

KaMS - authentication

Documentation technique

1. Le contexte

KaMS - authentication est un **micro service** de l'application KaMS. Il gère l'identification de l'utilisateur (user) :

- ***créer un compte***
- ***s'identifier***

2. La logique

a) L'enregistrement

i) méthode email/mdp

Lorsqu'un utilisateur s'enregistre il prodigue une adresse email et un mot de passe respectant les contraintes : au moins 8 caractères dont au minimum un chiffre, une lettre en majuscule, une lettre en minuscule et un symbole. Selon l'écran, l'appli détermine le rôle ("USER", "PRO", "ADMIN")

Ces informations sont alors stockées dans une table intermédiaire : UnverifiedUser qui stocke également un token de vérification pour chaque enregistrement.

L'application envoie un mail à l'adresse prodiguée contenant un lien d'activation.

Lorsque l'utilisateur clique sur ce lien, les données sont transférées de la table UnverifiedUser vers la table User et l'enregistrement de la table UnverifiedUser est supprimé.

ii) méthode OAuth

Lorsqu'un utilisateur se connecte pour la première fois avec un compte Google ou Github, un User est créé dans la base de données (sans passer par la phase de validation)

b) L'identification

i) méthode email/mdp

Pour accéder aux fonctionnalités de l'application, l'utilisateur doit s'identifier à travers un formulaire de login dans lequel il devra spécifier son adresse email et son mot de passe.

ii) méthode OAuth

Pour accéder à l'application, l'utilisateur peut aussi se connecter avec son compte Google ou Github.

3. Endpoints

a) Registration (email / password method)

URL : `http://localhost:8090/register`

Method : POST

Body parameter (Json) :

username

password

response (success) : `<201, "OK">`

response (fail) : `<400, User already exists message>`

call example :

```
$ curl -X POST http://localhost:8090/register -H "Content-Type: application/json" -d '{"username": "foo", "password": "bar"}'
```

b) Registration (email / password method) activation

URL : `http://localhost:8090/activate`

Method : GET

URL parameter : `activationToken`

response (success) : `<201, Created User (Json)>`

response (fail) :

`<400, Activation link expired message>`

`<404, token not found error message>`

call example :

```
$ curl -X GET http://localhost:8090/activate?activationToken=foobar
```

c) Authentication (email / password method)

URL : http://localhost:8090/auth

Method : POST

Body parameter (Json) :

username

password

response (success) : <200, the generated token>

response (fail) : <401, bad credentials message>

call example :

```
$ curl -X POST http://localhost:8090/auth -H "Content-Type: application/json" -d
{"username": "foo", "password": "bar"}
```

d) AuthToken validation

URL : http://localhost:8090/auth/token

Method : POST

Body parameter (Json) :

username

authToken

response (success) : <200, valid token message>

response (fail) :

<400, wrong token error message>

<401, expired session message>

call example :

```
$ curl -X POST http://localhost:8090/auth/token -H "Content-Type: application/json"
-d '{"username": "foo", "authToken": "bar"}'
```

4. MCD

User	UnverifiedUser
email : string password: string role: string	email: string password: string role: string verificationToken: string

5. MPD

User	
PK	user_id NOT NULL AUTO_INCREMENT
UK	email varchar(255) NOT NULL password varchar(255) role varchar(255)


UnverifiedUser	
PK	unverified_user_id NOT NULL AUTO_INCREMENT
UK	email varchar(255) NOT NULL password varchar(255) role varchar(255) verification_token varchar(255)

6. Stack Technique

Back-end :

	Java 17
	Gradle 7.5.1
	Spring boot 3.1.0 Spring Web, Spring Data JPA, Spring Security (JWT)
	Jacoco 0.8.9
	MySQL 8.0.29

Front-end :

	Angular 14.2.12
-------------------------------------------------------------------------------------	-----------------