



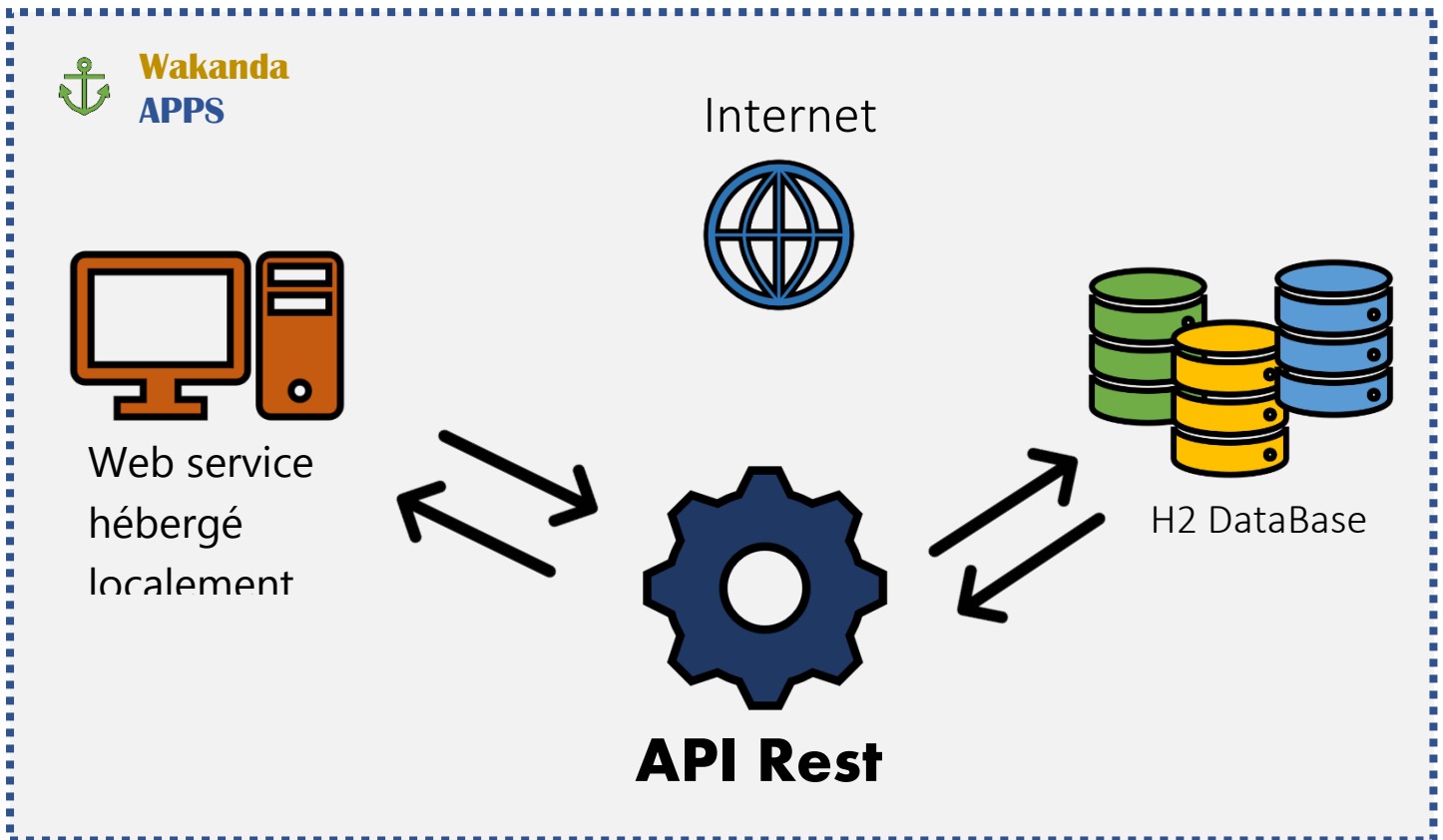
# Spécification

« Documentation technique »



API développée par étudiant ESTIA

## I. Contexte



Nous souhaitons réaliser une plateforme de gestion de message. Pour cela, nous allons d'abord réaliser l'API REST correspondante.

L'utilisateur de la plateforme doit pouvoir se connecter grâce à son nom d'utilisateur et son mot de passe. En fonction de son statut, administrateur ou non, l'utilisateur doit avoir droit à un contenu et des fonctionnalités différentes. Par exemple, l'administrateur doit pouvoir consulter tous les utilisateurs et tous les groupes. Il peut créer un nouvel utilisateur, un nouveau message ou un groupe.

L'API sera réalisée à l'aide du Framework Grails. Comme sur le schéma contextuel nous utiliserons le SGBD H2.

La suite de ce rapport sera consacrée à la documentation fonctionnelle et technique de l'API.

## II. Fonctionnalités et descriptions techniques

### II.1 Entité Message

Le résumer de toutes les fonctions à coder

Fonction	Descriptions « message() && messages »
F0	L'utilisateur doit pouvoir créer un message
F1	L'utilisateur doit pouvoir récupérer un message
F2	L'utilisateur doit pouvoir récupérer la liste de tous les messages
F3	L'utilisateur doit pouvoir modifier un message
F4	L'utilisateur doit pouvoir supprimer un message

Fonction	Descriptions « user() && users »
G0	L'administrateur doit pouvoir créer un utilisateur
G1	L'administrateur doit pouvoir récupérer les informations d'un utilisateur
G2	L'administrateur doit pouvoir récupérer la liste et les informations de tous les utilisateurs
G3	L'administrateur doit pouvoir modifier les données d'un utilisateur
G4	L'administrateur doit pouvoir supprimer un utilisateur

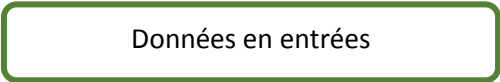
Fonction	Descriptions « MessageToUser() »
H0	L'administrateur doit pouvoir envoyer un message à un utilisateur en connaissant son id et du celui du message à envoyé
H1	L'administrateur doit pouvoir envoyer un message à un utilisateur en connaissant id de l'auteur et le contenu du message à envoyé

Fonction	Descriptions « MessageToGroup() »
I0	L'utilisateur doit pouvoir envoyer un message à un groupe en connaissant l'ID du group et ID du message
I1	L'utilisateur doit pouvoir envoyer un message (créer par lui) à un groupe en connaissant l'ID du group

## Note explicative avant de commencer la documentation fonctionnelle et technique

Dans la suite de ce document, nous avons nommer et donner une description a toutes les fonctions qui seront coder pour notre API.

Les données en entrées seront représentées par des carrés verts et les données en sortie par des rouges.

A green rounded rectangle with a thin border, containing the text "Données en entrées".

Données en entrées

A red rounded rectangle with a thin border, containing the text "Données en sortie".

Données en sortie

Pour la description technique, nous utiliseront à chaque fois une Méthodes CRUD et un URL.

**IMPORTANT :** Lors de la programmation, le codeur pourra s'il le souhaite retourner le message en langue anglaise.

## Description technique

F0

L'utilisateur doit pouvoir créer un message

Méthode : POST

Url : <http://localhost:8080/api/messages/id>

- Le contenu du message
- L'ID de l'auteur
- Le format de la réponse (XML, JSON)

- 201 , MESSAGE WAS CREATED
- Si non : ERROR 400

F1

L'utilisateur doit pouvoir récupérer un message

Méthode : GET

Url : <http://localhost:8080/api/message/id>

- L'ID de l'auteur
- Le format de la réponse (XML, JSON)

- 200
- Renvoi le message de l'auteur
- Si non ERROR 404

F2

L'utilisateur doit pouvoir récupérer la liste de tous les messages

Méthode : GET

Url : <http://localhost:8080/api/messages>

- La bonne ressource
- Le format de la réponse (XML, JSON)

- Renvoi tous les messages avec descriptifs
- Si non ERROR 404

F3

L'utilisateur doit pouvoir modifier un message

Méthode : PUT

Url : <http://localhost:8080/api/message/id>

- L'ID du message
- Le nouveau message
- Le format de la réponse (XML, JSON)

- UPDATE WAS SUCCES
- Si non : ERROR 400, « FAIL UPDATE
- ERROR 404 « MESSAGE NOT FOUND »

F4

L'utilisateur doit pouvoir créer un message

Méthode : DELETE

Url : <http://localhost:8080/api/message/id>

- L'ID du message
- Le format de la réponse (XML, JSON)

- 200 O, GOOD MESSAGE WAS DELETE
- Si message non trouvé : Erreur 404, MESSAGE NOT FOUND !

G0

L'administrateur doit pouvoir créer un utilisateur

Méthode : POST

Url : <http://localhost:8080/api/users/id>

- Les données d'un User (**username** (**unique**), **password** (password), **mail** (**unique**), dob, tel, firstName, lastName)
- Le format de la réponse (XML, JSON)

- 200 OK, nouveau user crée
- Si non crée : Erreur 405, message non trouvé
- 405

G1

L'administrateur doit pouvoir récupérer les informations d'un utilisateur

Méthode : GET

Url : <http://localhost:8080/api/user/2>

- L'ID de l'utilisateur
- Le format de la réponse (XML, JSON)

- 200
- Renvoi toutes les informations avec descriptifs de l'utilisateur
- Si non erreur 404

G2

L'administrateur doit pouvoir récupérer la liste et les informations de tous les utilisateurs

Méthode : GET

Url : <http://localhost:8080/api/users>

- Le format de la réponse (XML, JSON)

- 200
- Renvoi toutes les informations avec descriptifs des utilisateurs
- Si non ERROR 404

G3

L'administrateur doit pouvoir modifier les données d'un utilisateur

Méthode : PUT

Url : <http://localhost:8080/api/user/id>

- ID
- Les données d'un User (**username** (unique), **password** (password), **mail**(unique), dob, tel, firstName, lastName)

- 200 : UPDATE SUCCESSFUL FOR USER (nom)
- Si non : 400 ERROR UPDATE
- Erreur 404 si message non trouvé : USER NOT FOUND

G4

L'administrateur doit pouvoir supprimer un utilisateur

Méthode : DELETE

Url : <http://localhost:8080/api/user/id>

- L'ID de l'utilisateur
- Le format de la réponse (XML, JSON)

- 200 OK, USER DELETED
- 404 USER NOT FOUND
- 405

H0	L'administrateur doit pouvoir envoyer un message à un utilisateur en connaissant son id et du celui du message à envoyé
----	---

Méthode : POST

Url : <http://localhost:8080/api/messageToUser/id>

- L'ID du message (messageId)
- L'id de l'utilisateur(userId)
- Le format de la réponse (XML, JSON)

- 201 : MESSAGE WAS CORRECTLY SENDED
- 404 USER NOT FOUND/MESSAGE ID xx NOT FOUND
- 404 NOT FOUND !

H1	L'administrateur doit pouvoir envoyer un message à un utilisateur en connaissant id de l'auteur et le contenu du message à envoyé
----	---

Méthode : POST

Url : <http://localhost:8080/api/messageToUser/id>

- L'id de l'utilisateur(userId)
- L'id de l'auteur ( authorId)
- Le message (messageContent)
- Le format de la réponse (XML, JSON)

- 201 MESSAGE WAS CORRECTLY SEND
- 404 MESSAGE OR USER NOT FOUND
- 404 NOT FOUND !

I0	L'utilisateur doit pouvoir envoyer un message à un groupe en connaissant l'ID du group et ID du message
----	---

Méthode : POST

Url : <http://localhost:8080/api/messageToGroup/id>

- ID du group
- ID du message a envoyé
- Le format de la réponse (XML, JSON)

- 201, MESSAGE WAS SENDED
- Si non : Erreur 400, FAIL TO SEND MESSAGE
- 405



I1

L'administrateur doit pouvoir créer un utilisateur

Méthode : POST

Url : <http://localhost:8080/api/messageToGroup/id>

- ID du group
- ID de 'auteur
- Le contenu du message
- Le format de la réponse (XML, JSON)

- 201, MESSAGE WAS SENDED BY USER (name)
- Si non: Erreur 404, MESSAGE DOESN'T EXIST
- 400 ERROR TO SEND MESSAGE
- 404 GROUP NOT FOUND