**DOKUMENTACJA**

**PROJEKTU**

**Technologia: REST (Http)**

**Język programowania: Python**

**Skład osobowy grupy:**

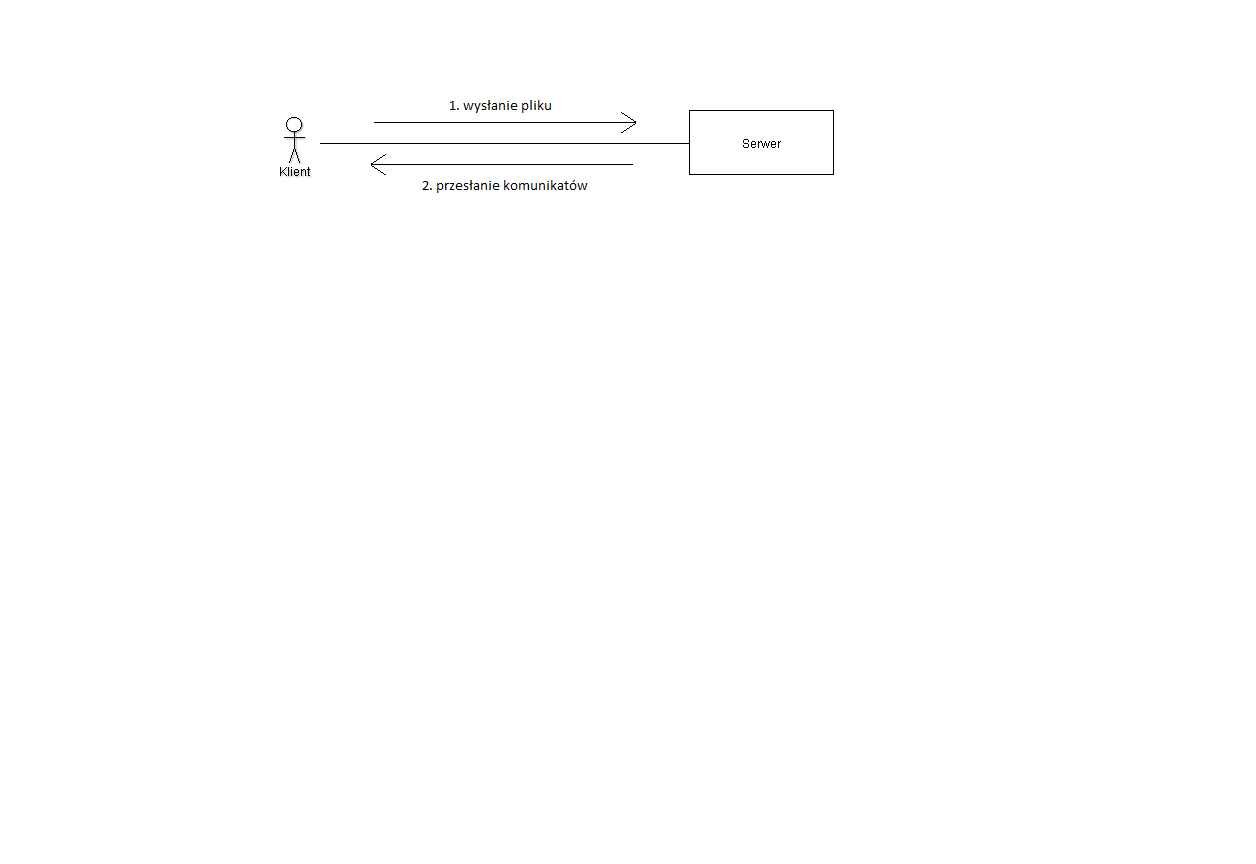
* **Kiełbania Piotr**
* **Kowalska Paulina**
* **Kurkowski Wojciech**
* **Machalska Anna**
* **Mazurek Paweł**
* **Pasierbiewicz Karolina**

**Analiza dostępności**

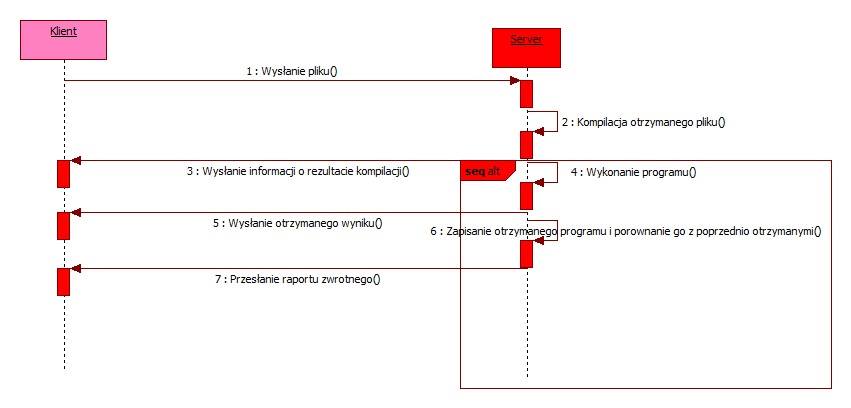
**REST** (ang. **Re**presentational **S**tate **T**ransfer) jest wzorcem narzucającym dobre praktyki tworzenia architektury aplikacji rozproszonych. Jest jednym z najpopularniejszych technologii wykorzystanych do implementacji wspomnianych aplikacji rozproszonych. Dostępność technologii REST dla wielu języków programowania w tym języka Python daje możliwość szybkiego i łatwego zaimplementowania prostej aplikacji rozproszonej wykonującej założone zadania przez projektanta.

**Diagramy UML**

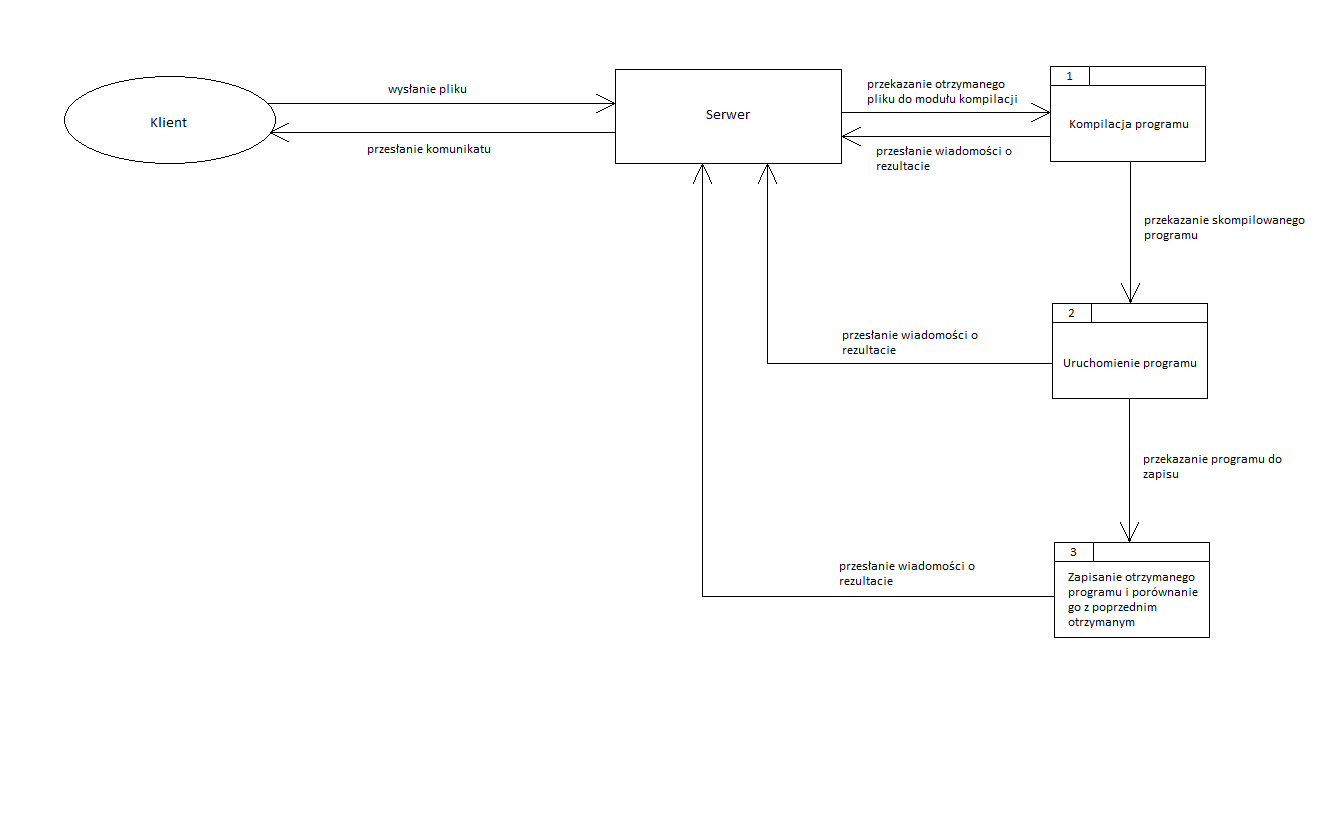
* **Diagram komunikacji**

****

* **Diagram sekwencji**

****

* **Diagram przepływu danych**

****

**Testy oprogramowania**

**Kody źródłowe oraz skrypty**

**server.py**

**from** http.server **import** HTTPServer, SimpleHTTPRequestHandler  
**from** socketserver **import** ThreadingMixIn  
  
*#from PythonREST import executor, compiler, reporter***import** executor, compiler, reporter  
  
  
*# import executor, compiler  
# import threading***class** Server(ThreadingMixIn, HTTPServer):  
 **def** run(self, IP, PORT):  
 *"""  
 starts server at given host and port* **:param** *IP: host's IP to start server* **:param** *PORT: server's port to listen requests* **:return***: None  
 """* server = Server((IP, PORT), MyHandler)  
 print(**"server started."**)  
 server.serve\_forever()  
  
  
**class** MyHandler(SimpleHTTPRequestHandler):  
 **def** \_set\_headers(self):  
 *"""sets headers"""* self.send\_response(200)  
 self.send\_header(**'content-type'**, **'text/html'**)  
 self.end\_headers()  
  
 **def** do\_PUT(self):  
 length = int(self.headers[**'Content-Length'**])  
 content = self.rfile.read(length)  
 self.\_set\_headers()  
 self.wfile.write(**"file successfully received\n"**.encode())  
  
 **try**:  
 c = compiler.Compiler()  
  
 c.compile\_file(content)  
 self.wfile.write(**"compilation successful\n"**.encode())  
 **except** Exception **as** e:  
 self.wfile.write(**"compilation failed. Reason:"**.\_\_add\_\_(e.args).encode())  
  
 test, result = executor.execute\_program(content)  
 **if** test:  
 self.wfile.write(**"program has been executed successfully and returned: "**.\_\_add\_\_(str(result)).encode())  
 **else**:  
 self.wfile.write(**"program execution failed "**.\_\_add\_\_(result).encode())  
  
 r = reporter.Reporter()  
 name = r.save\_program(content)  
 result = r.compare(name)  
 self.wfile.write(**"\nfiles with same content actually saved on server:\n"**.\_\_add\_\_(**"\n"**.join(result)).encode())

Klasa Server pozwala na uruchomić serwer na danym komputerze podając jego adres IP oraz port. Jeśli wywołanie metody Server((IP, PORT), MyHandler)gdzie argumentami jest adres IP, port oraz klasa MyHandler.

Klasa MyHandler odpowiedzialna jest za komunikację z połączonym klientem i odebraniem żądania od niego. Metoda \_set\_headers ustawia nagłówki odpowiedzi na żądanie. Metoda do\_PUT odpowiada za:

* Przygotowanie i wysłania komunikatu zwrotnego do klienta o odebranym pliku,
* Wywołania metody compile\_file z klasy compiler odpowiedzialnej za kompilację otrzymanego programu i wysłanie komunikatu o powodzeniu lub niepowodzeniu operacji do klienta,
* Wywołania metody execute\_program z klasy executor odpowiedzialnej za uruchomienie skompilowanego programu i powiadomienie o wyniku klienta,
* Zapisanie i porównanie otrzymanych programów przez klienta za pomocą metod save\_program oraz compare i powiadomienie użytkownika o wyniku operacji.

**client.py**

**import** http.client  
**import** time  
**from** PythonREST **import** config  
  
  
**class** Client:  
  
 IP = config.IP  
 PORT = config.PORT  
 *# file\_path = "D:\\Hello.py"* file\_path = **"D:\\longTimeProgram1.py"** file\_path2 = **"D:\\longTimeProgram2.py"** @staticmethod  
 **def** load\_file(self, file\_path):  
 *"""  
 loads file content from given path"* **:param** *file\_path: filepath to file* **:return***: file content  
 """* file = open(file\_path)  
 **return** file.read()  
  
 @staticmethod  
 **def** send\_file(self, name, content, server\_ip, PORT):  
 *"""  
 sends content to server defined in client module* **:param***:  
 name: name of file to send  
 content: content of file to send  
 server\_ip: ip of server where content should be sent  
 PORT: port where server is listening for requests* **:return***: server response  
 """* conn = http.client.HTTPConnection(server\_ip, PORT)  
 conn.request(**"PUT"**, **"/"**.\_\_add\_\_(name), content)  
 **return** conn.getresponse()  
  
 **def** run\_client(self, id):  
 *"""  
 fakes client behavior with following steps:  
 1. loads file from built-in filepath based on received id(1 or anything else)  
 2. sends content of loaded file to server defined by built-in IP and PORT variables* **:return***: None - prints received responses  
 """* time.sleep(2)  
 print(**"client started."**)  
 print(**"loading file..."**)  
 **if** id == 1:  
 content = Client.load\_file(self, file\_path=Client.file\_path)  
 **else**:  
 content = Client.load\_file(self, file\_path=Client.file\_path2)  
 print(**"file loaded. sending file..."**)  
 response = Client.send\_file(self, **"program1"**, content, Client.IP, Client.PORT)  
 print(response.status, response.reason, response.read().decode())

Klasa Client składa się z:

* Atrybutów
  + IP – przyjmująca adres IP serwera z pliku config.py
  + PORT – przyjmuje wartość portu serwera z pliku config.py
  + file\_path – przechowuje ścieżkę do pliku, który ma zostać przesłany na serwer
  + file\_path2 – przechowuje ścieżkę do pliku, który ma zostać przesłany na serwer
* Metody
  + load\_file(self, file\_path)- metoda odpowiedzialna za wczytanie pliku z podanej ścieżki
  + send\_file(self, name, content, server\_ip, PORT) – metoda tworząca połączenie między klientem a serwerem
  + run\_client(self, id)- metoda zarządzająca wysyłaniem plików

**compiler.py**

**import** py\_compile  
  
**class** Compiler:  
  
 **def** compile\_file(self, content):  
 *"""  
 tries to compile file content* **:param** *content:* **:return***:  
 compilation success: 1  
 compilation error: 0  
 """* **try**:  
 py\_compile.compile(content)  
 **return** 1, **"success"  
 except** Exception **as** e:  
 **return** 0, e.args

Klasa Compiler odpowiedzialna jest za kompilacje otrzymanego pliku. Metoda compile\_file wykonuję próbę kompilacji kiedy zakończy się powodzeniem zwraca 1 jeżeli nie zwraca wyjątek.

**executor.py**

**def** execute\_program(content):  
 *"""  
 executes given content of program.* **:param** *content: content of program to execute* **:return***: output of executed program  
 """* result = dict()  
 print(**"executing file..."**)  
 **try**:  
 exec(content, dict(), result)  
 print(**"file executed."**)  
 **return** 1, result[**"result"**]  
 **except** Exception **as** e:  
 **return** 0, str(e.args)

Funkcja odpowiedzialna za uruchomienie skompilowanego programu.

**config.py**

PORT = 8080  
IP = **"192.168.0.3"**

Plik zawiera adres IP serwera oraz jego port.

**reporter.py**

**import** os  
**from** builtins **import** staticmethod  
**import** filecmp  
  
  
**class** Reporter:  
  
 path = **"/server/programs/"  
  
 def** save\_program(self, content):  
 *"""  
 creates new file with a given content at server's path containing programs* **:param** *content: program's code* **:return***: name of saved file  
 """* self.create\_file\_directory\_if\_not\_exists(self, self.path)  
  
 files = len(os.listdir(self.path))  
 name = **"program"**.\_\_add\_\_(str(files+1).\_\_add\_\_(**".py"**))  
 file = open(self.path.\_\_add\_\_(name), **'w'**)  
 file.write(content.decode())  
 file.close()  
 **return** name  
  
 **def** compare(self, name):  
 *"""  
 compares content of file with given name from built-in directory to another file contents in this directory* **:param** *name: name of file to compare* **:return***: list of file names with same content  
 """* result = []  
 **for** i **in** (os.listdir(self.path)):  
 **if** i != name:  
 comparison = filecmp.cmp(self.path.\_\_add\_\_(i), self.path.\_\_add\_\_(name), shallow=**False**)  
 **if** comparison:  
 result.append(i)  
 **return** result  
  
 @staticmethod  
 **def** get\_file\_path(self):  
 **return** self.path  
  
 @staticmethod  
 **def** create\_file\_directory\_if\_not\_exists(self, path):  
 **if not** os.path.exists(path):  
 os.makedirs(path)  
  
 @staticmethod  
 **def** does\_directory\_exist(self, path):  
 **return** os.path.exists(path)

Klasa Reporter zawiera atrybut path zawierający ścieżkę zapisu wyniku wykonanych operacji. Metoda save\_program(self, content) zapisuje zawartość otrzymanego pliku na serwerze. Kolejna metoda compare(self, name) porównuje zapisane programy i zwraca informację czy są podobne.

Metody get\_file\_path(self),create\_file\_directory\_if\_not\_exists(self, path) oraz does\_directory\_exist(self, path)są odpowiedzialne za zarządzanie utworzeniem katalogu w którym zostaną umieszczone wyniki porównań.

**Programy przesyłane do serwera**

**example.py**

**def** f():  
 **return "Hello!"**result = f()

Funkcja f() wywołana zwraca napis „Hello!”.

**fail.py**

prin(**"this fails"**)

Wypisuje napis „this fails”

**longTimeProgram.py**

**def** f():  
 **import** time  
 **for** i **in** range(0, 5):  
 time.sleep(3)  
 **return "long time program finished after 15 seconds."**result = f()

Program pozwalający na mierzenie czasu. W naszym przypadku zatrzymuje czas na 3 s pięciokrotnie zwracając po zakończeniu pętli „long time program finished after 15 seconds.”

**sum.py**

**def** sum(a, b):  
 **return** a+b  
  
result = sum(1, 2)

Program składający się z funkcji pozwalającej na sumowanie dwóch liczb.

**Sposób uruchomienia projektu**

1. Instalacja środowiska IDE – PyCharm
2. Instalacja interpretera języka python
3. Podanie adresu serwera i portu w pliku config.py
4. Podanie ścieżki do plików, które mają zostać wysłane w module clineta

file\_path = **"D:\\longTimeProgram1.py"**file\_path2 = **"D:\\longTimeProgram2.py"**

1. Uruchomienie serwera za pomocą wiersza poleceń Windows:

python server.py

1. Uruchomienie serwera za pomocą wiersza poleceń Windows:

python client.py

**Analiza porównawcza implementacji**

**Bibliografia**

UML:

http://zasoby.open.agh.edu.pl/~09sbfraczek/

REST

<http://www.yarpo.pl/2012/07/29/rest-ciekawszy-sposob-na-komunikacje-client-server/>

<https://www.fullstackpython.com/api-creation.html>

<https://techietweak.wordpress.com/2015/03/30/http-restful-api-with-python-requests-library/>

<https://developers.google.com/drive/v3/web/quickstart/python>

<http://documentation.commvault.com/commvault/v10/article?p=features/rest_api/rest_api_getting_started_python.htm>