

Projet Test & Simulation

Introduction

Notre projet en Tests et Simulation s'inscrit dans la création d'une solution logicielle novatrice, combinant un robot physique à un système de supervision via une API REST. L'objectif central est d'optimiser les opérations automatisées tout en offrant une gestion centralisée. Ce rapport explore nos choix technologiques, notre architecture système, et met en lumière notre approche méthodique des tests. Nous partagerons nos réussites, défis et perspectives pour conclure sur une démonstration complète de nos compétences en développement logiciel.

Choix des outils et techniques

Langage :Java

Framework : Spring

Base de Données : PostgreSQL

Tests : JUnit, Postman

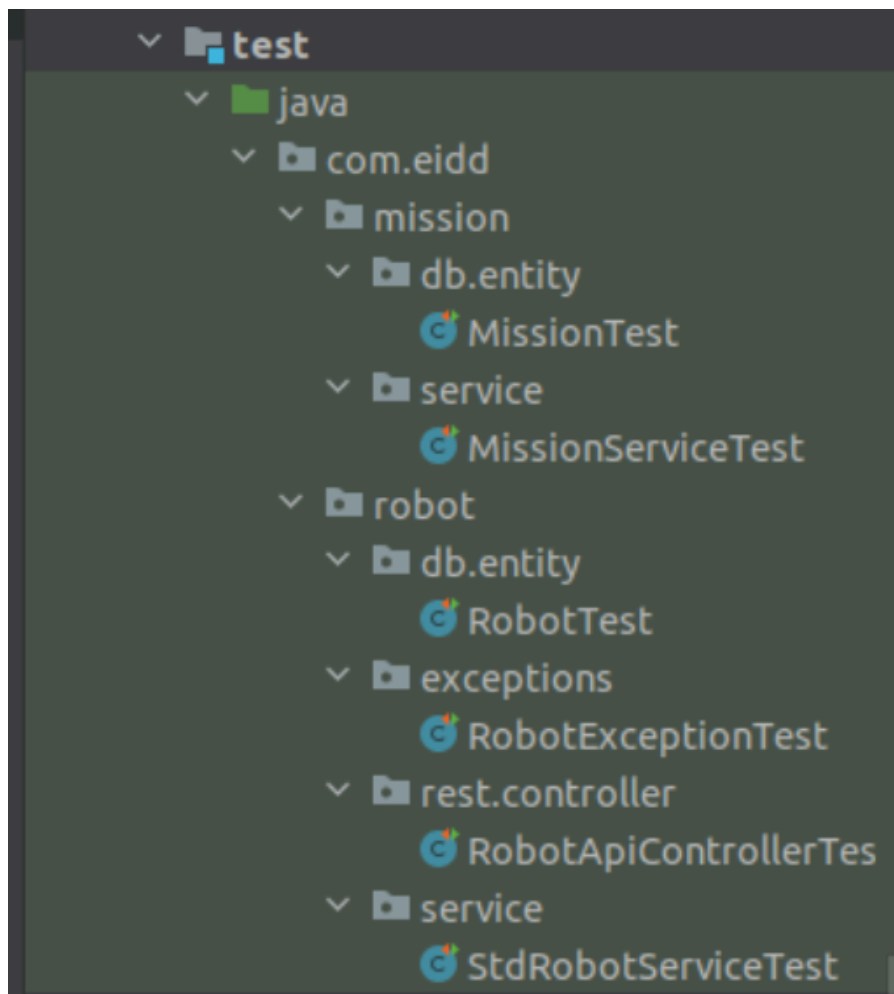
IDE : IntelliJ IDEA

Architecture Logicielle

Notre solution, développée en Java avec le framework Spring, adopte une architecture modulaire et évolutive. Les entités Java, telles que la classe Robot, sont conçues pour une gestion efficace des données, et la base de données utilise JPA pour garantir une représentation précise. Les services dédiés facilitent l'interaction entre les composantes du système.

L'architecture REST est essentielle, permettant une communication standardisée entre le robot et le système de supervision. Les points de terminaison de l'API REST simplifient l'échange de données. Cette approche offre une solution flexible et performante.

Tests Unitaires



Tests Unitaires pour la Classe entity Mission

La classe `Mission` représente une entité essentielle dans notre système, décrivant les caractéristiques d'une mission attribuée à un robot. Pour garantir la robustesse et la fiabilité de cette classe, des tests unitaires ont été élaborés en utilisant le framework de test `JUnit`. Ces tests visent à évaluer le bon fonctionnement des méthodes de la classe, en s'assurant que les attributs

sont correctement définis, les méthodes d'accès fonctionnent comme prévu, et les mécanismes d'égalité et de hachage sont implémentés correctement.

Test de Construction et d'Accès aux Attributs

Le premier ensemble de tests se concentre sur la construction d'objets `Mission` et la vérification des méthodes d'accès aux attributs. Nous utilisons la bibliothèque `AssertJ` pour formuler des assertions concises et expressives.

```
package com.eidd.mission.db.entity;

import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;

class MissionTest {

    @Test
    public void testing_class() {
        // given
        int id = 1;
        double x = 0.0;
        double y = 0.0;
        double theta = 0.0;

        Mission mission = new Mission(id, x, y, theta);
        assertThat(mission.getId()).isEqualTo(id);
        assertThat(mission.getId()).isEqualTo(id);
        assertThat(mission.getId()).isEqualTo(id);
        assertThat(mission.getId()).isEqualTo(id);
    }
}
```

Test des Méthodes Equals et hashCode

Un autre ensemble de tests évalue les méthodes `equals` et `hashCode` de la classe `Mission`. Ces méthodes sont cruciales pour garantir une comparaison correcte des objets.

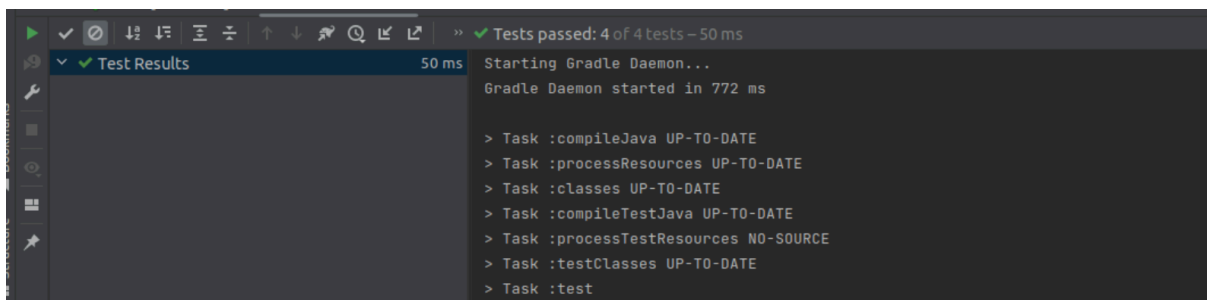
```

@Test
void equalsAndHashCode() {
    // given
    Mission mission1 = new Mission( id: 1, x: 10.0, y: 20.0, theta: 30.0);
    Mission mission2 = new Mission( id: 1, x: 10.0, y: 20.0, theta: 30.0);
    Mission mission3 = new Mission( id: 2, x: 15.0, y: 25.0, theta: 35.0);

    // then
    assertThat(mission1).isEqualTo(mission2);
    assertThat(mission1.hashCode()).isEqualTo(mission2.hashCode());
    assertThat(mission1).isNotEqualTo(mission3);
}

```

Résultat de l'ensemble des tests effectués sur la classe Mission



```

> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test

```

Tests Unitaires pour la Classe MissionService

1. Test de la Méthode getAll

Ce test évalue la méthode `getAll` du service en simulant le comportement du repository `MissionRepository`. En utilisant Mockito, une liste fictive de missions est créée, et le test vérifie que la méthode `getAll` retourne effectivement ces missions. De plus, le test valide que la méthode du repository associé, `findAll`, est appelée comme attendu.

```

@Test
void getAll() {
    //verify(missionRepository).findAll();
    // Arrange
    List<Mission> mockMissions = Collections.singletonList(new Mission(id: 1, x: 0.0, y: 0.0, theta: 0.0));
    Mockito.when(missionRepository.findAll()).thenReturn(mockMissions);

    List<Mission> result = missionService.getAll();

    assertThat(result).isEqualTo(mockMissions);
    verify(missionRepository).findAll();
}

```

2. Test de la Méthode **create**

Ce test examine la capacité du service à créer de nouvelles missions avec la méthode **create**. En utilisant Mockito, le comportement du repository est simulé pour garantir que la méthode **save** est correctement appelée avec la mission fournie. Le test vérifie ensuite que la mission retournée par le service correspond à celle créée.

```

@Test
void create() {
    Mission missionToCreate_1= new Mission(id: 1, x: 4.0, y: 2.0, theta: 10.0);
    Mockito.when(missionRepository.save(Mockito.any(Mission.class))).thenReturn(missionToCreate_1);

    //Act
    Mission createdMission_1= missionService.create(missionToCreate_1);

    //assert
    assertThat(createdMission_1).isEqualTo(missionToCreate_1);

    //verify
    Mockito.verify(missionRepository).save(missionToCreate_1);
}

```

3. Gestion des Exceptions

La classe **MissionService** doit être capable de gérer les exceptions de manière appropriée. Le test simule une exception lors de l'appel de la méthode **findAll** du repository. Il vérifie que le service capture correctement cette exception et la relance sous forme de **RobotException**, assurant ainsi un comportement robuste en cas d'erreur.

```

@Test
void get_all_should_catch_exception() {
    //doThrow(RobotException.class).when(missionRepository).findAll();
    doThrow(RobotException.class).when(missionRepository).findAll();
    //assertThatCode(missionService::getAll).isInstanceOf(RobotException.class);
    assertThatCode(missionService::getAll).isInstanceOf(RobotException.class);
}

```

4. Test de la Méthode `getMission`

Ce test évalue la méthode `getMission` du service en utilisant une liste simulée de missions. Il s'assure que le service retourne correctement les missions suivantes dans l'ordre prévu, et qu'il revient à la première mission après avoir atteint la fin de la liste. Cela garantit que la méthode `getMission` fonctionne conformément aux attentes.

```

@Test
void getMission() {
    List<Mission> missions = Arrays.asList(
        new Mission( id: 1, x: 4.0, y: 2.0, theta: 10.0),
        new Mission( id: 2, x: 5.0, y: 3.0, theta: 20.0)
    );

    Mockito.when(missionRepository.findAll()).thenReturn(missions);
    missionService.init();
    //Testing getMission
    Mission mission1 = missionService.getMission();
    Mission mission2 = missionService.getMission();
    // Asserting that missions are not null
    assertNotNull(mission1);
    assertNotNull(mission2);

    // Asserting that the returned missions are the expected ones
    assertEquals( expected: 1, mission1.getId());
    assertEquals( expected: 4.0, mission1.getX());
    assertEquals( expected: 2.0, mission1.getY());
    assertEquals( expected: 10.0, mission1.getTheta());

    assertEquals( expected: 2, mission2.getId());
    assertEquals( expected: 5.0, mission2.getX());
    assertEquals( expected: 3.0, mission2.getY());
    assertEquals( expected: 20.0, mission2.getTheta());
}
}

```

Tests Unitaires pour la Classe entity Robot

La classe de test **RobotTest** vise à évaluer le comportement des méthodes `equals` et `hashCode` de la classe `Robot`. Ces méthodes sont essentielles pour garantir la cohérence et la précision lors de la comparaison et du stockage des objets `Robot`. Voici une description détaillée de chaque test :

1. Test d'égalité des robots (testEquals) :

Ce test vérifie la méthode `equals` de la classe `Robot`. Deux robots sont considérés égaux s'ils ont les mêmes valeurs pour tous leurs attributs (`id`, `x`, `y`, `theta`, `v`, `ultraSound`).

- Cas de test 1 : Deux robots avec les mêmes attributs sont considérés égaux.
 - Procédure : Deux robots (robot1 et robot2) sont créés avec des attributs identiques.
 - Assertion : La méthode **assertEquals** vérifie que **robot1** est égal à **robot2**.
- Cas de test 2 : Deux robots avec un attribut différent ne sont pas considérés égaux.
 - Procédure : Deux robots (robot1 et robot3) sont créés avec des valeurs différentes pour l'attribut `id`.
 - Assertion : La méthode **assertNotEquals** vérifie que robot1 n'est pas égal à robot3.

```
public class RobotTest {  
  
    @Test  
    void testEquals() {  
        Robot robot1 = new Robot( id: 1, x: 2.0, y: 3.0, theta: 45.0, v: 5.0, ultraSound: 10.0);  
        Robot robot2 = new Robot( id: 1, x: 2.0, y: 3.0, theta: 45.0, v: 5.0, ultraSound: 10.0);  
        Robot robot3 = new Robot( id: 2, x: 4.0, y: 6.0, theta: 90.0, v: 7.0, ultraSound: 15.0);  
  
        // Test equal robots  
        assertEquals(robot1, robot2);  
        assertNotEquals(robot1, robot3);  
    }  
}
```

2. Test du code de hachage des robots (testHashCode) :

Ce test évalue la méthode **hashCode** de la classe `Robot`, qui doit générer un code de hachage cohérent pour des objets égaux.

- **Cas de test 1** : Deux robots égaux génèrent le même code de hachage.

- Procédure : Deux robots (`robot 1` et `robot 2`) sont créés avec des attributs identiques.

- Assertion : La méthode `assertEquals` vérifie que le code de hachage de `robot 1` est égal au code de hachage de `robot 2`.

- Cas de test 2 : Deux robots non égaux ne génèrent pas le même code de hachage.

- Procédure : Deux robots (`robot 1` et `robot 3`) sont créés avec des valeurs différentes pour l'attribut `id`.

- Assertion : La méthode `assertEquals` vérifie que le code de hachage de `robot 1` n'est pas égal au code de hachage de `robot 3`.

```
@Test
void testHashCode() {
    Robot robot1 = new Robot( id: 1, x: 2.0, y: 3.0, theta: 45.0, v: 5.0, ultraSound: 10.0)
    Robot robot2 = new Robot( id: 1, x: 2.0, y: 3.0, theta: 45.0, v: 5.0, ultraSound: 10.0)
    Robot robot3 = new Robot( id: 2, x: 4.0, y: 6.0, theta: 90.0, v: 7.0, ultraSound: 15.0)

    // Test hash code for equal and non-equal robots
    assertEquals(robot1.hashCode(), robot2.hashCode());
    assertEquals(robot1.hashCode(), robot3.hashCode());
}
```

Tests Unitaires pour la Classe RobotExceptions

- Deux robots (`robot1` et `robot2`) sont créés avec des attributs identiques.

- La méthode `assertEquals` vérifie que le code de hachage de `robot1` est égal au code de hachage de robot2.

- Cas de test 2: Deux robots non égaux ne génèrent pas le même code de hachage.

- Deux robots (`robot1` et `robot3`) sont créés avec des valeurs différentes pour l'attribut `id`.

- La méthode `assertNotEquals` vérifie que le code de hachage de `robot1` n'est pas égal au code de hachage de `robot3`.

Ces tests garantissent que la classe `Robot` implémente correctement les méthodes `equals` et `hashCode`, assurant ainsi une comparaison et un stockage corrects lorsqu'elle est utilisée dans des collections telles que des ensembles (`Set`) ou des cartes (`Map`).


```

public class RobotExceptionTest {

    @Test
    public void testRobotExceptionWithMessage() {
        RobotException exception = assertThrows(RobotException.class, () -> {
            throw new RobotException("Test message");
        });
        assertEquals("Test message", exception.getMessage());
        assertNull(exception.getDetail());
    }
}

```

1. Ce test vérifie la création d'une instance de `RobotException` en utilisant uniquement un message. En lançant une exception avec un message spécifique, le test utilise ensuite `assertThrows` pour s'assurer que l'exception capturée est bien une `RobotException`. Enfin, il vérifie que le message de l'exception correspond à celui spécifié lors de la création, et que la propriété `detail` de l'exception est nulle.

```

@Test
public void testRobotExceptionWithMessageAndCause() {
    Throwable cause = new IllegalArgumentException("Test cause");
    RobotException exception = assertThrows(RobotException.class, () -> {
        throw new RobotException("Test message", cause);
    });
    assertEquals("Test message", exception.getMessage());
    assertEquals(cause, exception.getCause());
    assertNull(exception.getDetail());
}

```

2. Ce test examine la création d'une `RobotException` avec un message et une cause. Il utilise `assertThrows` pour s'assurer que l'exception capturée est bien une `RobotException`. Le test vérifie ensuite que le message de l'exception correspond à celui spécifié lors de la création, que la cause de l'exception est celle spécifiée, et que la propriété `detail` est nulle.

```

}

@Test
public void testRobotExceptionWithMessageAndDetail() {
    RobotException exception = assertThrows(RobotException.class, () -> {
        throw new RobotException("Test message", "Test detail");
    });
    assertEquals("Test message", exception.getMessage());
    assertEquals("Test detail", exception.getDetail());
}

```

3. Ce test évalue la création d'une `RobotException` avec un message et un détail. En utilisant `assertThrows`, le test s'assure que l'exception capturée est de type `RobotException`. Il vérifie ensuite que le message de l'exception correspond à celui spécifié et que le détail de l'exception est correct.

```

@Test
public void testRobotExceptionWithMessageCauseAndDetail() {
    Throwable cause = new IllegalArgumentException("Test cause");
    RobotException exception = assertThrows(RobotException.class, () -> {
        throw new RobotException("Test message", cause, "Test detail");
    });
    assertEquals("Test message", exception.getMessage());
    assertEquals(cause, exception.getCause());
    assertEquals("Test detail", exception.getDetail());
}

```

4. Ce test explore la création d'une `RobotException` avec un message, une cause et un détail. À l'aide de `assertThrows`, il s'assure que l'exception capturée est une `RobotException`. Le test vérifie que le message, la cause et le détail de l'exception correspondent aux valeurs spécifiées lors de la création. En résumé, ces tests garantissent le bon fonctionnement des différentes façons de créer une `RobotException` et assurent que les propriétés telles que le message, la cause et le détail sont correctement attribuées.

La Classe Test RobotApiControllerRest

La classe de test `RobotApiControllerTest` évalue le comportement du contrôleur REST `RobotApiController`, qui semble être dédié à la gestion des robots. Plusieurs fonctions de test ont été mises en place pour garantir le bon fonctionnement de diverses opérations. La première fonction, `robotGet`, assure que la récupération de la liste des robots renvoie un code HTTP OK, en simulant la réponse du service de robot via Mockito.

La deuxième fonction, `robotIdDelete_RobotNotFound`, vérifie que la suppression d'un robot introuvable génère un code HTTP NOT_FOUND en simulant une exception d'absence du robot dans le service.

La troisième fonction, `robotIdGet_RobotFound`, s'assure que la récupération d'un robot existant retourne le robot avec un code HTTP OK.

La quatrième fonction, `robotIdPut_RobotException`, valide que la mise à jour d'un robot lance une exception appropriée et renvoie un code HTTP NOT_FOUND. Enfin, la cinquième fonction, `robotPost`, teste la création d'un robot, confirmant que le contrôleur retourne le robot créé avec un code HTTP OK. Ces tests couvrent divers aspects de la logique métier du contrôleur, garantissant son bon fonctionnement dans des scénarios variés.

La Classe Test StdRobotServiceTest

La classe de test `StdRobotServiceTest` évalue la fonctionnalité du service `StdRobotService` dédié à la gestion des opérations liées aux robots. Les différentes fonctions de test implémentées couvrent divers scénarios.

La première fonction, `getAll_shouldReturnListOfRobots`, assure que la récupération de la liste des robots fonctionne correctement en simulant la réponse du repository avec Mockito. La deuxième fonction, `find_shouldReturnRobotById`, teste la recherche d'un robot par son identifiant, garantissant que le service retourne le robot attendu. La troisième

fonction, ``create_shouldReturnCreatedRobot``, vérifie que la création d'un robot est correctement traitée par le service.

La quatrième fonction, ``update_shouldReturnUpdatedRobot``, s'assure que la mise à jour d'un robot est gérée avec succès.

La cinquième fonction, ``delete_shouldNotThrowException``, confirme que la suppression d'un robot ne génère pas d'exception.

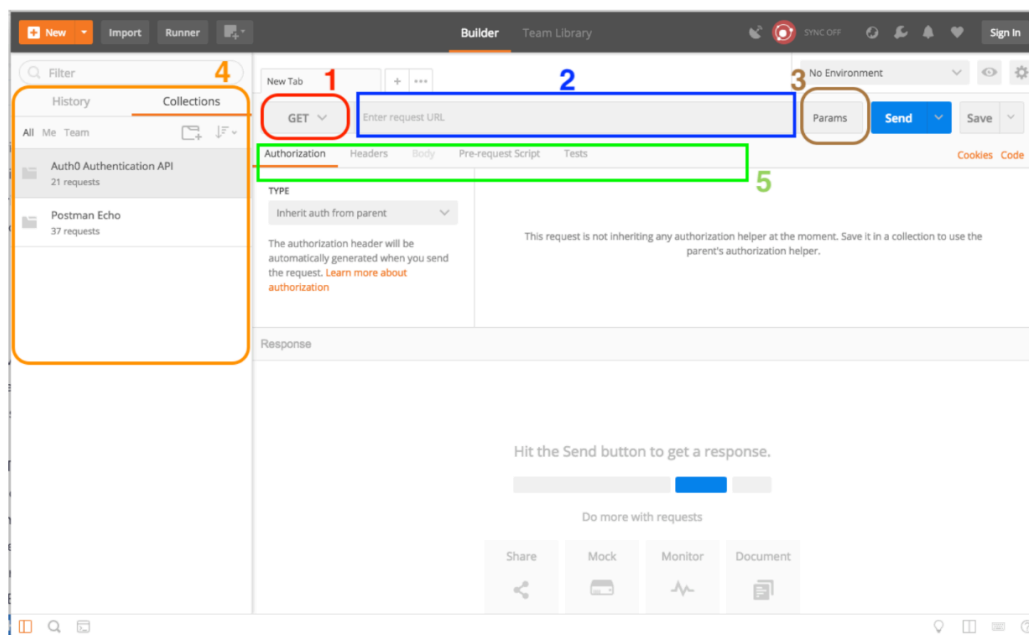
Enfin, la sixième fonction, ``update_whenRobotNotFound_shouldThrowException``, teste le scénario où la mise à jour d'un robot non trouvé déclenche une exception de type ``RobotException``. Ensemble, ces tests garantissent la robustesse et le bon fonctionnement du service ``StdRobotService`` dans divers contextes opérationnels.

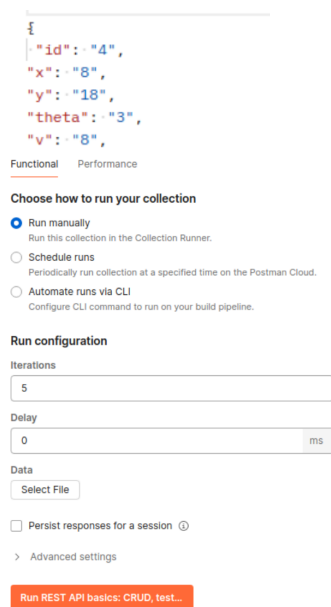
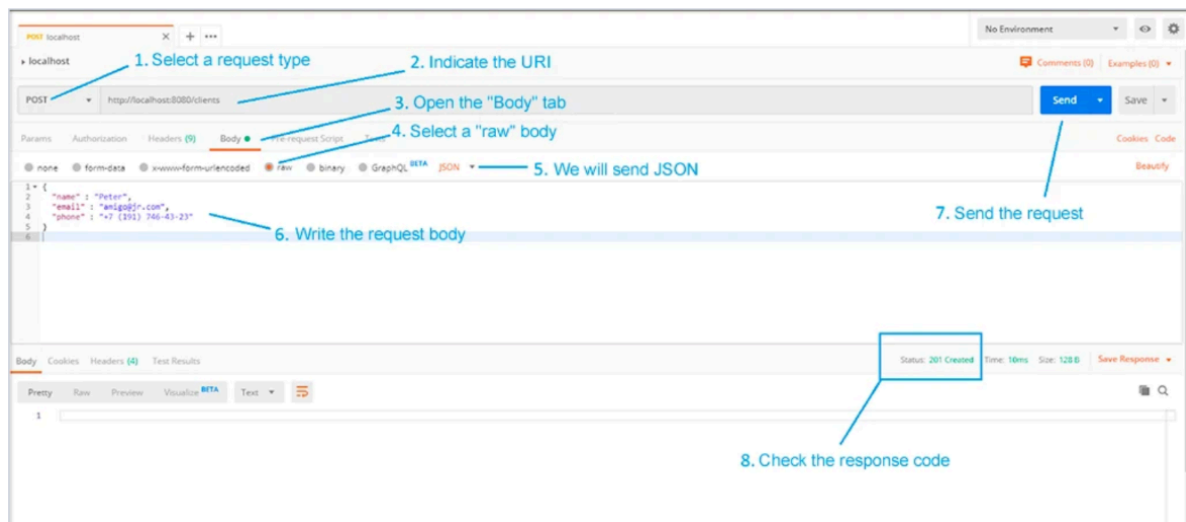
Test de validation avec Postman

Nous avons utilisé POSTMAN pour tester les services WEB RESTful de notre projet.

Nous avons pris connaissance des fonctionnalités à connaître pour nos test :

- Choix du type de requête à envoyer : GET,POST,PUT
- URL de l'API : `http://localhost:8085/Produits`
- Les paramètres à passer avec l'URL (Robots ou Robot 1)
- Corps d'une Requête HTTP(head,body,etc)





REST API basics: CRUD, test... - Run results Run Again Automate Run + New Run 📄 Export Results

Ran today at 08:13:38 - [View all runs](#)

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	Beta	5	3s 273ms	20	125 ms

All Tests Passed (15) Failed (5) Skipped (0) View Summary

- DELETE** Delete data
https://postman-rest-api-learner.glitch.me/info?id=1

200 OK 129 ms 267 B

PASS Successful DELETE request
- Iteration 2

GET Get data
https://postman-rest-api-learner.glitch.me/info?id=1

200 OK 119 ms 249 B

PASS Status code is 200
- GET** Post data
http://localhost:8085/robot/2

404 Not Found 6 ms 82 B

FAIL Successful POST request | AssertionError: expected 404 to be one of [200, 201]
- PUT** Update data
https://postman-rest-api-learner.glitch.me/info?id=1

200 OK 218 ms 333 B

PASS Successful PUT request
- DELETE** Delete data
https://postman-rest-api-learner.glitch.me/info?id=1

200 OK 121 ms 267 B

PASS Successful DELETE request
- Iteration 3

GET Get data
https://postman-rest-api-learner.glitch.me/info?id=1

200 OK 138 ms 249 B

PASS Status code is 200
- GET** Post data
http://localhost:8085/robot/2

404 Not Found 15 ms 82 B

Nous pouvons constater que nous avons 15 tests réussis contre 5 tests qui ont échoué.

CONCLUSION ET PERSPECTIVE

Au cours de ce projet, nous avons mis en place une architecture robuste et évolutive permettant la gestion efficace des robots. La classe `StdRobotService` offre un ensemble de fonctionnalités pour interagir avec la base de données et manipuler les entités robot. Les tests unitaires associés, présents dans la classe `StdRobotServiceTest`, garantissent la fiabilité et la stabilité des fonctionnalités principales.

La gestion des exceptions, notamment avec la classe `RobotException`, contribue à la robustesse du système en identifiant et en traitant les erreurs de manière explicite. Les tests unitaires associés à cette classe assurent la gestion appropriée des exceptions dans divers scénarios.

La classe d'entité `Robot` est bien définie et comprend des méthodes générées telles que `equals`, `hashCode`, et `toString`, garantissant une manipulation sûre des objets Robot au sein du système.

Au cours du développement, nous avons été confrontés à des défis tels que la gestion des verrous de base de données (`DB locks`) et les exceptions liées à la coordination des tests unitaires pour assurer une couverture exhaustive. Ces défis ont été surmontés grâce à des approches méticuleuses malgré que le projet ait été effectué de façon individuelle.

Dans les perspectives futures, le projet ouvre la voie à plusieurs perspectives d'amélioration. Des fonctionnalités telles que la gestion des mises à jour en masse, l'extension des tests pour couvrir davantage de cas, et l'exploration de nouvelles fonctionnalités robotiques pourraient être envisagées dans les versions futures du système.

