MUSIC SHEET WRITER
BILAN D'ARCHITECTURE D'AOUT

Jonathan Racaud, Antoine Simon, Jeremy Harrault, Julien Blondeel, Simon Daguenet, Florian Corradin

MUSIC SHEET WRITER

Objectifs du document

Résumé

Ce document présente l'architecture du projet Music Sheet Writer. Ce dernier est composé de plusieurs projets : un logiciel d'édition de partition à destination de Windows et Mac OS, d'un site internet vitrine ainsi que des applications mobiles pour iOS, Android et Windows Phone bénéficiant de fonctionnalités de partage.

Nous décrivons dans ce document les objectifs ayant un impact sur l'architecture du projet. Nous rappelons aussi les diverses contraintes auxquelles il fait face.

Chacune des parties suivant ce rappel concerne un aspect spécifique de chacun des projets. Nous commençons par un rappel de l'architecture global du projet, puis nous donnons la liste des cas d'utilisation. Nous représentons ensuite à l'aide de diagrammes UML (package et classe) l'architecture de chacun des projets et nous terminons enfin par les processus (diagrammes de séquences UML) implémentant les cas d'utilisations présentés.

Glossaire

– A –

API (*Application Programming Interface*) : une API est une interface de programmation. C'est une porte d'accès à des fonctionnalités en faisant abstraction de leurs fonctionnements.

– H –

HTTP (*Hypertext Transfer Protocol*) : HTTP est un protocole de communication client-serveur. Il peut fonctionner sur n'importe quelle connexion fiable. Les clients HTTP les plus connus sont les navigateurs Web permettant à un utilisateur d'accéder à un serveur contenant les données.

– R –

REST (*Representation State Tranfer*) : REST est un style d'architecture logiciel qui, la plupart du temps, est utilisé pour implémenter un Web Service. La communication se fait donc entre un client et serveur et utilise le protocole HTTP.

– W –

Web Service : un Web Service est un programme informatique de la famille des technologies web permettant la communication et l'échange de données entre applications. Il s'agit donc d'un ensemble de fonctionnalités exposées sur internet ou sur un intranet.

Description du document

Titre	Music Sheet Writer : Bilan d'architecture d'aout
Date	09/08/2015
Auteur	Jeremy Harrault
Responsable	Jonathan Racaud
E-mail	musicsheetwriter_2017@labeip.epitech.eu
Sujet	Bilan d'architecture d'aout
Version du modèle	1.0

Tableau des révisions

Date	**Auteur**	**Section(s)**	**Commentaire**
18/07/2015	Jeremy HARRAULT	Toutes	Création du document
19/07/2015	Jeremy HARRAULT	8. Implémentation 9. Vue données	Définition des ressources de l'API REST, définition de la base de données.
26/07/2015	Jeremy HARRAULT	9. Vue données	Modification du schéma de base de données.
09/08/2015	Jeremy HARRAULT	8. Implémentation 9. Vue données	Définition des entrées/sortie de l'API REST, Modification du schéma de base de données.
10/08/2015	Jonathan RACAUD	1. Introduction	Rédaction des points 1.1 et 1.2
14/08/2015	Simon DAGUENET	3. Architecture, buts et contraintes	Rédaction de la partie.
15/08/2015	Jonathan RACAUD	2. Représentation de l'architecture globale	Rédaction
15/08/2015	Jeremy HARRAULT	6. Vue Processus 8. Implémentation	Définition du séquencement de certains processus, Modification des entrées/sorties de l'API REST
16/08/2015	Florian CORRADIN	3. Architecture, buts et contraintes	Ajout architecture site web et mobile
16/08/2015	Julien BLONDEEL	6. Vue processus	Ajout des diagrammes de séquences du logiciel
16/08/2015	Jonathan RACAUD	4. Vue global 6. Vue processus 8.1.1 Vue globale	- Ajout des use cases - Ajout des premiers diagrammes de séquences des applications mobiles - Ajout du diagramme de package du logiciel

Table des matières

1. Introduction

1.1. Rappel de l'EIP

EPITECH est l'école de l'expertise informatique, transformant une passion en véritable expertise. L'apprentissage à EPITECH est fondé sur une pédagogie par projets, individuels ou en groupe, validant un certain nombre de connaissances et de notions à assimiler. Tout au long de leur cursus, les étudiants se familiarisent avec le milieu professionnel, notamment grâce aux stages en première, troisième et cinquième année d'une période de quatre à six mois. L'école forme les étudiants à s'adapter à des situations inhabituelles avec la mise en place de rush (projets à réaliser sur un week-end, sur des sujets et notions dont les élèves n'ont aucune connaissance) ou le départ à l'international pendant leur quatrième année ; année durant laquelle l'étudiant va devoir faire preuve d'autonomie et de capacité d'adaptation.

Les Epitech Innovative Projects sont des projets à réaliser sur le cycle master du cursus Epitech. Ils sont conçus à la manière d'un véritable projet entrepreneurial, dans toutes ses composantes : business, techno, design & communication. Un EIP est appelé à devenir une start-up viable. Le but de l'EIP est donc de faire découvrir aux étudiants le monde de l'entreprenariat en leur demandant de mettre un place un projet et de le réaliser en faisant face à des difficultés qu'ils n'avaient jusqu'alors pas rencontrées. Le principal obstacle est la gestion de groupe composé de membres dispersés dans des pays différents, faisant face alors aux problèmes de gestion du temps et des zones horaires pour leur quatrième année. Les problématiques de communication et de vente du produit sont aussi abordées.

1.2. Contexte et périmètre du projet

Music Sheet Writer est un logiciel d'édition de partition destiné aux musiciens néophytes et amateurs qui n'ont pas forcément les connaissances théoriques du solfège pour écrire leurs compositions. Il se présente donc comme tout logiciel d'édition de partition existant, mais apporte une fonctionnalité majeur : la génération d'une partition depuis un piano ou une guitare branchés à l'aide d'un câble JACK ou d'une interface audio USB.

Le mot d'ordre de Music Sheet Writer est d'être simple d'utilisation. En effet, en ajoutant cette fonctionnalité, nous simplifions la phase d'écriture lors de la composition d'une musique. Laissant l'utilisateur se concentrer sur la musique avant son écriture. Bien entendu, les musiciens aguerris ne seront pas en reste puisque Music Sheet Writer incorporera les outils qui leur permettront d'écrire leurs musiques de manière très précise.

Ce projet est composé de plusieurs projets. Le principal est le logiciel d'édition de partition à proprement parler. Il sera disponible à la fois sur les plateformes Windows et Mac OS. Le second projet est le site internet qui permettra de vendre le logiciel. Ce site internet aura aussi des fonctionnalités communautaires basiques (partage de partition, suivi d'utilisateurs, mise en favoris de partitions…). Enfin, trois applications mobiles seront aussi développées et elles permettront d'accéder à l'aspect communautaire développé autour de Music Sheet Writer.

2. Représentation de l'architecture globale

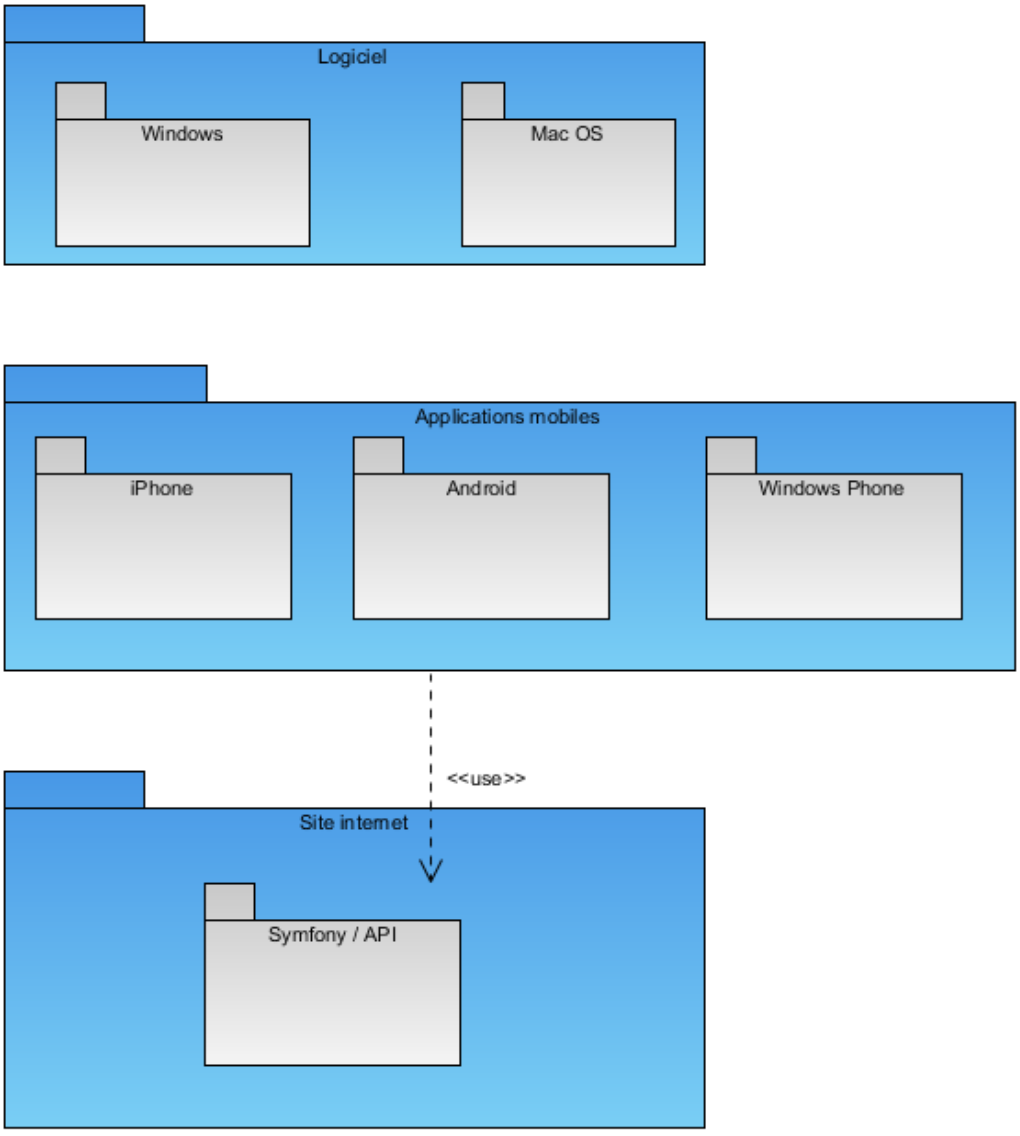
Le projet Music Sheet Writer est composé de nombreux sous-projets. Le principal est le logiciel d'édition de partition, destiné aux plateformes Windows et Mac.

Le second est le site internet qui servira de vitrine pour le logiciel et offrira une plateforme d'achat. Ce dernier donnera aussi accès à des fonctions communautaires telles que le partage de partition.

Enfin, les applications mobiles pour iPhone, Android et Windows Phone permettront d'avoir accès aux fonctionnalités communautaires développées sur le site internet. Chacune de ces applications sera développée dans leurs langages natifs.

Le diagramme ci-dessous donne une représentation graphique simplifiée de l'architecture du projet et de leurs intéractions.

Le reste de ce document décrit les cas d'utilisations, la logique et l'architecture de chacun de ces projets.



3. Architecture, buts et contraintes

3.1. Objectifs spécifiques ayant un impact sur l'architecture

Vous trouverez ci-dessous la liste des objectifs du projet ayant un impact sur ce dernier en termes d'architecture :

- Le logiciel :
  - La lecture d'une partition se fera avec des sons MIDI.
  - Nous utiliserons des librairies pour l'édition et la visualisation des partitions.
- Le site internet :
  - On utilisera HTTPS pour la sécurisation des données.
  - Le système de paiement se fera à l'aide d'une API tel que PAYPAL.

3.2. Contraintes fonctionnelles

Vous trouverez ci-dessous la liste des contraintes fonctionnelles des projets :

- Un projet Music Sheet Writer comprendra un fichier audio et un fichier MusicXML.

3.3. Contraintes non fonctionnelles

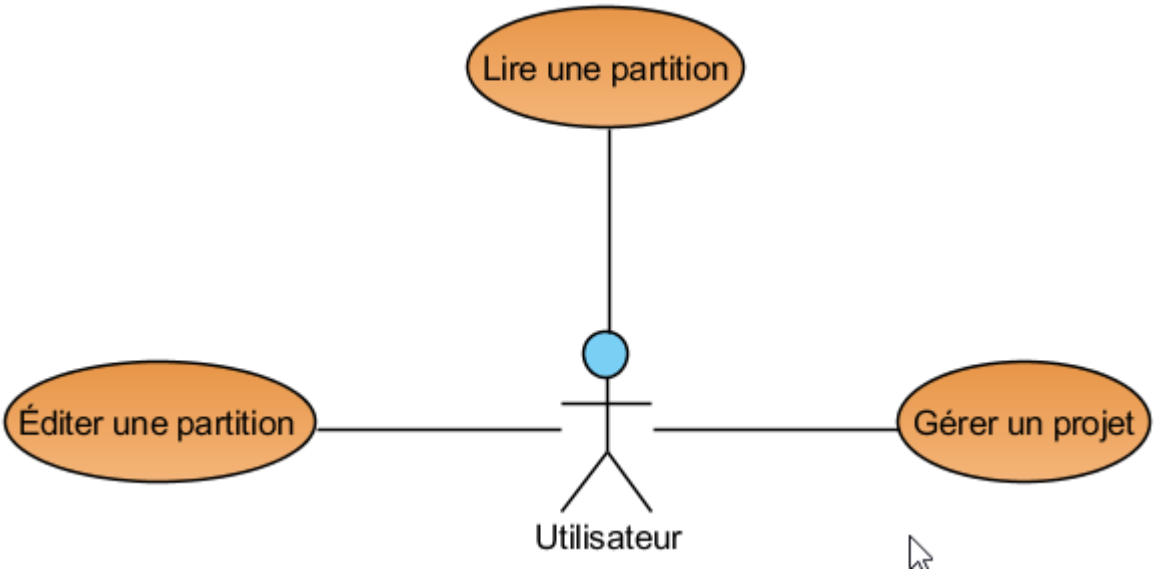
Vous trouverez ci-dessous la liste des contraintes non fonctionnelles du projet :

- Contraintes liées au logiciel :
  - Le solfège étant un système d'écriture universel et suivant des règles précises, les partitions que nous créerons respecteront ces dernières.
  - Dans le but d'avoir des fichiers qui peuvent être utilisés par d'autres logiciels, les partitions générées devront être compatibles avec le standard MusicXML.
  - Le logiciel doit être porté sur Windows et Mac.
- Contrainte liée aux applications mobiles :
  - Étant présent sur les plateformes Android, iOS et Windows Phone, ces dernières devront être développée indépendamment les unes des autres et dans leurs langages natif.
- Contraintes liées au site internet :
  - La transmission des données devra être sécurisée.
  - Les données sensibles des utilisateurs devront être sécurisées.
  - Le site internet devra être compatible avec deux bases de données différentes.

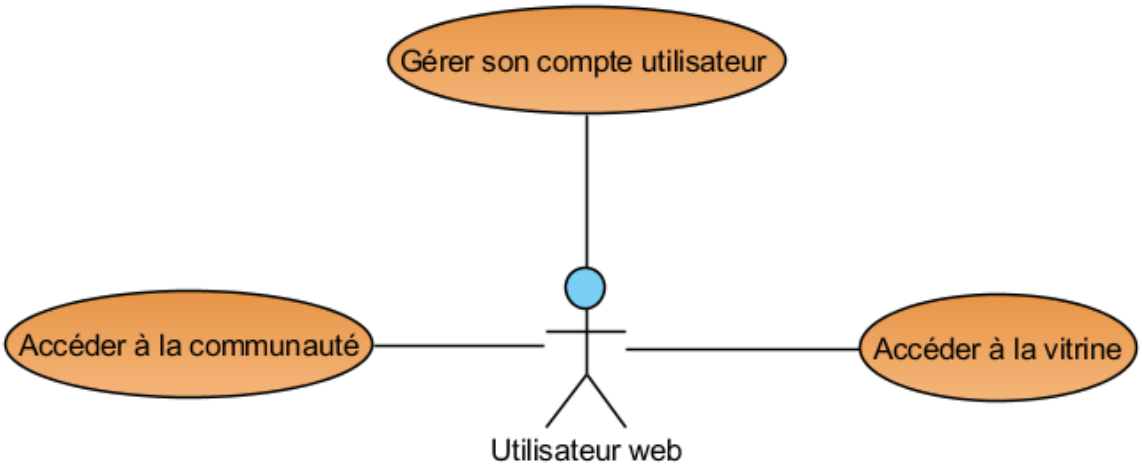
4. Vue globale du projet

4.1. Cas d'utilisations principaux

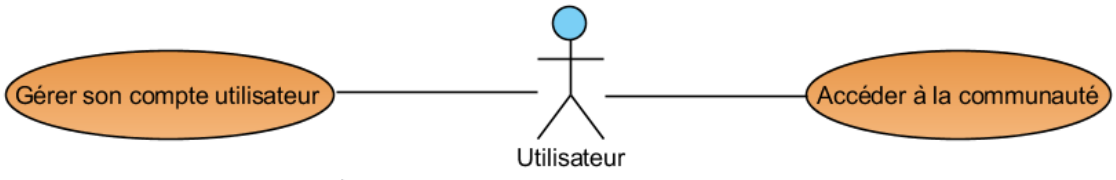
4.1.1. Le logiciel



4.1.2. Le site internet



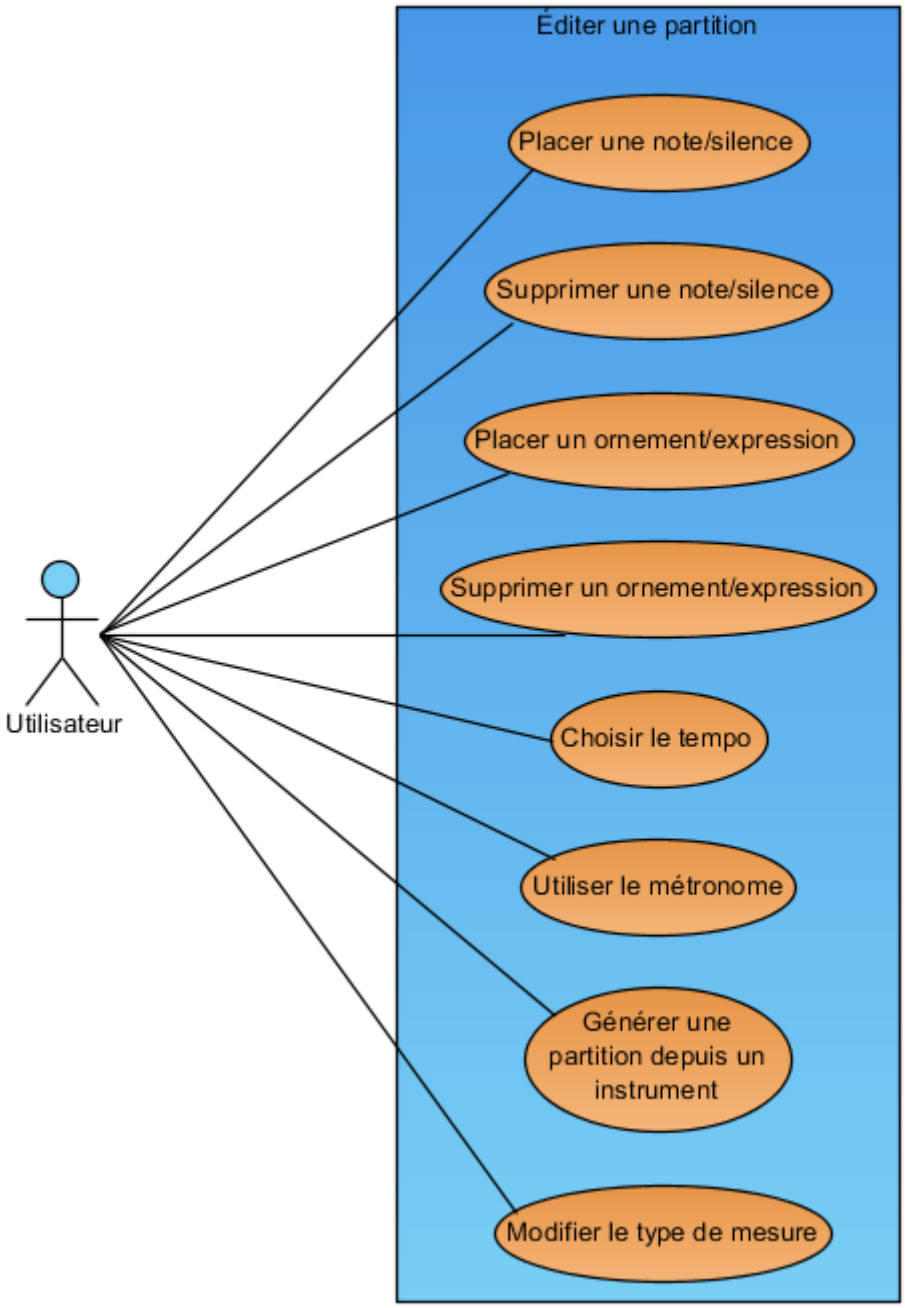
4.1.3. Les applications mobiles



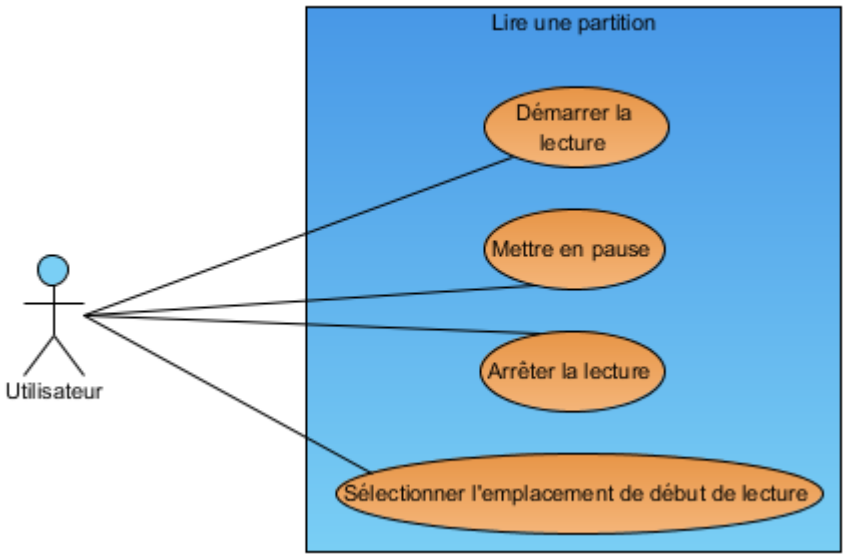
4.2. Cas d'utilisation détaillés

4.2.1. Le logiciel

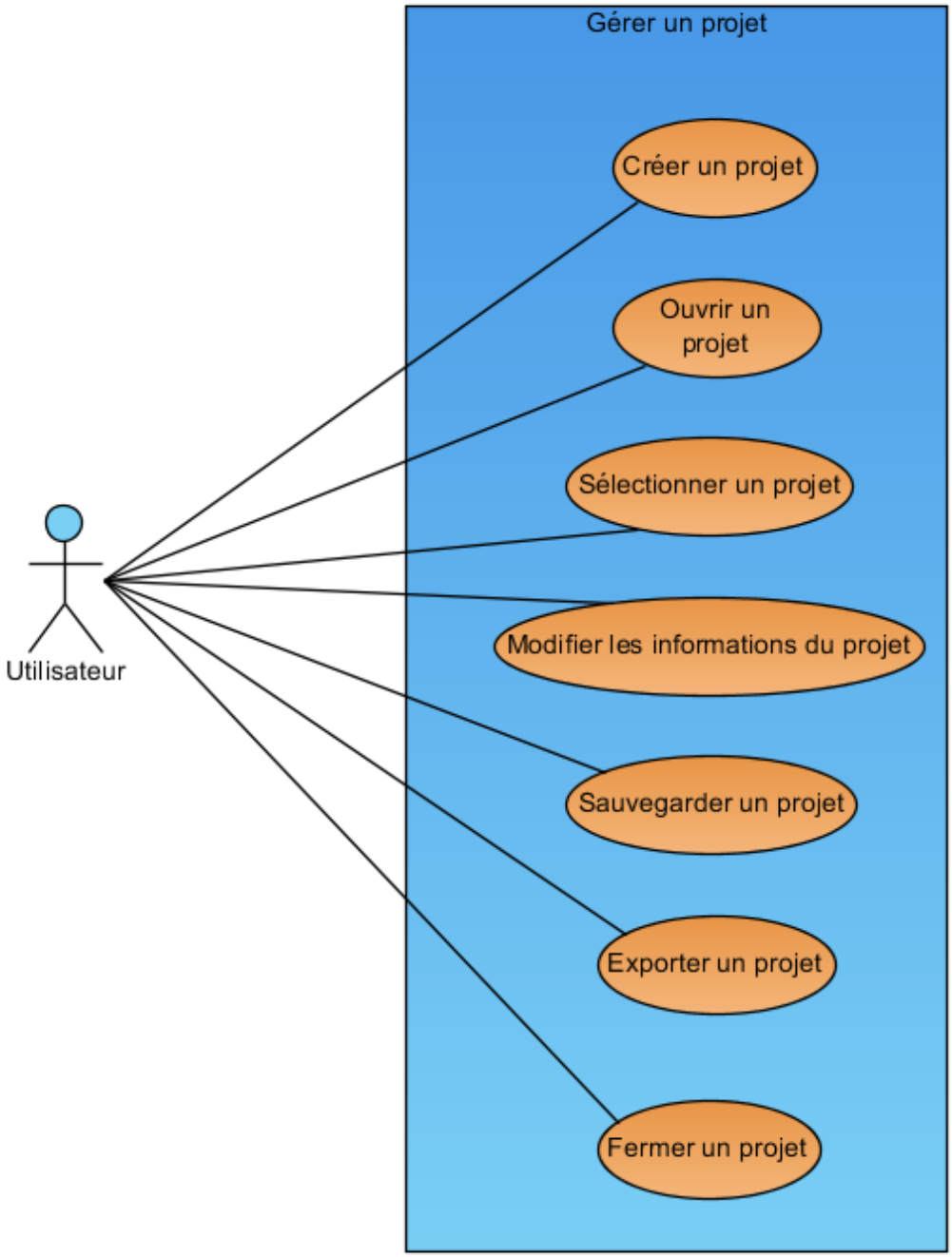
4.2.1.1. Éditer une partition



4.2.1.2. Lire une partition

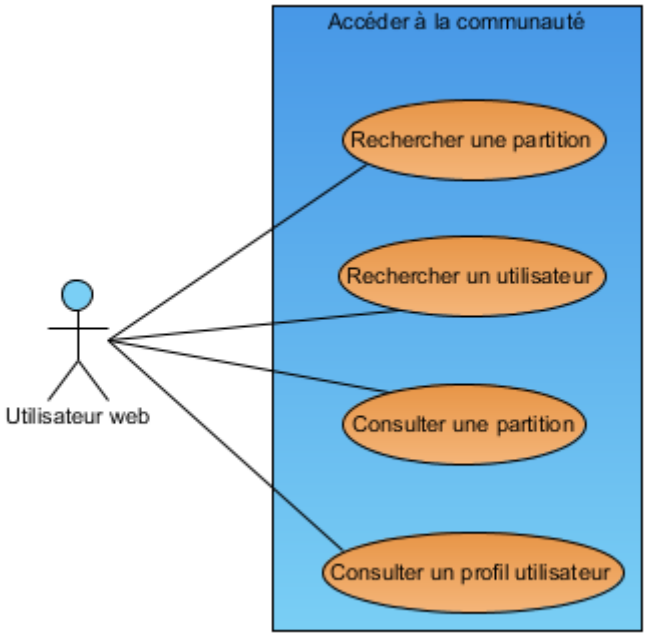


4.2.1.3. Gérer un projet

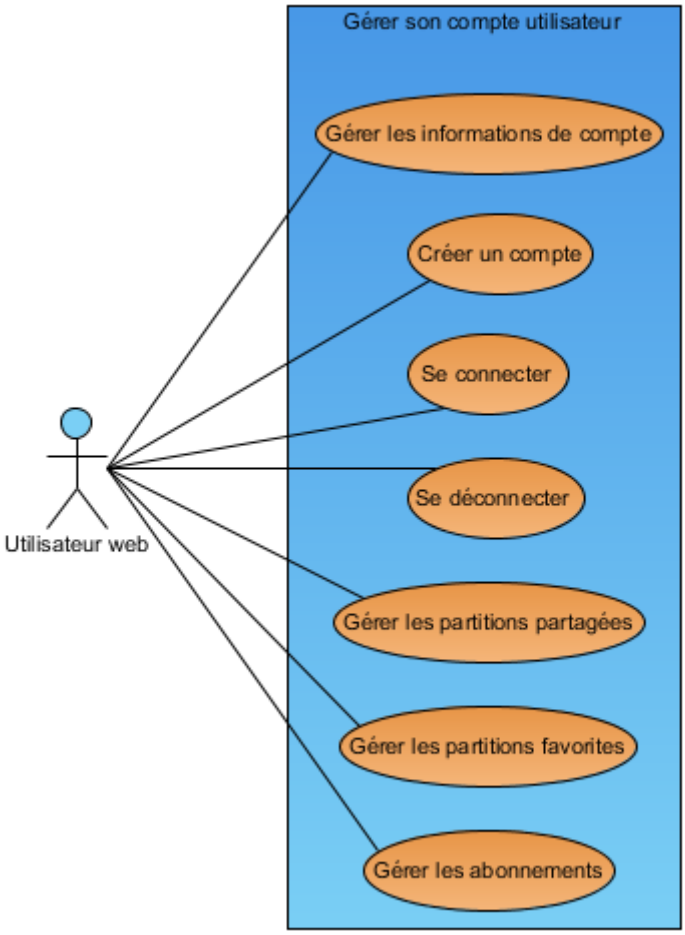


4.2.2. Le site internet

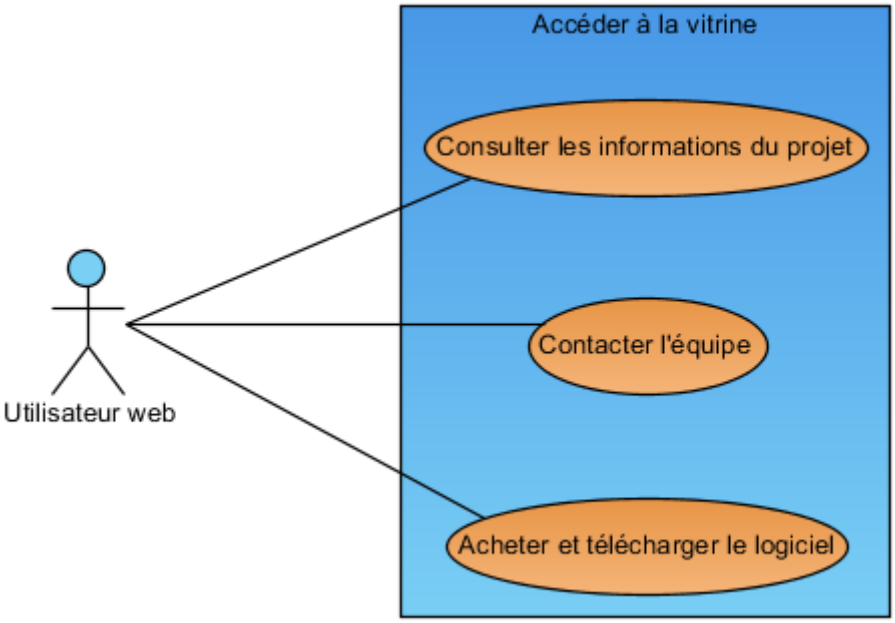
4.2.2.1. Accéder à la communauté



4.2.2.2. Gérer son compte utilisateur

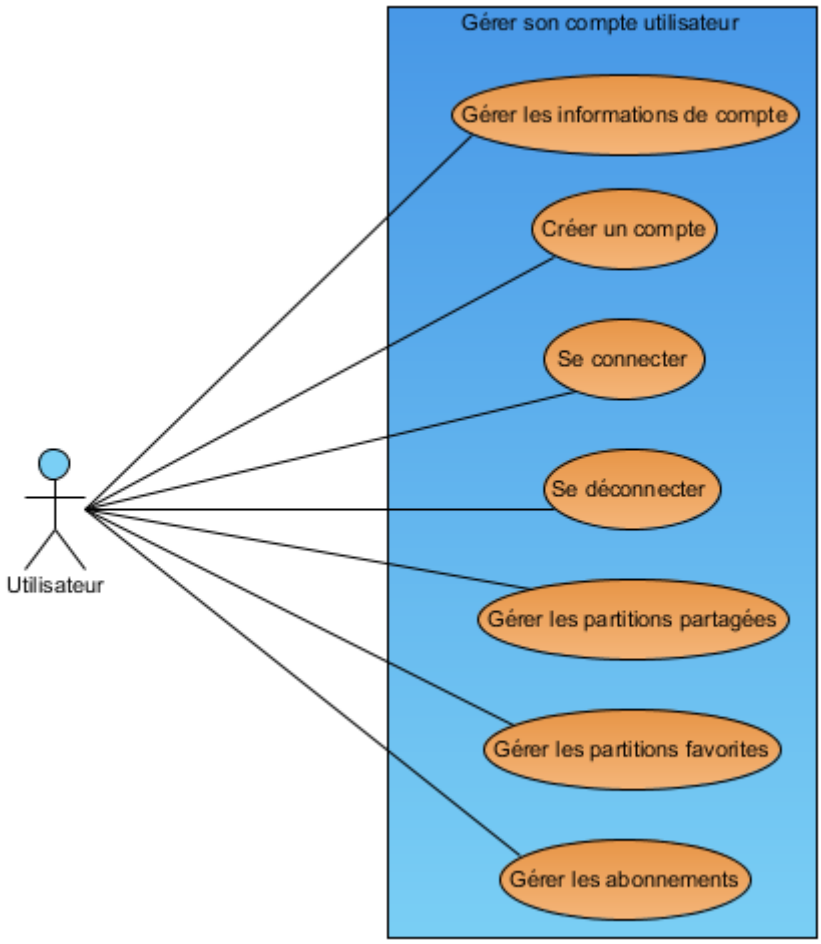


4.2.2.3. Accéder à la vitrine

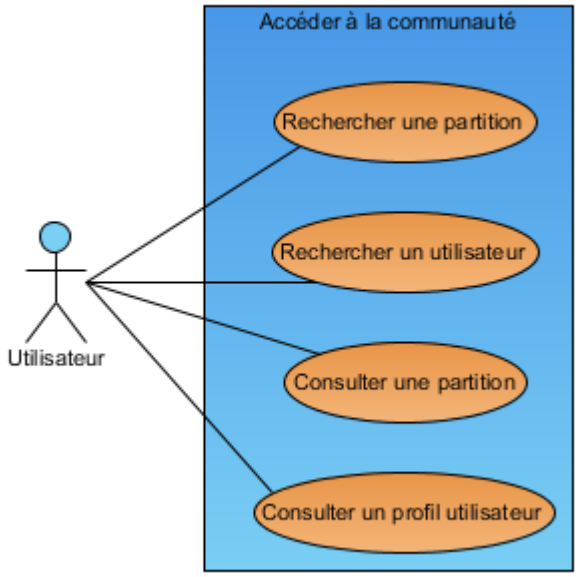


4.2.3. Les applications mobiles

4.2.3.1. Gérer son compte utilisateur



4.2.3.2. Accéder à la communauté



5. Vue logique de l'application

5.1. Le logiciel

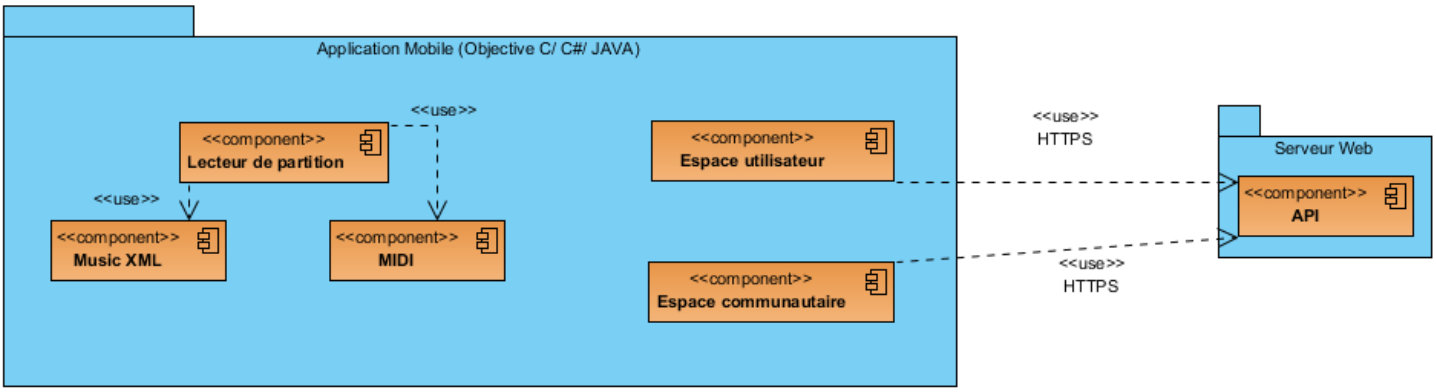
5.2. Les applications mobiles

L'architecture et la relation entre les différents composants des applications mobiles sont les mêmes sur les plateformes Android, iOS, Windows Phone.

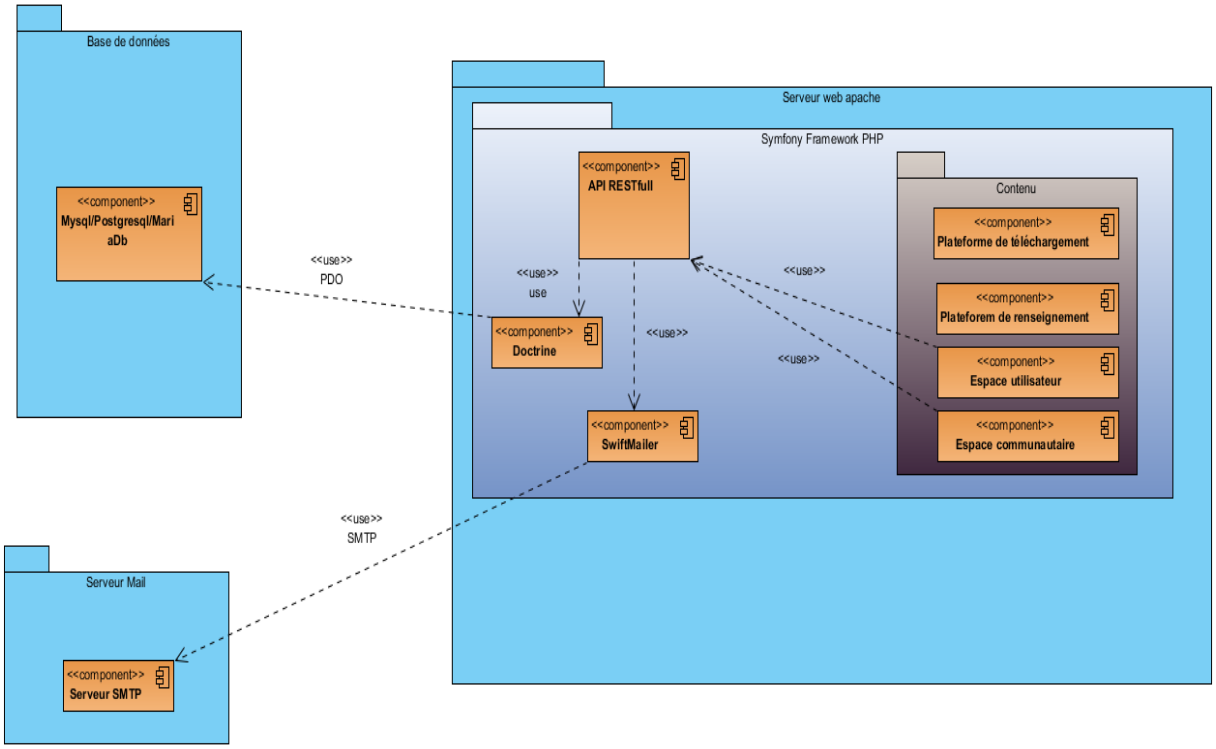
Chacune de ces applications sera développé dans son langage natif, Java pour Android, Objective C pour iOS et C# pour Windows.

Une application mobile comportera un lecteur de partition. Celles-ci seront traduites depuis des fichiers formatés en Music XML ou en MIDI. La traduction sera effectué par une bibliothèque Music XML et une MIDI compatible sur les trois plateformes de développement.

La deuxième partie comportera un espace dédié à l'utilisateur où celui-ci pourra avoir accès à la communauté Music Sheet Writer et accéder à son profil. Cette partie sera en relation avec une API REST hébergé sur le serveur web.



5.3. Le site internet



La partie site web sera développée en PHP sous un serveur apache. Pour organiser les différentes parties et la mise en place de l'API on utilisera le framework Symfony 2.

Il y a deux composants principaux qui seront l'API qui contiendra toutes les fonctionnalités liées à l'espace communautaire et utilisateur et le site Web.

L'API aura accès à la base de données grâce au service Doctrine de Symfony qui peut gérer plusieurs bases de données.

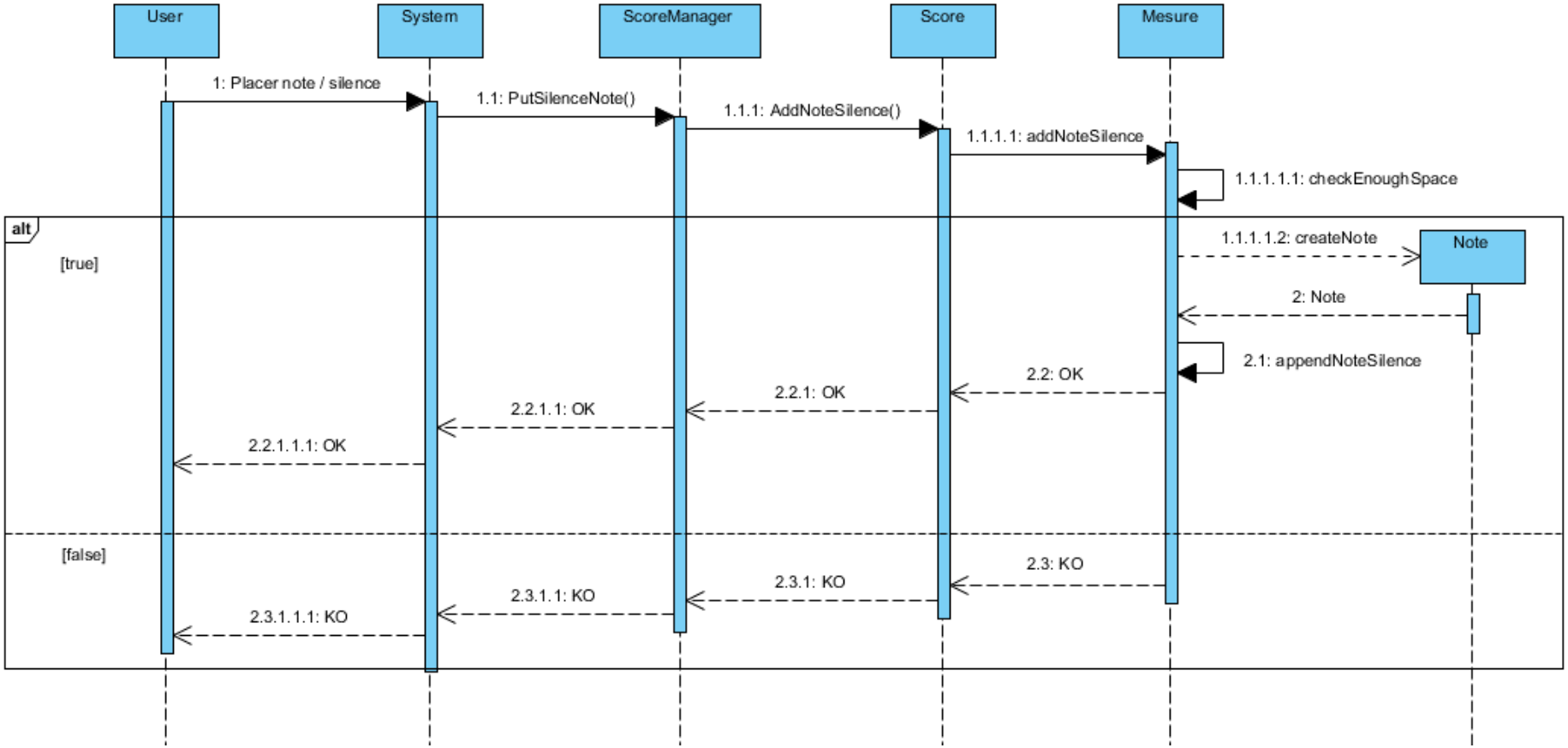
Pour l'envoie des emails on passera par le service SwiftMailer de Symfony.

6. Vue processus

6.1. Le logiciel

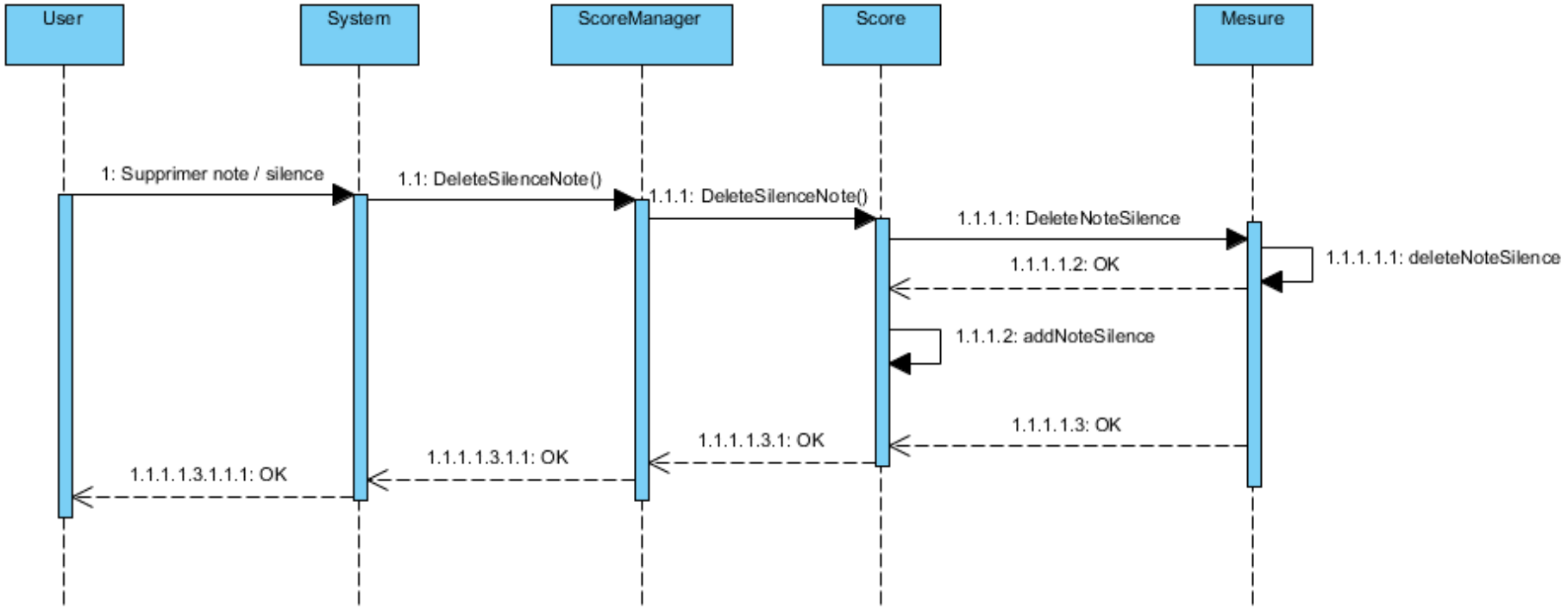
6.1.1. Éditer une partition

6.1.1.1. Placer une note/silence

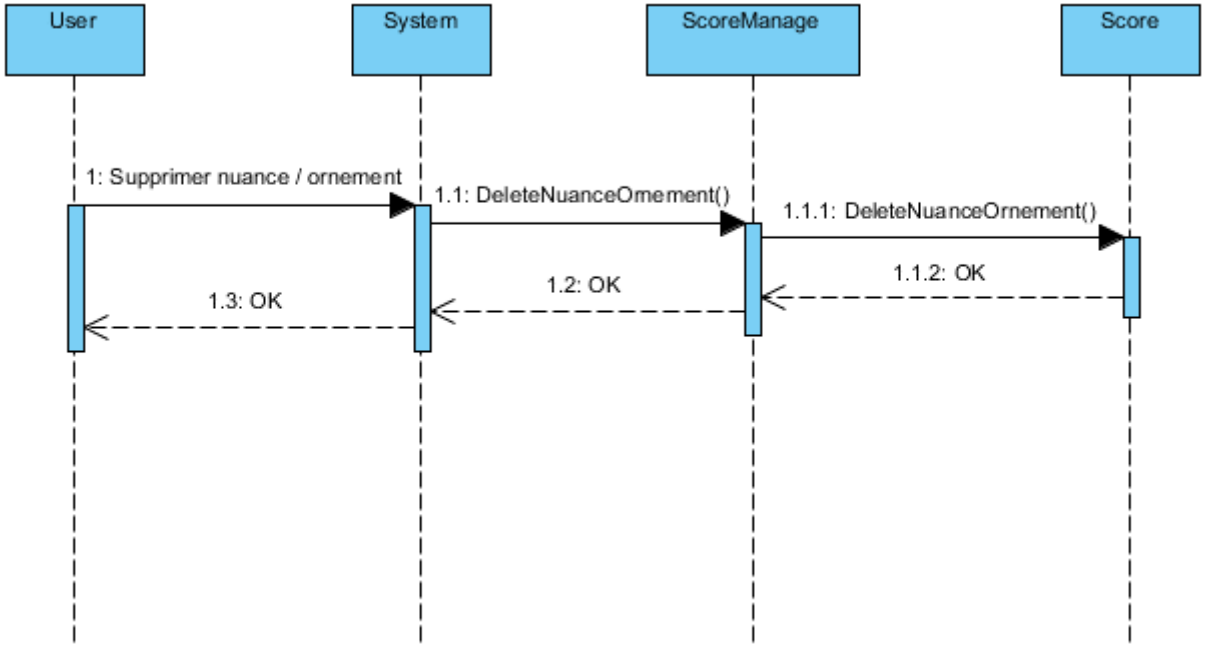


6.1.1.2. Placer un ornement/expression

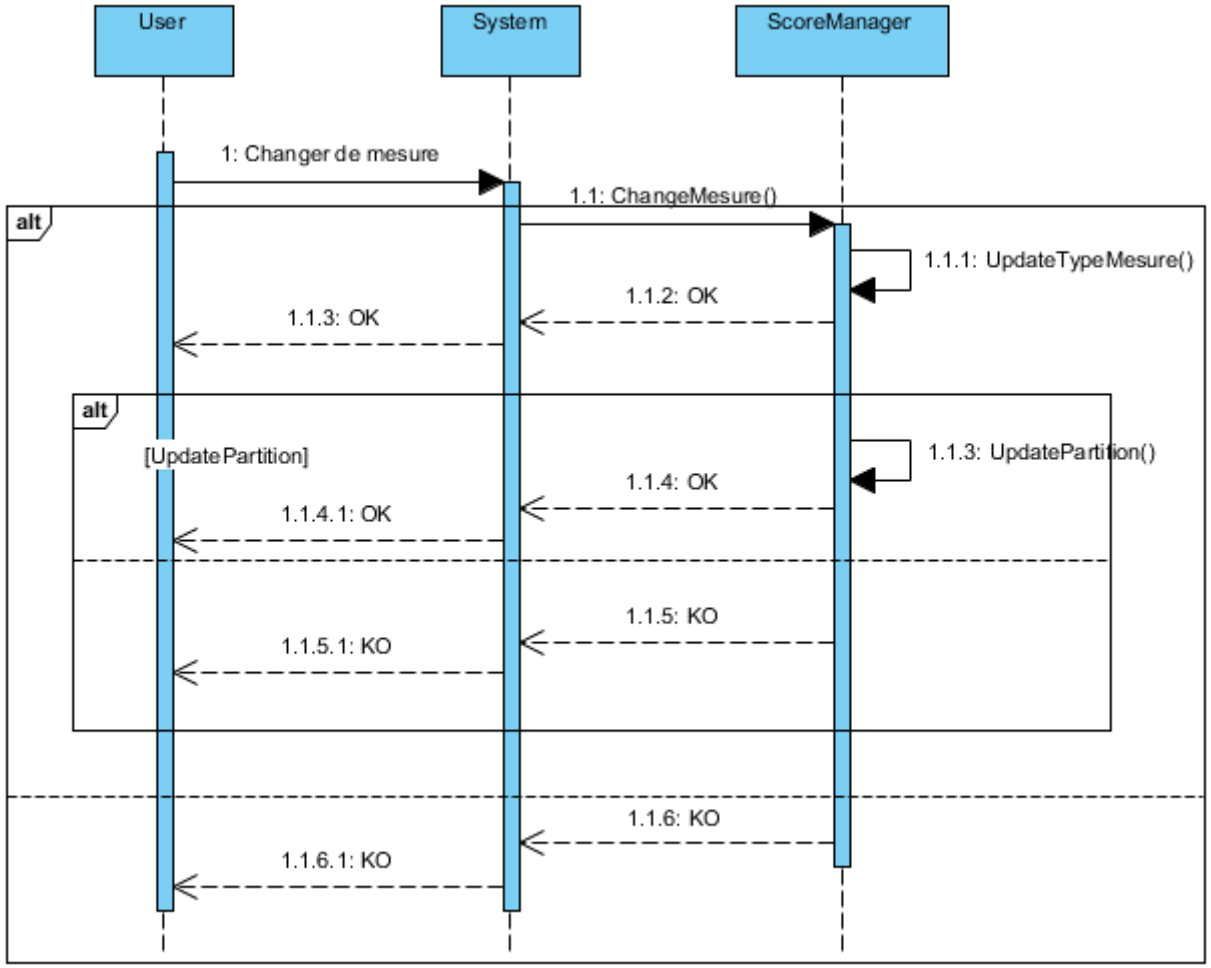
6.1.1.3. Supprimer une note/silence



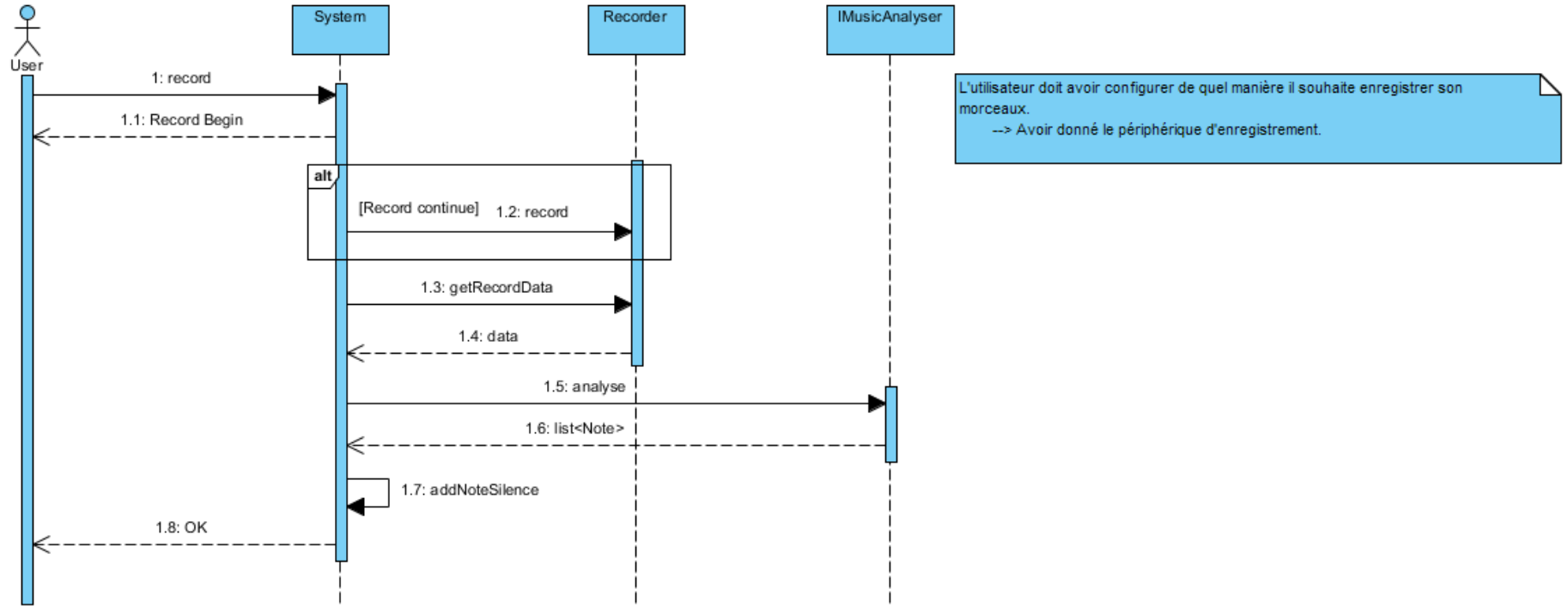
6.1.1.4. Supprimer un ornement/expression



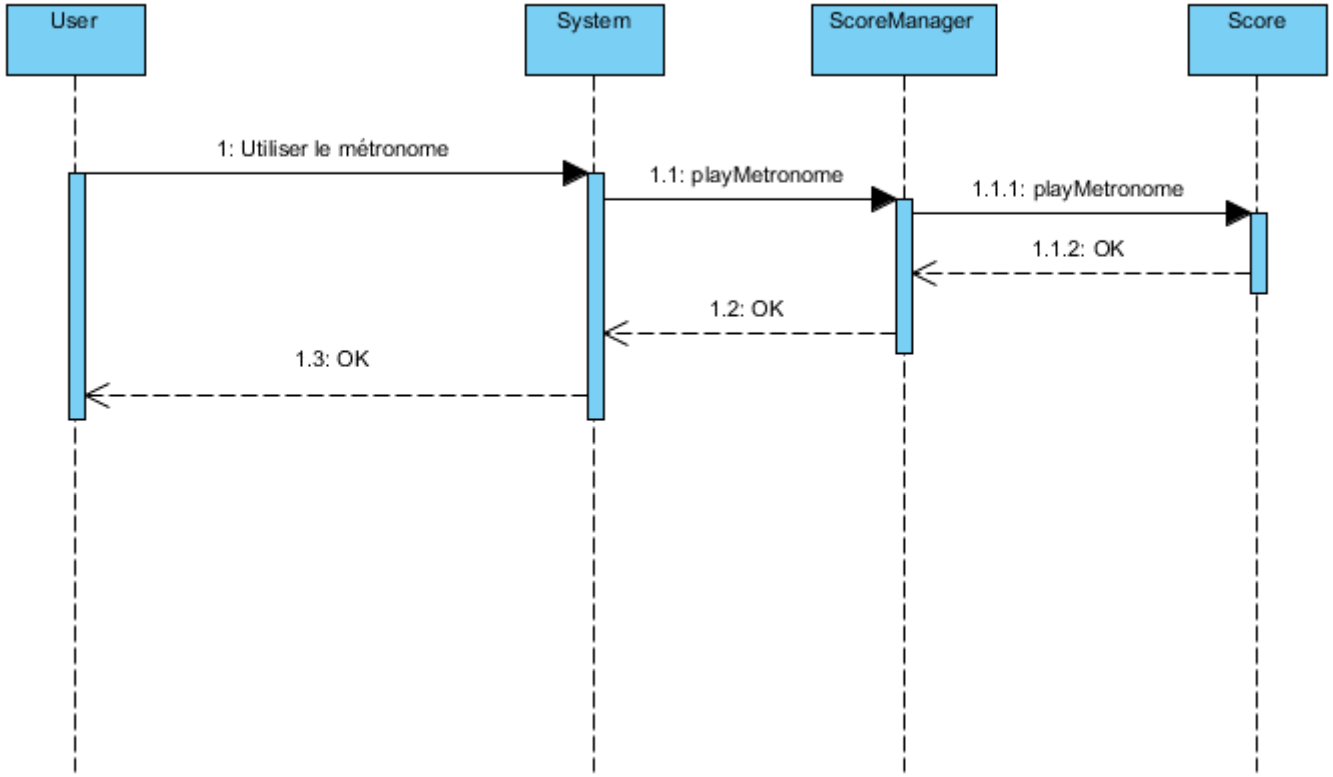
6.1.1.5. Modifier le type de mesure



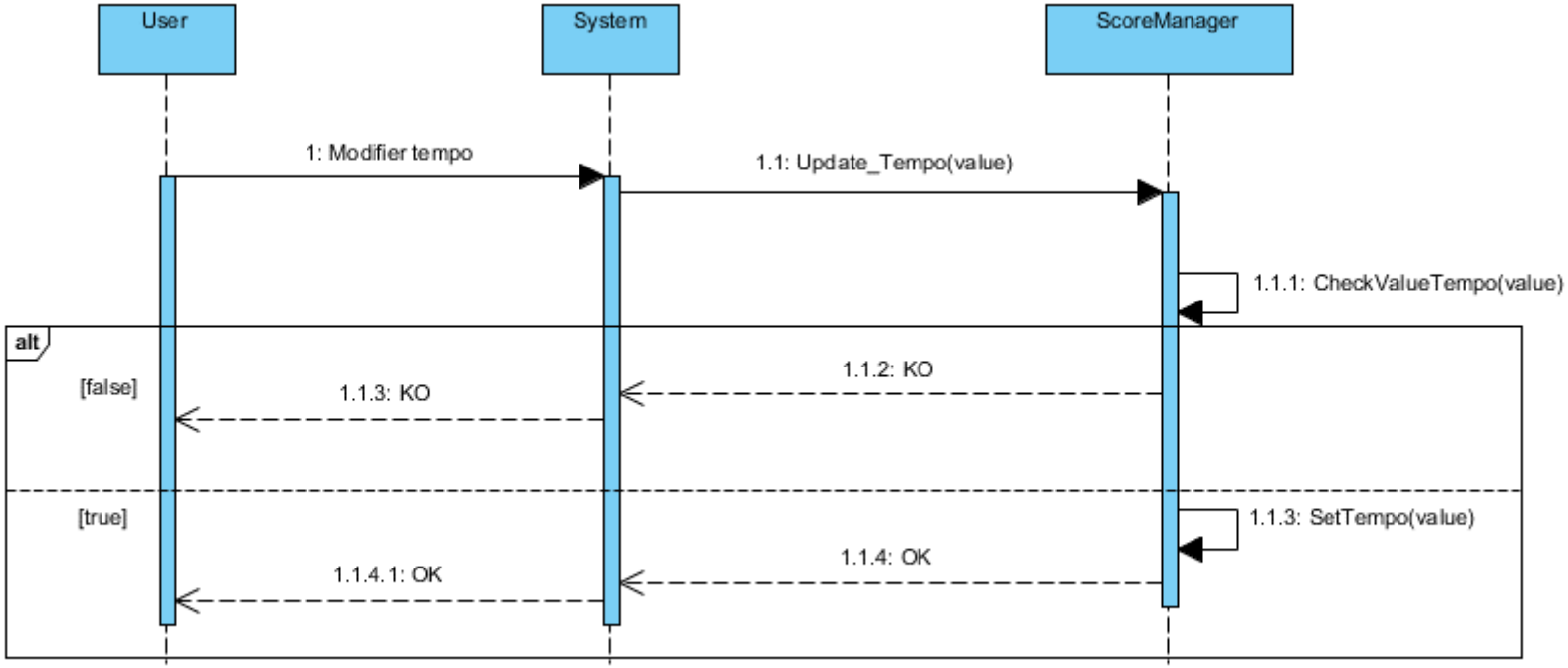
6.1.1.6. Générer une partition depuis un instrument de musique



6.1.1.7. Utiliser le métronome

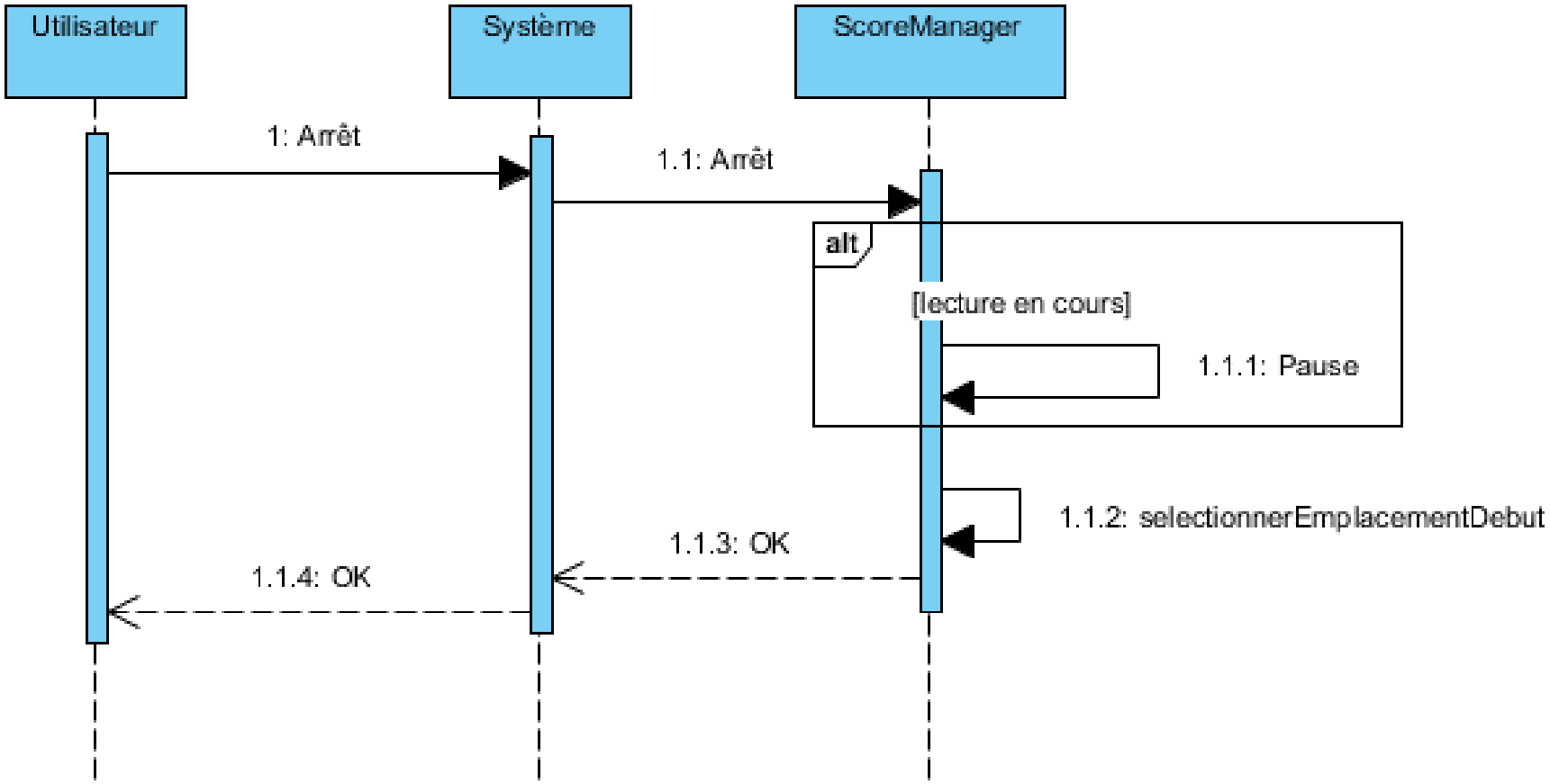


6.1.1.8. Choisir le tempo

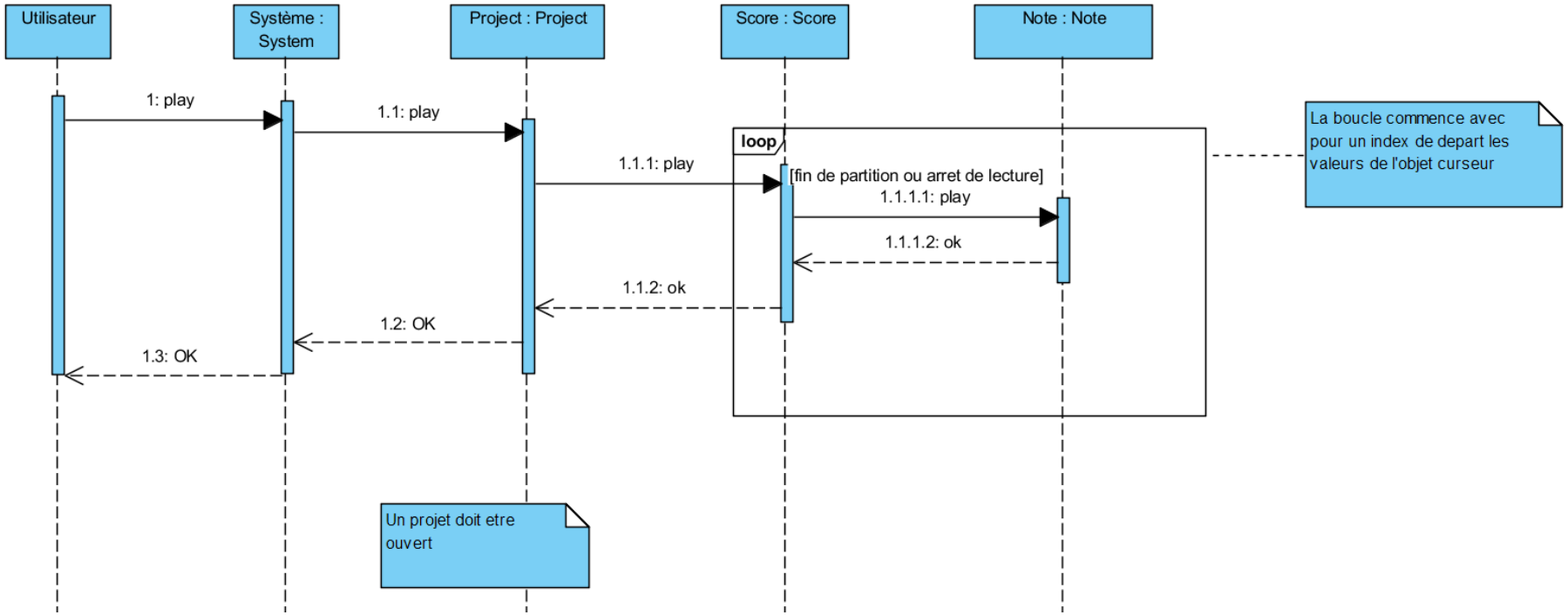


6.1.2. Lire une partition

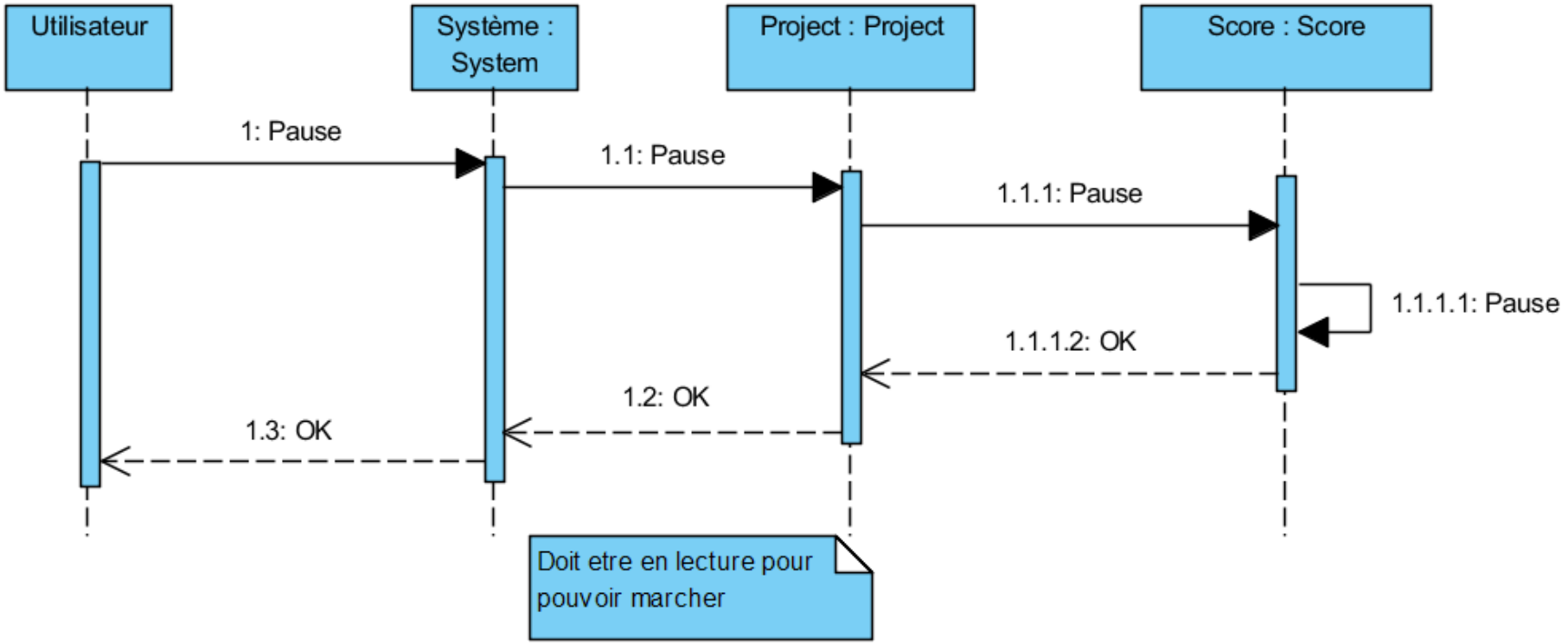
6.1.2.1. Arrêter la lecture



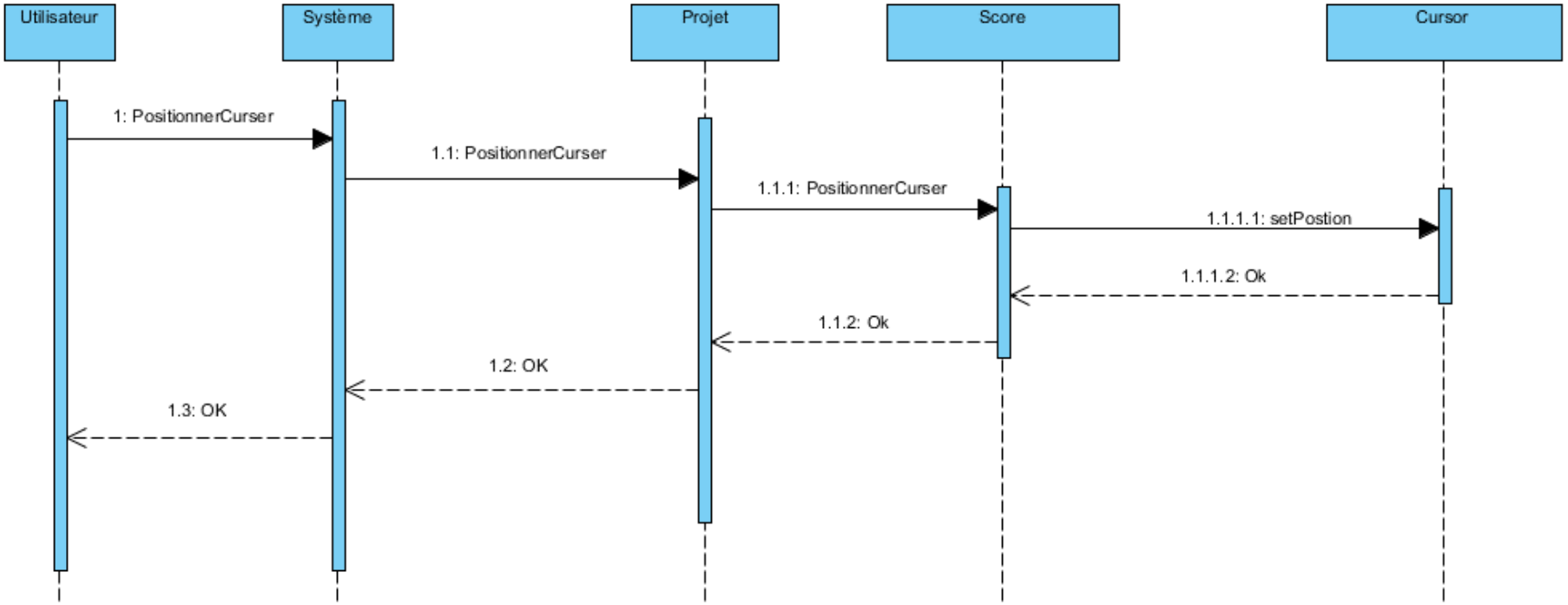
6.1.2.2. Démarrer la lecture



6.1.2.3. Mettre en pause la lecture

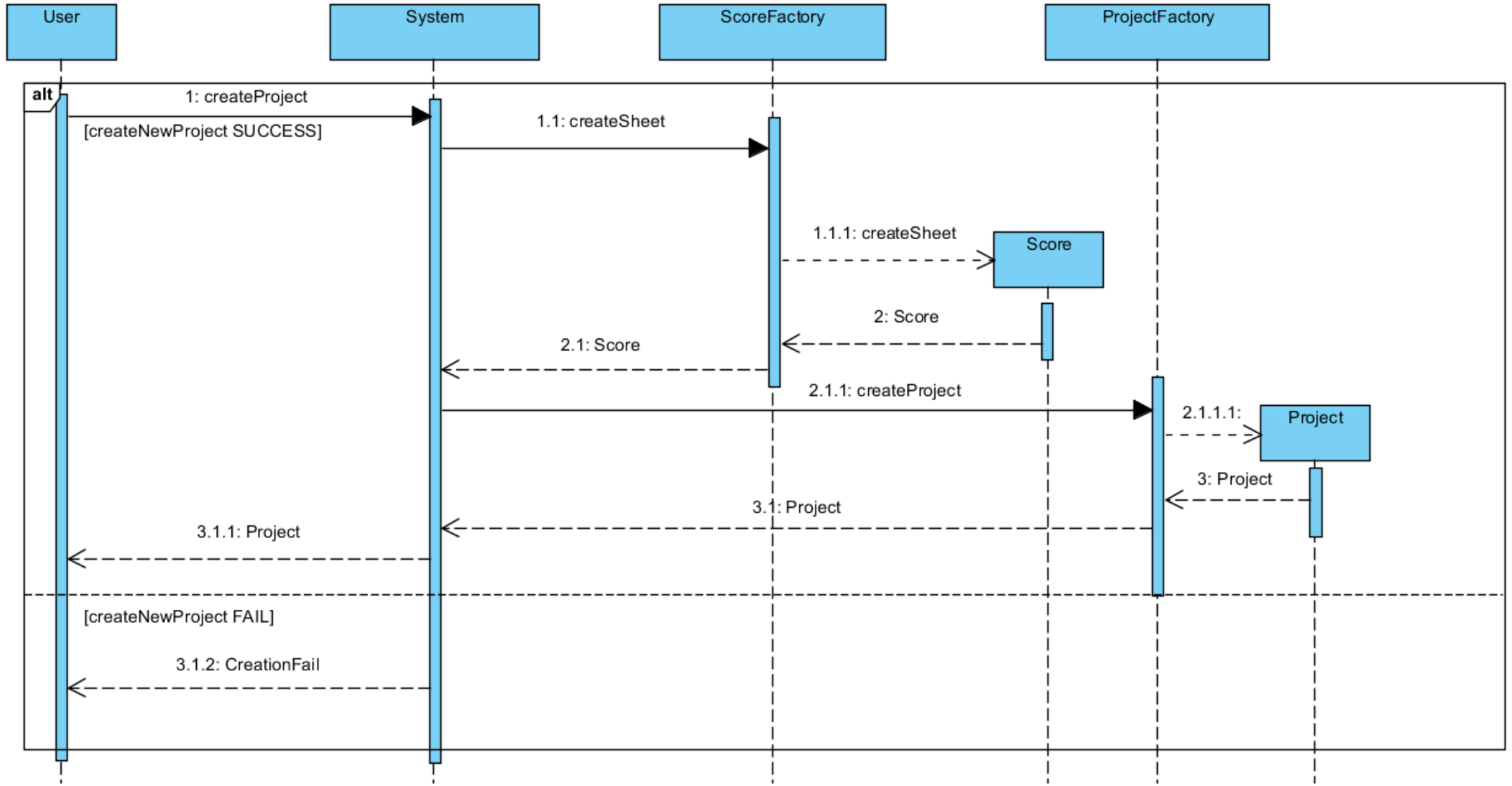


6.1.2.4. Sélectionner l'emplacement de début de lecture

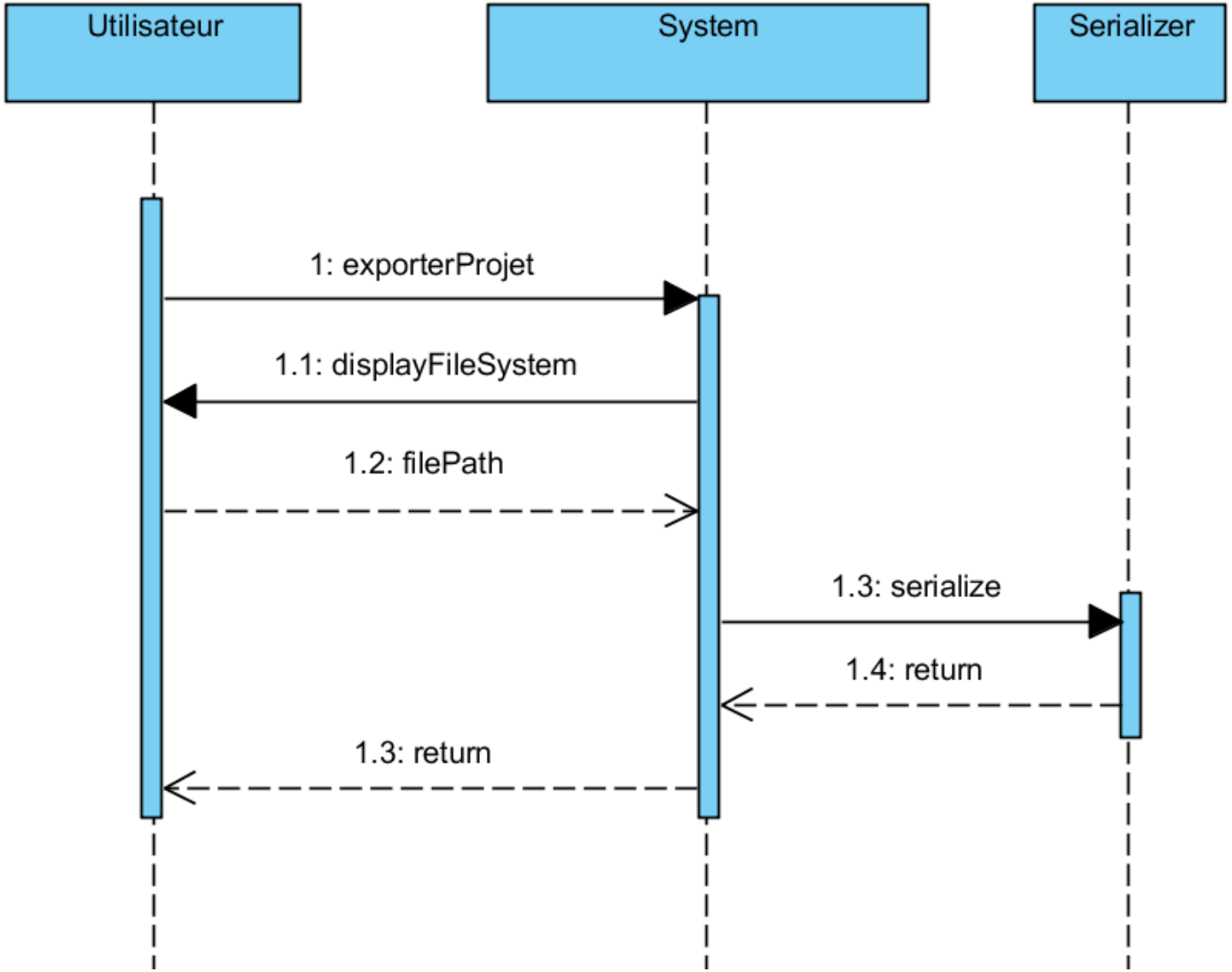


6.1.3. Gérer un projet

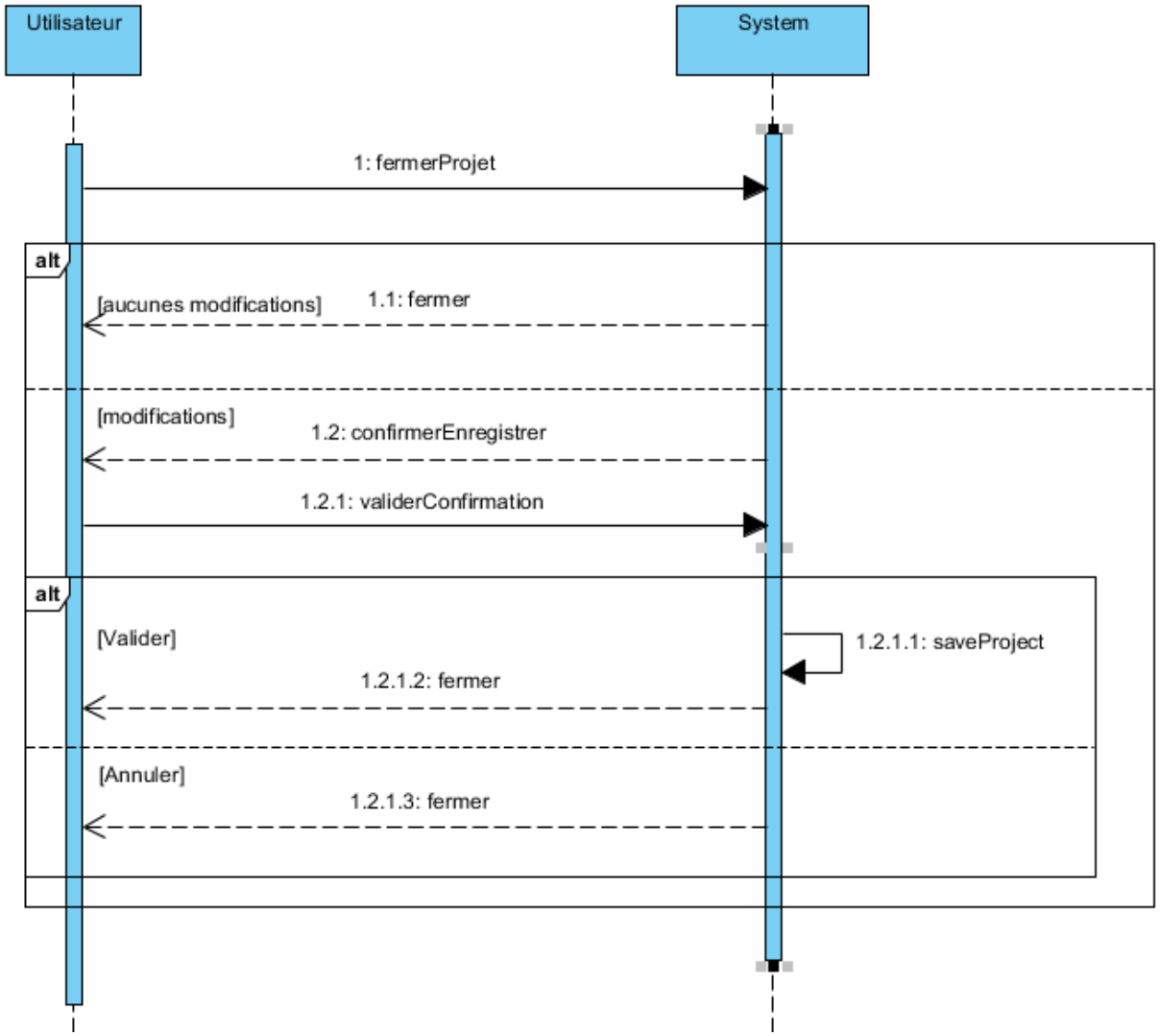
6.1.3.1. Créer un projet



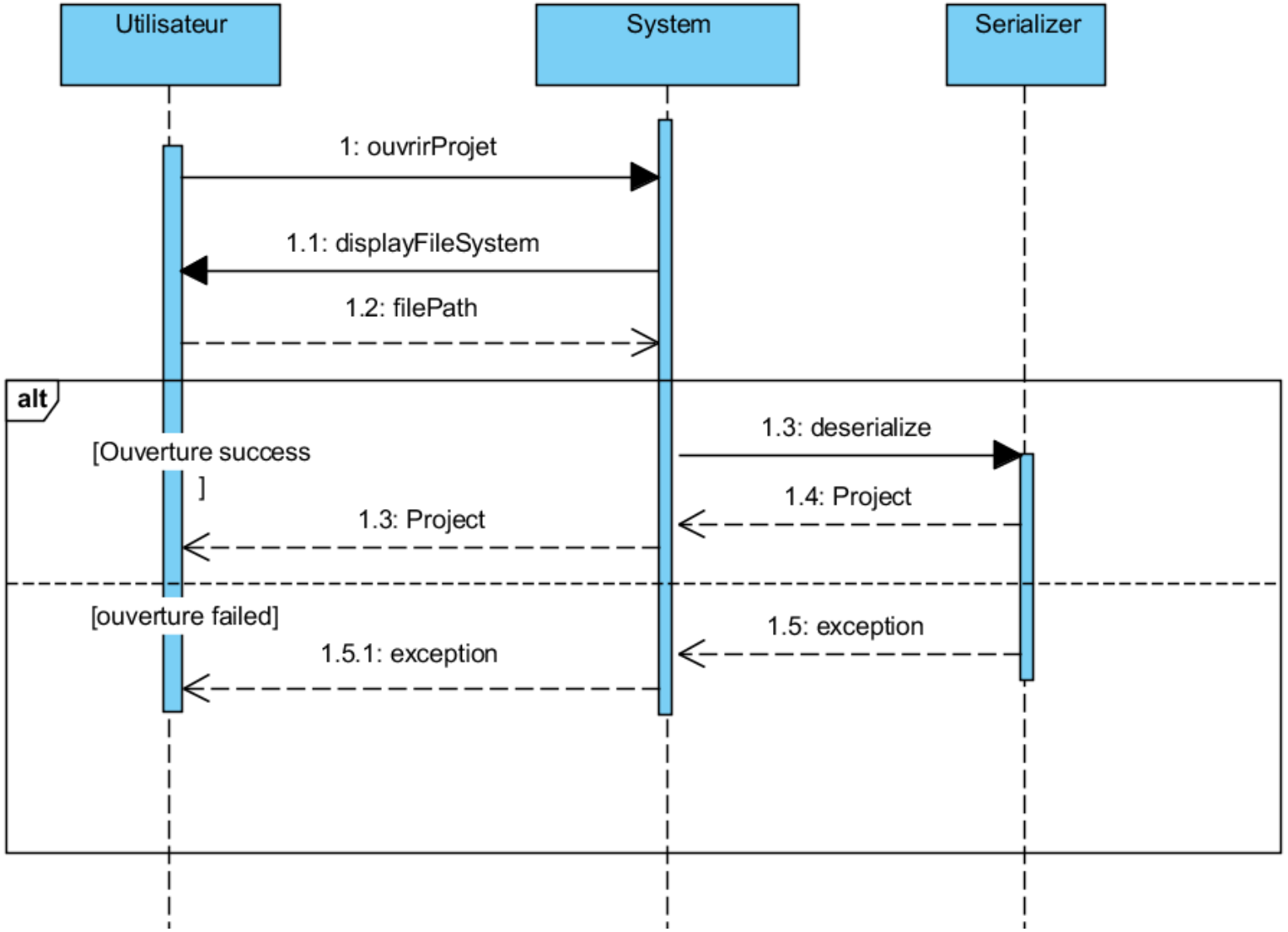
6.1.3.2. Exporter un projet



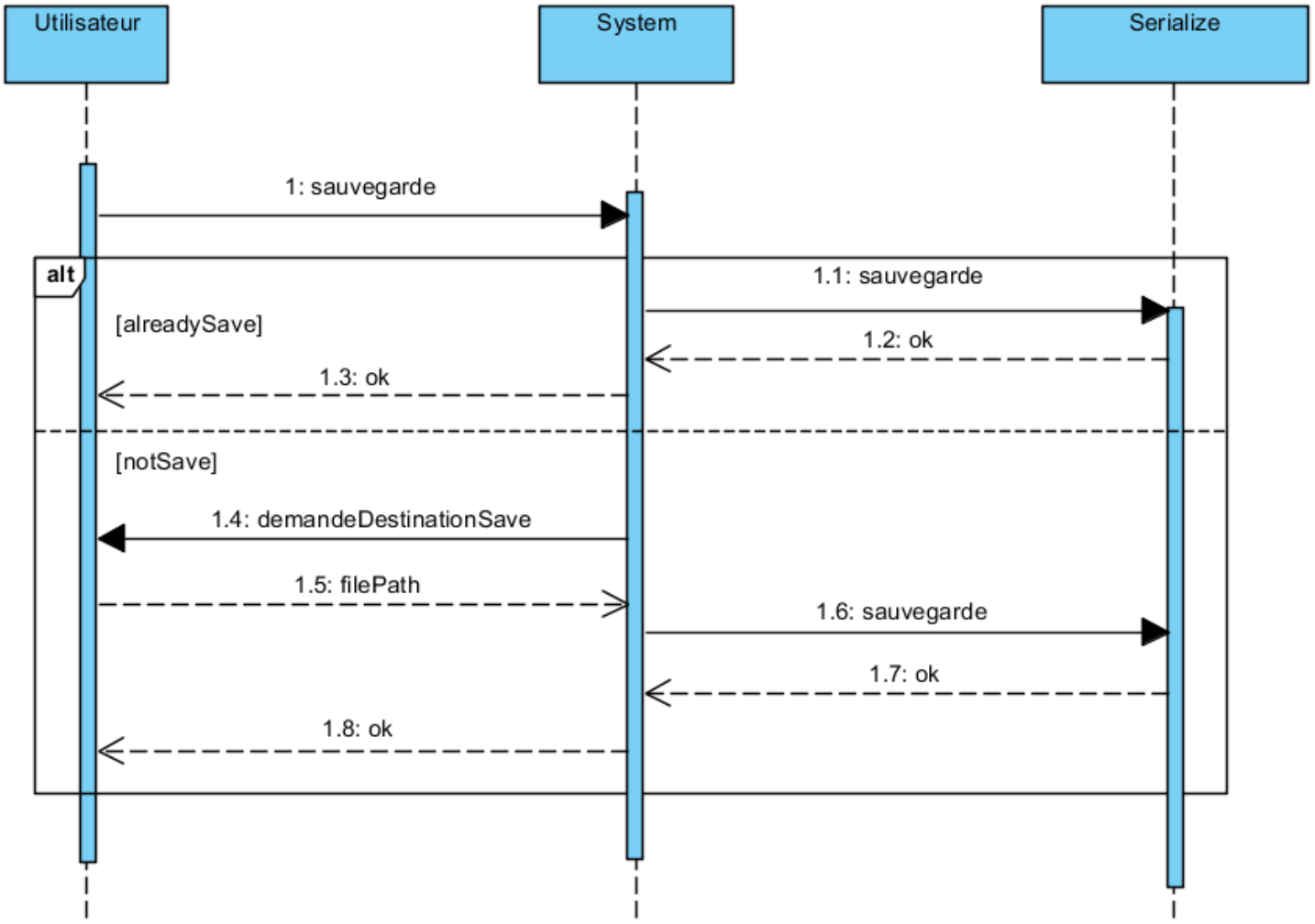
6.1.3.3. Fermer un projet



6.1.3.4. Ouvrir un projet



6.1.3.5. Sauvegarder un projet



6.1.3.6. Sélectionner un projet

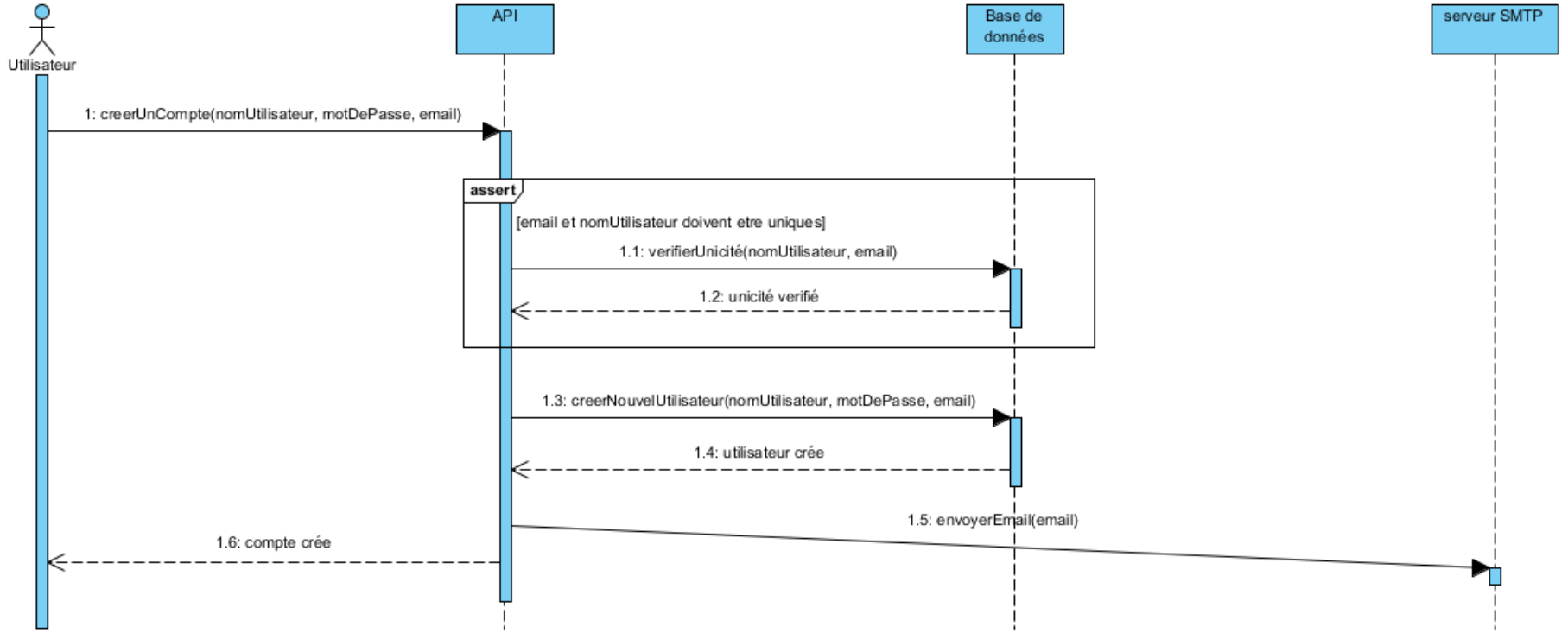
6.2. Le site internet

6.2.1. Création de compte

6.2.1.1. Description

La création d'un compte est soumise à plusieurs contraintes d'unicité. En effet, le nom d'utilisateur ainsi que l'adresse email ne doivent appartenir à aucun autre utilisateur. Si tel est le cas, un nouvel utilisateur est créé et un email de confirmation d'adresse mail est envoyé.

6.2.1.2. Vue dynamique

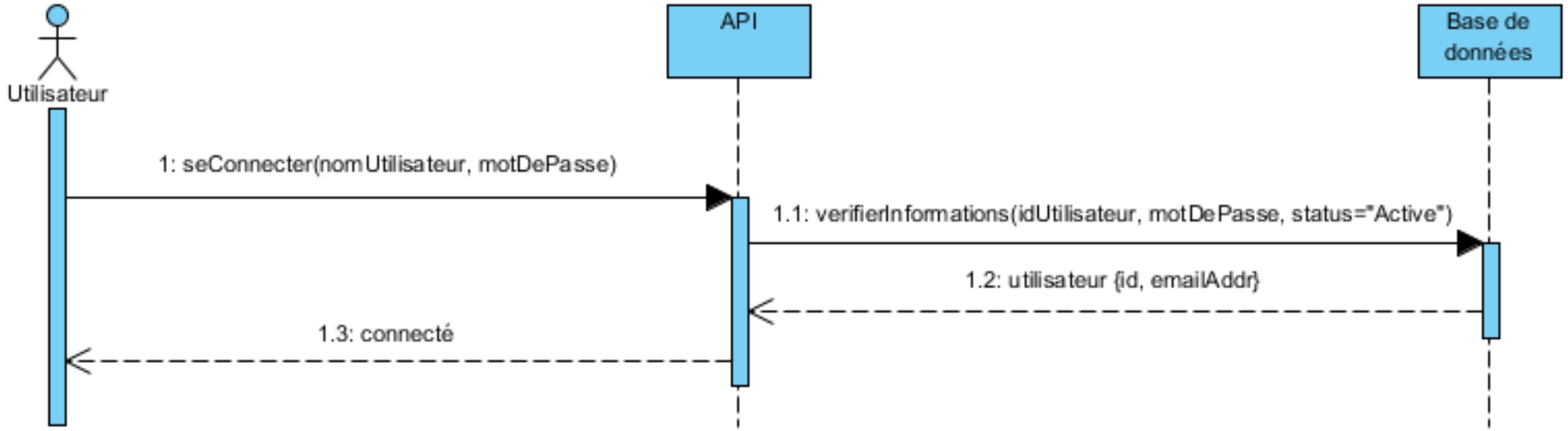


6.2.2. Connexion

6.2.2.1. Description

La connexion d'utilisateur se fait grâce au nom d'utilisateur et au mot de passe. Le compte doit être actif – c'est-à-dire qu'il ne doit pas avoir été fermé précédemment – pour pouvoir se connecter avec.

6.2.2.2. Vue dynamique

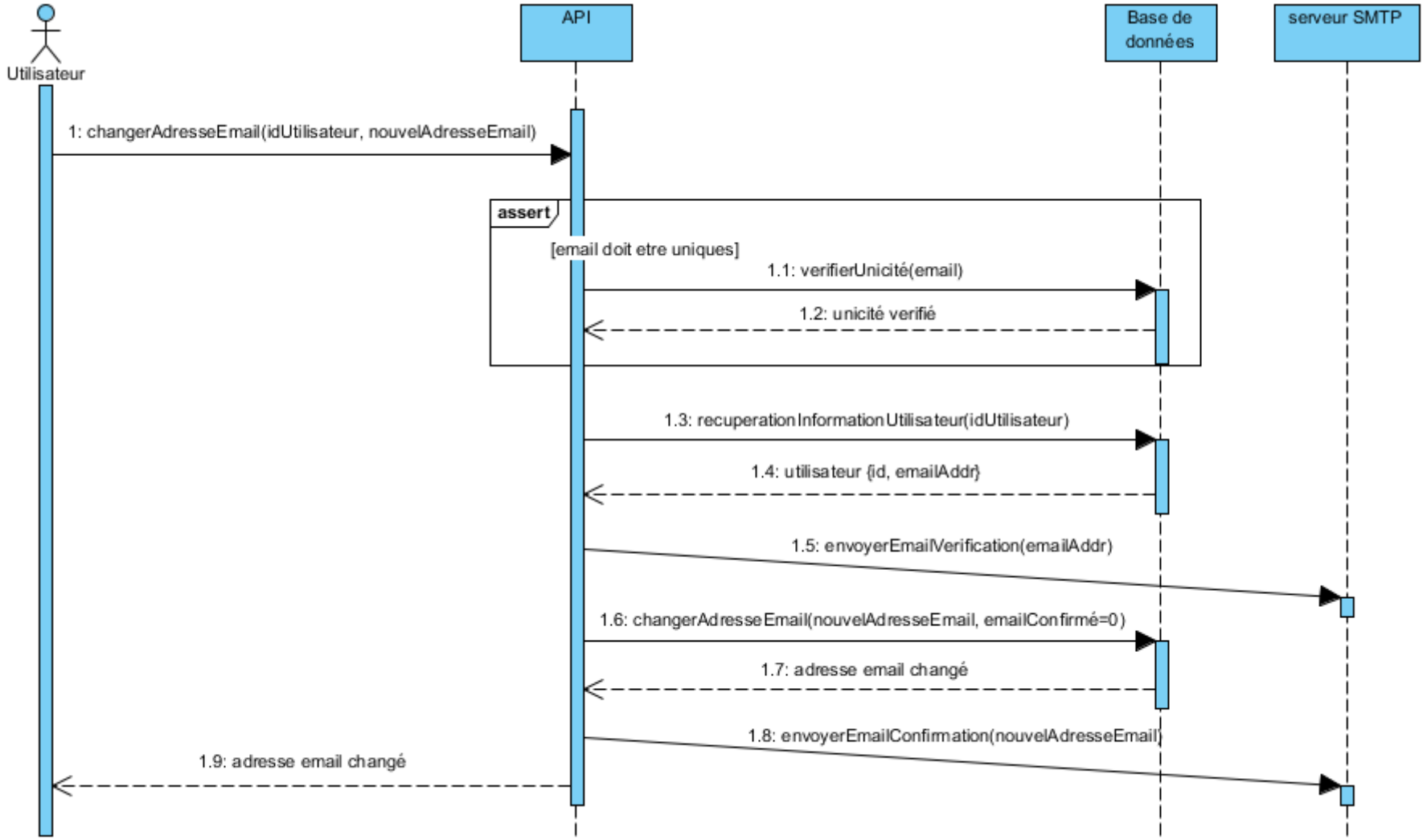


6.2.3. Changement d'adresse email

6.2.3.1. Description

Lors d'un changement d'adresse email, la nouvelle adresse ne doit pas déjà être attribué à un compte existant. SI elle est unique, alors un mail de vérification est envoyé à l'ancienne adresse email et un mail de confirmation à la nouvelle. L'utilisateur devra donc validé sa nouvelle adresse email.

6.2.3.2. Vue dynamique

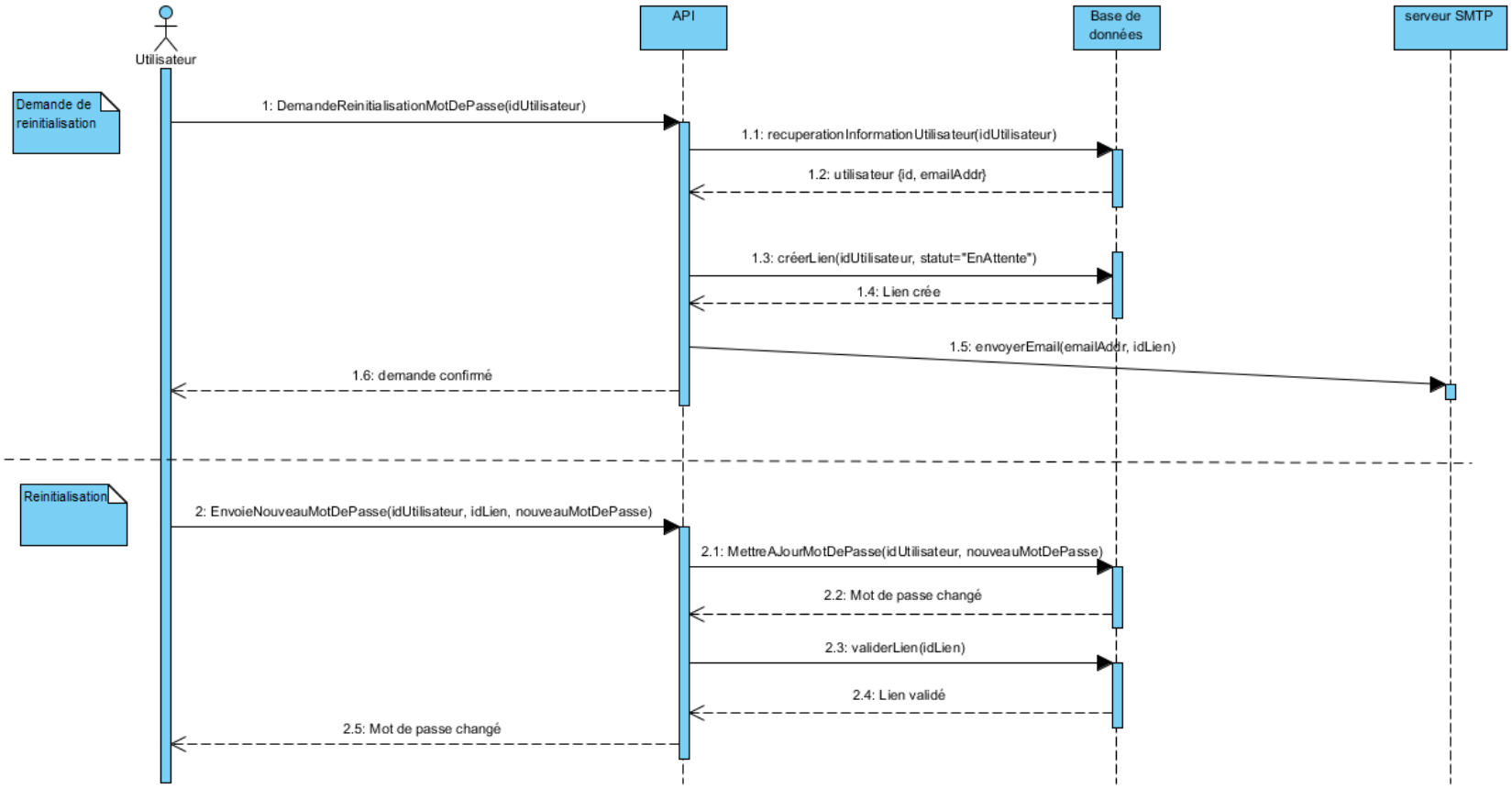


6.2.4. Réinitialisation de mot de passe

6.2.4.1. Description

La réinitialisation du mot de passe est un processus appliqué lorsqu'un utilisateur ne se souvient plus de son mot de passe. Ce processus se fait en deux temps. Tout d'abord, l'utilisateur fait une demande de réinitialisation. Un lien lui permettant de réinitialiser son mot de passe lui sera alors envoyé par mail. Une fois arrivé sur le lien, l'utilisateur pourra soumettre un nouveau mot de passe.

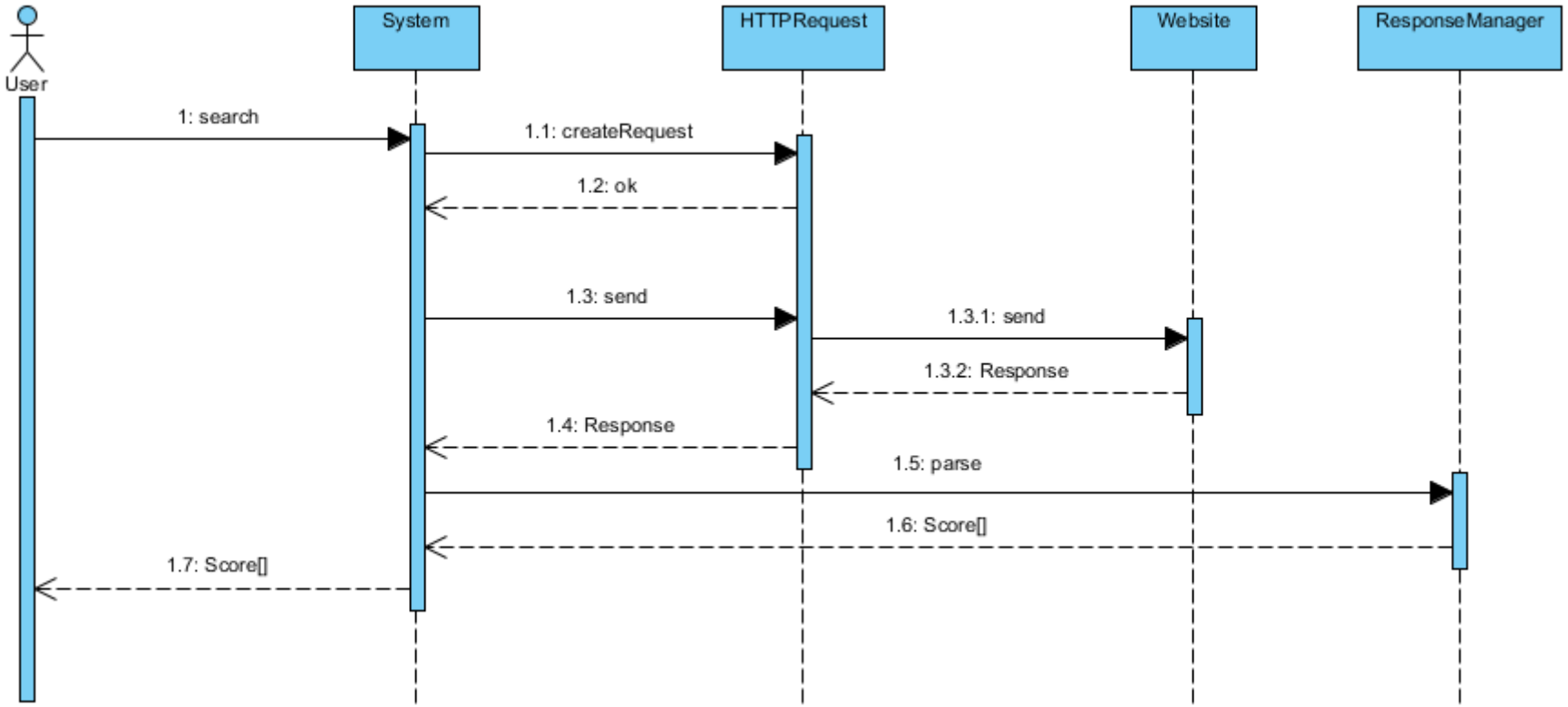
6.2.4.2. Vue dynamique



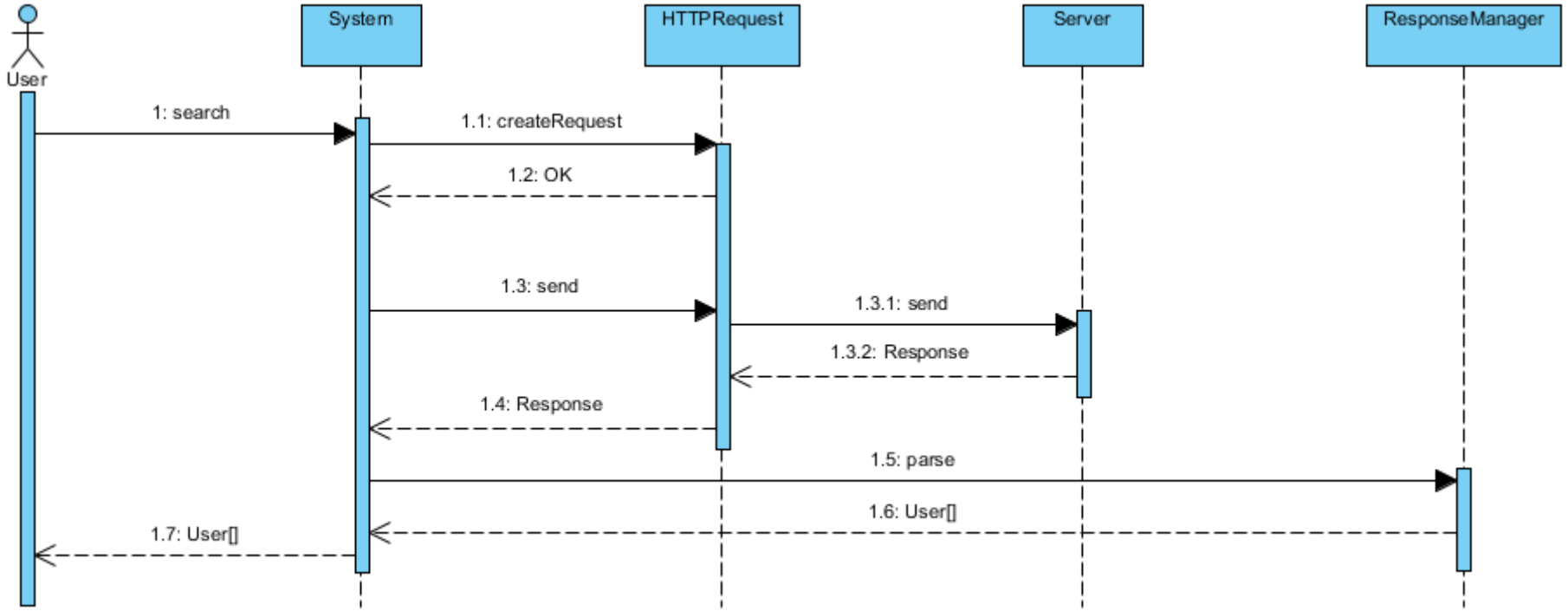
6.3. Les applications mobiles

6.3.1. Accéder à la communauté

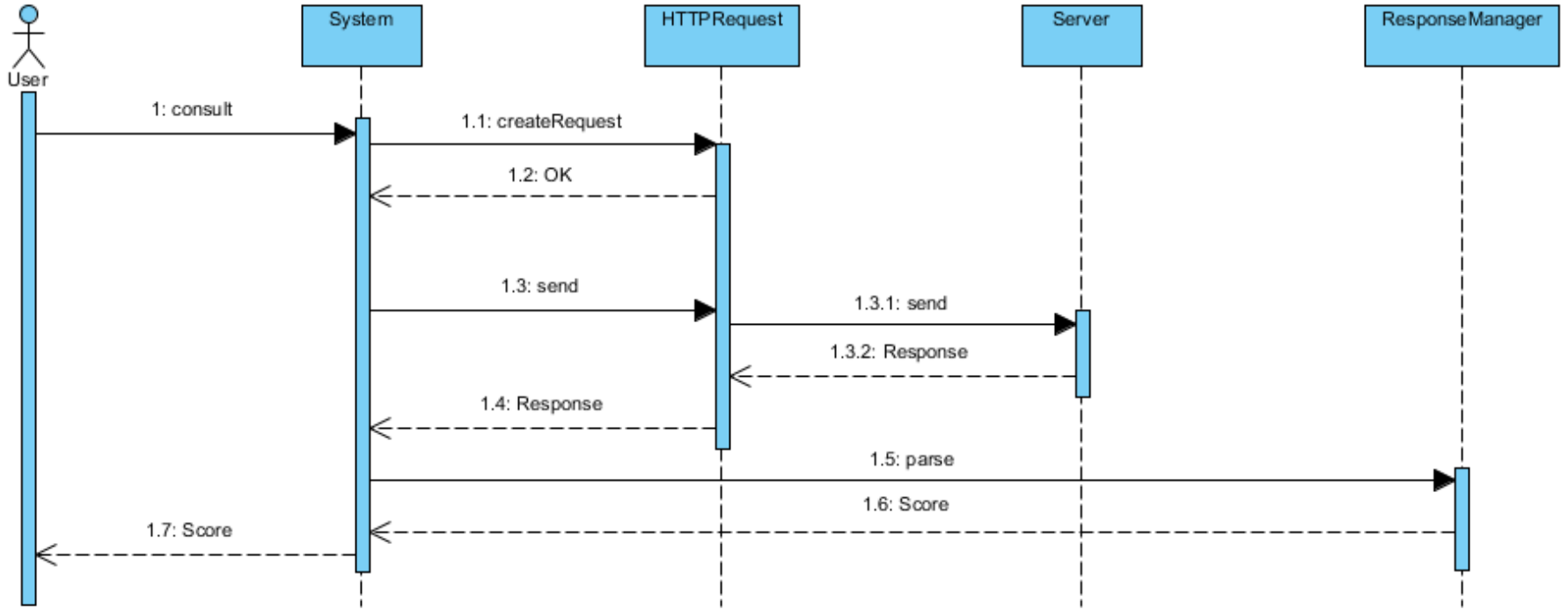
6.3.1.1. Rechercher une partition



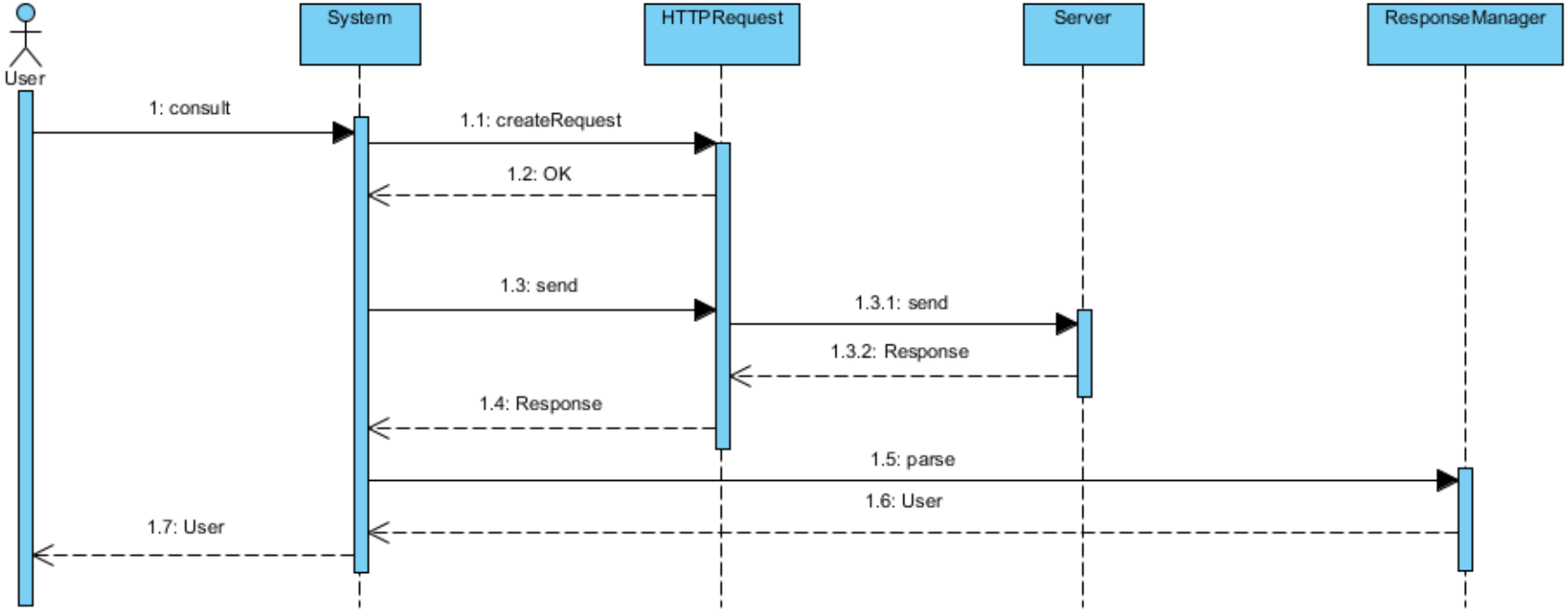
6.3.1.2. Rechercher un utilisateur



6.3.1.3. Consulter une partition

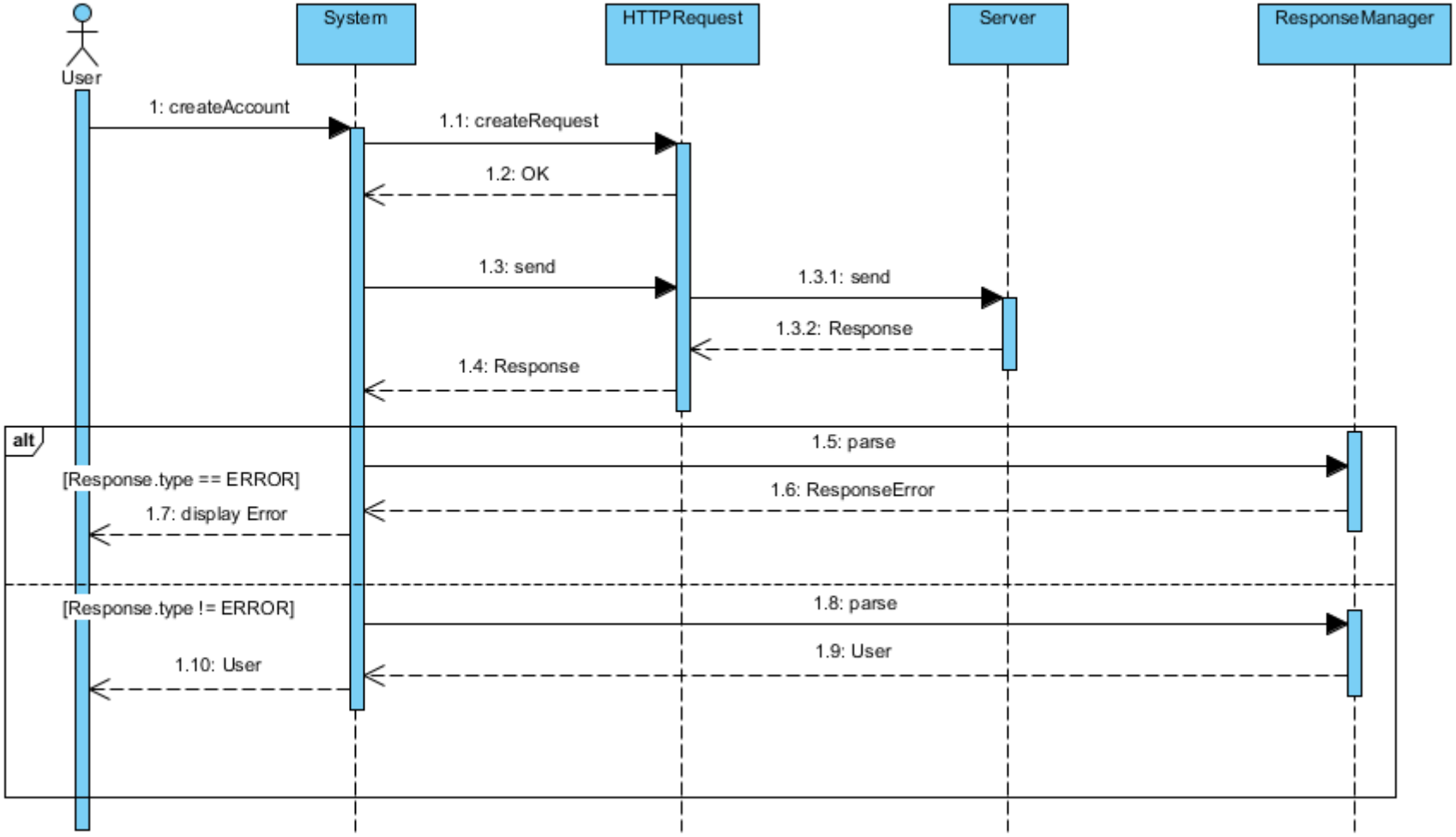


6.3.1.4. Consulter le profil d'un utilisateur

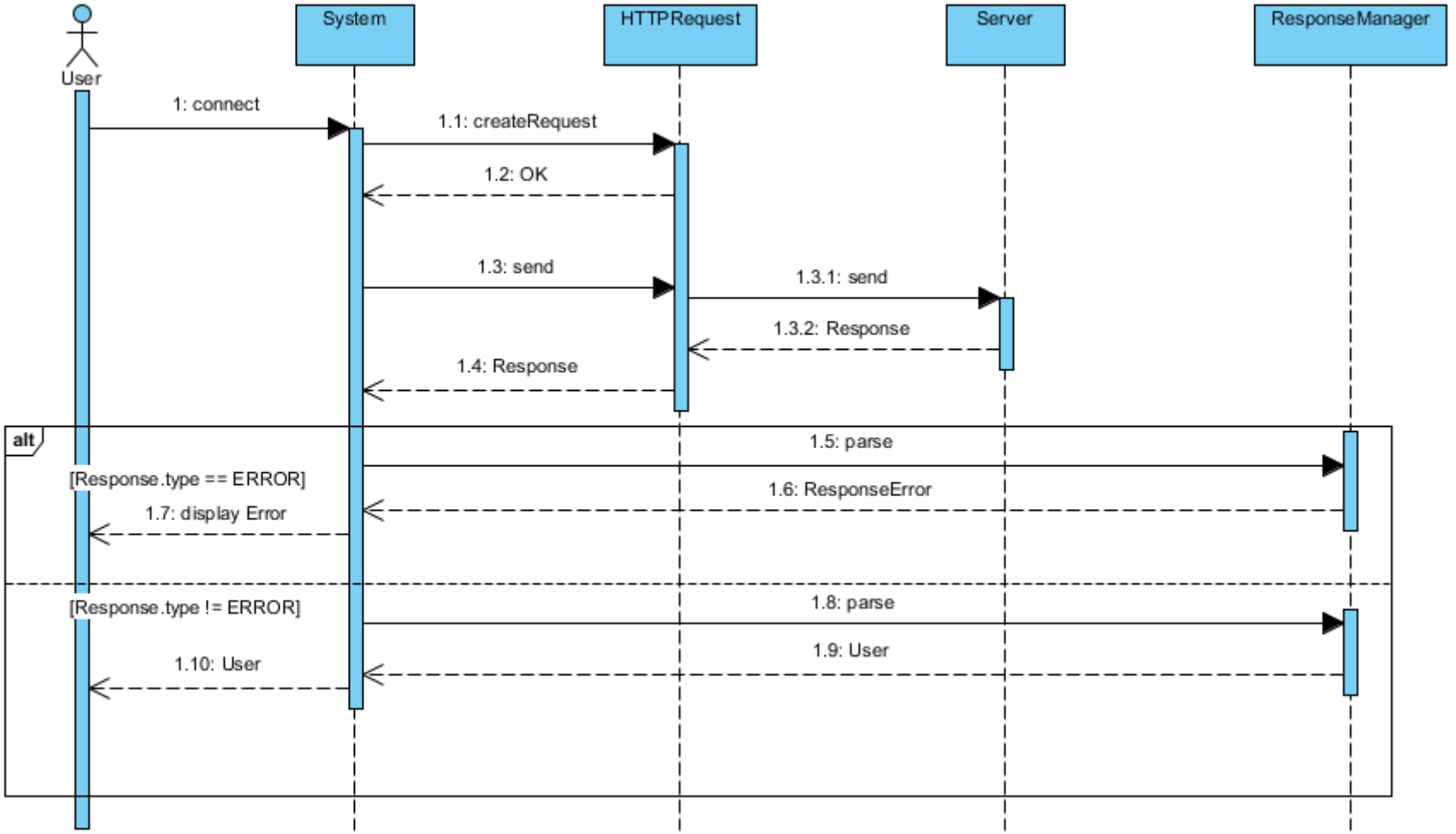


6.3.2. Gérer son compte utilisateur

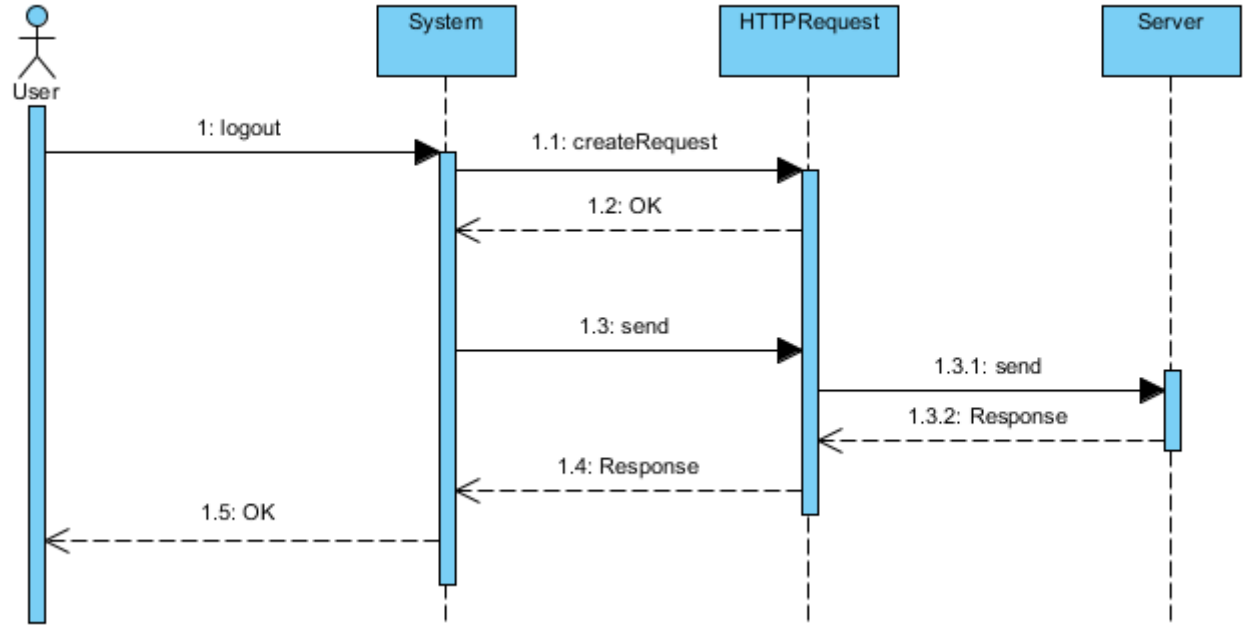
6.3.2.1. Créer un compte



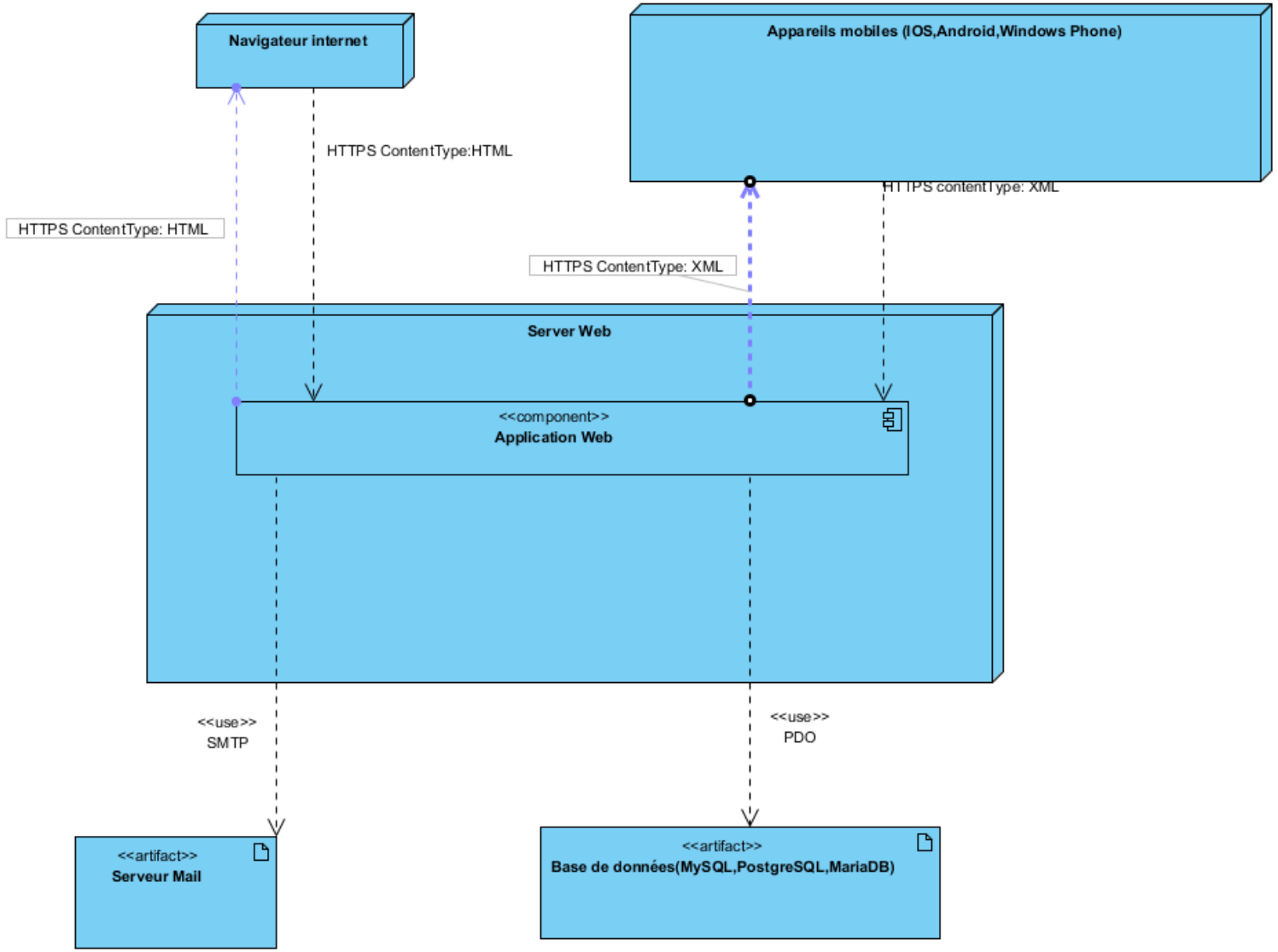
6.3.2.2. Se connecter



6.3.2.3. Se déconnecter



7. Vue déploiement


Navigateur internet
Appareils mobiles (IOS,Android,Windows Phone)
HTTPS ContentType:HTML
HTTPS contentType: XML
HTTPS ContentType: HTML
HTTPS ContentType: XML
Server Web
<<component>>
Application Web
<<use>>
SMTP
<<use>>
PDO
<<artifact>>
Serveur Mail
<<artifact>>
Base de données(MySQL,PostgreSQL,MariaDB)

Les échanges entre le serveur web, l'application mobile et les utilisateurs connectés depuis un navigateur web se fera par HTTPS.

Pour la mise en place du protocole HTTPS il nous faudra un certificat X509 certifié par une autorité de certification que l'on mettra en place dans la configuration du serveur Web.

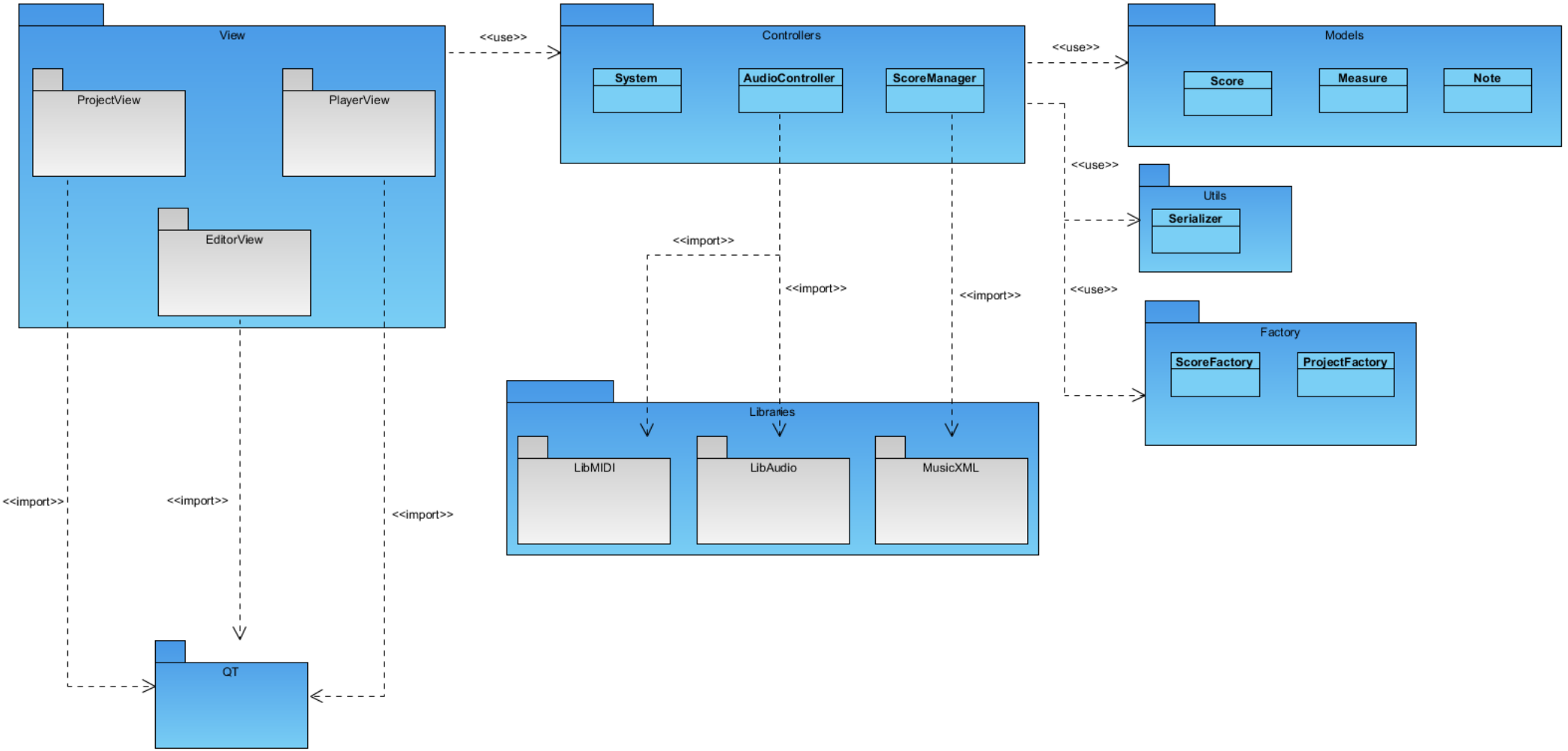
Les échanges entre l'application et le serveur de mail se feront par SMTP. Il faudra donc configurer un compte mail sur le serveur SMTP qui sera utilisé pour envoyer des mails par l'application.

Toutes la partie configuration sera gérée dans les fichiers de configurations de Symfony qui regroupent tous les paramètres nécessaires pour la mise en place des connexions entre l'application, le service de mail et la base de données.

8. Implémentation

8.1. Logiciel

8.1.1. Vue globale



8.2. Site Internet

8.2.1. Description

L'API REST est un Web Service. C'est l'interface entre la vue et la base de données. Elle est appelée en utilisant des requêtes HTTPS.

Les requêtes seront de la forme https://[adresse_ip]/musicsheetwriter_api/[URI] avec l'URI étant le chemin de la ressource. Pour le mobile, l'API sera accessible en utilisant l'URL https://[adresse_ip]/m/musicsheetwriter_api/[URI].

Les entrées/sortie de l'API seront en anglais.

Toutes les erreurs seront de la forme :

```
<error>
    <short_code>[short_code]</short_code>
    <message>[message]</message>
</error>
```

8.2.2. Vue en ressource

Ressource	*Méthode HTTP*	*Description*
/login	*POST*	Connecte un utilisateur à partir de ses identifiants de connexion.
	DELETE	Déconnecte un utilisateur
/users	*GET*	Récupère la liste des utilisateurs. Des filtres peuvent être ajoutés.
	POST	Enregistre un nouvel utilisateur
/users/[id]	*GET*	Récupère toutes les informations d'un utilisateur
	DELETE	Ferme un compte
/users/[id]/personal_data	*GET*	Récupère les informations personnelles d'un utilisateur
	PUT	Change les informations personnelles d'un utilisateur
/users/[id]/photograph	*GET*	Récupère la photo de profil d'un utilisateur
	PUT	Change la photo de profil d'un utilisateur. Est appelé après l'upload de la nouvelle photo
/users/[id]/credentials/username	*GET*	Récupère le pseudonyme d'un utilisateur
/users/[id]/credentials/email	*GET*	Récupère l'adresse email d'un utilisateur
	PUT	Change l'adresse email d'un utilisateur
/users/[id]/credentials/password	*POST*	Change le mot de passe d'un utilisateur
	PUT	Change le mot de passe d'un utilisateur en utilisant son ancien mot de passe
/users/[id]/credentials/password_link	*POST*	Crée un nouveau de lien de réinitialisation de mot de passe
	PUT	Valide un lien de réinitialisation de mot de passe (le mot de passe a été changé)
	DELETE	Invalide un lien de réinitialisation de mot de passe (le lien a expiré)
/users/[id]/subscriptions	*GET*	Récupère la liste des abonnements d'un utilisateur. Des filtres peuvent être ajoutés
	POST	Ajoute un nouvel abonnement à un utilisateur
/users/[id]/subscriptions/[id]	*DELETE*	Retire un abonnement d'un utilisateur
/users/[id]/subscribers	*GET*	Récupère la liste des abonnés d'un utilisateur. Des filtres peuvent être ajoutés.
/users/[id]/scores/own	*GET*	Récupère la liste des partitions d'un utilisateur
	POST	Ajoute une nouvelle partition à un utilisateur. Est appelé après l'upload d'un fichier de partition
/users/[id]/scores/own/[id]	*DELETE*	Retire une partition d'un utilisateur.
/users/[id]/scores/favourites	*GET*	Récupère la liste des partitions favorites d'un utilisateur
	POST	Ajoute une nouvelle partition favorite à un utilisateur
/users/[id]/scores/favourites/[id]	*DELETE*	Retire une partition favorite d'un utilisateur
/users/[id]/purchases	*GET*	Récupère la liste des achats d'un utilisateur
/scores	*GET*	Récupère la liste des partitions. Des filtres peuvent être ajoutés
/scores/[id]	*GET*	Récupère une partition
	DELETE	Supprime une partition
/purchases	*GET*	Récupère la liste des achats. Des filtres peuvent être ajoutés.
	POST	Crée un nouvel achat
/purchases/[id]	*GET*	Récupère les informations d'un achat.
	PUT	Met à jour les informations d'un achat

Ressource	*Méthode HTTP*	*Description*
	DELETE	*Désactive un achat.*
/contact	*POST*	*Envoie un email à l'équipe MSW.*

8.2.3. Interface

Requête			*Reponse*	
Ressource	*Méthode HTTP*	*Corps*	*Code HTTP*	*Corps*
/login	*POST*	*<credentials> <username>[username]</username> <password>[password]</password> </credentials>*	200	*<user id=[id]> <username>[username]</username> <last_activity_date>[last_activity_date]</last_activity_date> </user>*
			401	*<error> <shortcode>LOG-BADCREDENTIALS</shortcode> <message> Credentials not valid. </message> </error>*
			403	*<error> <shortcode>LOG-CLOSEDACC</shortcode> <message> Account closed </message> </error>*
	DELETE	*<credentials> <username>[username]</username> </credentials>*	200	EMPTY
/users *2 Des filtres peuvent être ajoutés en tant que paramètre GET - uname - fname - uname - email	*GET*	EMPTY	200	*<users count=[count]> <user index=[index] id=[id]> <username>[username]</username> <personal_data> <first_name>[first_name]</first_name> <surname>[surname]</surname] <email>[email]</email> <photograph> <url>[url]</url> </photograph> <personal_data> <message>[message]</message>*

Requête			*Reponse*	
Ressource	*Méthode HTTP*	*Corps*	*Code HTTP*	*Corps*
				<status>[status]</status> *<last_activity_date>[last_activity_date]</last_activity_date>* *<creation_date>[creation_date]</creation_date>* *</user>* *…* *</users>* The 'index' is the position (beginning by 0) of the tag among the list.
	POST	*<user>* *<username>[username]</username>* *<password>[password]</password* *<personal_data>* *<first_name>[first_name]</first_name>* *<surname>[surname]</surname]* *<email>[email]</email>* *</personal_data>* *</user>*	200	*<user id=[id]>* *<username>[username]</username>* *<personal_data>* *<first_name>[first_name]</first_name>* *<surname>[surname]</surname]* *<email>[email]</email>* *<photograph>* *<url>[url]</url>* *</photograph>* *</personal_data>* *<message>[message]</message>* *<status>[status]</status>* *<last_activity_date>[last_activity_date]</last_activity_date>* *<creation_date>[creation_date]</creation_date>* *</user>*
			400	*<error>* *<shortcode>USR-BADFIELD</shortcode>* *<message>* *[field] not correct* *</message>* *</error>* This error can be returned if a 'field' has a bad format.
			409	*<error>* *<shortcode>REG-NAMEALRUSED</shortcode>* *<message>* *Username already used* *</message>* *</error>*

Requête			*Reponse*	
Ressource	*Méthode HTTP*	*Corps*	*Code HTTP*	*Corps*
			409	*<error> <shortcode>REG-MAILALRUSED</shortcode> <message> Email address already used </message> </error>*
/users/[id] *1	*GET*	EMPTY	200	*<user id=[id]> <username>[username]</username> <personal_data> <first_name>[first_name]</first_name> <surname>[surname]</surname] <email>[email]</email> <photograph>[photograph]</photograph> <personal_data> <message>[message]</message> <status>[status></status> <last_activity_date>[last_activity_date]</last_activity_date> <creation_date>[creation_date]</creation_date> </user>*
	DELETE	EMPTY	200	*<user id=[id]> <status>[status></status> <closure_date></closure_date> </user>*
			403	*<error> <shortcode>CLO-USERALRCLO</shortcode> <message> User account already closed </message> </error>*
/users/[id]/personal_data *1	*GET*	EMPTY	200	*<personal_data> <first_name>[first_name]</first_name> <surname>[surname]</surname] <email>[email]</email> <personal_data>*
	PUT	*<personal_data> <first_name>[first_name]</first_name>*	200	*<personal_data> <first_name>[first_name]</first_name>*

Requête			*Reponse*	
Ressource	*Méthode HTTP*	*Corps*	*Code HTTP*	*Corps*
		<surname>[surname]</surname> <email>[email]</email> <personal_data>		*<surname>[surname]</surname> <email>[email]</email> <personal_data>*
			400	*<error> <shortcode>USR-BADFIELD</shortcode> <message> [field] field not correct </message> </error>* Cette erreur peut être retournée si un champ a un mauvais format.
/users/[id]/photograph [*1]	*GET*	EMPTY	200	*<photograph> <url>[url]</url> </photograph>*
	PUT	*<photograph> <url>[url]</url> </photograph>*	204	*<photograph> <url>[url]</url> </photograph>*
/users/[id]/credentials/username [*1]	*GET*	EMPTY	200	*<username> [username] </username>*
/users/[id]/credentials/email [*1]	*GET*	EMPTY	200	*<email> [email] </email>*
	PUT	*<email> [email] </email>*	200	*<email> [email] </email>*
			400	*<error> <shortcode>USR-BADFIELD</shortcode> <message> email field not correct </message> </error>*
/users/[id]/credentials/password [*1]	*POST*	*<password> <new_password> [new_password] </new_password> </password>*	204	EMPTY
			400	*<error> <shortcode>USR-BADFIELD</shortcode> <message> password field not correct*

Requête			*Reponse*	
Ressource	*Méthode HTTP*	*Corps*	*Code HTTP*	*Corps*
				</message> *</error>*
	PUT	*<password>* *<new_password>* *[new_password]* *</new_password>* *<current_password>* *[current_password]* *</current_password>* *</password>*	204	EMPTY
			400	*<error>* *<shortcode>USR-BADFIELD</shortcode>* *<message>* *password field not correct* *</message>* *</error>*
			403	*<error>* *<shortcode>USR-WRONGPASS</shortcode>* *<message>* *Current password not correct* *</message>* *</error>*
/users/[id]/credentials/password_link *1	*POST*	EMPTY	200	*<password_link id=[id]>* *<status>[status]</status>* *</password_link>* Le status devrait être « Pending » (en attente)
	PUT	*<password_link id=[id] />*	200	*<password_link id=[id]>* *<status>[status]</status>* *</password_link>* Le status devrait être « OK »
			403	*<error>* *<shortcode>PAS-LINKALRUSED</shortcode>* *<message>* *Password link already used* *</message>* *</error>*
			404	*<error>* *<shortcode>PAS-BADLINK</shortcode>* *<message>* *Password link not correct* *</message>* *</error>*

Requête			*Reponse*	
Ressource	*Méthode HTTP*	*Corps*	*Code HTTP*	*Corps*
	DELETE	*<password_link id=[id] />*	200	*<password_link id=[id]>* *<status>[status]</status>* *</password_link>* Le status devrait être « NOK »
			403	*<error>* *<shortcode>PAS-LINKALRUSED</shortcode>* *<message>* *Password link already used* *</message>* *</error>*
			404	*<error>* *<shortcode>PAS-BADLINK</shortcode>* *<message>* *Password link not correct* *</message>* *</error>*
/users/[id]/subscriptions [*1]	*GET*	EMPTY	200	*<subscriptions count=[count]>* *<user index=[index] id=[id]>* *<username>[username]</username>* *</user>* *…* *<subscriptions>* The 'index' is the position (beginning by 0) of the tag among the list.
	POST	*<user id=[id] />*	200	*<subscriptions count=[count]>* *<user index=[index] id=[id]>* *<username>[username]</username>* *</user>* *…* *<subscriptions>*
			404	*<error>* *<shortcode>SUB-BADUSERID</shortcode>* *<message>* *User ID not correct* *</message>* *</error>*

Requête			*Reponse*	
Ressource	*Méthode HTTP*	*Corps*	*Code HTTP*	*Corps*
/users/[id]/subscriptions/[id] *1	*DELETE*	EMPTY	200	*<user id=[id] />*
/users/[id]/subscribers *1	*GET*	EMPTY	200	*<subscriptions count=[count]>* *<user index=[index] id=[id]>* *<username>[username]</username>* *</user>* *…* *<subscriptions>* The 'index' is the position (beginning by 0) of the tag among the list.
/users/[id]/scores/own *1	*GET*	EMPTY	200	*<scores count=[count]>* *<score index=[index] id=[id]>* *<url>[url]</url>* *</score>* *…* *</scores>*
	POST	*<score>* *<url>[url]</url>* *</score>*	200	*<scores count=[count]>* *<score index=[index] id=[id]>* *<url>[url]</url>* *<user id=[id] />* *</score>* *…* *</scores>*
/users/[id]/scores/own/[id] *1	*DELETE*	EMPTY		*<score>* *<url>[url]</url>* *</score>*
/users/[id]/scores/favourites *1	*GET*	EMPTY	200	*<scores count=[count]>* *<score index=[index] id=[id]>* *<url>[url]</url>* *<user id=[id] />* *</score>* *…* *</scores>*
	POST	*<score>* *<url>[url]</url>* *</score>*	200	*<score id=[id]>* *<url>[url]</url>* *<user id=[id] />* *</score>*
/users/[id]/scores/favourites/[id] *1	*DELETE*	EMPTY		*<score>*

Requête			*Reponse*	
Ressource	*Méthode HTTP*	*Corps*	*Code HTTP*	*Corps*
				<url>[url]</url> *</score>*
/users/[id]/purchases [*1]	*GET*	EMPTY	200	*<purchases count=[count]>* *<purchase index=[index] id={id]>* *<product_key>[product_key]<product_key>* *<date>[date]</date>* *<user id=[id] />* *</purchase>* *</purchases>*
/scores	*GET*	EMPTY	200	*<scores count=[count]>* *<score index=[index] id=[id]>* *<url>[url]</url>* *<user id=[id] />* *</score>* *…* *</scores>*
/scores/[id]	*GET*	EMPTY	200	*<score id=[id]>* *<url>[url]</url>* *<user id=[id] />* *</score>*
	DELETE	*<score id=[id]>* *<url>[url]</url>* *<user id=[id] />* *</score>*	200	*<score id=[id]>* *<url>[url]</url>* *<user id=[id] />* *</score>*
			404	*<error>* *<shortcode>SCR-BADUSERID</shortcode>* *<message>* *User ID not correct* *</message>* *</error>*
/purchases	*GET*	EMPTY	200	*<purchases count=[count]>* *<purchase index=[index] id={id]>* *<product_key>[product_key]<product_key>* *<date>[date]</date>* *<user id=[id] />* *<status>[status]</status>* *</purchase>*

Requête			*Reponse*	
Ressource	*Méthode HTTP*	*Corps*	*Code HTTP*	*Corps*
				</purchases>
	POST	*<purchase index=[index] id={id]> \<product_key>[product_key]<product_key> \<date>[date]</date> \<user id=[id] /> </purchase>*	200	*<purchases count=[count]> \<purchase index=[index] id={id]> \<product_key>[product_key]<product_key> \<date>[date]</date> \<user id=[id] /> </purchase> </purchases>*
			404	*<error> \<shortcode>PUR-BADUSERID</shortcode> \<message> User ID not correct </message> </error>*
/purchases/[id]	*GET*	EMPTY	200	*<purchase id={id]> \<product_key>[product_key]<product_key> \<date>[date]</date> \<user id=[id] /> \<status>[status]</status> </purchase>*
	PUT	*<purchase> \<product_key>[product_key]<product_key> \<user id=[id] /> \<status>[status]</status> </purchase>*	200	*<purchase id={id]> \<product_key>[product_key]<product_key> \<date>[date]</date> \<user id=[id] /> \<status>[status]</status> </purchase>*
			404	*<error> \<shortcode>SUB-BADUSERID</shortcode> \<message> User ID not correct </message> </error>*
	DELETE	*<purchase id=[id] />*	200	*<purchase id={id]> \<product_key>[product_key]<product_key> \<date>[date]</date> \<user id=[id] /> \<status>[status]</status>*

Requête			*Reponse*	
Ressource	*Méthode HTTP*	*Corps*	*Code HTTP*	*Corps*
				</purchase>
/contact	*POST*	*<contact>* *<user id=[id] />* *<subject>[subject]</subject>* *<message>[message]</message>* *</contact>*	200	EMPTY
			404	*<error>* *<shortcode>CON-BADUSERID</shortcode>* *<message>* *User ID not correct* *</message>* *</error>*

Note:

*1 l'URL */users/[id]* et */users/[id]/** sont équivalent à */users/[username]* et */users/[username]/**. Ainsi, un pseudonyme ne peut pas être un nombre.

*2 quand un filtre est appliqué, sa valeur est interprétée comme un englobeur. De plus, si plusieurs filtres sont appliqués, ils sont associatifs.
Exemples:

/users?uname=boss retournera tous les utilisateurs dont le pseudonyme commence par *boss* (*boss, boss31, bossy, etc.*).
/users?uname=boss&fname=fr retournera tous les utilisateurs dont le pseudonyme commence par *boss* et dont le prénom commence par *fr*.

8.2.4. Liste des erreurs

Short code	*Code HTTP*	*Message*	*Explication (optionnel)*
LOG-BADCREDENTIALS	*401*	*Credentials not valid.*	
LOG-CLOSEDACC	*403*	*Account closed*	
USR-BADFIELD	*400*	*[field] field not correct*	
USR-WRONGPASS	*403*	*Current password not correct*	
REG-NAMEALRUSED	*409*	*Username already used*	
REG-MAILALRUSED	*409*	*Email address already used*	
CLO-USERALRCLO	*403*	*User account already closed*	
PAS-LINKALRUSED	*403*	*Password link already used*	
PAS-BADLINK	*404*	*Password link not correct*	
SUB-BADUSERID	*404*	*User ID not correct*	
SCO-BADUSERID	*404*	*User ID not correct*	
PUR-BADUSERID	*404*	*User ID not correct*	
CON-BADUSERID	*404*	*User ID not correct*	

8.2.5. Erreurs génériques

Certaines erreurs peuvent être retournées par toutes les requêtes et ne sont donc pas propre à une seule.

Short code	*Code HTTP*	*Message*	*Explication (optionnel)*
GLO-MISSING	*400*	*The following required parameter are missing: [-param1][-param2]…*	*Seuls les paramètres manquants et obligatoires peuvent être retournés avec cette erreur*
GLO-UNAUTHORIZED	*403*	*You must be logged to perform this action*	*Cette erreur est retournée lorsque qu'un utilisateur (connecté ou non) tente d'effectuer une action sur le compte d'un autre utilisateur*
GLO-RESNFOUND	*404*	*The requested resource is not found*	
GLO-METHNALLOW	*405*	*The method used is not allowed on the resource*	
GLO-BADROOT	*415*	*The XML root name is not correct*	
GLO-BADXML	*415*	*The requested XML is not correct*	
GLO-DBSERERR	*502*	*The database server does not respond*	
GLO-DBSCHERR	*500*	*An SQL error occurred*	*Cette erreur peut être causée par une anomalie dans le schéma de la base de données. Elle ne devrait pas être retournée en production*
GLO-INTERNERR	*500*	*An unknown error occurred*	*Cette erreur ne devrait pas être retournée en production*

8.3. Application mobiles

9. Vue données