



Politechnika Krakowska
Wydział Informatyki i Telekomunikacji

Studia Stacjonarne

Sprawozdanie z przedmiotu:

Zaawansowane Techniki Programowania

Laboratorium nr 3

Wykonał
Jan Kopeć

1. Tworzenie podstawowego kontrolera

Opis zadania:

Celem zadania jest zaimplementowanie podstawowego kontrolera do obsługi żądań HTTP w aplikacji backendowej. Należy stworzyć kontroler o nazwie UserController, który obsłuży operacje dodawania użytkowników oraz zwracania listy użytkowników. Wymagana jest walidacja danych w przypadku dodawania użytkownika.

Implementacja

Technologia i narzędzia: Python, Flask, Flasgger

Fragmenty kodu:

Lista użytkowników:

```
users = [
    {"id": "1", "name": "Jan Kopeć", "email": "jan.kopec@gmail.com"},
    {"id": "2", "name": "Elon Musk", "email": "elon.musk@x.com"},
    {"id": "3", "name": "Olaf Papuga", "email": "papuga@gmail.com"},
    {"id": "4", "name": "Julia Śliwa", "email": "julia.sliwa@gmail.com"},
    {"id": "5", "name": "Adam Kojder", "email": "adam.kojder@gmail.com"},
    {"id": "6", "name": "Maksymilian Nowak", "email": "maks.nowak@gmail.com"},
    {"id": "7", "name": "Piotr Kowalski", "email": "piotr.kowalski@gmail.com"},
    {"id": "8", "name": "Magdalena Wiśniewska", "email": "magdalena.wisniewska@gmail.com"},
    {"id": "9", "name": "Robert Lewandowski", "email": "robert.lewandowski@gmail.com"},
    {"id": "10", "name": "Agnieszka Zielińska", "email": "agnieszka.zielinska@gmail.com"}
]
```

Endpoint GET /users zwracający listę użytkowników:

```
# Endpoint GET /users
@app.route("/users", methods=["GET"])
def get_users():
    limit = request.args.get("limit", default=5, type=int)
    offset = request.args.get("offset", default=0, type=int)
    name_filter = request.args.get("name", default="", type=str)

    # Filtrowanie po nazwie
    filtered_users = [user for user in users if name_filter.lower() in user["name"].lower()]

    # Paginacja
    paginated_users = filtered_users[offset:offset+limit]

    return jsonify(paginated_users)
```

Endpoint POST /users pozwalający dodać nowego użytkownika z walidacją danych:

```
# Endpoint POST /users
@app.route("/users", methods=["POST"])
def create_user():
    data = request.get_json()

    # Walidacja danych
    if not data.get("name") or not is_valid_name(data["name"]):
        abort(400, description="Imię musi mieć co najmniej 3 znaki")
    if not data.get("email") or not is_valid_email(data["email"]):
        abort(400, description="Nieprawidłowy adres e-mail")

    # Tworzenie nowego ID
    new_id = generate_unique_id()
    user = {"id": new_id, "name": data["name"], "email": data["email"]}
    users.append(user)

    return jsonify(user), 201
```

Do wszystkich endpointów aplikacji dodano dokumentację przy użyciu narzędzia **Flasgger**, które integruje się z Swagger UI. Dzięki temu, każdy z punktów końcowych API został szczegółowo opisany, co umożliwia łatwe testowanie aplikacji w sposób interaktywny.

```
# Endpoint POST /users
@app.route("/users", methods=["POST"])
def create_user():
    """
    Endpoint do tworzenia nowego użytkownika.
    ---
    parameters:
      - name: body
        in: body
        required: true
        schema:
          type: object
          properties:
            name:
              type: string
              example: "Olga Nowak"
            email:
              type: string
              example: "olga.nowak@gmail.com"
    responses:
      201:
        description: Utworzony użytkownik
        schema:
          id: User
          properties:
            id:
              type: string
            name:
              type: string
            email:
              type: string
    """
    data = request.get_json()
```

Przykładowe wyniki:

Poprawnie wprowadzone dane skutkują utworzeniem nowego użytkownika:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- Endpoint:** /users (Description: Endpoint do tworzenia nowego użytkownika.)
- Parameters:** None
- Body:** A JSON object:

```
{  "email": "olga.nowak@gmail.com",  "name": "Olga Nowak"}
```
- Parameter content type:** application/json
- Execute:** Button to send the request.
- Responses:** Section showing the response details.
 - Response content type:** application/json
 - Code:** 201
 - Response body:**

```
{  "email": "olga.nowak@gmail.com",  "id": "39629",  "name": "Olga Nowak"}
```

W przypadku niepoprawnego imienia zwracany jest błąd wraz z komunikatem:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- Endpoint:** /users
- Body:** A JSON object:

```
{  "email": "olga.nowak@gmail.com",  "name": "Ol"}"
```
- Response:** 400 Error: BAD REQUEST
- Response body:**

```
{  "error": "Imię musi mieć co najmniej 3 znaki",  "status": 400}
```

W przypadku niepoprawnego adresu e-mail - braku znaku @ zwracany jest błąd wraz z komunikatem:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- Endpoint:** /users
- Body:** A JSON object:

```
{  "email": "olga.nowakgmail.com",  "name": "Olga Nowak"}"
```
- Response:** 400 Error: BAD REQUEST
- Response body:**

```
{  "error": "Nieprawidłowy adres e-mail",  "status": 400}
```

2. Obsługa parametrów i błędów w kontrolerze

Opis zadania:

Należy dodać endpoint GET /users/{id}, który zwróci szczegóły użytkownika na podstawie jego ID. W przypadku, gdy użytkownik o podanym ID nie istnieje, należy zwrócić odpowiedź 404 Not Found.

Fragment kodu:

Endpoint GET /users/{id} zwracający szczegóły użytkownika po ID:

```
@app.route("/users/<user_id>", methods=["GET"])
def get_user(user_id):
    user = next((u for u in users if u["id"] == user_id), None)
    if user is None:
        abort(404, description="Użytkownik nie znaleziony")
    return jsonify(user)
```

Przykładowe wyniki:

Zwracanie szczegółów użytkownika o ID 1:

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** /users/{user_id} Endpoint do pobierania użytkownika po ID.
- Parameters:** A table with one entry: user_id (string/path) with value 1.
- Buttons:** Execute (highlighted in blue), Clear.
- Responses:** A section with a dropdown for 'Response content type' set to 'application/json'.
- Curl:** curl -X GET "http://127.0.0.1:5000/users/1" -H "accept: application/json"
- Request URL:** http://127.0.0.1:5000/users/1
- Server response:** Code 200, Details.
- Response body:** A JSON object: {"email": "jan.kopec@gmail.com", "id": "1", "name": "Jan Kopec"}. There are icons for 'Copy' and 'Download' next to the body.

Gdy użytkownik o podanym ID nie istnieje zwracana jest odpowiedź z kodem 404 NOT FOUND:

The screenshot shows a REST client interface with the following details:

- Curl:** curl -X GET "http://127.0.0.1:5000/users/999" -H "accept: application/json"
- Request URL:** http://127.0.0.1:5000/users/999
- Server response:** Code 404, Details.
- Response body:** A JSON object: {"error": "Użytkownik nie znaleziony", "status": 404}. There are icons for 'Copy' and 'Download' next to the body.

3. Aktualizacja i usuwanie użytkownika

Opis zadania:

W ramach tego zadania należało dodać obsługę aktualizacji i usuwania użytkowników.

Fragmenty kodu:

PUT /users/{id}:

```
@app.route("/users/<user_id>", methods=["PUT"])
def update_user(user_id):

    user = next((u for u in users if u["id"] == user_id), None)
    if user is None:
        abort(404, description="Użytkownik nie znaleziony")

    data = request.get_json()

    # Walidacja danych
    if not data.get("name") or not is_valid_name(data["name"]):
        abort(400, description="Imię musi mieć co najmniej 3 znaki")
    if not data.get("email") or not is_valid_email(data["email"]):
        abort(400, description="Nieprawidłowy adres e-mail")

    # Aktualizacja użytkownika
    user["name"] = data["name"]
    user["email"] = data["email"]

    return jsonify(user)
```

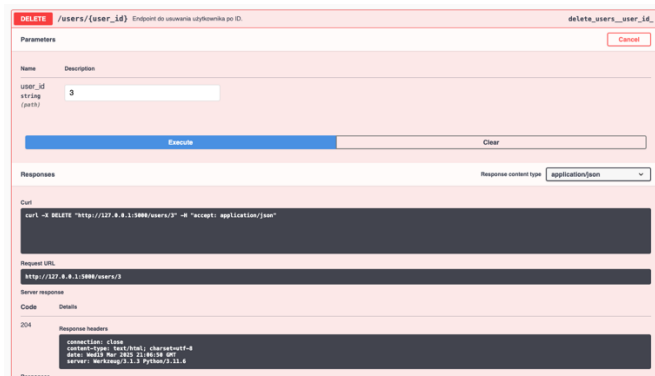
DELETE /users/{id}

```
@app.route("/users/<user_id>", methods=["DELETE"])
def delete_user(user_id):
    user = next((u for u in users if u["id"] == user_id), None)
    if user is None:
        abort(404, description="Użytkownik nie znaleziony")

    users.remove(user)
    return '', 204 # No Content
```

Przykładowe wyniki:

W przypadku usunięcia użytkownika o istniejącym ID zwracany jest kod 204:



Użytkownik jest aktualizowany, gdy wprowadzone dane są poprawne:

The screenshot shows a REST client interface for a PUT request to the endpoint `/users/{user_id}`. The `user_id` parameter is set to `1`. The request body is a JSON object: `{ "email": "janekp727@p.pl", "id": "1", "name": "Jan Koj" }`. The parameter content type is set to `application/json`. The response status is `200`, and the response body is the same JSON object as the request body.

W przypadku wprowadzania zbyt krótkiej nazwy użytkownika zwracany jest błąd:

The screenshot shows a REST client interface for a PUT request to the endpoint `/users/1`. The request body is a JSON object: `{ "email": "janekp727@p.pl", "id": "1", "name": "Ja" }`. The response status is `400` with the message `Error: BAD REQUEST`. The response body is a JSON object: `{ "error": "Imię musi mieć co najmniej 3 znaki", "status": 400 }`.

4. Paginacja i filtrowanie w GET /users

Opis: Rozszerz funkcjonalność *endpointu GET /users*, aby obsługiwał paginację i filtrowanie.

Fragment kodu:

```
limit = request.args.get("limit", default=10, type=int)
offset = request.args.get("offset", default=0, type=int)
name_filter = request.args.get("name", default="", type=str)

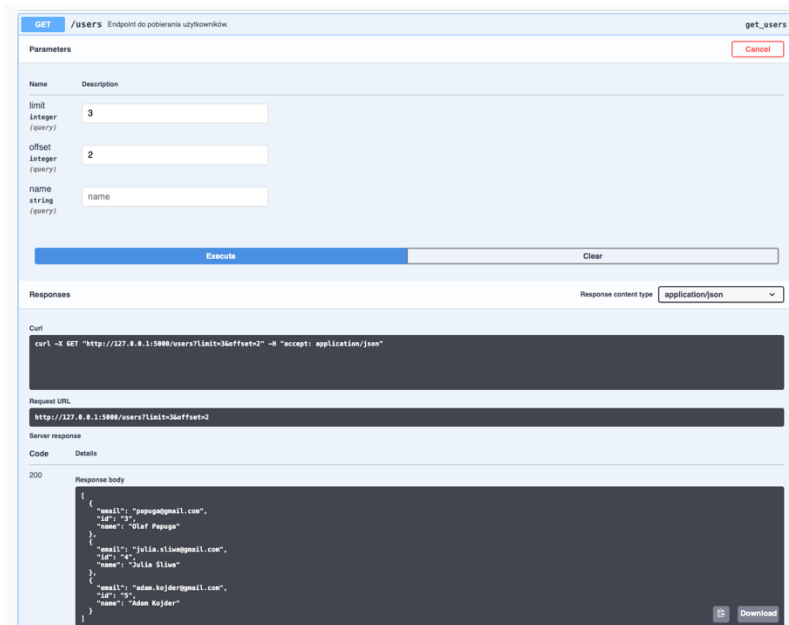
# Filtrowanie po nazwie
filtered_users = [user for user in users if name_filter.lower() in user["name"].lower()]

# Paginacja
paginated_users = filtered_users[offset:offset+limit]

return jsonify(paginated_users)
```

Przykładowy wynik:

Dla ustawionych parametrów: limit =3, offset = 2 zwracana jest w poprawny sposób lista 3 użytkowników o ID > 2:



The screenshot shows the Swagger UI for the `get_users` endpoint. The `Parameters` section is expanded, showing the following parameters:

| Name | Description |
|--------|-----------------|
| limit | Integer (query) |
| offset | Integer (query) |
| name | String (query) |

The `limit` parameter is set to 3, and the `offset` parameter is set to 2. The `name` parameter is empty.

The `Responses` section shows the response for the `200` status code. The response body is a JSON array of three user objects:

```
{
  "email": "pawel@gmail.com",
  "id": 3,
  "name": "Paweł Papuga"
},
{
  "email": "julia.silva@gmail.com",
  "id": 4,
  "name": "Julia Silva"
},
{
  "email": "adam.kojder@gmail.com",
  "id": 5,
  "name": "Adam Kojder"
}
```

5. Wnioski

Udało się zaimplementować aplikację backendową w Pythonie przy użyciu frameworka Flask oraz przetestowano za pomocą Swagger'a. Aplikacja obsługuje żądania HTTP dla użytkowników, w tym dodawanie, pobieranie, aktualizowanie i usuwanie użytkowników. Dodatkowo zaimplementowano walidację danych oraz obsługę błędów. Aplikację można by rozszerzyć o możliwość zapisywania danych użytkowników w bazie danych, co zapewniłoby trwałość danych. Ponadto warto by było dodać logowanie błędów i lepszą obsługę wyjątków.