Auteur : Keil Laurent 2012-2013
Version 1.1 du module "Ccampus"-côté serveur

Documentation : guide du programmeur

Client : Jean-Noël Colin

1. Introduction.

Ceci est un projet d'**application mobile d'e-learning**, plus précisément pour l'instant une extension de la plateforme WebCampus sur le système d'exploitation pour mobile Androïd. Il contient deux parties bien distinctes, une **partie côté client** (non abordé ici), traitant de l'interface utilisateur et de l'accès aux données récoltées par la **partie côté serveur** abordée dans cette documentation. L'intérêt de cette séparation est de permettre une abstraction concernant l'utilisation des données.

Dans ce projet, nous nous attardons premièrement à **l'authentification** sur un site d'e-learning, WebCampus dans ce cas-ci, à la vérification du client et la capture d'un **cookie** et nous pouvons dans un second temps récupérer les **données**, que l'on parse dans un **format JSON**, par cette connexion au serveur qu'on peut ensuite restituer à la partie client dans un format plus adapté au système d'exploitation client, Androïd dans le cadre de notre projet.

Au terme du projet, le client peut accéder à une interface sur Androïd qui lui permet d'accéder à des données « virtuelles » (par un mock, détaillé par la suite), lui simulant un accès à WebCampus. L'authentification côté serveur a été crée sur ChouetteCampus mais n'est pas fonctionnel sur Androïd... Le projet réuni ne possède qu'une lecture des données, aucune création ou modification de données n'est encore permise.

2. Structure du projet.

Pour une vue globale du projet, référez-vous au dossier « uml diagram » ou l'image en fin de document. Il est important de comprendre la hiérarchie de l'api. Une documentation détaillée est également accessible dans le dossier « javadoc ».

Je vais ainsi détailler brièvement chaque package de mon projet personnel, c'est-à-dire ceux correspondant à la partie non-client, et leur utilité...

2.1. Package main :

Il est le package principal utilisé pour démarrer l'application.

Main est la classe principale, elle permet dans le cas d'une utilisation interne au projet individuel de lancer un exemple de lecture des données soit en mode mock*, soit en mode WebService**.

Example propose un exemple d'utilisation des services de l'application en mode mock* et un exemple en mode WebServices**.

Parameters est utilisé pour déclarer les paramètres de l'application : mode MOCK* ou WS**

FactorySession lance une session de l'application en mode MOCK* ou WS** suivant les paramètres définis dans *Parameters.*

***Mock :** objet simulant le comportement d'objet réel de manière contrôlée, permettant de tester le comportement du programme sans avoir un accès aux objets réels.

Dans notre cas, pour ne pas devoir se connecter en ligne à un e-learning pour récolter des données, on crée un mock qui simule ces données, afin de tester le comportement de notre application sans données réelles.

****WS = WebServices :** Cette alternative donne un accès serveur à l'e-learning, pour l'instant ChouetteCampus. Elle offre donc une session réelle et on peut donc manipuler des données réelles avec l'application.

Attention : L'accès n'est permis sur Androïd que par le mock et l'application se déploie seulement sur ChouetteCampus.

2.2. Package api :

Ensemble d'interfaces de fonctionnalités consommant les WebServices (ou les données du mock) auxquelles le module client peut accéder. Je vais détailler chacune des interfaces afin d'en comprendre leur fonctionnement.

2.2.1. ***Session*** est la première interface sur laquelle la partie cliente accède. Un objet Session est alors crée, que ce soit en mode *mock* ou *ws*. Cet objet permet d'accéder à d'autres objets et outils de l'application. Vous pouvez vous référer à la javadoc pour une compréhension détaillée de chacune des classes. L'authentification sera expliquée dans le package *impl*.

Chaque méthode : *getUserData, getUpdates et getCourses* renvoie un objet contenant des informations particulière sur l'utilisateur de la session en cours.

Le principe général est de fournir des méthodes permettant d'obtenir un objet comportant les informations souhaitées.

Exemple : L'utilisateur devra d'abord se logger en fournissant son username et password par la méthode *getSession* de *FactorySession*. On obtient alors un objet *Session* qui permet d'accéder à l'objet *Courses* qui contient 2 méthodes *getCourse* auxquelles on doit fournir un identifiant du cours désiré pouvant être obtenu par la méthode *getAllCoursesID* ou *getAllCoursesSyscode*. L'objet *Course* renvoyé permet ensuite d'obtenir des annonces et des cours de la même manière. La classe *Example* du package *main* offre un exemple d'utilisation pratique.

2.2.2. L'interface ***UserData*** permet d'obtenir des informations séparément sur l'utilisateur.

2.2.3. L'interface ***Updates*** permet d'obtenir des informations sur les updates pour l'utilisateur connecté depuis sa dernière connexion. Elle contient une méthode *nbUpates* étant le nombre d'update de l'utilisateur (cfr. interface *Update*) et *getUpdatesTab* qui renvoie un tableau de *nbUpates* objets *Update.*

2.2.3.1. ***Update*** contient des informations séparément sur une modification particulière d'une ressource d'un module d'un cours de l'utilisateur connecté depuis sa dernière connexion.

2.2.4. L'interface ***Courses*** permet d'obtenir des informations sur les cours de l'utilisateur. Son objectif est de fournir un objet *Course* approprié par une des deux méthodes, en fournissant deux méthodes renvoyant des tableaux définissant chaque cours disponible. La méthode à utiliser est au choix de l'utilisateur.

2.2.4.1. ***Course*** permet d'obtenir des informations séparément sur un cours particulier de l'utilisateur. Entre-autre, l'interface permet d'obtenir 3 objets qui contiennent différentes informations sur le cours, plus précisément un objet *CourseTool* détaillant les outils disponibles du cours en question(pour ce projet seulement les annonces et les documents), un objet *Annonces* et un objet *Docs*.

2.2.4.1.1. ***CourseTool*** permet d'obtenir des informations séparément sur les outils disponibles et nouveautés du cours concerné de l'utilisateur de la session.

2.2.4.1.2. ***Annonces*** permet d'obtenir des informations sur les annonces d'un cours particulier de l'utilisateur. L'interface, de la même manière que l'interface *Courses*, de fournir deux tableaux définissant chaque objet annonce pouvant être récupérer par une des deux méthodes *getAnnonce.*

2.2.4.1.2.1. L'interface ***Annonce*** contient des informations sur une annonce particulière d'un cours particulier de l'utilisateur.

2.2.4.1.3. ***DocsAndFolders*** permet d'obtenir des informations sur les documents et dossiers de documents d'un cours particulier de l'utilisateur. Elle peut fournir un tableau des noms des dossiers et un des noms de documents du cours concerné. L'utilisateur peut alors obtenir l'objet représentant ce document ou dossier par les deux méthodes associées.

2.2.4.1.3.1. L'interface ***Doc*** contient des informations sur un document particulier d'un cours particulier de l'utilisateur.

2.2.4.1.3.2. L'interface ***Folder*** permet d'obtenir des informations sur un dossier particulier d'un cours particulier de l'utilisateur.

2.3. Package impl.mock :

Implémentation de l'api par un mock, c'est-à-dire en fournissant des données tests à l'application. Dans ce cas-ci, il n'y a pas d'authentification. L'utilisation se fait de la

manière décrite plus haut dans le package *api*.

Dans chacun des **contructeurs** d'objet : *UserDataMock, UpdatesMock, CourseToolMock, DocsAndFoldersMock* et *AnnoncesMock*, on construit un objet au format JSON en y plaçant des données tests. Excepté *UserDataMock* et *CourseToolMock*, on remplit un tableau contenant des identifiants de l'objet concerné afin que l'utilisateur puisse récupérer ces identifiants par une méthode spécifique. Les autres méthodes sont des utilitaires ou des **getters** permettant de récupérer de l'information d'un objet particulier. Dans certains cas, cette information est un objet qui est implémenté de la même manière.

Le principe du format **JSON** est de pouvoir récupérer les éléments d'une unité, un cours par-exemple, séparément et pouvoir obtenir cette information particulière par un getter fourni à l'utilisateur.

Attention : Chaque classe du packaque mock et impl.ws doit implémenter à la fois les interfaces de l'api mais aussi Serializable permettant la sérialisation d'objet sur Androïd.

2.4. Package impl.ws :

Implémentation de l'api par des webservices, c'est-à-dire par des données provenant d'un **serveur**, dans notre cas « *chouetteCampus* ». Le principe est le même que pour le mock, la différence est qu'on ne place plus aucune donnée test avec le constructeur mais on va chercher cette donnée sur le serveur et on y extrait les informations grâce à un **parser** du format JSON et par une **authentification** au serveur. Nous allons donc détailler les outils qui permettent cela...

Principe d'authentification :

Le constructeur *SessionWS* fait appel a la classe *Claroline* du package *impl.ws.services* qui authentifie le client par la classe *validation* et *Register*. La méthode *RegisterURL* permet alors de se connecter au serveur grâce aux données du client, crée une session et renvoie les informations d'un cookie.

closeSession coupe la connexion et supprime le cookie.

2.5. Package impl.ws.services :

La classe ***Claroline*** est celle qui est appelée par le constructeur ***SessionWS.*** Elle contient la méthode ***RegisterThread*** qui place les champs user et password, ***getToken*** fournissant le *token* (identifiant du client connecté pendant une session active), ***run*** qui capture le *token* en faisant un appel à la classe ***Register*** et la méthode ***login*** qui permet l'authentification du client après une validation par la classe ***Validation*** et renvoie le *token* du client.

C'est la classe ***Register*** qui permet la connexion au serveur indiqué en entrée par

une *url* dans la méthode ***RegisterURL***. Pour récupérer le client et son cookie, on envoie une requête http *post* au serveur avec l'url et le *login* et *password* du client comme paramètres.

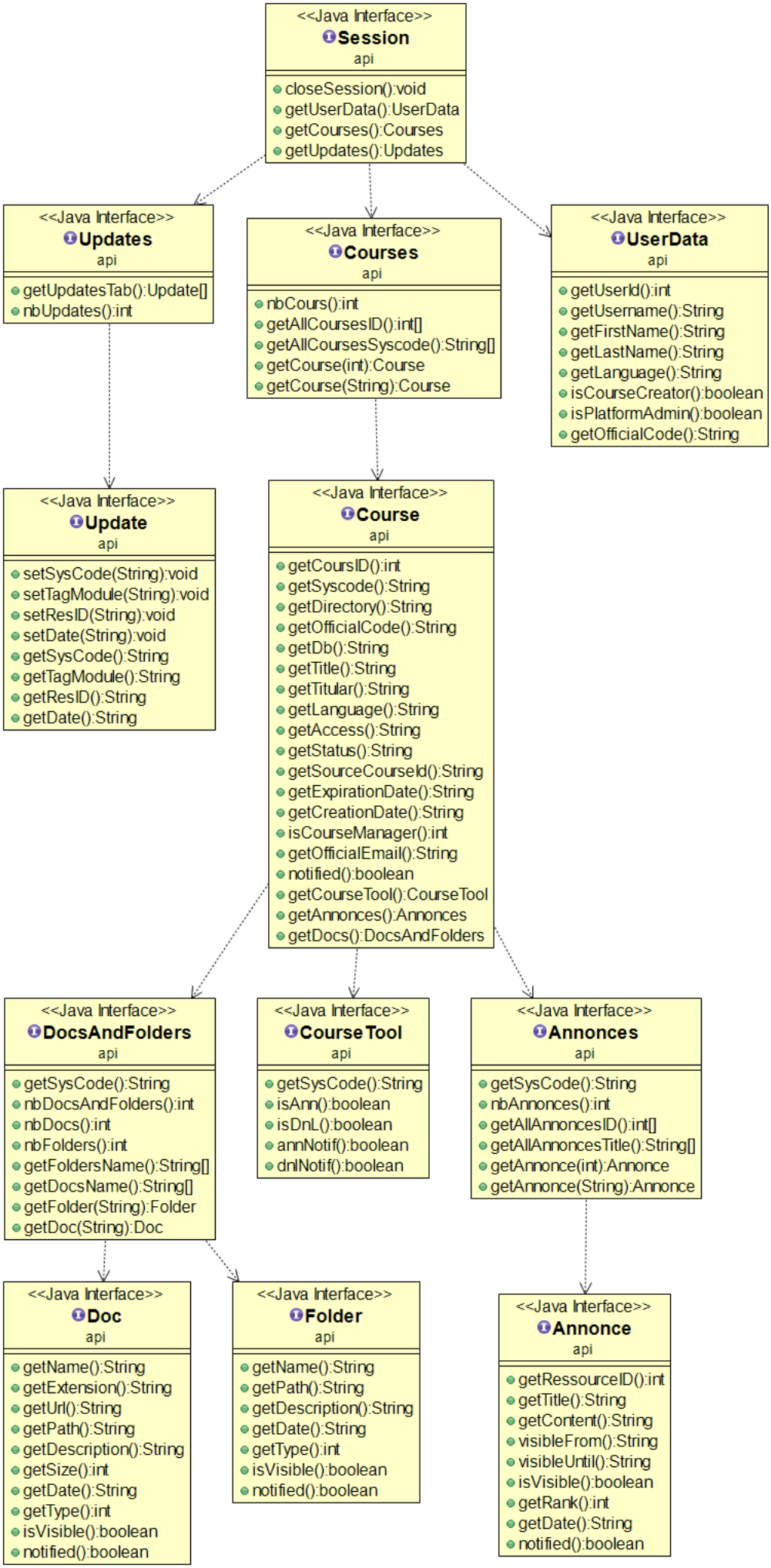
Cependant, puisque *ChouetteCampus* possède une connexion sécurisée SSL avec un certificat (*https*), nous devons avant cela ajouter des propriétés « Security.*setProperty()* ». On fait alors un appel à la classe ***LazySSLSocketFactory*** permettant d'automatiser la connexion SSL sécurisé avec un serveur avec la classe ***LazyTrustManager*** qui permet d'accepter tous les certificats d'un client sur un serveur.

JsonConvertor est une classe appelée dans le but de parser le format JSON du contenu d'un fichier à partir d'une url auquel l'utilisateur a le droit de connexion. Après s'être donc authentifié, on peut faire appel à 2 méthodes de cette classe permettant de renvoyer un objet JSON « ***getJSONObjectFromUrl*** » ou un tableau JSON «***getJSONArrayFromUrl*»**. Ces méthodes passent par une vérification de l'authentification du client, s'il est bien authentifié on envoie une requête *get* au serveur et on enregistre chaque élément du texte dans un objet ou tableau JSON. Chaque élément pourra ultérieurement être retraduit en texte si besoin de l'utilisateur dans les classes de *impl.ws*.

2.6. Package test :

Tests unitaires des différentes méthodes critiques du mock.

Concernant une suite possible du projet, la connexion avec Androïd est encore à réaliser (à voir s'il faut apporter des modifications dans cette partie ou celle côté client...) bien que les fondations sont déjà établies. D'autres modules que les documents et annonces devront être ajouté et une possibilité de création/modification de données devra être permise.


<<Java Interface>>
Session
api
closeSession():void
getUserData():UserData
getCourses():Courses
getUpdates():Updates
<<Java Interface>>
Updates
api
getUpdatesTab():Update[]
nbUpdates():int
<<Java Interface>>
Courses
api
nbCours():int
getAllCoursesID():int[]
getAllCoursesSyscode():String[]
getCourse(int):Course
getCourse(String):Course
<<Java Interface>>
UserData
api
getUserId():int
getUsername():String
getFirstName():String
getLastName():String
getLanguage():String
isCourseCreator():boolean
isPlatformAdmin():boolean
getOfficialCode():String
<<Java Interface>>
Update
api
setSysCode(String):void
setTagModule(String):void
setResID(String):void
setDate(String):void
getSysCode():String
getTagModule():String
getResID():String
getDate():String
<<Java Interface>>
Course
api
getCoursID():int
getSyscode():String
getDirectory():String
getOfficialCode():String
getDb():String
getTitle():String
getTitular():String
getLanguage():String
getAccess():String
getStatus():String
getSourceCourseId():String
getExpirationDate():String
getCreationDate():String
isCourseManager():int
getOfficialEmail():String
notified():boolean
getCourseTool():CourseTool
getAnnonces():Annonces
getDocs():DocsAndFolders
<<Java Interface>>
DocsAndFolders
api
getSysCode():String
nbDocsAndFolders():int
nbDocs():int
nbFolders():int
getFoldersName():String[]
getDocsName():String[]
getFolder(String):Folder
getDoc(String):Doc
<<Java Interface>>
CourseTool
api
getSysCode():String
isAnn():boolean
isDnL():boolean
annNotif():boolean
dnlNotif():boolean
<<Java Interface>>
Annonces
api
getSysCode():String
nbAnnonces():int
getAllAnnoncesID():int[]
getAllAnnoncesTitle():String[]
getAnnonce(int):Annonce
getAnnonce(String):Annonce
<<Java Interface>>
Doc
api
getName():String
getExtension():String
getUrl():String
getPath():String
getDescription():String
getSize():int
getDate():String
getType():int
isVisible():boolean
notified():boolean
<<Java Interface>>
Folder
api
getName():String
getPath():String
getDescription():String
getDate():String
getType():int
isVisible():boolean
notified():boolean
<<Java Interface>>
Annonce
api
getRessourceID():int
getTitle():String
getContent():String
visibleFrom():String
visibleUntil():String
isVisible():boolean
getRank():int
getDate():String
notified():boolean