Zarządanie Danymi

Flask REST API (część 1) – przykłady aplikacji

dr inż. Łukasz Piątek

Katedra Sztucznej Inteligencji

Wyższa Szkoła Informatyki i Zarządzania z siedzibą w Rzeszowie

Agenda

- **Implementacja przykładowych aplikacji typu *Flask REST API*:**
  - **prosta (podstawowa) aplikacja *Flask REST API*,**
  - **aplikacja umożliwiająca obsługę *CRUD* (ang., *Create*, *Read*, *Update* oraz/lub *Delete*),**
  - **aplikacja z autoryzacją dostępu do danych (z *Flask-JWT*)**
- **Wszystkie w/w aplikacje zaimplementowano w:**
  - **środowisku *Microsoft VS Studio Code*,**
  - **w połączeniu z wykorzystaniem programu *Postman*.**

Podstawowa aplikacja *Flask REST API* (1/3)

- ***Microsoft VS Studio Code*:**

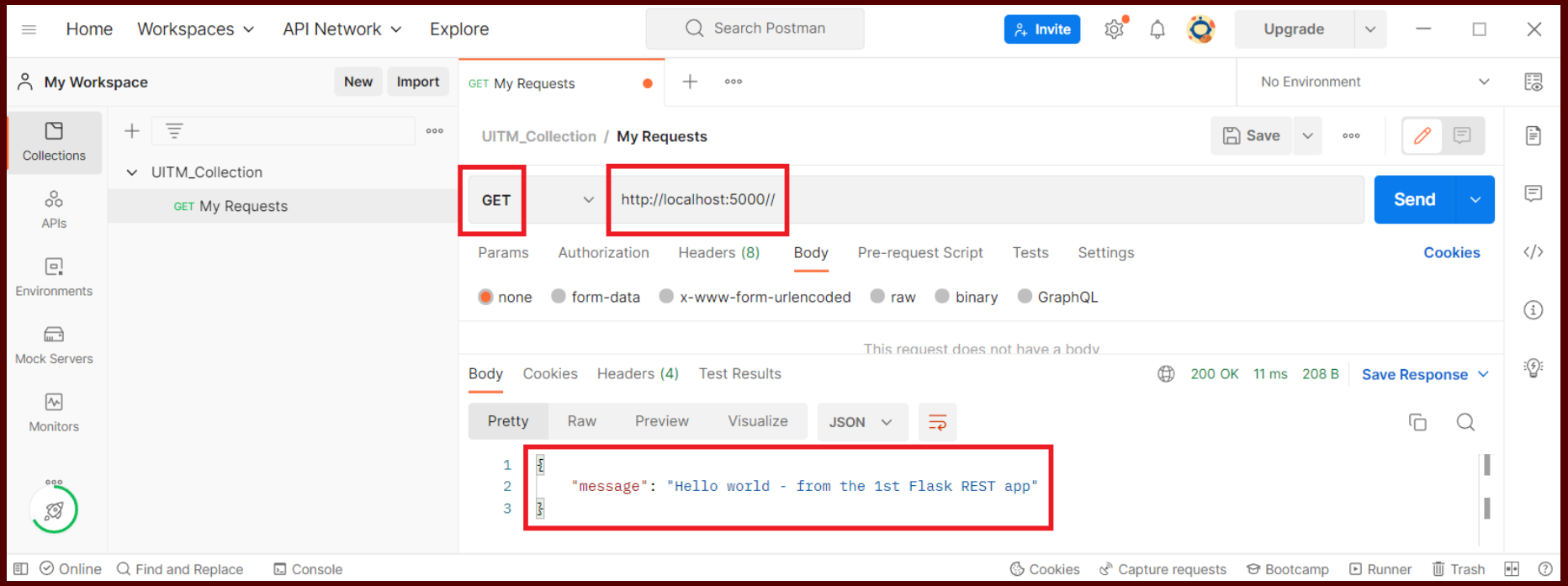
01_simple_api.py

```python
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class HelloWorld(Resource):
    def get(self):
        return {'message': 'Hello world - from the 1st Flask REST app'}

api.add_resource(HelloWorld, '/')

if __name__ == '__main__':
    app.run(debug=True)

```

Rys.1.A. **Implementacja prostej aplikacji *Flask REST API* w środowisku *Microsoft VS Studio Code***

Podstawowa aplikacja *Flask REST API* (1/3)

- *Postman*:



Rys.1.B. **Wysłanie zapytania (oraz odpowiedź) z aplikacji klienckiej** ***Postman***

Aplikacja *Flask REST CRUD API* (2/3)

- *Microsoft VS Studio Code* (część 1):

02_crud_api.py

```python
from flask import Flask, request
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

# We should call to the database!
# However, right now its just a list of dictionaries
# students = [{'name':'Andrusiv'},{name:'Cyran'},.....]
# Keep in mind, its in memory, it clears with every restart!
students = []
```

Rys.2.A.1. **Implementacja aplikacji *Flask REST CRUD API* w środowisku *Microsoft VS Studio Code* (1)**

Aplikacja *Flask REST CRUD API* (2/3)

- *Microsoft VS Studio Code* (część 2):

```python
class StudentNames(Resource):
    def get(self, name):
        print(students)

        # Cycle through list for students
        for stud in students:
            if stud['name'] == name:
                return stud

        # If it's request of a student not yet in the students list
        return {'name': None}, 404

    def post(self, name):
        # Add  the dictionary to list
        stud = {'name': name}
        students.append(stud)
        # Then return it back
        print(students)
        return stud

    def delete(self,name):

        # Cycle through list for students
        for ind, stud in enumerate(students):
            if stud['name'] == name:
                # don't really need to save this
                delted_stud = students.pop(ind)
                return {'note':'delete successful'}
```

Rys.2.A.2. **Implementacja aplikacji *Flask REST CRUD API* w środowisku *Microsoft VS Studio Code* (2)**

Aplikacja *Flask REST CRUD API* (2/3)

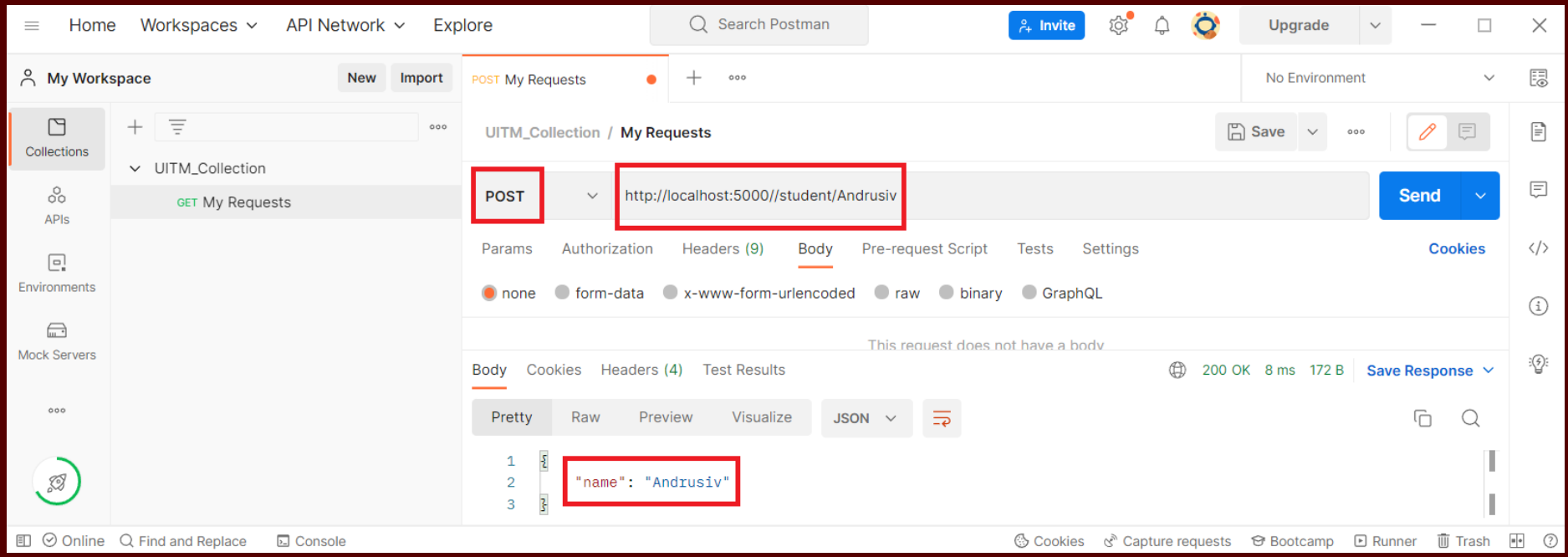
- *Microsoft VS Studio Code* (część 3):

```
42  class AllNames(Resource):
43
44      def get(self):
45          # return all STUDENTS
46          return {'students': students}
47
48
49  api.add_resource(StudentNames, '/student/<string:name>')
50  api.add_resource(AllNames,'/students')
51
52  if __name__ == '__main__':
53      app.run(debug=True)
54
```

Rys.2.A.3. **Implementacja aplikacji *Flask REST CRUD API* w środowisku *Microsoft VS Studio Code* (3)**

Aplikacja *Flask REST CRUD API* (2/3)

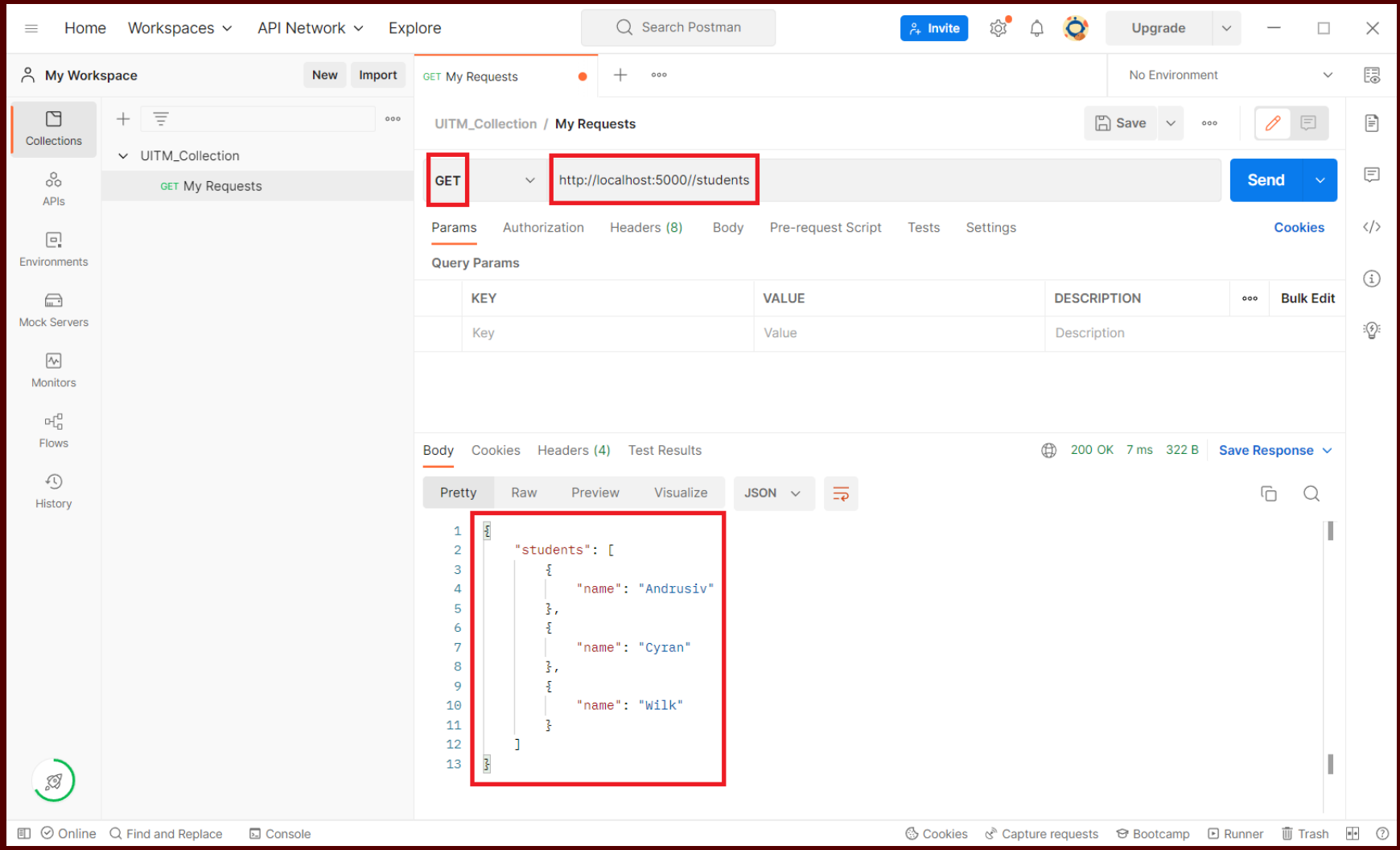
- ***Postman*** **(1):**



Rys.2.B.1. **Wysłanie zapytania (oraz odpowiedź) z aplikacji klienckiej *Postman* – *dodanie* nowego obiektu (*studenta*)**

Aplikacja *Flask REST CRUD API* (2/3)

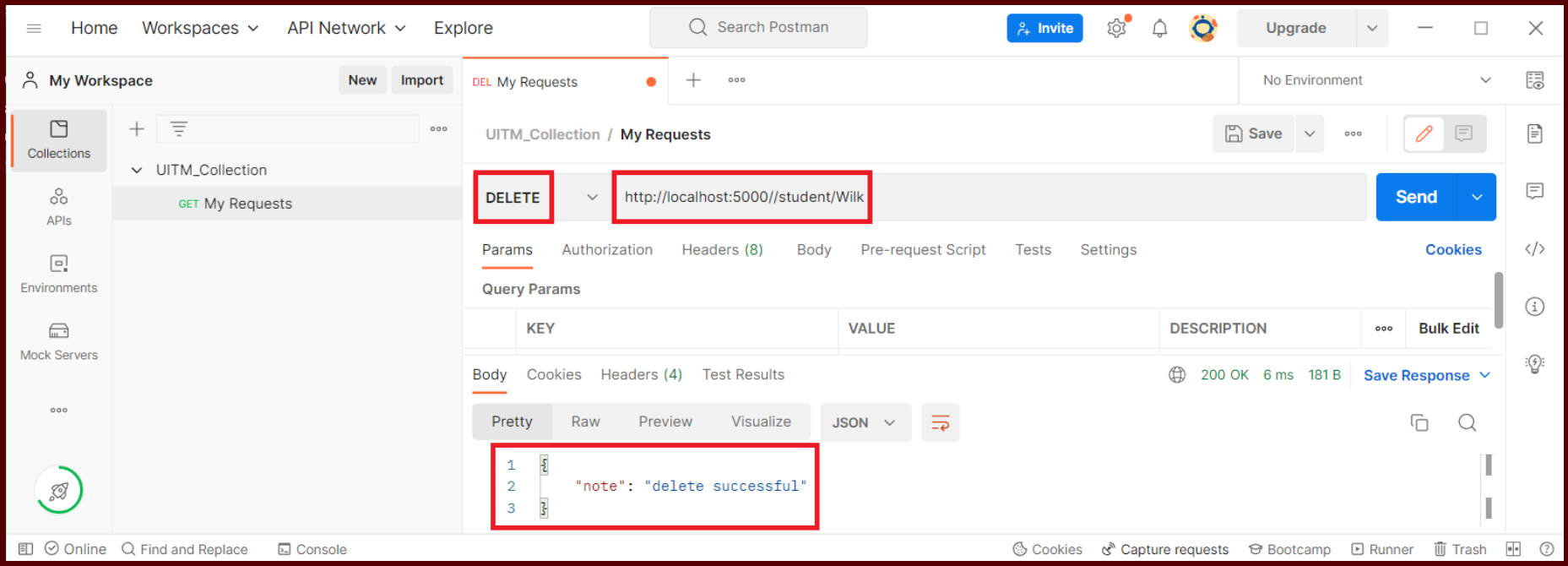
- ***Postman*** **(2):**



Rys.2.B.2. **Wysłanie zapytania (oraz odpowiedź) z aplikacji klienckiej *Postman* – wyświetlenie listy *wszystkich* studentów**

Aplikacja *Flask REST CRUD API* (2/3)

- ***Postman*** **(3):**



Rys.2.B.3. **Wysłanie zapytania (oraz odpowiedź) z aplikacji klienckiej *Postman* – *usuwanie* wybranego obiektu (*studenta*)**

Autoryzacja *Flask-JWT* (3/3)

- *Microsoft VS Studio Code* (1):

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

PS C:\Workspace\UITM_2022_2023> pip install Flask-JWT
```

Rys.3.A. **Instalacja pakietu** ***Flask-JWT***

user.py

03_flask_JWT_authorisation > user.py

```python
# user.py

class User():

    def __init__(self,id,username,password):

        self.id = id
        self.username = username
        self.password = password

    def __str__(self):
        return f"User ID: {self.id}"

```

Rys.3.A.1. **Implementacja aplikacji w** ***MS VS Studio Code*** **(plik** ***user.py*****)**

Autoryzacja *Flask-JWT* (3/3)

- *Microsoft VS Studio Code* (2):

secure_check.py

03_flask_JWT_authorisation > secure_check.py

```python
# secure_check.py

# In the real world/solution this should be a database table!
from user import User

users = [ User(1,'Piątek','my_secret_pass'),
          User(2,'Barabash','another_pass')]

username_table = {u.username: u for u in users}
userid_table = {u.id: u for u in users}


def authenticate(username, password):
    user = username_table.get(username, None)
    if user and password == user.password:
        return user

def identity(payload):
    user_id = payload['identity']
    return userid_table.get(user_id, None)

```

Rys.3.A.2. **Implementacja aplikacji w** ***MS VS Studio Code*** **(plik** ***secure_check.py*****)**

Autoryzacja *Flask-JWT* (3/3)

- *Microsoft VS Studio Code* (3.1):

auth_api.py

03_flask_JWT_authorisation > auth_api.py

```python
from flask import Flask, request
from flask_restful import Resource, Api
from secure_check import authenticate,identity
from flask_jwt import JWT ,jwt_required

app = Flask(__name__)
app.config['SECRET_KEY'] = 'mysecretkey'
api = Api(app)

jwt = JWT(app, authenticate, identity)

# We should call to the database!
# However, right now its just a list of dictionaries
# students = [{'name':'Andrusiv'},{name:'Cyran'},.....]
# Keep in mind, its in memory, it clears with every restart!
students = []
```

Rys.3.A.3.1. **Implementacja aplikacji w *MS VS Studio Code* (plik *auth_api.py*)**

Autoryzacja *Flask-JWT* (3/3)

- *Microsoft VS Studio Code* (3.2):

```
18  class StudentNames(Resource):
19      def get(self, name):
20          print(students)
21
22          # Cycle through list for students
23          for stud in students:
24              if stud['name'] == name:
25                  return stud
26
27          # If it's request of a student not yet in the students list
28          return {'name': None}, 404
29
30      def post(self, name):
31          # Add  the dictionary to list
32          stud = {'name': name}
33          students.append(stud)
34          # Then return it back
35          print(students)
36          return stud
37
38      def delete(self,name):
39
40          # Cycle through list for students
41          for ind, stud in enumerate(students):
42              if stud['name'] == name:
43                  # don't really need to save this
44                  delted_stud = students.pop(ind)
45                  return {'note':'delete successful'}
```

Rys.3.A.3.2. **Implementacja aplikacji w *MS VS Studio Code* (plik *auth_api.py*)**

Autoryzacja *Flask-JWT* (3/3)

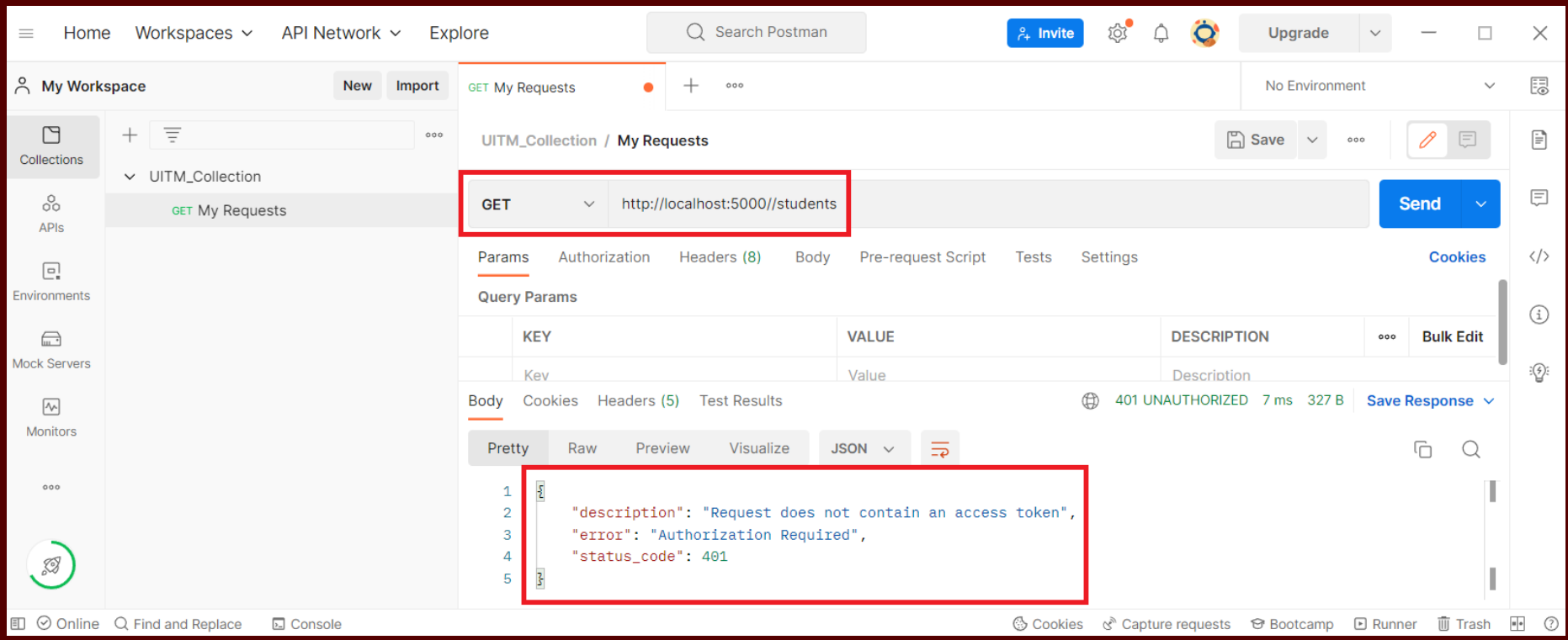
- *Microsoft VS Studio Code* (3.3):

```
47  class AllNames(Resource):
48
49      @jwt_required()
50      def get(self):
51          # return all STUDENTS
52          return {'students': students}
53
54
55  api.add_resource(StudentNames, '/student/<string:name>')
56  api.add_resource(AllNames,'/students')
57
58  if __name__ == '__main__':
59      app.run(debug=True)
60
```

Rys.3.A.3.3. **Implementacja aplikacji w *MS VS Studio Code* (plik *auth_api.py*)**

Autoryzacja *Flask-JWT* (3/3)

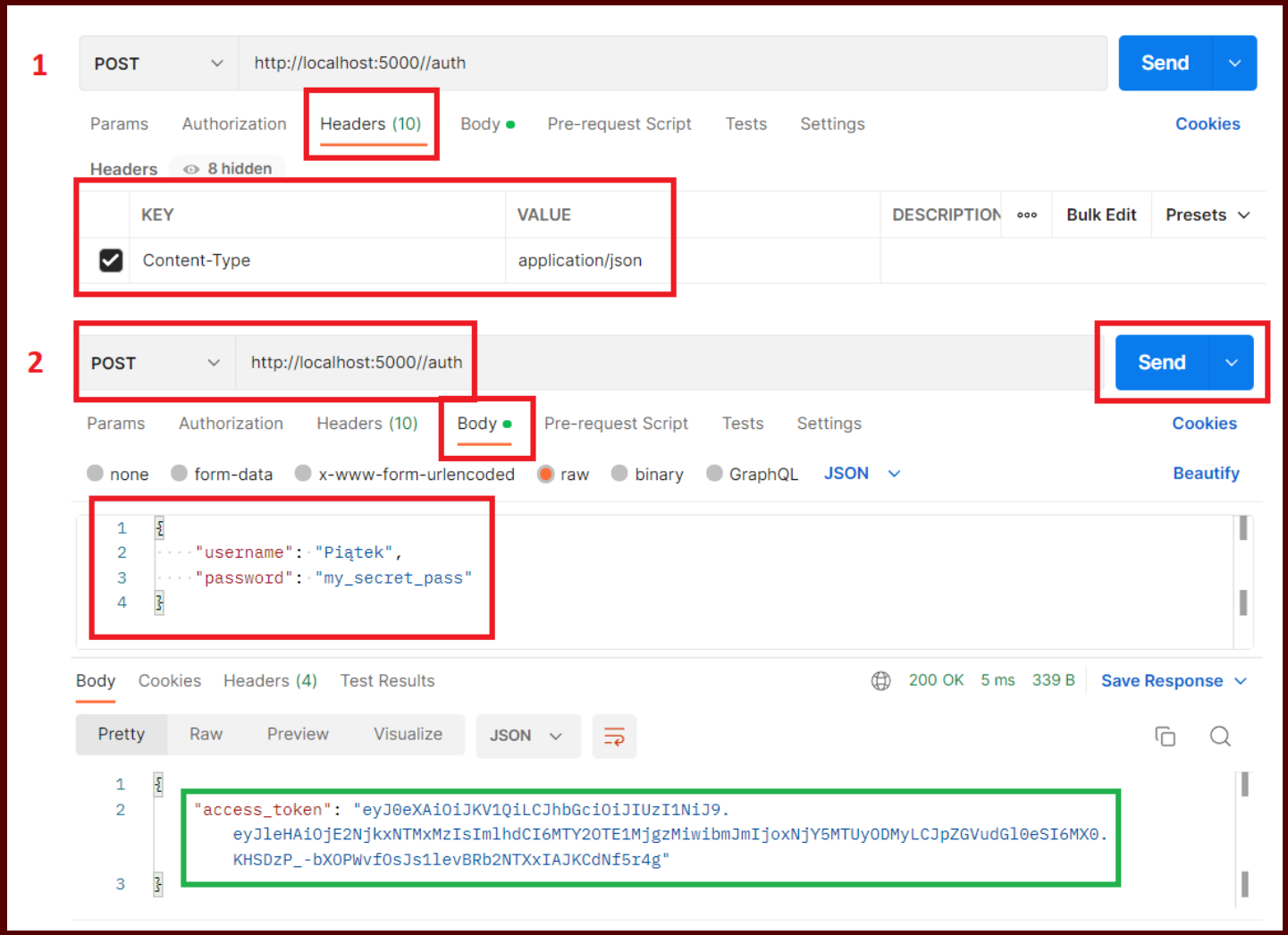
- ***Postman*** **(1):**



Rys.3.B.1. Próba wyświetlenia listy studentów bez autoryzowanego dostępu w aplikacji klienckiej Postman

Autoryzacja *Flask-JWT* (3/3)

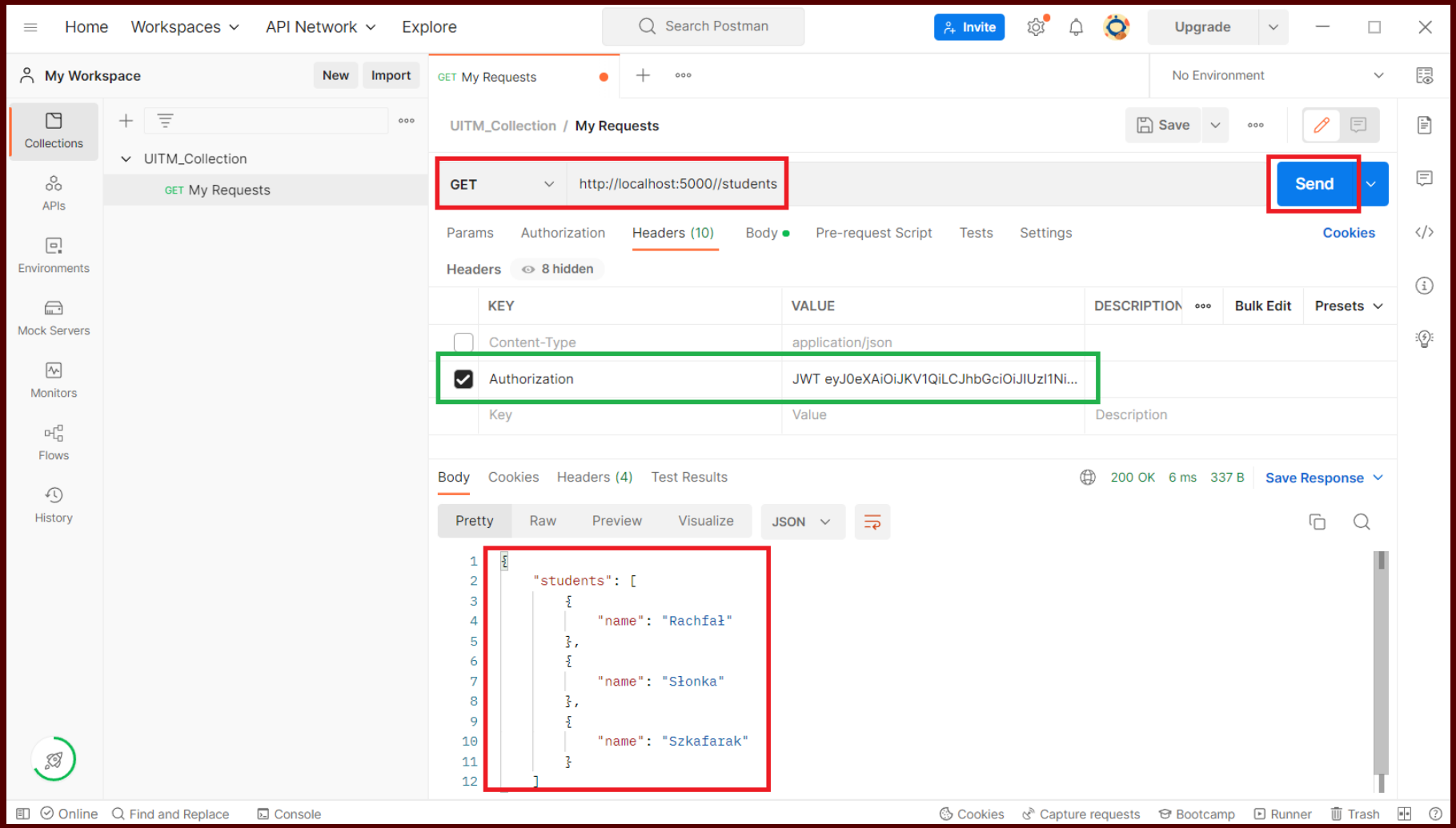
- *Postman* (2):



Rys.3.B.2. **Autoryzacja dostępu dla wybranego użytkownika (*Piątek*) w aplikacji klienckiej *Postman***

Autoryzacja *Flask-JWT* (3/3)

- ***Postman*** **(3):**



Rys.3.B.3. **Wyświetlenie listy studentów po uzyskaniu autoryzacji dla wybranego użytkownika (*Piątek*) w aplikacji klienckiej *Postman***

Wyższa Szkoła Informatyki i Zarządzania
ul. Sucharskiego 2, 35-225 Rzeszów, Polska

Dziękuję za uwagę…