Projet Développement JAVA V 2023-2024



Une image contenant fleur

Description générée automatiquement avec une confiance moyenne

Produced by Popadiuc Claudiu

Supervise by Mr Bilal Ben Abdelkader

Table des matières

[1. Architecture Générale 3](#_Toc155547951)

[2. Choix Technique 7](#_Toc155547952)

[3. Utilisation de l’API 9](#_Toc155547953)

# 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 :

Une image contenant texte, capture d’écran

Description générée automatiquement

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 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.

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 :

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

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 :

@OneToOne dans la classe Character avec @JoinColumn(name = "devilfruit\_id")

@OneToOne(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")

@OneToMany(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")

@OneToMany(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 débuter, 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.

Dans le cadre de mon projet, j'ai également utilisé RestTemplate pour interagir avec une deuxième API. Il a été employé pour effectuer des requêtes HTTP vers une API externe dédiée aux combats de personnages, facilitant ainsi l'initialisation de ces combats au sein de mon 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.

Une image contenant texte, Visage humain, personne, habits

Description générée automatiquement

Au moment des screens je n’avais pas encore ajouté des relations tel que ManyToMany, OneToOne et OneToMany dans mes modèles et après les avoirs ajouter ça reste similaire à l’exception de Fight Tactics, maintenant le résultat est légèrement diffèrent mais casi similaire :

Une image contenant texte, Visage humain, chapeau, Accessoire de mode

Description générée automatiquement

# 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.

Une image contenant capture d’écran

Description générée automatiquement

**Ajouter un nouveau personnage :**

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

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

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.

Une image contenant capture d’écran, texte

Description générée automatiquement

Récupérer les personnages par job :

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

Retourne les personnages ayant la fonction spécifiée.

Une image contenant capture d’écran, texte

Description générée automatiquement

Récupérer les personnages par rang :

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

Retourne les personnages ayant le rang spécifié.

Vue Postman :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

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.

Une image contenant capture d’écran, texte

Description générée automatiquement

Intégration d'API externes (CharacterController et RestTemplate) :

Utilise RestTemplate pour interagir avec une API de combat externe pour initier des combats de personnages.

Pour tester ma seconde API, vous devez démarrer les 2 projets, et voici la requête à envoyer :

<http://localhost:8080/api/characters/fight/Luffy/Zorro>, ainsi l’api principale ira demander l’information sur un combat entre 2 personnages à l’api secondaire et renverra le résultat :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Récupérer les personnages par ID :

Point de terminaison : http://localhost:8080/api/characters/getByID/2

Retourne le personnage dont l’ID vaux 2

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

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 vaux 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 :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

On affiche l’id 5 :

Une image contenant capture d’écran, texte, logiciel, Logiciel multimédia

Description générée automatiquement

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](http://localhost:8080/api/crews%20/getByID/1)

Pour le put :

[http://localhost:8080/api/crews /put/1](http://localhost:8080/api/crews%20/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>

En ce qui concerne les tests d’intégration il faut juste run les 5 classes de tests disponible pour vérifier les Endpoints de l’application, voici pour les personnages, sinon c’est toujours le même principe :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement