

Projet Développement JAVA V 2023-2024



Produced by Popadiuc Claudiu

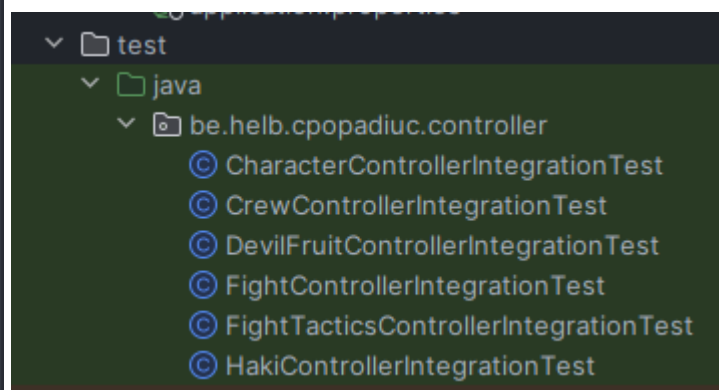
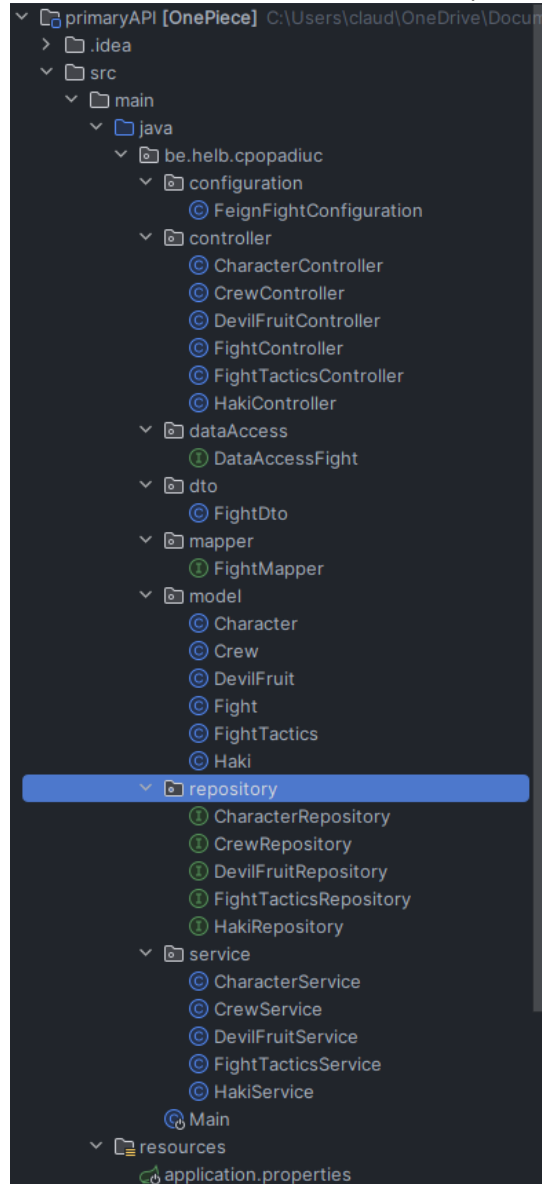
Supervise by Mr Bilal Ben Abdelkader

Table des matières

1. Architecture Générale	3
2. Choix Technique	6
3. Utilisation de l'API	9

1. Architecture Générale

L'API One-Piece est une application Spring Boot conçue pour gérer les informations sur les personnages de l'univers One-Piece. Elle utilise une base de données PostgreSQL pour stocker les données des personnages et fournit divers points d'accès pour effectuer des opérations CRUD sur les personnages. De plus, elle inclut des fonctionnalités telles que la récupération des personnages par équipage, travail, rang, et l'initiation de combats entre les personnages dans une api secondaire. Je vais vous présenter l'architecture générale du projet et vous montrer brièvement comment cela se présente :



Dans le projet, vous pouvez observer un package "controller" qui est responsable du traitement des requêtes HTTP et de l'interaction avec la couche de service. Il met en œuvre les opérations CRUD ainsi que des fonctionnalités supplémentaires telles que l'ajout de personnages, d'équipages, de fruits du démon, de styles de combat, combat et des hakis.

Un autre package est dédié au modèle ("Model"), lequel définit la structure de données d'un personnage, y compris ses attributs et ses relations avec d'autres entités, telles que "one-to-many" ou "many-to-one".

Il existe également un package "Repository" qui utilise Spring Data JPA pour fournir des opérations CRUD pour l'entité "Character" et les autres modèles. Il inclut des méthodes de requête personnalisées permettant de récupérer les personnages en fonction de différents critères.

Le package "Service" contient la logique métier pour les opérations liées aux contrôleurs. Il gère les interfaces entre le contrôleur et le référentiel, prenant en charge les transactions et les traitements supplémentaires.

Un autre fichier à noter est la configuration de la base de données ("application.properties") qui configure les propriétés de connexion à la base de données, telles que l'URL, le nom d'utilisateur et le mot de passe. Il définit également la configuration automatique DDL d'Hibernate pour les mises à jour du schéma, et dans notre cas, nous utiliserons pgAdmin 4.

Le reste des fichiers est nécessaire pour l'API secondaire, tels que la configuration, l'accès aux données, les objets de transfert de données (DTO) et le mapping.

Enfin, pour conclure, nous avons les tests d'intégration qui seront utilisés pour tester les endpoints de l'API. Nous aurons une classe de test dédiée à chaque modèle.

Quant à l'api secondaire elle suit le même principe que la base.

Dans mon model j'ai aussi different type de relation :

Personnage - Fruit du Démon (One-to-One)

Description : Un personnage peut avoir au plus un Fruit du Démon, ou aucun (null). De même, un Fruit du Démon peut être possédé par un personnage ou aucun.

Annotations :

@ManyToOne dans la classe Character avec @JoinColumn(name = "devilfruit_id")

@ManyToOne(mappedBy = "devilFruit") dans la classe DevilFruit avec @JsonIgnore

Personnage - Équipage (Many-to-One)

Description : De nombreux personnages peuvent appartenir à un équipage. Chaque personnage est associé à au plus un équipage.

Annotations :

@ManyToOne dans la classe Character avec @JoinColumn(name = "crew_id")

@ManyToOne(mappedBy = "crew") dans la classe Crew avec @JsonIgnore

Personnage - Haki (Many-to-One)

Description : De nombreux personnages peuvent posséder un type de Haki. Chaque personnage est associé à au plus un type de Haki qui sera son meilleurs haki.

Annotations :

@ManyToOne dans la classe Character avec @JoinColumn(name = "haki_id")

@ManyToOne(mappedBy = "haki") dans la classe Haki avec @JsonIgnore

Personnage - Tactiques de Combat (Many-to-Many)

Description : De nombreux personnages peuvent utiliser plusieurs tactiques de combat, et une tactique de combat peut être utilisée par plusieurs personnages.



Annotations :

@ManyToMany dans la classe Character avec @JoinTable spécifiant la table de mappage

@ManyToMany(mappedBy = "fightTactics") dans la classe FightTactics avec @JsonIgnore

Équipage - Personnage (One-to-Many)

Description : Un équipage peut avoir plusieurs personnages, mais chaque personnage appartient à exactement un équipage.

Annotations :

@OneToMany(mappedBy = "crew") dans la classe Crew avec @JsonIgnore

Fruit du Démon - Personnage (One-to-One)

Description : Un Fruit du Démon est associé à au plus un personnage, et un personnage peut avoir au plus un Fruit du Démon.

Annotations :

@OneToOne(mappedBy = "devilFruit") dans la classe DevilFruit avec @JsonIgnore

Tactiques de Combat - Personnage (Many-to-Many)

Description : De nombreux personnages peuvent utiliser plusieurs tactiques de combat, et une tactique de combat peut être employée par plusieurs personnages.

Annotations :

@ManyToMany(mappedBy = "fightTactics") dans la classe FightTactics avec @JsonIgnore

Haki - Personnage (One-to-Many)

Description : Un type de Haki peut être possédé par plusieurs personnages, mais chaque personnage est associé à au plus un type de Haki.

Annotations :

@OneToMany(mappedBy = "haki") dans la classe Haki avec @JsonIgnore

2. Choix Technique

Dans cette section du document, je vais aborder les choix techniques effectués dans le cadre de mon application. Pour débiter, j'ai opté pour Spring Boot, principalement en raison de son imposition mais également en raison de sa simplicité, de sa convention par rapport à la configuration et de son support intégré pour les applications web.

Concernant la base de données, j'ai choisi PostgreSQL en raison de ses capacités en tant que base de données relationnelle et de sa compatibilité avec Spring Data JPA. Cette combinaison offre une gestion efficace des entités et simplifie les opérations de persistance.

Voici une explication brève des principales dépendances utilisées dans mon projet :

Spring Boot Starter Tomcat :

Description : Fournit un serveur web Tomcat intégré à Spring Boot.

Raison : Utile pour le déploiement d'applications web Spring Boot.

Spring Boot Starter Web

Description : Contient les dépendances nécessaires pour développer des applications web avec Spring Boot.

Raison : Nécessaire pour créer des API REST et des applications web.

Spring Boot Starter Test :

Description : Contient les dépendances pour les tests unitaires et d'intégration dans les projets Spring Boot.

Raison : Facilite l'écriture de tests automatisés pour assurer la qualité du code.

Spring Boot Starter Data JPA:

Description : Facilite l'intégration de Spring Data JPA pour l'accès aux bases de données relationnelles.

Raison : Simplifie l'interaction avec la base de données en utilisant des repositories et des requêtes JPA.

PostgreSQL JDBC Driver :

Description : Driver JDBC pour PostgreSQL, permettant la connexion à une base de données PostgreSQL.

Raison : Nécessaire pour utiliser une base de données PostgreSQL avec l'application.

Project Lombok :

Description : Réduit la verbosité du code en générant automatiquement des méthodes telles que les getters, setters et constructeurs.

Raison : Améliore la lisibilité du code en éliminant le code boilerplate.

Rest Assured :

Description : Facilite les tests automatisés des API REST en fournissant une syntaxe fluide pour les assertions.

Raison : Utilisé dans les tests d'intégration pour valider le comportement des API REST.

MapStruct :

Description : Framework de mapping objet, générant automatiquement du code de mappage entre objets Java.

Raison : Simplifie la conversion entre les entités JPA et les DTO (Data Transfer Objects).

Feign :

Description : Facilite la création de clients HTTP déclaratifs pour les services web.

Raison : Utilisé pour simplifier la consommation de services web externes dans l'application.

Par ailleurs, j'ai intégré un frontend à mon application en utilisant le partage de ressources entre origines (CORS). La configuration a été ajustée pour autoriser les requêtes provenant de `http://localhost:3000`. Pour lancer le frontend, vous pouvez installer React si ce n'est pas déjà fait. Ensuite, accédez au dossier "frontend" dans le terminal, exécutez la commande `npm install`, puis lancez la commande `npm start`. La page React se lancera automatiquement, vous permettant d'apprécier l'aspect visuel de mon application.

Ces choix techniques ont été guidés par la recherche d'efficacité, de simplicité et de compatibilité, contribuant ainsi à la conception globale de l'application.

Voici une image de mon frontend.



Au moment des screens je n'avais pas encore ajouté des relations tel que ManyToMany, OneToOne et OneToMany dans mes modèles et après les avoir ajoutés ça reste similaire à l'exception de Fight Tactics, maintenant le résultat est légèrement différent mais quasi similaire :

Roronoa Zoro



Swordsman

Job: Pirate

Bounty: 1111000000

Devil Fruit: N/A

Crew: Straw Hat Pirates

Haki: Armament Haki

Fight Onigiri (Melee)

Tactics: Santoryu: Three Thousand Worlds (Melee)

Monkey D. Luffy



Captain

Job: Pirate

Bounty: 3000000000

Devil Fruit: Gomu Gomu no Mi

Crew: Straw Hat Pirates

Haki: Armament Haki

Fight Tactics: Gum-Gum Gatling (Melee)

Gum-Gum Pistol (Melee)

3. Utilisation de l'API

Je vais expliquer le projet pour le contrôleur de personnage car c'est le même principe pour les autres contrôleurs.

Récupérer tous les personnages :

Point de terminaison : <http://localhost:8080/api/characters>

Renvoie une liste de tous les personnages de la base de données.

```
// http://localhost:8080/api/characters
[
  {
    "id": 5,
    "name": "Monkey D. Garp",
    "rank": "Vice Admiral",
    "job": "Marine",
    "bounty": 0,
    "devilFruit": null,
    "crew": null,
    "haki": {
      "id": 2,
      "nameHaki": "Busô-shoku no Haki",
      "descriptionHaki": "Weaponry Haki, This form of Haki creates an invisible armor that is one of the few ways to touch people who have eaten a Demon Fruit and counter their attacks."
    },
    "fightTactics": {
      "id": 5,
      "nameTactics": "Fist of Love",
      "type": "Busoshoku Haki: Advanced Armament"
    },
    "imageUrl": "https://pm1.aminoapps.com/6952/a8f89be4e601acd4f9b32043801d6e93d8a46884r1-1280-720v2_hq.jpg"
  },
]
```

Ajouter un nouveau personnage :

Point de terminaison : <http://localhost:8080/api/characters/add>

```
1  {
2    "name": "Shanks",
3    "rank": "Yonko",
4    "job": "Pirate",
5    "bounty": 400000000,
6    "imageUrl": "https://staticg.sportskeeda.com/editor/2023/09/27939-16937994561538-1920.jpg",
7    "devilFruit": null,
8    "crew": {
9      "id": 3
10   },
11   "haki": {
12     "id": 2
13   },
14   "fightTactics": [
15     {
16       "id": 5
17     },
18     {
19       "id": 6
20     }
21   ]
22 }
23
24
25
26
```



Faite attention à ajouter d'abord les devilFruit,crew,haki et fightTactics afin de récupérer leurs ID, sinon vous pouvez le faire de cette manière :

```
{
  "name": "Monkey D. Garp",
  "rank": "Vice Admiral",
  "job": "Marine",
  "bounty": 11,
  "imageUrl": "https://pm1.aminoapps.com/6952/a8f89be4e601acd4f9b32043801d6e93d8a40884r1-1280-720v2_hq.jpg"
}
```

Supprimer un personnage par son ID :

Point de terminaison : <http://localhost:8080/api/characters/2>

Supprime le personnage avec l'ID spécifié.

Récupérer les personnages par ID d'équipage :

Point de terminaison : <http://localhost:8080/api/characters/byCrew/1>

Retourne les personnages appartenant à l'équipage spécifié.

```
// http://localhost:8080/api/characters/byCrew/1
[
  {
    "id": 2,
    "name": "Monkey D. Luffy",
    "rank": "Yonko",
    "job": "Captain",
    "bounty": 300000000,
    "devilFruit": {
      "id": 2,
      "nameFruit": "Gomu Gomu no Mi",
      "abilities": "Grants the ability to stretch one's body like rubber",
      "type": "Paramecia"
    },
    "crew": {
      "id": 1,
      "nameCrew": "Straw Hat Pirates",
      "shipName": "Thousand Sunny",
      "numberPirate": 10
    },
    "haki": {
      "id": 3,
      "nameHaki": "Hao-shoku no Haki",
      "descriptionHaki": "Haki of Kings, his form of Haki is more or less powerful, depending on the will of the user, and cannot be strengthened by any means (other than experience and willpower).",
      "fightTactics": {
        "id": 2,
        "nameTactics": "Gum-Gum Gatling",
        "type": "A rapid barrage of punches using rubber-based powers."
      }
    },
    "imageUrl": "https://lh3.googleusercontent.com/Rvg5lh-fQwbpEGDpb3YJz5Rtq3_28Tvo2h330RCd1z16lv5ghitDmH6hcWAS7g8d6_8VOpQYcCem880F7QQA64ZKkEQn71zuF1LDsXwCh4ywf_-z6Vnxb8gmbQyK_PWf7cztzDd1W0TEEUHX66dP60"
  }
]
```

Récupérer les personnages par job :

Point de terminaison : <http://localhost:8080/api/characters/byJob/Captain>

Retourne les personnages ayant la fonction spécifiée.

```
// http://localhost:8080/api/characters/byJob/Captain

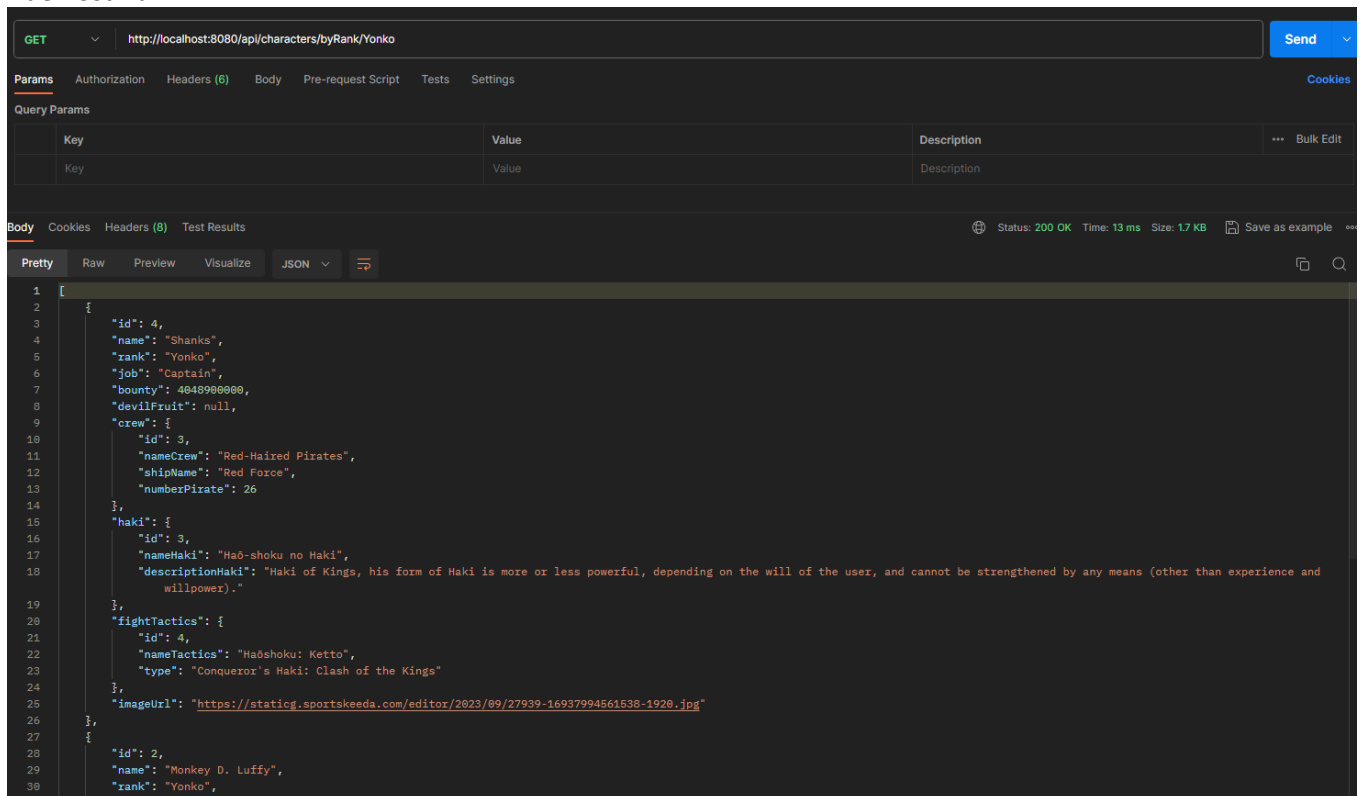
[
  {
    "id": 4,
    "name": "Shanks",
    "rank": "Yonko",
    "job": "Captain",
    "bounty": 4048900000,
    "devilFruit": null,
    "crew": {
      "id": 3,
      "nameCrew": "Red-Haired Pirates",
      "shipName": "Red Force",
      "numberPirate": 26
    },
    "haki": {
      "id": 3,
      "nameHaki": "Haō-shoku no Haki",
      "descriptionHaki": "Haki of Kings, his form of Haki is more or less powerful, depending on the will of the user, and cannot be strengthened by any means (other than experience and willpower).",
    },
    "fightTactics": {
      "id": 4,
      "nameTactics": "Haōshoku: Ketto",
      "type": "Conqueror's Haki: Clash of the Kings"
    },
    "imageUrl": "https://staticg.sportskeeda.com/editor/2023/09/27939-16937994561538-1920.jpg"
  },
  {
    "id": 2,
    "name": "Monkey D. Luffy",
    "rank": "Yonko",
    "job": "Captain",
    "bounty": 3000000000,
    "devilFruit": {
      "id": 2,
      "nameFruit": "Gomu Gomu no Mi",
      "abilities": "Grants the ability to stretch one's body like rubber",
      "type": "Paramecia"
    },
    "crew": {
      "id": 1,
      "nameCrew": "Straw Hat Pirates",
      "shipName": "Thousand Sunny",
    }
  }
]
```

Récupérer les personnages par rang :

Point de terminaison : <http://localhost:8080/api/characters/byRank/yonko>

Retourne les personnages ayant le rang spécifié.

Vue Postman :



The screenshot shows a Postman interface with a GET request to `http://localhost:8080/api/characters/byRank/Yonko`. The response is a JSON array containing two objects representing pirate characters.

```
[
  {
    "id": 4,
    "name": "Shanks",
    "rank": "Yonko",
    "job": "Captain",
    "bounty": 4048900000,
    "devilFruit": null,
    "crew": {
      "id": 3,
      "nameCrew": "Red-Haired Pirates",
      "shipName": "Red Force",
      "numberPirate": 26
    },
    "haki": {
      "id": 3,
      "nameHaki": "Haō-shoku no Haki",
      "descriptionHaki": "Haki of Kings, his form of Haki is more or less powerful, depending on the will of the user, and cannot be strengthened by any means (other than experience and willpower).",
    },
    "fightTactics": {
      "id": 4,
      "nameTactics": "Haōshoku: Ketto",
      "type": "Conqueror's Haki: Clash of the Kings"
    },
    "imageUrl": "https://staticg.sportskeeda.com/editor/2023/09/27939-16937994561538-1920.jpg"
  },
  {
    "id": 2,
    "name": "Monkey D. Luffy",
    "rank": "Yonko",
    "job": "Captain",
    "bounty": 3000000000,
    "devilFruit": {
      "id": 2,
      "nameFruit": "Gomu Gomu no Mi",
      "abilities": "Grants the ability to stretch one's body like rubber",
      "type": "Paramecia"
    },
    "crew": {
      "id": 1,
      "nameCrew": "Straw Hat Pirates",
      "shipName": "Thousand Sunny",
    }
  }
]
```

Récupérer les personnages avec une prime élevée et aucun fruit du démon :

Point de terminaison : <http://localhost:8080/api/characters/highBountyAndNoDevilFruit>

Retourne les personnages dont la prime est supérieure à 1 000 000 000 et qui n'ont pas de fruit du démon.

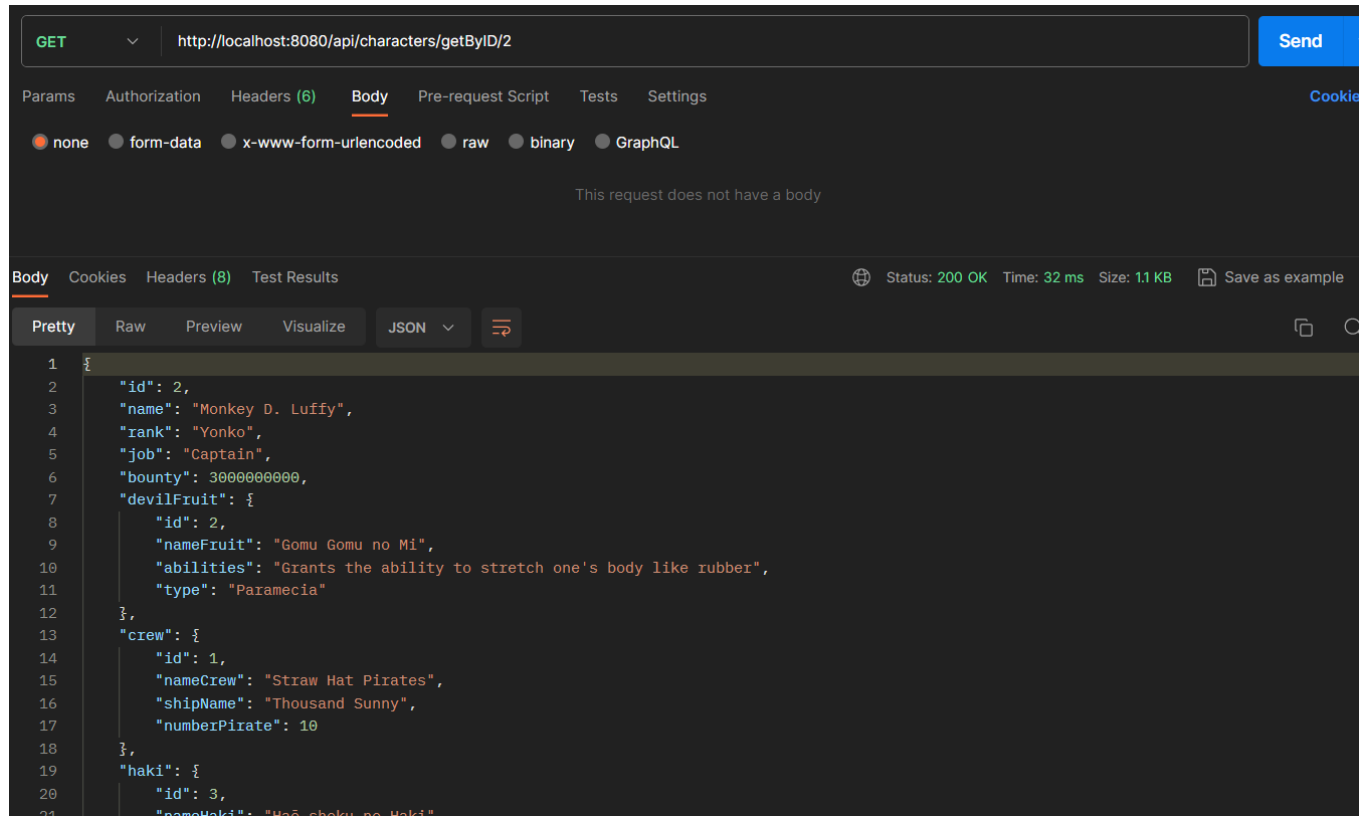
```
// http://localhost:8080/api/characters/highBountyAndNoDevilFruit
```

```
[
  {
    "id": 4,
    "name": "Shanks",
    "rank": "Yonko",
    "job": "Captain",
    "bounty": 4048900000,
    "devilFruit": null,
    "crew": {
      "id": 3,
      "nameCrew": "Red-Haired Pirates",
      "shipName": "Red Force",
      "numberPirate": 26
    },
    "haki": {
      "id": 3,
      "nameHaki": "Haō-shoku no Haki",
      "descriptionHaki": "Haki of Kings, his form of Haki is more or less powerful, depending on the will of the user, and cannot be strengthened by any means (other than experience and willpower).",
    },
    "fightTactics": {
      "id": 4,
      "nameTactics": "Haōshoku: Ketto",
      "type": "Conqueror's Haki: Clash of the Kings"
    },
    "imageUrl": "https://staticg.sportskeeda.com/editor/2023/09/27939-16937994561538-1920.jpg"
  }
]
```

Récupérer les personnages par ID :

Point de terminaison : <http://localhost:8080/api/characters/getById/2>

Retourne le personnage dont l'ID vaut 2



Modifier les données d'un personnage selon son ID

Point de terminaison : <http://localhost:8080/api/characters/put/5>

Modifie les données du personnage dont l'ID vaut 5

Voici un avant de l'id 5 :

```

{
  "id": 5,
  "name": "Monkey D. Garp",
  "rank": "Vice Admiral",
  "job": "Marine",
  "bounty": 0,
  "devilFruit": null,
  "crew": null,
  "haki": {
    "id": 2,
    "nameHaki": "Busō-shoku no Haki",
    "descriptionHaki": "Weaponry Haki, This form of Haki creates an invisible armor that is one of the few ways to touch people who have eaten a Demon Fruit and counter their attacks."
  },
  "fightTactics": {
    "id": 5,

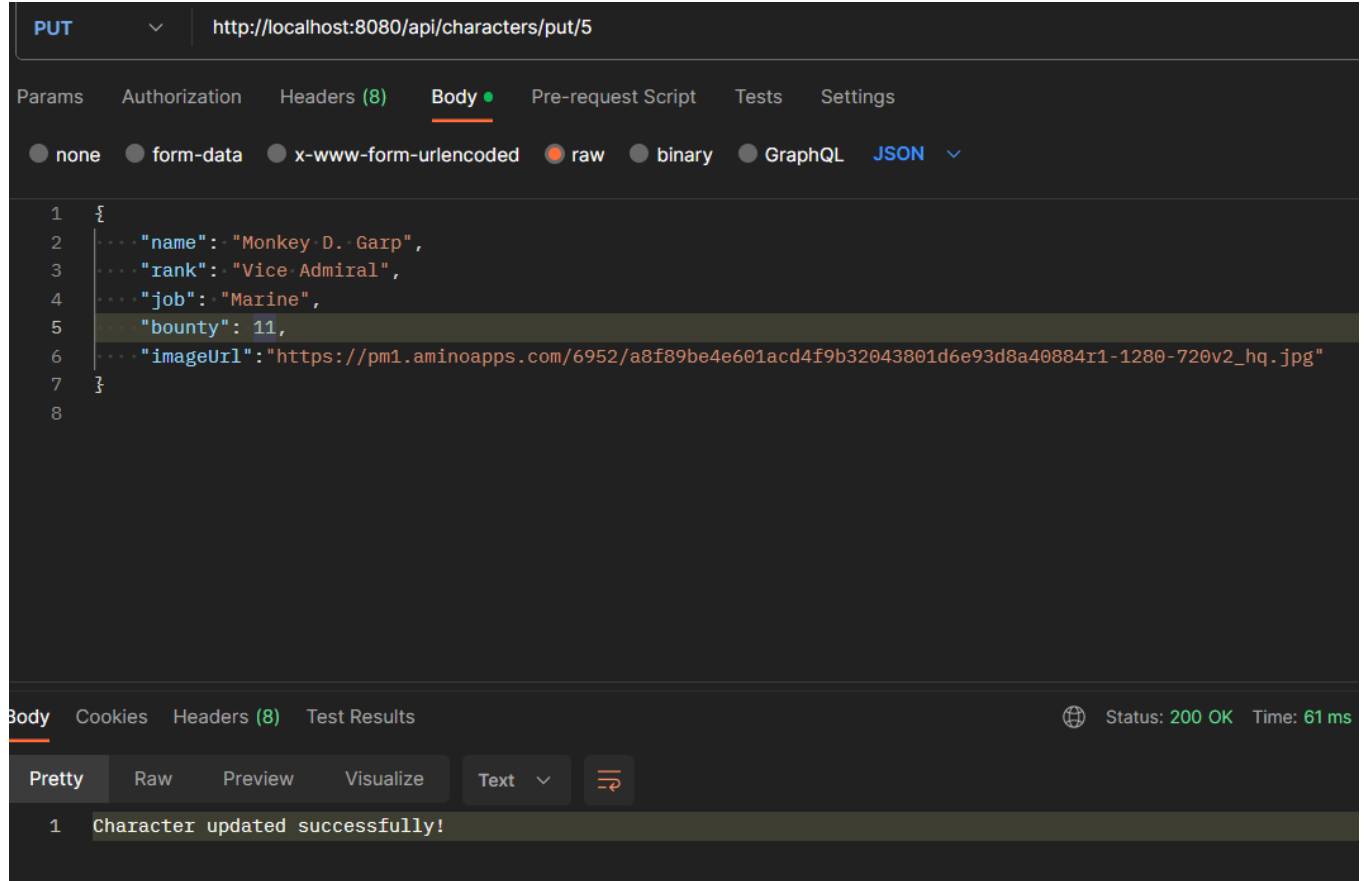
```

```

    "nameTactics": "Fist of Love",
    "type": "Busoshoku Haki: Advanced Armament"
  },
  "imageUrl": "https://pm1.aminoapps.com/6952/a8f89be4e601acd4f9b32043801d6e93d8a40884r1-1280-720v2_hq.jpg"
}

```

Et voici un après avec la requête put de la bounty à 11 :



PUT ▼ http://localhost:8080/api/characters/put/5

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL **JSON** ▼

```

1  {
2    ...."name": "Monkey D. Garp",
3    ...."rank": "Vice Admiral",
4    ...."job": "Marine",
5    ...."bounty": 11,
6    ...."imageUrl": "https://pm1.aminoapps.com/6952/a8f89be4e601acd4f9b32043801d6e93d8a40884r1-1280-720v2_hq.jpg"
7  }
8

```

Body Cookies Headers (8) Test Results 🌐 Status: 200 OK Time: 61 ms

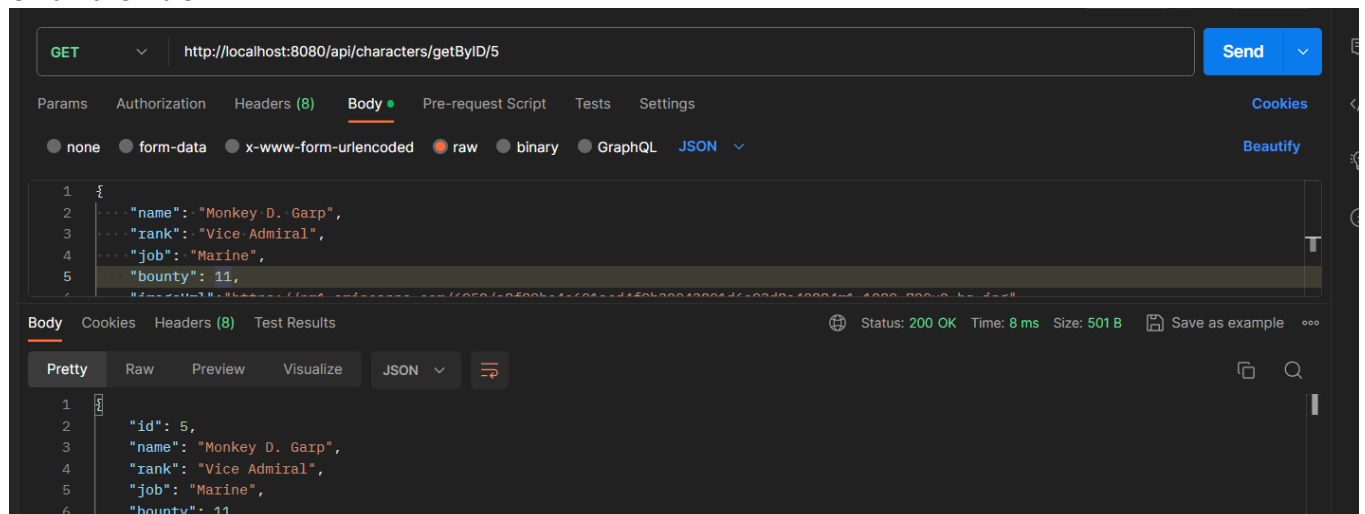
Pretty Raw Preview Visualize Text ▼ ≡

```

1  Character updated successfully!

```

On affiche l'id 5 :



GET ▼ http://localhost:8080/api/characters/getByID/5 Send ▼

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings Cookies Beautify

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL **JSON** ▼

```

1  {
2    ...."name": "Monkey D. Garp",
3    ...."rank": "Vice Admiral",
4    ...."job": "Marine",
5    ...."bounty": 11,
6    ...."id": 5

```

Body Cookies Headers (8) Test Results 🌐 Status: 200 OK Time: 8 ms Size: 501 B Save as example ⋮

Pretty Raw Preview Visualize **JSON** ▼ ≡ 🔍

```

1  {
2    "id": 5,
3    "name": "Monkey D. Garp",
4    "rank": "Vice Admiral",
5    "job": "Marine",
6    "bounty": 11,

```



En ce qui concerne les autres Endpoints des autres contrôleurs, c'est toujours le même principe, pour les fights tactique, vous avez pour voir toutes les tactiques :

<http://localhost:8080/api/fight-tactics>

Pour ajouter une tactique :

<http://localhost:8080/api/fight-tactics/add>

Pour supprimer une tactique :

<http://localhost:8080/api/fight-tactics/1>

Pour le getByID :

<http://localhost:8080/api/fight-tactics/getByID/1>

Pour le put :

<http://localhost:8080/api/fight-tactics/put/1>

Ainsi de suite pour les autres contrôleurs.

Crew :

Afficher Tout :

<http://localhost:8080/api/crews/>

Ajout :

<http://localhost:8080/api/crews/add>

Suppression :

<http://localhost:8080/api/crews/1>

Pour le getByID :

<http://localhost:8080/api/crews/getByID/1>

Pour le put :

<http://localhost:8080/api/crews/put/1>

Devil Fruits :

Afficher Tout :

<http://localhost:8080/api/devil-fruits/>

Ajout :

<http://localhost:8080/api/devil-fruits/add>

Suppression :

<http://localhost:8080/api/devil-fruits/1>

Pour le getByID :

<http://localhost:8080/api/devil-fruits/getByID/1>

Pour le put :

<http://localhost:8080/api/devil-fruits/put/1>

Haki :

Afficher Tout :

<http://localhost:8080/api/haki/>

Ajout :

<http://localhost:8080/api/haki/add>

Suppression :

<http://localhost:8080/api/haki/1>

Pour le getByID :



<http://localhost:8080/api/haki/getById/1>

Pour le put :

<http://localhost:8080/api/haki/put/1>

Run la seconde API

Récupérer les combats depuis la première API

Point de terminaison : <http://localhost:8080/fights/getAllFights>

Retourne les combats

Similaire depuis l'api secondaire avec port 8081 :

<http://localhost:8081/fights/getAllFights>

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/fights/getAllFights`
- Method:** GET
- Query Params:** A table with 4 columns: Key, Value, Description, and an ellipsis. The first row contains 'Key', 'Value', 'Description', and an ellipsis.
- Body:** The response is displayed in JSON format (Pretty view):

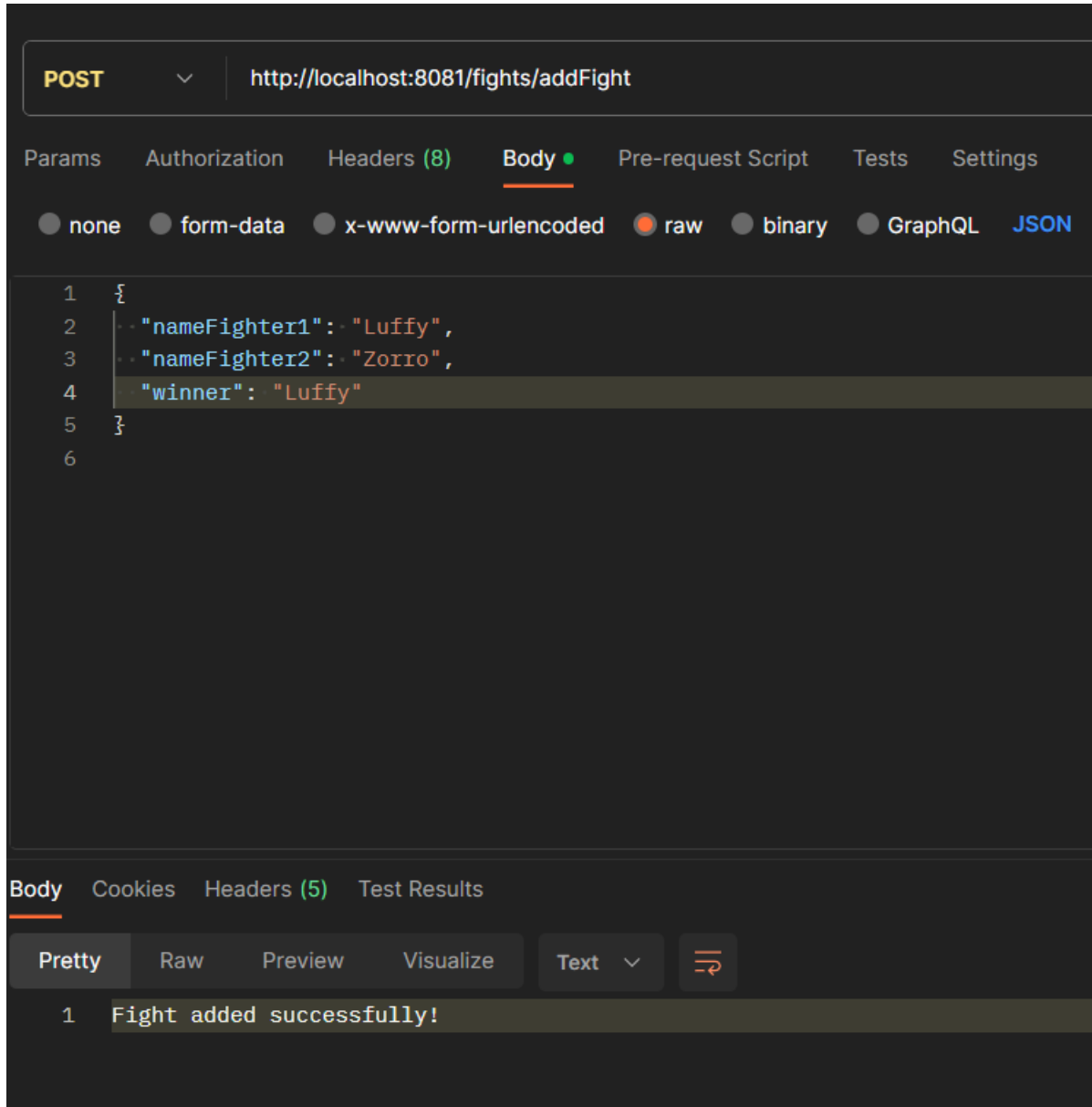
```
1 [
2   {
3     "id": 1,
4     "nameFighter1": "Luffy",
5     "nameFighter2": "Zorro",
6     "winner": "Luffy"
7   }
8 ]
```
- Status:** 200 OK
- Time:** 190 ms
- Size:** 237 B

Ajouter un combat depuis la seconde API

Point de terminaison : <http://localhost:8080/fights/getAllFights>

Retourne le combat ajouté

<http://localhost:8081/fights/addFight>



The screenshot shows a REST client interface. At the top, the method is **POST** and the URL is `http://localhost:8081/fights/addFight`. Below the URL bar, there are tabs for **Params**, **Authorization**, **Headers (8)**, **Body** (selected), **Pre-request Script**, **Tests**, and **Settings**. Under the **Body** tab, there are radio buttons for **none**, **form-data**, **x-www-form-urlencoded**, **raw** (selected), **binary**, **GraphQL**, and **JSON**. The body content is a JSON object:

```
1 {  
2   "nameFighter1": "Luffy",  
3   "nameFighter2": "Zorro",  
4   "winner": "Luffy"  
5 }  
6
```

Below the body editor, there are tabs for **Body** (selected), **Cookies**, **Headers (5)**, and **Test Results**. Under the **Body** tab, there are buttons for **Pretty** (selected), **Raw**, **Preview**, **Visualize**, **Text**, and a dropdown menu. The response body is:

```
1 Fight added successfully!
```

Pour le reste get by id / delete, c'est similaire que dans la première api mais avec le port 8081 :

Suppression :

<http://localhost:8081/fights/deleteFight/1>

Pour le getByID:

<http://localhost:8081/fights/findFight/1>

En ce qui concerne les tests d'intégration il faut juste run les 6 classes de tests disponible pour vérifier les Endpoints de l'application, voici pour les personnages, sinon c'est toujours le même principe :

```

✓ CharacterControllerIntegrationTest 1 sec 533 ms ✓ Tests passed: 8 of 8 tests – 1 sec 533 ms
  ✓ testAddCharacter() 913 ms
  ✓ testUpdateCharacter() 31 ms
  ✓ testGetAllCharacters() 422 ms
  ✓ testGetCharactersByCrew() 55 ms
  ✓ testGetCharactersByRank() 33 ms
  ✓ testGetCharacterById() 25 ms
  ✓ testGetCharactersByJob() 25 ms
  ✓ testGetCharactersWithHighBountyAr 29 ms

C:\Users\claud\.jdk\openjdk-19.0.1\bin\java.exe ...
21:40:57.761 [main] DEBUG org.springframework.test.context.Bootst
.DefaultCacheAwareContextLoaderDelegate]
21:40:57.769 [main] DEBUG org.springframework.test.context.Bootst
.DefaultBootstrapContext(java.lang.Class,org.springframework.te
21:40:57.804 [main] DEBUG org.springframework.test.context.Bootst
.CharacterControllerIntegrationTest] from class [org.springfram
21:40:57.813 [main] INFO org.springframework.boot.test.context.S
[be.helb.cpopadiuc.controller.CharacterControllerIntegrationTes
21:40:57.816 [main] DEBUG org.springframework.test.context.support

```

Similaire pour l'api secondaire :

```

Run FightControllerIntegrationTest x
✓ FightControllerIntegrationTest (b) 1 sec 783 ms ✓ Tests passed: 4 of 4 tests – 1 sec 783 ms
  ✓ testDeleteFight() 1 sec 72 ms
  ✓ testAddFight() 192 ms
  ✓ testGetFightById() 26 ms
  ✓ testGetAllFights() 493 ms

"C:\Program Files\Microsoft\jdk-11.0.16-hotspot\bin\java.exe" ...
21:40:07.509 [main] DEBUG org.springframework.test.context.Bootst
.DefaultCacheAwareContextLoaderDelegate]
21:40:07.517 [main] DEBUG org.springframework.test.context.Bootst
.DefaultBootstrapContext(java.lang.Class,org.springframework.test.context.CacheAwareCont
21:40:07.557 [main] DEBUG org.springframework.test.context.Bootst
.FightControllerIntegrationTest] from class [org.springframework.boot.test.context.Sprin
21:40:07.569 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBoots
[be.helb.cpopadiuc.controller.FightControllerIntegrationTest], using SpringBootTestContextLo
21:40:07.574 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader
.FightControllerIntegrationTest]: class path resource [be/helb/cpopadiuc/controller/Figh
21:40:07.575 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader
.FightControllerIntegrationTest]: class path resource [be/helb/cpopadiuc/controller/Figh
21:40:07.575 [main] INFO org.springframework.test.context.support.AbstractContextLoader -
.FightControllerIntegrationTest]: no resource found for suffixes {-context.xml, Context.
21:40:07.576 [main] INFO org.springframework.test.context.support.AnnotationConfigContext

```