

Marcin Czernek

Nr albumu 39924

Semestr 3 informatyka niestacjonarne

## Projekt laboratorium - Java Spring REST API – Sprawozdanie

Projekt jest restowym api w frameworku Spring. Składa się ona z dwóch modeli Pet i User oraz odpowiednich do tych klas serwisów, kontrolerów i repozytoriów. Obsługuje 4 żądania HTTP

GET, POST, PATCH i DELETE. Do przetestowania jej wykorzystam program Postman w którym będą wyświetlały się wszystkie dane. Zamiast html jest połączenie z bazą danych do komunikacji z bazą danych będzie służył Hibernate. Konfiguruję go w application.properties

```
1  spring.jpa.properties.hibernate.hbm2ddl.auto=create
2  spring.datasource.url=jdbc:h2:file:./bazaDanych
3  spring.h2.console.enabled=true
4  spring.h2.console.path=/console
5
6
7  spring.jpa.show-sql=true
8  spring.jpa.properties.hibernate.format_sql=true
9
```

Dodatkowo też konieczna jest biblioteka h2database w pom.xml umożliwia ona komunikację z bazą danych.

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

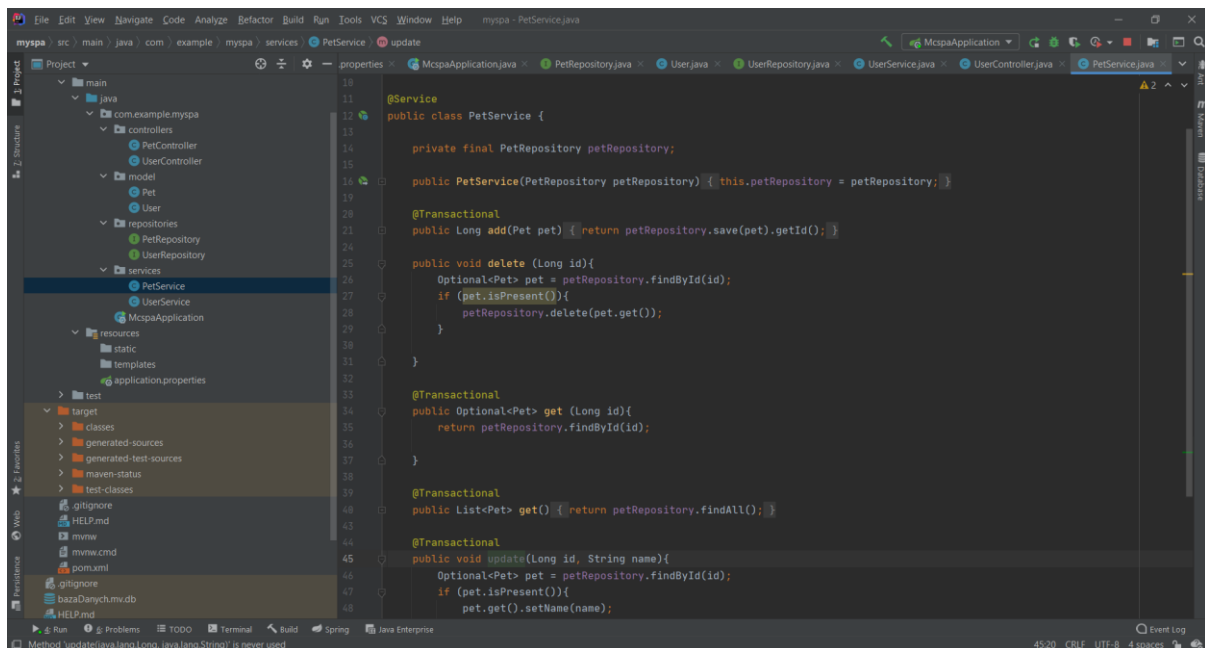
Serwisy do tych dwóch modeli (User i Pet) zarządzają danymi a repozytoria odwołują się do bazy danych.

W modelach są obiekty które będą zapisywane do bazy danych, o tym informuje anotacja Entity.

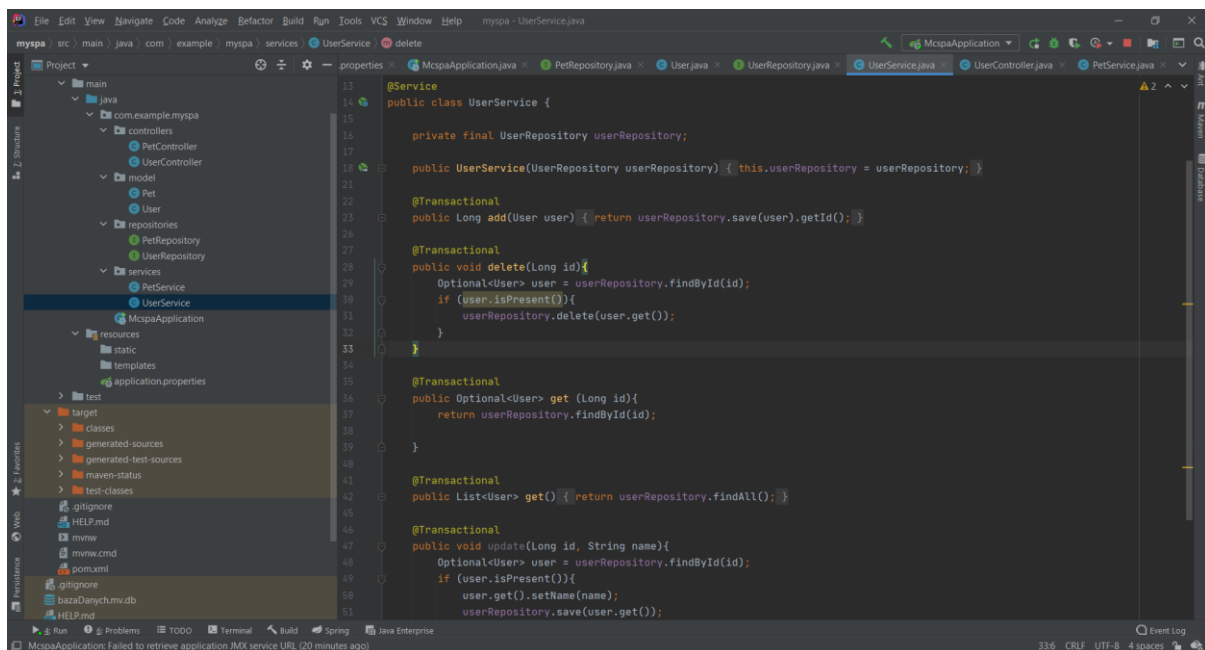
Odczytywane będą po id które będą same tworzone przez dodanie anotacji @Id i @GeneratedValue + Gettery i Settery.

W interfejsach repozytorium jest rozszerzenie JpaRepository które umożliwia podstawowe zapytania do bazy danych.

Serwisy do Pet i User mają zdefiniowane wszystkie funkcjonalności potrzebne aplikacji. Zawierają funkcje “add”, “get”, “patch” i “delete” dla nazwy użytkownika i nazwy jego pupila (Pet).



```
18
19
20 @Service
21 public class PetService {
22
23     private final PetRepository petRepository;
24
25     public PetService(PetRepository petRepository) { this.petRepository = petRepository; }
26
27     @Transactional
28     public Long add(Pet pet) { return petRepository.save(pet).getId(); }
29
30     public void delete(Long id){
31         Optional<Pet> pet = petRepository.findById(id);
32         if (pet.isPresent()){
33             petRepository.delete(pet.get());
34         }
35     }
36
37     @Transactional
38     public Optional<Pet> get(Long id){
39         return petRepository.findById(id);
40     }
41
42     @Transactional
43     public List<Pet> get() { return petRepository.findAll(); }
44
45     @Transactional
46     public void update(Long id, String name){
47         Optional<Pet> pet = petRepository.findById(id);
48         if (pet.isPresent()){
49             pet.get().setName(name);
50         }
51     }
52 }
```



```
13
14 @Service
15 public class UserService {
16
17     private final UserRepository userRepository;
18
19     public UserService(UserRepository userRepository) { this.userRepository = userRepository; }
20
21     @Transactional
22     public Long add(User user) { return userRepository.save(user).getId(); }
23
24     @Transactional
25     public void delete(Long id){
26         Optional<User> user = userRepository.findById(id);
27         if (user.isPresent()){
28             userRepository.delete(user.get());
29         }
30     }
31
32     @Transactional
33     public Optional<User> get(Long id){
34         return userRepository.findById(id);
35     }
36
37     @Transactional
38     public List<User> get() { return userRepository.findAll(); }
39
40     @Transactional
41     public void update(Long id, String name){
42         Optional<User> user = userRepository.findById(id);
43         if (user.isPresent()){
44             user.get().setName(name);
45             userRepository.save(user.get());
46         }
47     }
48 }
```

W rest kontrolerach są mapowania adresów dla klas usera i pet. Zawiera 4 podstawowe endpointy:

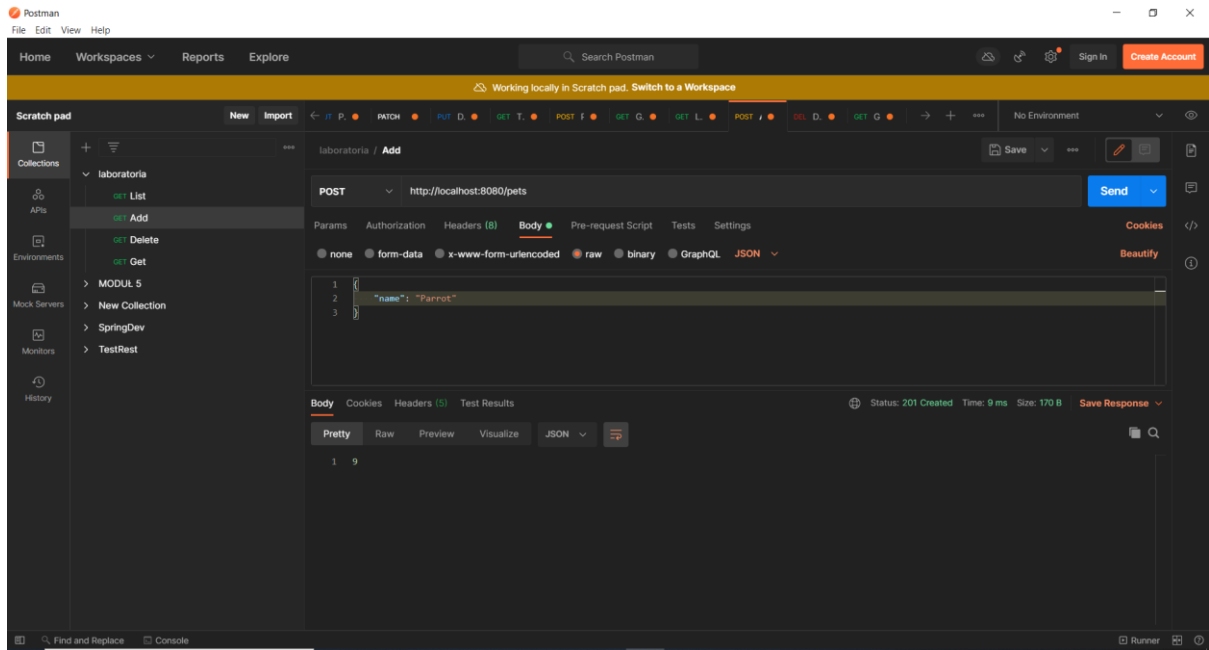
“Get” który tworzy tabelę i wyświetla listę użytkowników i pupilów. Zapytanie typu Post będzie dodawać nowych użytkowników i pupilów automatycznie tworząc im id. Delete usuwa po id użytkownika i pupila. A Patch aktualizuje dane, ale nie wstawia nowych tylko podmienia jakieś miejsca.

```
12 import org.springframework.web.bind.annotation.*;
13 import org.springframework.http.ResponseEntity;
14
15 @RestController
16 @RequestMapping("/pets")
17 public class PetController {
18
19     PetService petService;
20     PetController(PetService petService) { this.petService = petService; }
21
22     @GetMapping
23     public ResponseEntity<List<Pet>> list() { return new ResponseEntity(petService.get(), HttpStatus.OK); }
24
25     @PostMapping
26     public ResponseEntity<Long> add(@RequestBody Pet pet) {
27         return new ResponseEntity(petService.add(pet), HttpStatus.CREATED);
28     }
29
30     @DeleteMapping("/{id}")
31     public ResponseEntity<Void> delete(@PathVariable Long id) {
32         petService.delete(id);
33         return new ResponseEntity(HttpStatus.OK);
34     }
35
36     @GetMapping("/{id}")
37     public ResponseEntity<Pet> get(@PathVariable Long id) {
38         Optional<Pet> pet = petService.get(id);
39
40         if (pet.isPresent()) {
41             return new ResponseEntity(pet.get(), HttpStatus.OK);
42         } else {
43             return new ResponseEntity(HttpStatus.NO_CONTENT);
44         }
45     }
46 }
47
48 }
```

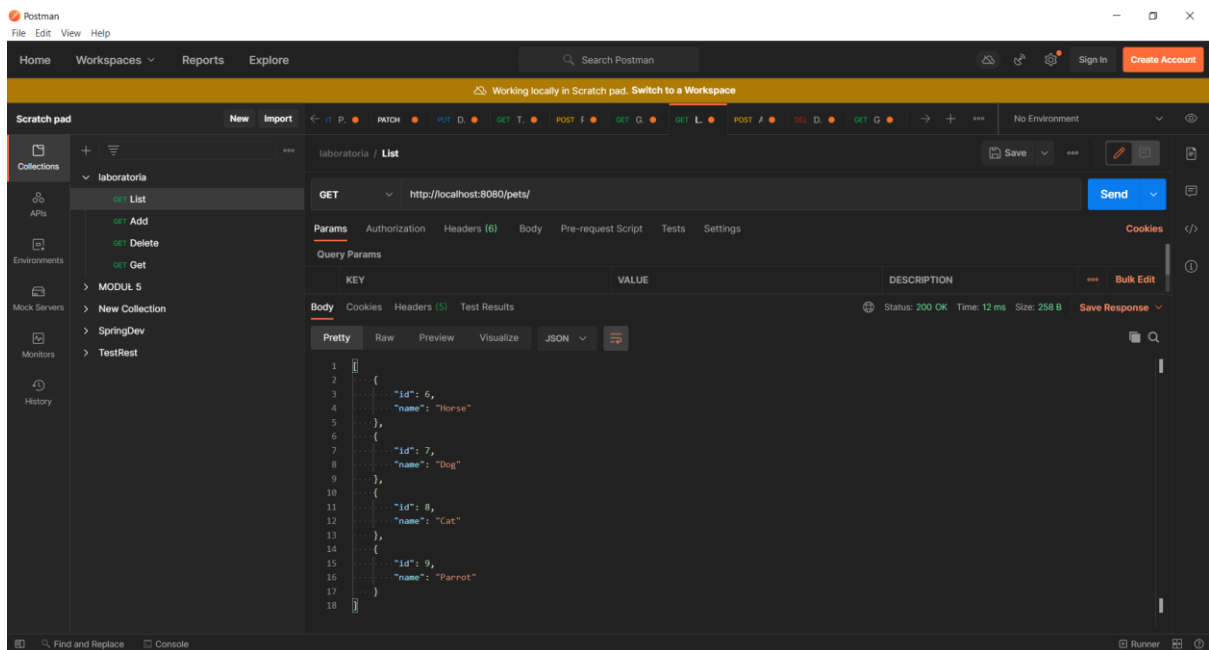
```
11 import org.springframework.web.bind.annotation.*;
12 import org.springframework.http.ResponseEntity;
13
14 @RestController
15 @RequestMapping("/users")
16 public class UserController {
17
18     UserService userService;
19     UserController(UserService userService) { this.userService = userService; }
20
21     @GetMapping
22     public ResponseEntity<List<User>> list() { return new ResponseEntity(userService.get(), HttpStatus.OK); }
23
24     @PostMapping
25     public ResponseEntity<Long> add(@RequestBody User user) {
26         return new ResponseEntity(userService.add(user), HttpStatus.CREATED);
27     }
28
29     @DeleteMapping("/{id}")
30     public ResponseEntity<Void> delete(@PathVariable Long id) {
31         userService.delete(id);
32         return new ResponseEntity(HttpStatus.OK);
33     }
34
35     @GetMapping("/{id}")
36     public ResponseEntity<User> get(@PathVariable Long id) {
37         Optional<User> user = userService.get(id);
38
39         if (user.isPresent()) {
40             return new ResponseEntity(user.get(), HttpStatus.OK);
41         } else {
42             return new ResponseEntity(HttpStatus.NO_CONTENT);
43         }
44     }
45 }
46
47 }
```

Testowanie żądań w Postmanie na klasie Pet:

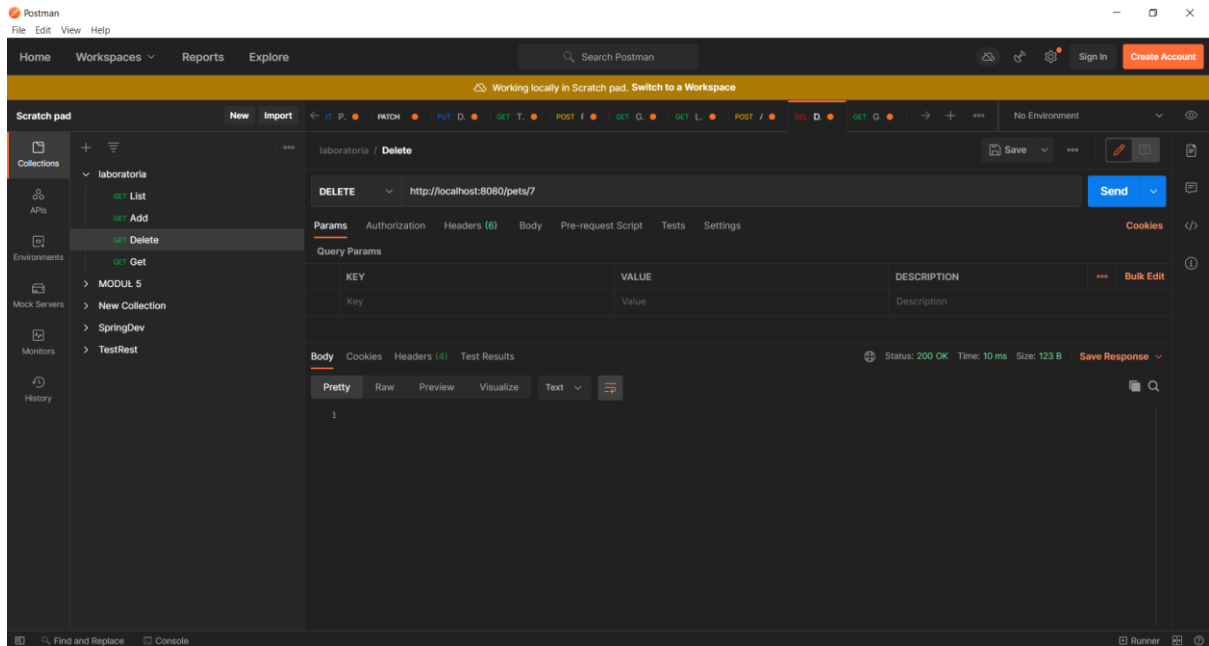
Post - przesyłam 3 nowe zwierzęta, do serwera. Zwierzętom dodanym do bazy wygenerowały się nowe id.



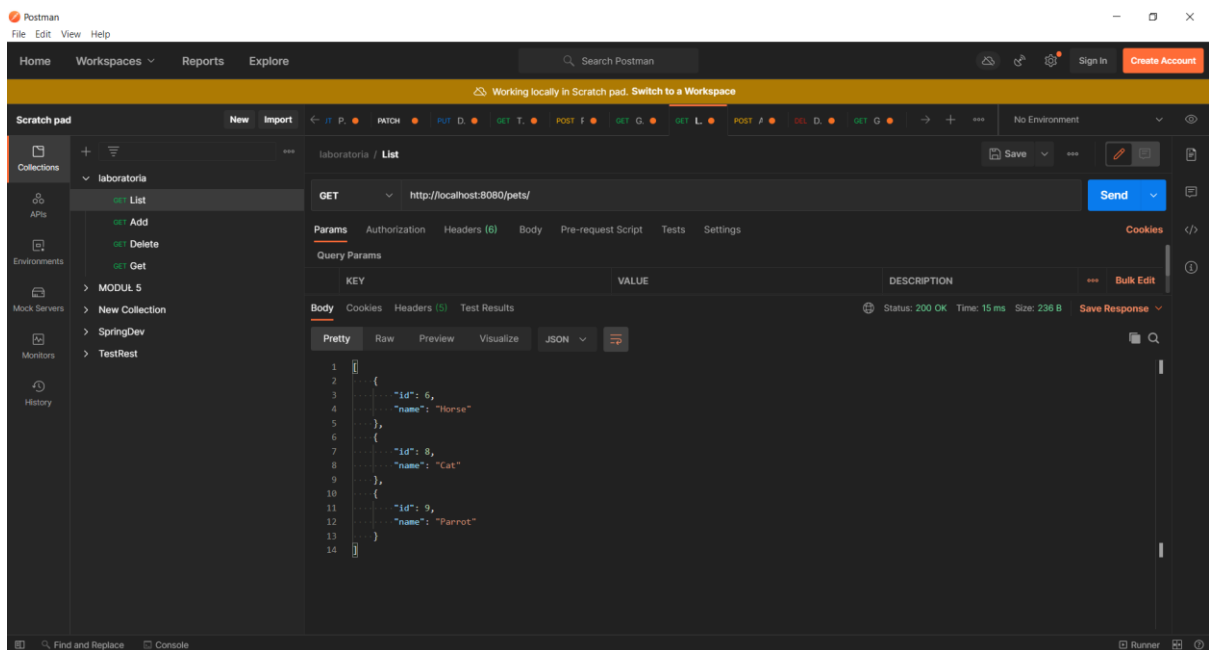
Get – sprawdzam listę wszystkich zwierząt. Są trzy nowe w bazie wygenerowane za pomocą żądania post.



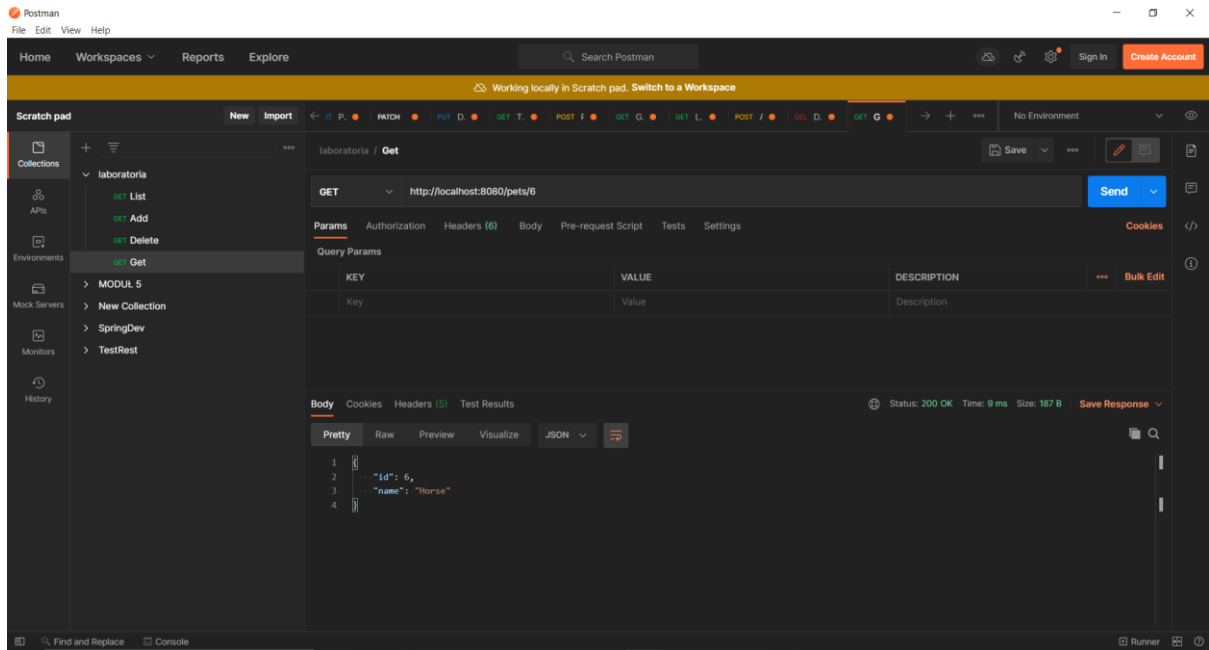
Delete - kasuję wybranego zwierzątka wpisując w endpointzie jego id. Dla testu usunę pupila o id 7 z bazy danych



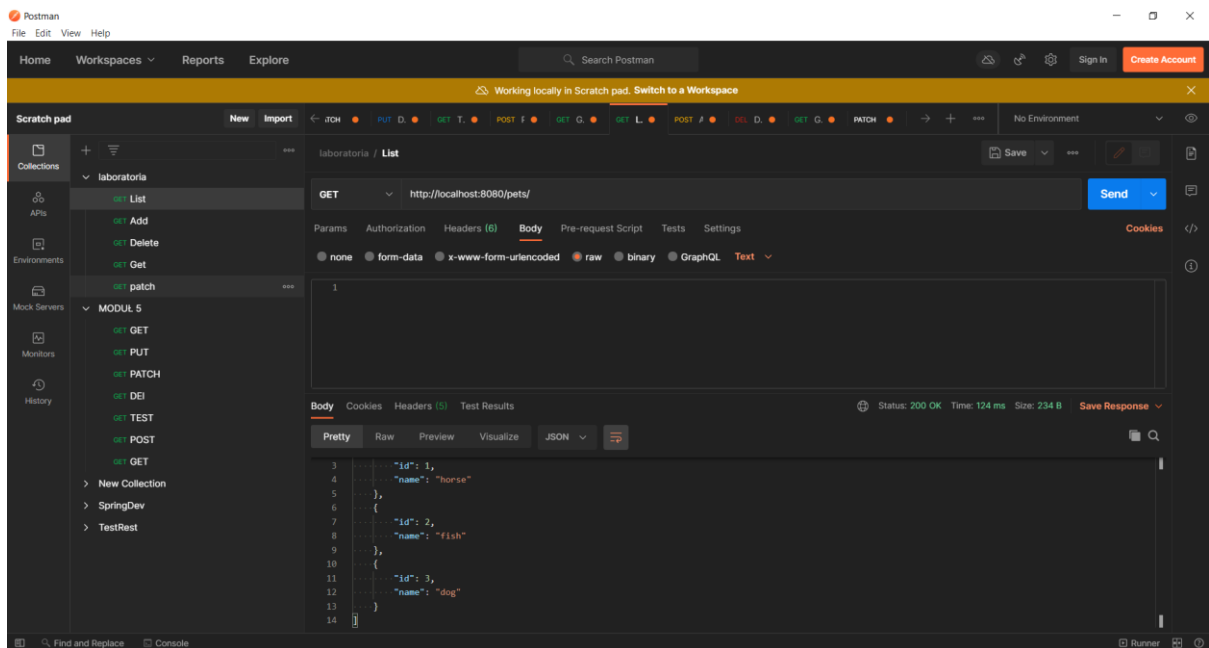
Na liście nie ma już tego obiektu "pet", czyli pupila:



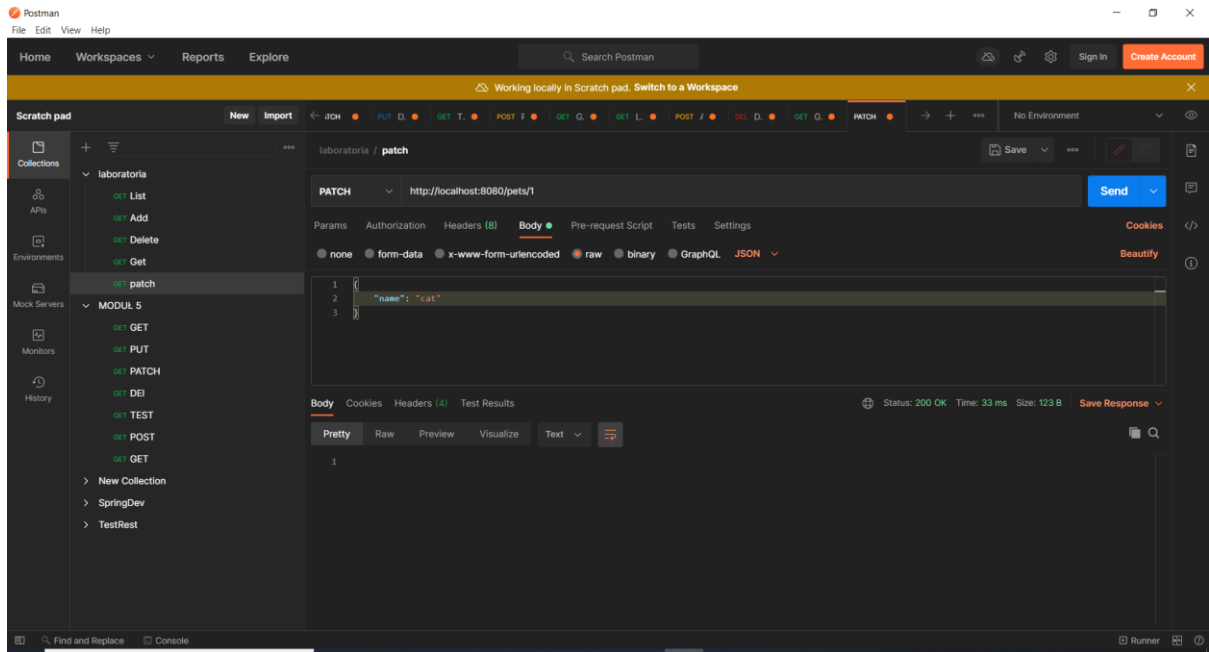
Za pomocą żądania get pobieram wybranego pupila podając id w endpointzie:



Patch - metoda Patch będzie służyła do aktualizowania danych i konwencja jest taka, że PATCH zmienia tylko część danych nie całe.



Użyję metody patch na wartości o id 1 (horse) i zamienię name na “cat”:



Nazwa obiektu “pet” o id 1 z horse została zaktualizowana do “cat”.

