Rozproszone Systemy Informatyczne

Raport - Ćwiczenie 4

Paweł Kluska, 260391

Katsiaryna Ziatsikava, 245891

Została zaimplementowana klasa **Person**, w której zdefiniowaliśmy atrybuty klasy, konsruktory, gettery i settery.

```
29 usages
public class Person {
    4 usages
    private int id;
    4 usages
    private String name;
    4 usages
    private int age;
    4 usages
    private String email;

    no usages
    public Person() {
    }
    1 usage
    public Person(int id, String name, int age, String email) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.email = email;
    }
```

```
    16 usages
    public int getId() { return id; }
    no usages
    public void setId(int id) { this.id = id; }
    1 usage
    public String getName() { return name; }
    1 usage
    public void setName(String name) { this.name = name; }
    1 usage
    public int getAge() { return age; }
    1 usage
    public void setAge(int age) { this.age = age; }
    1 usage
    public String getEmail() { return email; }
    1 usage
    public void setEmail(String email) { this.email = email; }

    @Override
    public String toString() {
        return "Person{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                ", email='" + email + '\'' +
                '}';
    }
```

Został zdefiniowany interfejs **PersonRepositoty,** który zawiera opercje CRUD, plus aktualny rozmiar arraylisty oraz wykaz zawartości całej arraylisty.

```java
import com.example.server.exceptions.BadRequestEx;
import com.example.server.exceptions.PersonNotFoundEx;
import java.util.List;

2 usages  1 implementation
public interface PersonRepository {
    1 usage  1 implementation
    List<Person> getAllPersons();
    1 usage  1 implementation
    Person getPerson(int id) throws PersonNotFoundEx;
    1 usage  1 implementation
    Person updatePerson(Person person) throws PersonNotFoundEx;
    1 usage  1 implementation
    boolean deletePerson(int id) throws PersonNotFoundEx;
    1 usage  1 implementation
    Person addPerson(Person person) throws BadRequestEx;
    1 usage  1 implementation
    int countPersons();
}
```

Została stworzona klasa **PersonRepositoryImpl** implementująca interfejs **PersonRepository.** Zostały tutaj zaimplementowane wszystkie metody zdefiniowane w interfejsie, w razie sytuacji niestandardowych zostaną wyrzucone odpowiednie wyjątki.

```java
import com.example.server.exceptions.BadRequestEx;
import com.example.server.exceptions.PersonNotFoundEx;

import java.util.ArrayList;
import java.util.List;

1 usage
public class PersonRepositoryImpl implements PersonRepository {

    11 usages
    private List<Person> personList;
    1 usage
    public PersonRepositoryImpl() {
        personList = new ArrayList<Person>();
        personList.add(new Person( id: 1,  name: "Jan",  age: 20,  email: "jan@gmail.com"));
        personList.add(new Person( id: 2,  name: "Adam",  age: 25,  email: "adam@gmail.com"));
    }
    1 usage
    @Override
    public List<Person> getAllPersons() { return personList; }
```

```
1 usage
@Override
public Person getPerson(int id) throws PersonNotFoundEx {
    for (Person thePerson : personList) {
        if (thePerson.getId() == id) {
            return thePerson;
        }
    }
    throw new PersonNotFoundEx(id);
}

1 usage
@Override
public Person updatePerson(Person person) throws PersonNotFoundEx {
    for (Person thePerson : personList) {
        if (thePerson.getId() == person.getId()) {
            thePerson.setName(person.getName());
            thePerson.setAge(person.getAge());
            thePerson.setEmail(person.getEmail());
            return thePerson;
        }
    }
    throw new PersonNotFoundEx(person.getId());
}
```

```
    @Override
    public boolean deletePerson(int id) throws PersonNotFoundEx {
        for (Person thePerson : personList) {
            if (thePerson.getId() == id) {
                personList.remove(thePerson);
                return true;
            }
        }
        throw new PersonNotFoundEx(id);
    }
    1 usage
    @Override
    public Person addPerson(Person person) throws BadRequestEx {
        for (Person thePerson : personList) {
            if (thePerson.getId() == person.getId()) {
                throw new BadRequestEx(person.getId());
            }
        }
        personList.add(person);
        return person;
    }
    1 usage
    @Override
    public int countPersons() {
        return personList.size();
    }
}
```

Zdefiniowaliśmy klasę wyjątek **BadRequestEx** z adnotacją @ResponseStatus z wartością HttpStatus.BAD_REQUEST, co oznacza, że wyjątek spowoduje zwrócenie kodu HTTP 400. Klasa ma dwa konstruktory: jeden bezargumentowy, który ustawia domyślną wiadomość, a drugi, który przyjmuje id osoby i tworzy komunikat o błędzie, zawierający ten id.

```java
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

11 usages
@ResponseStatus(HttpStatus.BAD_REQUEST)
public class BadRequestEx extends RuntimeException {
    no usages
    public BadRequestEx() {
        super("The specified person does not exist");
    }
    1 usage
    public BadRequestEx(int id) {
        super(String.valueOf(id));
    }
}
```

Zdefiniowaliśmy również klasę wyjątku **PersonNotFoundEx** z adnotacją @ResponseStatus z wartością HttpStatus.NOT_FOUND, co oznacza, że wyjątek spowoduje zwrócenie kodu HTTP 404. Klasa również ma dwa konstruktory.

```java
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

18 usages
@ResponseStatus(code = HttpStatus.NOT_FOUND)
public class PersonNotFoundEx extends RuntimeException {
    no usages
    public PersonNotFoundEx() {
        super("The specified person does not exist");
    }
    3 usages
    public PersonNotFoundEx(int id) {
        super(String.valueOf(id));
    }
}
```

PersonController

Zaimplementowaliśmy klasę kontroler z metodami dla operacji CRUD oraz zainicjowaliśmy zmienną repozytorium **repository**. Klasę oznaczyliśmy adnotacją @RestController oraz @RequestMapping(“/persons”). Adnotacje te oznaczają, że mamy do czynienia z REST API oraz wszystkie metody w kontrolerze będą miały obsługiwać żądania rozpoczynające się od części “/persons”. Metody kontrolera mają adnotacje @GetMapping, @PostMapping, @PutMapping i @DeleteMapping. Każda z nich odpowiada metodzie HTTP. Metody korzystają z repozytorium do obsługi żądań.

W metodach kontrolera są budowane odpowiedzi serwera, są to odpowiedzi HTTP zawierające odpowiednie JSONy.. Korzystamy tutaj z architektury HAETOAS. Do każdego zwracanego JSONa dopisujemy linki do innych endpointów naszego serwisu. Kod kontrolera prezentuje się następująco:

```
import com.example.server.exceptions.BadRequestEx;
import com.example.server.exceptions.PersonNotFoundEx;
import net.minidev.json.JSONObject;
import org.springframework.hateoas.CollectionModel;
import org.springframework.hateoas.EntityModel;
import org.springframework.web.bind.annotation.*;

import java.util.List;

import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.methodOn;
```

```
@RestController
@RequestMapping("/persons")
public class PersonController {
    6 usages
    private final PersonRepository repository = new PersonRepositoryImpl();

    7 usages
    @GetMapping
    public CollectionModel<EntityModel<Person>> getAllPersons() {
        System.out.println("...called GET");
        List<EntityModel<Person>> persons =
                repository.getAllPersons().stream().map(person ->
                        EntityModel.of(person,
                                linkTo(methodOn(PersonController.class)
                                        .getPerson(person.getId())).withRel("get"),
                                linkTo(methodOn(PersonController.class)
                                        .deletePerson(person.getId())).withRel("delete"),
                                linkTo(methodOn(PersonController.class)
                                        .getAllPersons()).withRel("list all")
                        )
                ).toList();
        return CollectionModel.of(persons,
                linkTo(methodOn(PersonController.class)
                        .getAllPersons()).withSelfRel());
    }
```

```
5 usages
@GetMapping("/{id}")
public EntityModel<Person> getPerson(@PathVariable int id) {
    System.out.println("...called GET");
    try {
        Person person = repository.getPerson(id);
        return EntityModel.of(person,
                linkTo(methodOn(PersonController.class).getPerson(person.getId())).withSelfRel(),
                linkTo(methodOn(PersonController.class).deletePerson(person.getId())).withRel("delete"),
                linkTo(methodOn(PersonController.class).getAllPersons()).withRel("list all")
        );
    } catch (PersonNotFoundEx e) {
        System.out.println("...GET Exception");
        throw e;
    }
}
```

```
@PostMapping
public EntityModel<Person> addPerson(@RequestBody Person person) {
    try {
        System.out.println("...called POST");
        Person addedPerson = repository.addPerson(person);
        return EntityModel.of(person,
                linkTo(methodOn(PersonController.class).getPerson(addedPerson.getId())).withRel("get"),
                linkTo(methodOn(PersonController.class).deletePerson(addedPerson.getId())).withRel("delete"),
                linkTo(methodOn(PersonController.class).getAllPersons()).withRel("list all")
        );
    } catch (BadRequestEx e) {
        System.out.println("...POST Exception");
        throw e;
    }
}
```

```
@PutMapping
public EntityModel<Person> updatePerson(@RequestBody Person person) {
    System.out.println("...called PUT");
    try {
        Person updatedPerson = repository.updatePerson(person);
        return EntityModel.of(person,
                linkTo(methodOn(PersonController.class).getPerson(updatedPerson.getId())).withRel("get"),
                linkTo(methodOn(PersonController.class).deletePerson(updatedPerson.getId())).withRel("delete"),
                linkTo(methodOn(PersonController.class).getAllPersons()).withRel("list all")
        );
    } catch (PersonNotFoundEx | BadRequestEx e) {
        System.out.println("...PUT Exception");
        throw e;
    }
}
```

```java
@DeleteMapping("/{id}")
public EntityModel<JSONObject> deletePerson(@PathVariable int id) {
    System.out.println("...called DELETE");
    try {
        boolean deleted = repository.deletePerson(id);
        JSONObject jsonObject = new JSONObject();
        jsonObject.put("isDeleted",deleted );

        return EntityModel.of(jsonObject,
                linkTo(methodOn(PersonController.class).getPerson(id)).withRel("get"),
                linkTo(methodOn(PersonController.class).deletePerson(id)).withRel("delete"),
                linkTo(methodOn(PersonController.class).getAllPersons()).withRel("list all")
        );
    } catch (PersonNotFoundEx e) {
        System.out.println("...DELETE Exception");
        throw e;
    }
}
```

```java
@GetMapping("/count")
public EntityModel<JSONObject> countPersons() {
    System.out.println("...called GET");
    int count = repository.countPersons();
    JSONObject jsonObject = new JSONObject();
    jsonObject.put("count", count);

    return EntityModel.of(jsonObject,
            linkTo(methodOn(PersonController.class).countPersons()).withSelfRel(),
            linkTo(methodOn(PersonController.class).getAllPersons()).withRel("list all")
    );
}
```

FaultController

Zdefiniowaliśmy kontroler do obsługi wyjątków, który pozwala przechwytywać rzucane wyjątki i odpowiednio definiować odpowiedzi. Klasę oznaczyliśmy adnotacją @ControllerAdvice. Zdefiniowaliśmy metody-handlery dla wyjątków i dodaliśmy adnotacje @ResponseBody, @ExceptionHandler i @ResponseStatus.

```java
import com.example.server.exceptions.BadRequestEx;
import com.example.server.exceptions.PersonNotFoundEx;
import org.springframework.hateoas.MediaTypes;
import org.springframework.hateoas.mediatype.problem.Problem;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
```

```java
@ControllerAdvice
public class FaultController {

    no usages
    @ResponseBody
    @ExceptionHandler(PersonNotFoundEx.class)
    @ResponseStatus(value = HttpStatus.NOT_FOUND)
    ResponseEntity<?> PNFEHandler(PersonNotFoundEx e) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND)
                .header(HttpHeaders.CONTENT_TYPE, MediaTypes.HTTP_PROBLEM_DETAILS_JSON_VALUE)
                .body(Problem.create().withStatus(HttpStatus.NOT_FOUND)
                        .withTitle(HttpStatus.NOT_FOUND.name())
                        .withDetail("The person of ID=" + e.getMessage() + " DOES NOT EXIST"));

    }
```

```java
@ResponseBody
@ExceptionHandler(BadRequestEx.class)
@ResponseStatus(value = HttpStatus.BAD_REQUEST)
ResponseEntity<?> BREHandler(BadRequestEx e) {
    return ResponseEntity.status(HttpStatus.BAD_REQUEST)
            .header(HttpHeaders.CONTENT_TYPE, MediaTypes.HTTP_PROBLEM_DETAILS_JSON_VALUE)
            .body(Problem.create().withStatus(HttpStatus.BAD_REQUEST)
                    .withTitle(HttpStatus.BAD_REQUEST.name())
                    .withDetail("The person of ID=" + e.getMessage() + " ALREADY EXISTS"));
}
```

Main

```java
import com.example.MyDate;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

1 usage
@SpringBootApplication
public class RsiLab9Application implements CommandLineRunner {

    no usages
    public static void main(String[] args) {
        SpringApplication.run(RsiLab9Application.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        MyDate.info();
    }
}
```

Klient

Został też zaimplementowany prosty klient, który podłącza się do naszego api i wykonuje wszystkie zdefiniowane wcześniej metody. Do podłączenia się do serwera wykorzystaliśmy obiekt klasy RestTemplate. Z powodów małych problemów z odczytywaniem wartości ze zwróconych Jsonów wykorzystaliśmy również klasę ObjectMapper, która pozwala zamieniać Stringi zawierające strukturę Jsona na obiekty Javy POJO.

Menu główne prezentuje się następująco:

```java
public class Client {
    public static void main(String[] args) throws JsonProcessingException {
        MyData.info();
        RestTemplate restTemplate = new RestTemplate();
        Scanner scanner = new Scanner(System.in);
        ObjectMapper objectMapper = new ObjectMapper();

        int option = 0;

        while (option != 7) {
            System.out.println("1. Wyświetl wszystkich");
            System.out.println("2. Wyświetl użytkownika o podanym id");
            System.out.println("3. Dodaj użytkownika");
            System.out.println("4. Zaktualizuj użytkownika");
            System.out.println("5. Usuń użytkownika");
            System.out.println("6. Policz użytkowników");
            System.out.println("7. Wyjdź");
```

```java
try {

    System.out.println("Wybierz opcję:");
    option = scanner.nextInt();

    switch (option) {
        case 1:
            String jsonString = restTemplate.getForObject( url: "http://localhost:8080/persons", String.class);
            JsonNode jsonNode = objectMapper.readTree(jsonString);
            JsonNode rootNode = jsonNode.get("_embedded");
            if (rootNode == null) {
                System.out.println("Brak użytkowników");
                break;
            }
            JsonNode personsNode = rootNode.get("personList");
            List<Person> personList = objectMapper.readValue(personsNode.toString(),
                    new TypeReference<List<Person>>() {
                    });
            personList.forEach(System.out::println);
            break;
        case 2:
            System.out.println("Podaj id użytkownika");
            int id = scanner.nextInt();
            Person person = restTemplate.getForObject( url: "http://localhost:8080/persons/" + id, Person.class);
            System.out.println(person);
            break;
```

```java
        case 3:
            System.out.println("Podaj id użytkownika");
            int id2 = scanner.nextInt();
            System.out.println("Podaj imię");
            String name = scanner.next();
            System.out.println("Podaj wiek");
            int age = scanner.nextInt();
            System.out.println("Podaj email");
            String email = scanner.next();
            Person newPerson = restTemplate
                    .postForObject( url: "http://localhost:8080/persons",
                            new Person(id2, name, age, email), Person.class);
            System.out.println(newPerson);
            break;
        case 4:
            System.out.println("Podaj id użytkownika");
            int id3 = scanner.nextInt();
            System.out.println("Podaj imię");
            String name2 = scanner.next();
            System.out.println("Podaj wiek");
            int age2 = scanner.nextInt();
            System.out.println("Podaj email");
            String email2 = scanner.next();

            HttpEntity<Person> requestEntity = new HttpEntity<>(new Person(id3, name2, age2, email2));

            Person updatedPerson = restTemplate.exchange( url: "http://localhost:8080/persons",
                    HttpMethod.PUT,
                    requestEntity,
                    Person.class).getBody();
            System.out.println(updatedPerson);
            break;
```

```
        case 5:
            System.out.println("Podaj id użytkownika");
            int id4 = scanner.nextInt();
            restTemplate.exchange( url: "http://localhost:8080/persons/" + id4,
                    HttpMethod.DELETE,
                    requestEntity: null, String.class);

            System.out.println("Deleted");
            break;
        case 6:
            String jsonString2 = restTemplate.getForObject( url: "http://localhost:8080/persons/count", String.class);
            JsonNode jsonNode1 = objectMapper.readTree(jsonString2);
            JsonNode rootNode2 = jsonNode1.get("count");
            String personCount = objectMapper.readValue(rootNode2.toString(),
                    new TypeReference<String>() {
                    });

            System.out.println("Liczba użytkowników: " + personCount);
            break;

        case 7:
            System.out.println("Wyjdź");
            break;
    }
} catch (HttpClientErrorException e) {
    System.out.println("Błąd: " + e.getMessage());
}
catch (InputMismatchException e) {
    System.out.println("Wprowadź poprawne dane");
}
```

Wszystkie możliwe błędy zostały obsłużone, jeśli jakiś się pojawi zostanie wyświetlona informacja o błędzie.

Działanie systemu

System uruchomiliśmy w konfiguracji dwu maszynowej. Działanie systemu po stronie serwera prezentuje się następująco:

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.0.6)

2023-05-09T13:49:14.593+02:00  INFO 14044 --- [           main] com.example.server.RsiLab9Application
2023-05-09T13:49:14.607+02:00  INFO 14044 --- [           main] com.example.server.RsiLab9Application
2023-05-09T13:49:20.069+02:00  INFO 14044 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2023-05-09T13:49:20.123+02:00  INFO 14044 --- [           main] o.apache.catalina.core.StandardService
2023-05-09T13:49:20.123+02:00  INFO 14044 --- [           main] o.apache.catalina.core.StandardEngine
2023-05-09T13:49:20.481+02:00  INFO 14044 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2023-05-09T13:49:20.486+02:00  INFO 14044 --- [           main] w.s.c.ServletWebServerApplicationContext
2023-05-09T13:49:22.078+02:00  INFO 14044 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2023-05-09T13:49:22.124+02:00  INFO 14044 --- [           main] com.example.server.RsiLab9Application
Katsiaryna Ziatsikava - 245891
Paweł Kluska - 260391
9 MAY 13:49:22
19.0.2+7-44
Windows 10
192.168.56.1
```

```
...called GET
...called GET
...called POST
...called GET
...called POST
...POST Exception
...called PUT
...called GET
...called GET
...called DELETE
...called DELETE
...called DELETE
...called GET
```

Działanie klienta przezentuje się następująco:

```
Katsiaryna Ziatsikava - 245891
Paweł Kluska - 260391
9 MAY 15:42:17
19.0.2+7-44
Windows 10
192.168.56.1
1. Wyświetl wszystkich
2. Wyświetl użytkownika o podanym id
3. Dodaj użytkownika
4. Zaktualizuj użytkownika
5. Usuń użytkownika
6. Policz użytkowników
7. Wyjdź
Wybierz opcję:
```

Na początku przetestowaliśmy wyświetlenie wszystkich użytkowników

```
Wybierz opcję:
1
15:43:42.194 [main] DEBUG org.springframework.web.client.RestTemplate -- HTTP GET http://localhost:8080/persons
15:43:42.294 [main] DEBUG org.springframework.web.client.RestTemplate -- Accept=[text/plain, application/json, application/*+json, */*]
15:43:43.627 [main] DEBUG org.springframework.web.client.RestTemplate -- Response 200 OK
15:43:43.713 [main] DEBUG org.springframework.web.client.RestTemplate -- Reading to [java.lang.String] as "application/json"
Person{id=1, name='Jan', age=20, email='jan@gmail.com'}
Person{id=2, name='Adam', age=25, email='adam@gmail.com'}
```

Dalej wyświetliliśmy użytkownika po id

```
Wybierz opcję:
2
Podaj id użytkownika
2
15:45:27.687 [main] DEBUG org.springframework.web.client.RestTemplate -- HTTP GET http://localhost:8080/persons/2
15:45:27.693 [main] DEBUG org.springframework.web.client.RestTemplate -- Accept=[application/json, application/*+json]
15:45:27.723 [main] DEBUG org.springframework.web.client.RestTemplate -- Response 200 OK
15:45:27.731 [main] DEBUG org.springframework.web.client.RestTemplate -- Reading to [com.example.client.Person]
Person{id=2, name='Adam', age=25, email='adam@gmail.com'}
```

Spróbowaliśmy dodać nowego użytkownika

```
Wybierz opcję:
3
Podaj id użytkownika
3
Podaj imię
Kamil
Podaj wiek
25
Podaj email
kamil@wp.pl
15:47:30.987 [main] DEBUG org.springframework.web.client.RestTemplate -- HTTP POST http://localhost:8080/persons
15:47:30.987 [main] DEBUG org.springframework.web.client.RestTemplate -- Accept=[application/json, application/*+json]
15:47:31.019 [main] DEBUG org.springframework.web.client.RestTemplate -- Writing [Person{id=3, name='Kamil', age=25, email='kamil@wp.pl'}] w
15:47:31.111 [main] DEBUG org.springframework.web.client.RestTemplate -- Response 200 OK
15:47:31.111 [main] DEBUG org.springframework.web.client.RestTemplate -- Reading to [com.example.client.Person]
Person{id=3, name='Kamil', age=25, email='kamil@wp.pl'}
```

Wyświetliliśmy listę wszystkich użytkowników, żeby upewnić się, że nowy użytkownik został dodany.

```
Person{id=1, name='Jan', age=20, email='jan@gmail.com'}
Person{id=2, name='Adam', age=25, email='adam@gmail.com'}
Person{id=3, name='Kamil', age=25, email='kamil@wp.pl'}
```

Spróbowaliśmy dodać użytkownika o już istniejącym id, to spowodowało,że wyrzuciło wyjątek.

```
Wybierz opcję:
3
Podaj id użytkownika
3
Podaj imię
Michal
Podaj wiek
25
Podaj email
michal@wp.pl
15:49:42.742 [main] DEBUG org.springframework.web.client.RestTemplate -- HTTP POST http://localhost:8080/persons
15:49:42.742 [main] DEBUG org.springframework.web.client.RestTemplate -- Accept=[application/json, application/*+json]
15:49:42.742 [main] DEBUG org.springframework.web.client.RestTemplate -- Writing [Person{id=3, name='Michal', age=25, email='michal@wp.pl
15:49:42.875 [main] DEBUG org.springframework.web.client.RestTemplate -- Response 400 BAD_REQUEST
Błąd: 400 : "{"title":"BAD_REQUEST","status":400,"detail":"The person of ID=3 ALREADY EXISTS"}"
```

Spróbowaliśmy zaktualizować użytkownika

```
Wybierz opcję:
4
Podaj id użytkownika
3
Podaj imię
Adam
Podaj wiek
34
Podaj email
adam@wp.pl
15:52:10.578 [main] DEBUG org.springframework.web.client.RestTemplate -- HTTP PUT http://localhost:8080/persons
15:52:10.579 [main] DEBUG org.springframework.web.client.RestTemplate -- Accept=[application/json, application/*+json]
15:52:10.579 [main] DEBUG org.springframework.web.client.RestTemplate -- Writing [Person{id=3, name='Adam', age=34, email='adam@wp.pl'}]
15:52:10.590 [main] DEBUG org.springframework.web.client.RestTemplate -- Response 200 OK
15:52:10.591 [main] DEBUG org.springframework.web.client.RestTemplate -- Reading to [com.example.client.Person]
Person{id=3, name='Adam', age=34, email='adam@wp.pl'}
```

```
Person{id=1, name='Jan', age=20, email='jan@gmail.com'}
Person{id=2, name='Adam', age=25, email='adam@gmail.com'}
Person{id=3, name='Adam', age=34, email='adam@wp.pl'}
```

Wyświetliliśmy liczbę użytkowników.

```
Wybierz opcję:
6
15:55:06.584 [main] DEBUG org.springframework.web.client.RestTemplate -- HTTP GET http://localhost:8080/persons/count
15:55:06.585 [main] DEBUG org.springframework.web.client.RestTemplate -- Accept=[text/plain, application/json, application/*+json, */*]
15:55:06.597 [main] DEBUG org.springframework.web.client.RestTemplate -- Response 200 OK
15:55:06.598 [main] DEBUG org.springframework.web.client.RestTemplate -- Reading to [java.lang.String] as "application/json"
Liczba użytkowników: 3
```

Dalej spróbowaliśmy usunąć wszystkich użytkowników i wyświetliliśmy wszystkich użytkowników.

```
Wybierz opcję:
1
15:56:57.749 [main] DEBUG org.springframework.web.client.RestTemplate -- HTTP GET http://localhost:8080/persons
15:56:57.749 [main] DEBUG org.springframework.web.client.RestTemplate -- Accept=[text/plain, application/json, application/*+json, */*]
15:56:57.749 [main] DEBUG org.springframework.web.client.RestTemplate -- Response 200 OK
15:56:57.765 [main] DEBUG org.springframework.web.client.RestTemplate -- Reading to [java.lang.String] as "application/json"
Brak użytkowników
```