2025 — 2026

1TI – 1ACS



UCLL HOGESCHOOL

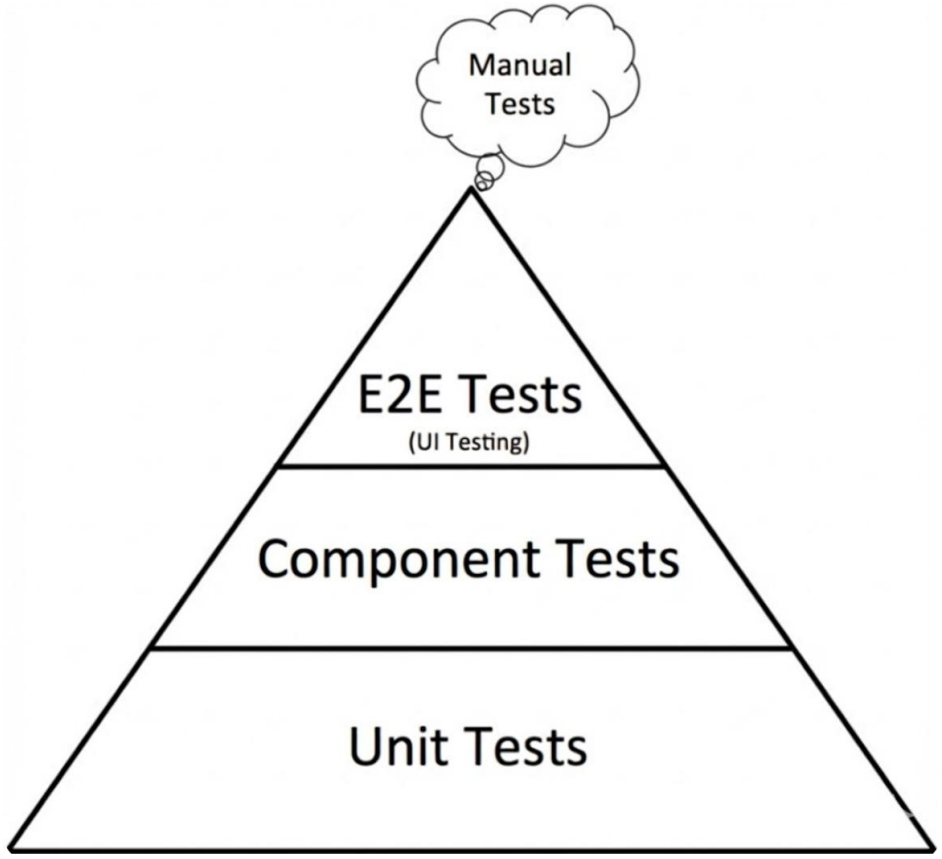
Back-End Development

Spring Boot — HTTP API Testing

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

TESTING WITH SPRING — TESTING THE HTTP API

- It is interesting to test an entire HTTP call
- To do this our test needs to start-up Spring so we can test the entire chain
- This is a Component Test because of the increased scope and the inclusion of Spring in the test



TESTING WITH SPRING — TESTING THE HTTP API

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

- Including this Spring module enables support for using Spring in tests
  - It also includes JUnit so it is from now on the only testing dependency we need
- Scope test tells maven that we only need this dependency in the test phase
  - Maven does not need to include the dependency when packaging the artifact

TESTING WITH SPRING — TESTING THE HTTP API

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
    <scope>test</scope>
</dependency>
```

- Including this Spring module enables an HTTP client that we can use in our component tests

TESTING WITH SPRING — TESTING THE HTTP API

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@AutoConfigureWebTestClient
public class PonyTest {

    private WebTestClient client;
    private PonyRepository repository;

    @Autowired
    public PonyTest(WebTestClient client, PonyRepository repository) {
        this.client = client;
        this.repository = repository;
    }


    @Test
    public void given3Ponies_whenInvokingGetPony_then3PoniesAreReturned() {
        client.get()
                .uri("/pony")
                .exchange()
                .expectStatus().is2xxSuccessful()
                .expectBody()
                .json("""
                        [
                         {"name": "Bella", "age": 5},
                         {"name": "Luna", "age": 7},
                         {"name": "Angel", "age": 12}
                        ]
                        """);

    }
}
```

@SpringBootTest makes sure that our entire backend with Spring is started before running the test

TESTING WITH SPRING — TESTING THE HTTP API

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@AutoConfigureWebTestClient
public class PonyTest {

    private WebTestClient client;
    private PonyRepository repository;

    @Autowired
    public PonyTest(WebTestClient client, PonyRepository repository) {
        this.client = client;
        this.repository = repository;
    }


    @Test
    public void given3Ponies_whenInvokingGetPony_then3PoniesAreReturned() {
        client.get()
                .uri("/pony")
                .exchange()
                .expectStatus().is2xxSuccessful()
                .expectBody()
                .json("""
                        [
                         {"name": "Bella", "age": 5},
                         {"name": "Luna", "age": 7},
                         {"name": "Angel", "age": 12}
                        ]
                        """);

    }
}
```

@AutoConfigureWebTestClient configures a client that is then put inside our test class by Spring at the @Autowired annotation

TESTING WITH SPRING — TESTING THE HTTP API

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@AutoConfigureWebTestClient
public class PonyTest {

    private WebTestClient client;
    private PonyRepository repository;

    @Autowired
    public PonyTest(WebTestClient client, PonyRepository repository) {
        this.client = client;
        this.repository = repository;
    }


    @Test
    public void given3Ponies_whenInvokingGetPony_then3PoniesAreReturned() {
        client.get()
                .uri("/pony")
                .exchange()
                .expectStatus().is2xxSuccessful()
                .expectBody()
                .json("""
                        [
                         {"name": "Bella", "age": 5},
                         {"name": "Luna", "age": 7},
                         {"name": "Angel", "age": 12}
                        ]
                        """);

    }
}
```

We program the client to call our API and check the result.

TESTING WITH SPRING — TESTING THE HTTP API

```
@Test
public void givenNoPonyTornado_whenInvokingPostTornado_thenPonyTornadoIsSaved() {
   client.post()
         .uri("/pony")
         .header("Content-Type", "application/json")
         .bodyValue("{\"name\":\"Tornado\",\"age\":17}")
         .exchange()
         .expectStatus().is2xxSuccessful()
         .expectBody()
         .json("""
               {
                "name": "Tornado",
                "age": 17
               }
               """);

   Assertions.assertEquals("Tornado", repository.findPonyByName("Tornado").get().getName());
}
```

- We program the client to call our API and check the result
- We check the repository to make sure that the call did what it needed to do

TESTING WITH SPRING — TESTING THE HTTP API

```
@Test
public void givenPonyBella_whenInvokingDeleteBella_thenPonyBellaIsRemoved() {
    client.delete()
            .uri("/pony/Bella")
            .exchange()
            .expectStatus().is2xxSuccessful();

    Assertions.assertThrows(RuntimeException.class, () -> repository.findPonyByName("Bella"));
}
```

- We program the client to call our API and check the result
- We check the repository to make sure that the call did what it needed to do

TESTING WITH SPRING — TESTING THE HTTP API

```
@Test
public void givenPonyBella_whenInvokingUpdateBella_thenNewInformationIsSavedInDb() {
    client.put()
            .uri("/pony/Bella")
            .header("Content-Type", "application/json")
            .bodyValue("{\"name\":\"TheNicest\",\"age\":17}")
            .exchange()
            .expectStatus().is2xxSuccessful()
            .expectBody()
            .json("{\"name\":\"TheNicest\",\"age\":17}");

    Assertions.assertThrows(RuntimeException.class, () -> repository.findPonyByName("Bella"));
    Assertions.assertEquals("TheNicest", repository.findPonyByName("TheNicest").getName());
}
```

- We program the client to call our API and check the result
- We check the repository to make sure that the call did what it needed to do

TESTING WITH SPRING — TESTING THE HTTP API

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@AutoConfigureWebTestClient
public class HttpTest {

    ...

    @AfterEach
    public void resetData() {
        repository.resetRepositoryData();
    }

    // ...
}
```

```
@Repository
public class PonyRepository {

    // ...

    public void resetRepositoryData() {
        ponies = new ArrayList<>();
        ponies.add(new Pony("Bella", 5));
        ponies.add(new Pony("Luna", 7));
        ponies.add(new Pony("Angel", 12));
    }
}
```

- @AfterEach runs a method after each @Test method in the same class

TESTING WITH SPRING — TESTING THE HTTP API

```java
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@AutoConfigureWebTestClient
public class HttpTest {

    …

    @AfterEach
    public void resetData() {
        repository.resetRepositoryData();
    }

    // …
}
```

```java
@Repository
public class PonyRepository {

    // …

    public void resetRepositoryData() {
        ponies = new ArrayList<>();
        ponies.add(new Pony("Bella", 5));
        ponies.add(new Pony("Luna", 7));
        ponies.add(new Pony("Angel", 12));
    }
}
```

- The data is reset after every test using an extra utility method in our repository

TESTING WITH SPRING — TESTING THE HTTP API

- Component Tests should be written sparingly since they take a long time to execute

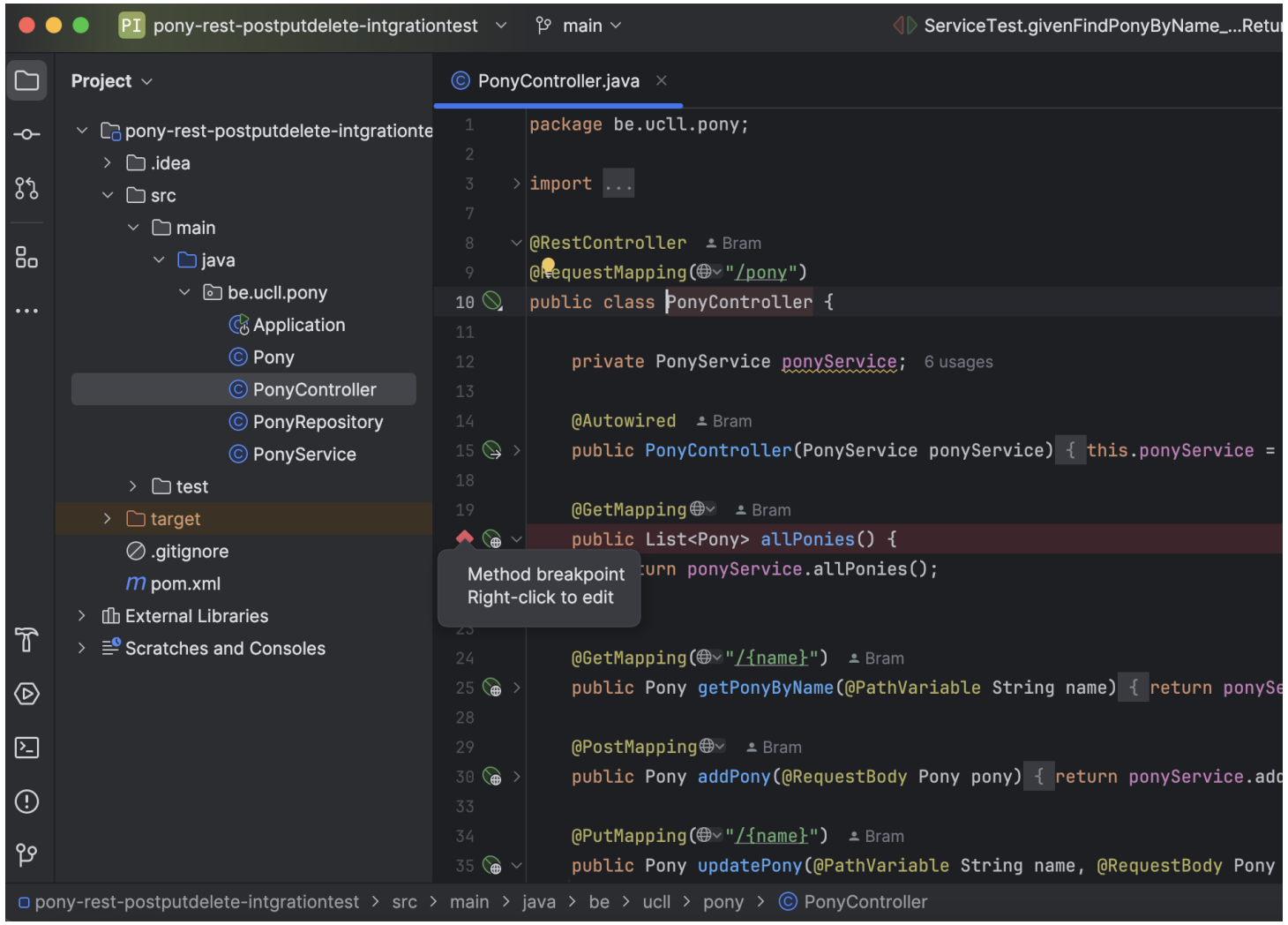
- A good rule of thumb for now is to write one component test for every HTTP method and many unit tests
  - All logic should be covered by unit tests
  - The component test provides additional coverage by checking one codepath (the happy case) in the running backend. This way the Spring framework is also included in the tests.

DEBUGGING WITH SPRING BOOT

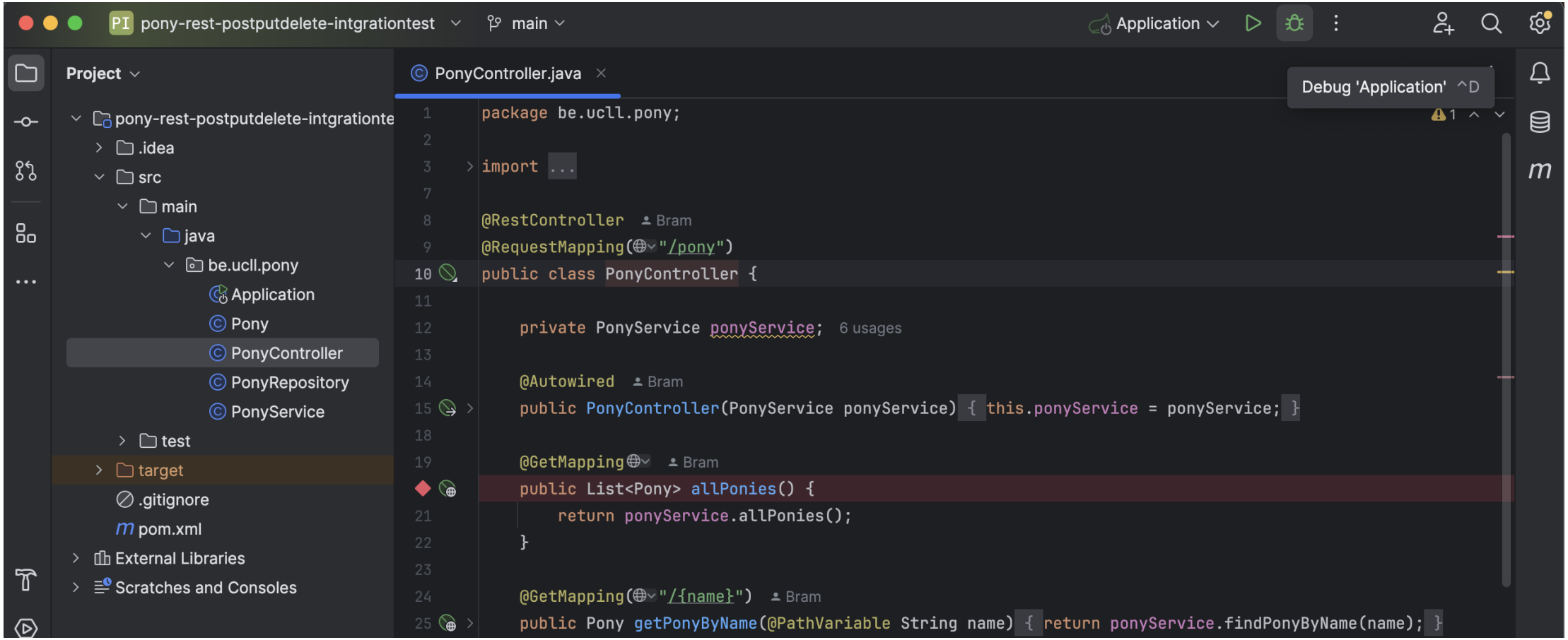
- Happens the same way as with plain Java
  - But with ”step into” and “step over” you will encounter a lot of framework code

- Most of the time easier to place breakpoints and to go from breakpoint to breakpoint

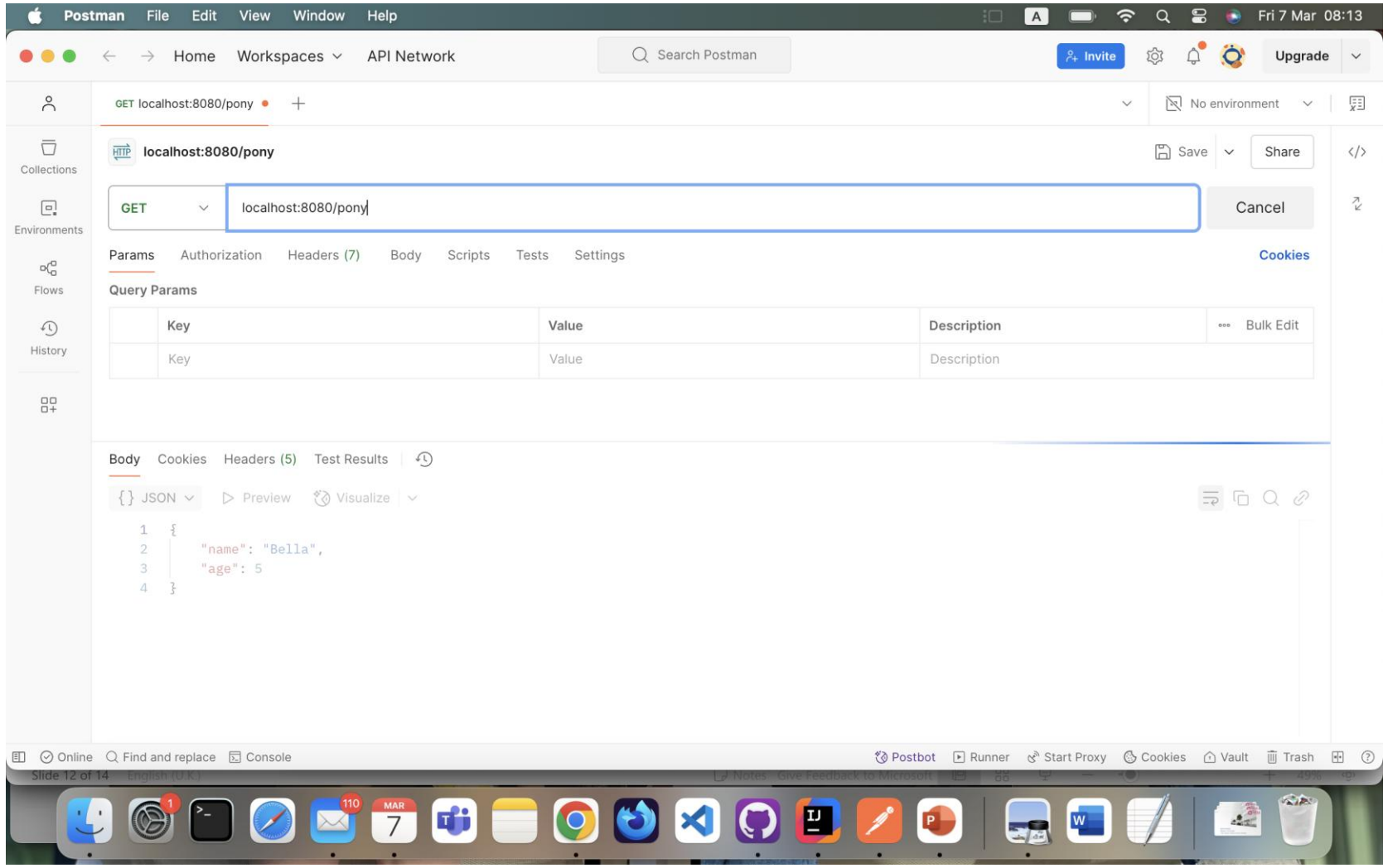
DEBUGGING – SET A BREAKPOINT



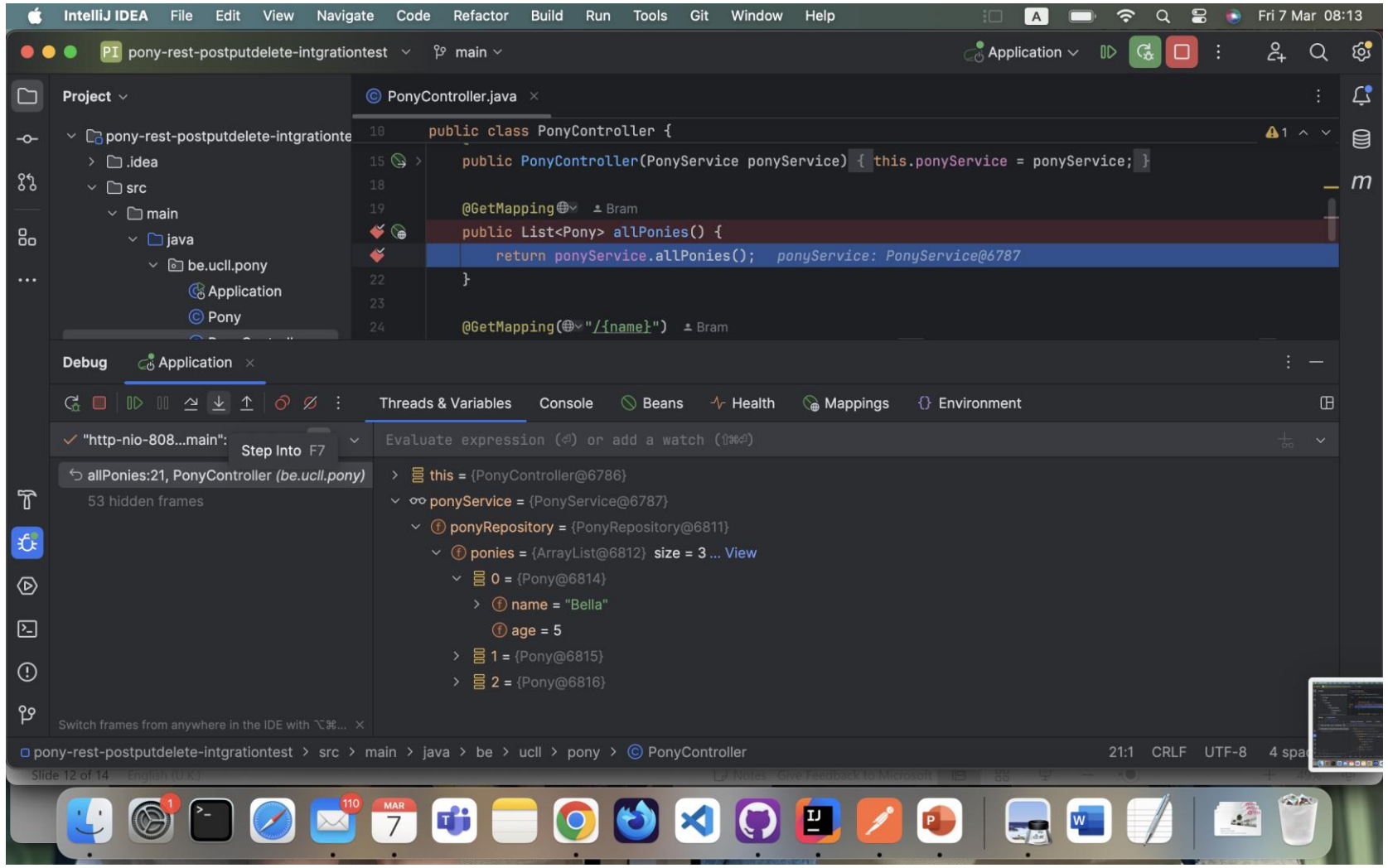
DEBUGGING – START THE DEBUGGER



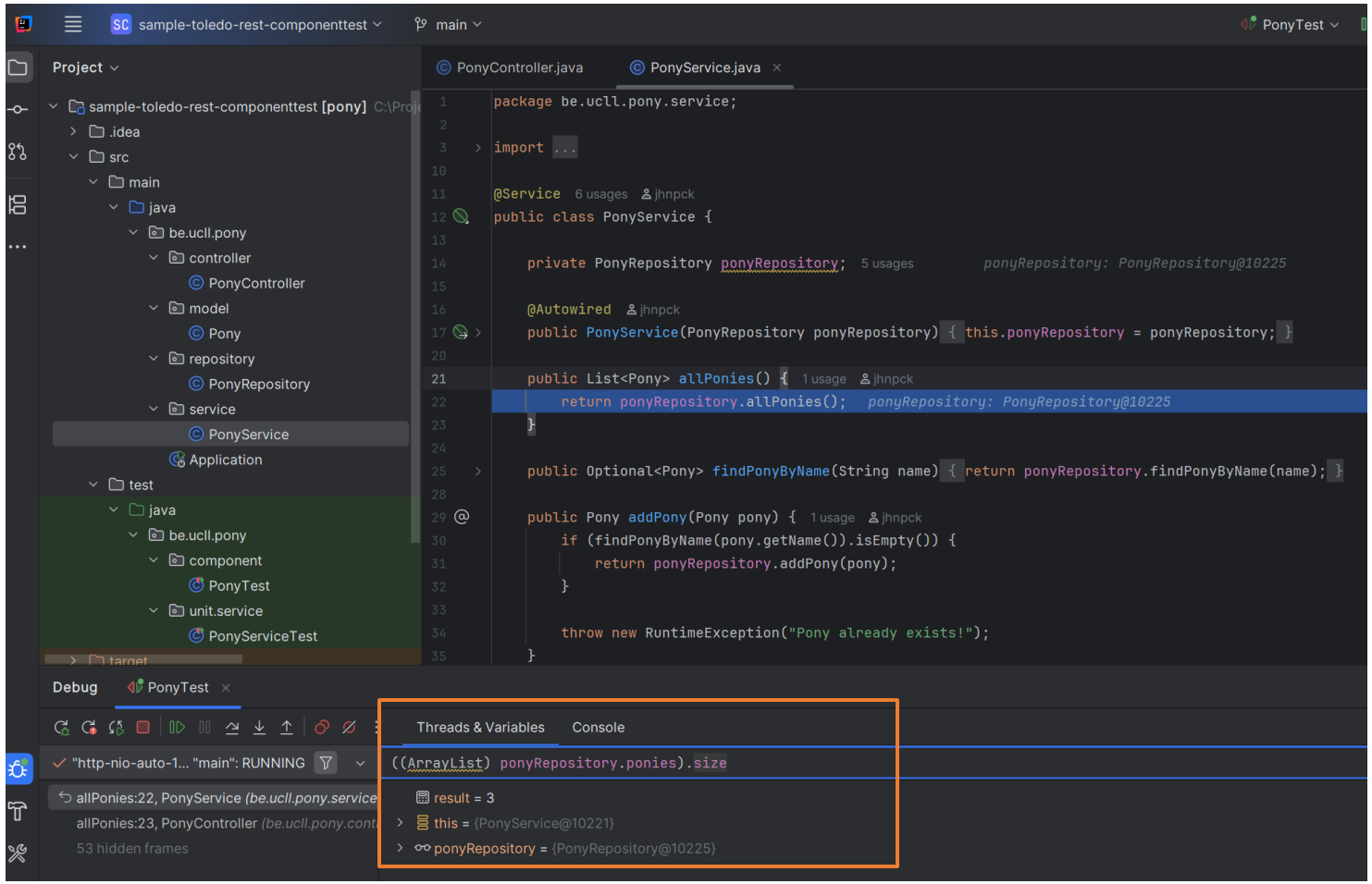
DEBUGGING – CALL THE END-POINT



DEBUGGING – GO BACK TO INTELLIJ AND STEP INTO OR ...



DEBUGGING – EVALUATE EXPRESSION



- While paused at a breakpoint, the “Evaluate Expression” tool is used here to execute a custom snippet of Java code to check the exact size of the internal ponies list.
- This feature allows you to instantly call methods, inspect object states and test complex logic in real-time without altering your source code.

EXERCISE @CLASS

- Exercise 1



EXERCISE @HOME

- Exercise 2
- Exercise 3



