API REST - TME1

1 TME 1 : Interrogation d'une API REST

Dans ce TME, l'objectif est de comprendre le fonctionnement d'une API REST, comment on peut interagir avec une API REST et qu'est-ce qu'un web service. Pour cela, nous vous fournissons une API REST qui correspond au serveur du site web du projet. Elle est hébergée sur des serveurs privés, vous ne verrez donc pas le code :)

1.1 De quels outils avons nous besoin dans ce TME ?

- une connection internet
- un navigateur web
- du logiciel Postman (https://www.postman.com/)
- un terminal

1.2 Web services

L'API que nous vous fournissons est composée de plusieurs services web qui sont détaillés par un ensemble d'attributs :

- Nom du service
- URL d'accès (avec protocolle HTTP)
- Une description
- Une liste de paramètres d'entrée
- Le message de sortie avec des exemples
- La liste des erreurs possibles

Vous trouverez ci-dessous la description de ces services. Prenez-en connaissance, vérifiez que vous avez bien compris ce qu'est un web service (faites le lien avec ce qui a été dit en TD).

1.2.1 Services liés à l'utilisateur

Nom du Web service : **createUser**
URL : POST /api/user
Description : Permet de créer un nouvel utilisateur
Paramètre(s) en entrée (dans le champs body) : login, password, confirmedpassword, lastname, fisrtname
Format de sortie : ok/erreur
Exemple de sortie : { "id" : "1" } ou ≪ Missing Fields ≫
Erreurs possibles : user existe déjà, bd injoignable, manque un paramètre

Nom du Web service : **login**
URL : POST /api/user/login
Description : permet de créer une clé de connexion validée pendant
Paramètre(s) en entrée (dans le champs body) : login, password
Format de sortie : clé/erreur
Exemple de sortie : { "status" : "403", "message" : "ogin et/ou le mot de passe invalide(s)" } ou {"status" : "200", "message" : "Login et mot de passe acceptés" }
Erreurs possibles : user inconnu (401), password incorrect (401), multiconnexion, erreur interne

Nom du Web service : **logout**
URL : DELETE /api/user/userid/logout
Description : permet de fermer une connexion (session) d'un utilisateur
Paramètre(s) en entrée :
Format de sortie : ok/erreur
Exemple de sortie : { "status" : "401", "message" : "Utilisateur inconnu" } ou "status" : "200", "message" : "session fermée" }
Erreurs possibles : aucun user ne correspond au userid en paramètre

Nom du Web service : **getUser**
URL : GET /api/user/userid
Description : permet d'afficher le user avec l'id userid
Paramètre(s) en entrée :
Format de sortie : ok(informarions sur le user)/erreur
Exemple de sortie : { "login" : "pikachu", "password" : "dlskdlsjg", "lastname" : "chu", "firstname" : Pika" ou { "status" : "404", "message" : "Not Found" }
Erreurs possibles : aucun user ne correspond au 'id' userid

Nom du Web service : deleteUser
URL : DELETE /api/user/userid
Description : permet de supprimer un utilisateur de la base de données users.
Paramètre(s) en entrée :
Format de sortie : ok/erreur
Exemple de sortie : « delete user 1 » dans le cas où user de id = 1 a été supprimé correctement ou {status :"401", "message" : "Utilisateur inconnu"} dans le cas où user avec id = userid n'existe pas
Erreurs possibles : aucun user avec id userid existe

Nom du Web service : **getUserInfo**
URL : GET /api/user/infos
Description : permet d'afficher le nombre d'utilisateurs dans la base
Paramètre(s) en entrée :
Format de sortie : { "count" : 2 }
Erreurs possibles :

1.2.2 Services liés aux amis

Nom du Web service : **createFriend**
URL : POST /apifriends/user/userid/friends
Description : permet de créer un nouveau lien d'amitié depuis le user de id userid.
Paramètre(s) en entrée : { "login" : "turbo400" } (permet de donner l'identifiant de l'utilisateur courant qui est connecté et qui fait des actions (en gros vous, qui voulez ajouter userid en ami)
Format de sortie : ok/erreur
Exemple de sortie : { "id" : "1" } ou { "status" : "401", "message" : "user inconnu" }, "on ne peut pas se suivre soi-même"
Erreurs possibles : il n'existe pas de user avec ce login, déjà amis, user(userid) n'existe pas

Nom du Web service : **getListFriends**
URL : GET /apifriends/user/userid/friends
Description : permet d'obtenir la liste de tous les amis du user avec id userid
Paramètre(s) en entrée :
Format de sortie : ok/erreur
Exemple de sortie : { "login_friends" : "pikachu, sacha, ..." } ou { "status" : "401", "message" : "user id n'existe pas dans la table friends" }
Erreurs possibles : aucun user n'existe avec l'id userid

Nom du Web service : **getFriendRelationship**
URL : GET /apifriends/user/userid1/friends/userid2
Description : permet d'obtenir le lien de relation entre userid1 et userid2
Paramètre(s) en entrée :
Format de sortie : ok/erreur
Exemple de sortie : { "login_friends" : "pikachu, sacha, ..." } ou { "status" : "401", "message" : "user id n'existe pas dans la table friends" }
Erreurs possibles : aucun user n'existe avec l'id userid

Nom du Web service : **deleteUser**
URL : DELETE /apifirends/user/userid/friends/friendid
Description : permet d'arreter de suivre un user
Paramètre(s) en entrée :

Format de sortie : ok/erreur
Exemple de sortie : { "status" : "200", "message" : "user {req.params.userid} unfollowed user {req.params.friendid}" } ou { "status" : "401", "message" : "user(s) inconnu(s)" }
Erreurs possibles : un des users n'existe pas, user (userid) ne suit pas user (friendid)

Nom du Web service : **getFriendInfo**
URL : GET /apifriends/user/userid/infos
Description : permet d'obtenir les informations sur les relations d'amitié de l'utilisateur userid
Paramètre(s) en entrée :
Format de sortie : JSON
Exemple de sortie : { "login_friends" : "pikachu, sacha, ..." }
Erreurs possibles : aucun user n'existe avec l'id userid

1.2.3 Services liés aux messages

Nom du Web service : **createMessage**
URL : POST /apimessages/user/userid/messages
Description : permet au user dont l'id est userid de créer un nouveau message
Paramètre(s) en entrée : texte_message, userid
Format de sortie : ok/erreur
Exemple de sortie : { "id" : "id_message" } ou { "status" : "401", "message" : "user(s) inconnu(s)" } + missing fields
Erreurs possibles : user avec userid n'existe pas

Nom du Web service : **setMessage**
URL : PUT /apimessages/user/userid/messages/messageid
Description : permet de modifier un message dont l'id est message id au user (userid)
Paramètre(s) en entrée : nv_message, userid, messageid
Format de sortie : ok/erreur
Exemple de sortie : { "message" : "nv_message" } ou { "status" : "401", "message" : "user inconnu" } ou encore { "status" : "401", "message" : "message n'existe pas"} + Missing fields
Erreurs possibles : aucun user (userid) n'a pas de message (messageid), userid n'existe pas

Nom du Web service : **deleteMessage**
URL : DELETE /apimessages/user/userid/messages/
Description : permet de supprimer un message (messageid) du user (userid)
Paramètre(s) en entrée (dans le body) : { "message" : "Il fait beau aujourd'hui :)" }
Format de sortie : ok/erreur
Exemple de sortie : { "status" : "200", "message" : "message {req.params.messageid} deleted" } ou { "status" : "401", "message" : "user inconnu" } + Message n'existe pas
Erreurs possibles : aucun user (userid) n'existe, user (userid) n'a pas de message (messageid).

Nom du Web service : **getListMessage**
URL : GET /apimessages/messages
Description : permet d'obtenir l'affichage de tous les messages de la base de donnée.
Paramètre(s) en entrée :
Format de sortie : ok/erreur
Exemple de sortie : { "messages" : "message1, message2, message3" }
Erreurs possibles :

Nom du Web service : **getListMessageFromFriend**
URL : GET /apimessages/user/userid/messages/friendid
Description : permet d'obtenir l'affichage de tous les messages d'un ami friendid de user dont l'id est userid.
Paramètre(s) en entrée :
Format de sortie : ok/erreur
Exemple de sortie : { "messages" : "message1, message2, message3" } ou { "status" : "401", "message" : "user inconnu" }
Erreurs possibles : aucun user (userid) n'existe.

Nom du Web service : **getListMessageFromAllFriend**

URL : GET /apimessages/user/userid/messages/friends
Description : permet d'obtenir l'affichage de tous les messages de tous les amisde user dont l'id est userid.
Paramètre(s) en entrée :
Format de sortie : ok/erreur
Exemple de sortie : { "messages" : "message1, message2, message3" } ou user inconnu + user sans ami + amis de l'user sans message
Erreurs possibles : voir plus haut.

Nom du Web service : **getInfoMessageUser**
URL : GET /apimessages/user/userid/infos
Description : permet d'obtenir les stats sur les messages de l'user userid.
Paramètre(s) en entrée :
Format de sortie : ok/erreur
Exemple de sortie : { "count" : "2" }
Erreurs possibles :

Nom du Web service : **getInfoAllMessage**
URL : GET /apimessages/infos
Description : permet d'obtenir les stats sur les messages de tous les users.
Paramètre(s) en entrée :
Format de sortie : ok/erreur
Exemple de sortie : { "count" : "20" }
Erreurs possibles :

1.3 Interaction avec le serveur via Postman et le navigateur web

Nous avons hébergé le serveur composé des web services à l'adresse suivante : http://technoweb.lip6.fr:4443.
Plusieurs points sont importants à comprendre pour interroger l'API REST du site web :

- Pour accéder aux webservices, il suffit de concaténer l'adresse web avec les URL associées à chaque service. Par exemple, pour avoir les informations sur les messages, on peut utiliser l'URL : http://technoweb.lip6.fr/apimessages/infos.
- Le protocole HTTP est important à regarder. Seule l'interrogation par GET est possible sur un navigateur : saisir une URL sur un navigateur revient à interroger le serveur avec un protocole HTTP GET. Pour les autres modes de transmission, on doit fonctionner différemment.

De façon générale, quand on souhaite tester une API REST à distance, on utilise le logiciel Postman. Ce logiciel permet de saisir l'ensemble des requêtes de test, sans restriction de protocole HTTP GET. Vous pouvez alors pour chaque service web préciser le protocole HTTP utilisé, l'URL du service web et les paramètres associés.

Une autre technique consiste en utiliser la commande `curl` sur le terminal (https://curl.se/docs/manpage.html). Un exemple d'utilisation pour le service web login : `curl -X POST -H "Content-Type: application/json" -d '{"login":"pikachu","password":"1234"}' http://technoweb.lip6.fr:4443/api/user/login`

1.4 Travail à faire

1. Ouvrir Postman. Créer une collection intitulée "testBirdy-intro". Créer un ensemble de requêtes qui permettent de :
   - Créer 4 utilisateurs
   - Créer des relations d'amitié entre ces utilisateurs
   - Créer 1 ou 2 messages pour 3 des utilisateurs
   - Afficher les informations des utilisateurs, amis, messages
   - Apporter des modifications, supprimer des utilisateurs, amis, messages
   - Connecter et déconnecter un utilisateur
   - ....

Vous devez utiliser l'ensemble des services web décrits en section 1.2. Pour information, la ligne paramètre en entrée dans la description des services doit être saisie dans le champ "body" de la requête (et non Params).

2. Créer aussi des requêtes qui comportent volontairement des erreurs : utilisateur inexistant, paramètres inexistants, etc...

3. Pour les requêtes GET, afficher les résultats sur un navigateur.

4. Tester l'interrogation de l'API avec des commandes curl sur votre terminal. Enregistrez ces commandes sur un fichier, cela peut vous servir tout au long du projet pour tester votre code.