Projet Web API

Afin de mettre en place un nouveau jeu de type Gatcha, vous allez devoir gérer la mise en place de différentes API pour rendre le système opérationnel.

Afin de rendre le jeu facile à déployer, vous devrez utiliser docker et des bases de données mongodb.
Dans l'idéal, vous fournirez des fichiers JSON contenant les données à importer dans les collections mongo afin d'avoir une base pour tester l'application.
Vous devrez fournir dans ce projet un readme expliquant les différentes étapes pour lancer le projet avec l'aide d'un docker-compose.
Le projet ***COMPLET*** doit tourner sous docker, pas uniquement les bases mongo.

Concrètement le projet doit être simple à lancer sur n'importe quel ordinateur en récupérant juste le code et les ressources associées.

L'utilisation de SpringBoot est fortement conseillée.

En bonus, chaque API devra être couverte par des tests unitaires, et des vrais, pas de *assert(true).isTrue()* sinon c'est 0.

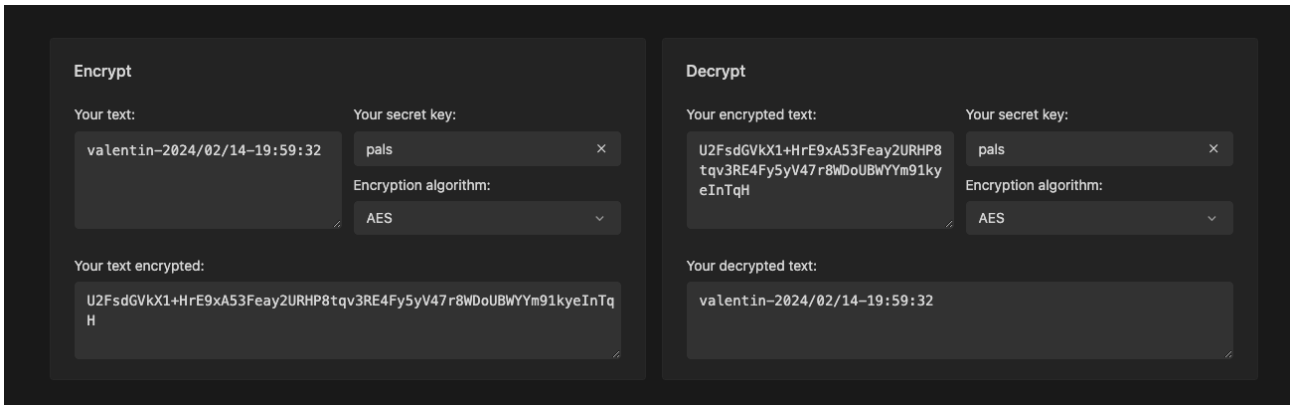
Ce travail pourra être réalisé en groupes de 4 personnes maximum.
L'utilisation de git vous est conseillée pour ce genre de travail d'équipe.

API d’authentification

La première et la plus importante : une API pour gérer l’authentification. Elle prendra en :

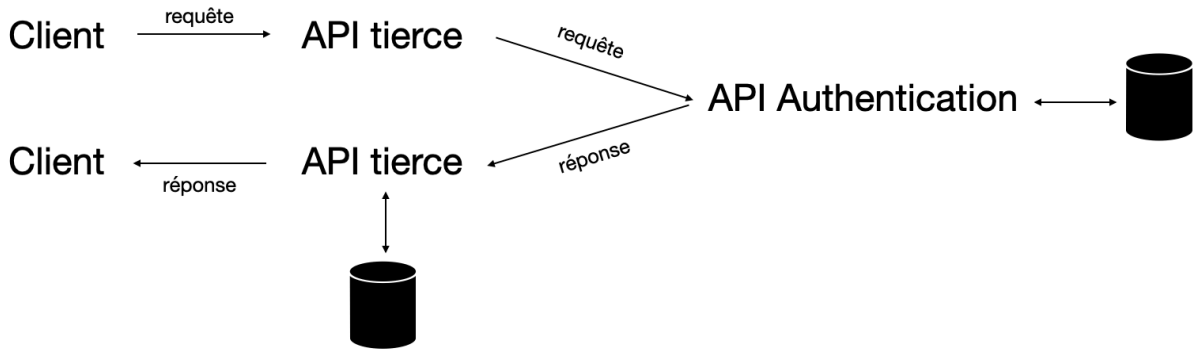
- entrée :
  - identifiant
  - password
- sortie en cas de 200 :
  - token

L’objectif de cette API sera de générer un token d’authentification (fait maison).
Dans une base seront stockés les identifiants et mots de passe (pas nécessaire de s’embêter avec du cryptage, on veut faire simple).
Lors d’un appel à l’API avec ces informations, si le mot de passe correspond à l’identifiant, on retourne un token auto-généré qu’on va stocker en base en se basant sur les informations d’authentification.
L’objectif est de générer un token encrypté constitué de : username-date(YYYY/MM/DD)-heure (HH:mm:ss). Par exemple :



Ce token sera valable une heure, et sa date d’expiration devra donc être enregistrée dans la base.

A chaque appel d’une autre API du système, ce token devra être passé à et vérifié par l’API d’authentification. Si le token est toujours valide, l’API d’authentification valide l’appel et retourne le username lié au token, puis met à jour la date d’expiration à [maintenant + une heure].
Si le token est expiré, l’API renvoie une erreur 401, et demande donc une nouvelle authentification à l’utilisateur.



Par exemple :

Je requête l'API du système permettant de gérer les différents monstres. Afin de lui signifier que je suis connecté, je lui passe dans les headers le token que j'ai récupéré précédemment en appelant l'API authentification avec mes identifiants. L'API monstres appelle alors l'API authentification pour valider le token.
Cas 1 : mon token est valide -> l'API authentification retourne mon username pour l'API monstre et celle-ci exécute ma requête.
Cas 2 : mon token est expiré -> l'API authentification retourne une erreur 401 et demande un token valide. L'API monstre transfère cette réponse en retour.

API Joueur

L'API Joueur est utilisée pour gérer les infos de compte utilisateurs qui sont :
identifiant
level (va de 0 à 50)
experience (commence à 50 pas d'xp pour level up au niveau 1 pour passer niveau 2 puis est multiplié par 1,1 à chaque niveau : donc il faudra 50 * 1,1 = 55 xp pour passer au niveau 3)
List<monstres> (taille conditionnée par le niveau = commence à 10 puis + 1 pour chaque level). Ces monstres sont représentés dans cette liste par un id unique.

Elle doit pouvoir gérer :

- la récupération de toutes les informations du profil
- la récupération de la liste de monstre
- la récupération du niveau du joueur
- un gain d'expérience (quantité passée en paramètre) et retourner le nouveau statut utilisateur
- un gain de niveau (reset l'expérience, augmente le seuil de level up et augmente la taille max de la liste de monstres) et retourner le nouveau statut utilisateur
- l'acquisition d'un nouveau monstre
- la suppression d'un monstre

API Monstres

L'API Monstres permet de gérer les monstres *de tous les joueurs*. On doit pouvoir augmenter leurs niveaux grâce à des points d'xp qu'ils acquièrent au fur et à mesure des combats. Chaque niveau gagné apporte une augmentation des stats du monstre, et un point d'amélioration de compétence à distribuer pour ce monstre.

Un monstre est défini par les statistiques suivantes :
- type élémentaire (feu / eau / vent)
- hp (health points)
- atk (attaque)
- def (defense)
- vit (vitesse)

Et par *3 compétences* qui sont elles mêmes constituées de :
- dégâts de base
- ratio de dégâts supplémentaires en fonction d'une des 4 stats
- cooldown
- niveau d'amélioration
- niveau d'amélioration maximum

Cette API ne sera évidemment accessible qu'avec un token d'authentification valide, et un joueur ne pourra voir via cette API que les monstres qui lui sont associés.

API Invocations

L'API d'invocations permettra de générer aléatoirement un monstre pour un joueur en se basant sur la base totale des monstres existants.
Pour chaque monstre unique, on retrouvera un taux de probabilité d'invocation, ainsi que ses stats de base. Pour rappel :
- type élémentaire (feu / eau / vent)
- hp (health points)
- atk (attaque)
- def (defense)
- vit (vitesse)

Et ses 3 compétences.

Afin que le système d'invocation et le système de gestion des monstres déjà invoqués ne soient pas interdépendants, on retrouvera une base de données contenant uniquement les infos de base du monstre + le taux de probabilité d'invocation pour l'API Invocation.

Calcul de l'invocation (un exemple de façon de faire) :

Il faudra prendre en compte le taux d'invocation exprimé en pourcentage pour générer le monstre invoqué. Vous êtes libre de gérer ça comme il se faut, votre algorithme d'invocation sera testé en étant appelé plusieurs milliers de fois, afin de prouver que les pourcentages défini, sont bien respectés.
Il vous est donc vivement conseillé de faire un test unitaire conséquent vous permettant de valider votre propre algorithme, afin de vous éviter toute mauvaise surprise lors du test.

Lors d'une invocation :

L'API Invocation envoie l'information du monstre invoqué à l'API Monstres.
L'API Monstre renvoie alors l'id du monstre généré dans sa base à l'API Invocation.
L'API Invocation transmet l'id ainsi récupéré à l'API Joueur qui le stocke dans sa liste de monstres.

Afin d'être sécurisé en cas de problème réseau ou de serveur down pour une API, l'API d'Invocation doit stocker dans une base tampon les informations qu'elle génère, transmet, et reçoit.
Ces informations permettront de re-générer chaque étape des invocations plus tard si nécessaire.

L'API d'Invocation devra donc prévoir une *fonctionnalité de re-création* de toutes les invocations.

Ajout "front" et JS :

Afin de rendre l'invocation plus agréable, vous allez devoir créer un front contenant un bouton dont la fonction JS permettra d'invoquer un monstre.
Lors du clic sur le bouton, un appel à l'API invocation devra se faire, et on affichera sur la page les informations du montre invoqué.
Aucune obligation de présentation, si vous aimez ça et que vous avez le temps vous pouvez faire un joli front, sinon le principal est surtout la fonctionnalité.

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

API Combat (BONUS)

Le but de cette API sera de simuler un combat automatique entre 2 monstres. Les monstres exécuteront leurs compétences dans l'ordre décroissant de leur numéro tant que le cooldown d'utilisation est à 0.

API prend en entrée :
- 2 monstres

API retourne :
- monstre/joueur vainqueur
- numéro du combat

L'API enregistrera dans une base les logs de combat, et permettra de faire une rediffusion à partir du numéro de combat qu'elle génère elle-même.

On doit pouvoir accéder également à l'historique de tous les combats afin de pouvoir choisir une rediffusion basé le numéro.

Exemple :

T1 : Monstre 1 utilise compétence 3 (qui a 4 tours de cooldown)
T2 : Monstre 1 utilise compétence 2 (qui a 2 tours de cooldown)
T3 : Monstre 1 utilise compétence 1 (qui a 0 tour de cooldown)
T4 : Monstre 1 utilise compétence 2
T5 : Monstre 1 utilise compétence 3
T3 : Monstre 1 utilise compétence 1
T3 : Monstre 1 utilise compétence 2

Etc…