Web API - Dev

Zadania do wykonania:

1. Pobierz i zainstaluj Intellij Community edition <https://www.jetbrains.com/idea/download/#section=windows>
2. Pobierz i zainstaluj Python 2.7.0 (MSI Installer)

<https://www.python.org/download/releases/2.7/>

1. Pobierz i zainstaluj Java JDK 1.8

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

1. Pobierz i zainstaluj Git Bash

<https://git-scm.com/downloads>

1. Dodaj lub zmodyfikuj zmienne środowiskowe:
   1. Przejdź do ‘Panel sterowania\System i zabezpieczenia\System’
   2. Wybierz ‘Zaawansowane ustawienia systemu’
   3. Wybierz ‘Zmienne środowiskowe’
   4. Dodaj lub zmodyfikuj ‘JAVA\_HOME’, wartość: C:\Program Files\Java\jdk1.8.0\_151
   5. Dodaj lub zmodyfikuj ‘Path’, wartości: C:\Python27; C:\Python27\Scripts; C:\Program Files\Git\cmd; C:\Program Files\Git\bin
2. Otwórz Git Bash i stwórz na dysku C: strukturę katalogów TechnicznyTester -> api -> resources oraz TechnicznyTester -> api -> rest
3. Z lokalizacji <https://github.com/przemastro/Arqonia> skopiuj plik distribute\_setup.py do katalogu TechnicznyTester
4. Otwórz konsolę cmd i wykonaj następujące czynności:
   1. stworzyć plik get-pip.py i wypełnić go zawartością z<https://pip.pypa.io/en/stable/installing/>
   2. I kolejną komendę "pip install flask"
   3. I komendę "pip install flask-restful"
5. W katalogu resources stwórz plik env.properties i wypełnij go następującymi danymi:

[Server]

server.address=localhost

server.port=5001

server.service=Yes

[DatabaseConnection]

database.address=Driver={SQL Server};Server=”Your Server Name”;Database=Recruitment;Trusted\_Connection=yes;uid=”Your uid”;pwd=

1. W katalogu stwórz plik Queries.properties i wypełnij go jednym wierszem:

[DatabaseQueries]

1. W katalogu rest dodaj plik api.py, w którym stwórz serwis restowy, który będzie składał się z następujących elementów:
   1. Sekcji odpowiedzialnej za wczytanie parametrów z pliku konfiguracyjnego env.properties
   2. Parser’a, który będzie zawierał definicje parametrów do metody POST dodającej nowego kandydata do bazy
   3. Dwóch klas, z których każda będzie zawierała po jednej metodzie HTTP. Pierwsza klasa będzie zawierała metodę GET, druga klasa będzie zawierała metodę POST
   4. Definicji endpointów: statycznego /<rest\_id> i ‘dynamicznego’ /addCandidate
   5. Sekcji odpowiedzialnej za obsługę requestów CORS
2. W katalogu rest dodaj drugi plik jsonBuilder.py, w którym stwórz dwie funkcje:
   1. *candidatesProcessed()* - odpowiedzialna za wczytanie danych z tabeli *dbo.candidatesProcessed* i sklejenie w postaci Json’a
   2. *addCandidate(name, lastName, level, area, salary, grade)* - odpowiedzialna za zainsertowanie nowego rekordu, odpalenie procedury składowanej *dbo.recruitmentProcedure* i zwrócenie nowego Json’a z danymi z tabeli *dbo.candidatesProcessed*
3. W konsoli git bash przejdź do katalogu rest i wykonaj komendę ‘python api.py’ jeśli serwis nie wstanie lub co gorsza pojawią się błędy doinstaluj brakujące pakiety komendą ‘pip install [nazwa pakietu]’

Podpowiedzi:

1. N/A
2. N/A
3. N/A
4. N/A
5. N/A
6. Komenda ‘cd [nazwa katalogu]’ pozwala przechodzić pomiędzy katalogami

Komenda ‘mkdir [nazwa katalogu]’ pozwala stworzyć katalog

Komenda ‘cd ..’ pozwala przejść katalog wyżej/cofnąć się

Komenda ‘ls’ listuje nam zawartość katalogu

Komenda ‘touch [nazwa pliku]’ dodaje nam plik

1. N/A
2. N/A
3. N/A
4. Przykładowe zapytanie, które można by tutaj wrzucić wyglądałoby następująco

database.getName=select Name from [Recruitment].[dbo].[candidatesProcessed] where ID=

1. Należy pamiętać aby moduł pythonowy mógł korzystać z jakichś funkcji/metod itd. muszą one być wpierw zaimportowane. Wrzuca się to zawsze na początku pliku:

from flask import Flask, jsonify, request

from flask\_restful import reqparse, Api, Resource

from jsonBuilder import candidatesProcessed, addCandidate

import ConfigParser

from threading import \*

Cały serwis/kod musi się znajdować pomiędzy:

app = Flask(\_\_name\_\_)

api = Api(app)

A

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=False, host=serverAddress, port=serverPort, threaded=True, use\_reloader=True)

Ostatnia linijka to nic innego jak uruchomienie serwisu z parametrami z env.properties.

* 1. Parametry te należy przedtem zdefiniować:

config = ConfigParser.RawConfigParser()

config.read('../resources/env.properties')

serverAddress = config.get('Server', 'server.address');

serverPort = int(config.get('Server', 'server.port'));

serverService = config.get('Server', 'server.service');

* 1. Parser’a definiujemy w następujący sposób:

parser = reqparse.RequestParser()

parser.add\_argument('name', type=str)

parser.add\_argument('lastName', type=str)

parser.add\_argument('level', type=str)

parser.add\_argument('area', type=str)

parser.add\_argument('salary', type=str)

parser.add\_argument('grade', type=str)

Od razu zdefiniowaliśmy także parametry do metody POST

* 1. Klasa pierwsza zawierająca metodę GET będzie się odnosiła do statycznego endpoint’a, który będzie jedynie wykorzystywany przy pierwszym wejściu do aplikacji, albo przy odświeżaniu okienka. Będzie ona statyczna, czyli dopóki nie wyłączymy serwisu będzie zwracała zawsze te same początkowe dane.

class Rest(Resource):

def get(self, rest\_id):

return jsonify(REST[rest\_id])

Wykorzystaliśmy w tym przypadku zmienną REST, którą należy spopulować/zdefiniować przed wykorzystaniem jej w klasie:

REST = {'data': candidatesProcessed()}

Jak widać jest to ni mniej ni więcej lista z jednym parametrem data i wartościami z funkcji *candidatesProcessed()*. O tym jeszcze za chwilę. Aby ta lista stała się w pełni JSON-em musi zostać “zjsonowana” słówkiem “jsonify”

Druga klasa zawiera metodę POST, która to wywoływać będzie funkcję wrzucającą nowego kandydata do tabelki *dbo.candidates*, odpali procedurę składowaną i zwróci nową listę kandydatów z tabeli *dbo.candidatesProcessed*

class RestAddCandidate(Resource):

def post(self):

args = parser.parse\_args()

data = addCandidate(args['name'], args['lastName'], args['level'], args['area'], args['salary'], args['grade'])

return jsonify(data)

* 1. Endpointy definiowane są poniżej klas i są z nimi ściśle związane:

api.add\_resource(Rest, '/<rest\_id>')

api.add\_resource(RestAddCandidate, '/addCandidate')

* 1. Na koniec obsługa Cors

@app.after\_request

def after\_request(response):

response.headers.add('Access-Control-Allow-Origin', '\*')

response.headers.add('Access-Control-Allow-Headers', 'Content-Type,Authorization,SessionId,Email')

response.headers.add('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE')

response.headers.add("Access-Control-Max-Age", "3600");

response.headers.add("Access-Control-Allow-Headers", "x-requested-with");

response.headers.add("Connection", "keep-alive");

response.headers.add("Vary", "Accept-Encoding");

return response

1. I znowu musimy rozpocząć moduł od pobrania potrzebnych modułów, metod i funkcji, które będą wykorzystywane:

import pyodbc

import ast

import ConfigParser

import time

Oraz zdefiniowania configParser’a do przyszłego wyciągania zapytań z pliku Queries.properties oraz nazwy bazy z pliku env.properties

config = ConfigParser.RawConfigParser()

config.read('../resources/env.properties')

dbAddress = config.get('DatabaseConnection', 'database.address');

queries = ConfigParser.RawConfigParser()

queries.read('../resources/queries.properties')

cnx = pyodbc.connect(dbAddress)

cursor = cnx.cursor()

* 1. Zacznijmy od zdefiniowania prawie pustej funkcji. Na chwilę obecną zawiera od zestaw słówek try, except i else celem wychwycenia jakichś wyjątków, które mogą się zdarzyć w trakcie egzekucji. Mamy również obsługę połączenia z bazą i zdefiniowanie tzw. cursor’a, który wyciąga dane z bazy.

def candidatesProcessed():

try:

cnx = pyodbc.connect(dbAddress)

cursor = cnx.cursor()

except:

print 'errors in candidatesProcessed function'

else:

cnx.close()

Po zdefiniowaniu cursora można przystąpić do wyciągania linijka po linijce danych z tabeli dbo.candidatesProcessed.

getIdsFromCandidatesProcessed = queries.get('DatabaseQueries', 'database.getIdsFromCandidatesProcessed');

getIds = fetch\_all(getIdsFromCandidatesProcessed)

i = 0

controller = ''

if(len(getIds) != 0):

for counter in getIds:

id = str(counter)

i = i + 1

getName = (queries.get('DatabaseQueries', 'database.getName') + id)

getLastName = (queries.get('DatabaseQueries', 'database.getLastName') + id)

getAccepted = (queries.get('DatabaseQueries', 'database.getAccepted') + id)

object = {'id': id, 'name': str(fetch\_one(getName)), 'lastName': str(fetch\_one(getLastName)),

'accepted': str(fetch\_one(getAccepted))}

controller = str(object) + ',' + controller

controller = ast.literal\_eval(controller[:-1])

cursor.close()

if(i==1):

data = [controller]

elif(i>1):

data = controller

else:

data = [{'id': '', 'name': '', 'lastName': '', 'accepted': ''}]

return data

Co my tutaj widzimy?

Po pierwsze wyciągamy do zmiennej getIds liczbę rekordów w tabeli *dbo.candidatesProcessed*. Następnie iteracyjnie wyciągamy name, lastName i accepted i w zmiennych object i controller sklejamy listę pod przyszłego JSON’a.

Na dole mamy jeszcze obsługę przypadków gdy jest tylko jeden rekord albo nie ma ich wcale.

Przy czytaniu wartości skorzystaliśmy z dwóch przez nas zdefiniowanych funkcji fetch\_one - wyciągnij jedną wartość z kolumny

def fetch\_one(get\_value):

cursor.execute(get\_value)

Value = cursor.fetchone()

Value = Value[0]

return Value

fetch\_all - wyciągnij wszystkie wartości z kolumny

def fetch\_all(get\_value):

cursor.execute(get\_value)

Value = cursor.fetchall()

Value = [oc[0] for oc in Value]

return Value

* 1. Zaczynamy również od zdefiniowania nie zaimplementowanej funkcji

def addCandidate(name, lastName, level, area, salary, grade):

try:

cnx = pyodbc.connect(dbAddress)

cursor = cnx.cursor()

except:

print 'errors in addCandidate function'

else:

cnx.close()

Tym razem jest to funkcja z parametrami. Nasz funkcja będzie robiła 3 rzeczy:

Zainsertuj na podstawie parametrów nowy rekord z nowym ID do tabeli *dbo.candidates*. Odpal procedurę składowaną i stwórz nową listę pod przyszłego JSON’a na podstawie nowych danych z tabeli *dbo.candidatesProcessed*.

getLastId = queries.get('DatabaseQueries', 'database.getLastIdFromCandidates')

cursor.execute(getLastId)

lastId = cursor.fetchone()

if lastId is None:

lastId = 1

else:

lastId = lastId[0] + 1

lastId = str(lastId)

insertCandidate = "insert into [Recruitment].[dbo].[candidates] " \

"values('"+lastId+"', '"+name+"', '"+lastName+"', '"+level+"', '"+area+"', '"+salary+"', '"+grade+"')"

cursor.execute(insertCandidate)

cnx.commit()

time.sleep(1)

processData = "exec [Recruitment].[dbo].[recruitmentProcedure]"

cursor.execute(processData)

cnx.commit()

data = candidatesProcessed()

return data

1. N/A

Rozwiązanie:

Pełne rozwiązanie znajduje się w pliku App.7z