

Claroffline :

Conception et développement du bundle offline pour Claroline

Promoteur :
Marc LOBELLE

Lecteurs :
Chantal PONCIN
Stéphane KLEIN

Mémoire présenté en vue de l'obtention
du grade de master 60 crédits
en sciences informatiques

Par :
Pierre-Yves LÉGÉNA
Vincent VAN OUYTSEL

Louvain-la-Neuve
Année Académique 2013-2014



Abstract

Nowadays e-learning is taking a major part in education. Every place in the world doesn't have reliable connections, thus everybody doesn't enjoy the same access to these e-learning platforms. The purpose of this master thesis is to create and develop a tool for Claroline learning management system. The goal of this tool is to allow a user to work on the platform whenever he has or not got an Internet access. In practice, the user has to be able to launch the pages of the website when his computer is offline and synchronize when he has an Internet access. This synchronization has to be bidirectional, the resources produced offline have to be uploaded online and the resources newly available online have to be downloaded on his personal computer.

As Claroline is built on the Symfony's PHP web development framework, we've designed this tool as a Symfony's bundle. More precisely, our bundle is a plug-in designed to be easily integrable into the Claroline platform. Moreover, concerning the synchronization module, we had to think about the way to diffuse our program because the users have to install a copy of the Claroline on their own computer to run it offline.

The synchronization process that we have designed is divided in three main parts : the creation, the transfer and the loading. The first part, the creation, is responsible for assembling all the modifications that have been done on the platform since the last synchronization. The transfer is the part in charge of the communication between the personal computer and the distant server. The third part, the loading, implements the integration of modifications inside a platform.

Technically, the articulation of the different parts of the global process are implemented in the `SynchronisationManager`. All the modifications made on a platform are stored in a XML file called manifest created in the `CreationManager`. The manifest is stored with the resource file in a zip archive and given to the `TransferManager`. This archive is divided into multiple fragments and transferred to the server via HTTP POST requests. On the server, the archive is loaded in such way to resolve the potential conflicts. Then the server creates its own archive containing the update of the job that has been done online. Then this synchronization archive of the server is downloaded on the personal computer before being loaded on it. Finally, we update the synchronization time stored in the `UserSynchronized` entity.

To deploy the code on an existing Claroline platform, the plug-in has to be installed via Composer package manager. For the installation on the personal computer, the users must have a PHP server and a database management system. We provide an installation package containing Xampp, a web server available for the main operating systems, and the source code of Claroline with the integrated `OfflineBundle`. Our package allows to broadcast an empty version of Claroline with no data inside. No data means also that it has no user

register in the database, so we've created a view asking the credentials of the user on the first launch. With those information we are allowed to retrieve the profile of the user from the distant server.

At present, our bundle handles the management of the following types of resources : texts, files, directories and forum. We've designed the code to make it easy to add new types of resources. A tag has been created to regroup the services managing a type of resource for the OfflineBundle. Thoses services have to implement some methods that indicate to the synchronization process how to save, create and duplicate a resource of the given type. The entire bundle has been developed in collaboration with Claroline, the code has been made modular to be improved and to be more maintainable.

Résumé

De nos jours, l'e-learning (apprentissage en ligne) prend une place de plus en plus importante dans l'éducation. Toutes les régions du monde ne disposent pas d'une connexion Internet performante. De ce fait, tout le monde ne dispose pas d'un accès équivalent aux plateformes d'apprentissage en ligne. Ce mémoire a pour objectif de créer et développer un outil pour la plateforme d'apprentissage en ligne, Claroline. Cet outil a pour but de permettre à un utilisateur de travailler sur la plateforme dès lors qu'il n'a pas d'accès Internet. En pratique, un utilisateur doit être capable de charger les pages du site lorsque son ordinateur est déconnecté. De plus, il doit pouvoir se synchroniser lorsqu'il dispose d'un accès Internet. Cette synchronisation doit être bidirectionnelle, en ce sens que les ressources produites lorsqu'il était déconnecté doivent être envoyées sur le serveur distant et que les nouvelles ressources disponibles sur ce serveur doivent être téléchargées sur son ordinateur personnel.

Comme Claroline est développé à l'aide du framework de développement web PHP Symfony, nous avons conçu notre outil comme un bundle de Symfony. Plus précisément, notre bundle est conçu comme un plug-in facilement intégrable à Claroline. En plus du module de synchronisation, nous avons dû imaginer le moyen de diffuser notre programme. En effet, les utilisateurs voulant travailler déconnectés sur la plateforme, doivent pouvoir disposer d'une copie de Claroline sur leur ordinateur.

La conception du processus de synchronisation a été divisée en trois parties principales : la création, le transfert et le chargement. La première partie, la création, est chargée d'assembler les différentes modifications effectuées depuis une date de référence sur une plateforme. La seconde partie, le transfert est responsable quant à lui, de la communication entre l'ordinateur personnel et le serveur distant. Enfin, la troisième partie, le chargement, implémente l'intégration des modifications au sein d'une plateforme.

Techniquement, l'articulation de ces différentes parties en un processus global est implémentée dans le `SynchronisationManager`. Toutes les modifications faites sur une plateforme sont stockées dans un fichier XML que nous appelons manifeste et créons dans la classe `CreationManager`. Ce manifeste est enregistré avec les ressources fichiers dans une archive zip et est donnée au `TransferManager`. Cette archive est divisée en de multiples fragments et est transférée sur le serveur distant grâce à des requêtes HTTP POST. L'archive est chargée sur le serveur de manière à résoudre les conflits potentiels. Ensuite, le serveur crée sa propre archive de synchronisation qui sera téléchargée sur l'ordinateur personnel avant d'y être chargée. Finalement, nous mettons à jour la date de synchronisation et l'enregistrons dans l'entité `UserSynchronized`.

Pour déployer le code sur une plateforme Claroline existante, le plug-in doit être installé grâce au gestionnaire de paquets Composer. Pour l'installation sur l'ordinateur personnel, les utilisateurs doivent disposer d'un serveur PHP et d'un système de gestion de base de

données. Nous fournissons un paquet d'installation contenant le serveur web Xampp, car celui-ci est disponible pour les principaux systèmes d'exploitation ainsi que le code source de Claroline dans lequel nous avons pris soin d'intégrer le code du *OfflineBundle*.

Notre installateur est diffusé sous forme d'une version vierge de Claroline. Cela signifie qu'elle ne contient aucune donnée, par extension, aucun utilisateur n'y est inscrit. C'est pour cela que nous avons créé une vue demandant à l'utilisateur d'entrer ses identifiants ainsi que l'adresse de référence de la plateforme à synchroniser lors du premier démarrage. Avec ces informations, nous sommes à même de récupérer son profil depuis le serveur distant et de le recréer en base de données.

Actuellement, notre bundle est capable de gérer les types de ressource suivants : les textes, les fichiers, les répertoires et les forums. Nous avons conçu notre code de manière à ce qu'il soit facile d'ajouter de nouveaux types de ressources. Un tag a été créé pour regrouper les types de ressource pris en charge par *OfflineBundle*. Ces services implémentent des méthodes qui indiquent au processus de synchronisation comment sauver, créer et dupliquer un type donné de ressource. L'ensemble de notre bundle a été développé en collaboration avec les développeurs de Claroline, nous avons écrit notre code de façon modulaire, ce qui le rend plus facilement maintenable et permet de l'améliorer.

Remerciements

Au terme de ce mémoire, nous tenons à remercier certaines personnes sans qui ce travail n'aurait pu être réalisé.

Tout d'abord, nous aimerions remercier notre promoteur, Marc Lobelle ainsi que Philippe Mercenier pour l'encadrement et les conseils qu'ils nous ont apportés tout au long de ce projet. Nous souhaitons remercier chaleureusement Stephane Klein qui nous a accompagné durant toute l'année. Son soutien et ses précieux conseils ont grandement contribué à la réussite de ce travail.

Un grand merci également à Nicolas Godfraind pour le temps qu'il nous a consacré. Nous remercions aussi toute l'équipe de Claroline pour l'accueil et l'ambiance de travail agréable au sein de leurs locaux. Nous aimerions aussi remercier nos familles et nos amis qui nous ont toujours soutenus pendant la durée de nos études. Un merci particulier à Marie Dechamps, Elsa Boonen et Nicolas Burny pour leurs relectures et corrections orthographiques.

En dernier, nous tenons chacun à remercier notre binôme respectif pour l'excellente année passée en sa compagnie. La motivation et les ressources dont il a fait preuve ont grandement contribué à la bonne réalisation de ce mémoire.

Table des matières

Abstract	3
Résumé	5
Remerciements	7
1 Introduction	13
2 Lexique Claroline	15
2.1 Lexique Claroline	15
2.2 Vocabulaire informatique	16
3 Mise en contexte	17
4 Choix de conception et outils	19
4.1 Choix de la solution	19
4.2 Technologies utilisées	20
4.2.1 Symfony	20
4.2.2 HTML	21
4.2.3 Behat	21
4.2.4 PHP CS Fixer	22
4.3 Découpe en sous problèmes : la synchronisation	22
4.3.1 Création d'une archive	22
4.3.2 Transfert d'une archive	22
4.3.3 Chargement d'une archive	23
4.4 Découpe en sous problèmes : l'installation	23
4.5 Modélisation du OfflineBundle	23
4.5.1 Les routes	23
4.5.2 Les classes contrôleurs	23
4.5.3 Les classes managers	24
4.5.4 Les classes repository	24
4.5.5 Utilisation des paramètres	24
4.5.6 Twigs	24
4.5.7 UserSynchronized	25
5 Le processus global	27
5.1 Le processus de base	27
5.2 De multiples paquets	28
5.3 Ajout des dates de synchronisation et d'envoi	28
5.4 Gestion de la résistance aux coupures	31

5.4.1	L'implémentation de <code>SynchronisationManager</code>	31
5.4.2	Coupures électriques	32
6	Création de l'archive	33
6.1	Le manifeste	33
6.1.1	Définition : EXtensible Markup Language (XML)	33
6.1.2	Définition : JavaScript Object Notation (JSON)	33
6.1.3	Explication du choix	34
6.2	Implémentation	34
6.2.1	Services Taggés : l'injection des dépendances	34
6.2.2	Ecriture du manifeste	35
6.3	Spécificités des différents types de ressources	39
6.3.1	Ressource Texte	39
6.3.2	Ressource Fichier	40
6.3.3	Ressource Répertoire	40
6.3.4	Ressource Forum	41
6.3.5	Espaces d'activités	42
6.4	Récapitulatif	43
6.5	Ajout d'une nouvelle ressource	44
7	Transfert de l'archive	49
7.1	<code>TransferManager</code>	49
7.1.1	Des requêtes POST	49
7.1.2	<i>Buzz</i> et JSON	49
7.2	<code>SynchronisationController</code>	50
7.2.1	Les routes	50
7.2.2	Mécaniques de tests	51
7.3	Division en multiples fragments	52
7.3.1	Détermination de la taille des fragments	52
7.3.2	Modifications dans l'implémentation	55
7.3.3	Stockage temporaire	56
7.4	Authentification	56
7.4.1	Implémentation	57
7.5	Gestion des erreurs	57
7.6	Alternatives aux requêtes POST	58
7.6.1	WebSocket	58
7.6.2	Torrent	59
7.7	Amélioration du transfert	59
7.7.1	Création d'un RamDisk	60
8	Chargement de l'archive	61
8.1	Gestion des espaces d'activités	61
8.1.1	Gestion du créateur	62
8.1.2	Gestion des rôles	63

<i>TABLE DES MATIÈRES</i>	11
8.2 Gestion des ressources	63
8.2.1 Création d'une nouvelle ressource	64
8.2.2 Mise à jour d'une ressource	65
8.2.3 Création de doublon d'une ressource	65
8.2.4 Création et mise à jour des Forums	66
8.2.5 Affichage des opérations	67
8.3 Cas problématique : la date de modification	67
8.4 Cas problématique : parents disparus	68
8.5 Mécanisme de gestion du hashname	69
8.5.1 Scénarios	70
9 Installation	71
9.1 Ajout du plug-in OfflineBundle sur un serveur Claroline	71
9.1.1 Ajout automatique	71
9.1.2 Ajout manuel	72
9.2 Création d'un installateur	73
9.2.1 Choix de la solution	73
9.2.2 Création d'un installateur pour Windows	74
9.2.3 Créer le paquet de ressources pour Linux	80
10 Récupération d'un compte	83
10.1 Première connexion	83
10.1.1 Message d'erreurs	85
10.2 Ajout d'un autre utilisateur	85
10.3 Modification de l'adresse du site	86
11 Discussion	87
11.1 La fonction de suppression	87
11.2 Inscription à un nouvel espace d'activités	88
11.3 Gestion des fuseaux horaires	89
12 Conclusion	91
Bibliographie	93
Annexes	I
A OfflineRessource Exemple	III
B Inno Setup	VII
B.1 Utilisation du wizard	VII
B.2 Script Inno Setup	XIII

C	Manuel d'utilisation	XVII
C.1	Introduction	XVII
C.2	Installer Claroffline sur son ordinateur	XVII
C.2.1	Windows	XVII
C.2.2	Linux	XIX
C.2.3	Autres systèmes d'exploitation	XXII
C.3	Configurer son compte	XXII
C.4	Se synchroniser	XXII
C.5	S'inscrire à un nouveau cours	XXIV
C.6	Cas particulier de synchronisation	XXIV
C.7	Ajout d'un autre utilisateur	XXV
C.8	Le panneau de paramètres	XXV
C.9	Les messages d'erreurs	XXV
C.10	Désinstallation	XXVI
C.11	Changement de mot de passe	XXVII
D	Code Source	XXIX

Introduction

Dans le présent document, vous trouverez une mise en contexte encadrant notre travail et décrivant les différentes contraintes nous étant imposées. Ensuite, un chapitre présentera nos choix de conceptions et notre découpe en sous-problèmes. Ce chapitre sera suivi par une description plus en profondeur de notre processus global décrivant le fonctionnement de notre application dans son ensemble.

Ce point sera suivi par cinq autres chapitres décrivant avec précision les différents sous-problèmes que nous avons analysés. Enfin, un chapitre dédié aux discussions sur les limites de notre travail et les pistes de développement terminera ce document avant une note de conclusion. Ce document constitue une documentation pour les développeurs qui vont contribuer à notre travail. Nous y détaillons tous les choix d'implémentation que nous avons faits et pourquoi nous avons opéré en ce sens.

En annexe, une documentation pour les administrateurs qui voudraient ajouter notre bundle au sein de leur plateforme Claroline Connect est également fournie. Enfin, une documentation pour l'utilisateur voulant synchroniser sa plateforme est disponible au travers de vues sur le menus dédiés du site internet.

Enfin, il est à noter que durant toute la réalisation de ce projet, nous sommes restés en contact avec les développeurs de Claroline qui ont approuvé notre travail.

Lexique Claroline

Le lecteur pourra trouver ci-dessous une définition concise des termes régulièrement utilisés tout au long du présent document afin de l'aider dans sa compréhension du texte.

2.1 Lexique Claroline

Espace d'activités : Aussi appelé *workspace* en anglais, il est à Claroline Connect ce qu'était un cours dans la première version de Claroline. Il s'agit donc d'un espace sur le site regroupant des ressources (fichiers, forums,...), disposant d'un manager (professeur) organisant l'espace d'activités et de collaborateurs (étudiants) consultant son contenu.

Ressource : Au sein de la plateforme Claroline, les ressources sont les composantes d'un espace d'activités, il peut s'agir de fichiers PDF ou *PowerPoint*, d'un Forum ou d'une Annonce par exemple.

Type de ressource - Fichier : Au sein d'un espace d'activités, ce type de ressource définit un fichier chargé par son gestionnaire. Il peut s'agir de fichier *PowerPoint*, de *Portable Document File* (PDF) ou de tout autre type de document.

Type de ressource - Texte : Il s'agit d'une zone de texte éditée directement depuis la plateforme. Cette zone peut contenir tout type de texte, allant du simple texte informatif au texte dit riche tel que du HTML.

Type de ressource - Répertoire : Il est l'équivalent d'un dossier au sein d'un système d'exploitation classique ; il s'agit d'une ressource en rassemblant d'autres. En d'autres termes, une ressource répertoire est donc un regroupement d'autres ressources de tout type au sein d'un espace d'activités.

Type de ressource - Forum : Les forums sont des espaces d'échanges virtuels sur lesquels des utilisateurs ont la possibilité de consulter des messages postés par les autres utilisateurs et d'y répondre en postant un message de leur choix. Ces messages sont classés par catégories et sujets.

Catégorie : Les catégories sont à la base d'un Forum et regroupent plusieurs sujets. Elles sont en quelque sorte un répertoire de sujets.

Sujet : Au sein de chaque catégorie se trouvent des sujets ; ces sujets sont un ensemble de messages. Ils déterminent le thème des conversations.

Message : Au sein de chaque sujet se trouvent plusieurs messages. Ils sont l'expression participative des utilisateurs du forum.

2.2 Vocabulaire informatique

Bundle : Un Bundle est un ensemble de fichiers (classes PHP, JavaScripts, ...) implémentant une fonctionnalité. Nous définissons *Symfony* et expliquons le fonctionnement des bundles dans le chapitre 4.2.1 page 20. [26]

Conteneur de services : Un conteneur de services est un objet s'occupant de l'instanciation, de l'organisation et de la récupération des Services au sein d'une application. [28]

Listener : Un Listener (ou écouteur) d'évènements est un élément à l'écoute d'évènements, c'est-à-dire des changements d'états. Les Listeners permettent de spécifier un certain comportement lorsque des évènements précis sont rencontrés.

Métadonnées : Les métadonnées sont des données sur les données. Par exemple, dans le cas d'un document PDF, l'utilisateur peut disposer de la taille du fichier, de sa date de création ou encore de son auteur.

Repository : Les Repository (ou dépôts) sont des classes en PHP utilisées afin d'aider à la récupération de différentes instanciations d'une classe particulière.

Requête SQL : Le Structured Query Language (SQL) est un langage de requête, c'est-à-dire un langage informatique destiné à interroger et manipuler des bases de données. Ce dernier permettant de chercher, ajouter, voire supprimer des données. SQL manipule des bases de données relationnelles, des stocks d'informations organisés selon le modèle de données relationnel.

Requête DQL : Le Doctrine Query Language (ou DQL) est similaire au SQL à l'exception que les requêtes s'effectuent sur un objet plutôt que sur les lignes d'une table.

Service : Terme générique désignant tout objet PHP effectuant une tâche spécifique. Dans *Symfony* un Service est souvent configuré et récupéré par un conteneur de services. [27]

YAML : YAML est un format de représentation de données par sérialisation Unicode. Il est fortement utilisé dans les fichiers de configuration de *Symfony*.

Mise en contexte

Notre mémoire trouve son origine dans un besoin des pays en voie de développement : pouvoir continuer un processus d'apprentissage sur les plateformes d'e-learning, terme désignant l'apprentissage sur Internet, alors que l'accès à celui-ci est limité.

Aujourd'hui les plateformes d'apprentissage en ligne sont devenues des outils incontournables dans le partage des connaissances et l'apprentissage des étudiants. Ces plateformes permettent aux formateurs et professeurs de créer un parcours pédagogique incluant notamment des lectures et évaluations. Par ailleurs, les étudiants ont bien souvent l'occasion de participer et de poser leurs questions au travers de forums, ce qui favorise le partage et l'interactivité entre les membres d'un cours. Toutefois, pour profiter de l'ensemble des services fournis par ces plateformes, il faut disposer d'une connexion Internet. Il n'y a pas de grande limitation pour les pays occidentaux étant donné les multiples possibilités de connexions. Mais, la situation n'est pas identique partout et la connectivité dans certaines zones du monde, notamment en Afrique, reste assez faible.

C'est dans ce cadre que nous nous sommes intéressés au mémoire proposé par Marc LOBELLE, professeur à l'Université Catholique de Louvain-la-Neuve et avons décidé de contribuer au développement de la plateforme Claroline. Ce travail rencontrait nos attentes, car il s'agissait d'un projet de développement dont l'utilité pourrait être directe. De plus, en tant qu'utilisateur régulier de la plateforme, développer une extension permettant son utilisation en l'absence d'accès à Internet, nous semblait particulièrement intéressant.

Claroline est un logiciel Open Source permettant de déployer une plateforme d'apprentissage sur Internet. Le développement de ce *Learning Management System* (LMS) libre et gratuit a été initié en 2001 par l'Université Catholique de Louvain¹ et est aujourd'hui piloté par le *Consortium Claroline*. Au sein de ce logiciel, un utilisateur a la possibilité de créer ou de rejoindre des espaces d'activités². Si un utilisateur est gestionnaire d'un espace d'activités, il a la capacité d'y créer des ressources³. Celles-ci seront alors accessibles à l'ensemble des utilisateurs membres de cet espace. Pour plus d'information sur le projet Claroline, nous invitons le lecteur à se rendre sur le site web officiel de la plateforme⁴.

1. Université Catholique de Louvain-la-Neuve - <http://uclouvain.be>

2. cfr. Lexique, chapitre 2.1 [page 15](#)

3. cfr. Lexique, chapitre 2.1 [page 15](#)

4. Claroline - <http://claroline.net/type/claroline>

Ce travail s'adresse à tous les utilisateurs d'une plateforme Claroline souhaitant continuer à travailler sans y être connecté en permanence. Un groupe d'utilisateurs potentiels du travail que nous avons développé serait composé d'étudiants ne disposant pas de connexion Internet fiable et/ou stable. Ces derniers pourraient donc se synchroniser ponctuellement sur la plateforme de référence lorsqu'ils se rendent à leur université ou dans tout autre lieu leur fournissant un accès au WEB.

Mais d'autres utilisateurs pourraient être intéressés par ce projet. En effet, nous avons également eu l'occasion de rencontrer une personne travaillant pour *Médecin Sans Frontière*⁵. Leurs équipes partant en mission possèdent une clé USB sur laquelle des cours sont chargés au préalable. Sur place, ce personnel ne dispose pas forcément d'un accès Internet constant mais doit cependant être à même de continuer sa formation et également d'envoyer ses questions par l'intermédiaire du système de forum.

L'objectif final de ce mémoire est de créer un plug-in pour Claroline, venant compléter les services proposés par la plateforme à ses utilisateurs. Celui-ci permet de synchroniser une plateforme en ligne, le serveur distant, et un ordinateur personnel disposant de l'application Claroffline. Dans la pratique, un utilisateur pourrait, s'il dispose d'une connexion à Internet, lancer le processus de synchronisation. Au cours de ce dernier, le plug-in va récupérer le contenu créé depuis la dernière synchronisation sur le serveur distant afin de les enregistrer sur l'ordinateur personnel. Et par la suite, il va charger sur le serveur distant, le travail qui a été effectué sur l'ordinateur personnel. On parle alors de synchronisation bidirectionnelle.

Notre application Claroffline, se destine à tout utilisateur de *Claroline Connect* désireux de pouvoir utiliser la plateforme sans disposer d'Internet. Il n'y a pas de restriction majeure à l'utilisation de notre plateforme, tout ordinateur quelque soit son système d'exploitation, capable de faire fonctionner un serveur PHP 5.4 ou supérieur et une système de gestion de base de données tel que MySql sera capable d'exécuter notre application.

5. Médecin Sans Frontières - <http://msf.org>

Choix de conception et outils

Dans ce chapitre nous allons commencer par expliquer le choix de notre solution parmi les possibilités que nous avons envisagées. Ensuite nous détaillerons les technologies utilisées pour réaliser cette solution. Enfin, nous expliquerons notre découpe en sous-problèmes avant de terminer par les détails de modélisation d'*OfflineBundle*.

4.1 Choix de la solution

Afin de réaliser ce projet, nous avons envisagé trois solutions possibles. La première d'entre elles, notre choix, est une solution directement intégrée à Claroline comme n'importe quel autre module composant l'outil. L'avantage principal de cette solution est qu'elle permet à l'utilisateur de continuer à travailler sur Claroline, qu'il dispose ou non d'une connexion Internet. Il retrouve ainsi la totalité de l'environnement et des fonctionnalités lui étant familières. De plus notre choix permet une synchronisation bidirectionnelle, ce qui est une des contraintes pour ce projet, et, de manière analogue aux autres solutions, permet de minimiser l'ensemble des informations échangées sur le réseau.

En revanche, le problème majeur de notre solution est le temps de latence du chargement des pages du site de Claroline via un serveur apache exécuté sur un ordinateur aux capacités hardware plus réduites. Etant donné que nous travaillons à mettre hors-ligne la plateforme Claroline, l'ensemble des contraintes s'appliquant à Claroline s'appliquent également à notre projet. Cela requiert également plus d'espace de stockage. Ces deux défauts ne se trouvent pas dans la troisième solution décrite ci-dessous.

La deuxième solution envisagée est l'utilisation du *local storage* d'HTML5 pour enregistrer via la cache les informations nécessaires au redémarrage de Claroline sans une connexion Internet. Le problème de cette solution est lié au décalage entre les technologies utilisées par Claroline et les possibilités offertes par HTML5. En effet, l'implémentation de Claroline utilise majoritairement des technologies web déployées côté serveur alors que dans ce cas-ci, nous aurions dû développer l'application côté utilisateur en HTML5. Le problème était donc l'ampleur dû travail à fournir dans le temps imparti dès lors que nous avons la contrainte de rendre utilisable l'ensemble de la plateforme sans disposer d'une connexion Internet.

La dernière solution à laquelle nous avons pensé était la création d'un logiciel détaché de Claroline, qui s'occuperait de synchroniser l'ensemble des ressources dans un répertoire défini par l'utilisateur. Cette solution aurait été similaire à d'autres logiciels connus tel que

Dropbox¹ ou Google Drive².

Les désavantages de cette dernière solution étaient nombreux : l'utilisateur n'aurait plus accès à l'interface qu'il connaît. Nous aurions dû concevoir des solutions pour pouvoir lire sur l'ordinateur toutes les ressources spécifiques telles que les forums ou les wikis, alors que tout ce travail est déjà implémenté dans Claroline. Enfin, les possibilités d'extension du logiciel sont plus limitées qu'avec notre solution. Par exemple, il semble compliqué d'envisager une mise hors-ligne des fonctionnalités de modifications des paramètres d'un compte.

4.2 Technologies utilisées

Nous avons utilisé le langage PHP qui est l'un des plus utilisés sur le WEB et le framework *Symfony* afin de faciliter l'intégration avec la plateforme Claroline car cette dernière est développée avec *Symfony*. Le langage PHP constitue un *langage de script généraliste et open source spécialement conçu pour le développement d'application web. Il peut être intégré facilement au HTML*[21]. Une deuxième définition qui, à notre sens, résume bien PHP est la suivante : *PHP est un langage de programmation compilé à la volée principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale.*

4.2.1 Symfony

Symfony est un framework de développement WEB écrit en PHP et se basant sur le patron *Modèle Vue Controller* (MVC).[3] Lancé en 2005, ce framework en est actuellement à sa deuxième version. Un framework est un cadre de travail créé par des développeurs pour d'autres développeurs ; il n'est pas conçu pour des utilisateurs finaux. En d'autres termes, il s'agit donc d'une série d'outils servant à concevoir les fondations de l'architecture et les grandes lignes d'un logiciel. Concrètement, *Symfony* permet grâce à ses composants un gain de productivité pour la mise en place d'un site Internet[1].

Le principe du patron *Modèle Vue Controller* est de structurer les responsabilités au sein d'une application conçue en orienté objet. Celui-ci propose une découpe des classes en trois catégories : le modèle représentant le coeur algorithmique de l'application, la vue contenant les interactions avec l'utilisateur et le controller servant de lien entre la vue et le modèle.

Les bundles

L'architecture d'une plateforme *Symfony* est découpée en plusieurs parties nommées bundles. Ces derniers sont à la source de la modularité du framework. En effet, ils permettent d'intégrer simplement une fonctionnalité qui aurait été développée par quelqu'un d'autre. Nous pouvons citer par exemple *Buzz* le client HTTP que nous utilisons pour implémenter

1. <https://www.dropbox.com/>

2. <https://drive.google.com>

notre transfert, *twig-js-bundle* le moteur de rendu javascript des templates *Twig* ou encore *hwi-oauth-bundle* permettant l'authentification via les réseaux sociaux.

Le projet Claroline est lui même divisé en plusieurs bundles ; citons notamment le *CoreBundle*, reprenant les éléments de base nécessaires au fonctionnement de Claroline ou encore le *ForumBundle* s'occupant comme son nom l'indique de la gestion des ressources forums. La liste complète des bundles de Claroline peut être consultée avec le code source du projet distribué sur Github³. C'est dans ce contexte que vient s'ajouter notre *OfflineBundle* pour Claroline. Il est destiné à être intégré à Claroline, comme n'importe quel autre bundle.

Composer

Composer est un gestionnaire de dépendances écrit en *PHP* utilisant certains composants de *Symfony*. C'est ce gestionnaire de dépendances qui est utilisé par *Claroline*. Par exemple, *ClarolineCoreBundle* requiert *ClarolineForumBundle*. Lorsque l'administrateur système met à jour sa plateforme Claroline, *Composer* est exécuté et ira chercher la dernière version de *ForumBundle* sur le dépôt référence de Claroline. La liste exhaustive des bundles dont dépend un projet se trouve dans le fichier `composer.json` situé à la racine de son répertoire.

Il existe des conventions de nommages pour les bundles, ceux-ci résultent de la concaténation du nom du développeur (ou de l'entreprise développant le bundle) et du nom du bundle. Cette nomenclature détermine également l'emplacement des fichiers dans le projet, ceux-ci sont ordonnés dans le dossier `vendor` et suivant une arborescence similaire au nom.

4.2.2 HTML

HyperText Markup Language (HTML) est le langage standard utilisé pour créer des pages Web. Il s'agit d'un langage fonctionnant par balise c'est-à-dire qu'il est composé d'éléments HTML consistant en une série de balises fonctionnant le plus souvent par paires (exemple : `<p>... </p>` pour un paragraphe) bien qu'il existe quelques exceptions comme par exemple la balise `
` pour faire un retour à la ligne au sein d'un document.[?]

4.2.3 Behat

Afin de tester certaines fonctionnalités de notre plateforme, nous avons utilisé un outil de test orienté développement appelé Behat⁴. L'avantage de cet outil est qu'il permet d'écrire des scénarios (situations rencontrées par l'utilisateur) liés à du code source qui exécute les tests. Chaque ligne d'un scénario correspond à une fonction générique ou spécifique dans notre code.

Nous avons choisi d'utiliser Behat car il s'agissait d'un outil de tests déjà utilisé par Claroline mais également parce qu'il est très similaire à Cucumber⁵, un outil de test avec

3. <https://github.com/claroline>

4. Site internet de Behat : <http://docs.behat.org/en/v2.5/>

5. Site internet de Cucumber : <http://cukes.info/>

scénarios que nous avons largement utilisé avec le langage Ruby dans le cadre du cours de *Software Development Project*.

4.2.4 PHP CS Fixer

PHP CS Fixer⁶ est un outil permettant de s'assurer qu'un ensemble de fichier *PHP* d'un répertoire respecte la syntaxe et les conventions d'alignement. Nous avons utilisé PHP CS Fixer afin de corriger l'indentation de l'ensemble des classes *PHP* que nous avons implémentées [29].

4.3 Découpe en sous problèmes : la synchronisation

Maintenant que le cadre a été défini et que les principaux outils ont été présentés, nous pouvons passer à la découpe en sous problèmes de ce projet. Deux parties majeures nous sont apparues : la synchronisation et l'installation. La première partie, la synchronisation, se charge, à la demande de l'utilisateur, de mettre à jour le serveur distant avec le contenu nouvellement créé sur l'application Claroffline et réciproquement. En d'autres termes, l'objectif de cette partie du travail est de maintenir l'utilisateur à jour, tant au niveau des nouvelles ressources créées en ligne au sein de ses espaces d'activités que du contenu qu'il a créé hors-ligne sur son ordinateur personnel.

Lors de notre analyse sur cette phase de synchronisation, nous avons encore découpé la tâche en trois sous-parties : un premier module créant une archive contenant les modifications effectuées depuis une date repère. Un second, responsable de transférer les données d'une plateforme à une autre. Et le dernier chargeant les modifications contenues au sein d'une archive vers une plateforme. Ces trois sous-sections disposeront chacune d'un chapitre pour y détailler avec précision le travail qui y a été apporté. Nous avons pu établir une découpe claire des tâches, ce qui soutient l'idée que notre modélisation est bien conçue.

4.3.1 Création d'une archive

La création de l'archive consiste à récupérer toutes les modifications qui ont eu lieu dans les espaces d'activités⁷ auquel un utilisateur est inscrit depuis la dernière synchronisation et d'intégrer celle-ci dans un fichier transférable d'une plateforme à une autre ; en l'occurrence une archive zip.

4.3.2 Transfert d'une archive

La seconde étape du processus consiste à transférer cette archive d'une plateforme à une autre. Pour ce faire, nous nous sommes reposés sur le protocole HTTP et transférons les données au travers de requêtes POST. Par ailleurs, nous avons dû prendre en compte le fait que notre projet s'adresse à des zones du monde où la connectivité est limitée. Il a donc

6. Dépôt officiel de PHP CS Fixer : <https://github.com/fabpot/PHP-CS-Fixer>

7. cfr. Lexique, chapitre 2.1 page 15

fallu s'arranger pour que le transfert soit le plus robuste possible et puisse reprendre là où il s'était arrêté tout en minimisant les échanges multiples d'une même donnée.

4.3.3 Chargement d'une archive

Le chargement de l'archive est l'étape lors de laquelle nous intégrons à une plateforme une archive comprenant une série de changements, de travaux qui ont été effectués par l'utilisateur. Cette étape inclut la résolution des éventuels conflits qui pourraient avoir lieu entre deux ressources similaires voire identiques.

4.4 Découpe en sous problèmes : l'installation

La seconde partie du travail, l'installation, se charge de mettre à disposition une plateforme utilisable par l'utilisateur déconnecté. Il s'agit d'une plateforme que l'utilisateur sera à même de mettre en place sur son ordinateur personnel ou sur tout autre dispositif de stockage de masse. Cette plateforme disposera des espaces d'activités de l'utilisateur et de la possibilité de se synchroniser.

Nous avons fait le choix de concevoir d'installation de notre application Claroffline comme une plateforme Claroline vierge de tout contenu. Elle pourra alors être téléchargée et directement utilisée. L'utilisateur devra, une fois le logiciel téléchargé et démarré, entrer ses identifiants pour ensuite effectuer la première synchronisation qui téléchargera l'ensemble des ressources des différents espaces d'activités auxquels il est inscrit. Vous trouverez l'ensemble des détails à propos de l'installation dans le chapitre [9 page 71](#).

4.5 Modélisation du OfflineBundle

Dans cette section nous allons expliquer les différents composants de *Symfony* et la manière dont nous les avons articulés afin de modéliser notre bundle.

4.5.1 Les routes

Les routes sont les URL du site que l'on définit et auxquelles on associe des actions. Dans le cas de ce *OfflineBundle*, nous avons entre autre défini la route `"/sync/"` ; route que nous avons associée à la page d'accueil de notre module de synchronisation sur laquelle l'utilisateur reçoit les instructions pour synchroniser son ordinateur personnel. Le framework *Symfony* appliquant le modèle de design MVC, les routes sont définies au sein du contrôleur qui rappelle le sert de lien entre le modèle et les vues.

4.5.2 Les classes contrôleurs

Les contrôleurs sont les classes qui articulent l'application entre le modèle et les vues. Au sein d'*OfflineBundle*, nous avons défini deux contrôleurs : *OfflineController* et

`SynchronisationController`. Les actions de notre bundle sont réparties en deux contrôleurs distincts car ceux-ci ont des usages et des règles de sécurité différentes.

`OfflineController` est utilisé par l'application `Claroffline` pour la gestion des actions de l'utilisateur au sein des vues de notre module. L'accès à ensemble des actions qu'il permet est restreint à un utilisateur connecté sur la plateforme.

Le `SynchronisationController` s'occupe quant à lui de prendre en charge sur le serveur distant les requêtes faites par les utilisateurs déconnectés. De ce fait, l'accès à ses routes doit être libre ; la sécurité y sera gérée à l'aide d'une identification établie sur base d'informations contenues dans la requête.

4.5.3 Les classes managers

Un manager est une classe du modèle regroupant un ensemble de fonctions contribuant à un même but. Cette organisation a pour objectif d'alléger les contrôleurs, ces derniers devant être les plus concis possible. Dans le cas d'*OfflineBundle* nous avons créé quatre managers : `CreationManager` qui contient les fonctions se chargeant de la création de l'archive, `LoadingManager` qui contient les fonctions permettant le chargement d'une archive, `TransferManager` intégrant l'ensemble des fonctions utiles dans le transfert d'une archive et `SynchronisationManager` intégrant l'implémentation de l'articulation des différentes parties.

4.5.4 Les classes repository

Les repository sont un autre type de classe ; ils regroupent l'ensemble des requêtes de base de données utilisées pour accéder à un même type d'entité. Dans notre cas, nous n'avons implémenté qu'un repository, `UserSynchronizedRepository`, puisque nous n'avons qu'une seule entité enregistrée en base données, `UserSynchronized`.

4.5.5 Utilisation des paramètres

Pour la réalisation d'*OfflineBundle* nous avons utilisés les paramètres de *Symfony* afin d'enregistrer les variables globales à l'ensemble du bundle dont nous avons besoin. Ces paramètres sont enregistrés dans le fichier `Resource/config/services.yml`. Ils ont l'avantages de pouvoir être chargés directement dans le constructeur d'une classe.

Dans ces paramètres nous enregistrons notamment l'emplacement des répertoires utilisés pour stocker temporairement les archives chargées et téléchargées lors des synchronisations ou encore la localisation du fichier de configuration.

4.5.6 Twigs

Twig est le moteur de template par défaut de *Symfony 2*. Il s'agit de fichiers HTML dans lequel des tags supplémentaires sont ajoutés de manière à être générés dynamiquement lors

de l'interprétation du code. Twig fonctionne en compilant le template la première fois qu'il est demandé et est ensuite sauvegardé en cache. Ce système fait qu'en production, un template qui a déjà été compilé et chargé en cache ne sera plus modifié même si les sources du template sont changées. Pour forcer une nouvelle compilation de ces templates, une suppression de la cache est requise.

La syntaxe de Twig est fortement inspirée du système de template de *Django*, un web framework *Python* populaire. On retrouve l'ensemble des fichiers twigs dans le répertoire `Resources/Views`. Pour *OfflineBundle* nous l'utilisons entre autre pour afficher : le formulaire de première synchronisation, la vue informative pour l'utilisateur et les résultats de la synchronisation.

4.5.7 UserSynchronized

Dans *Symfony*, les *Entity* représentent des objets du modèle. Ceux-ci sont rendus persistants dans une base de données à l'aide de *Doctrine*, l'ORM utilisé dans *Symfony*, donc avec *Claroline*. En programmation orientée objet, un ORM aussi appelé *Object Relational Mapping*, est une technique de programmation autorisant le développeur à travailler directement avec une base de données orientée objet ; l'ORM définissant les correspondances avec la base de données relationnelles.

UserSynchronized est une *Entity* que nous avons créée dans l'*OfflineBundle* afin de répondre à notre nécessité de retenir toutes les informations relatives à la synchronisation. Cet objet contient donc une référence à l'utilisateur, la date de dernière synchronisation, la date d'envoi du dernier paquet, le status dans lequel le dernier transfert s'est arrêté⁸ ainsi qu'un champ `filename` utile pour pouvoir redémarrer depuis certains états. Le schéma de la base donnée est illustré sur la FIGURE 4.1.

Pour implémenter la création de cet objet au sein de *Claroline* nous avons dû utiliser le système de migrations de base de données incluses avec l'application. L'ensemble des explications relatives au système de migrations de *Claroline* peut être trouvé sur le lien suivant : <https://github.com/claroline/MigrationBundle>.

8. Voir explication de l'implémentation du processus global 5.1 page 27

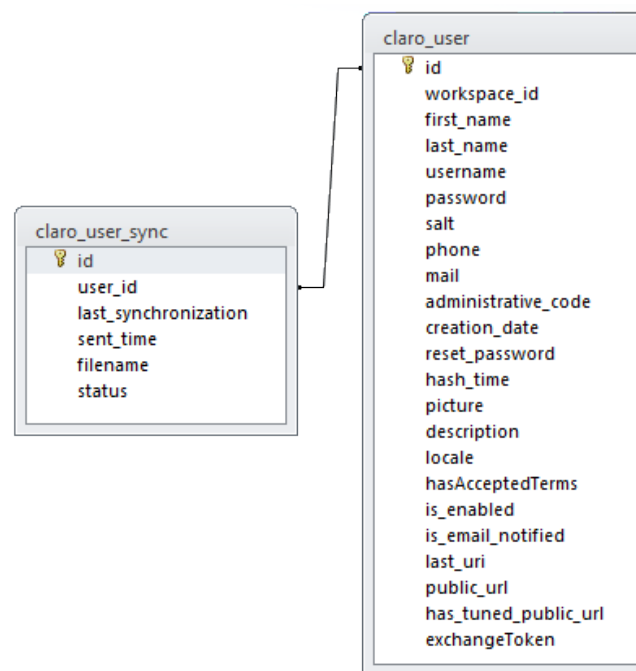


FIGURE 4.1 – Schéma de la table claro_user_sync représentant l'objet UserSynchronized

Le processus global

Dans ce chapitre, nous allons présenter comment nous combinons les trois parties principales, que nous venons de décrire, afin de réaliser la synchronisation complète d'un utilisateur déconnecté avec la plateforme Claroline de référence. Nous commençons avec le processus de manière simplifiée afin d'en présenter les bases. Ce schéma sera compléxifié tout au long de ce chapitre dans le but de présenter les différentes réflexions et solutions que nous y avons apportées.

5.1 Le processus de base

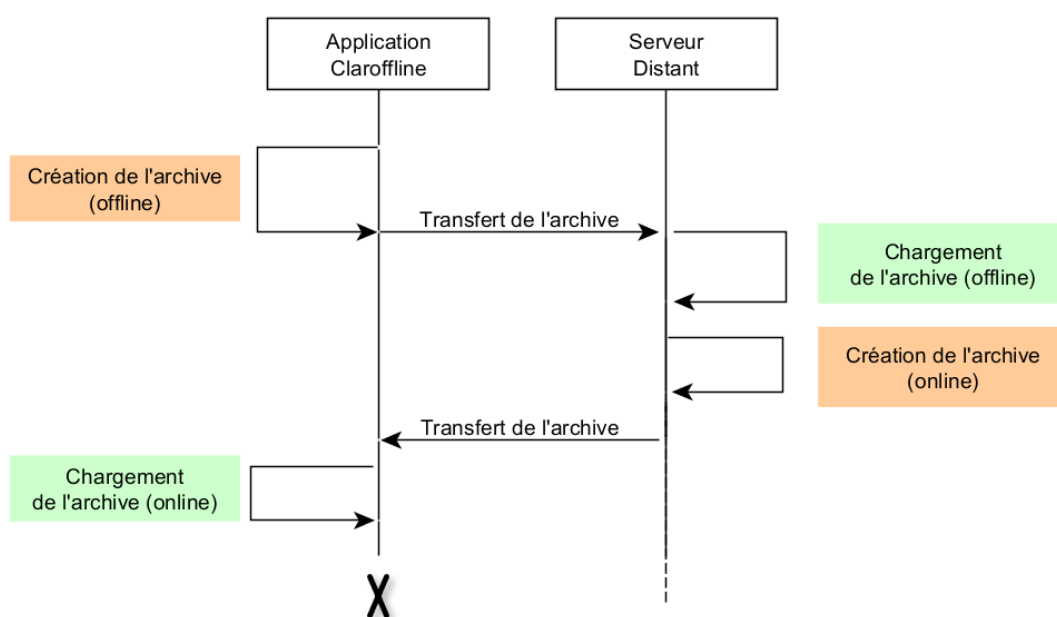


FIGURE 5.1 – Schéma du processus complet

La FIGURE 5.1 illustre de manière synthétique le processus complet de synchronisation que nous avons mis en place. De haut en bas, l'axe vertical représente l'avancement dans le temps. Comme nous pouvons le voir sur cette figure, le processus de synchronisation

commence par la création de l'archive de synchronisation sur l'application Claroffline. Cette archive est ensuite transférée sur le serveur distant où elle sera chargée. Durant le chargement, les éventuels conflits seront résolus. Le serveur en ligne créera alors à son tour son archive de synchronisation. Un transfert de cette archive vers l'application Claroffline est ensuite effectué. Le processus se termine par le chargement sur l'application Claroffline du nouveau contenu reçu depuis le serveur distant.

Les conflits sont résolus en ligne, ce qui signifie que toutes les résolutions de conflits seront similaires pour tous les utilisateurs. Nous pouvons voir sur la [FIGURE 5.1](#), au sein du serveur distant, que le processus de chargement s'effectue avant celui de création. En procédant comme tel, les conflits sont gérés par le serveur. Leurs résolutions seront alors ajoutées à l'archive transférée à l'application Claroffline.

5.2 De multiples paquets

Un élément important à noter sur la [FIGURE 5.1](#) est que le transfert y est représenté de manière simplifiée. Dans notre implémentation, comme nous le verrons dans le [chapitre 7 page 49](#), les archives sont découpées en de multiples paquets. Cette découpe est faite afin de permettre un transfert sur les connexions plus modestes et ainsi éviter les transferts lourds. Ces derniers pouvant être corrompus et nécessitant de réenvoyer plusieurs fois les mêmes données en cas d'échec. En envoyant de multiples petits paquets, chaque segment peut être validé indépendamment et le cas échéant seuls les éléments nécessaires sont retransférés. Pour garantir la sécurité de l'accès aux données, l'implémentation de notre système de transfert prend également en charge une identification de l'utilisateur à chaque requête à l'aide d'un identifiant unique.

5.3 Ajout des dates de synchronisation et d'envoi

Afin d'augmenter le nombre de cas gérés par notre processus, ce schéma doit encore être complexifié. Il est nécessaire de pouvoir retenir le dernier moment lors duquel nous nous sommes synchronisés, de manière à ne pas transférer l'ensemble des ressources de chaque espace d'activités à chaque synchronisation. En conservant cette information nous serons à même de détecter les ressources modifiées depuis la dernière synchronisation de l'utilisateur. Nous avons décidé d'enregistrer cette information dans l'entité `UserSynchronized` que nous avons créé à cet effet.

La date de dernière synchronisation est transférée par l'application Claroffline dans la requête de synchronisation. De cette manière, le serveur distant utilise la date que nous lui fournissons et répond en nous transmettant l'ensemble des changements ayant eu lieu depuis ce moment. Nous récupérons de ce fait le travail effectué qu'il nous manquait. A la fin du processus, la date de dernière synchronisation est alors mise à jour sur l'application Claroffline. Dans l'éventualité où l'utilisateur possède plusieurs plateformes déconnectées, chacune d'entre elles dispose de sa propre date de synchronisation. Cela permet à un utilisateur de

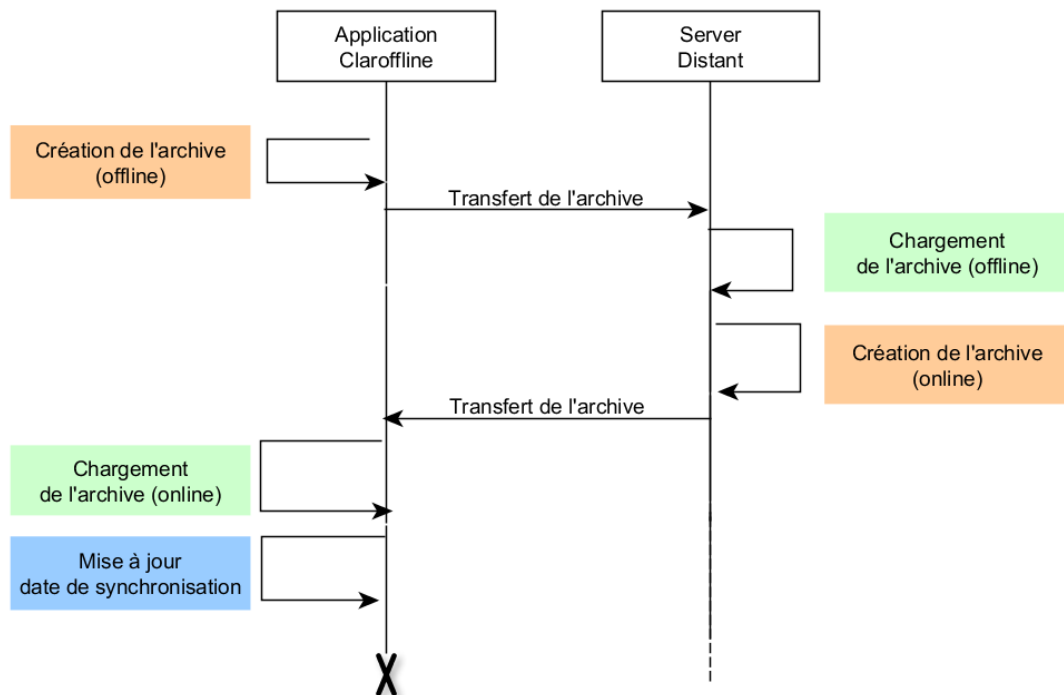


FIGURE 5.2 – Schéma du processus complet avec date de synchronisation

synchroniser plusieurs postes de travail différents. En effet, son compte n'est pas lié à une date de synchronisation enregistrée en ligne.

Toutefois nous avons remarqué que ce schéma présentait encore un défaut. Si une ressource est créée sur l'serveur distant pendant la période où un utilisateur déconnecté se synchronise, cette ressource ne sera jamais téléchargée sur l'application Claroffline. En effet, la date de synchronisation de l'utilisateur n'étant mise à jour sur la plateforme déconnectée qu'après traitement de l'archive reçue depuis le serveur distant ; la date de synchronisation sera forcément postérieure à la date de création de la ressource créée pendant la synchronisation. En conséquence, la ressource créée en ligne ne sera jamais intégrée à l'archive de synchronisation.

La solution que nous avons apportée pour résoudre ce problème est l'enregistrement de la date d'envoi de la requête. Cette date, sauvegardée dans le champ `sentTime` de notre entité `UserSynchronized`, sera utilisée comme valeur pour la date de synchronisation. Lorsque cette date est mise à jour, sa valeur sera donc la date à laquelle le processus a commencé plutôt que le moment où il s'est terminé. Il en résulte que la prochaine demande de synchronisation s'effectuera exactement depuis le début de la synchronisation précédente. Il n'y aura

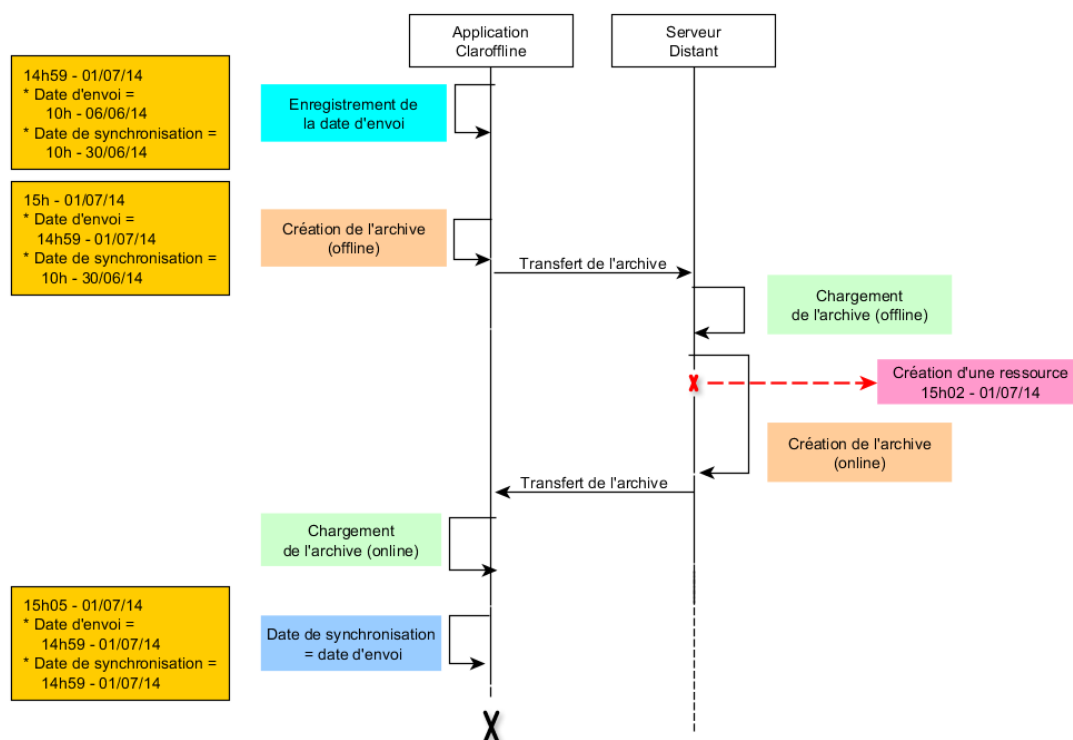


FIGURE 5.3 – Création d'une ressource pendant la synchronisation

plus de ressource ignorée par le processus.

Par exemple sur la figure 5.3, nous pouvons constater que l'utilisateur lance une synchronisation à 14h59. La date d'envoi de l'objet `UserSynchronized` de l'application Claroffline est alors mise à jour avec cette valeur. Lors de la création de l'archive du côté du serveur distant, une ressource est créée (à 15h02 sur le schéma). A la réception de l'archive par l'application Claroffline, la date de dernière synchronisation de l'objet `UserSynchronized` sera mise à jour avec la valeur de la date d'envoi, soit 14h59.

Grâce à cela, la prochaine fois que l'utilisateur lancera une synchronisation, il récupérera toutes les ressources créées depuis le 01/07 à 14h59 ce qui inclura celle créée à 15h02. En l'absence de la date d'envoi, la date de synchronisation aurait eu la valeur de 15h05 et, à la synchronisation suivante, la ressource créée à 15h02 aurait été ignorée.

5.4 Gestion de la résistance aux coupures

Nous avons conçu ce processus de sorte qu'il puisse être redémarré à chaque étape si l'une d'entre elle venait à échouer. Une des premières caractéristiques importantes est notre choix de ne mettre à jour la date de synchronisation qu'à la fin du processus complet. De cette manière, l'étape de création de l'archive peut être reprise comme s'il n'y avait pas eu de coupure. En effet, pour une date donnée, cette opération produit le même résultat. Par conséquent, si nous relançons le processus après un échec, la date n'aura pas été modifiée et donc la sortie de la fonction de création sera identique à celle obtenue lors de la précédente exécution.

L'implémentation du processus complet, a été organisée au sein du `synchronisationManager`. Cette implémentation a été découpée en différentes étapes, de telle sorte que si le processus est interrompu à un moment ou à un autre, il puisse être relancé automatiquement à l'étape voulue. Cela permet ainsi d'éviter de recommencer inutilement les opérations ayant déjà été réalisées et de minimiser l'utilisation du réseau. Afin d'assurer ce redémarrage automatique il nous faut enregistrer la dernière étape effectuée, pour cela nous sauvegardons à la fin de chaque étape le numéro de celle-ci dans le champ `status` de l'objet `UserSynchronized`. Il est important de préciser que cette implémentation permet d'ajouter facilement une nouvelle étape au processus.

Outre le champ `status` dans `UserSynchronized`, le champ `filename` est également utilisé pour la reprise du processus en cas de coupures. En effet, certaines étapes requièrent la mémorisation d'un nom de fichier, en l'occurrence le nom de l'archive devant être chargé ou inversement, téléchargé. La présence de cette information peut influencer sur la reprise du processus. Par exemple si le processus coupe au moment de l'envoi de l'archive et puis reprend, il va tout d'abord tester la présence de cette archive référencée par le `filename`. Si elle n'est plus disponible (pour une raison ou une autre) il va alors recommencer la construction de cette archive. Ici encore, nous retrouvons un élément qui nous conforte quant à la robustesse de notre processus.

5.4.1 L'implémentation de `SynchronisationManager`

L'ensemble des états que nous avons identifiés sur le processus global sont stockés sous forme de constantes au sein de l'entité `UserSynchronized`. Ces états sont les suivants : `SUCCESS_SYNC`, `STARTED_UPLOAD`, `FAIL_UPLOAD`, `SUCCESS_UPLOAD`, `FAIL_DOWNLOAD` et `SUCCESS_DOWNLOAD`. L'implémentation de l'articulation des étapes se fait dans le `SynchronisationManager`, pour cela nous utilisons la fonction `SynchronizeUser`. Cette fonction consiste en un `switch` qui définit l'étape devant être exécutée en fonction du `status` de l'entité `UserSynchronized`. Comme cette action est effectuée sur l'application Claroffline, il y a quatre étapes : créer l'archive de l'application Claroffline, charger celle-ci en ligne, télécharger l'archive du serveur distant et charger le contenu de celle-ci au sein de l'application Claroffline.

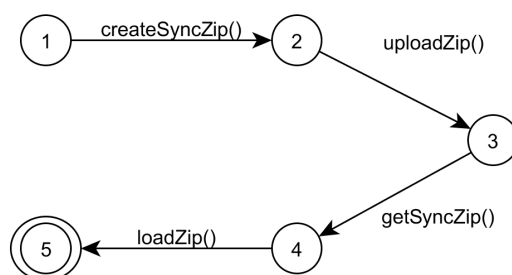


FIGURE 5.4 – Schéma des transitions du processus de synchronisation entre les étapes

Etape	Status
1	UserSynchronized::SUCCESS_SYNC
2	UserSynchronized::STARTED_UPLOAD
3	UserSynchronized::SUCCESS_UPLOAD
4	UserSynchronized::SUCCESS_DOWNLOAD
5	UserSynchronized::SUCCESS_SYNC

TABLE 5.1 – Correspondance entre les status UserSynchronized et les étapes de synchronisation

L'enchevêtrement des états et étapes de notre SynchronisationManager est représenté sur la figure 5.4. Lorsque nous sommes à l'étape 1 et que l'archive est créée nous passons à l'étape 2. De là, lorsque nous envoyons cette archive au serveur distant nous passons à l'étape 3. Durant cette dernière, l'application Claroffline télécharge l'archive du serveur distant ; ce qui nous amène à l'étape 4, qui consiste à charger cette archive au sein de l'application Claroffline. Après cette dernière étape, le processus est achevé. La table 5.1 fait la correspondance entre les étapes et les états enregistrés.

5.4.2 Coupures électriques

Une des contraintes évoquées par notre promoteur et que nous avons pris en compte est le cas des coupures électriques, qui peuvent arriver de manière inopinée. Les coupures électriques induisent une perte du contenu enregistré dans la mémoire temporaire (Random Access Memory). Nous avons donc décidé d'utiliser la mémoire persistante de l'ordinateur, soit le disque dur, soit le solid state drive (SSD) afin d'enregistrer les archives créées et chargées. Tous les fragments de paquets envoyés sont stockés sur la machine. Une fois que l'ensemble des fragments sont réunis, le paquet est recomposé. Les différents fragments sont bien entendu supprimés du répertoire après usage.

Création de l'archive

L'objectif de notre premier sous-problème est de créer une archive contenant l'ensemble des modifications apportées à une plateforme par rapport à une date de référence. Nous avons structuré cette archive avec deux éléments. Le premier étant un fichier XML que nous avons appelé "Manifest.xml", comprenant des indications sur toutes les modifications apportées par l'utilisateur depuis la dernière synchronisation. Le second est un dossier contenant l'ensemble des ressources de type Fichier créées et/ou modifiées par l'utilisateur .

6.1 Le manifeste

Le manifeste est un fichier que nous créons, il nous permet de stocker l'ensemble des informations nécessaires à la création ou à la mise à jour des espaces d'activités et des ressources qu'ils contiennent. Pour créer un tel fichier, deux possibilités efficaces, éprouvées et reconnues s'offraient à nous : le XML et le JSON. L'explication de ces deux alternatives et la justification de notre choix se trouvent ci-après.

6.1.1 Définition : EXtensible Markup Language (XML)

Le XML est un format de rédaction de document texte dérivé du standard (ISO 8879)¹. Ce langage, fut conçu à l'origine pour répondre aux besoins de publications électroniques à grande échelle. Aujourd'hui, il occupe une place importante dans les échanges de données sur le WEB ou dans d'autres systèmes, citons par exemple son usage important sur la plateforme mobile Android.

Le rôle du XML est de permettre le transfert de données comparativement au *HyperText Markup Language* (HTML) qui sert quant à lui à afficher du contenu. Le langage XML autorise le développeur à créer ses propres balises, ce qui permet d'ajouter une couche de sémantique à la syntaxe accompagnant les données. De plus, ce langage est devenu une recommandation du *World Wide Web Consortium* en 1998.

6.1.2 Définition : JavaScript Object Notation (JSON)

De manière similaire au XML, JSON est un format de données textuelles permettant l'enregistrement structuré d'informations en vue de les transférer. La différence entre ces deux

1. <http://www.w3.org/XML/>

formats réside dans la syntaxe utilisée pour l'enregistrement. Effectivement JSON utilise une syntaxe proche de celle Javascript² mais est indépendant de ce langage de programmation.

6.1.3 Explication du choix

Etant donné que ces deux solutions sont équivalentes; notre choix s'est porté sur le XML pour l'écriture du manifeste car il s'agit tout d'abord de la première solution complète correspondant à nos besoins. En effet, ces documents sont légers et autorisent un parsing efficace; ils rencontrent donc les deux caractéristiques recherchées. Cette solution nous est naturellement venue à l'esprit car nous avons déjà travaillé avec ce format de données. De plus, il constitue un standard en la matière.

D'un point de vue implémentation, nous avons utilisé le *Document Object Model (DOM)*. Ce dernier est un standard du W3C³ pour décrire une interface indépendante de tout langage de programmation qui permet à des programmes ou des scripts d'accéder ou de mettre à jour le contenu, la structure ou le style de documents XML et HTML[?]. Ce type d'interface disponible pour PHP, nous a permis de créer et analyser efficacement notre XML.

6.2 Implémentation

6.2.1 Services Taggés : l'injection des dépendances

Nous avons créé un Service pour chaque type de ressource de Claroline prise en compte par *OfflineBundle*. Un Service est un objet PHP effectuant une tâche particulière qui est utilisée tout au long de l'application. Le principal avantage à travailler avec les Services est qu'ils permettent de séparer les différentes fonctionnalités, ce qui est une bonne pratique en programmation orientée objet.

Nos Services prennent la forme d'une classe PHP étendant une classe abstraite. Chacun de nos Services définit les fonctions nécessaires à la gestion de leurs ressources par notre bundle. Pour identifier ces Services, nous leur apposons un Tag, c'est-à-dire une chaîne de caractères génériques. L'utilisation d'un Tag d'identification nous permet de demander au constructeur de conteneur de services⁴ de nous rendre la liste de tous les Services possédant un Tag donné. Le Tag que nous apposons est `claroline_offline.offline`.

Lors de la passe de compilation nous sommes à même de rassembler les Services possédant notre Tag et d'en faire prendre connaissance à nos deux managers responsables de la gestion de l'archive que sont `CreationManager` et `LoadingManager`. Durant cette passe, une structure de données contenant l'ensemble des Services est créée. Tous ces Services seront indexés par le type de ressource qu'ils supportent, ce qui facilite grandement leur

2. Javascript est un langage de programmation de scripts utilisés sur les pages web interactives. Pour plus d'informations : <https://developer.mozilla.org/fr/docs/JavaScript>

3. W3C - <http://www.w3.org>

4. Voir lexique : 2.2 page 16

utilisation par la suite.

De plus, l'utilisation de services taggés nous permet de facilement ajouter de nouvelles ressources à celles déjà gérées par notre bundle. La section 6.5 de ce chapitre est consacrée à l'explication de l'ajout d'un nouveau service destiné à gérer un nouveau type de ressource. Sur la FIGURE 6.2 se trouve le détail de nos Services Offline. Sur la partie supérieure du graphe se trouve notre classe abstraite `OfflineElement`, cette dernière comprend tous les attributs et méthodes utilisés par les autres classes Offline. Deux classes étendent cette dernière, `OfflineWorkspace`, qui comprend les méthodes relatives aux espaces d'activités et `OfflineResource`, qui est une classe abstraite qui reprend les cinq signatures de méthodes nécessaires à la synchronisation des ressources. Tout service destiné à un type particulier de ressource à synchroniser doit étendre `OfflineResource`. De manière plus imagée, la FIGURE ?? résume la hiérarchie de l'héritage entre nos classes.

6.2.2 Ecriture du manifeste

DOMDocument

Le manifeste prend la forme d'un objet `DOMDocument`. L'utilisation d'un `DOMDocument` nous permet d'ajouter facilement des éléments au document grâce aux fonctions `createAttribute()` et `appendChild()`. De plus, sur le `DOMDocument`, il existe une fonction `createCDATASection()` utilisée pour transférer le contenu ne devant pas être interprété en XML ce qui est le cas des ressources de type texte et des messages des ressources de type forum. Le rôle des sections `CDATA` sera expliqué dans les parties dédiées à ces ressources.

Section Description

Dans un premier temps, nous écrivons au sein du manifeste une description globale nécessaire lors de l'étape de chargement. Avec ses informations nous serons capables de vérifier l'identité de l'utilisateur. Cette description comprend :

- la date de création de l'archive
- la date de dernière synchronisation de l'utilisateur
- le pseudonyme de l'utilisateur
- l'adresse e-mail de l'utilisateur

Section Ressources

Une fois l'écriture de la description terminée, nous ajoutons au manifeste l'ensemble des modifications apportées aux différents espaces d'activités dont l'utilisateur est membre. Nous avons décidé de ne conserver que les ressources modifiées ou créées car ce projet se destinant à être utilisé dans des zones où la connectivité est faible, voire instable, chaque information à transférer compte d'autant plus. Il nous a donc paru important d'éviter au maximum de surcharger notre archive.

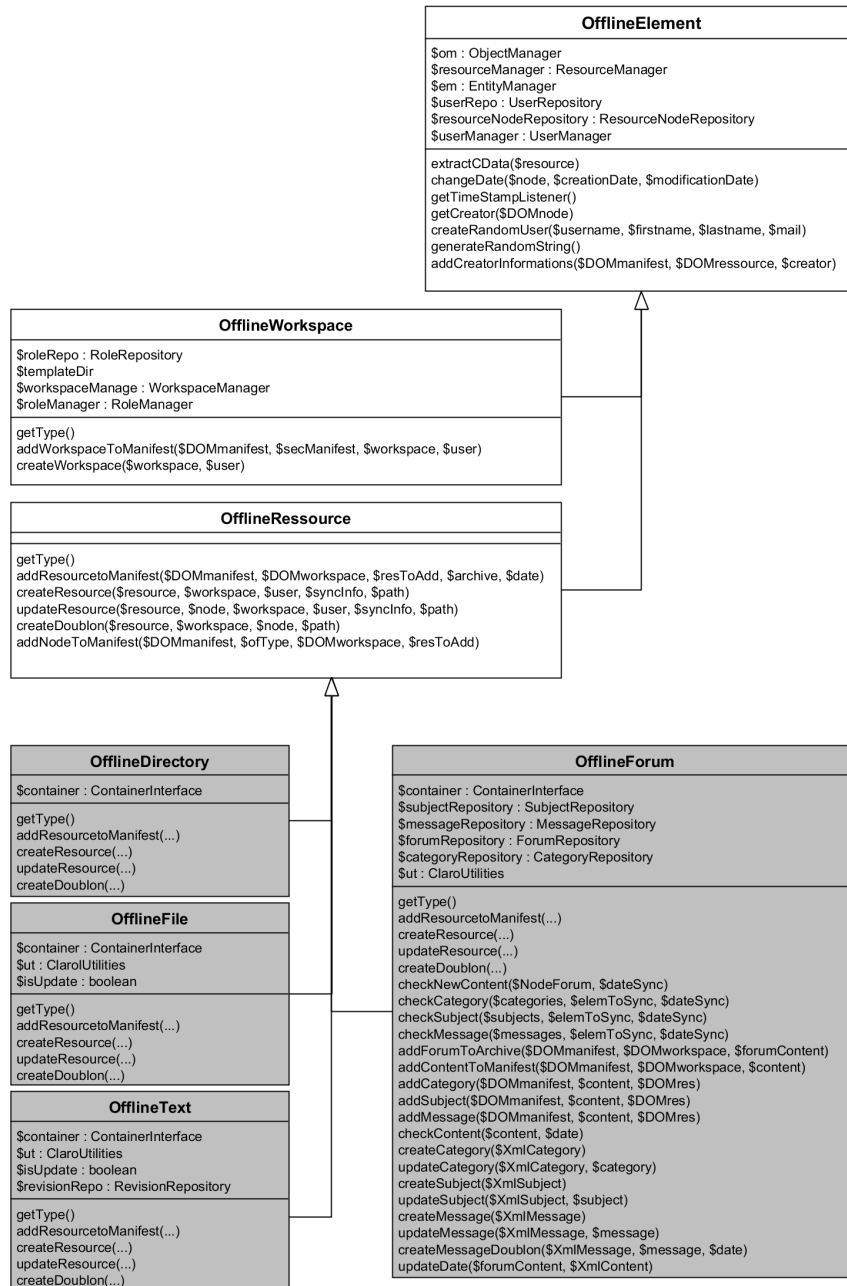


FIGURE 6.1 – Schéma UML des Services Offline

Avant de détailler davantage les spécificités de chaque ressource au point 6.3, il nous est nécessaire d'expliquer la manière dont les ressources sont schématisées dans la base de données. Comme montré sur la FIGURE 6.3, la représentation d'une ressource en base de

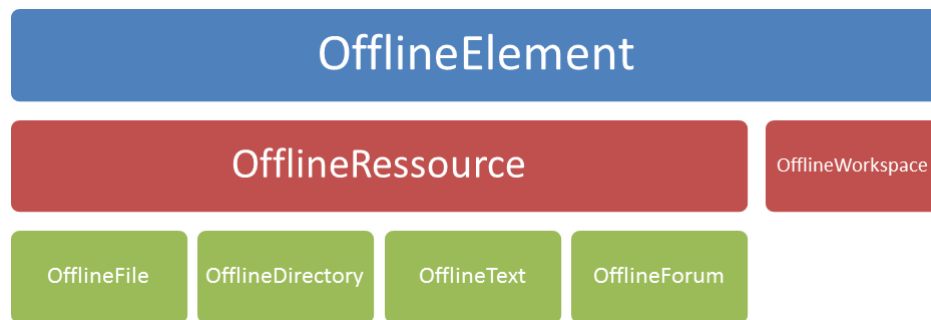


FIGURE 6.2 – Représentation imagée de l'héritage de nos classes de ressources Offline

données se base sur l'entité `resource_node`. Celle-ci contient la référence vers le type de la ressource, le `resource_type` de même que l'identifiant de l'espace d'activités, le `Workspace`, auquel cette ressource appartient. Par ailleurs, chaque ressource contient l'identifiant de l'entité `resource_node` qui lui correspond.

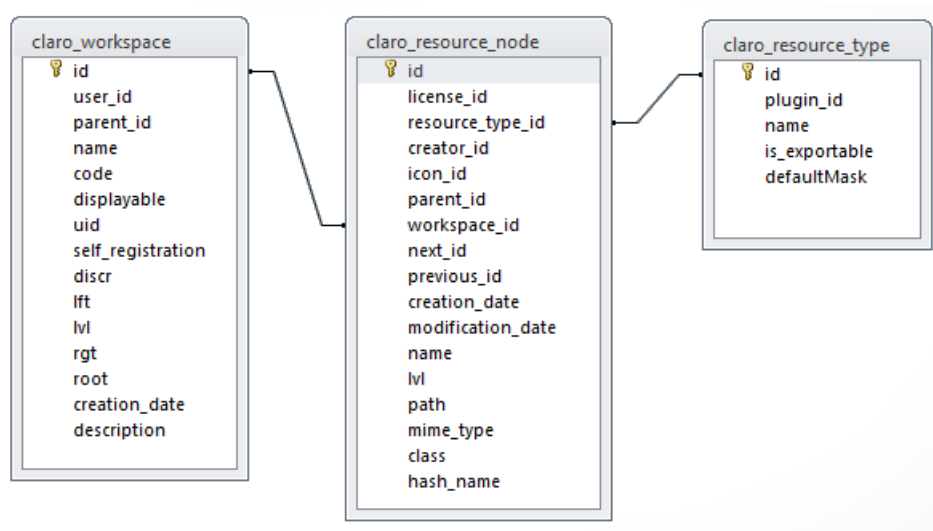


FIGURE 6.3 – Schéma général de la représentation d'une ressource en base de données

Lors de notre étape de création de l'archive, nous retrouvons les ressources à synchroniser comme suit. Nous effectuons une première requête sur la base de données dans le but de récupérer l'entièreté des espaces d'activités auxquels l'utilisateur est inscrit. Une fois cette liste d'espaces d'activités obtenue, nous utilisons une requête DQL conçue par nos soins qui a pour but de nous retourner en une seule fois toutes les ressources de cet espace d'activités qui sont à synchroniser et qui correspondent aux types de ressources gérés par nos services

décrits ci-dessus. Une requête DQL est similaire à une requête SQL, la principale différence étant que l'on raisonne en terme d'objets.

```
private function findResourceToSync(Workspace $workspace ,
                                   $types , $date)
{
    $query =
        $this->resourceNodeRepo->createQueryBuilder('res')
        ->join('res.resourceType', 'type')
        ->where('res.workspace = :workspace')
        ->andWhere('res.modificationDate > :date')
        ->andWhere('type.name IN (:types)')
        ->setParameter('workspace', $workspace)
        ->setParameter('types', $types)
        ->setParameter('date', $date)
        ->getQuery();

    return $query->getResult();
}
```

Afin d'obtenir les ressources à synchroniser nous paramétrons la requête DQL avec la date de dernière synchronisation (\$date). Pour rappel, les ressources à synchroniser sont celles dont la date de modification est postérieure à la date de synchronisation de l'utilisateur.

Pour s'assurer que le type de la ressource correspond aux types gérés par nos Services, nous passons en paramètre à la requête un tableau comprenant tous les types supportés. Ce tableau, \$types, est obtenu en utilisant la fonction `array_keys()` qui retourne l'ensemble des clés d'un tableau. La fonction `array_keys()` est appliquée dans notre cas à la variable \$offline de notre manager, qui est la structure de données comprenant tous nos Services taggés indexés par le type de ressource.

Ainsi nous ne conservons que les ressources ayant été créées et/ou modifiées depuis la date de dernière synchronisation de l'utilisateur. Ressources qu'il est par conséquent nécessaire de transmettre à l'autre plateforme.

Il nous reste maintenant à ajouter au manifeste les informations utiles à l'étape de chargement (cfr. 8 page 61). Cela comprend notamment le type de la ressource, sa clé unique appelée `hashname` en base de données ainsi que son créateur. L'opération s'effectue simplement grâce à la structure de données \$offline citée précédemment. En effet, étant donné qu'elle contient nos services indexés par leur type, on peut facilement accéder à la méthode d'écriture du manifeste du bon service de la manière suivante :

```
foreach ($ressourcesToSync as $res) {  
  
    $domManifest =  
        $this->offline [ $res->getResourceType()->getName() ]  
        ->addResourceToManifest (...);  
  
}
```

Où `$res->getResourceType()->getName()` nous donne le nom, sous forme d'une variable string, du type de la ressource.

6.3 Spécificités des différents types de ressources

Ci-dessous se trouvent les spécificités propres à chaque type de ressource et la manière dont nous les avons traitées. De plus, pour chaque ressource nous citons le nom du Service associé. Ces Services peuvent être retrouvés dans le dossier `Model\Resource` de notre *OfflineBundle*.

6.3.1 Ressource Texte

Lorsque nous avons une ressource de type Texte, il est également nécessaire d'aller chercher en base de données le texte auquel elle fait référence. Comme illustré sur la [FIGURE 6.4](#), ce texte est stocké sous l'entité `text_revision`. Comme vu précédemment, nous sommes capables de retrouver l'entité `text` sur base du `resource_node` qui lui est lié. A partir de cette entité `text`, nous pouvons atteindre l'entité `text_revision` désirée.

Comme expliqué dans la définition⁵, les ressources textes ont la particularité de pouvoir être riches. Toutefois cette richesse fut pour nous une source de complexité dans notre méthode d'enregistrement. Par exemple, s'il y a présence de balises HTML à l'intérieur du contenu du texte, cela cause une erreur. En effet, il ne peut y avoir présence de balises au sein d'un attribut XML.

Pour ce problème, le XML propose une solution : la section `CDATA`. Celle-ci offre la particularité de ne pas interpréter les balises contenues à l'intérieur jusqu'à la chaîne de caractère fermante qui, bien entendu, ne peut être utilisée comme séquence au sein du contenu. C'est-à-dire que ce dernier est considéré comme du texte et non pas comme du XML. Par ailleurs, il n'est pas autorisé à une section `CDATA` d'en contenir une autre. Notons toutefois que ce problème est géré par la fonction de l'API `DOMDocument` de PHP que nous utilisons pour écrire le contenu des ressources textes dans le manifeste.

La gestion des ressources textes dans notre programme est entièrement implémentée par le Service `OfflineText`.

5. cfr. Lexique, [CHAPITRE 2.1 page 15](#)

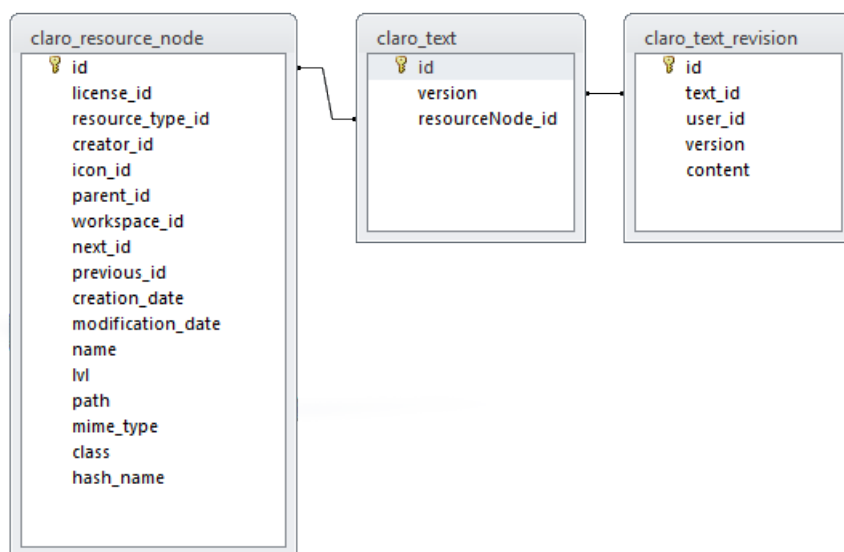


FIGURE 6.4 – Schéma d'une ressource Texte dans la base de données

6.3.2 Ressource Fichier

Pour rappel, une ressource du type Fichier est un document chargé au sein d'un espace d'activités par un de ses gestionnaires⁶. Ces ressources ont besoin d'une information supplémentaire dans le manifeste : la taille du fichier. De plus, ces ressources font appel à un document chargé par l'utilisateur. Ce document est placé au sein d'un répertoire particulier par exemple *files*⁷. La FIGURE 6.5 page suivante nous montre que l'entité *file* contient une clé *hashname* qui nous permet de retrouver le répertoire au sein duquel se trouve le document.

Il nous est nécessaire en cas de modification ou création d'incorporer ce document à notre archive. Comme l'entité *file* connaît le nom du document au sein du répertoire de la plateforme, il nous est alors possible de le retrouver et d'en faire une copie à l'intérieur d'un dossier de notre archive que nous avons nommée *data*.

La Service responsable de la gestion des ressources de type fichier est *OfflineFile*.

6.3.3 Ressource Répertoire

Une ressource Répertoire⁸ correspond à un dossier au sein d'un système d'exploitation classique, il s'agit d'une ressource en rassemblant d'autres. Chaque ressource, tout type confondu, se trouve à l'intérieur d'une ressource Répertoire ou d'un espace d'activités, ce qui forme une hiérarchie cohérente sous forme d'arborescence. Le point d'entrée de cette

6. cfr. Lexique, CHAPITRE 2.1 page 15

7. Le chemin complet de ce dossier est défini dans le fichier *parameters.yml* du *CoreBundle* de Caroline

8. cfr. Lexique, CHAPITRE 2.1 page 15

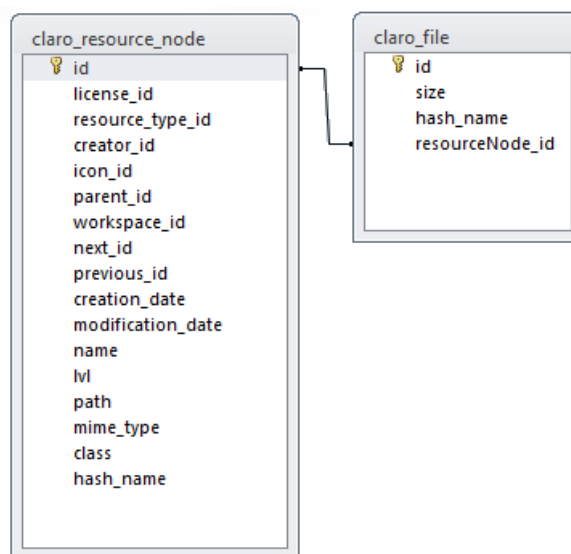


FIGURE 6.5 – Schéma d'une ressource Fichier dans la base de données

arborescence étant l'espace d'activités. Cela signifie qu'une ressource Répertoire peut-être contenue dans une ressource Répertoire mais qu'en revanche, un espace d'activités ne peut quant à lui être contenu dans un Répertoire.

En base de données, cette hiérarchie est symbolisée par le fait que chaque `resource_node` connaisse l'identifiant de son parent. C'est-à-dire de la `resource_node` correspondant à la ressource Répertoire ou à l'espace d'activités la contenant directement. La gestion de ces ressources ne nous a pas posé de problème particulier. Néanmoins, lors de l'analyse de ces ressources, nous avons découvert le problème des parents disparus dont nous discutons dans le chapitre du chargement⁹. Le Service OfflineDirectory prend en charge la gestion des ressources répertoires.

6.3.4 Ressource Forum

Pour les ressources Forums, implémentées dans `OfflineForum`, une procédure différente est requise. Comme vu dans le LEXIQUE, un forum est composé de Catégories, elles-mêmes composées de Sujets contenant un ou plusieurs Messages (illustration FIGURE 6.6)¹⁰. Du fait de cette structure particulière, il nous a fallu trouver un autre moyen de détecter les modifications apportées aux ressources Forums. En effet, dans la base de données, lorsqu'une nouvelle Catégorie, un nouveau Sujet ou Message est créé sur un forum, la date de modification de la `resource_node` correspondant à ce dernier n'est pas mise-à-jour. Par conséquent, lorsque nous traitons les ressources de type Forum, nous n'avons aucun moyen

9. cfr. CHAPITRE 8.4 page 68

10. cfr. Lexique, CHAPITRE 2.1 page 15

de savoir si le contenu de cette ressource a été modifié.

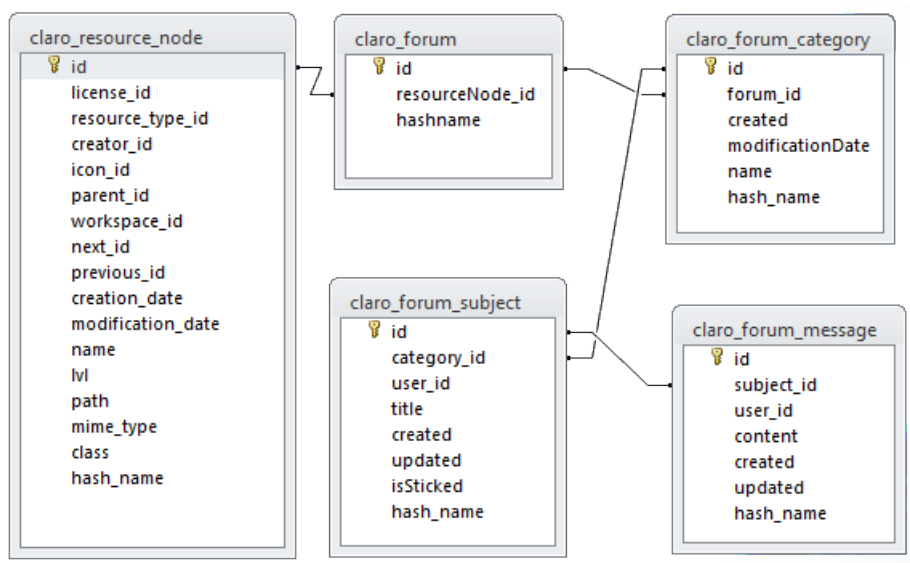


FIGURE 6.6 – Schéma d'une ressource Forum dans la base de données

Pour résoudre ce problème, nous faisons appel à une sous-méthode qui va parcourir l'ensemble des Forums. Pour chaque forum nous trouvons, à l'aide de requêtes, l'ensemble des catégories, sujets et messages le composant. Nous parcourons ensuite ces éléments et à chaque fois qu'une catégorie, qu'un sujet ou qu'un message ont été créés ou modifiés ; c'est-à-dire qu'elle possède en base de données une date de modification ultérieure à la date de dernière synchronisation de l'utilisateur, nous l'ajoutons à une liste dont les éléments seront par la suite ajoutés au manifeste.

Il est à noter que, tout comme pour les ressources Texte, le contenu des messages des forums contiennent des balises HTML. Par conséquent, ils nous a aussi été nécessaire de placer ce contenu dans une section CDATA au sein du manifeste.

6.3.5 Espaces d'activités

Un espace d'activités n'est pas une ressource à proprement parler. Toutefois, il est similaire à une ressource Répertoire, en ce sens que l'entité `resource_node` lui correspondant est du type Répertoire et qu'il est destiné à contenir d'autres ressources. De plus, les `resource_node` correspondant aux espaces d'activités ont la particularité d'avoir un champ `parent` enregistré à la valeur NULL et de cette manière, constituent la racine de l'arborescence des ressources.

En outre, les espaces activités définissent les rôles des utilisateurs, c'est-à-dire les possibilités d'actions dont ils disposent sur l'ensemble des ressources qu'ils contiennent. Ces rôles prennent la forme d'entités `claro_role` au sein de la base de données. Le lien entre un utilisateur et le rôle dont il dispose sur un espace d'activités est fait au sein de la table `claro_user_role`. Pour compléter le panel d'informations nécessaires à l'entité `Workspace`, une table `claro_workspace` existe. La Figure 6.7 illustre le schéma d'un rôle en base de données.

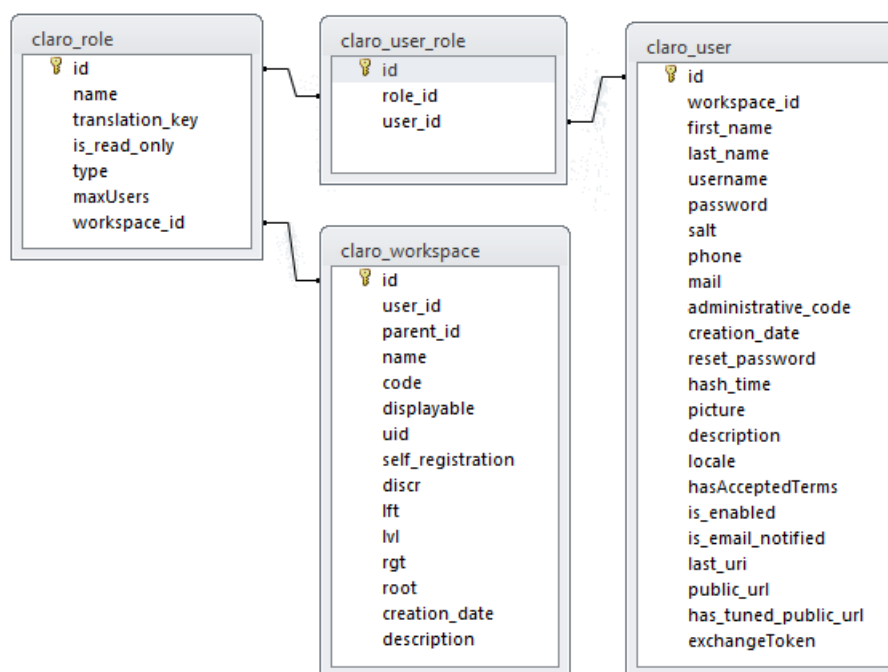


FIGURE 6.7 – Schéma d'un rôle dans la base de données

6.4 Récapitulatif

A la fin de l'étape de création, nous avons une archive contenant :

1. Un fichier XML divisé en plusieurs sections :
 - Une section `Description` contenant les métadonnées¹¹ du manifeste.
 - Des sections `Workspace` dont les attributs contiennent les informations nécessaires pour retrouver ou créer les espaces d'activités. Au sein de ces sections se trouvent également les éléments à synchroniser du `Workspace` :
 - Des éléments `Ressources`
 - Des éléments de descriptions des `Forums`

11. cfr. Lexique, CHAPITRE 2.2 page 16

2. Un dossier data contenant les éventuelles ressources Fichier à transférer.

Tous les éléments de cette archive ne concernent que les ressources et espaces d'activités modifiés ou créés depuis la date de dernière synchronisation de l'utilisateur, date obtenue par l'objet `user_sync` créé par nos soins.

6.5 Ajout d'une nouvelle ressource

Notre bundle ne gérant pas l'entièreté des types de ressources existant et le nombre de ceux-ci étant amené à augmenter, cette section est ici à titre d'exemple supplémentaire pour quiconque souhaiterait ajouter un type de ressource parmi ceux gérées par *OfflineBundle*.

Ajouter un nouveau type ressource est relativement simple. Dans un premier temps, il faut créer une classe PHP. Chez nous les ressources *Offline* sont enregistrées dans le dossier `Claroline/OfflineBundle/Model/Resource`. Idéalement cette classe devrait se nommer *Offline* suivi du type de la ressource (par exemple : `OfflineWiki.php`). Une fois ce fichier créé, il faut s'assurer de plusieurs choses.

Premièrement, afin que cette classe soit un *Service*, il faut faire usage de la librairie : `JMS\DiExtraBundle\Annotation`. Cette dernière permet d'utiliser les annotations au sein de la classe ce qui facilitera grandement le travail. Les annotations permettent d'ajouter des repères pour la compilation. Dans notre cas, il faut ajouter *Service* (`"claroline_offline.offline.NOM_DU_SERVICE"`), (par exemple : *Service* (`"claroline_offline.offline.wiki"`)) avant l'en-tête de déclaration de classe.

Deuxièmement, le *Service* doit étendre la classe abstraite *OfflineResource*. Etendre cette classe va permettre d'utiliser les fonctions et arguments communs aux autres *Services*. Pour que l'implémentation soit valide, il faut également redéfinir cinq méthodes nécessaires pour la synchronisation :

getType() : Cette méthode doit renvoyer le type de la ressource sous forme de string (par exemple `'text'` pour les ressources de type *Text* ou `'claroline_forum'` pour les ressources de type *Forum*). La liste des noms des différents types de ressources se trouve en base de données dans la table `claro_resource_type`. Cette fonction sera utilisée afin d'indexer le *Service* et d'y faire référence lors de la gestion de l'archive, comme expliqué précédemment [6.2.1 page 35](#). Par exemple, on peut voir que le type des ressources *Wiki* est `'icap_wiki'`.

```
// Return the type of resource supported by this service
public function getType(){
    return 'example';
}
```

addResourceToManifest() : Cette méthode va être appelée lors de la phase de création de l'archive. Il faut y indiquer les éléments qui sont intéressants et nécessaires pour de la synchronisation. Idéalement cette méthode devrait faire appel à la méthode `addNodeToManifest()` de la classe `OfflineResource`, cette dernière se charge déjà d'ajouter les informations nécessaires à la synchronisation du `resource_node` associé à la ressource. Cette étape demande un certain travail d'investigation puisqu'il faut savoir quelles sont les informations dites importantes. Dans le cas d'une ressource Wiki, il nous faut aller voir le code du bundle traitant cette ressource et analyser les entités persistées.

Nous pouvons voir qu'une entité Wiki est composée de plusieurs entités Sections ; ces Sections étant elles-même complétées par des Contributions. Pour que ces ressources Wiki soient gérées par *OfflineBundle*, il faudrait enregistrer dans le manifeste les informations nécessaire pour reconstruire les entités précédemment citées. En plus des trois entités spécifiques aux Wiki, une ressource node est créé de manière similaire à toutes les autres ressources de Claroline.

```
public function addResourceToManifest($domManifest,
    $domWorkspace, ResourceNode $resToAdd,
    ZipArchive $archive, $date)
{
    parent::addNodeToManifest($domManifest,
        $this->getType(), $domWorkspace, $resToAdd);
    /*
     *   Ajouter ici les informations supplémentaires
     *   nécessaires a la bonne gestion
     *   de la ressource par la suite.
     */
    return $domManifest;
}
```

createResource() : Cette méthode va être appelée durant la phase de chargement de l'archive¹² s'il s'avère que la ressource a besoin d'être créée. Il faut initialiser ici les champs nécessaires à la création de la ressource désirée et appeler la méthode `create()` du `ResourceManager` qui se chargera de créer la ressource. La méthode `createResource()` renvoie un objet de type `SyncInfo`, qui est une structure de données que nous avons créée et qui sera expliquée dans le chapitre dédié au chargement de l'archive¹³.

12. cfr. CHAPITRE 8.4 page 68

13. cfr. CHAPITRE 8.4 page 68

```
public function createResource($resource ,
    Workspace $workspace , User $user ,
    SyncInfo $wsInfo , $path)
{
    /*
     * Ajouter ici les operations necessaires
     * a la re-creation de la ressource.
     */
    return $wsInfo;
}
```

updateResource() : Cette méthode va être appelée durant la phase de chargement de l'archive s'il s'avère que la ressource existe déjà et doit donc être modifiée. Il faut ici gérer une éventuelle mise-à-jour de la ressource. Cette méthode est également susceptible d'appeler la méthode **createDoublon()** expliquée ci-après.

```
public function updateResource($resource ,
    ResourceNode $node , Workspace $workspace ,
    User $user , SyncInfo $wsInfo , $path)
{
    /*
     * Ajouter ici les operations necessaires
     * a la mise-a-jour de la ressource.
     */
    return $wsInfo;
}
```

createDoublon() : Cette méthode est appelée lorsqu'une mise-à-jour d'une ressource est nécessaire mais que cette ressource a également été modifiée sur l'autre plateforme. Cette méthode peut servir à réaliser un compromis. Par exemple, pour nos ressources de type **Text** nous créons une nouvelle ressource et lui ajoutons un tag '@offline' pour les différencier. Ainsi sur la plateforme en ligne, il est aisé de faire la différence entre la ressource modifiée en ligne et celle modifiée hors-ligne.

```
public function createDoublon($resource ,
    Workspace $workspace , ResourceNode $node , $path)
{
    /*
     * Rajouter ici les operations necessaires
     * a creation d'un doublon pour la ressource
     * si necessaire.
     */
}
```

```
*/  
return ;  
}
```

Finalement, il faut écrire le constructeur de ce Service afin d'initialiser tous les attributs déclarés par sa classe mère et tous les attributs spécifiques dont il aurait besoin. Les attributs indispensables au bon fonctionnement du Service et déclarés dans la classe mère sont les suivants :

ObjectManager om : Cet objet permet au Service d'accéder aux différents dépôts (Repository).

Un dépôt permet d'effectuer des requêtes sur un type particulier d'objet. Il existe un dépôt par entité (User, ResourceNode, Workspace, ...)

ResourceManager resourceManager : Cet objet va permettre au Service d'accéder aux différentes fonctions applicables aux ressources (comme la fonction de création par exemple)

UserManager userManager : Cet objet va permettre au Service d'accéder aux différentes fonctions applicables aux utilisateurs.

EntityManager em : Cet objet n'est utilisé qu'afin de mettre à jour les différentes dates de création et modification. Son utilité sera expliquée dans le chapitre dédié au chargement ((cfr. [CHAPITRE 8.4 page 68](#))

Une fois ces étapes effectuées, nous avons donc un service capable de gérer une nouvelle ressource, dans notre cas les Wiki. Une dernière étape à effectuer est d'ajouter notre tag à ce service. Comme expliqué précédemment, ce tag sera repéré lors de passe de compilation et le service sera alors accessible dans notre `CreationManager` et notre `LoadingManager`. Ceux-ci vont prendre automatiquement connaissance du nouveau type de ressource à gérer. Il est en effet peu intéressant de s'occuper des ressources d'un type non-pris en compte par notre Bundle.

Vous trouverez un exemple complet de ce à quoi doit ressembler un service désireux d'être pris en compte par notre Bundle en Annexe [A page III](#).

Transfert de l'archive

Dans ce chapitre nous aborderons en détail le fonctionnement du transfert de données entre les deux plateformes. Les objectifs de cette tâche étaient donc de pouvoir assurer qu'une archive créée au sein de l'application Claroffline ou du serveur distant soit transférée de manière intégrale, complète et sans erreur à l'autre plateforme. Au sein du processus global détaillé précédemment, nous avons vu que le transfert est utilisé principalement deux fois. La première fois pour charger sur le serveur distant l'archive créée sur l'application Claroffline. Et la seconde fois, pour télécharger l'archive de synchronisation du serveur afin de l'enregistrer sur Claroffline.

Ce problème est complexe car le transfert doit être autant que possible résistant aux coupures et utilisable sur des connexions à faible bande passante. De plus, celui-ci doit être effectué dans la mesure du possible de manière sécurisée. L'implémentation du transfert est principalement réalisée dans deux classes PHP : `SynchronisationController` contenant l'implémentation des routes que nous contactons et `TransferManager` contenant les fonctions utilisées pour les routines d'exécution.

7.1 TransferManager

7.1.1 Des requêtes POST

Une requête POST est un des types de requêtes possible du protocole HTTP. Une requête POST est utilisée pour faire en sorte qu'un serveur web accepte des données incluses dans le corps de la requête et destinées à être stockées. Elle est principalement utilisée pour charger un fichier ou soumettre un formulaire. Dans notre cas, nous nous en sommes servis pour charger l'archive que nous avons construite dans l'étape de création.

7.1.2 Buzz et JSON

Les outils utilisés pour réaliser la communication entre les deux serveurs sont *Buzz* et *JSON*. *Buzz* est un client HTTP distribué sous la forme d'un bundle de *Symfony*. Désireux de faire un travail s'intégrant facilement à la plateforme, nous avons utilisé *Buzz* car les développeurs de Claroline s'en servent déjà. De plus, celui-ci remplissait parfaitement nos attentes. *JSON* quant à lui est utilisé pour structurer le contenu de la requête.

Nous utilisons *Buzz* afin de créer les requêtes. Pour cela, nous créons un client `Curl` afin d'obtenir un objet `Browser` sur lequel nous pouvons exécuter nos requêtes POST. Ces

dernières nécessitent une route à contacter et le contenu à transférer[8]. *Buzz* impose que le contenu de la requête soit formaté en string, c'est pourquoi nous utilisons la fonction `json_encode()`. Cette fonction est un outil très pratique car elle permet de transformer un tableau PHP en un string structuré qui pourra aisément être décodé à l'aide de la fonction `json_decode()` par la suite.

Lors de l'envoi nous construisons un tableau contenant l'ensemble des métadonnées¹ ainsi que l'archive qui est envoyée à l'autre plateforme. C'est donc ce tableau qui est encodé en JSON et décodé à la réception. Un champ du tableau que nous envoyons contient l'archive sous forme de données binaires. Afin de ne pas l'interpréter, il faut convertir cette séquence binaire en texte.

Pour palier à ce problème nous utilisons la fonction `base64_encode()` à l'envoi et `base64_decode()` à la réception. L'objet de la fonction `base64` est précisément de transformer du contenu binaire en un encodage 8bits afin qu'il soit supporté dans divers type d'applications telles que les emails ou l'échange de données sur Internet [20].

7.2 SynchronisationController

7.2.1 Les routes

Comme nous venons de l'expliquer, l'ensemble des échanges d'informations se fait au travers de routes. Ces dernières sont implémentées au sein de `SynchronisationController` car nous avons choisi de laisser l'accès libre aux routes de ce contrôleur. Dans le cas contraire, nous n'aurions pas été à même de contacter les routes de ce contrôleur sans être connecté sur le serveur distant. Pour garantir la sécurité, nous avons mis en place un système d'authentification avec les requêtes. Ce système d'authentification est expliqué dans la section 7.4 de ce chapitre.

L'ensemble des échanges effectués entre les deux plateformes l'est toujours à la demande du client. Les données devant être transférées de l'application *Claroffline* vers le serveur distant sont transmises au sein du contenu de la requête. Tandis que les données transférées depuis le serveur distant vers l'application *Claroffline* arrivent sur l'application *Claroffline* par l'intermédiaire de la réponse à la requête HTTP. L'échange se fait à la demande du client car l'application *Claroffline* ne dispose pas d'une adresse sur laquelle le serveur en ligne peut la contacter.

Voici la liste exhaustive des routes que nous avons implémentées dans `SynchronisationController` suivie pour chacune d'entre elles d'une brève description.

`claro_sync_upload_zip` : Cette route est contactée pour charger une archive de synchronisation sur le serveur distant.

1. Voir lexique chapitre 2.2 page 16

claro_sync_get_zip : Cette route est contactée pour télécharger une archive de synchronisation depuis le serveur distant vers l'application Claroffline.

claro_sync_unlink : Cette route permet de supprimer une archive présente sur le serveur distant. Elle est utile en fin de synchronisation pour effacer les fichiers devenus inutiles.

claro_sync_user : Cette route permet de récupérer les informations utiles à la création d'un utilisateur déconnecté.

claro_sync_last_uploaded : Cette route sert à la reprise d'un processus de synchronisation, elle permet de connaître le dernier fragment qui a été chargé en ligne. La découpe en multiples fragments est expliquée dans la section 7.3 de ce chapitre.

claro_sync_number_of_packets_to_download : Cette route permet à l'application Claroffline de connaître le nombre de paquets à télécharger. Elle est également utilisée pour l'envoi d'une archive en de multiples fragments.

claro_sync_config : Cette route est celle utilisée pour la configuration d'un profil utilisateur.

7.2.2 Mécaniques de tests

La communication entre deux plateformes n'est pas une action habituelle, il a donc fallu réfléchir à la manière dont nous allions pouvoir tester son fonctionnement.

Client REST

Afin de tester les routes que nous avons créées dans `SynchronisationController`, nous avons utilisé un add-on pour le navigateur Google Chrome (*Simple REST Client*²) nous permettant de formuler des requêtes POST [5]. En effet, à la différence des requêtes GET qui sont elles, destinées à aller récupérer des informations, les requêtes POST ne peuvent être générées directement par le navigateur Internet.

Environnement de développement

Afin de tester le transfert entre deux serveurs, nous avons dupliqué la plateforme. Le serveur web de développement sur notre ordinateur personnel comportait donc deux Claroline. Appelons *Claroline 1* la plateforme représentant le serveur distant et *Claroline 2* la plateforme connectée. *Claroline 1* possédait l'adresse du serveur local à contacter pour atteindre *Claroline 2*. Procéder comme tel nous a permis de constater facilement le bon fonctionnement du processus puisqu'il suffisait de vérifier si les résultats étaient ceux escomptés (création et mise à jour des ressources).

Nous avons alors pu faire des tests plus complexes, en exécutant les actions de *Claroline 1* qui contactent *Claroline 2* et observer le résultat. Pour s'assurer du bon fonctionnement et de l'indépendance des deux plateformes, nous avons également fait des tests en plaçant

2. Télécharger Simple REST Client - <https://chrome.google.com/webstore/detail/simple-rest-client/fhjcajmcbmldlhcmfajhfbgofnpcjmb/reviews>

chacune des plateformes sur des serveurs web différents. Dans notre cas nous avons utilisé *IIS* et *XAMPP*. Cette séparation a été possible car nous avons démarré les serveurs sur des ports différents.

7.3 Division en multiples fragments

Pour que le transfert soit fonctionnel sur une connexion Internet de faible débit, il nous a fallu diviser l'archive en plus petits éléments que nous appelons fragments. De cette manière, un minimum de données sont perdues en cas de coupure de la connexion, ce qui signifie également qu'il ne faudra réenvoyer qu'une petite quantité de données. Sans cette découpe de l'archive en plusieurs fragments, l'ensemble de l'archive de synchronisation devrait être retransférée en cas d'échec. De plus l'envoi s'effectuerait en un seul bloc, ce qui peut poser problème lorsqu'elle est volumineuse ou que la connexion est très instable.

Nous avons fixé la taille de ces fragments à 512Ko. Cette valeur a été déterminée au travers de quelques tests de performance que nous décrivons ci-dessous. Si l'administrateur souhaite adapter celle-ci car elle ne correspond pas à sa situation ou qu'il rencontre des problèmes de performance, il suffit de changer la valeur du paramètre `claroline.synchronisation.frag_size` du fichier `services.yml`.

7.3.1 Détermination de la taille des fragments

Afin de déterminer la taille idéale des fragments nous avons effectué un échange répété d'une archive de synchronisation entre deux plateformes Claroline. Les variables de cet échange furent la taille de l'archive transportée et la taille des fragments créés par la plateforme. Pour ce test les deux plateformes Claroline étaient installées localement sur nos ordinateurs de développement, les temps calculés sont donc essentiellement fonction de la performance de nos ordinateurs de travail et non de la qualité de la transmission entre les deux plateformes Claroline, celle-ci étant pratiquement parfaite.

Les conditions du tests étaient les suivantes : nous avons transféré dix fois chaque archive et pris la moyenne du temps de transfert. Le temps de transfert fut calculé comme étant la différence entre le temps précédent le début du transfert et le moment où la confirmation de réception du dernier fragment est reçue. Nos cinq archives transférées avaient approximativement les tailles suivantes : ~1Mo (856Ko), ~5Mo (4581Ko), ~10Mo (10169Ko), ~25Mo (25072Ko) et ~50Mo (49288Ko). La valeur entre parenthèses correspond à la taille exacte des archives que nous avons utilisées pour nos tests. La taille des fragments a, quant à elle, eu les quatre valeurs suivantes : 128Ko, 256Ko, 512Ko et 1024Ko.

La [FIGURE 7.1](#) et la [FIGURE 7.2](#) sont deux illustrations des résultats, que nous avons obtenus. Avant d'analyser ces résultats il est important de rappeler que le temps de transfert est directement proportionnel aux performances de l'ordinateur ayant servi pour les tests. Ce

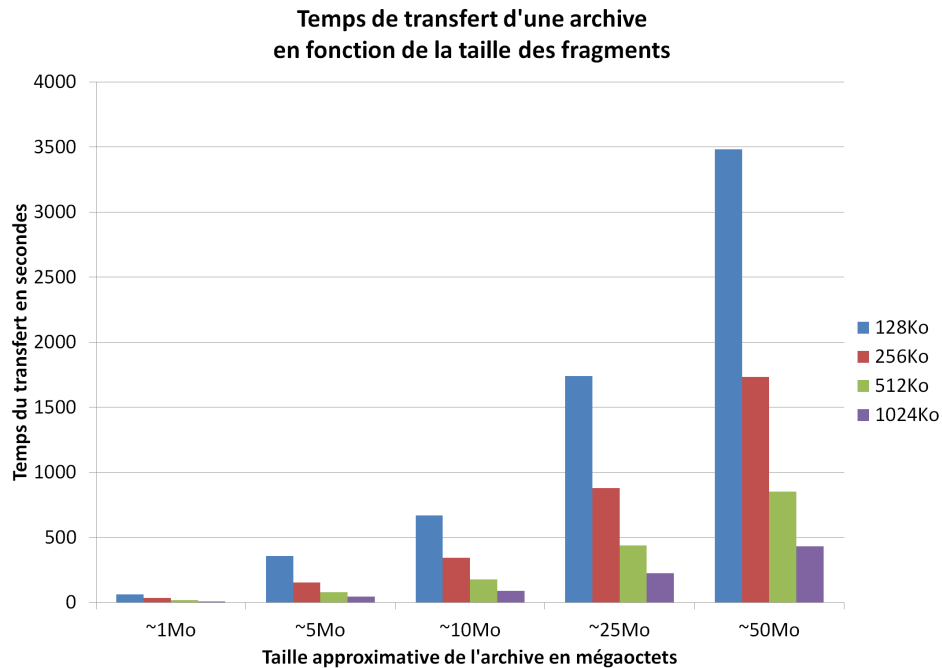


FIGURE 7.1 – Graphique du temps de transfert d'archives de synchronisation en fonction de la taille des fragments

qu'il est intéressant d'observer sur ces graphes est le gain relatif que l'on obtient en faisant varier la taille des fragments pour une archive de taille donnée et non le temps de transfert comme valeur absolue.

Les résultats obtenus par notre méthode de test sont cohérents. En effet, en diminuant significativement la taille des fragments créés par notre système de transfert, le nombre de requêtes envoyées à la seconde plateforme va croître proportionnellement. La FIGURE 7.1 illustre clairement l'augmentation du temps nécessaire pour le transfert d'une archive de plus grande taille et le gain que peut apporter une découpe en fragments plus grands. En effet, nous pouvons voir que passer de fragments de 256Ko à 512Ko divise le temps de transfert d'une archive par deux. Ce gain est d'autant plus intéressant que le temps de transfert requis est élevé, autrement dit que la taille de l'archive augmente.

Comme nous pouvons l'observer sur la FIGURE 7.2, le temps de transfert diminue de manière exponentielle lorsque l'on augmente la taille des fragments. Toutefois il est important de nuancer cette affirmation car même si elle n'est pas présente sur notre graphe, il doit exister une limite supérieure. En effet, si la taille maximale est augmentée de manière trop importante, des erreurs de transfert surviendront. Il faudra dès lors réenvoyer les fragments et le temps total augmentera. Afin d'être complet, le TABLEAU 7.1 contient l'ensemble des

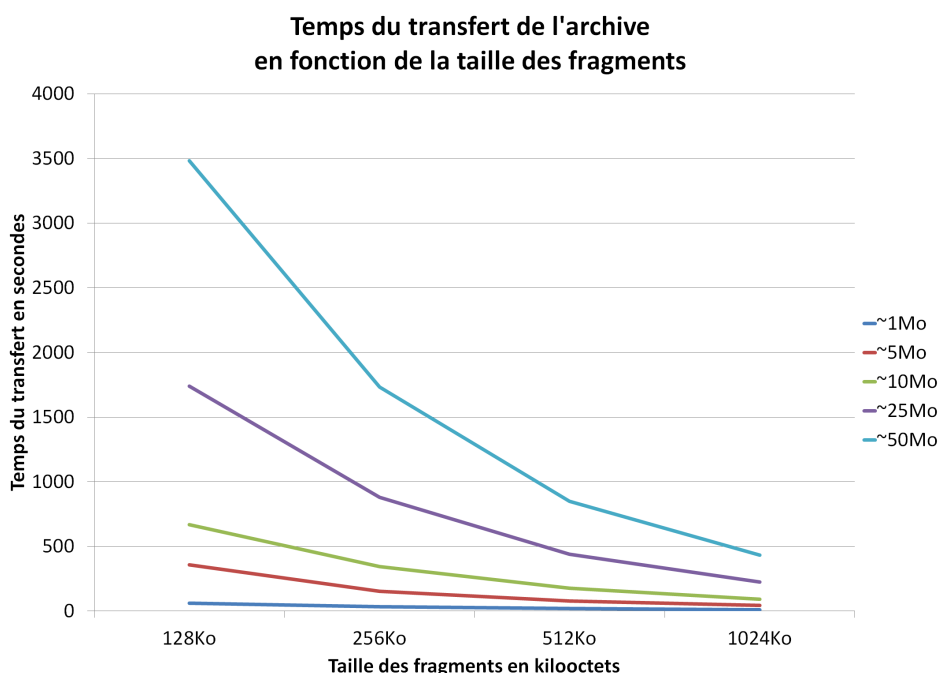


FIGURE 7.2 – Graphique du temps de transfert d'archives de synchronisation en fonction de la taille des fragments

résultats que nous avons obtenus. Il montre pour chaque taille de fragments et pour chaque taille d'archive de synchronisation le temps de transfert ayant été mesuré sur l'ordinateur de test.

Finalement, pour établir le choix de la taille par défaut des fragments au sein du *OfflineBundle*, nous sommes partis de l'hypothèse que l'utilisateur du bundle se trouve dans un pays à faible connectivité et dont la bande passante est limitée. Mettons le débit de la connexion internet comme allant 56Ko/s soit une connexion que l'on pourrait qualifier de mauvaise. De plus, la taille maximale par défaut des fichiers dans Claroline étant fixée à 2Mo, nous avons posé comme hypothèse que l'archive de synchronisation devrait contenir moins de 25Mo de données. Cette hypothèse nous semble englober la plupart des cas. Sur base de nos mesures, nous avons réglé la taille du fragment par défaut à 512Ko.

Partant des hypothèses que nous venons de décrire, le choix de fragments à 512Ko signifie qu'un fragment serait transféré en moins de 10 secondes sur une ligne à faible débit (56Ko/s). Cette situation semble acceptable dans la mesure où nous fixons le timeout de la requête Curl effectuée avec Buzz à 60 secondes. De plus, les fragments de 512Ko montrent des performances acceptables pour les archives de moins de 25Mo dans nos tests. Enfin, ce choix nous semble être l'équilibre idéal entre performances et stabilité.

Taille des fragments	Taille de l'archive de synchronisation				
Ko	~1Mo	~5Mo	~10Mo	~25Mo	~50Mo
128	61s	356s	669s	1740s	3481s
256	33s	153s	342s	880s	1734s
512	18s	79s	176s	438s	850s
1024	9s	45s	90s	224s	432s

TABLE 7.1 – Résultat des tests de transfert d'archives de synchronisation en faisant varier la taille de l'archive et la taille des fragments

7.3.2 Modifications dans l'implémentation

Pour permettre l'envoi d'une archive en plusieurs fragments des fonctions spécifiques ont été créées dans le `TransferManager`. La fonction, `getTotalFragments()`, permet de déterminer le nombre de fragments à transférer pour une archive donnée. La fonction `getFragment()` retourne quant à elle le nième fragment demandé. Ces informations permettent de compléter les métadonnées envoyées avec une requête. La plateforme réceptionnant la requête connaîtra maintenant le numéro du fragment par rapport au total de fragments à transférer.

Une fonction est également responsable d'enregistrer le fragment reçu sur le disque dur. Lorsque l'ensemble des fragments a été transféré, une fonction se charge de les rassembler afin de reconstruire l'archive initiale. Pour en vérifier l'intégrité, nous utilisons une fonction de hashage établie sur le fichier initial. Cette information est enregistrée dans les métadonnées transmises à la plateforme. Cette dernière pourra donc comparer le résultat de sa propre fonction de hashage sur le fichier reconstitué avec celui figurant dans les métadonnées. Si le hash est identique entre le fichier d'origine et le fichier reconstitué, nous concluons que ces deux-ci ont été correctement transférés. Dans le cas contraire une erreur est retournée à la plateforme déconnectée, ce qui entraînera le redémarrage du transfert.

En cas de redémarrage du processus de transfert après une interruption volontaire ou non, le dernier paquet transféré peut être retrouvé. Cette information n'est pas enregistrée dans la base de données contrairement au status du processus global, mais nous pouvons la déterminer en regardant au sein des fichiers dans le répertoire où sont stockées temporairement les données. Pour ce faire, la fonction `getDownloadStop()` du `SynchronisationManager` établit le dernier fragment transféré sur base du nom du fichier et du contenu du dossier temporaire. Sachant que chaque fragment enregistré possède un nom de fichier construit sur

base du nom de l'archive concaténée avec son numéro de fragment.

7.3.3 Stockage temporaire

Afin de stocker les paquets transmis lors d'un échange, nous avons fixé deux sous-répertoires du dossier `claroline.param.files_directory:`. Le premier sert à stocker les archives qui vont être envoyées au serveur distant et le deuxième à stocker l'archive téléchargée. Le chemin d'accès de ces deux répertoires sont paramétrables, est fixé au sein du fichier de paramètre `services.yml`. Le répertoire `DOWN` contient l'archive créée par la plateforme et le répertoire `UP` contient celle qui est téléchargée depuis l'autre plateforme. Ces deux dossiers sont également divisés en sous dossier dont le nom est l'id de l'utilisateur.

7.4 Authentification

Afin de garantir la sécurité de notre transfert nous avons mis en place un système d'authentification sur les requêtes pour pallier à l'absence de limitation d'accès sur les routes utilisées. Cette authentification est réalisée à l'aide de notre classe `OfflineAuthenticator`. Nous avons écrit une condition ternaire, si l'utilisateur est authentifié alors l'action continue, sinon la requête échoue avec le status 401.

Pour authentifier l'utilisateur, il faut utiliser des informations qui lui sont personnelles ; il est courant de transmettre le nom d'utilisateur et le mot de passe pour se connecter à une plateforme. Dans notre cas, nous avons fait le choix d'utiliser un token d'identification qui est unique à chaque utilisateur. Ce token est un champ (`exchangeToken`) que nous avons ajouté à l'entité `User` et qui est créé par la fonction suivante : `hash("sha256", $this->username.time().rand())`. Le hash est établi comme unique et est créé sur base du nom d'utilisateur, de la date à laquelle il est créé ainsi que d'un facteur aléatoire.

Nous avons choisi d'utiliser un token pour s'identifier lors de l'échange plutôt qu'une combinaison habituelle nom d'utilisateur et mot de passe car celle-ci présentait selon nous des défauts dans le contexte de *OfflineBundle*. Premièrement, la méthode d'authentification utilisée requiert que nous lui transmettions un mot de passe en clair ; c'est à dire un mot de passe qui ne soit pas encrypté. Or, au sein de la base de données, le mot de passe n'est pas enregistré comme tel et est bien entendu crypté. Deux options s'offraient alors à nous : soit nous demandions le mot de passe à l'utilisateur à chaque demande de synchronisation, ce que nous jugions inutile. Soit nous aurions du effectuer une comparaison entre les deux mots de passe cryptés.

Cette deuxième option pose également un problème, le mot de passe crypté résulte du choix de l'utilisateur et d'un champ unique appelé `salt` généré aléatoirement à la création de l'utilisateur. Ce qui signifie que pour un même mot de passe choisi par un utilisateur, le mot de passe enregistré de manière cryptée dans la base de donnée est différent. En utilisant le mot de passe crypté, nous nous confrontons à des problèmes de synchronisation de mots de

se passe entre le serveur distant et l'application Claroffline. De plus, notre *OfflineBundle* étant focalisé sur l'apprentissage au travers de Claroline, se concentrer sur la synchronisation des ressources nous a semblé plus important que les paramètres utilisateurs tels que la mise à jour du mot de passe.

Enfin, utiliser un token c'est utiliser une donnée moins sensible que le mot de passe sachant que l'usage du token est limité aux synchronisations. Envoyer le mot de passe en clair ou en crypté n'aurait pas été souhaitable si la plateforme à contacter ne dispose pas d'un certificat lui permettant d'établir une connexion sécurisée HTTPS.

7.4.1 Implémentation

L'authentification dans Claroline utilise le système de Security Context de *Symfony*. Le *SecurityContext* est la classe de *Symfony* qui se charge de protéger l'accès aux routes et aux différents services. Lors de la connexion d'un utilisateur un *Token* est ajouté au *SecurityContext* afin d'identifier l'utilisateur jusqu'à la fin de sa session. Ce *Token* permet alors au système de droits de fonctionner. Des classes *Voter* se chargent de définir si l'utilisateur identifié dans le *SecurityContext* peut ou non exécuter une action.

Dans Claroline, l'authentification se fait à l'aide du nom d'utilisateur et du mot de passe. Il permettent de créer un *UsernamePasswordToken* qui est alors ajouté dans le *SecurityContext*. Comme nous utilisons un identifiant différent du mot de passe, l'*exchangeToken*, pour connecter l'utilisateur nous avons dû créer un nouvel objet *Token* de *Symfony* pour "remplacer" le *UsernamePasswordToken*. *UserExchangeToken* est notre objet étendant le *Token* de *Symfony* que nous créons sur base de l'attribut *exchangeToken* de l'utilisateur. Ce *UserExchangeToken* est créé et placé dans le *SecurityToken* par la classe *OfflineAuthenticator* qui est appelée lors de l'authentification d'une requête de connexion.

Avec ce *UserExchangeToken*, l'utilisateur disposera pendant des mêmes droits que s'il s'était identifié avec un *UsernamePasswordToken*. Cela signifie que les actions effectuées par une requête de synchronisation vérifieront que l'utilisateur qui se synchronise a bel et bien les droits pour créer les ressources qu'il crée dans les différents espaces d'activités. C'est-à-dire, que si un utilisateur modifie ces droits sur l'application Claroffline et tente de synchroniser des ressources qu'il a créées hors-ligne, la synchronisation échouera car cet utilisateur n'a pas les droits pour créer des ressources sur le serveur distant.

7.5 Gestion des erreurs

Toujours dans l'idée d'implémenter un système de transfert qui soit le plus stable possible, nous utilisons les status des requêtes HTTP³ afin d'obtenir des informations sur la manière

3. La liste complète des codes HTTP peut être consultée sur http://fr.wikipedia.org/wiki/Liste_des_codes_HTTP

dont la requête s'est exécutée. Si l'exécution s'est déroulée comme prévu, le status 200 est retourné. Par contre, si l'authentification a échoué nous retournons le code 401. De cette façon, nous pouvons interpréter le code lors de la réponse retournée au client offline et décider de relancer le transfert ou d'avertir l'utilisateur. C'est la méthode `analyseStatusCode()` du `TransfertManager` qui est chargée de l'analyse des codes d'erreur et de déclencher si nécessaire une exception.

Dans l'éventualité où nous avons une erreur de procédure sur la plateforme en ligne, le code 424 est retourné. Par erreur de procédure, nous entendons une erreur de chargement ou de création. Une des causes pouvant provoquer une erreur de chargement est la présence d'un fragment erroné reçu lors d'un transfert. Dans le cas d'un code 424 une erreur est déclenchée et un message d'avertissement est affiché à l'utilisateur. Un autre type d'erreur que peut rencontrer notre module de transfert est une erreur de timeout déclenchée par *Buzz*. Le timeout signifie que l'hôte n'a pas pu être joint dans le temps imparti. Le transfert du fragment est alors redémarré une fois. Si un second timeout est reçu, l'exception est lancée.

Afin de donner à l'utilisateur la meilleure information possible, nous interceptons au sein des `Controllers` générant la vue du transfert les exceptions qui pourraient être déclenchée par la méthode `analyseStatusCode()`. De cette manière il nous est possible d'afficher un message à l'utilisateur et dans la mesure du possible de lui conseiller une solution. La méthode qui définit le message d'erreur à afficher sur base de l'exception attrapée est `getMessage()` du `TransferManager`.

7.6 Alternatives aux requêtes POST

Pour l'ensemble des modules de ce travail, nous avons exploré plusieurs pistes avant de fixer notre choix sur une solution. C'est également le cas pour le module de transfert.

7.6.1 WebSocket

En consultant le mémoire de Cédric Vanderperren de 2010, une autre idée d'implémentation pour le transfert nous est venue [2]. Cette idée consiste en la création d'un socket entre les deux plateformes. Le socket permet d'ouvrir une connexion persistante entre deux entités jusqu'à ce que l'une d'entre elles décide de mettre fin à l'échange. Cette solution semblait séduisante car elle évite d'ouvrir et de fermer une connexion à chaque envoi de fragments. Toutefois, en discutant avec Stéphane Klein, il est apparu que ce type de procédure en PHP est bloquant pour le serveur qui l'exécute. En d'autres termes, cela signifie que le serveur ne peut effectuer d'autres actions tant qu'un transfert est en cours. Et ceci est gênant car cela pourrait bloquer des utilisateurs du serveur sur lequel la requête est faite.

Pour solutionner ce problème, il est possible de faire tourner un daemon. C'est-à-dire, d'exécuter le processus du transfert sur un thread indépendant. De cette manière le serveur peut traiter le transfert en parallèle des actions des autres utilisateurs. Pour implémenter un

daemon, il faut utiliser l'API PCNTL de PHP. Cependant cette librairie n'existe pas sur Windows, ce qui est problématique pour notre projet car certains de nos utilisateurs travailleront sur cette plateforme. En définitive et compte tenu de ces difficultés, nous avons fait le choix d'implémenter nous-même les fonctionnalités dont nous avons besoin pour notre protocole telles que nous les avons détaillées dans ce chapitre.

7.6.2 Torrent

Une troisième idée que nous avons examinée est l'usage d'un protocole de torrent pour échanger les fichiers. Ce type de protocole est utilisé pour diffuser un même fichier à plusieurs utilisateurs, chacun d'entre eux étant à la fois un émetteur et un récepteur du contenu. L'objectif de cette technique est de répartir la charge réseau sur l'ensemble des participants à l'échange. Dans le cas d'*OfflineBundle*, ce type de protocole semble peu intéressant car notre échange ne concerne que deux entités, un client et un serveur. Aucun avantage ne serait donc tiré de l'aspect réparti et collaboratif de ce protocole. De plus, ce type de protocole est bloqué par certains pare-feu sur certains réseaux, il y aurait donc des restrictions d'usage.

7.7 Amélioration du transfert

Le transfert écrit actuellement chaque petit paquet sur le disque dur. Les performances de ce processus peuvent être améliorée si l'on passe par un RamDisk ou si l'implémentation était adaptée pour utiliser une liste virtuelle. En effet, notre implémentation ne devrait pas poser de problème au niveau de l'ordinateur personnel de l'étudiant mais pourrait être une contrainte sur une infrastructure plus importante car l'écriture sur disque dur est une opération coûteuse en temps.

C'est pourquoi nous conseillons d'utiliser un RamDisk coté serveur afin d'y stocker les données temporaires utilisées par le module de transfert. Un RamDisk est une partition montée dans la mémoire vive (RAM) d'un serveur ou d'un ordinateur. Du fait de sa localisation en mémoire RAM l'accès aux données s'y trouvant est beaucoup plus rapide et les performances en sont grandement améliorée. Toutefois les informations stockées dans ce type de mémoire sont effacées lorsque l'alimentation est coupée. Notre solution a avant tout été pensée dans l'idée d'être résistante, même aux coupures électriques, c'est pourquoi nous écrivons les données temporaire sur le disque dur. Si nous partons du principe que le serveur où est hébergé la plateforme Claroline distante dispose d'une infrastructure matérielle plus conséquente et n'est pas exposé aux risques de coupures électriques, le RamDisk nous semble être une bonne solution pour éviter de multiples accès disque dur coûteux en temps.

Une alternative à notre implémentation consisterait à stocker les fragments reçus dans une liste virtuelle et de n'enregistrer sur disque dur que le fichier global une fois le dernier fragment échangé. Toutefois ce système ne permet de reprendre le transfert là où l'on s'était arrêté s'il vient à être interrompu. Retransférer l'ensemble des données serait requis, ce qui est ici ce que nous essayons d'éviter.

7.7.1 Création d'un RamDisk

Plusieurs systèmes existent pour mettre en place un RamDisk sur un ordinateur. Sur Windows il existe *ImDisk* [30], cette petite application ajoute une option au panneau de configuration qui permet de monter d'une partition dans la RAM. Pour les systèmes Linux il faut monter la partition dans le terminal avec l'option `-t tmpfs`. Il est également possible d'appliquer une taille maximale[32].

Chargement de l'archive

La troisième étape de notre processus de synchronisation est le chargement de l'archive. L'objectif étant ici de charger l'ensemble des modifications rassemblées lors de l'étape de création au sein d'une archive. Ceci comprend la création des nouvelles ressources, la mise à jour de certaines ressources et la gestion des conflits éventuels entraînés par la modification d'une même ressource au sein de deux plateformes. Il s'agit donc de charger le travail fourni sur l'ordinateur personnel vers le serveur distant et réciproquement de télécharger le nouveau contenu disponible sur le serveur distant vers l'ordinateur personnel.

8.1 Gestion des espaces d'activités

Dans un premier temps, le serveur distant charge la section `description` du manifeste, ce qui nous permet entre autres de vérifier sur base de son `exchangeToken` que l'utilisateur effectuant le chargement est bel et bien celui décrit dans le manifeste. Après quoi, nous rassemblons tous les espaces d'activités du manifeste et chargeons au sein du serveur distant les ressources contenues dans ceux-ci.

Nous tentons de trouver l'homologue de ces espaces d'activités au sein de la base de données du serveur distant. Chaque espace d'activités étant rendu unique par son attribut `GUID`, c'est sur base de ce dernier que la recherche est effectuée. Les espaces d'activités n'ayant pas d'homologue dans la base de données vont être créés sur base des informations contenues dans le manifeste. Pour ce faire, nous appelons la méthode `createWorkspace()` de notre `Service OfflineWorkspace`.

Nous nous inspirons de la méthode de création de Claroline pour implémenter la fonction `createWorkspace()` de notre service; à ceci près que la méthode du noyau de Claroline se base sur un formulaire complété par un utilisateur. Afin de réutiliser au mieux la fonction du `WorkspaceManager` de Claroline nous précomplétons un objet formulaire avec les informations que nous avons extraites du manifeste. A l'origine, dans Claroline le `GUID` est généré automatiquement par la fonction de création de l'espace d'activité. Néanmoins dans notre cas cela n'était pas idéal. En effet, il fallait garder la consistance entre la base de données de l'application Claroline et du site en ligne. Ce `GUID` sert de clé unique afin de distinguer les différents espaces d'activités.

Pour résoudre ce problème, nous avons ajouté parmi les informations du Manifest, le `GUID` de l'espace d'activités. Ce `GUID` est ensuite placé dans le formulaire que nous précomplétons. Enfin, nous avons modifié la fonction de création de Claroline afin que celle-ci

vérifie si le formulaire qu'elle reçoit contient ou non un champ GUID avec une valeur différentes de NULL, si c'est le cas alors le GUID n'est pas généré automatiquement et est à la place initialisé à la valeur présente dans le formulaire.

Il était essentiel de fixer le GUID à la bonne valeur à la création car c'est à ce moment que les rôles associés à l'espace d'activités sont créés. Le nom de ces rôles est établi sur base du GUID et sont mis en lecture seule après leur création.

8.1.1 Gestion du créateur

Lors de la création d'un espace d'activités et des ressources qu'il contient, nous avons dû faire face au problème de la gestion du créateur. En effet, chaque entité `resource_node` possède un champ indiquant l'id du créateur de la ressource et il fallait s'assurer de la cohérence de ce champ entre les bases de données de l'application Claroffline et les bases de données du serveur distant. Rappelons que pour chaque ressource et espace d'activités une entrée `resource_node` leur est associée en base de données.

Comme tout élément que nous échangeons lors de la synchronisation nous avons besoin d'un élément unique pour identifier le créateur de la ressource ; cet élément est son `username`. En aucun cas, nous ne pouvons nous baser sur l'id du créateur puisqu'il n'est pas garanti que ce dernier sera le même dans les deux bases de données. Le champ `username` est unique et par conséquent, nous pouvons facilement retrouver l'utilisateur correspondant en base de données. Mais ceci n'est valable que dans un échange de l'application Claroffline vers le serveur distant.

Le problème s'avère plus complexe pour amener une ressource existante depuis le serveur distant vers l'application Claroffline. En effet, le créateur de cette ressource n'est pas forcément présent dans la base de données de la plateforme personnelle. Nous avons trois options :

1. Une première possibilité aurait été de ne pas enregistrer le créateur. Mais cette option n'était pas envisageable dans la mesure où ce champ est requis pour la création d'une ressource.
2. Une seconde possibilité aurait consisté à mettre un utilisateur identique choisi arbitrairement comme créateur de la ressource. Par exemple, l'utilisateur effectuant la synchronisation. Mais il n'était pas raisonnable de mettre cet utilisateur comme créateur car les données auraient été faussées lors de la mise en ligne d'une éventuelle mise-à-jour de cette ressource.
3. La troisième possibilité et celle que nous avons choisie consiste à conserver une référence vers le créateur original de la ressource.

Pour pouvoir établir une référence vers le créateur original de la ressource, nous procédons en deux temps. Dans un premier temps, nous vérifions si le créateur de la ressource que l'on

désire créer est présent ou non dans la base de données de l'application Claroffline. Si ce n'est pas le cas, nous créons un profil fantôme chargé de symboliser ce créateur en base de données. Nous entendons par là un profil d'utilisateur ayant le même nom, prénom, e-mail il et nom d'utilisateur que le créateur original mais, pour des raisons de sécurité, nous générons l'`exchangeToken` et le mot de passe de manière aléatoire.

8.1.2 Gestion des rôles

Une autre chose importante dont nous devons nous assurer est le respect des rôles. Un rôle est un ensemble de permissions ; par exemple un enseignant disposera du rôle de manager pour un espace d'activités défini, il aura donc l'autorisation de modifier l'ensemble des ressources de cet espace ainsi que leurs propriétés. A contrario, un étudiant aura le rôle de collaborateur, il aura donc la possibilité de consulter l'ensemble des ressources d'un espace d'activités mais pas de les modifier. C'est d'ailleurs de cette manière que l'on peut savoir à quel espace d'activités un étudiant est lié, il s'agit de l'ensemble des rôles dont il dispose. Une table en base de données reprend les connexions entre User et Role.

Un utilisateur ayant un statut défini dans un espace d'activités sur le serveur distant doit conserver ce statut sur l'application Claroffline. Etant donné que les rôles sont générés sur base du GUID de l'espace d'activités et que nous faisons en sorte que ce GUID soit le même dans les deux bases de données, nous sommes alors à même d'injecter le rôle dans le Manifest et d'associer celui-ci à l'utilisateur se synchronisant lors de l'étape de chargement. Cette association est implémentée à l'aide de la méthode `associateRole()` du `RoleManager` du *CoreBundle* de Claroline.

8.2 Gestion des ressources

Maintenant que nous avons récupéré un espace d'activités depuis le manifeste, que nous avons établi le créateur et que nous avons restauré le rôle de l'utilisateur, nous devons charger les ressources modifiées de cet espace d'activités présentent dans le manifeste. L'arbre de décision appliqué est illustré à la [FIGURE 8.1](#). Pour chercher dans la base de données le `resource_node` correspondant à celui détaillé dans le manifeste, nous nous appuyons sur la clé unique `hashname`.

Si aucune ressource n'a été trouvée avec cette clé, cela signifie qu'elle est nouvelle et doit être créée. Dans le cas contraire nous allons comparer la date de modification de cette ressource avec la date de synchronisation présente dans le manifeste. Si la date de modification est inférieure à la date de synchronisation passée dans le manifeste, cela signifie donc que cette ressource n'a plus été modifiée depuis la dernière synchronisation. En outre, la ressource est présente dans le manifeste, ce qui signifie qu'elle a été modifiée sur l'application Claroffline. En conséquence, nous remplaçons l'ancienne ressource par celle transmise dans le manifeste.

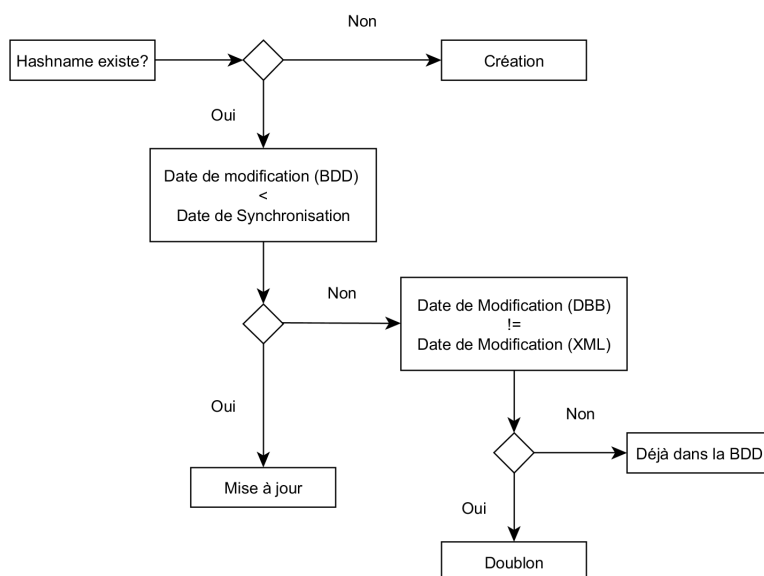


FIGURE 8.1 – Arbre de décision pour le chargement d'une ressource

Arriver dans le dernier cas signifie que la ressource a été modifiée sur les deux plateformes. Nous comparons alors les dates de modifications présentes dans le manifeste et dans la base de données. Si elle sont différentes, nous créons un double de cette ressource. Si elles sont égales, nous considérons que ces ressources sont identiques et nous n'allons pas plus loin.

8.2.1 Création d'une nouvelle ressource

Dans le cas où une ressource nouvelle doit être nouvellement créée, nous nous servons de la méthode de création de nos services taggés. Comme expliqué au chapitre sur la création de l'archive, chaque type de ressource pris en compte par notre bundle dispose d'un service propre. Le `LoadingManager`, tout comme le `CreationManager`, dispose de l'ensemble de ces services indexés par le type de ressource qu'ils gèrent, ainsi il nous est aisé d'appeler la méthode de création de ressources du bon service en se basant sur le type de la ressource que nous traitons. Nous pouvons facilement récupérer le type de la ressource puisqu'il constitue un attribut dans le Manifeste.

La méthode de création de ressources de chacun de ces services récupère les informations au sein du Manifeste et initialise les champs nécessaires à la bonne création de la ressource. Une fois ces champs initialisés, nous créons alors la ressource et la persistons en base de données. Une dernière étape à effectuer est la mise à jour des dates. En effet, il nous semblait important que les ressources dans la base de données disposent de leurs dates de création et de modification originales et non de celle datant de la synchronisation. Pour effectuer cette

mise à jour nous avons du créer une fonction, `ChangeDate()`, mettant à jour les deux dates. Cette mise-à-jour des dates affecte également les Log.

Ces Log sont des traces de l'activité d'un utilisateur et il est donc intéressant de leur donner les valeurs originales encore une fois. Si nous ne procédions pas comme cela, les dates de création, modification et les Log resteraient à la date de synchronisation et cela se symboliseraient par des pics élevés d'activités qui ne seraient pas forcément représentatifs du travail de l'utilisateur.

Outre ces manipulations, certaines ressources nécessitent un travail plus important. Voyons ci-dessous quels sont les détails d'implémentation spécifiques aux ressources. La gestion des Forums étant plus conséquente une partie entière lui est consacrée.

Ressource Texte

Pour les ressources du type Texte, il nous a également été nécessaire d'effectuer deux autres tâches. La première consiste en la création d'une entité `text_revision` symbolisant le texte riche à proprement parler dans la base de données. La seconde tâche étant l'extraction du contenu de la section CDATA du manifeste afin de récupérer le texte désiré. Nous avons réalisé cette seconde tâche au moyen d'une méthode implémentée par nos soins. Après quoi, il nous reste à initialiser les différents champs requis et à initialiser `text_revision` avec le contenu de la section CDATA que nous venons d'extraire.

Ressource Fichier

Comme expliqué dans la section de la création de l'archive, la particularité des ressources du type Fichier, est d'être lié à un fichier spécifique. C'est pourquoi, il nous faut extraire ce document du dossier data de l'archive et copier celui-ci dans le répertoire destiné aux données de la plateforme de destination. Ce répertoire est paramétré dans le fichier *parameters.yml* du CoreBundle de Caroline.

8.2.2 Mise à jour d'une ressource

Si la méthode de mise-à-jour est appelée cela signifie donc qu'une ressource est présente dans l'archive mais n'a pas été modifiée sur la plateforme exécutant l'opération de chargement. Dans ce cas, nous écrasons le contenu de la ressource obsolète et le remplaçons par celui de la nouvelle ressource. Nous avons procédé comme cela, car il ne nous paraissait pas pertinent de conserver en base de données une ressource qui ne serait plus à jour.

8.2.3 Création de doublon d'une ressource

La création d'un doublon intervient dans l'éventualité où une ressource a été modifiée sur chacune des deux plateformes. Ce que nous appelons création d'un doublon est la création d'un double d'une ressource déjà existante en vue de conserver les deux versions. Afin de créer ce double, nous modifions le `hashname` de la ressource de la base de données et

chargeons la ressource contenue dans le manifeste en lui ajoutant *@offline* à la fin de son nom. Dès lors, la résolution du conflit est rendue visible à l'utilisateur par l'ajout du *@offline* dans le nom de la ressource.

La création d'un doublon n'a pas de gestion spécifique pour les ressources du type texte, forum ou répertoire. En revanche, lorsque nous créons un *doublon* d'une ressource de type fichier, il est également important de gérer les fichiers auxquels ces ressources font référence. Pour la ressource mise à jour depuis le manifeste, il nous suffit d'aller placer ce fichier dans le bon dossier au sein de la plateforme exécutant le chargement. Cependant, il faut au préalable modifier le nom du fichier correspondant à la ressource stockée dans base de données.

En effet, si nous ne modifions pas le nom du fichier sur la plateforme exécutant le chargement, ce dernier sera écrasé par celui qui arrive. De plus, le nom du fichier est basé sur le `hashname` de la `resource_node` lui faisant référence. Or, lorsque nous créons un doublon, nous modifions cette référence. Par conséquent, le nom du fichier doit être adapté.

8.2.4 Création et mise à jour des Forums

Création d'un Forum

De par leur structure spécifique, les ressources de type forum disposent d'une gestion particulière. Celle-ci commence par créer le `resource_node` associé à la ressource forum avant de se préoccuper de son contenu. Pour récupérer aisément le contenu dans le Manifeste, nous avons créé un second type de balise : les balises `<forum ... />`. Nous avons également implémenté une série de fonctions dans notre service taggé traitant les ressources du type Forums (`OfflineForum`). Ces fonctions vérifient l'existence du contenu de ce forum, qu'il s'agisse d'une catégorie, d'un sujet ou d'un message. De manière similaire aux ressources, nous avons dû ajouter un identifiant unique à ces trois différents types d'entités afin de pouvoir conserver la consistance entre les deux bases de données.

Une fois l'existence du contenu testée, nous déterminons si nous le créons directement ou si ce contenu doit être mis-à-jour. C'est le cas, si les dates de modifications figurant dans le Manifeste sont postérieures aux dates de modifications figurant dans la base de données. Des informations supplémentaires sont à noter pour les entités du type message. En effet, leur contenu est enregistré dans des balises du type `CDATA` comme c'est le cas des ressources du type texte. Les messages sont également susceptibles d'être doublonnés à l'instar des entités catégories et sujets. Après discussion avec Stéphane Klein, il nous a paru intéressant de conserver les deux messages dans le cas où le message est modifié sur le serveur par un administrateur et sur l'application Claroffline par l'utilisateur. Les messages créés pendant la synchronisation possèdent un tag indiquant l'heure de leur création originale.

8.2.5 Affichage des opérations

Afin d'avertir l'utilisateur des modifications apportées à son application Claroffline nous avons créé une classe `SyncInfo`. Une instance de `SyncInfo` contient pour chaque espace d'activités : son nom, son code, la liste des ressources qui y ont été créées, la liste des ressources qui y ont été mises à jour et la liste des ressources qui ont été doublonnées. Au fur et à mesure du chargement de l'archive, nous remplissons un tableau d'instances de `SyncInfo` et affichons ce dernier lors de la fin de l'exécution du processus sur l'application Claroffline.

8.3 Cas problématique : la date de modification

Au cours de cette phase de réalisation, nous nous servions abondamment de la date de modification des ressources afin de savoir si celles-ci avaient été modifiées depuis notre dernière synchronisation par exemple. Malheureusement, nous avons rencontré un bug récurrent avec ces dates. Ce bug provenait d'un 'Listener'¹ chargé de mettre automatiquement à jour la date de modification des ressources dès qu'un champ de ces dernières était modifié.

Le problème étant qu'à la création de nouvelles ressources, le champ `textttnext_id` de la dernière ressource présente dans la base de données était automatiquement mis à jour afin de préserver un certain ordre entre les ressources d'un même espace. Or cette mise à jour automatique déclenchait le Listener cité précédemment et ce dernier modifiait alors la date de modification de la ressource, ce qui entraînait souvent des incohérences ou des erreurs.

Exemple 1

Prenons le cas de deux plateformes, une en ligne (le serveur distant) l'autre hors-ligne (l'ordinateur personnel d'un utilisateur), avec un espace d'activités et dans cet espace 2 ressources : R1 et R2.

Imaginons que, côté 'en ligne' un utilisateur modifie R1 et que, côté 'hors-ligne', l'utilisateur modifie les deux ressources et tente de se synchroniser.

Suivant notre démarche, un Manifest va être créé contenant la description des deux ressources R1 et R2 modifiées par l'utilisateur, en parsant ce Manifest du côté en-ligne, nous voyons que la ressource R1 est déjà présente, car R1 côté 'en ligne' et 'hors-ligne' aura le même `hashname`. On va donc regarder les dates de modifications de ces deux ressources R1, étant donné qu'elles sont différentes (puisque R1 a aussi été modifiée par un utilisateur sur la plateforme en ligne), un doublon va être créé.

Mais, la création de ce doublon va faire une nouvelle entrée dans la base de données et donc mettre à jour le champ `next_id` de R2 et, via le Listener, mettre à jour sa date de modification.

1. cfr Lexique [2.2](#) page 16

Dés lors, lorsque nous arrivons sur la ressource R2 dans le manifest, nous allons voir si elle n'est pas déjà présente en base de données, comme c'est le cas nous comparons leurs deux dates de modifications, comme elles sont différentes nous créons un doublon de R2 alors qu'à la base R2 n'a pas été modifiée en ligne et donc aurait du être remplacée par le R2 du Manifest.

Exemple 2

Une autre situation, moins complexe, est la suivante : soit deux ressources R1 et R2, une plateforme en ligne (le serveur) et une plateforme déconnectée (l'ordinateur personnelle de l'utilisateur). R1 est une ressource présente à la fois en ligne et hors-ligne et R2 une ressource nouvellement créée par l'utilisateur sur sa machine personnelle. Prenons l'hypothèse que R1 n'a été modifiée sur aucune des plateformes.

Lorsque l'utilisateur va lancer sa synchronisation, la ressource R1 ne va pas être prise en compte mais la ressource R2 oui, celle-ci étant nouvelle. R2 va donc être recréée sur la plateforme en ligne et va entraîner une nouvelle entrée dans la base de données et la mise à jour du champ 'next_id' de R1. La date de modification de R1 ayant été modifiée, cette ressource sera prise en compte et ramenée sur la machine personnelle de l'utilisateur alors qu'aucune 'véritable' modification n'y a été apportée.

Solution

Fort heureusement et grâce à Stéphane Klein de chez Claroline, nous avons pu résoudre ce problème. L'idée de base était de désactiver le 'Listener' responsable de ces mises à jour de dates automatiques. Néanmoins, ce dernier étant également utilisé et important pour d'autres parties de la plateforme Claroline, il a été décidé de lui faire adopter le comportement désiré pour notre cas, au lieu de le surcharger. C'est-à-dire ignorer les modifications apportées à la position d'un `resource_node`.

8.4 Cas problématique : parents disparus

Lors de notre analyse, nous avons vu un autre problème qui pourrait surgir.

Pour expliquer cela, une petite mise en contexte s'impose. Dans la base de données, chaque ressource dispose d'un champ lui indiquant sa ressource 'parent', c'est-à-dire la ressource contenant cette ressource. Or un problème susceptible de se produire est le suivant : que se passera-t-il, lors de la synchronisation, si un utilisateur déconnecté crée une ressource au sein d'un répertoire et que ce même répertoire a été supprimé sur le site en ligne ?

Une possibilité était, lorsque nous constatons l'absence d'une ressource 'parent' dans la base de données de la plateforme en ligne, de simplement nier la ressource à créer en partant du postulat que, si ce répertoire a été supprimé, les ressources contenues dans celui-ci

devraient l'être aussi.

Néanmoins, nous avons opté pour une autre solution moins 'punitiv', si lors de la création nous constatons que la ressource 'parent' n'existe plus, nous indiquons alors l'espace d'activités de cette ressource comme étant son parent, ainsi nous nous assurons que la ressource soit toujours créée. Notre solution se base sur le postulat que si la ressource est présente dans le Manifeste c'est qu'elle a été modifiée et, de ce fait, est importante pour l'utilisateur.

8.5 Mécanisme de gestion du hashname

Pour que notre application soit fonctionnelle avec plusieurs administrateurs gérant les cours, nous avons repensé la logique que nous avons autour des `hashname`. Initialement nous pensions pouvoir nous contenter de l'utilisation de la création de doublon et de la mise-à-jour comme expliquée ci-dessus. Malheureusement, dans certains cas cette solution peut poser problèmes.

En effet, si plusieurs utilisateurs manipulent une même ressource et se synchronisent lors de timings différents, la référence vers la ressource originale sur le serveur distant sera modifiée à son tour et les informations en base de données risquent de ne plus être pertinentes.

Pour résoudre ce problème nous avons eu l'idée de créer un `Listener` sur les événements de Doctrine. Lorsqu'un utilisateur de Claroffline met à jour une ressource, un événement est déclenché et capturé par notre `Listener`. Ce dernier vérifie alors si cette mise-à-jour opère sur une ressource, si tel est le cas nous vérifions alors que l'utilisateur courant est le créateur ou non de la dite ressource. Dans la négative, nous générons alors un nouveau `hashname` pour cette ressource.

En procédant comme cela, nous nous assurons d'éviter des conflits importants sur le serveur distant. En effet, si plusieurs utilisateurs ont la possibilité de modifier une ressource et qu'ils se synchronisent, une 'version' propre à chaque utilisateur sera créée. Ainsi aucune de leurs modifications ne sera supprimée et la ressource originale ne verra pas son `hashname` modifié, par conséquent, elle restera la ressource de référence.

Si jamais l'utilisateur opérant une modification depuis Claroffline est le créateur de la ressource, nous gardons alors notre ancienne logique. En effet, nous estimons que si l'utilisateur est le créateur de la ressource, lorsqu'il se synchronise c'est cette nouvelle ressource qui devrait constituer la référence et non l'ancienne. Qui plus est, si l'ancienne ressource disponible sur le serveur distant n'a pas été modifiée (par le créateur ou un admin par exemple), il nous semble logique d'écraser cette ressource obsolète au profit de la nouvelle. De plus, il nous semblait peu pertinent de changer le `hashname`, et donc de créer une nouvelle ressource à la création, si la modification vient du créateur de la ressource lui-même.

Si maintenant le créateur de la ressource vient à modifier cette dernière sur une plateforme avec Claroffline et sur le serveur Distant ou une seconde plateforme. Alors nous gardons également notre ancienne logique de création de doublon. Nous trouvons qu'il peut être intéressant de conserver alors les deux modifications car elles pourraient très bien être pertinentes toutes les deux.

Enfin, comme nous l'avons mentionné, nous n'effectuons cette mise-à-jour du hashname que si elle est effectuée sur une plateforme Claroffline et non sur le serveur distant. En effet, nous pensons que les ressources sur ce dernier constituent la référence pour les utilisateurs, les hashnames ne devraient donc pas être modifiés à la moindre édition de la ressource.

8.5.1 Scénarios

Scénario 1

Soit une ressource R1 et son créateur U1. Avec notre Listener, si U1 met à jour sa ressource sur son application Claroffline et qu'un autre utilisateur (U1 ou un Admin) met cette ressource à jour sur le serveur distant. Un doublon va être conservé afin de conserver les deux modifications. En effet, puisque U1 est le créateur de la ressource, un nouvel hashname ne sera pas généré suite aux modifications apportées par ce dernier sur son application Claroffline. Les deux versions de la ressource R1 auront donc même hashname ce qui va entraîné la création de ce double.

Scénario 2

Soit une ressource R1 et son créateur U1. Si ce dernier met à jour sa ressource sur son application Claroffline et que la ressource n'a pas été modifiée sur le serveur distant, la nouvelle version (celle provenant de l'application Claroffline) va remplacer l'ancienne et servir de nouvelle référence.

Scénario 3

Soit une ressource R1 et son créateur U1, et soit un autre utilisateur U2 ayant les droits nécessaires à la modification de R1. Si U2 modifie R1 sur son application Claroffline, un nouveau hashname va être généré. En effet, même s'il peut modifier la ressource, il n'en est pas le propriétaire. Lors de la synchronisation, la ressource modifiée par U2 va être considérée comme une nouvelle ressource (puisque'elle dispose d'un nouveau hashname) et va être créée sur le serveur distant. Ceci nous semble intéressant dans la mesure où les modifications de U2 peuvent être pertinentes, néanmoins n'étant pas le propriétaire de la ressource il nous semblait plus sécurisé de ne pas écraser cette dernière au profit de celle éditée par U2.

CHAPITRE 9

Installation

L'objectif de cette partie du travail est double. D'une part, il faut permettre à un utilisateur de récupérer sur son ordinateur personnel l'application Claroffline qui lui permettra d'utiliser Claroline sans accès Internet. D'autre part, il faut détailler comment un administrateur d'une plateforme Claroline peut ajouter notre plug-in sur son serveur pour que l'ensemble des utilisateurs de sa plateforme puissent disposer des fonctionnalités.

Nous commencerons ce chapitre par l'explication de la procédure d'ajout du plug-in *OfflineBundle* à un serveur déjà existant. Ensuite, nous détaillerons la manière dont nous avons procédé pour réaliser l'installateur de l'application Claroffline, compte tenu des contraintes qui nous étaient imposées et des différentes options existantes.

9.1 Ajout du plug-in OfflineBundle sur un serveur Claroline

Afin que les utilisateurs d'une plateforme Claroline puissent utiliser l'application Claroffline sur un ordinateur personnel, il faut au préalable que l'administrateur de la plateforme Claroline ajoute le plug-in *OfflineBundle* sur la plateforme sur laquelle les utilisateurs veulent se synchroniser.

Il existe deux façons d'ajouter le plug-in à votre serveur Claroline : soit installer automatiquement le plug-in à l'aide du gestionnaire de packet Composer, soit ajouter manuellement les fichiers dans les sources de Claroline en clonant le dépôt Git de notre projet.

9.1.1 Ajout automatique

L'ajout du plug-in *OfflineBundle* à l'aide de *Composer* se fait pratiquement automatiquement. Pour installer le plug-in de cette façon, vous devez ajouter une dépendance à *OfflineBundle* dans le fichier `composer.json` principal de votre application. Actuellement *OfflineBundle* n'est pas encore disponible sur Packagist, le dépôt principal de *Composer*, il faut donc ajouter l'adresse de notre dépôt dans les sources explorées par *Composer*. Les deux lignes à ajouter sont les suivantes :

Ajout de la dépendance

```
"require": {  
    "php": ">=5.4.1",
```

```

    "sensio/distribution-bundle": "~2.3.4",
    "claroline/bundle-recorder": "~3.0",
    "claroline/core-bundle": "~3.0",
    [...]
    "claroline/offline-bundle": "dev-master"
}

```

Ajout du dépôt

```

"repositories": [
    {
        "type": "vcs",
        "url": "https://github.com/vVYou/OfflineBundle"
    }
],

```

Mise à jour des sources par Composer

Maintenant que nous avons mis les dépendances de Composer à jour il faut exécuter ce programme pour mettre à jour les sources de la plateforme Claroline. Pour ce faire, il faut ouvrir un terminal, se rendre dans le répertoire contenant les sources et entrer la commande : `$ composer update`.

Mise à jour de Claroline

Une fois que les sources ont été mises à jour par *Composer*, vous devez mettre à jour Claroline pour qu'il exécute les migrations de base de données requises. Depuis le répertoire contenant les sources, entrez la commande : `$ php app/console claroline:update`

9.1.2 Ajout manuel

L'ajout manuel du plug-in *OfflineBundle* à votre plateforme Claroline n'est pas recommandé ; toutefois cette variante manuelle est documentée car l'outil Composer peut s'avérer capricieux.

Copie des sources

La première étape consiste à effectuer un clone de dépôt Git où sont disponibles les sources. Le clone du dépôt doit être effectué dans le répertoire précis suivant :

Claroline/vendor/claroline/offline-bundle/Claroline/.

Pour cloner le dépôt entrez la commande :

```
$ git clone https://github.com/vVYou/OfflineBundle.git
```

Vous devriez maintenant avoir l'arborescence suivante

Claroline/vendor/claroline/offline-bundle/Claroline/OfflineBundle/\$sources, il est important que celle-ci soit strictement respectée.

Ajout du bundle dans le loader

Maintenant que les sources font partie du projet, il faut déclarer leur existence à *Composer* afin qu'elles soient chargées avec *Symfony* au démarrage de la plateforme. Cette opération requiert la modification de deux fichiers : `Claroline/app/config/bundles.ini` et `Claroline/vendor/composer/autoload_namespaces.php`.

A la fin du fichier `bundles.ini` vous devez ajouter la ligne suivante à la suite des autres :

```
Claroline\OfflineBundle\ClarolineOfflineBundle = true
```

Dans le fichier `autoload_namespaces.php` vous devez ajouter l'entrée suivante au tableau retourné.

```
'Claroline\OfflineBundle' =>
    array(\ $vendorDir . '/claroline/offline-bundle')}
```

Exécution des migrations et updaters

Enfin, vous devez mettre à jour la base de données en exécutant les migrations puis les updaters. Ceci peut être fait avec une seule commande de *Claroline*. Cette commande est la suivante : `"$ php app/console claroline:test:update 1 2 ClarolineOfflineBundle"`.

9.2 Création d'un installateur

Plusieurs contraintes étaient à prendre en compte lors de la création de l'installateur de l'application *Claroline*. La première d'entre elles est la simplicité pour l'utilisateur. Une des orientations demandées était le fait qu'il soit le plus simple possible pour un utilisateur d'installer *Claroline* muni du *OfflineBundle* sur sa machine. Une autre contrainte est que notre installateur puisse fonctionner sur les systèmes d'exploitation *Windows* et *Linux*.

Comme notre projet est implémenté comme un bundle *Symfony* venant se joindre aux sources *Claroline*, nous avons besoin de pouvoir exécuter le code PHP de *Claroline* sur l'ordinateur personnel de l'utilisateur. Par ailleurs, *Claroline* a besoin d'une base de données pour fonctionner. De plus, l'utilisateur doit avoir la possibilité de démarrer sans disposer de connexion Internet, nous avons donc choisi de doter chaque utilisateur d'un serveur *Apache*. C'est sur celui-ci que nous intégrerons les fichiers et la base de données de *Claroline*.

9.2.1 Choix de la solution

La question suivante qui nous a occupés est de savoir de quelle manière serait réalisée cette intégration des fichiers de *Claroline* sur le serveur *apache* qui équiperait notre installateur. Deux options se sont présentées à nous : générer dynamiquement une plateforme

pour chaque utilisateur souhaitant se synchroniser et personnaliser celle-ci immédiatement en y intégrant ses cours personnels ou créer une version générique de l'installateur qui ne disposerait d'aucun contenu.

Nous choisis la deuxième option, créer une version générique de la plateforme Claroline sans aucun contenu. Nous dirons que cette version de Claroline est vierge, car aucun espace d'activités, ressources ou utilisateurs n'y sont enregistrés. Ce type de diffusion comporte un avantage majeur, il est économe en ressource à plusieurs niveaux. D'une part, il évite de générer une plateforme spécifique pour chaque utilisateur. Et d'autre part il permet de copier cette plateforme installateur d'un ordinateur à un autre sans avoir à le télécharger à chaque fois.

Le fait de pouvoir copier l'installateur localement d'un ordinateur à un autre permet d'éviter de saturer la connexion. Par exemple, dans une salle informatique avec une faible connectivité, si un grand nombre d'utilisateurs souhaitent obtenir la plateforme en même temps, il sera plus efficace de copier le fichier d'un poste de travail à un autre que de le télécharger depuis chaque poste de travail et le serveur distant. Si l'on pousse cette réflexion plus loin, nous pourrions même envisager qu'un chargé de cours ou une administration prépare un ensemble de CDs ou de clés USB qui contiendraient une copie du logiciel ; il n'y aurait ensuite plus qu'à distribuer ces copies matérielles.

Nous avons choisi d'adapter la manière dont nous distribuons le code en fonction du système d'exploitation de l'utilisateur car nous n'avons pas trouvé les mêmes outils pour les deux systèmes. Pour Windows, nos recherches nous ont amenés à trouver un outil appelé Inno Setup, permettant de compacter des fichiers sources et de les assembler en un fichier exécutable. Nous expliquerons donc ci-dessous comment procéder pour créer un installateur de Claroline intégrant tous les éléments requis pour les utilisateurs de Windows.

En revanche, nous n'avons pas trouvé d'outils similaires pour le système Linux. Malgré de multiples essais, nous n'avons pas non plus trouvé d'outils concluants permettant de créer un serveur sur un stockage externe tel qu'une clé USB. Dès lors, nous proposons pour ce système d'exploitation une installation plus générique. Nous fournissons à l'utilisateur les références vers les documentations lui permettant de réaliser l'installation du serveur Apache et MySQL, après quoi il disposera d'un paquet conçu par nos soins contenant les sources de la plateforme et un script générant la base de données et y chargeant les données requises. La création de ce paquet est également expliquée ci-dessous.

9.2.2 Création d'un installateur pour Windows

Avec le code source de ce travail nous fournissons un installateur de Claroline distribuable auprès des utilisateurs Windows. Nous sommes conscients des limites déterminées par notre installateur, il n'a été créé que pour une version déterminée de Claroline et du *OfflineBundle*. C'est pourquoi vous trouverez dans cette section l'ensemble des informations

nécessaires pour pouvoir créer vous même cet installateur.

Télécharger les outils

Pour créer l'installateur, nous devons commencer par réunir les différents composants qui y seront intégrés : un serveur web, une plateforme claroline et un navigateur. Prenez soin dans de vous assurez que le serveur web que vous choisissiez supporte PHP 5.4.1 ou supérieur, en effet, cette version de PHP est requise pour exécuter Claroline. Voici les outils que nous avons choisis, ainsi que les liens permettant de les télécharger :

- XAMPP¹, tel que défini par ses développeurs, est "une distribution Apache entièrement gratuite, facile à installer qui contient MySQL, PHP et Perl. Le package open source XAMPP a été mis en place pour être incroyablement facile à installer et à utiliser". Bien que ce ne soit peut-être pas la plus performante, nous avons choisi de proposer cette distribution du serveur Apache pour notre installateur car cette distribution est proposée pour les plateformes Linux, Windows et Mac. Elle pourrait donc être installée manuellement par des utilisateurs utilisant d'autres plateformes ou utilisée par d'autres développeurs pour compiler d'autres installateurs.
- Claroline², il s'agit du lien pour télécharger les sources. Sachez toute fois que les sources de votre serveur distant suffisent.
- Google Chrome Portable³, l'idée d'intégrer ce navigateur nous permet de nous assurer que l'utilisateur disposera d'un navigateur à jour et performant. De plus, Google Chrome dispose d'un mode permettant de simuler un site web comme étant une application native. Lorsque ce mode est activé, le navigateur s'affiche en plein écran, faisant disparaître la barre des tâches et la barre de navigation. Ce mode nous semble particulièrement intéressant car combiné avec un raccourci sur le bureau, l'utilisateur sera immergé dans l'expérience Claroline et ne sera pas perturbé par l'URL de la plateforme déconnectée⁴. En utilisant le mode offline de Claroline de cette manière, l'utilisateur sera à même de distinguer de manière plus claire son accès au serveur distant en naviguant sur son chemin habituel et l'application Claroffline à laquelle il accedera par le raccourci sur son bureau.
- Inno Setup⁵, est un logiciel open source permettant de créer des installateurs pour Windows. Il permet notamment au développeur qui s'en sert de compresser les sources de son application en un fichier .exe et de créer le wizard d'installation de ce logiciel. C'est à l'aide de ce programme que nous réaliserons notre fichier exécutable de

1. Télécharger Xampp sur <https://www.apachefriends.org/fr/index.html>

2. Télécharger Claroline sur <http://www.claroline.net/>

3. Google Chrome Portable est téléchargeable à l'adresse : http://portableapps.com/apps/internet/google_chrome_portable

4. localhost/Claroline/web/app.php

5. Site Internet d'Inno Setup : <http://www.jrsoftware.org/isinfo.php>

l'application Claroffline.

Assembler les outils

Maintenant que nous avons rassemblé les outils dont nous avons besoin, l'objectif est de les mettre ensemble pour obtenir un dossier contenant notre serveur web muni de Claroline et de Google Chrome. Nous pourrions alors utiliser Inno Setup dans un second temps afin de créer le fichier final.

Installer Xampp

Pour l'installation de Xampp, nous suivons les instructions recommandées, en ce compris le répertoire d'installation C:\xampp car il nous servira de répertoire principal. Une fois l'installation terminée, plusieurs aménagements dans le fichier configuration de PHP doivent être fait. Nous vous invitons donc à ouvrir le fichier `PHP.ini` afin de procéder aux changements suivants :

1. Le champ `memory_limit` doit être supérieure ou égale à 256Mb

```
; Maximum amount of memory a script may consume (128MB)
; http://php.net/memory-limit
memory_limit=256M
```

2. Le champ `max_execution_time` doit être mis à 60 sec

```
;;;;;;;;;;;;;
; Resource Limits ;
;;;;;;;;;;;;;

; Maximum execution time of each script, in seconds
; http://php.net/max-execution-time
; Note: This directive is hardcoded to 0 for the CLI SAPI
max_execution_time=60
```

3. Enfin, pour des questions de performances nous vous conseillons d'activer l'extension `zend_opcache` et désactiver `XDEBUG`

```
[eAccelerator]
zend_extension = "C:\xampp\php\ext\php_eaccelerator_ts.dll"

;[XDebug]
;zend_extension = "C:\xampp\php\ext\php_xdebug.dll"
;xdebug.profiler_append = 0
;xdebug.profiler_enable = 1
```

```
;xdebug.profiler_enable_trigger = 0
;xdebug.profiler_output_dir = "C:\xampp\tmp"
;xdebug.profiler_output_name = "cachegrind.out.%t-%s"
;xdebug.remote_enable = 0
;xdebug.remote_handler = "dbgp"
;xdebug.remote_host = "127.0.0.1"
;xdebug.trace_output_dir = "C:\xampp\tmp"
```

Installer Google Chrome Portable

Une fois Xampp installé, passons à Google Chrome Portable. Pour installer ce navigateur, il nous suffit d'exécuter le fichier que nous venons de télécharger. Le wizard d'installation nous demande alors dans quel répertoire nous souhaitons l'installer. Nous choisirons "C : \xampp\GoogleChromePortable". Attendons la fin de l'exécution de l'installation, nous pouvons alors vérifier que celle-ci s'est correctement déroulée en lançant Google Chrome. Ce dernier doit être stocké à l'emplacement référencé et doit être capable d'afficher le rendu d'une page web de votre choix. Si l'installation a rencontré des problèmes nous vous invitons à vous rendre dans la documentation de Google Chrome⁶.

Installer Claroline

Maintenant que nous avons installé notre navigateur et notre serveur web nous devons y ajouter Claroline. Nous détaillerons ici comment intégrer la plateforme en partant de l'installation de Claroline de base. Si votre plateforme comporte des spécificités qui lui sont propres telle qu'un template ou des extensions particulières, veuillez passer au point suivant.

Pour installer Claroline à partir du WebInstaller fourni par le site <http://www.claroline.net/> il faut extraire le contenu de l'archive contenant les fichiers de Claroline dans le dossier web du serveur; dans le cas de Xampp ce dossier est xampp\htdocs. Ce répertoire est l'équivalent de www pour le serveur Wamp. Il s'agit du répertoire qui sera considéré par le serveur web comme le répertoire contenant les fichiers à interpréter.

À ce niveau nous disposons des fichiers de Claroline disponibles sur un serveur web et accessibles via un navigateur. Toutefois, la plateforme claroline n'est pas encore configurée. Nous allons donc démarrer le serveur Apache et Mysql et nous connecter au site. Pour démarrer le serveur Apache, rendez-vous dans le dossier d'installation de Xampp, exécutez xampp-control.exe (voir Figure 9.1), et cliquez ensuite sur le bouton *start* faisant face à Apache et à MySql (si vous rencontrez des erreurs à cette étape, veuillez vous référer à la documentation disponible en ligne). Si les deux services se sont correctement lancés le bouton start se grise et le bouton stop devient actif; ceci est illustré sur la Figure 9.2.

6. <https://support.google.com/chrome/>

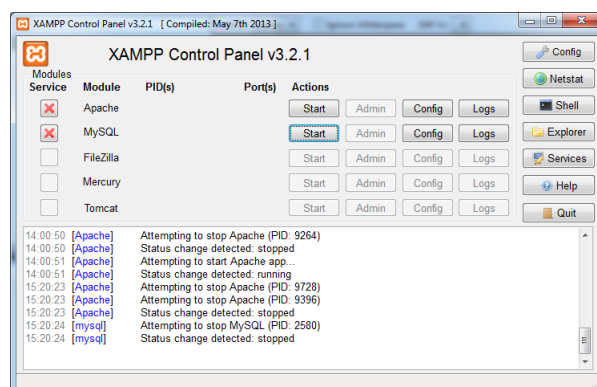


FIGURE 9.1 – Fenêtre Xampp control

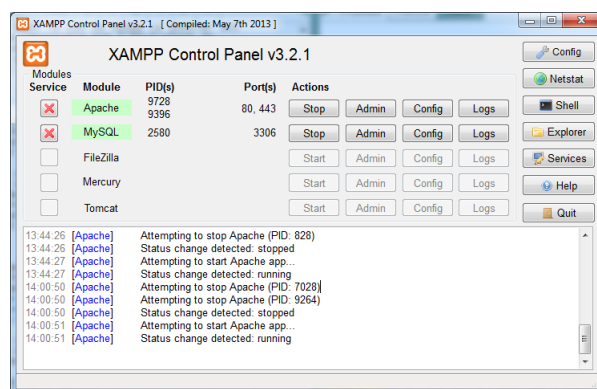


FIGURE 9.2 – Serveur Apache et MySql démarrés sous Xampp

Vous pouvez alors vous connecter sur la page localhost/Claroline/web/install.php pour exécuter l'installation de claroline afin que la plateforme soit configurée.

Vous arrivez alors sur une vue à cinq panneaux vous interrogeant sur la configuration. La première étape vérifie que les pré-requis soient respectés pour l'installation, ce devrait être le cas. La seconde étape vise à créer le profil administrateur de la plateforme, nous vous proposons d'entrer les informations données dans la Table 9.1 sachant que ce profil ne devrait pas être utilisé ultérieurement et ne devrait pas nuire au bon fonctionnement de l'extension offline.

La dernière étape est celle de la configuration du serveur mail et ne nous concerne pas je vous invite donc à cliquer sur skip. Si cette dernière étape s'est déroulée sans erreur nous disposons maintenant de tout ce dont nous avons besoin pour distribuer claroline équipé du bundle offline. Enfin le configurateur de claroline s'exécute. Veuillez maintenant ajouter *OfflineBundle* au sein des plug-in de Claroline tel qu'expliqué en début de chapitre. Après

Prénom	root
Nom	root
Nom d'utilisateur	root
Mot de passe	password
Email	root@claro.net

TABLE 9.1 – Informations pour la création de l'utilisateur administrateur

quoi, nous vous invitons à passer à l'étape de finalisation. [9.2.2](#).

Installer un Claroline personnalisé

Dans le cas où vous disposez d'une plateforme personnalisée, la solution consiste à cloner manuellement les fichiers sources de votre plateforme dans le répertoire `xampp\htdocs`. Par cloner nous entendons copier les fichiers et créer une base de données contenant les informations minimales nécessaires. Avant de suivre les étapes ci-dessous, assurez-vous d'avoir préalablement installé le plug-in *OfflineBundle* tel qu'expliqué en début de chapitre sur votre plateforme Claroline.

La première étape consiste à copier tous les fichiers de votre plateforme, à l'exception du répertoire contenant les ressources dans le répertoire web de Xampp. Ensuite vous devez modifier le fichier `Claroline\app\config\parameters.yml` pour y introduire les informations de connexion à la base de données. Si vous n'avez pas changé ces paramètres lors de l'installation de Xamp, les paramètres par défaut sont administrateur 'root' `database_user: root` et pas de password (`database_password: ~`).

Lors de la dernière étape, il faut créer la base de données minimale nécessaire au fonctionnement de Claroline. Pour ce faire, toujours avec le serveur Apache et MySQL démarré, ouvrez un terminal et déplacez-vous dans le dossier `claro_install\Claroline`. Entrez la commande suivante : `" $ php app/console claroline:install "`. Cette commande va créer la base de données et exécuter les *Fixtures*⁷ Vous disposez maintenant d'un clone de votre Claroline personnalisé, veuillez terminer la création de l'installateur en suivant les instructions du dernier point.

La plateforme pour l'utilisateur déconnecté

Pour faciliter la vie de l'utilisateur déconnecté nous avons écrit un petit script qui automatise le lancement de Xampp et ouvre Google Chrome Portable en mode application native sur

7. Les *Fixtures* sont des composants de Claroline chargeant en base de données des valeurs requises pour le bon fonctionnement de la plateforme

la page d'accueil de Claroline. Nous enregistrons ce fichier sous le nom de `Clarooffline.bat` dans le répertoire de notre installation, soit le répertoire `xampp`.

```
ECHO "Now we start Apache"
ECHO "Now we start MySQL"
xampp_start.exe

ECHO "Let's start chrome"
GoogleChromePortable\GoogleChromePortable.exe
    --app="http://localhost/Claroline/web/app_offline.php"

ECHO "Push any key to stop the program"
pause

ECHO "Shutting down Apache"
ECHO "Shutting down MySQL"
xampp_stop.exe
```

Nous devons maintenant vérifier que le fichier `app_offline.php` se trouve bien dans le répertoire web de Claroline (`xampp\htdocs\Claroline\web\`). Nous devons également supprimer le fichier `xampp\htdocs\Claroline\app\config\sync_config.yml` si toutefois il existait. Enfin, assurons nous que le fichier `xampp\htdocs\Claroline\app\config\is_installed.php` ait bien le contenu suivant :

```
<?php

return true;
```

Nous pouvons maintenant compiler l'ensemble du dossier `xampp` à l'aide d'Inno Setup. Afin de ne pas surcharger ce chapitre nous avons consacré l'annexe B page VII pour expliquer le fonctionnement complet de la compilation avec Inno Setup.

9.2.3 Créer le paquet de ressources pour Linux

Pour la version Linux de l'installateur de Claroline, nous avons choisi de charger l'utilisateur d'installer personnellement le serveur Apache et MySQL. Pour ces utilisateurs qui installent eux-mêmes leur serveur web, nous fournissons le code de Claroline accompagné d'un script chargé d'installer la plateforme sur le serveur. Ce script créera la base de données et exécutera les fixtures. Pendant que celui-ci s'exécute, une page de chargement sera alors affichée à l'utilisateur.

Avant de compresser les sources de Claroline dans une archive pour former le paquet unique, assurez-vous que :

- Le fichier `Claroline/app/config/is_installed.php` soit bien à la valeur `false` :

```
<?php  
  
return false;
```

- Les fichiers `app_offline.php`, `loading_install.php` et `offline_install.php` soient présents dans le dossier `Claroline/web/`. S'ils ne sont pas présents, vous pouvez les copier depuis les sources de notre bundle, ils sont enregistrés dans le répertoire `Claroline/vendor/claroline/offline-bundle/Claroline/OfflineBundle/Ressources/`.
- Le fichier `Claroline/app/config/sync_config.yml` n'existe pas. Dans le cas contraire, il faut le supprimer.

Vos sources sont maintenant prêtes à être compressées. Veuillez créer une nouvelle archive et y inclure l'ensemble de votre dossier `Claroline/`. Notons que cette distribution des sources peut convenir à tout utilisateur installant lui-même son serveur Apache sur son ordinateur personnel, indépendamment de son système d'exploitation.

Récupération d'un compte

10.1 Première connexion

En plus des routes que nous utilisons pour le processus de synchronisation, il est tout à fait possible d'en mettre d'autres en place. C'est d'ailleurs ce que nous avons fait pour la première connexion. En effet, avant de pouvoir échanger des archives entre l'ordinateur personnel et le serveur distant nous devons récupérer l'`exchangeToken` d'un utilisateur. Dans ce chapitre nous allons vous détailler la manière dont nous pouvons ajouter un utilisateur à la plateforme fonctionnant en mode hors-ligne.

Afin de s'assurer que l'utilisateur récupère son compte, nous avons créé un `Listener`¹. Ce dernier est à l'écoute de tout événement de connexion et vérifie si l'utilisateur a déjà récupéré son compte. Si tel est le cas, le `Listener` n'effectue aucune tâche particulière et laisse l'utilisateur faire ce qu'il désire. Dans le cas contraire, le `Listener` va rediriger l'utilisateur vers la vue lui demandant de récupérer son compte (illustré sur la Figure C.9).

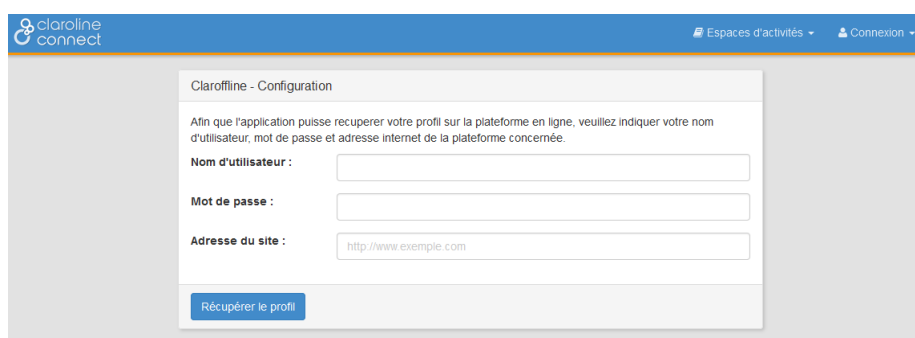
The image shows a web interface titled 'Claroline - Configuration'. At the top, there is a blue header with the 'claroline connect' logo on the left and navigation links 'Espaces d'activités' and 'Connexion' on the right. The main content area is a light gray box containing the configuration form. The form has a title 'Claroline - Configuration' and a subtitle 'Afin que l'application puisse récupérer votre profil sur la plateforme en ligne, veuillez indiquer votre nom d'utilisateur, mot de passe et adresse internet de la plateforme concernée.' Below this, there are three input fields: 'Nom d'utilisateur :', 'Mot de passe :', and 'Adresse du site :'. The 'Adresse du site' field has a pre-filled value 'http://www.exemple.com'. At the bottom of the form is a blue button labeled 'Récupérer le profil'.

FIGURE 10.1 – Vue de configuration affichée à l'aide du `Listener` lors du premier démarrage

Dans le `Listener`, afin d'établir s'il s'agit ou non de la première connexion de l'utilisateur, nous utilisons le fichier `"app/config/sync_config.yml"` enregistré dans les fichiers de Claroline. Si ce fichier n'est pas présent alors nous considérons que l'utilisateur n'a pas encore récupéré les informations de son compte sur le serveur distant. En effet, lorsque nous récupérons un profil, nous créons le dit fichier. Toutefois, il ne faut pas que ce `Listener` agisse sur le serveur distant,

1. Voir Lexique 2.2 page 16

c'est pourquoi nous avons ajouté une seconde condition. Cette seconde condition consiste à tester l'URL sur laquelle nous sommes connectés. S'il s'agit de `localhost`, c'est-à-dire l'adresse du serveur Apache exécuté localement sur l'ordinateur personnel de l'utilisateur déconnecté, alors nous savons que nous sommes sur la plateforme destinée à travailler sans connexion.

Au final, si l'utilisateur n'a pas encore initialisé son compte (créé le fichier `app/config/sync_config.yml`) et tente d'utiliser l'application Claroline déconnectée (via `localhost`), alors nous déclenchons une redirection vers la fenêtre de configuration illustrée à la Figure C.9. Sur cette fenêtre, l'utilisateur est invité à entrer son nom d'utilisateur, son mot de passe et l'adresse de la plateforme à contacter. Lorsque l'utilisateur a entré ces informations, il doit cliquer sur le bouton **RÉCUPÉRER LE PROFIL**. Nous exécutons alors l'action se chargeant de contacter et de récupérer l'identité de l'utilisateur sur le serveur distant que l'utilisateur a précédemment indiqué.

L'action se déroule comme suit :

1. L'ordinateur personnel effectue une requête sur une route précise du serveur donné par l'utilisateur. Dans cette requête, nous insérons les identifiants donnés dans le formulaire.
2. Si le serveur arrive à authentifier l'utilisateur sur base de ses identifiants, il répond à la requête en retournant l'ensemble des informations nécessaires pour la création d'un profil utilisateur. Dans le cas où il n'a pas pu confirmer l'identité, il retourne un code d'erreur HTTP 424.
3. Enfin nous créons un profil utilisateur sur la plateforme Claroline de l'ordinateur personnel et créons notre fichier `app/config/sync_config.yml`.

De manière plus détaillée, nous devons récupérer les informations de l'utilisateur pour pouvoir créer celui-ci dans la base de données locale. Lors de l'étape une, pour télécharger ces informations incluant l'`exchangeToken`, car rappelons-le la plateforme installable ne dispose d'aucunes informations, nous contactons la route `claro_sync_user` qui requiert le nom de l'utilisateur et le mot de passe pour s'identifier. Cette route est la seule exception qui n'utilise pas le token d'échange car c'est précisément celle-ci qui nous permet de le récupérer sur le serveur distant. Une fois le compte récupéré, nous enregistrons quelques informations sur l'utilisateur dans un fichier YAML, que nous appelons `sync_config`. Ce fichier a la forme d'un tableau de tableau où chaque sous-tableau représente un utilisateur et contient son nom d'utilisateur, son adresse mail et l'url qu'il a entré pour charger son compte.

Comme nous l'avons expliqué dans la section sur l'authentification², la méthode d'authentification requiert le password non crypté ; c'est donc de cette manière que nous le transférons dans le contenu de la requête. Pour cette raison, il est important que la route de contact de la plateforme en ligne soit sécurisée au moyen du protocole HTTPS. Nous n'avons malheureusement pas pu tester l'utilisation de *Buzz* avec une adresse HTTPS car nous ne disposons

2. Section Authentification, voir Chapitre 7 page 49

pas de certificat. Après vérification dans la documentation et discussion avec Stéphane Klein, aucun changement ne devrait être fait dans notre code pour utiliser le protocole HTTPS. Seule l'adresse à contacter, donnée par l'utilisateur lors de la configuration doit comporter le préfixe.

Lors de la seconde étape, la méthode `getUserAsTab()` que nous avons créée dans l'entité `User` du `CoreBundle` est appelée si l'authentification a réussi. Cette méthode se charge de collecter les informations dont nous avons besoin pour recréer l'utilisateur sur l'ordinateur personnel. Ces informations sont alors envoyées comme réponse à la requête formulée lors de la première étape.

Enfin la dernière étape se charge de créer l'utilisateur dans la base de données de la plateforme de l'ordinateur personnel. Cet utilisateur est recréé avec un `exchangeToken` identique à celui qu'il a sur le serveur distant. Pour que la base de données locale de l'ordinateur personnelle soit consistante avec la base de données du serveur distant, nous devons également mettre à jour le GUID de l'espace d'activités personnel de l'utilisateur pour qu'il coïncide avec celui en ligne.

10.1.1 Message d'erreurs

Le processus de récupération de compte peut déclencher des erreurs. Une partie d'entre elles étant similaire à celles rencontrées lors de la synchronisation, nous ne les reprendrons pas ici. En plus de ces dernières, deux erreurs sont spécifiques à la tentative de récupération de compte : celle qui intervient lorsque l'on tente de récupérer un compte qui l'a déjà été sur ce même ordinateur personnel et la seconde qui se présente lorsque l'utilisateur tente de récupérer un compte qui n'existe pas sur la plateforme qu'il a référencée.

10.2 Ajout d'un autre utilisateur

La gestion de plusieurs utilisateurs sur un même ordinateur personnel utilisant Clarofline est intégrée à notre bundle. En plus du premier utilisateur qui s'est synchronisé sur la plateforme installée sur l'ordinateur personnel, un autre utilisateur peut récupérer son compte s'il le désire. Pour cela, il lui suffit de reprendre le formulaire que nous imposons au premier utilisateur. Si le nouvel utilisateur entre des informations correctes, son compte pourra être récupéré de la même façon que le fut celui du premier utilisateur. Lorsque nous récupérons un compte, après création du nouvel utilisateur dans la base de données, nous ajoutons un nouveau tableau contenant les informations de ce nouvel utilisateur dans le fichier `sync_config.yml`.

Toutefois, il est important d'avoir en tête que les informations de ces deux utilisateurs seront présentes dans la même base de données et donc que le propriétaire de l'ordinateur pourrait avoir accès à l'ensemble de ces informations s'il consulte la base de données de l'application.

10.3 Modification de l'adresse du site

En plus de la vue permettant à l'utilisateur d'ajouter un autre compte à sa plateforme, nous avons également créé une vue pour lui permettre d'éditer l'adresse du site internet enregistrée dans le fichier `sync_config.yml`. L'idée était d'autoriser l'utilisateur à modifier l'url si nécessaire, par exemple si le site avec lequel il se synchronisait a changé de nom de domaine. Pour cela, il suffit à l'utilisateur de se rendre sur la vue du plugin Claroffline, comme s'il désirait se synchroniser et de cliquer sur le bouton `PARAMÈTRES`.

CHAPITRE 11

Discussion

Ce chapitre traitera de réflexions que nous avons menées sur certaines problématiques que nous avons soulevées en réalisant ce travail ainsi que des limites de ce dernier. Enfin nous donnerons des pistes pour d'éventuelles contributions futures.

11.1 La fonction de suppression

Une fonction qui pourrait poser problème dans l'usage de notre bundle, du fait qu'elle n'est pas implémentée, est la fonction de suppression. En effet, dans le menu de gestion d'une ressource, il est possible pour l'utilisateur disposant des droits appropriés de supprimer une ressource d'un espace d'activités. Or, si une ressource est supprimée dans un espace d'activités sur le serveur distant ou sur l'ordinateur utilisant Claroffline cette modification ne sera pas communiquée par notre bundle. Une ressource supprimée sur le serveur distant ne le sera donc pas sur l'ordinateur utilisant Claroffline. Par exemple, si un professeur en vient à supprimer une ressource, l'étudiant continuera à en disposer même après synchronisation.

A l'heure actuelle, la suppression d'une ressource est directe. Quand une ressource est supprimée les entrées dans la base de données et les éventuels fichiers (pour les ressources Fichier) sont effacées immédiatement. Nous avons pensé à une solution pour gérer cette suppression directe, néanmoins lors d'une discussion avec Stéphane Klein, ce dernier nous a annoncé que la gestion d'une suppression plus douce des ressources était en discussion mais ne faisait pas partie des priorités. Avec Stéphane Klein, nous avons alors déduit qu'il n'était pas pertinent de notre part d'essayer d'implémenter une gestion du "hard delete" vu qu'elle serait amenée à être remplacée.

Cette suppression plus douce ("soft delete") consisterait à rendre invisible pour l'utilisateur une ressource dite supprimée et à n'effacer effectivement les entrées en base de données et les fichiers que dans un second temps. Intégrer de telles modifications nécessite une manipulation majeure de la fonction de suppression de Claroline. Effectuer ces modifications nous même nous aurait cependant dissocié du développement principal de Claroline. Or notre but premier était de faire un plug-in directement intégrable à la version principale et non pas sur une version qui nous était propre.

Néanmoins, nous avons réfléchi à une manière pour notre *OfflineBundle* d'implémenter cette suppression douce. Un champ `isDelete` pourrait être ajouté à l'entité `resource_node` pour déterminer si celle-ci est dans un état visible ou non pour l'utilisateur. Lorsque celui-ci

effectuerait une action de suppression sur une ressource, le champ `isDelete` sera enregistré à la valeur `vrai`. Du fait de la modification d'un attribut de la variable, sa date de modification serait mise à jour, ce qui aurait pour conséquence d'intégrer cette ressource dans la synchronisation suivante. Et par l'ajout du champ dans les valeurs transmises dans le manifeste, la ressource serait également rendue invisible sur l'autre plateforme.

Une variante à cette solution consiste à ne pas synchroniser le champ `isDelete` mais à le comparer. Dans le cas où les deux champs sont vrais alors la ressource est supprimée. Etant donné que cette question n'est pas fixée chez Claroline, elle est restée en suspens chez nous également. Toutefois, les mises à jour à effectuer dans notre code sont mineures. Pour que cela soit pris en charge dans notre code, la seule adaptation à faire serait d'ajouter le champ `isDelete` dans `OfflineResource`.

11.2 Inscription à un nouvel espace d'activités

Une question de réflexion qui nous a occupé pour le travail du *OfflineBundle* est celle de l'inscription d'un étudiant à un nouvel espace d'activités. Cette possibilité est en effet essentielle car il ne faudrait pas qu'un étudiant ait à télécharger une nouvelle fois l'ensemble de la plateforme uniquement pour y ajouter un espace d'activités. Notre travail pour rendre la synchronisation la plus légère possible serait plus utile.

Dans notre implémentation actuelle, pour s'inscrire à un nouvel espace d'activités, un étudiant doit se connecter sur son site de référence via un navigateur web, s'inscrire au cours de son choix et ensuite lancer une synchronisation depuis son ordinateur personnel utilisant Claroline. De cette manière, l'ensemble des ressources nouvellement mises à sa disposition en ligne seront téléchargées.

Nous sommes arrivés à ce choix car après réflexion, l'inscription hors ligne (sans connexion Internet) à un espace d'activités ne montre que peu d'intérêt. En effet, nous avons analysé deux possibilités de gérer l'inscription à un espace d'activités dit publique sans avoir d'accès Internet.

Soit, quand l'utilisateur s'inscrit à un nouvel espace d'activités, il bénéficie directement de l'ensemble des ressources de cet espace. Cette solution implique le téléchargement préalable de toutes les ressources de tous les espaces d'activités publiques. Ceci représente une quantité relativement élevée d'informations transférées et stockées sur l'ordinateur. De plus, si l'utilisateur ne désire s'inscrire qu'à peu, voire aucun d'espace d'activités, ces informations seront inutiles.

Soit, l'utilisateur pourrait s'inscrire à un nouvel espace d'activités mais ne disposerait pas de son contenu. Le contenu de celui-ci ne serait téléchargé que lors de la synchronisation suivante requérant, elle, une connexion Internet. Nous avons sérieusement commencé à envisager cette solution et l'avons implémenté comme suit. Lors de l'étape de synchro-

nisation, nous ajoutons la liste des espaces d'activités publiques en fin de manifeste. Cette liste d'espace d'activités publiques était alors chargée et les espaces d'activités inexistantes dans la base de données de l'ordinateur utilisant Claroffline étaient créés. En procédant ainsi, l'utilisateur voyait apparaître les espaces d'activités sur la page d'inscription.

Toutefois, après de plus amples analyses, nous avons abandonné cette solution car elle présente un défaut de taille. Elle ne permet pas de tenir compte efficacement des contraintes d'inscription à un espace d'activités. Ces contraintes peuvent être diverses : un nombre maximum d'inscrits, une période limitée d'inscription,... Au final nous avons décidé d'écarter la possibilité de s'inscrire à un cours hors ligne car l'intérêt en était limité comparativement à la complexité engendrée sur le bundle.

11.3 Gestion des fuseaux horaires

Lors de la phase d'analyse de ce projet, M. Mercenier nous a rappelé qu'il était possible de rencontrer un cas dans lequel un utilisateur serait amené à se synchroniser depuis un fuseau horaire différent de celui sur lequel est localisé la plateforme de référence. En effet, cette remarque soulève des questions avec notre module de synchronisation puisque les données transmises dépendent de la date de la synchronisation et de la date de modification des ressources. La date actuellement utilisée est l'heure de l'ordinateur qui exécute Claroline. Le problème ne survient donc que lorsque le fuseau horaire diffère entre l'ordinateur utilisant Claroffline et le serveur distant.

Actuellement, notre code ne prend pas cette spécificité en compte, nous comparons la date de l'ordinateur avec la date sauvegardée en base de données. Une solution qui pourrait être appliquée pour prendre en charge les fuseaux horaires serait d'enregistrer en base de données toutes les dates sur un fuseau horaire fixé arbitrairement, par exemple GMT. A chaque utilisation de ces dates il faudrait alors adapter celles-ci au fuseau horaire de la machine. Bien entendu, pour appliquer cette solution, il faudrait réviser l'ensemble du code de Claroline et ne pas se limiter au *OfflineBundle*. Etant donné le temps imparti et ayant mis l'accent sur les fonctionnalités premières de notre bundle, nous n'avons pu réviser l'ensemble du code de Claroline.

Conclusion

Le but de ce projet était de créer un module permettant d'avoir accès à Claroline tout en étant déconnecté et également de pouvoir se synchroniser dans le but d'envoyer le travail réalisé sans connexion et de télécharger les nouveaux contenus disponibles sur le serveur principal. Ce travail s'articule sur deux parties principales : l'installation et la synchronisation. Cette dernière se divisant elle-même en 3 modules : la création d'une archive, le transfert de cette archive et le chargement. Ces trois modules étant articulés par un service principal.

Le premier de nos modules est celui se chargeant de la création de l'archive. Ce module analyse les espaces d'activités de l'utilisateur afin d'en extraire, à l'aide de requêtes sur la base de données, les différentes ressources à synchroniser. Une fois ces ressources trouvées, notre module ajoute leurs descriptions au sein d'un fichier XML et ajoute les éventuelles fichiers nécessaires au sein de l'archive destinée à être transmise.

Le second module, le transfert est responsable de l'échange de données entre le serveur distant et l'application Claroffline. Ce transfert s'effectue en fragmentant l'archive en plusieurs segments avant d'envoyer ces derniers sur l'autre plateformes grâce à des requêtes POST.

Notre troisième module se charge de décompresser et traiter l'archive créer par le module de création. Ce dernier va parcourir le fichier XML reçu et créer ou mettre à jour les différentes ressources sur base des informations dont il dispose. Notre module de chargement tout comme celui de création font usage des services taggés de Symfony. Ceux-ci permettent de facilement ajouter une ressource parmi celles gérées par nos modules.

L'installation quant à elle consistait à trouver un moyen efficace de diffuser l'application Claroffline à tout les utilisateurs qui souhaiteraient s'en servir.

Ce projet fut l'occasion de nous plonger dans un projet professionnel conséquent et en constant développement. il a également constitué une opportunité d'acquérir des connaissances en PHP et plus particulièrement sur le framework *Symfony*. Nous avons également pu apporter des modifications au *CoreBundle* de Claroline afin d'ajouter les modifications nécessaires au fonctionnement de notre *OfflineBundle*. A l'heure à laquelle nous écrivons ces lignes, ces modifications ont été acceptées et seront intégrées dans la prochaine mise-à-jour du noyau.

Nous avons pleinement conscience que la plateforme Claroline est amenée à évoluer avec le temps, et a d'ailleurs déjà évolué depuis le début de notre projet. Nous avons conçu notre travail en gardant cette dimension à l'esprit, c'est pourquoi nous avons fait notre maximum pour qu'il soit facile à maintenir et à étendre par la suite. En effet, comme décrit dans ce document, une simple dépendance à notre bundle et une extension de notre classe `OfflineRessource` suffit à ajouter une ressource à celles déjà prises en compte.

Finalement, en développant *OfflineBundle*, nous sommes heureux d'avoir apporté notre contribution à la plateforme Claroline car nous sommes après tout des utilisateurs de celle-ci depuis près de cinq ans. En effet, après ces années d'utilisation de la plateforme Claroline dans le cadre de nos cours, nous avons pu satisfaire notre curiosité, découvrir et travailler au développement de cet outil ainsi qu'ajouter notre contribution à la plateforme.

Bibliographie

- [1] Alexandre, B. *Développez votre site web avec le framework Symfony2*. OpenClassroom, 2013. 512 p.
- [2] Vanderperren, C. 2010. *Un outil de réunions à distance pour Claroline*. Mémoire de licence non publié, Université catholique de Louvain. Ecole polytechnique de Louvain, Louvain-la-Neuve.
- [3] Potencier, F. 2014. *Symfony Documentation*. En ligne <http://symfony.com/>.
- [4] Potencier, F. 2014. *Doctrine et les bases de données*. En ligne <http://symfony.com/fr/doc/current/book/doctrine.html>.
- [5] Selier, J. 2010. *Simple REST Client*. En ligne <https://chrome.google.com/webstore/detail/simple-rest-client/fhjcajmcblldhlcimfajhfbgofnpcjmb/reviews>.
- [6] Wikipedia. 2014. Liste des codes HTTP. En ligne http://fr.wikipedia.org/wiki/Liste_des_codes_HTTP, consulté le 24 juin 2014.
- [7] Apache Friends. Xampp. En ligne <https://www.apachefriends.org/fr/index.html>, consulté le 24 juin 2014.
- [8] Wallsmith, K. 2014. Buzz. En ligne <https://github.com/kriswallsmith/Buzz>, consulté le 24 mars 2014.
- [9] MySQL. 2010. Reference Manual. En ligne <http://dev.mysql.com/doc/refman/5.0/fr/>.
- [10] MySQL Manual. 2010. mysqldump — A Database Backup Program. En ligne <http://dev.mysql.com/doc/refman/5.0/en/mysqldump.html>.
- [11] Knp Labs. 2014. Behat Documentation. En ligne <http://docs.behat.org/>
- [12] Knp Labs. 2014. http://docs.behat.org/cookbook/behat_and_mink.html. En ligne http://docs.behat.org/cookbook/behat_and_mink.html
- [13] Knp Labs. 2014. *Mink Documentation*. En ligne http://docs.behat.org/cookbook/using_the_profiler_with_minkbundle.html.
- [14] Ubuntu-fr. 2014. *Serveur HTTP Apache 2*. En ligne <http://doc.ubuntu-fr.org/apache2>.
- [15] Ubuntu-fr. 2014. *MySQL*. En ligne <http://doc.ubuntu-fr.org/mysql>.
- [16] NovaWay. 2014. *LES REPOSITORY ET LES MANAGERS D'ENTITÉ DANS SYMFONY2*. En ligne <http://www.lafermeduweb.net/tutorial/les-repository-et-les-managers-d-entite-dans-symfony2-p99.html>.
- [17] SensioLab. 2014. *PHP-CS-Fixer*. En ligne [PHP-CS-Fixer](http://sensiolabs.com/fixer/).
- [18] W3schools. 2014. *XML*. En ligne <http://www.w3schools.com/xml/>.
- [19] W3schools. 2014. *JSON*. En ligne <http://www.w3schools.com/json/>.

- [20] PHP Manual. 2014. *base64_encode*. En ligne <http://be2.php.net/manual/fr/function.base64-encode.php>.
- [21] PHP Manual. 2014. *Définition de PHP*. En ligne <http://php.net/manual/fr/intro-what-is.php>
- [22] PHP Manual. 2014. *DOMDocument*. En ligne <http://www.php.net/manual/fr/class.domdocument.php>.
- [23] PHP Manual. 2014. *ZipArchive*. En ligne <http://be2.php.net/manual/fr/class.ziparchive.php>.
- [24] PHP Manual. 2014. *ZipArchive*. En ligne <http://be2.php.net/manual/fr/class.ziparchive.php>.
- [25] PHP Manual. 2014. *Fonctions Hash*. En ligne <http://www.php.net/manual/fr/ref.hash.php>.
- [26] Symfony Glossary. *Définition Bundle*. En ligne <http://symfony.com/fr/doc/current/glossary.html#bundle>
- [27] Symfony Glossary. *Définition Service*. En ligne <http://symfony.com/fr/doc/current/glossary.html#term-service>
- [28] Symfony Glossary. *Définition Conteneur de services*. En ligne http://symfony.com/fr/doc/current/book/service_container.html
- [29] PHP CS Fixer. 2014. *Dépôt du projet*. En ligne <https://github.com/fabpot/PHP-CS-Fixer>.
- [30] Création Ram Disk Windows. *Ram Disk Windows Lien 1*. En ligne <http://www.ltr-data.se/opencode.html/#ImDisk>
- [31] Création Ram Disk Windows. *Ram Disk Windows Lien 2*. En ligne <http://forums.cnetfrance.fr/topic/1173078-creer-un-ramdisk-disque-virtuel-en-ram-avec-imdisk/>
- [32] Création Ram Disk Linux. *Ram Disk Linux*. En ligne <http://www.generation-linux.fr/index.php?post/2009/05/04/tmpfs-%3A-utiliser-sa-ram-comme-repertoire-de-stockage>

Annexes

OfflineResource Exemple

Vous trouverez ci-dessous le code source d'exemple pour l'ajout d'une ressource au OfflineBundle.

```
<?php

/*
 * This file is part of the Claroline Connect package.
 *
 * (c) Claroline Consortium <consortium@claroline.net>
 *
 * For the full copyright and license information,
 * please view the LICENSE
 * file that was distributed with this source code.
 */

namespace Claroline\OfflineBundle\Model\Resource;

use Claroline\CoreBundle\Entity\User;
use Claroline\CoreBundle\Entity\Resource\ResourceNode;
use Claroline\CoreBundle\Entity\Workspace\Workspace;
use Claroline\CoreBundle\Manager\ResourceManager;
use Claroline\CoreBundle\Persistence\ObjectManager;
use Claroline\OfflineBundle\Model\SyncInfo;
use Symfony\Component\DependencyInjection\ContainerInterface;
use JMS\DiExtraBundle\Annotation as DI;
use \DOMDocument;
use \DateTime;
use \ZipArchive;

/**
 * @DI\Service("claroline_offline.offline.example")
 * @DI\Tag("claroline_offline.offline")
 */
class OfflineExample extends OfflineResource
```

```

{

    /**
     * Constructor.
     *
     * @DI\InjectParams({
     *     "container"
     *     = @DI\Inject("service_container")
     * })
     */
    public function __construct(
        ContainerInterface $container
    )
    {
        $this->container = $container;
    }

    // Return the type of resource supported by this service
    public function getType(){
        return 'example';
    }

    /**
     * Add informations required to check and recreated
     * a resource if necessary.
     *
     * @param \Claroline\CoreBundle\Entity\Resource\ResourceNode
     *         $resToAdd
     * @param \ZipArchive $archive
     */
    public function addResourceToManifest($domManifest,
        $domWorkspace, ResourceNode $resToAdd,
        ZipArchive $archive, $date)
    {
        parent::addNodeToManifest($domManifest,
            $this->getType(), $domWorkspace, $resToAdd);
        /*
         * Rajouter ici les informations supplementaires
         * necessaires a la bonne gestion
         * de la ressource par la suite.
         */
        return $domManifest;
    }
}

```

```

/**
 * Create a resource of the type supported
 *      by the service based on the XML file.
 *
 * @param \Claroline\CoreBundle\Entity\Workspace\Workspace
 *      $workspace
 * @param \Claroline\CoreBundle\Entity\User $user
 * @param \Claroline\OfflineBundle\Model\SyncInfo $wsInfo
 * @param string $path
 *
 * @return \Claroline\OfflineBundle\Model\SyncInfo
 */
public function createResource($resource,
                               Workspace $workspace, User $user,
                               SyncInfo $wsInfo, $path)
{
    /*
     * Rajouter ici les operations necessaires
     *      a la re-creation de la ressource.
     */
    return $wsInfo;
}

/**
 * Update a resource of the type supported
 *      by the service based on the XML file.
 *
 * @param \Claroline\CoreBundle\Entity\Resource\ResourceNode
 *      $node
 * @param \Claroline\CoreBundle\Entity\Workspace\Workspace
 *      $workspace
 * @param \Claroline\CoreBundle\Entity\User $user
 * @param \Claroline\OfflineBundle\Model\SyncInfo
 *      $wsInfo
 *
 * @param string $path
 *
 * @return \Claroline\OfflineBundle\Model\SyncInfo
 *
 */
public function updateResource($resource, ResourceNode $node,
                               Workspace $workspace, User $user, SyncInfo $wsInfo, $path)
{

```

```
    /*
    *   Rajouter ici les operations necessaires
    *       a la mise-a-jour de la ressource.
    */
    return $wsInfo;
}

/**
 * Create a copy of the resource in case of conflict
 * (e.g. if a ressource has been modified both offline
 * and online)
 *
 * @param \Claroline\CoreBundle\Entity\Workspace\Workspace
 *                                     $workspace
 * @param \Claroline\CoreBundle\Entity\Resource\ResourceNode
 *                                     $node
 * @param string $path
 */
public function createDoublon($resource, Workspace $workspace,
                             ResourceNode $node, $path)
{
    /*
    *   Rajouter ici les operations necessaires a creation
    *       d'un doublon pour la ressource si necessaire.
    */
    return;
}
}
```

ANNEXE B

Inno Setup

Dans cette annexe nous détaillerons comment compiler les sources de Claroline pour Windows afin de créer un fichier exécutable distribuable et facilitant l'installation pour les utilisateurs de ce système d'exploitation. Nous réaliserons cette opération en utilisant l'outil Inno Setup et en nous basant sur les sources préparées en suivant les instructions détaillées dans le chapitre sur l'installation [9 page 71](#).

A l'ouverture, Inno Setup vous propose de créer un nouveau script de compilation de l'installateur sur base d'un fichier vierge ou d'un wizard. Si vous souhaitez être guidé dans la réalisation du script par le wizard qui permet de le générer, nous détaillerons ci-dessous les différentes étapes et la manière dont nous avons configuré notre script. Si vous le souhaitez, notre script est directement disponible au point [B.2](#).

B.1 Utilisation du wizard

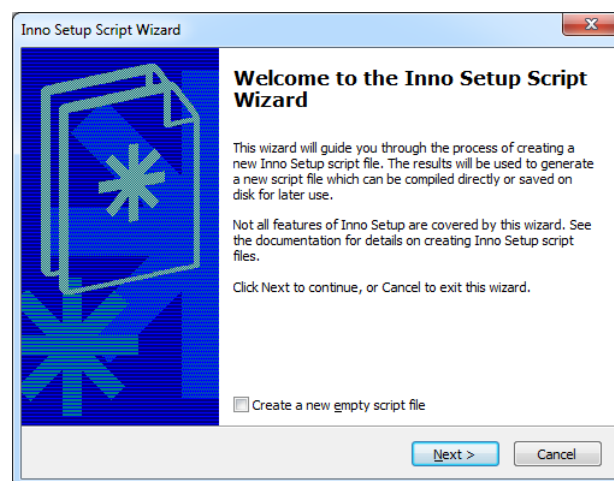


FIGURE B.1 – Etape 1 du wizard de configuration d'Inno Setup

Passé la fenêtre d'accueil, illustrée à la Figure [B.1](#), vous êtes amenés à entrer le nom que vous souhaitez donner à l'application, sa version ainsi que les références sur le publieur (voir Figure [B.2](#)).

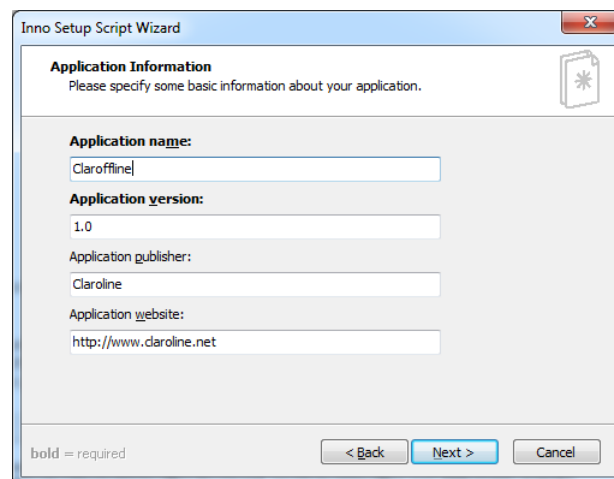


FIGURE B.2 – Etape 2 du wizard de configuration d'Inno Setup

Sur la troisième vue du wizard (voir Figure B.3), nous avons à choisir l'emplacement disque dur qui sera conseillé à l'utilisateur pour installer notre application Claroffline. Une case à cocher permet également d'autoriser l'utilisateur à changer cette option lorsqu'il installera l'application.

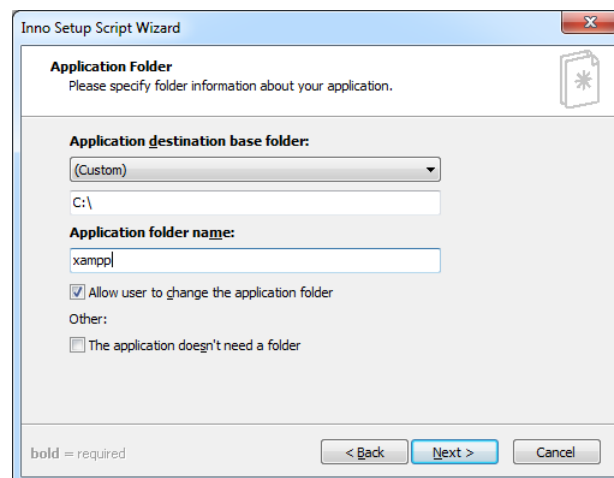


FIGURE B.3 – Etape 3 du wizard de configuration d'Inno Setup

Lors de la quatrième étape (Figure B.4), nous devons donner le point d'entrée principal de notre programme. Dans notre cas, ce point d'entrée est le script `Claroffline.bat` que nous avons écrit. De plus, dans la moitié inférieure de la fenêtre, il faut référencer le dossier

contenant toutes les sources du programme.

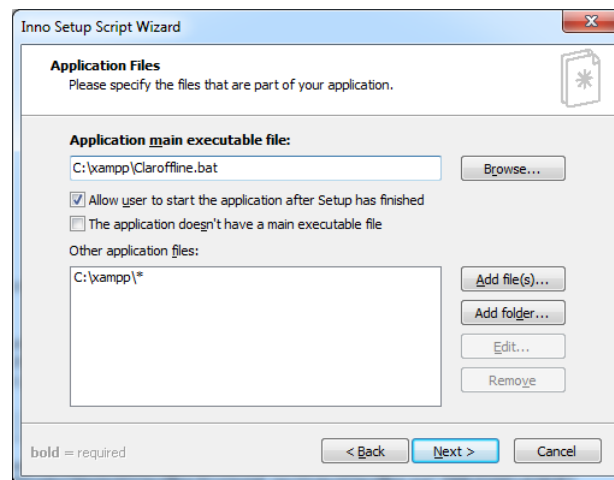


FIGURE B.4 – Etape 4 du wizard de configuration d’Inno Setup

Le cinquième panneau de ce formulaire (Figure B.5) demande le nom qu’aura l’application dans le menu de démarrage et permet de créer de manière automatique l’outil de désinstallation.

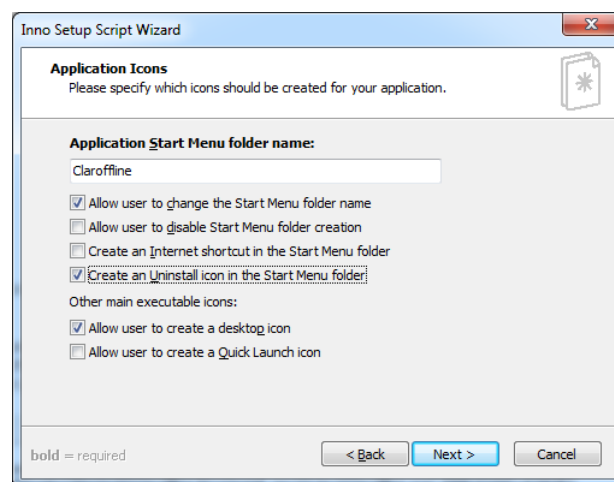


FIGURE B.5 – Etape 5 du wizard de configuration d’Inno Setup

Le sixième panneau (Figure B.6) permet d’ajouter une licence au logiciel. Nous avons repris la licence disponible sur le dépôt principal de Claroline sur Github.

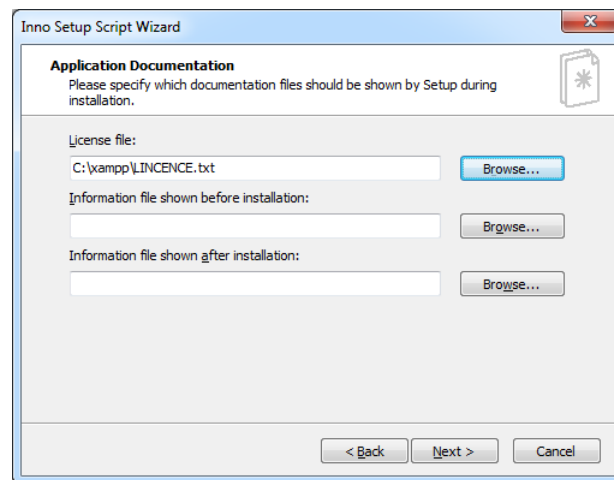


FIGURE B.6 – Etape 6 du wizard de configuration d'Inno Setup

Le septième panneau (Figure B.7) est consacré au choix de la langue dans laquelle sera disponible l'installateur. Nous avons sélectionné anglais, français et espagnol car ce sont les trois langues actuellement disponible pour Claroline.

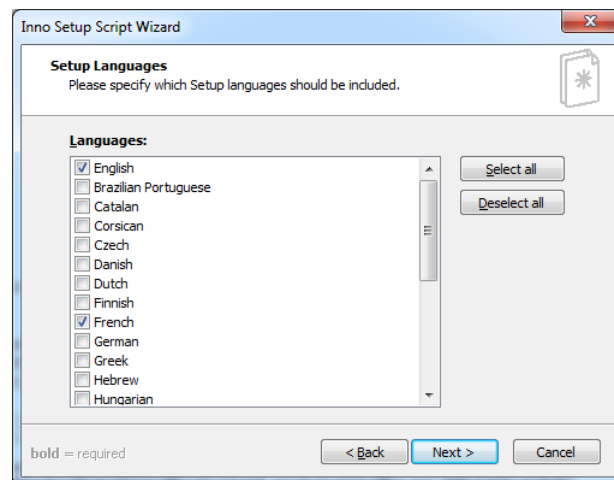


FIGURE B.7 – Etape 7 du wizard de configuration d'Inno Setup

L'avant dernier panneau (Figure B.8) permet de choisir le nom du fichier .exe et de lui ajouter une icône.

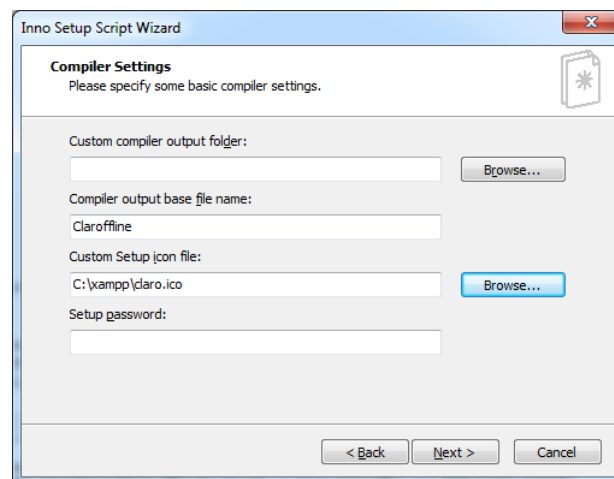


FIGURE B.8 – Etape 8 du wizard de configuration d’Inno Setup

Enfin, sur la dernière fenêtre (Figure B.9), il nous reste à confirmer de générer le script.

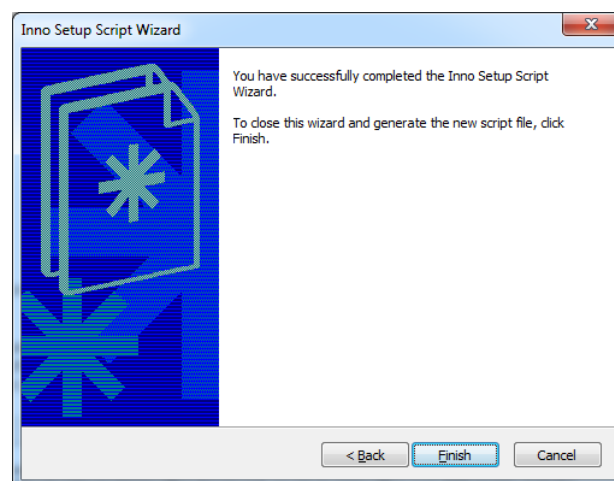


FIGURE B.9 – Etape 9 du wizard de configuration d’Inno Setup

Avant d’accepter la compilation de ce script, nous avons encore quelques éléments à y ajouter. En effet, Xampp nécessite d’ajouter des variables d’environnement à Windows¹. C’est pourquoi il faut que la personne chargée de l’installation les ajoute à l’installation et

1. Le code permettant d’ajouter des variable d’environnement dans un script d’Inno Setup a été trouvé sur dans un message du forum du site Internet développez.com (<http://www.developpez.net/forums/d484811/envIRONNEMENTS-developpement/delphi/outils/inno-setup-modifier-d-cle-registre/>)

les enlève à la désinstallation. Le code à ajouter dans le script d'Inno Setup est le suivant :

```
[Registry]
Root: HKLM;
  SubKey: "SYSTEM\CurrentControlSet\Control
          \Session Manager\Environment\";
  ValueType: string; ValueName: "Path";
  ValueData: "{reg:HKLM\SYSTEM\CurrentControlSet\Control
              \Session Manager\Environment\,Path};
              {app}\php"
Root: HKLM;
  SubKey: "SYSTEM\CurrentControlSet\Control
          \Session Manager\Environment\";
  ValueType: string; ValueName: "Path";
  ValueData: "{reg:HKLM\SYSTEM\CurrentControlSet\Control
              \Session Manager\Environment\,Path};
              C:\Windows\System32"

[Code]
procedure CurUninstallStepChanged(CurUninstallStep: TUninstallStep);
var
  Path, AppDir: string;
  Index: Integer;
begin
  if CurUninstallStep = usUninstall then
  begin
    if RegQueryStringValue(HKEY_LOCAL_MACHINE,
      'SYSTEM\CurrentControlSet\Control
      \Session Manager\Environment\,
      'Path', Path) then
    begin
      AppDir := ExpandConstant('{app}\php');
      Index := Pos(AppDir, Path);
      Delete(Path, Index-1, Length(AppDir)+1);
      RegWriteStringValue(HKEY_LOCAL_MACHINE,
        'SYSTEM\CurrentControlSet\Control
        \Session Manager\Environment\,
        'Path', Path);
    end;
  end;
end;
```

De plus, le contenu du répertoire référencé par les variables d'environnement n'est chargé

qu'à l'ouverture d'une session Windows. Nous allons donc ajouter une dernière fonction supplémentaire demandant à l'utilisateur de redémarrer après l'installation. Le code à ajouter dans le script est le suivant :

```
[Setup]
AlwaysRestart = yes
```

Maintenant que le script est complet, nous vous invitons à le compiler. Vous disposerez maintenant d'un fichier exécutable pour l'application Claroffline.

B.2 Script Inno Setup

Le script ci-dessous est le script complet que nous avons exécuté avec Inno Setup. Il nous a permis de compiler les sources pour en faire un fichier exécutable, qui est notre installateur de l'application Claroffline pour Windows.

```
; Script generated by the Inno Setup Script Wizard.
; SEE THE DOCUMENTATION FOR DETAILS
; ON CREATING INNO SETUP SCRIPT FILES!

[Setup]
; NOTE: The value of AppId uniquely identifies this application.
; Do not use the same AppId value
;       in installers for other applications.
; (To generate a new GUID, click Tools | Generate GUID inside the IDE.)
AppId={{AEC3BE9F-49B3-485F-9392-F08654B04220}}
AppName=Claroffline
AppVersion=1.0
; AppVerName=Claroffline 1.0
AppPublisher=Claroline
AppPublisherURL=http://www.claroline.net
AppSupportURL=http://www.claroline.net
AppUpdatesURL=http://www.claroline.net
DefaultDirName=C:\xampp
DefaultGroupName=Claroffline
LicenseFile=C:\xampp\LINCENCE.txt
OutputBaseFilename=Claroffline
SetupIconFile=C:\xampp\claro.ico
Compression=lzma
SolidCompression=yes
```

```

[Registry]
Root: HKLM;
  SubKey: "SYSTEM\CurrentControlSet\Control
          \Session Manager\Environment\";
  ValueType: string; ValueName: "Path";
  ValueData: "{reg:HKLM\SYSTEM\CurrentControlSet\Control
              \Session Manager\Environment\,Path};
              {app}\php"
Root: HKLM;
  SubKey: "SYSTEM\CurrentControlSet\Control
          \Session Manager\Environment\";
  ValueType: string; ValueName: "Path";
  ValueData: "{reg:HKLM\SYSTEM\CurrentControlSet\Control
              \Session Manager\Environment\,Path};
              C:\Windows\System32"

[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"
Name: "french"; MessagesFile: "compiler:Languages\French.isl"
Name: "spanish"; MessagesFile: "compiler:Languages\Spanish.isl"

[Tasks]
Name: "desktopicon";
  Description: "{cm:CreateDesktopIcon}"; GroupDescription:
              "{cm:AdditionalIcons}";
  Flags: unchecked

[Files]
Source: "C:\xampp\Claroffline.bat"; DestDir: "{app}";
  Flags: ignoreversion
Source: "C:\xampp\*";
  DestDir: "{app}";
  Flags: ignoreversion recursesubdirs createallsubdirs
; NOTE: Don't use "Flags: ignoreversion" on any shared system files

[Icons]
Name: "{group}\Claroffline"; Filename: "{app}\Claroffline.bat"
Name: "{group}\{cm:UninstallProgram,Claroffline}";
  Filename: "{uninstallexe}"
Name: "{commondesktop}\Claroffline";
  Filename: "{app}\Claroffline.bat";
  Tasks: desktopicon

```

```
[Run]
Filename: "{app}\Claroffline.bat"; Description:
        "{cm:LaunchProgram,Claroffline}";
        Flags: shellexec postinstall skipifsilent

[Code]
procedure CurUninstallStepChanged(CurUninstallStep: TUninstallStep);
var
    Path, AppDir: string;
    Index: Integer;
begin
    if CurUninstallStep = usUninstall then
    begin
        if RegQueryStringValue(HKEY_LOCAL_MACHINE,
            'SYSTEM\CurrentControlSet\Control
                                \Session Manager\Environment\ ',
            'Path', Path) then
        begin
            AppDir := ExpandConstant('{app}\php');
            Index := Pos(AppDir, Path);
            Delete(Path, Index-1, Length(AppDir)+1);
            RegWriteStringValue(HKEY_LOCAL_MACHINE,
                'SYSTEM\CurrentControlSet\Control
                                \Session Manager\Environment\ ',
                'Path', Path);
        end;
    end;
end;

[Setup]
AlwaysRestart = yes
```


Manuel d'utilisation

C.1 Introduction

Bienvenue sur le manuel d'utilisation du OfflineBundle de Claroline. Ce bundle permet à l'utilisateur de synchroniser son ordinateur personnel avec un serveur claroline distant. En quoi consiste la fonctionnalité offline de notre bundle ? La fonctionnalité offline vous permet d'ouvrir Claroline et de travailler sur la plateforme dès lors que vous n'avez pas de connexion internet. Vous disposerez alors d'un accès à l'ensemble de vos espaces d'activités et aux différentes ressources qu'ils contiennent ¹. Si vous êtes formateur, vous pourrez également continuer à produire du contenu pour les espaces d'activités dont vous êtes manager.

Lorsque vous, étudiants ou formateurs, disposerez d'un accès à Internet, vous pourrez lancer une synchronisation avec le serveur distant. Cette synchronisation se chargera d'envoyer sur le serveur distant tout le travail que vous avez effectué hors ligne. Et, dans l'autre sens, de récupérer l'ensemble des nouveaux contenus disponibles en ligne afin de les ajouter sur le Claroline installé sur votre ordinateur. Les ressources présentes dans l'application Claroffline installée sur votre ordinateur personnel sera maintenant à jour et vous pourrez travailler avec les nouvelles ressources même si vous n'avez plus accès à Internet.

C.2 Installer Claroffline sur son ordinateur

Si vous lisez ce manuel, nous supposons que vous êtes utilisateur d'une plateforme Claroline et que vous disposez d'un installateur de Claroffline créé par l'administrateur de votre serveur Claroline. Dans le but de proposer des instructions les plus spécifiques possibles, nous invitons les utilisateurs Windows à se rendre à la section [C.2.1](#), les utilisateurs Linux à passer au point [C.2.2](#). Si vous êtes un utilisateur avancé disposant des connaissances techniques nécessaires et que vous souhaitez comprendre le fonctionnement de l'installateur et des prérequis pour disposer de Claroline sur votre ordinateur personnel nous vous invitons à lire le [Chapitre 9 page 71](#) détaillant l'installation d'*OfflineBundle*

C.2.1 Windows

Pour installer l'application Claroffline sous Windows, commencez par télécharger l'installateur. Celui-ci devrait vous être fournis sur le site internet de la plateforme Claroline que

1. Nous parlons ici de l'ensemble des ressources des différents types gérés par notre bundle. Au moment où ce document est écrit, les types pris en compte sont les suivants : les textes, les fichiers, les répertoires et les forums

vous utilisez. Une fois le fichier exécutable téléchargé, double cliquez dessus. Une première fenêtre s'affiche, vous proposant le choix de la langue dans laquelle seront écrites les instructions d'installation (Figure C.1).

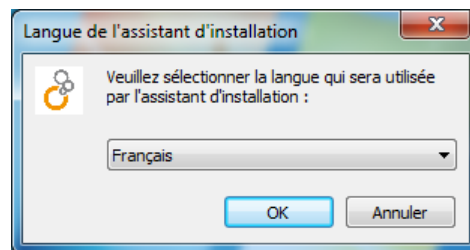


FIGURE C.1 – Vue de configuration affichée à l'aide du Listener lors du premier démarrage

Vous arrivez ensuite sur une fenêtre (Figure C.2) vous confirmant que vous allez installer Claroffline.

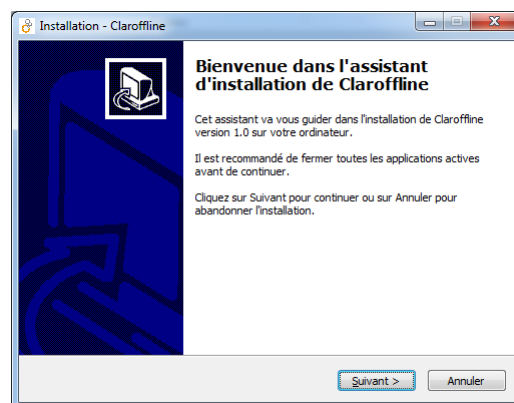


FIGURE C.2 – Etape 1 du wizard d'installation

L'étape suivante (Figure C.3) consiste à confirmer que vous acceptez la licence d'utilisation de Claroline.

La troisième étape (Figure C.4) est le choix du répertoire d'installation de la plateforme. Nous vous conseillons vivement de laisser C:\xampp.

La quatrième étape (Figure C.5) vous permet de configurer le nom qu'aura l'application dans votre menu démarrer.

La cinquième étape (Figure C.6) vous permet d'automatiser la création d'une icône sur le bureau.

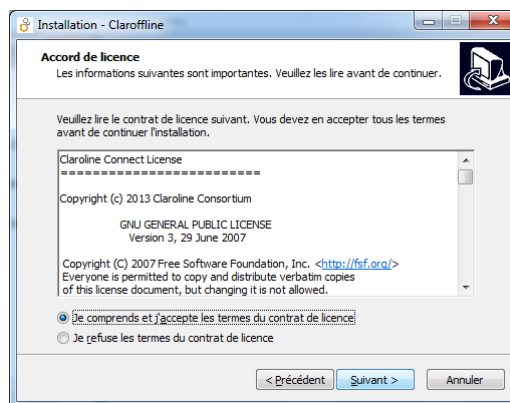


FIGURE C.3 – Etape 2 du wizard d'installation

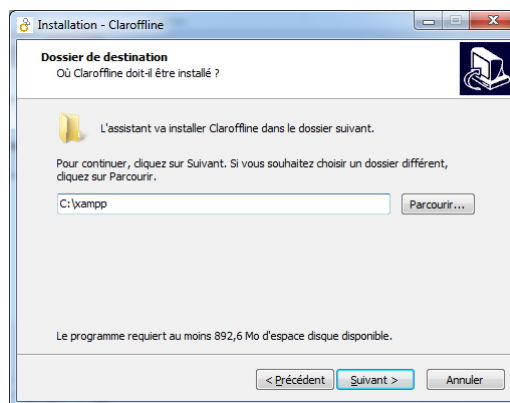


FIGURE C.4 – Etape 3 du wizard d'installation

Enfin, la sixième et dernière étape (Figure C.7) vous demande une dernière confirmation avant de lancer effectivement l'installation.

Après un petit temps de chargement, l'application Claroffline sera installée sur votre ordinateur personnel. Vous pouvez dès lors démarrer la plateforme et passer à l'étape de récupération de votre compte au point C.3.

C.2.2 Linux

La première étape pour installer Claroffline sur votre ordinateur linux est d'installer PHP 5.4 ou supérieur, Apache 2 et MySql. Pour l'installation de MySql, un nom d'utilisateur et un mot de passe vous seront demandé. Nous vous proposons de laisser les valeurs par défaut (root et pas de password). Si toutefois vous souhaitez les changer, il faudra modifier le fichier

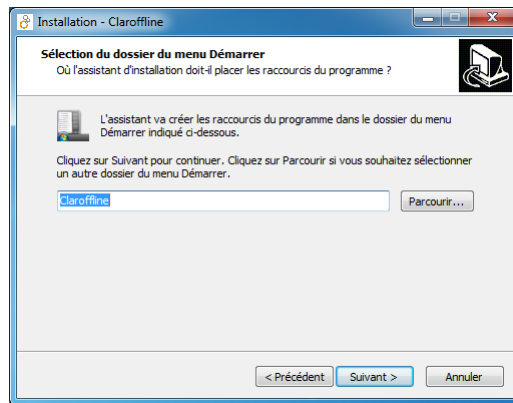


FIGURE C.5 – Etape 4 du wizard d'installation

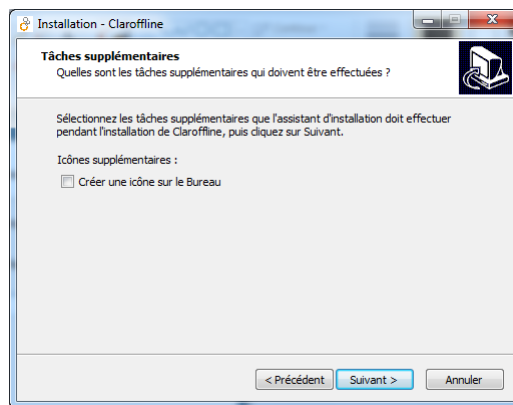


FIGURE C.6 – Etape 5 du wizard d'installation

`~Claroline/app/config/parameters.yml` contenant la configuration de Claroline. Pour installer les dits paquets, utilisez la commande :

```
$ sudo apt-get install php5 php5-mysql php-apc  
    apache2 libapache2-mod-php5 apache2-doc  
    mysql-server mysql-client php5-curl
```

Une fois ces quatre paquets installés, décompressez l'installateur que vous avez téléchargé depuis votre serveur Claroline. Établissez que vous avez décompressé l'archive dans votre dossier `/Claroline`. Vous devez maintenant ajouter des permissions sur les dossiers `~Claroline/app/cache/`, `~Claroline/app/config/`, `~Claroline/app/logs/`, `~files/`, `~templates/` et `~web/`. Pour ce faire, entrez la commande suivante :

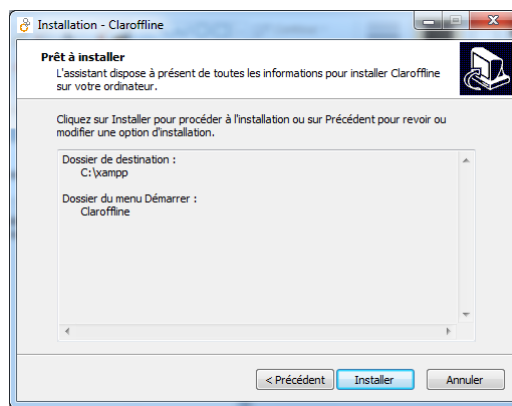


FIGURE C.7 – Etape 6 du wizard d'installation

```
$ sudo chmod 777 -R ~/Claroline/app/cache/ ~/Claroline/app/config/  
~/Claroline/app/logs/ ~/Claroline/files/  
~/Claroline/templates/ ~/Claroline/web/
```

L'étape suivante consiste à créer un lien symbolique entre votre dossier contenant les sources et le répertoire appelé par le serveur apache. En d'autres mot, nous indiquons à Apache où se trouve le code de Claroffline. Pour ce faire, entrez la commande :

```
$ sudo ln -s ~/Claroline/ /var/www/html/
```

Veuillez enfin démarrer votre navigateur et entrer l'adresse : http://localhost/claroline/web/app_offline.php. Vous arriverez alors sur une page effectuant la fin de l'installation de Claroffline (Figure C.8). Dès que celle-ci est terminée (cela peut prendre quelques minutes), vous arriverez sur la page de configuration de votre application Claroffline (Figure C.9).

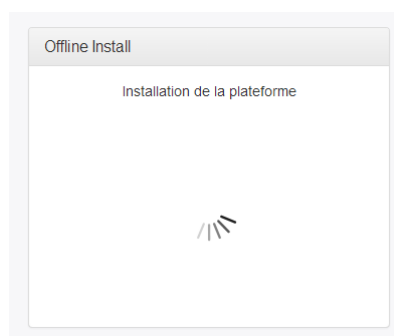


FIGURE C.8 – Vue du chargement de fin d'installation de Claroffline

La documentation

Si vous rencontrez des problèmes avec votre installation de PHP, d'Apache ou de MySQL nous vous invitons à vous rendre sur les documentations d'installation :

PHP : <http://doc.ubuntu-fr.org/php>

Apache : <http://doc.ubuntu-fr.org/apache2#installation>

MySQL : <http://doc.ubuntu-fr.org/mysql>

C.2.3 Autres systèmes d'exploitation

Si vous n'utilisez ni Windows, ni Linux, nous n'avons pas prévu d'installateur intégrant les différents composants requis pour faire fonctionner Claroffline. Rassurez-vous, Claroline et son plugin OfflineBundle ne nécessitent pas de pré-requis très spécifique. Nous vous invitons à lire le Chapitre discutant de l'installation [9 page 71](#) afin de vous informer sur les différents composants nécessaire à la plateforme. Il est possible que vous soyez confronté à des concepts plus techniques. Nous nous excusons pour les désagréments occasionnés.

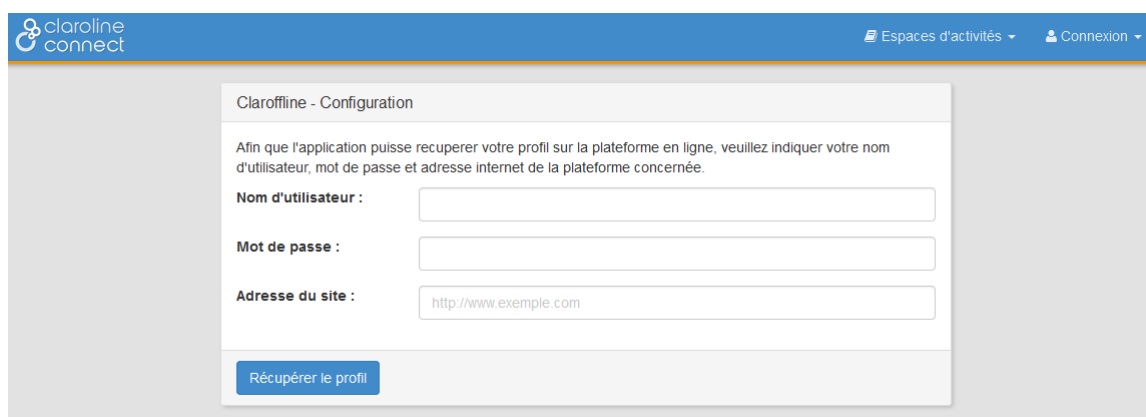
C.3 Configurer son compte

Une fois la plateforme installée et démarrée, la première chose que vous serez amené à faire est récupérer votre compte. Au démarrage, vous serez donc redirigé vers un formulaire (voir Figure [C.9](#)) vous demandant votre nom d'utilisateur, votre mot de passe et le serveur Claroline à contacter. Dans le champ contenant le serveur à contacter, vous devez entrer l'adresse internet de la plateforme Claroline avec laquelle vous souhaitez vous synchroniser. Il s'agit de l'adresse que vous entrez dans votre navigateur pour accéder à Claroline. Cette adresse doit référencer la page d'accueil et non une page spécifique. Si l'adresse entrée est incorrecte, vous risquez de ne pas réussir à récupérer votre compte et voir un message d'erreur s'afficher spécifiant que nous n'avons pas pu contacter le serveur que vous référencez.

Une fois votre compte correctement chargé, vous êtes redirigé sur la vue principale de notre bundle. Cette vue, qui sera la même que pour la synchronisation est illustrée sur la Figure [C.10](#). Lorsque vous êtes redirigé sur cette page, seule votre compte a été récupéré, notre bundle n'a pas encore récupéré vos espaces personnels et les différentes ressources qui les composent. Dès lors, nous vous invitons à effectuer immédiatement une synchronisation afin de récupérer l'ensemble des ressources de chacun des espaces d'activités auxquels vous êtes inscrit.

C.4 Se synchroniser

Comment faire pour se synchroniser ? Se synchroniser, c'est-à-dire envoyer le travail effectué hors-ligne sur le serveur distant et récupérer le nouveau contenu disponible en ligne, peut se faire via la page du OfflineBundle à laquelle on accède par le menu dépliant de gauche



The screenshot shows the 'Claroline - Configuration' window. At the top, there's a blue header with the 'claroline connect' logo on the left and 'Espaces d'activités' and 'Connexion' on the right. The main content area has a title 'Claroline - Configuration' and a paragraph: 'Afin que l'application puisse récupérer votre profil sur la plateforme en ligne, veuillez indiquer votre nom d'utilisateur, mot de passe et adresse internet de la plateforme concernée.' Below this are three input fields: 'Nom d'utilisateur :', 'Mot de passe :', and 'Adresse du site :'. The 'Adresse du site :' field contains the text 'http://www.exemple.com'. At the bottom of the form is a blue button labeled 'Récupérer le profil'.

FIGURE C.9 – Vue de configuration affichée à l'aide du Listener lors du premier démarrage

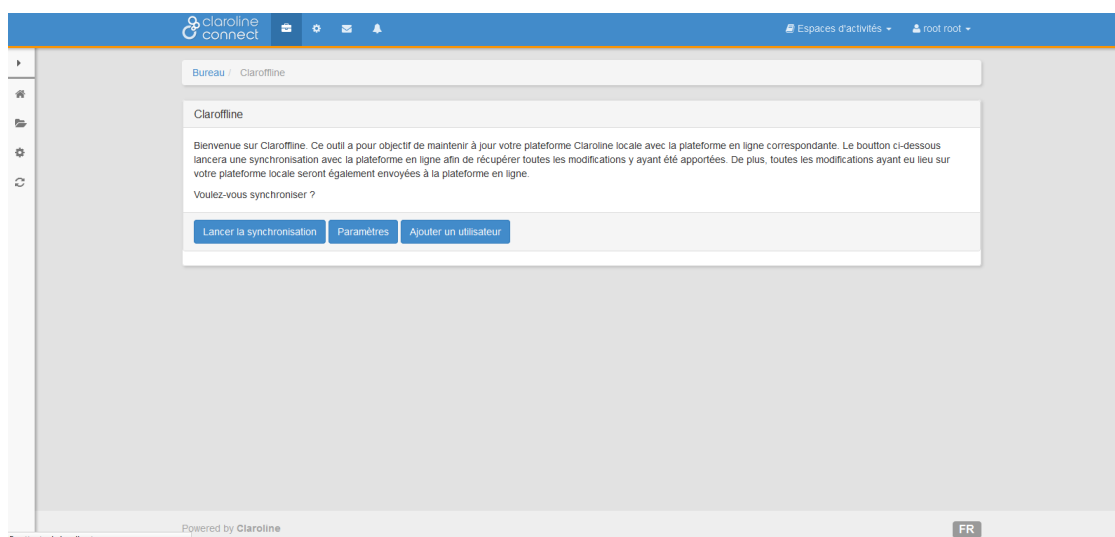


FIGURE C.10 – Vue d'accueil du OfflineBundle

(voir Figure C.11). Lorsque que vous êtes sur la vue de synchronisation (voir Figure C.10), cliquez sur le bouton, "Lancer la synchronisation" et celle-ci s'effectuera. Vous verrez alors apparaître un tag d'information "Veuillez patienter" s'afficher en haut de page. La synchronisation peut prendre un temps potentiellement long, si votre connexion est de faible qualité ou que vous devez transférer une grande quantité de ressources.

Enfin, quand la synchronisation s'est terminée sans échec, la vue se met à jour et vous bénéficiez d'un aperçu des ressources qui ont été créées et de l'espace d'activités auquel elles appartiennent.

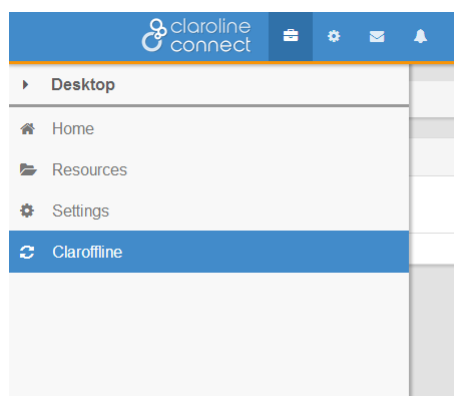


FIGURE C.11 – L'accès vers OfflineBundle depuis le menu dépliant de gauche

C.5 S'inscrire à un nouveau cours

Comment procéder pour vous inscrire à un nouveau cours et récupérer l'ensemble de ses ressources sur votre Claroline ? L'inscription à un nouveau cours s'effectue en passant par la version connectée du Claroline avec laquelle vous vous synchronisez. Nous vous invitons donc à vous connecter via un navigateur web sur la plateforme de référence. Une fois connecté sur le site, inscrivez-vous au nouveau cours de votre choix. Une fois que ceci est fait, fermez le site Internet et ouvrez votre plateforme Claroline. Après exécution, vous devriez avoir récupéré l'ensemble des ressources de ce nouvel espace d'activités.

C.6 Cas particulier de synchronisation

Que se passe-t-il en cas d'échec de la synchronisation ? Si la synchronisation échoue ou qu'elle est interrompue pendant qu'elle s'exécutait, elle reprendra la fois suivante là où elle s'est arrêtée. Ceci est fait dans le but de minimiser les données qui devraient être retransférées entre l'application Claroline et le serveur distant.

Par ailleurs, vous devez savoir que nous enregistrons comme référence la date et l'heure à laquelle le processus de synchronisation est lancé. Cette date nous sert à déterminer les ressources qui sont transférées. Dans la pratique, cela signifie que si vous initiez le transfert à une date donnée, que la synchronisation est interrompue et est reprise ultérieurement, le travail que vous aurez fait durant l'interruption ne sera pas inclus dans la synchronisation que vous venez d'achever. Pour qu'il soit transmis, vous devez réinitier une synchronisation avec le serveur distant.

Voici une explication par l'exemple :

1. Le 1er janvier, vous décidez de vous synchroniser. Pour une raison particulière la syn-

chronisation est interrompue (coupure du réseau, perte d'alimentation électrique, arrêt inopiné de l'ordinateur,...)

2. Le 2 janvier, vous travaillez sur votre application Claroline. Vous postez vos questions sur le forum et ajoutez vos fichiers en ligne.
3. Le 3 janvier, vous avez à nouveau un accès Internet vous permettant de vous synchroniser. La synchronisation reprendra alors là où elle s'était arrêté le 1er janvier et lorsqu'elle sera terminée, la date du 1er janvier sera enregistrée en mémoire comme date de dernière synchronisation. Pour que votre travail du 2 janvier soit effectivement envoyé en ligne, vous devez alors procéder à une nouvelle synchronisation.

C.7 Ajout d'un autre utilisateur

Depuis la vue de synchronisation (Figure C.10), un bouton AJOUTER UN UTILISATEUR est disponible. Si vous cliquez dessus, vous serez alors redirigé vers une vue semblable à celle qui vous a été présentée lors de la première connexion (Figure C.9). Vous pouvez alors permettre à un autre utilisateur de récupérer son compte. De manière similaire à la façon dont vous avez procédé, entrez les identifiants de ce second utilisateur ainsi que l'adresse du serveur Claroline distant à contacter (celui-ci doit être le même que le vôtre). Ce deuxième compte sera alors recréé et vous pourrez vous connecter et vous synchroniser avec ces mêmes identifiant sur l'application Claroffline.

C.8 Le panneau de paramètres

Depuis la vue de synchronisation (Figure C.10), un troisième bouton, PARAMÈTRES est disponible. Si vous cliquez dessus, vous arrivez sur une vue (voir Figure C.12) vous permettant d'éditer l'adresse de la plateforme avec laquelle vous vous synchronisez. Ceci pourrait être intéressant si votre plateforme change d'adresse, ainsi vous n'auriez pas à retélécharger toute la plateforme et les synchronisations. Attention toutefois, il est à noter que vous ne pourrez pas vous synchroniser avec une plateforme Claroline différente que celle avec laquelle vous avez initié la synchronisation. Si vous souhaitez changer de plateforme de référence, vous devez désinstaller et réinstaller votre application Claroffline.

C.9 Les messages d'erreurs

Voici la liste des différents messages d'erreurs que vous pourriez voir apparaître ainsi que leur signification :

- "L'authentification a échoué, le nom d'utilisateur et/ou le mot de passe semble(nt) être erroné(s)." Cette erreur signifie que les identifiants que vous avez entrés n'ont pas pu être confirmés sur la plateforme Claroline que vous essayez de contacter. Vérifiez donc que votre nom d'utilisateur est correct et assurez-vous de ne pas vous être trompé dans votre mot de passe.

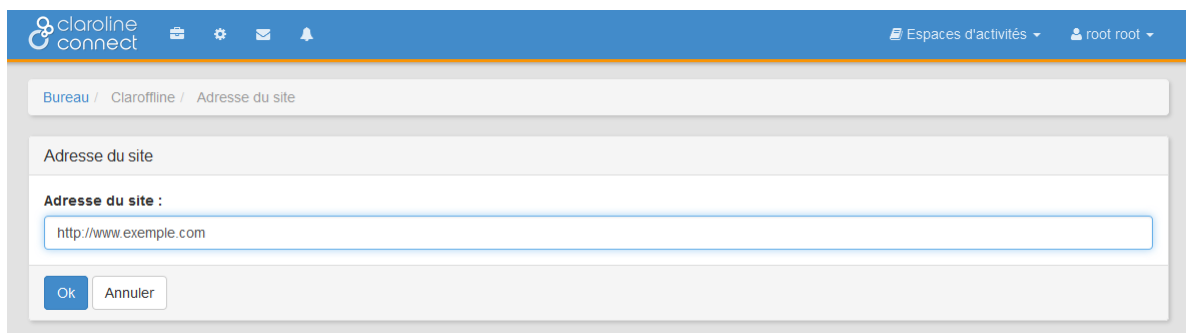


FIGURE C.12 – Vue d'édition des paramètres

- *"Une erreur est survenue du côté du serveur, veuillez réessayer dans quelques instants"*. Cette erreur signifie que le serveur distant a rencontré un problème ; il peut s'agir notamment d'une erreur liée à des erreurs de transfert ou à votre synchronisation. Comme suggéré dans le message, essayez de réexécuter la synchronisation un peu plus tard afin de voir si l'erreur persiste.
- *"Impossible d'atteindre le serveur. [...]"* Cette erreur signifie que le serveur est injoignable, peut-être rencontrez-vous des problèmes avec votre accès Internet ou le site est-il en maintenance ou indisponible pour le moment. Si le problème persiste, assurez-vous que le serveur distant avec lequel vous vous synchronisez n'a pas changé d'adresse, auquel cas vous pouvez modifier l'URL grâce au panneau de paramètres expliqué précédemment.
- *"Une erreur est survenue en contactant le serveur distant. Veuillez réessayer plus tard."* Ce message se produit lorsque le transfert a rencontré un problème dû à un timeout. C'est-à-dire que l'application Claroline n'a pas reçu de réponse du serveur distant. Ce problème peut être rencontré si votre connexion Internet est particulièrement lente ; dans ce cas, voyez avec le responsable de votre plateforme Claroline afin de régler la taille des fragments transférés entre les deux plateformes. L'ensemble de la documentation technique sur ce problème se trouve au chapitre 7 page 49 de ce document.
- *"Le compte de cet utilisateur a déjà été récupéré !"* Cette erreur survient lorsque vous essayez de récupérer un compte sur votre application Claroline mais que celui-ci est déjà présent. Vous ne devez de ce fait plus le récupérer et vous pouvez directement l'utiliser.

C.10 Désinstallation

Pour désinstaller l'application sur un ordinateur équipé de Windows, rendez-vous dans le menu démarrez, cliquez sur le programme *Claroline* puis sur *désinstaller*. Enfin, confirmez que vous souhaitez désinstaller l'application.

C.11 Changement de mot de passe

Maintenant que vous utilisez l'application Claroffline, vous vous demandez peut-être comment sont gérés vos changement de mot de passe ? Ce que vous devez savoir c'est que les mots de passe ne sont pas synchronisés entre l'application Claroffline et le serveur distant. Concrètement si vous modifiez votre mot de passe sur une des deux plateformes, votre mot de passe ne sera pas changé sur l'autre. Vous devrez alors vous identifier sur chacune des plateformes (l'application Claroffline et le serveur distant) avec leur propre mot de passe. Toutefois rien ne vous empêche de changer votre mot de passe sur les deux plateformes et de choisir un nouveau mot de passe identique. Il vous suffira à ce moment de changer votre mot de passe sur une plateforme et d'effectuer la même opération une seconde fois sur l'autre plateforme.

ANNEXE D

Code Source

Le code source du *OfflineBundle* est accessible en ligne sur le site *Github*. *Github* est le service d'hébergement du code source web utilisé comme dépôt pour le projet Claroline.

L'URL des sources de notre bundle est : <https://github.com/vVYou/OfflineBundle>.

Une version électronique en PDF de ce mémoire y est également disponible.

L'installateur Windows et Linux sont disponibles sur le CD accompagnant ainsi que sur la page <http://claroline.net/type/claroffline> du site internet de Claroline.