2025 — 2026

1TI – 1ACS



UCLL HOGESCHOOL

Back-End Development

Spring Boot — HTTP API POST, PUT and DELETE

N. Jacobs, J. Pieck, E. Steegmans, B. Van Impe, S. Van Peborgh, V. Witters

CONTENTS

- HTTP POST Method
- HTTP PUT Method
- HTTP DELETE Method

HTTP BUILDING BLOCKS

- Remember that the HTTP protocol governs the rules of communication
  - Every *resource* is given a URL (**U**niform **R**esource **L**ocator) to identify it
  - Methods can be performed on a resource
    - Every method consists of a request and a response
  - Requests & responses consist of a body and a header
    - The header contains metadata
      - Who executed this request?
      - In what format does the sender of the request expect a response from the backend?
    - The body contains the data of the request or response
  - HTTP is *stateless*
    - Each request - response cycle is self-contained
  - HTTP is not secure by default
    - Request and response are sent in plain text
    - The HTTPS (HTTP **S**ecure) extension encrypts messages

HTTP BUILDING BLOCKS — URL

- Every HTTP resource has a URL
  - The root URL of your backend is http://localhost:<port>
    - "localhost" stands for your computer, since the backend is running on your pc
    - You can see the port your backend started at in the logs of your application
  - Every resource is mapped to a subdirectory of the root URL
    - For example
      - http://localhost:8080/pony
      - http://localhost:8080/jockey/Craig/horse
  - You can perform operations on your backend by invoking methods on URLs

- The combination of method and URL must be unique within the backend

HTTP BUILDING BLOCKS — METHODS

- The method indicates what type of request is being sent to the URL
  - GET: retrieving information
  - POST: storing new information
  - PUT: updating existing information
  - DELETE: removing information

- Other, more specialised methods exist
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

- The combination of method and URL must be unique within the backend

HTTP BUILDING BLOCKS — STATUS CODES

- The status code of the response indicates whether the backend could succesfully process the request
  - 2xx – Succes
  - 3xx – Redirect
  - 4xx – Client Error
  - 5xx – Server Error

- Other, more specialised status codes exist
  - https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

HTTP POST Method

HTTP PUT Method

HTTP DELETE Method

HTTP POST Method

IMPLEMENTING HTTP POST

```
@RestController
@RequestMapping("/pony")
public class PonyController {

    // ...

    @PostMapping
    public Pony addPony(@RequestBody Pony pony) {
        return ponyService.addPony(pony);
    }
}
```

- The body of the HTTP POST request gets mapped to the method-parameter annotated with @RequestBody

IMPLEMENTING HTTP POST – (DE)SERIALIZATION

- The mapping of an HTTP request body to a domain object is called *deserialization*
  - Spring Boot does this automatically with the help of a library called *Jackson*
  - If your request body is structured in the same way as your class this process is painless, otherwhise an exception will be thrown or fields will be left empty
  - Deserialisation uses the constructors and setters you defined in your code
    - If a constructor without arguments is present, that constructur and setters will be used
    - If no constructor without arguments is present but **one** constructor with arguments is, then that constructor will be used
    - Otherwise an exception will be thrown

- The translation of a domain object to a JSON response is the inverse operation and is called *serialization*
  - The fields present in the JSON will be based on the **getters** of your class, not on the fields!

IMPLEMENTING HTTP POST

```java
@Service
public class PonyService {

    private PonyRepository ponyRepository;

    @Autowired
    public PonyService(PonyRepository ponyRepository) {
        this.ponyRepository = ponyRepository;
    }

    public Pony addPony(Pony pony) {
        if (findPonyByName(pony.getName()).isEmpty()) {
            return ponyRepository.addPony(pony);
        }

        throw new RuntimeException("Pony already exists!");
    }
}
```

```java
@Repository
public class PonyRepository {

    private List<Pony> ponies;

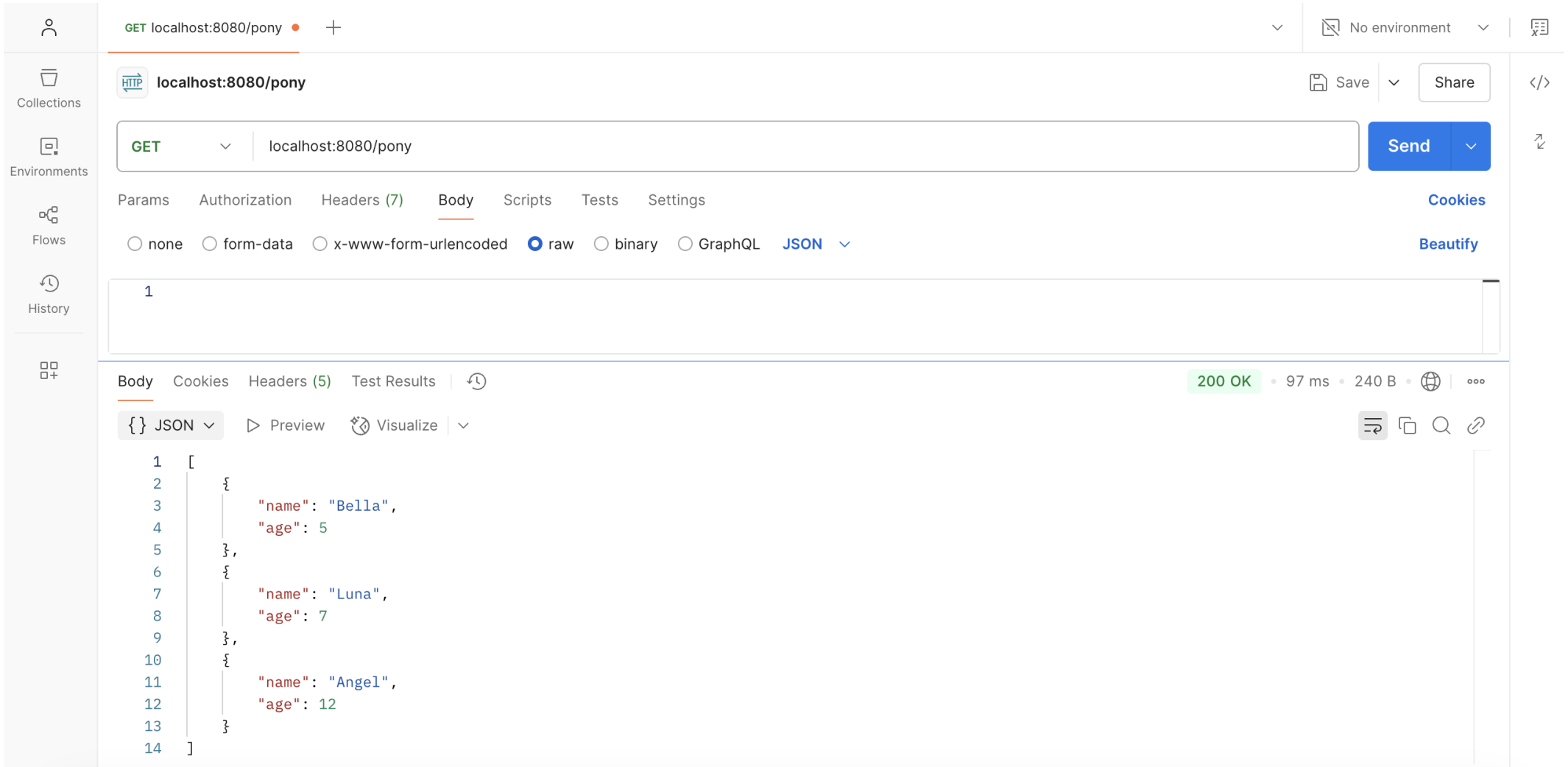
    public PonyRepository() {
        ponies = new ArrayList<>();
        // ...
    }

    // ...

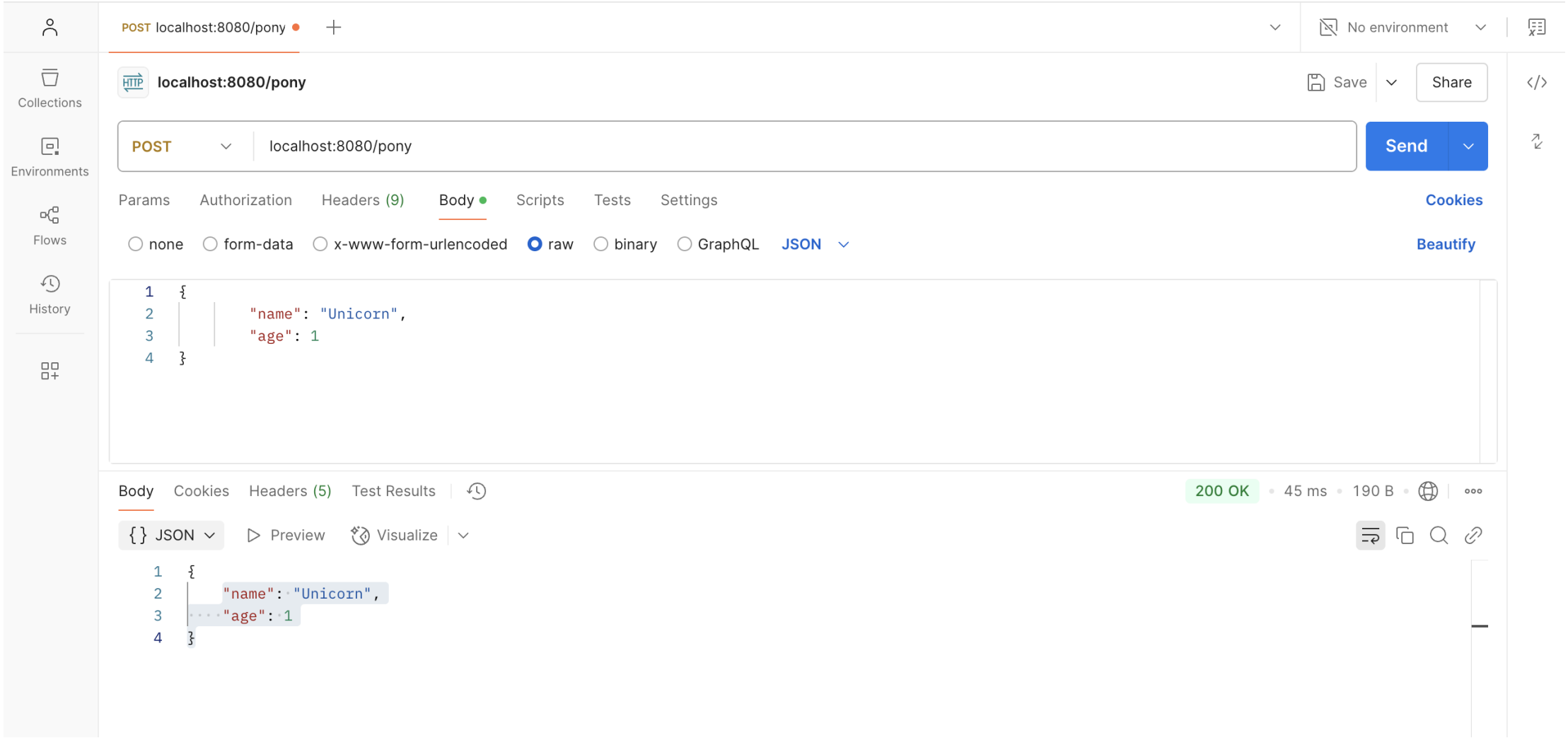
    public Pony addPony(Pony pony) {
        ponies.add(pony);
        return pony;
    }
}
```

- The service performs business validation and calls the repository
- The repository saves the object

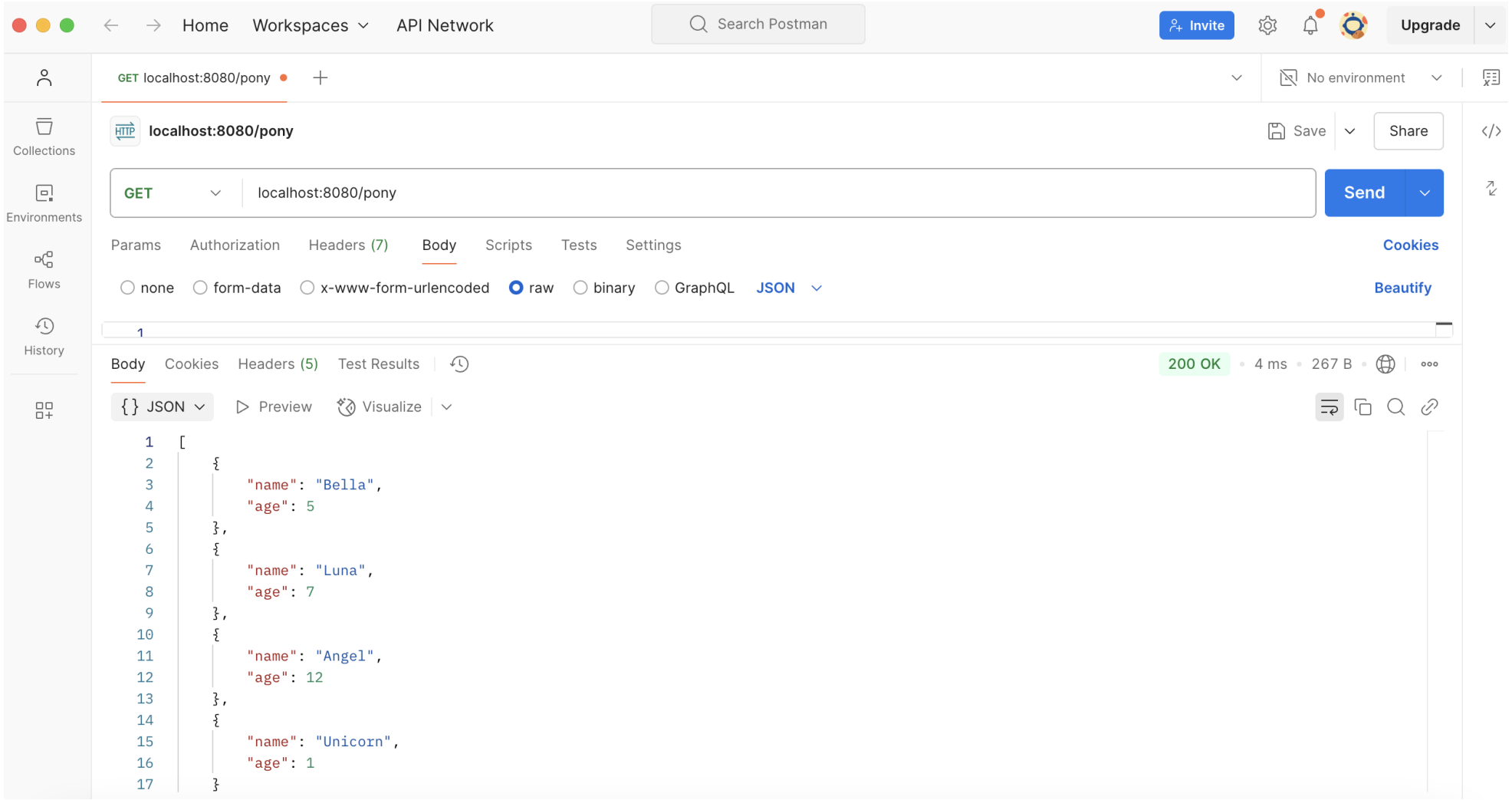
HTTP GET BEFORE POST



HTTP POST


POST localhost:8080/pony
No environment
Collections
Environments
Flows
History
localhost:8080/pony
Save
Share
POST
localhost:8080/pony
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"name": "Unicorn",
"age": 1
}
Body
Cookies
Headers (5)
Test Results
200 OK
45 ms
190 B
JSON
Preview
Visualize
{
"name": "Unicorn",
"age": 1
}

HTTP GET AFTER POST



HTTP POST Method

HTTP PUT Method

HTTP DELETE Method

HTTP PUT Method

IMPLEMENTING HTTP PUT

```
@RestController
@RequestMapping("/pony")
public class PonyController {

    private PonyService ponyService;

    @Autowired
    public PonyController(PonyService ponyService) {
        this.ponyService = ponyService;
    }

    // ...

    @PutMapping("/{name}")
    public Pony updatePony(@PathVariable String name, @RequestBody Pony pony) {
        return ponyService.updatePony(name, pony);
    }
}
```

- The identifier of the pony being updated is passed along via a @PathVariable
- The new data is passed along via the @RequestBody

IMPLEMENTING HTTP PUT

```
@Service
public class PonyService {

    private PonyRepository ponyRepository;

    @Autowired
    public PonyService(PonyRepository ponyRepository) {
        this.ponyRepository = ponyRepository;
    }

    // ...

    public Pony updatePony(String name, Pony newInformation) {
        Pony pony = findPonyByName(name);
        pony.updateNameAndAge(newInformation.getName(), newInformation.getAge());
        return pony;
    }
}
```

```
public class Pony {

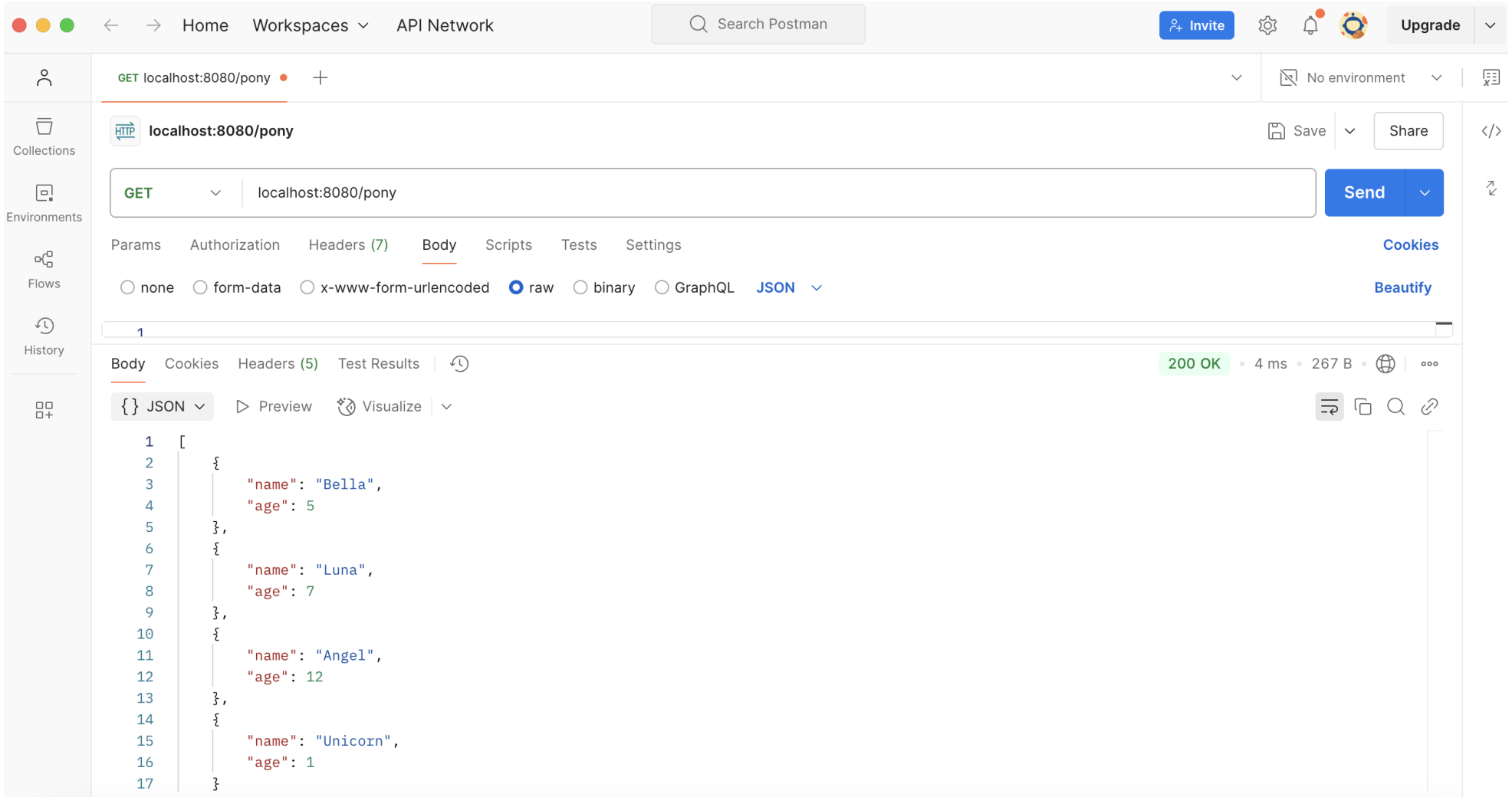
    private String name;
    private int age;

    // ...

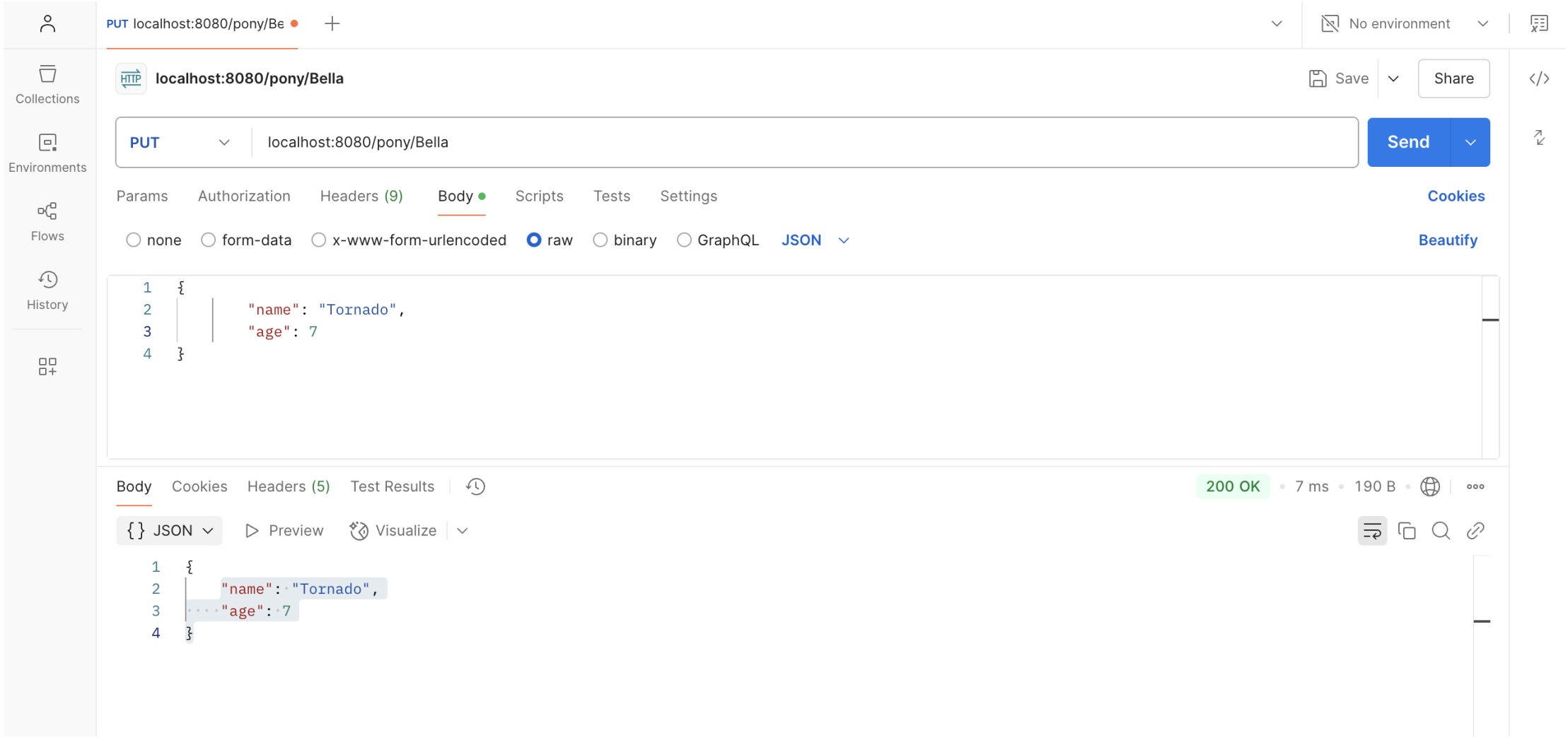
    public void updateNameAndAge(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

- We retrieve the pony to be updated via the repository and update the necessary information
- In general, care should be spent deciding what fields can be updated
  - This should be done from the perspective of the object being updated

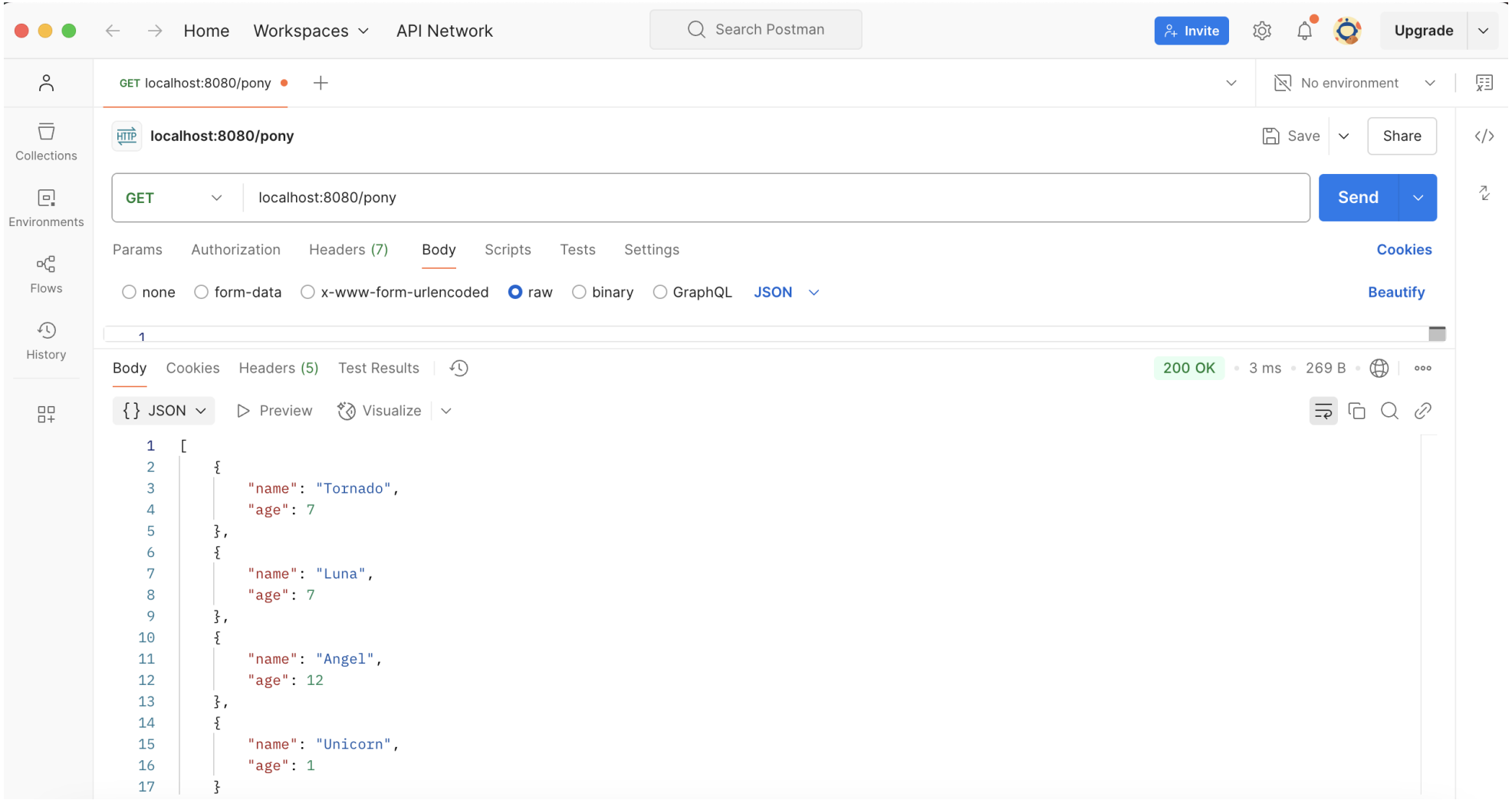
HTTP GET BEFORE PUT



HTTP PUT


PUT localhost:8080/pony/Be
No environment
Collections
Environments
Flows
History
localhost:8080/pony/Bella
Save
Share
PUT
localhost:8080/pony/Bella
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"name": "Tornado",
"age": 7
}
Body
Cookies
Headers (5)
Test Results
200 OK
7 ms
190 B
JSON
Preview
Visualize
{
"name": "Tornado",
"age": 7
}

HTTP GET AFTER PUT



HTTP POST Method

HTTP PUT Method

HTTP DELETE Method

HTTP DELETE Method

IMPLEMENTING HTTP DELETE

```
@RestController
@RequestMapping("/pony")
public class PonyController {

    private PonyService ponyService;

    @Autowired
    public PonyController(PonyService ponyService) {
        this.ponyService = ponyService;
    }

    // ...

@DeleteMapping("/{name}")
 public void removePony(@PathVariable String name) {
 ponyService.removePony(name);
 }
}
```

- The specific pony being removed is passed along via a @PathVariable

IMPLEMENTING HTTP DELETE

```java
@Service
public class PonyService {

    private PonyRepository ponyRepository;

    @Autowired
    public PonyService(PonyRepository ponyRepository) {
        this.ponyRepository = ponyRepository;
    }

    // ...

    public void removePony(String name) {
        Pony pony = findPonyByName(name);
        ponyRepository.removePony(pony);
    }
}
```

```java
@Repository
public class PonyRepository {

    private List<Pony> ponies;

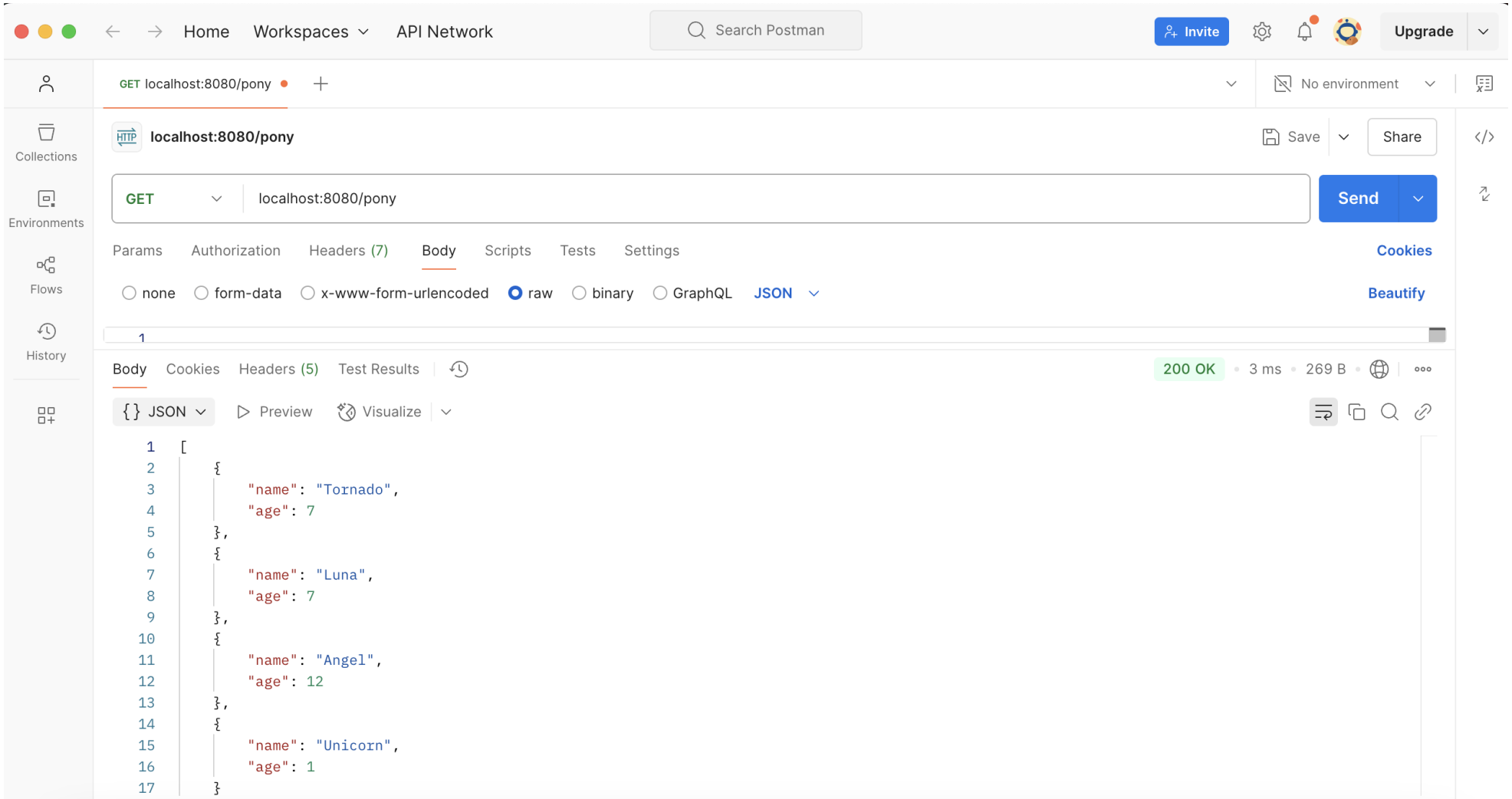
    public PonyRepository() {
        ponies = new ArrayList<>();
        // ...
    }

    // ...

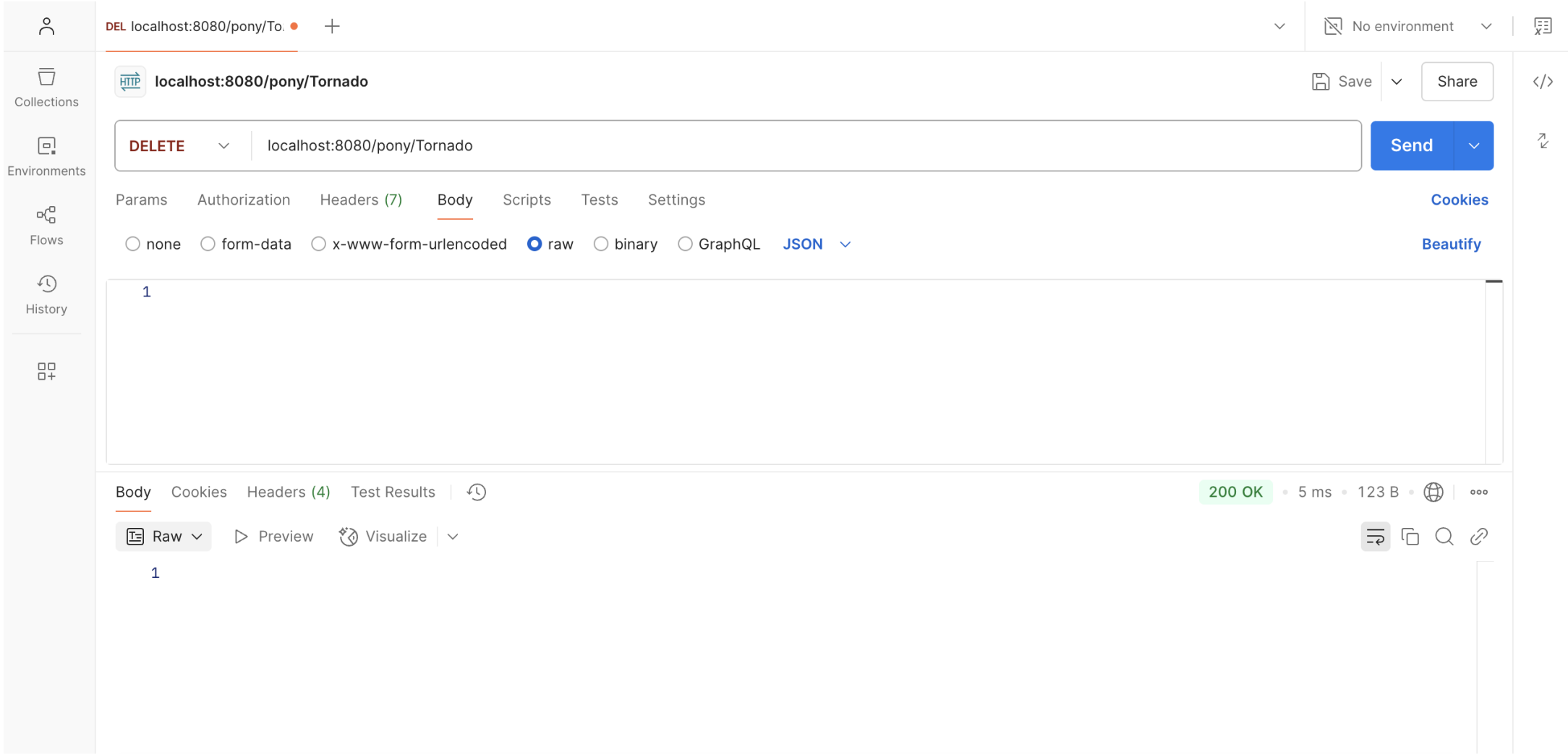
    public void removePony(Pony pony) {
        ponies.remove(pony);
    }
}
```

- We retrieve the pony that should be removed from the repository
- We instruct the repository to remove this pony

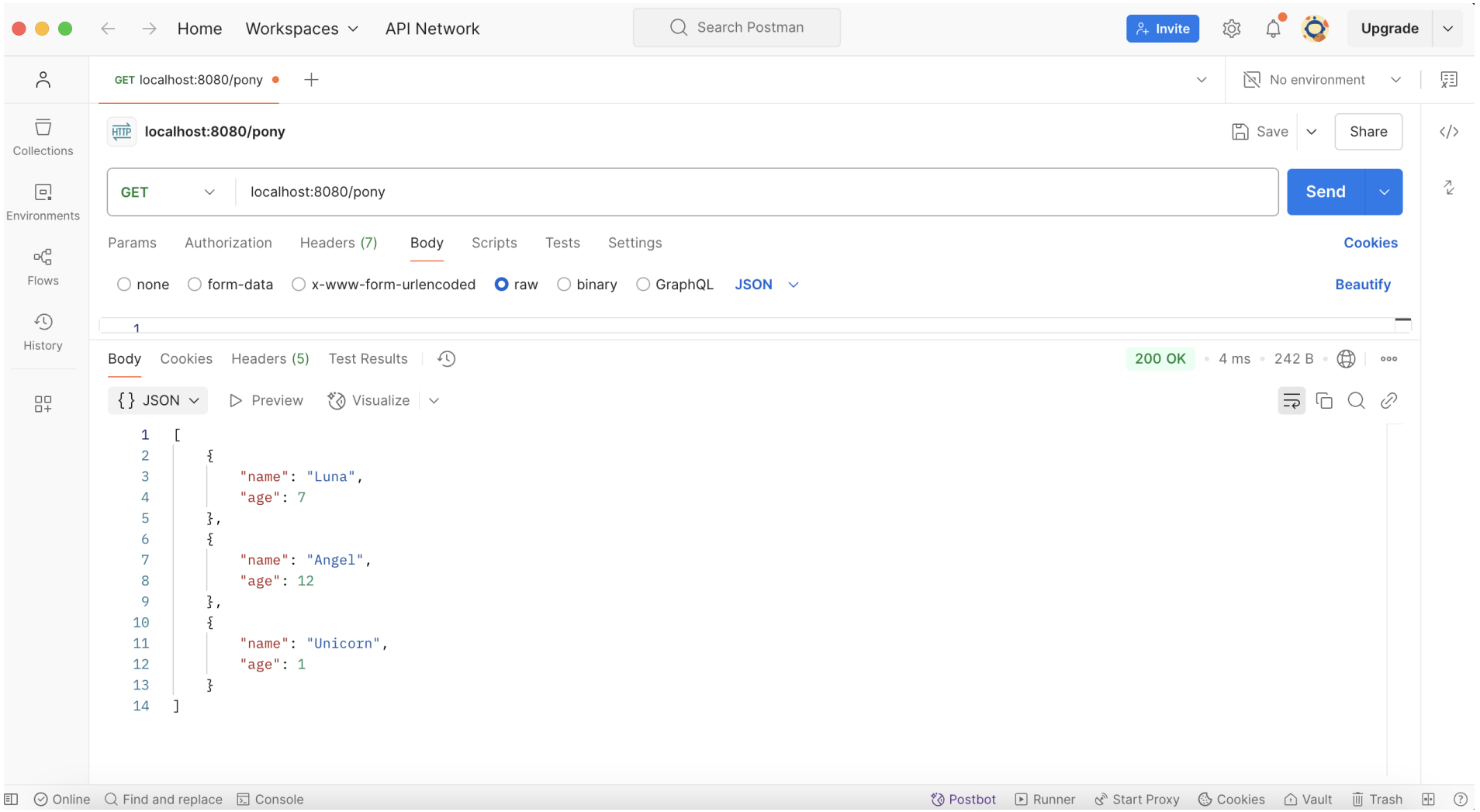
HTTP GET BEFORE DELETE



HTTP DELETE


DEL localhost:8080/pony/To.
No environment
Collections
Environments
Flows
History
localhost:8080/pony/Tornado
Save
Share
DELETE
localhost:8080/pony/Tornado
Send
Params
Authorization
Headers (7)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1
Body
Cookies
Headers (4)
Test Results
200 OK
5 ms
123 B
Raw
Preview
Visualize
1

HTTP GET AFTER DELETE



EXERCISES @CLASS

- exercise1
- exercise2
- exercise3
- exercise4
- exercise5
- exercise6



EXERCISES @HOME

- exercise7



