WEB SERVICES REST

COMPTE RENDU DE PROJET: API REST POUR UNE PLATEFORME DE GESTION DE JOUEURS

MEMBRES DU GROUPE:

- BOUKOU Grace
- KADRI MOUNKAILA Askia Mohamed

ENVIRONNEMENT DE DÉVELOPPEMENT:

Java version: java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13, mixed mode)

IDE: IntelliJ IDEA Ultimate Edition

Grails version: 3.3.8

BASE DE DONNEES UTILISEE: SGBD H2

DESCRIPTION DE L'API:

Nous avons conçu une API Rest fournissant un point d'entrées sur des entités de notre plateforme de gestion de joueurs développée en grails.
Sur nos différentes entités nous pouvons réaliser les opérations suivantes:

- User

Pour le singulier:

- GET: permet de récupérer un User en spécifiant son Id en paramètre.
  Exemple de requête: http://localhost:8081/tp/api/user/1
  Type de réponse: {

```
"id": 1,
"dateCreated": "2018-10-14T14:20:15Z",
"passwordExpired": false,
"username": "admin",
"accountLocked": false,
"accountExpired": false,
"enabled": true,
"avatar": null
```

  }

- POST: permet de créer un nouveau User dans la base en spécifiant son nom, son mot de passe et son rôle. Tous les autres paramètres (enabled, accountExpired, accountLocked, passwordExpired) auront de valeurs par défaut qui pourront par la suite être modifiée par l'admin. Sa photo de profile est également mise par défaut et pourra être modifiée. Il faudra forcément changer le username car il est unique, autrement il y aura une erreur.
  Exemple de requête: http://localhost:8081/tp/api/user/
  En spécifiant les paramètre du User à rajouter dans le body de la requête sous format JSON

- PUT: permet de mettre à jour un User dans la base s'il existe en précisant les paramètres à mettre à jour. S'il n'existe pas un code d'erreur est renvoyé en spécifiant qu'il n'existe pas ou que la mise à jour est impossible en donnant la raison.
  Exemple de requête: http://localhost:8081/tp/api/user/2
  En spécifiant les paramètre du User à modifier dans le body de la requête sous format JSON

- DELETE: devrait permettre de supprimer un utilisateur de la base, cependant vu les contraintes référentielle, cet utilisateur n'est pas vraiment supprimé. On met juste son attribut enabled à false pour pouvoir le cacher dans la liste d'index des users. Autrement il aurait fallu supprimer l'Id du User dans toutes les tables matchs et messages avant de pouvoir le supprimer.
  Exemple de requête: http://localhost:8081/tp/api/user/2

Pour le pluriel:

- GET: permet de récupérer tous les User de la base. Ce verbe s'utilise sans paramètre.
  Exemple de requête: http://localhost:8081/tp/api/users

- POST: permet de créer plusieurs utilisateurs à la volée. Pour chaque utilisateur à créer un code d'erreur spécifique est envoyé si la création s'est mal passée ou qu'il y avait une erreur de syntaxe.
  Exemple de requête: http://localhost:8081/tp/api/users
  En spécifiant le body les paramètres de tous les Users à rajouter

- Match

Pour le singulier:

- GET: permet de récupérer un Match avec tous ses paramètres en spécifiant son Id en paramètre.
  Exemple de requête: http://localhost:8081/tp/api/match/2

Type de retour:


```
{
    "id": 2,
    "winnerScore": 20,
    "dateCreated": "2018-10-14T16:50:21Z",
    "looserScore": 15,
    "winner": {
        "id": 3
    },
    "looser": {
        "id": 2
    }
}
```


- POST: permet de créer un nouveau Match dans la base tout en spécifiant le winner et le looser avec leur scores respectif.
  Exemple de requête: http://localhost:8081/tp/api/match/

- PUT: permet de faire une mise à jour d'un Match si le Match existe.
  Exemple de requête: http://localhost:8081/tp/api/match/2

- DELETE: permet de faire supprimer un Match existant en donnant son Id en paramètre
  Exemple de requête: http://localhost:8081/tp/api/match/1

Pour le pluriel:

- GET: permet de recupérer tous les Matchs de la base
  Exemple de requête: http://localhost:8081/tp/api/matchs

- POST: permet de créer plusieurs matchs entre utilisateurs d'emblée en spécifiant dans les corps de la requête les paramètres des Matchs.
  Exemple de requête: http://localhost:8081/tp/api/matchs
  Exemple de body:


```
[
        {
                "winnerScore": 2,
            "looserScore": 1,
            "winner": {
                "id": 3
            },
            "looser": {
                "id": 2
            }
        },
  ...
]
```


- Message

Pour le singulier:

- GET: permet de récupérer un Message avec tous ses paramètres en spécifiant son Id en paramètre.

- POST: permet de créer un nouveau Message dans la base tout en spécifiant le target, author et le contenu du Message.

- PUT: permet de faire une mise à jour d'un Message si le Message existe.

- DELETE: permet de faire supprimer un Message existant

Pour le pluriel:

- GET: permet de récupérer tous les messages de la base
- POST: créé plusieurs messages entre utilisateurs

Toutes ces méthodes prennent en entrée (si elles en ont) des fichiers JSON dans le body, et produisent en sortie des fichiers du même type en renvoyant des HTTP Code Status appropriés (200 pour une analyse correcte de la ressource, 201 si la création d'une instance s'est bien passée, 400 si la syntaxe de la requête est incorrecte, 404 si la ressource demandée est introuvable et 405 si la méthode utilisée n'est pas applicable à la ressource demandée).
Toutes ces opérations peuvent être réalisées par un utilisateur ayant n'importe quel rôle à condition que celui ci soit connecté.

PROCÉDURES DE TEST:

Pour les Tests, nous avons produit une collection postman incluant un dossier pour chacun des domaines (User, Match, Message) dont chacun contient toutes les requêtes possible sur le domaine avec un test de vérification des codes d'erreurs associé. Le lancement d'une requête génère un Header contenant **X-Application-Context, Content-Type, Transfer-Encoding, Date,** un body contenant le render de la requête, et le résultat des tests. Pour tester les différents codes d'erreur, il suffit de modifier l'Id d'un User par exemple dans une requête GET pour voir si le code retourné est correct. Dans tous les cas cela ne change pas le résultat du code attendu. Pour lancer les tests il suffit d'importer l'un des fichiers JSON contenus dans le dossier **POSTMAN**, dans postman et les exécuter dans le bon ordre.

SÉCURITÉ:

Hormis la sécurité basique gérée à la main lors de la création de l'API, la sécurité est gérée par SpringSecurityRest qui crée des points d'entrée pour les utilisateurs connectées et restreint l'accès ou non à certaines ressources selon le rôle. Lorsqu'un utilisateur n'est pas connecté ou inconnu des utilisateurs de la base, tous les accès (quelque soit la ressource) lui sont totalement refusés (le serveur répond par une réponse 401)