Tester c’est douter ?


TESTER
C'EST DOUTER
COMMITSTRIP.COM

La galaxie des tests

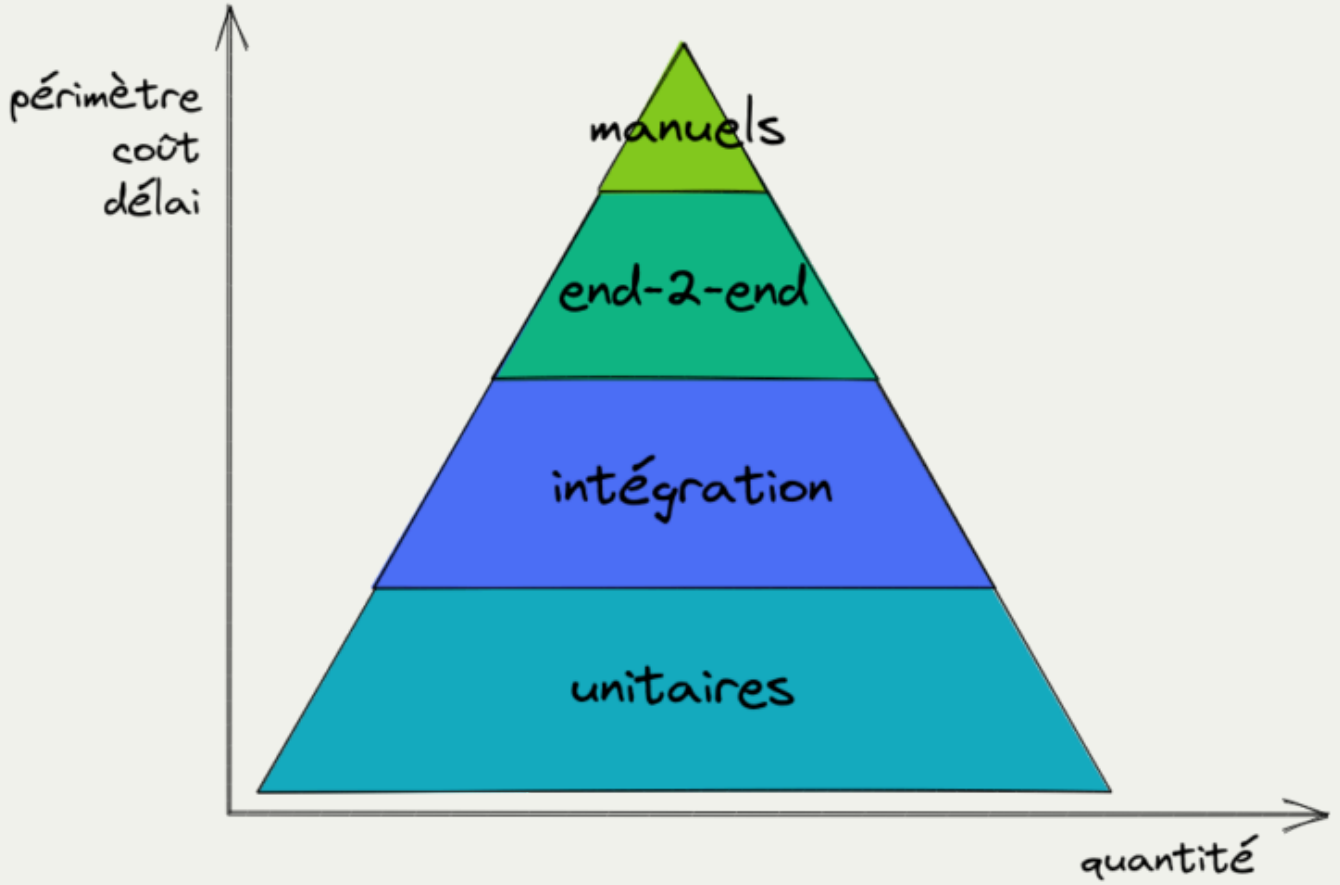
La galaxie des tests

- Les tests manuels
- Les tests de charge
- Les tests end-to-end
- Les tests de contrat
- Les tests d’intégrations
- Les tests unitaires

La galaxie des tests

- ~~Les tests manuels~~
- ~~Les tests de charge~~
- ~~Les tests end-to-end~~
- ~~Les tests de contrat~~
- Les tests d’intégrations
- Les tests unitaires

La pyramide


périmètre
coût
délai
manuels
end-2-end
intégration
unitaires
quantité

Les tests unitaires

F.I.R.S.T

- 🏃 Fast // Rapide
- 🏝️ Independent // Indépendant
- 🔁 Repeatable // Répétable
- ✅ Self-validating // Auto-validé
- 📉 Thorough // Complet

🏃 Fast // Rapide

- on veut un feedback rapide
- on veut rester concentré
- test lent = pas exécuté ⇒ test inutile

 Independant // Indépendant

- une seule raison d’échouer
- pas de dépendance extérieure
  - système de fichier, bdd, random, date, …
  - autre test

🔁 Repeatable // Répétable

- Toujours le même résultat
- Sinon, pas de confiance dans le test ⇒ test inutile

✓ Self-validating // Auto-validé

- la validation est automatique
- les conditions de validation sont incluses dans le test

📉 Thorough // Complet

- les cas d’usage nominaux (*happy path*)
- les cas aux limites (*edge case*)
- les cas d’erreurs ultimes
  - base de données HS
  - HTTP 500 sur une API REST

En pratique

On a besoin au minimum :

- d'**une syntaxe** pour écrire définir les tests
- d’instructions pour vérifier les résultats : **les assertions**
- d'**un outil d’exécution**

Idéalement, on pourra aussi s’aider :

- d’un outil de mesure du code testé : **le coverage**
- d’instructions pour faciliter l’isolation : **les mocks**

Quelques outils

- Jest, pour *javascript, typescript*
- JUnit / AssertJ (assertions) / Jacoco (coverage) / Mockito (mocks) pour *java*

Certains langages récents intègrent nativement des outils de tests :

- Rust
- Elixir

Ok, et je met ça où ?

- ça dépend de votre outil
- généralement séparé du code de production ⇒ garder des limites claires
  ⇒ ne pas envoyer du code de test en production par erreur

Exemples :

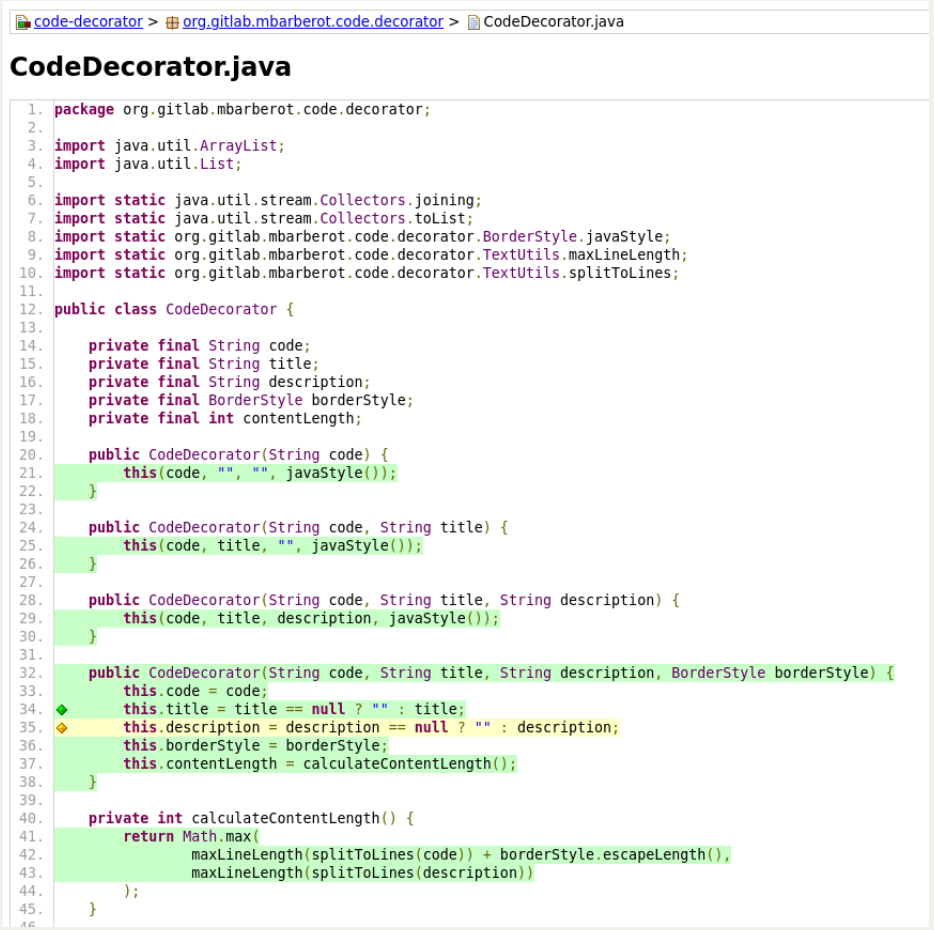
- Jest : `__tests__`
- Java : `src/test/java`

Exécution

```
mvn test
```

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running feature.DescriptionTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.362 s -- in feature.DescriptionTest
[INFO] Running feature.ContentTest
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.018 s -- in feature.ContentTest
[INFO] Running feature.SandboxTest
/* ***** Lorem ipsum ****************************** */
/* * This is a test                               * */
/* * With a multiline description                 * */
/* *                                              * */
/* * */ System.out.println('Code to decorate'); /* * */
/* ********************************************** */
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.007 s -- in feature.SandboxTest
[INFO] Running feature.BorderTest
[INFO] Running feature.BorderTest$BorderLeftTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.060 s -- in feature.BorderTest$BorderLeftTest
[INFO] Running feature.BorderTest$BorderBottomTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.007 s -- in feature.BorderTest$BorderBottomTest
[INFO] Running feature.BorderTest$BorderRightTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.013 s -- in feature.BorderTest$BorderRightTest
[INFO] Running feature.BorderTest$BorderTopTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.020 s -- in feature.BorderTest$BorderTopTest
[INFO] Tests run: 0, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.164 s -- in feature.BorderTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 18, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  6.588 s
[INFO] Finished at: 2025-05-07T22:28:37+02:00
[INFO] ------------------------------------------------------------------------
```

Mesure de la couverture du code


code-decorator > org.gitlab.mbarberot.code.decorator > CodeDecorator.java

CodeDecorator.java

```
1.  package org.gitlab.mbarberot.code.decorator;
2.
3.  import java.util.ArrayList;
4.  import java.util.List;
5.
6.  import static java.util.stream.Collectors.joining;
7.  import static java.util.stream.Collectors.toList;
8.  import static org.gitlab.mbarberot.code.decorator.BorderStyle.javaStyle;
9.  import static org.gitlab.mbarberot.code.decorator.TextUtils.maxLineLength;
10. import static org.gitlab.mbarberot.code.decorator.TextUtils.splitToLines;
11.
12. public class CodeDecorator {
13.
14.     private final String code;
15.     private final String title;
16.     private final String description;
17.     private final BorderStyle borderStyle;
18.     private final int contentLength;
19.
20.     public CodeDecorator(String code) {
21.         this(code, "", "", javaStyle());
22.     }
23.
24.     public CodeDecorator(String code, String title) {
25.         this(code, title, "", javaStyle());
26.     }
27.
28.     public CodeDecorator(String code, String title, String description) {
29.         this(code, title, description, javaStyle());
30.     }
31.
32.     public CodeDecorator(String code, String title, String description, BorderStyle borderStyle) {
33.         this.code = code;
34.         this.title = title == null ? "" : title;
35.         this.description = description == null ? "" : description;
36.         this.borderStyle = borderStyle;
37.         this.contentLength = calculateContentLength();
38.     }
39.
40.     private int calculateContentLength() {
41.         return Math.max(
42.                 maxLineLength(splitToLines(code)) + borderStyle.escapeLength(),
43.                 maxLineLength(splitToLines(description))
44.         );
45.     }
```


Allons-y !

Anatomie d’un test

Trois étapes :

- Arrange
- Act
- Assert

Anatomie d’un test

```
1      @Test
2      void rollDice_return1() {
3          /* Arrange */
4          var randomGenerator = mock(RandomGenerator.class);
5          when(randomGenerator.nextInt()).thenReturn(1);
6          var dice = new Dice(randomGenerator);
7
8          /* Act */
9          int result = dice.roll();
10
11         /* Assert */
12         assertThat(result).isEqualTo(1);
13     }
```

Les assertions

- pour vérifier le résultat obtenu
- idéalement une seule par test

Les assertions (exemple)

En JS/TS avec Jest (🌐 Documentation)

```
expect(age).toEqual(34)
expect(age).not.toBeLessThan(64)
expect(password).toMatch("[A-Za-z0-9]+")
```

En Java avec AssertJ (🌐 Documentation)

```
assertThat(age).isBetween(18, 100);
assertThat(wordList).containsExactlyInAnyOrder("foo", "border");
assertThat(password).matches("[a-zA-Z0-9+-$*!]+")
```

Le coverage

- mesure du code exercé par les tests
- utile pour détecter les zones non testées
- attention à la quête impossible des 100%

- Exemple de rapport de coverage

Les mocks

- permettent de donner rapidement une implémentation différente
- on peut faire des assertions sur son utilisation

Les mocks (exemple)

```java
@Test
void rollDice_return1_withMock() {
    /* Arrange */
    var randomGenerator = mock(RandomGenerator.class);
    when(randomGenerator.nextInt()).thenReturn(1);
    var dice = new Dice(randomGenerator);

    /* Act */
    int result = dice.roll();

    /* Assert */
    assertThat(result).isEqualTo(1);
    verify(randomGenerator, times(1)).nextInt();
}
```

- 🌐 Documentation Jest : Mock
- 🌐 Documentation Mockito : Mock

Les doublures de test

- permettent également de donner une implémentation différente
- avantage : réutilisable dans d'autres tests ou dans le code de production
- inconvénient : plus difficile de faire des assertions dessus

Les doublures de test (exemple)

```
@Test
void rollDice_return1_withTestDouble() {
    /* Arrange */
    RandomGenerator randomGenerator = () -> 1;
    var dice = new Dice(randomGenerator);

    /* Act */
    int result = dice.roll();

    /* Assert */
    assertThat(result).isEqualTo(1);
}
```

Maintenir le code de test

> *Vous devez mettre autant de soin dans le code de test que dans le code de production*
> *—Mathieu Barberot*

- Le code doit rester facile à lire
- Les code smells existent aussi dans les tests

Clarifier les assertions

- simplifier la syntaxe
- fusionner plusieurs assertions
- extraire le code dans une fonction

Exemple : Avant refactoring

```
test("splice can replace an element", () => {
    // Arrange
    const animals = ["Cat", "Dog", "Bird", "Lion", "Elephant", "Ant"]

    // Act
    const removedItems = animals.splice(3, 1, "Lizard")

    // Assert
    expect(removedItems).toEqual(["Lion"])
    expect(animals).not.toEqual(expect.arrayContaining(["Lion"]))
    expect(animals).toEqual(expect.arrayContaining(["Lizard"]))
})
```

Exemple : Après refactoring

```
test("splice can replace an element", () => {
    // Arrange
    const animals = ["Cat", "Dog", "Bird", "Lion", "Elephant", "Ant"]

    // Act
    const removedItems = animals.splice(3, 1, "Lizard")

    // Assert
    expectArray(animals, {
        removedItems: ["Lion"],
        addedItems: ["Lizard"],
    })
})
```

```
function expectArray(
    actual: string[], { removedItems, addedItems }: ExpectedArrayItemsChanges
) {
    expect(actual).not.toEqual(expect.arrayContaining(removedItems))
    expect(actual).toEqual(expect.arrayContaining(addedItems))
}
```

Eviter la duplication

- quand les tests sont quasiment des copier/coller

Exemple : Avant refactoring

```
@Test
void formatsDateToDayMonthYear() {
    // Arrange
    // Act
    String formatted = parse("2023-01-01").format(ofPattern("dd/MM/yyyy"));
    // Assert
    assertThat(formatted).isEqualTo("01/01/2023");
}

@Test
void formatsDateToMonthNameAndDay() {
    // Arrange
    // Act
    String formatted = parse("2024-12-25").format(ofPattern("MMMM d"));
    // Assert
    assertThat(formatted).isEqualTo("December 25");
}

@Test
void formatsDateToIso() { /* ... */ }
```

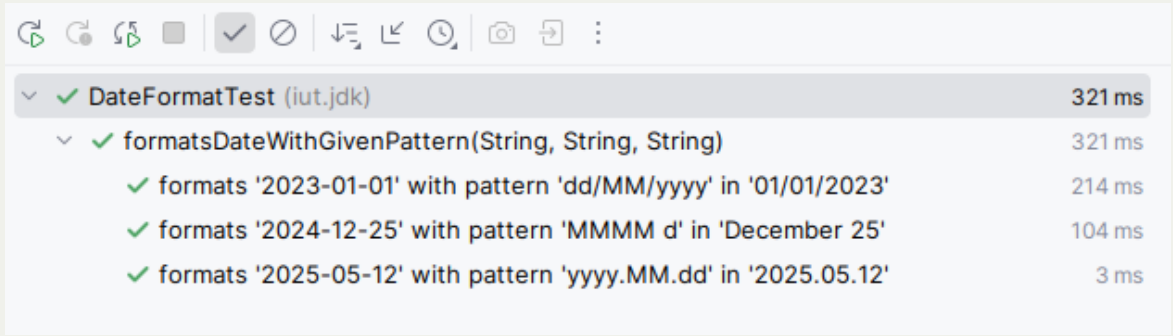
Exemple : Tests paramétrés

```
    @ParameterizedTest(name = "formats ''{0}'' with pattern ''{1}'' in ''{2}''")
    @CsvSource({
            "2023-01-01,    dd/MM/yyyy,     01/01/2023",
            "2024-12-25,    MMMM d,         December 25",
            "2025-05-12,    yyyy.MM.dd,     2025.05.12"
    })
    void formatsDateWithGivenPattern(String input, String pattern, String expecte
        // Arrange
        LocalDate date = parse(input);

        // Act
        String formatted = date.format(ofPattern(pattern));

        // Assert
        assertThat(formatted).isEqualTo(expected);
    }
```

Résultat


DateFormatTest (iut.jdk) 321 ms
formatsDateWithGivenPattern(String, String, String) 321 ms
formats '2023-01-01' with pattern 'dd/MM/yyyy' in '01/01/2023' 214 ms
formats '2024-12-25' with pattern 'MMMM d' in 'December 25' 104 ms
formats '2025-05-12' with pattern 'yyyy.MM.dd' in '2025.05.12' 3 ms

Factoriser `// Arrange`

- clarifier le setup du test
- réutiliser le code dans plusieurs tests

Exemple

```
@Test
void anItemCanBeAddedIntoACart() {
    // Arrange
    Cart cart = new Cart();

    // Act
    cart.addItem(new Item(1, "Keyboard", 100));

    // Assert
    assertThat(cart.getItems()).hasSize(1);
}
```

BeforeEach / AfterEach

```
private Cart cart;

@BeforeEach
void setUp() {
    cart = new Cart();
}

@Test
void anItemCanBeAddedIntoACartUsingBeforeEach() {
    // Arrange
    // Act
    cart.addItem(new Item(1, "Keyboard", 100));

    // Assert
    assertThat(cart.getItems()).hasSize(1);
}
```

BeforeAll / AfterAll

- ⚠ cela peut rompre l'indépendance de vos tests !
- peut être considéré comme un code smell

Limitations

- Duplication possible entre plusieurs suites de tests
- Séparation du code = moins de lisibilité
- Des tests peuvent ne pas utiliser la totalité du beforeEach

Factories

- Factorisation plus générique car réutilisable entre les suites
- Nommage des fonctions pour la lisibilité
- Plusieurs méthodes pour configurer le strict minimum à chaque test

Factories

```
class CartFactory {
    public static Cart createEmptyCart() {
        return new Cart();
    }
}

@Test
void multipleItemsCanBeAddedIntoACartUsingFactory() {
    // Arrange
    Cart emptyCart = CartFactory.createEmptyCart();

    // Act
    emptyCart.addItem(new Item(1, "Keyboard", 100));
    emptyCart.addItem(new Item(2, "Mouse", 50));

    // Assert
    assertThat(emptyCart.getItems()).hasSize(2);
}
```

Aller plus loin

- Les personas
  - pousser le concept de factories
  - incarner des types d'utilisateurs
  - requiert une coordination de l'équipe

Persona (Wikipedia)

Persona (exemple)

```
class LillyFactory {
    public static Cart createCart() {
        Cart cart = new Cart();
        cart.addItem(new Item(1, "Gamer Keyboard", 200));
        cart.addItem(new Item(2, "Gamer Mouse", 100));
        return cart;
    }
}

@Test
void lillyHasAnExpensiveCart() {
    // Arrange
    // Act
    Cart lillysCart = LillyFactory.createCart();

    // Assert
    assertThat(lillysCart.getTotal()).isGreaterThan(250);
}
```

Les tests d’intégration

Les critères

- Tester que les différentes parties sont bien branchées

Exemple:

- Je tourne le volant $\Rightarrow$ les roues tournent

La règle

- Comme les tests unitaires
- Mais avec moins de contraintes

F.I.R.S.T

- 🏃 Fast
  - Autant que possible
- 🏝️ Independant
  - On accepte des dépendances, mais maîtrisées
    - Utiliser une base de données éphèmere
    - Utiliser des faux services tiers…

F.I.R.S.T

- 🔁 Repeatable
  - Toujours !
- ✅ Self-validated
  - Toujours !
- 🌕 Thorough
  - Le *happy path* et des *edge cases*

Avec quels outils ?

Pour une API :

- Postman (ou équivalent)
- Playwright
- Hurl / Jetbrains Http Client
- RestAssured *pour Java*

Avec quels outils ?

Pour une SPA

- Cypress
- Playwright
- Jest

Tester une API REST

Avec Postman :

1. Documenter avec Open API / Swagger

```
@Operation(summary = "Create a rover")
@APIResponse(description = "success", responseCode = "201")
@POST
@Produces(MediaType.APPLICATION_JSON)
fun createRover(roverCreationDto: RoverCreationDto): RestResponse<String> {
    val rover = roverRepository.create(roverCreationDto.name)
    return ResponseBuilder
        .created<String>(URI.create("/api/rover/${rover.id}"))
        .build()
}
```

Tester une API REST

Avec Postman :

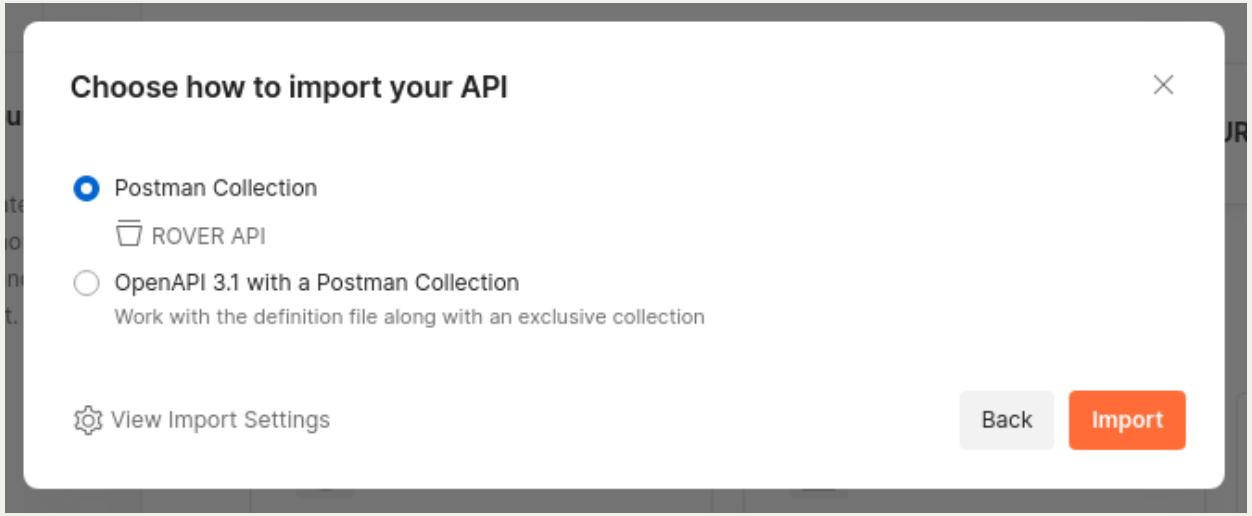
1. Documenter avec Open API / Swagger

```
@Schema(description = "Data to create a rover")
data class RoverCreationDto(
    @Schema(description = "Name of the rover", required = true)
    val name: String
)
```

Tester une API REST

Avec Postman :

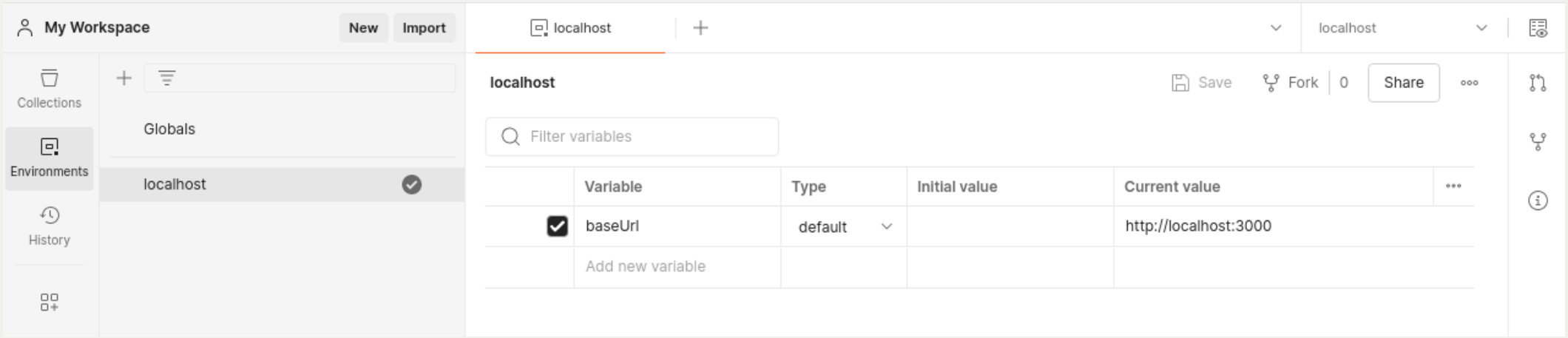
1. Documenter avec Open API / Swagger
2. Importer dans le client REST



Tester une API REST

Avec Postman :

1. Documenter avec Open API / Swagger
2. Importer dans le client REST, puis configurer

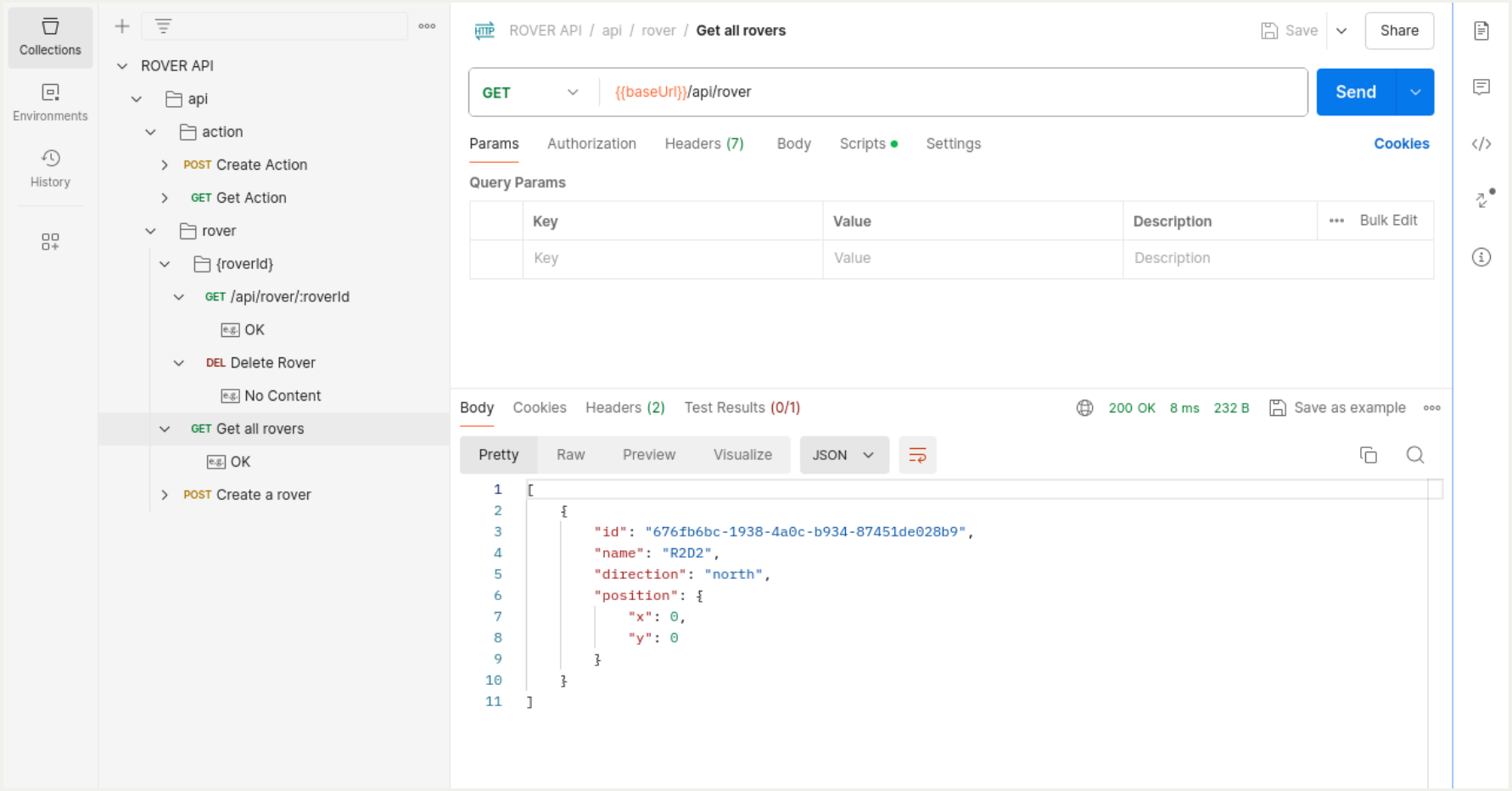


Tester une API REST

Avec Postman :

1. Documenter avec Open API / Swagger
2. Importer dans le client REST
3. Exécuter une suite de requête

Tester une API REST

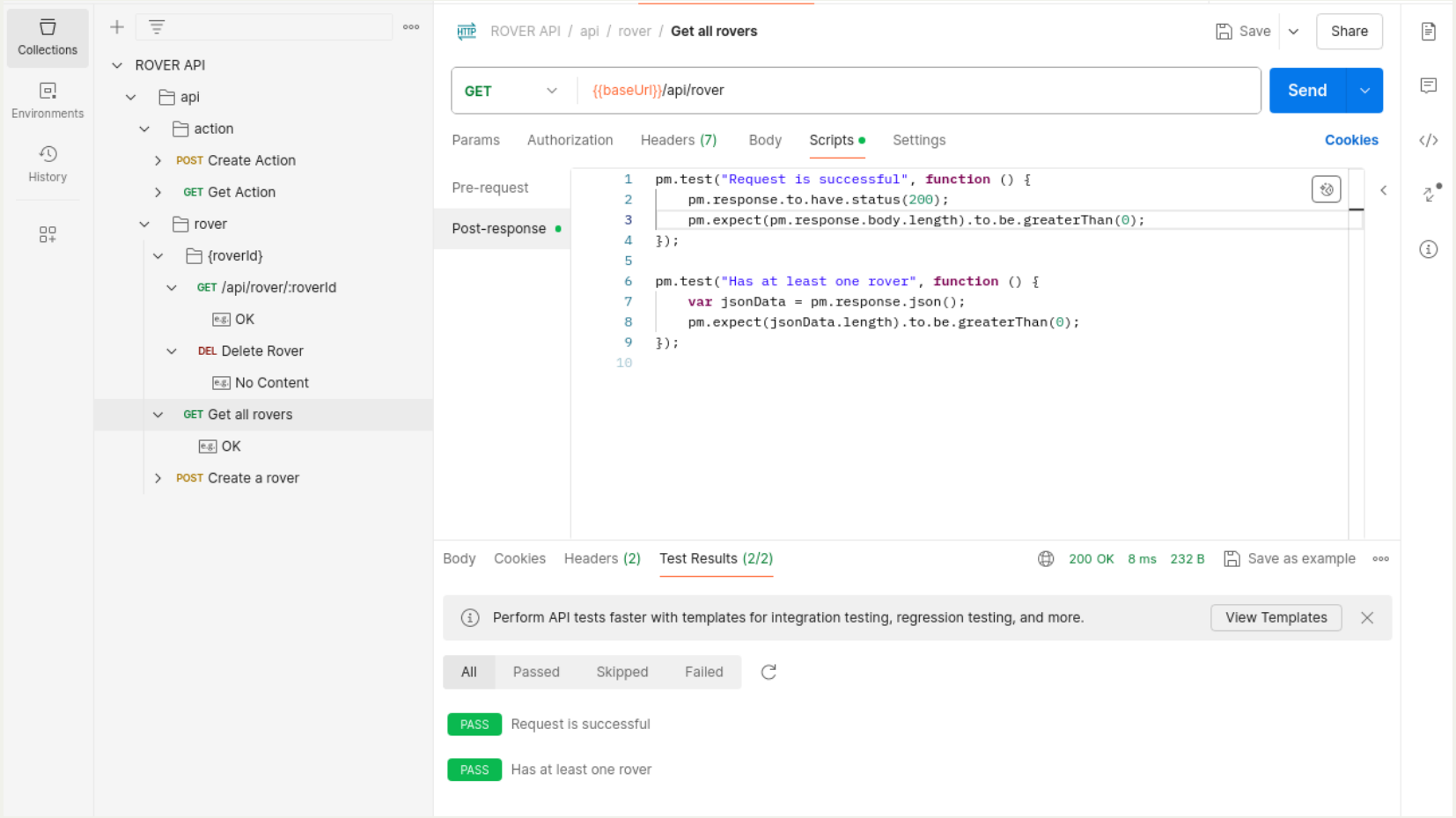


Tester une API REST

Avec Postman :

1. Documenter avec Open API / Swagger
2. Importer dans le client REST
3. Exécuter une suite de requête
4. Écrire un test

Tester une API REST



Tester une SPA

Avec Playwright:

1. Installer et initialiser Playwright [1]

```
npm init playwright@latest
```

---

1. https://playwright.dev/docs/intro

Tester une SPA

Avec Playwright:

1. Installer et initialiser Playwright
2. Ecrire un test

```
test('Home page has the right title in page metadata', async ({ page }) => {
  // Arrange
  // Act
  await page.goto('/');

  // Assert
  await expect(page).toHaveTitle("Keyboard Factory");
});
```

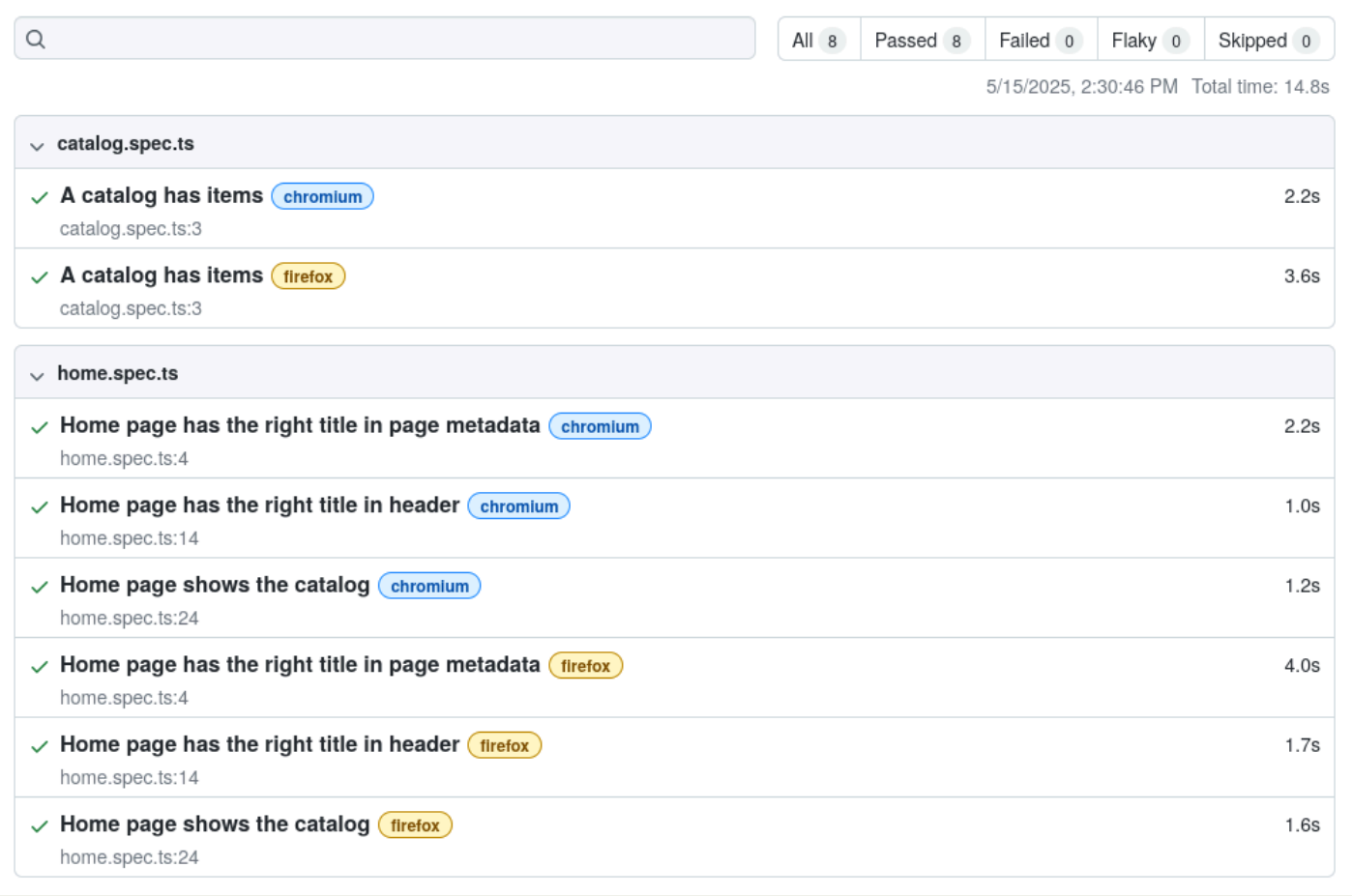
Tester une SPA

Avec Playwright:

1. Installer et initialiser Playwright
2. Ecrire un test
3. Exécuter une suite de test

```
npx playwright test
```

Tester une SPA



Merci de votre attention