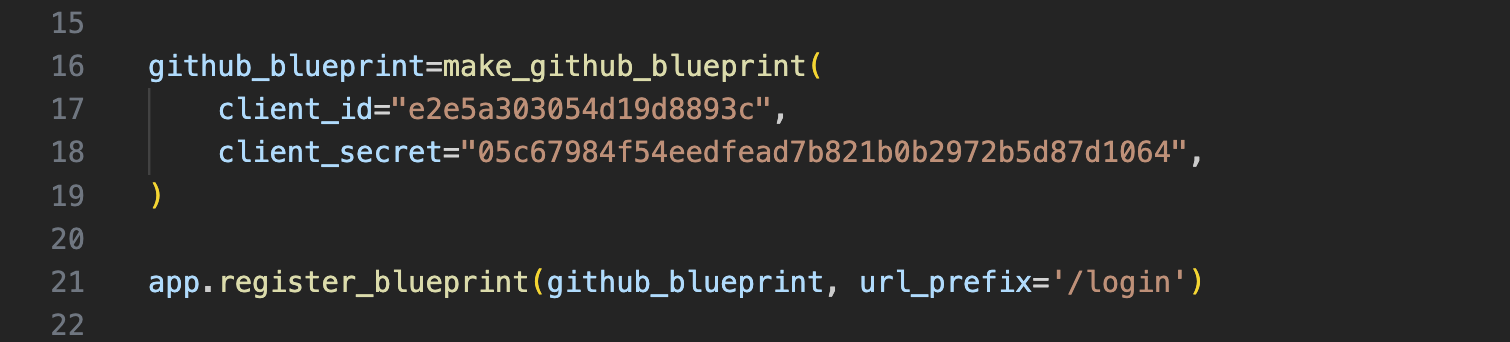
Następnie musimy zaimportować funkcjonalność:

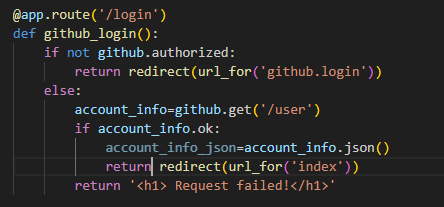
from flask\_dance.contrib.github import\_github\_blueprint, github

Import secrets

Import os

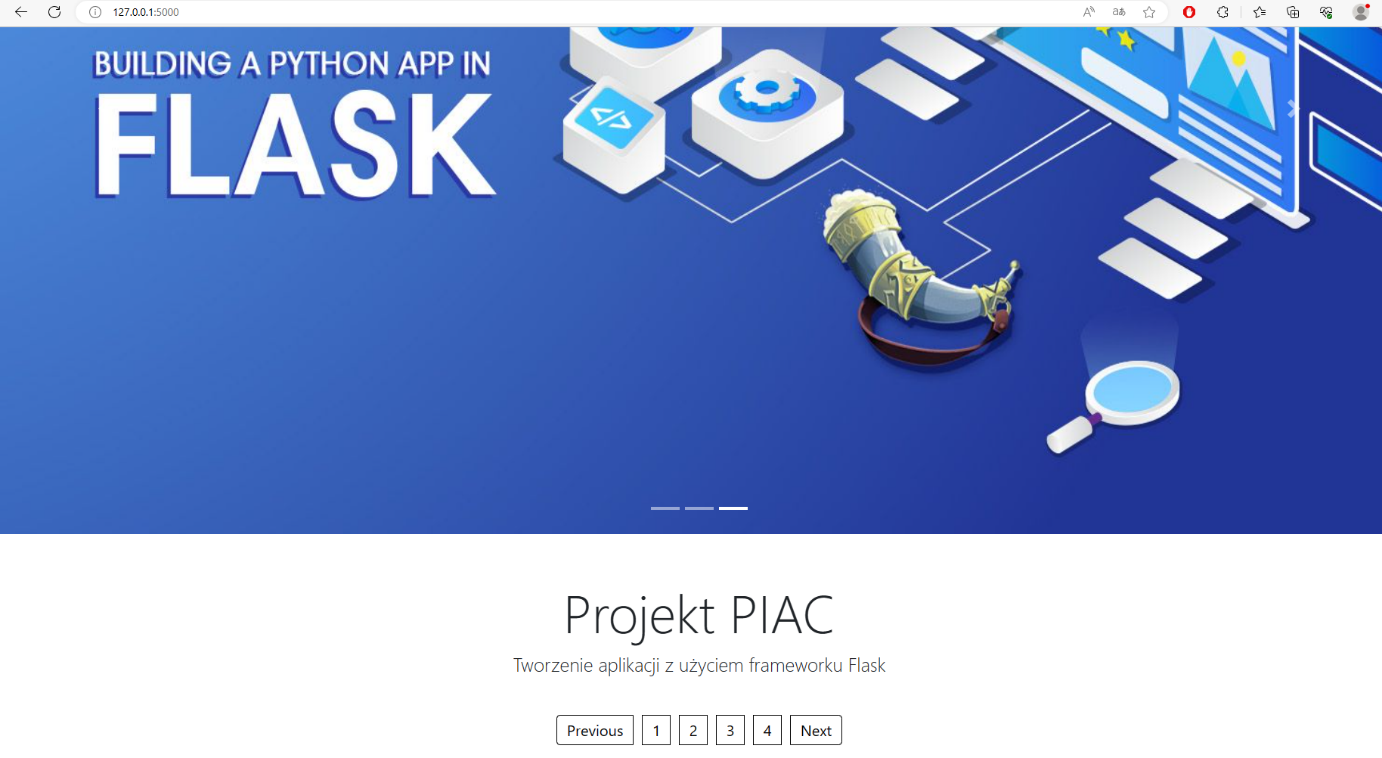
Następnie generujmy sekretny klucz i zezwalamy na połączenie w lokalnym środowisku bez https tylko przez http na potrzeby testów. W środowisku chmurowym nie powinniśmy zezwalać na takie rozwiązanie.

W tym miejscu wklejamy swój wygenerowany klucz z github oraz identyfikator klienta. Wywołujemy tutaj logowanie przez serwis GitHub oraz zwracamy informację o poprawnym logowaniu z informacją i nazwie naszej nazwy użytkownika.



Aby po poprawnym logowaniu przekierowywało nas na stronę główną musimy zaktualizować kod i dodać w nim

**return redirect(url\_for(‘index’))**

****

Dzięki OAuth użytkownik może udostępniać swoje zasoby z innych aplikacji. Ma możliwość logowania do aplikacji za pośrednictwem innych serwisów bez konieczności zakładania dodatkowego konta. Jeśli użytkownik udzieli odpowiednich zezwoleń aplikacje mogą pobrać informacje z innych kont użytkownika np zdjęcia.

Protokół OAuth jest niezależny od platformy dlatego można go skonfigurować jedną autoryzację dla aplikacji internetowej oraz mobilnej. Osobne konta nie są zawsze wymagane dla wybranych serwisów zależnie od polityki serwisu. Jeśli nie jest to Koneczne dobrą opcją będzie skorzystanie z OAuth.

Zaletami OAuth będzie ograniczenie zakładania wielu kont w wielu serwisach. OAuth nie przekazuje haseł do serwisów na których się logujemy co nie wpłynie negatywnie na bezpieczeństwo. Wadami OAuth może być złożona implementacja zastosowania w aplikacji. Jeśli deweloperzy źle skonfigurują autoryzację możemy narazić się na niebezpieczeństwo.

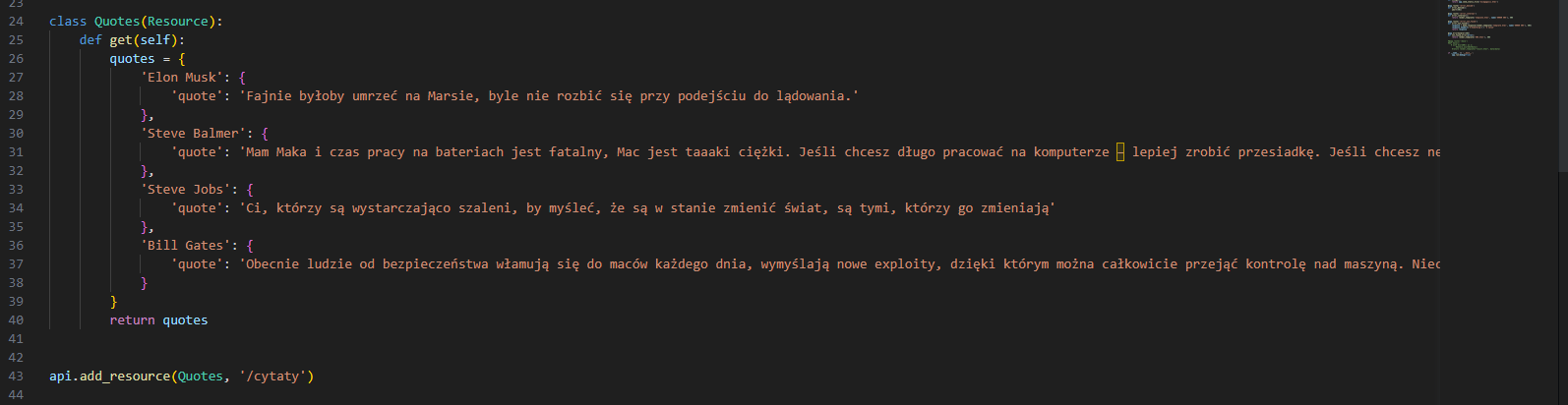
**Laboratorium 6**

W tym laboratorium skupimy się na tworzeniu REST API czyli uniwersalnego interfejsu http do komunikacji pomiędzy oprogramowaniem klienta a serwerem za pośrednictwem sieci. Flask i python jest jednym z najwyżej ocenianych technologii do tworzenia REST API. Flask-RESTful to rozszerzenie Flask do towrzenia tych interfejsów. Aby zainstalować to rozszerzeni używamy komendy

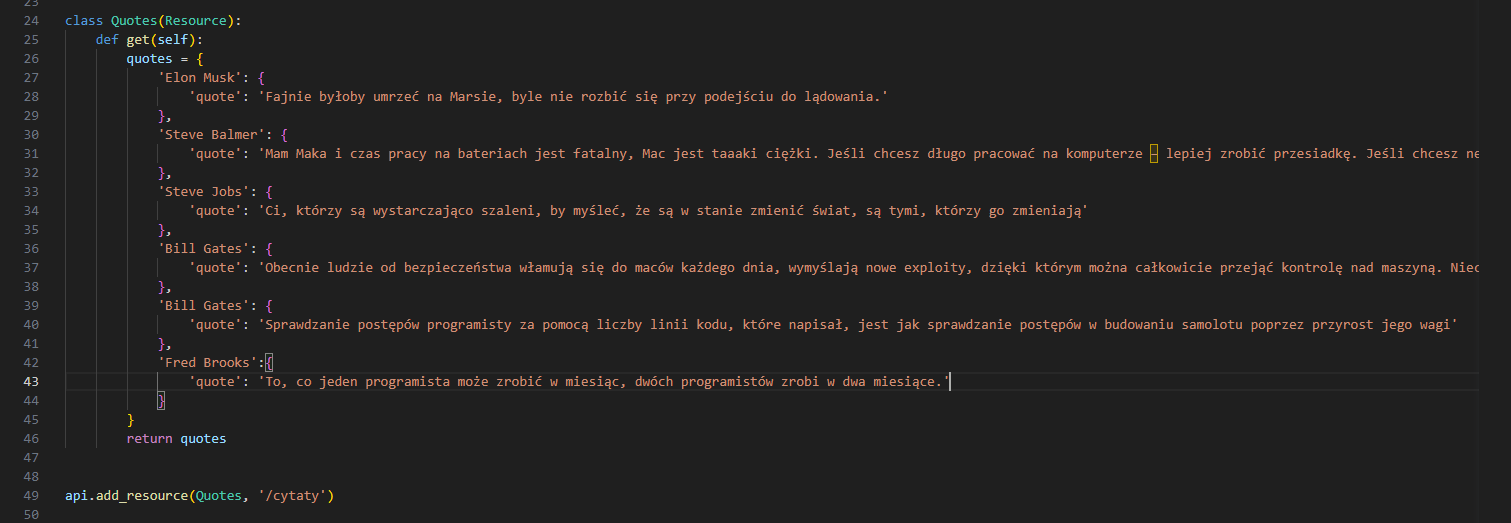
**pip3 install flask-restful**

Musimy również zaimportować to rozszerzenie

**from flask\_restful import Resource, Api**



W tym przykładzie stworzyliśmy serwer API wysyłający cytaty. Reprezentuje go klasa Quotes oraz metoda get.

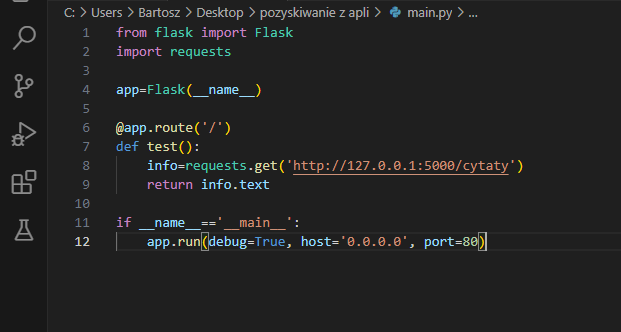


Po dodaniu swoich cytatów one również znajdują się na wyświetlanej liście przez API.

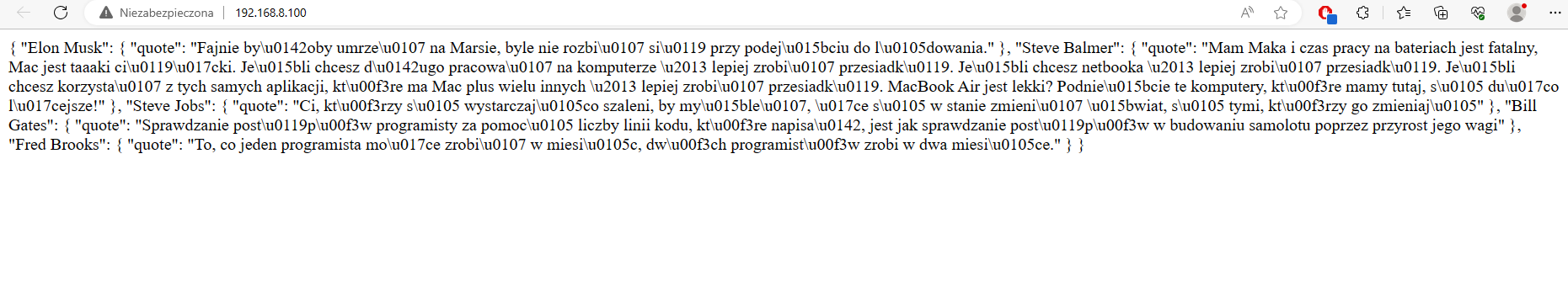


Po dodaniu metody put, post oraz delete kod wygląda jak na obrazku powyżej. Przy użyciu metody put możemy dodawać nowe cytaty do serwera podając cytat i autora. Metodą post możemy aktualizować już istniejące cytaty. Metoda deleta natomiast usuwa już istniejące cytaty jeśli podany autora cytatu.

Teraz stworzymy aplikację pobierającą dane z serwera REST API. Tworzymy nowy projekt i instalujemy bibliotekę Flask oraz request. Aby dwa projekty działały na jednym systemie musimy w drugim projekcie zmienić port na inny.

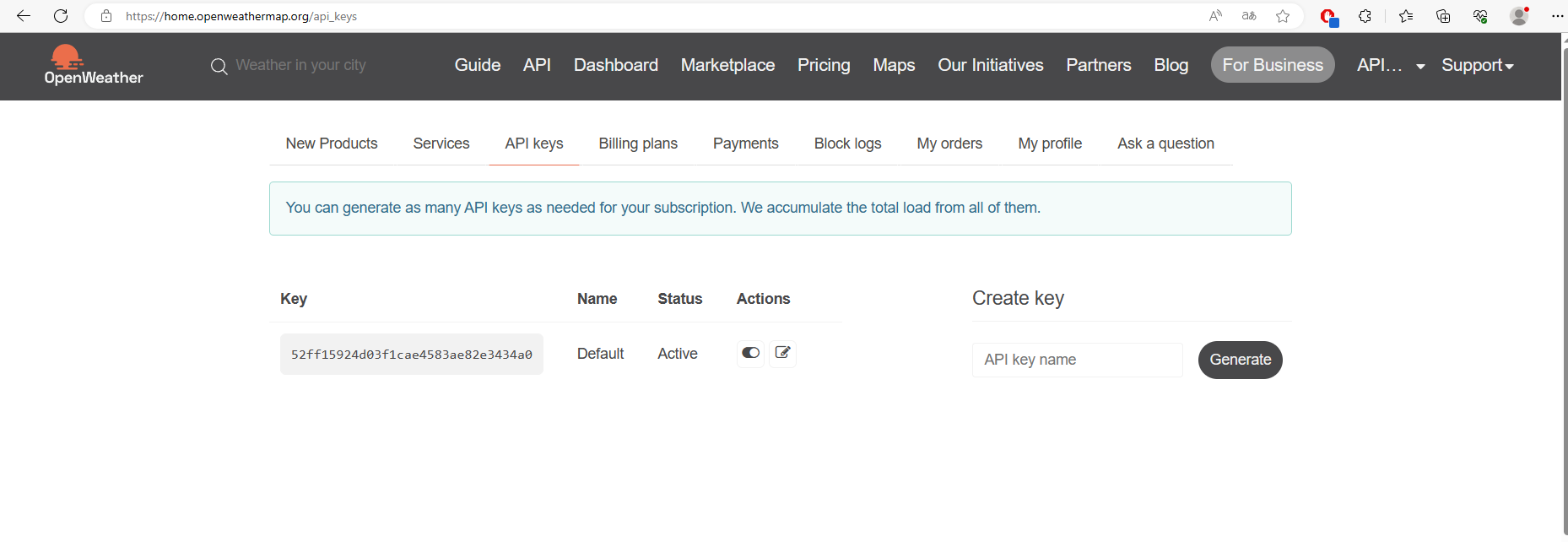


Powyższy kod prezentuje aplikację pobierająca dane z serwera REST API na porcie 80.



Po połączeniu się z serwerem na stronie wyświetlają się cytaty stworzone w poprzednim projekcie.

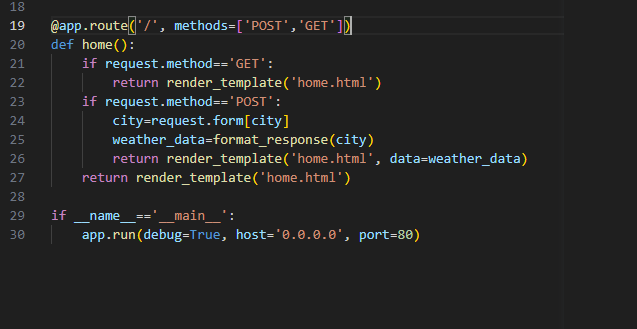
Teraz stworzymy aplikację pobierającą pogodę z serwisu. Aby to zrobić będziemy musieli musieć zarejestrować się w serwisie OpenWeatherMap aby wygenerować swój „klucz API” do użycia w kodzie.



Rejestrujemy się i w zakładce „API” mamy podstronę „My API key”



Tworzymy funkcję w której zawieramy klucz API oraz parametry jakie nas interesują.



Dodajemy routing oraz metodą która renderuje plik home.html.

W przypadku metody POST wybieramy miasto i dla tego miasta generuje nam się odpowiedz na podstawie danych. Metoda GET tylko stronę.

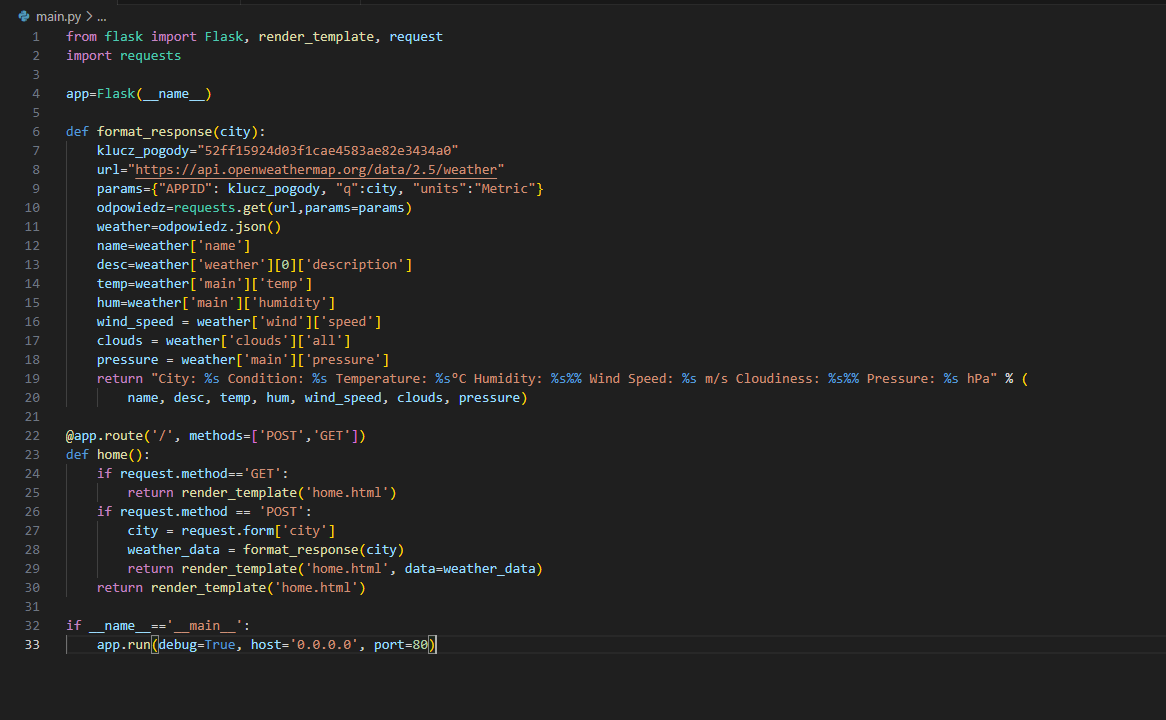


Powyższy kod przedstawia plik home.html, który wyświetla stronę z pogodą. Dodaliśmy prosty styl graficzny i bootstrap aby poprawić wizualne aspekty strony.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Na stronie można wpisać nazwę miasta i API wygeneruje odpowiedz o pogodzie dla wybranego miasta.



Wprowadzamy prostą aktualizację kodu o:

wind\_speed = weather['wind']['speed']

    clouds = weather['clouds']['all']

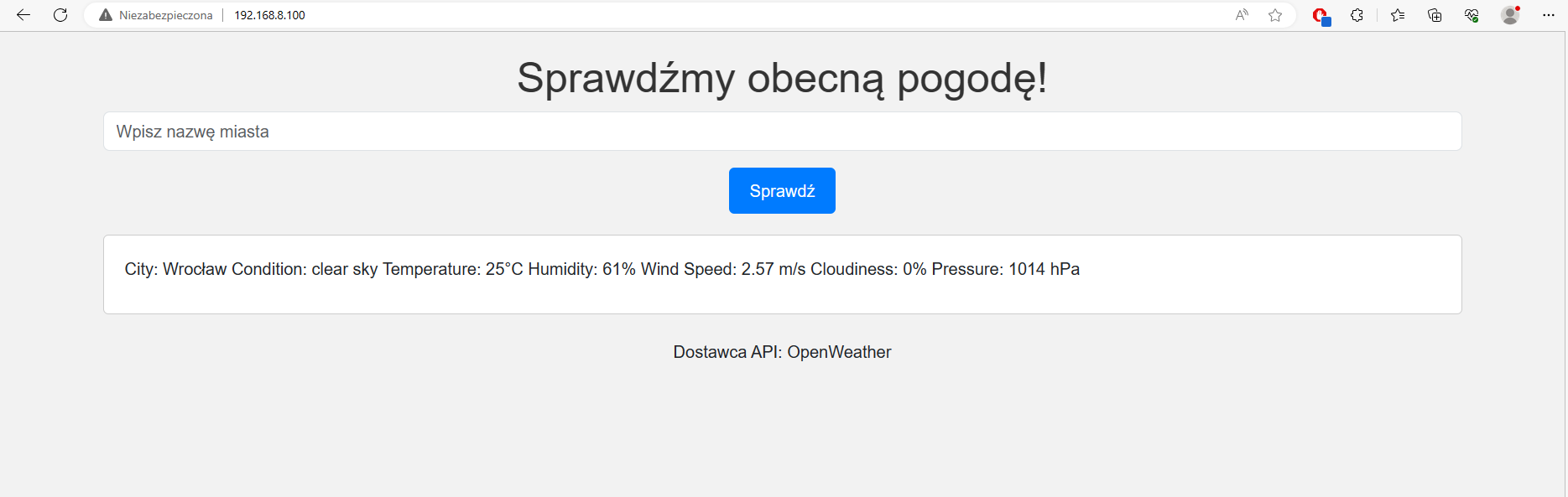
    pressure = weather['main']['pressure']

która dodaje informacje o wietrze, zachmurzeniu i ciśnieniu.

Aby poprawnie wyświetlić informację nie możemy zapomnieć o aktualizacji linijki return.

return "City: %s Condition: %s Temperature: %s°C Humidity: %s%% Wind Speed: %s m/s Cloudiness: %s%% Pressure: %s hPa" % (

        name, desc, temp, hum, wind\_speed, clouds, pressure)



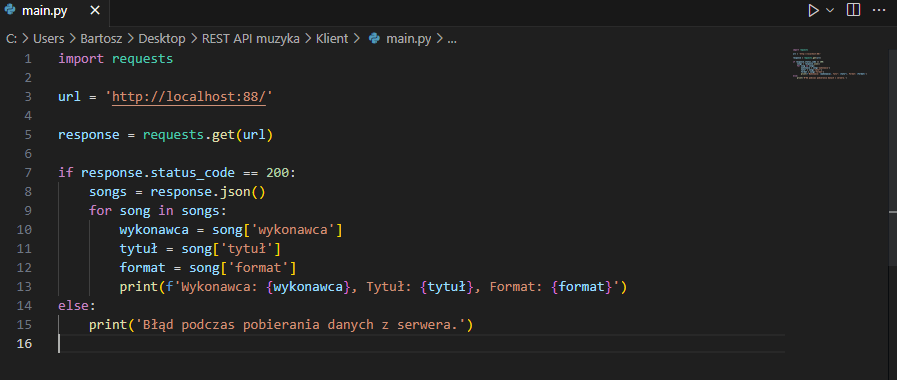
Teraz na stronie wyświetlają się dodatkowe informacje o pogodzie zaciągane z serwera REST API.

Częstotliwość aktualizacji danych API zależy od subskrypcji. Dla planu darmowego jest to czas do 2 godzin.

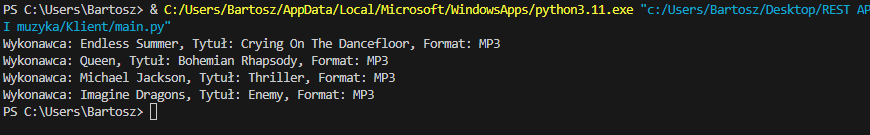
Implementacja własnego serwera REST API



Powyższy kod przedstawia serwer REST API. Instalujemy flaska i tworzymy klasę, w której tworzymy funkcję get do wyświetlania informacji o utworach. Wypisujemy listę utworów w tablicy i tworzymy zasoby dodając api.add\_resources. Teraz nasze api będzie dostępne na serwerze po urychomieniu.



Tworzymy prostego klienta wyświetlającego wykonawcę, tytuł i format utworu, które odczytuje z serwera.



Po odpaleniu klienta dane z serwera ukazują się w terminalu po stronie klienta.

Na platformie Microsoft Azure możliwe jest hostowanie serwera REST API za pomocą usługi chmurowej Azure App Service. Azure App Service oferuje elastyczne i skalowalne środowisko do hostowania różnych rodzajów aplikacji, w tym serwerów REST API.