UDS - 2009
Documentation développeur

Dispositif de diffusion de cours en ligne
AudioVideoCours (Univ-R AV)

http://audiovideocours.u-strasbg.fr
http://univ-rav.u-strasbg.fr

Sommaire

I. Présentation du projet Audiovidéocours

1. Description fonctionnelle du projet

Audiovideocours est un dispositif créé et déployé par l'Université de Strasbourg. Ses objectifs sont à la fois de favoriser la mise à disposition de nouveaux contenus pédagogiques et d'encourager l'utilisation des TIC dans les méthodes d'enseignement. Il permet aux enseignants d'enregistrer leurs cours sous forme audio ou vidéo, en retransmettant leurs supports de cours informatisés.
Le système est constitué d'un client d'enregistrement ainsi que d'un site permettant le classement et la visualisation des enregistrements.

- Le client d'enregistrement permet aux enseignants de réaliser très simplement, sans l'intervention d'un technicien, différents types d'enregistrements de leurs cours :
    - Des enregistrements en direct dans les salles et amphithéâtres automatisés de l'université grâce au dispositif Univ-R2A.
    - Des enregistrements en différé possibles dans les salles et amphithéâtres automatisés, mais aussi sur leur ordinateur personnel.
- Le site web permet d'une part de visualiser les cours diffusés en direct. D'autre part, les enregistrements en différé sont archivés et classés sur le site sous forme de supports de cours qui mêlent son ou vidéo et captures d'écran synchronisées. Il est possible de lire ceux-ci à travers une interface de visualisation qui permet de naviguer dans la vidéo ou dans la bande son et dans les captures d'écran.

Le site ayant été placé sous licence GPL et mis en partage, il est susceptible d'être utilisé et modifié par d'autres établissements.

2. Fonctionnalités de la solution

Voici une liste des fonctionnalités du site web :

- Recevoir les données audio et vidéo envoyées depuis les clients
- Diffuser les contenus en direct et en différé sur la plupart des systèmes d'exploitation du marché
- Proposer des contenus publics accessibles à tout le monde
- Proposer des contenus privés protégés par un code d'accès
- Proposer une fonction de recherche et de filtrage des cours
- Proposer une fonction de recherche des cours par tags
- Disposer d'un dispositif de téléchargement des cours audio et vidéo
- Présenter les derniers ajouts de cours sous forme de flux rss, podcasting
- Etre disponible en plusieurs langues, en donnant la possibilité de rajouter facilement une nouvelle langue
- Proposer plusieurs thèmes visuels, en donnant la possibilité d'en rajouter de nouveaux facilement
- Contenir des outils d'administration permettant de gérer la base de données
- Etre interopérable avec l'application Univ-R développée par l'ULP Multimédia
- Disposer d'une documentation technique et d'une aide utilisateur
- Disposer d'un fichier de configuration pour définir les paramètres de l'application tels que le dépôt des ressources

3. Choix technologiques

Les Servlet et les JSP ont été choisies comme solution pour la création de l'application web dynamique afin de garder une certaine homogénéité au sein de l'environnement technique de l'ULP Multimédia. La gratuité de cette solution est un avantage par rapport à une solution concurrente payante.

PostgreSQL a été choisi comme système de gestion de base de données, toujours pour garder une homogénéité dans les projets de l'entreprise et profiter de l'avantage de la gratuité.

L'application web dynamique et la base de données sont hébergées sur un même serveur Linux.

JAVA 1.5

POSTGRES 8.1

TOMCAT 5.5, APACHE 2.2,

SERVEUR LINUX (DEBIAN ETCH)

4. Autres programmes requis

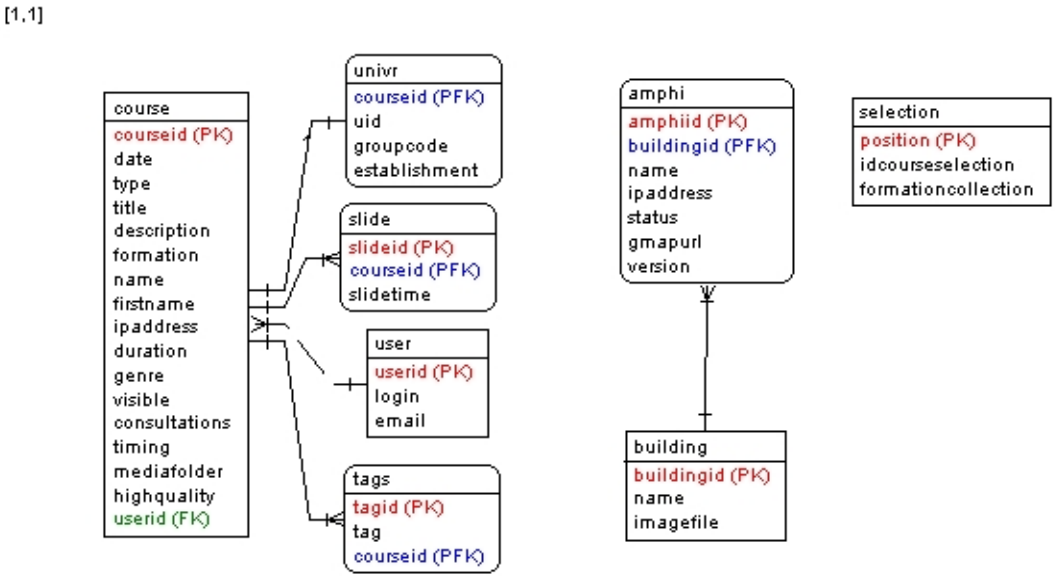
Pour pouvoir mettre en place le site AudioVideoCours, il vous faut installer un certain nombre de logiciels, programmes, et librairies :

- Le serveur web *Apache 2*
- Le connecteur *Tomcat* pour *Apache*
- Le *Java Runtime Environment (JRE)* version 1.5
- Le SGBD PostgreSQL
- Un serveur FTP (proFTPD)
- Pour encoder les fichiers podcast, il faut les programmes *Oggenc 1.0.2*, *Lame 3.97-0.0sarge1*, *Mp32ogg 0.11-7*, *Mplayer et mencoder 1.0-rc1svn20070225-0.3etch1, lltag 0.13.1, ffmpeg 20070329-0.0etch1(debian-multimedia version complete)*
- Pour créer et décompacter des archives, les programmes *zip* 2.32-1 et *unzip 5.52-9etch1*
- Le programme *MP3info 0.8.4-9.2* pour afficher et ajouter des tags MP3
- *Python 2.5* (et *python-dev*)
- La librairie *PIL* (Python Imaging Library : (http://www.pythonware.com/products/pil/) pour lancer le script de vérification des miniatures.

*L*a librairie Python pour créer des PDF (www.reportlab.org) pour lancer le script de création des PDF des diapos de cours.*Serveur fms 3 sous linux, licence illimité*

II. Eléments techniques

1. La base de données

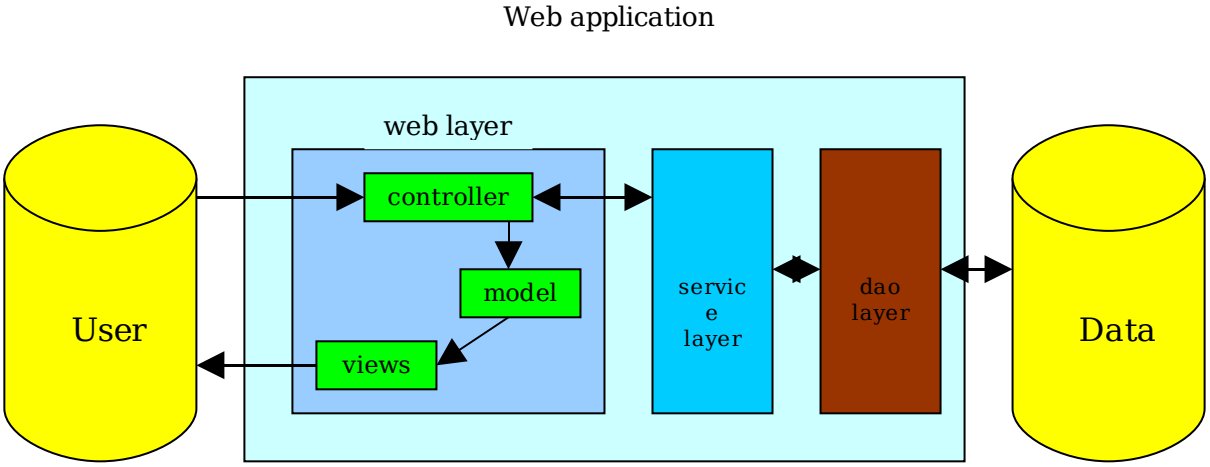


La base de données comprend donc 8 tables :

Toutes les informations associées à un cours sont enregistrées dans la table *[course]*.
A chaque cours correspond un certain nombre de diapositives de cours ayant un temps de synchronisation distinct. C'est le rôle de la table *[slide]* de gérer le nombre de diapositives avec le point de synchronisation de chacune.
Les informations complémentaires d'un cours qui le lient à l'application Univ-R sont enregistrées dans la table *[univr]*.
La table *[building]* permet de stocker les différents bâtiments du campus. A ceux-ci correspondent plusieurs amphithéâtres, représentés dans la table *[amphi]*.
La table [user] contient les informations des utilisateurs, la table [tags] correspond à la liste des tags (mots-clés) associés aux cours et la table [selection] correspond aux cours affichés dans les listes « Sélection » et « Collection » de la page d'accueil.

2. Architecture de l'application

L'application utilise une architecture MVC 3-tier.



2.1. Description de l'architecture 3-tier

La couche *[dao]* s'occupe de l'accès aux données à partir de plusieurs sources : données persistantes au sein du SGBD, données sur le système de fichiers du serveur, données provenant de la base de données Univ-R pour l'intégration avec cette plate-forme.

La couche *[service]* implémente les algorithmes métier de l'application. Cette couche est indépendante de l'interface avec l'utilisateur.

La couche *[web]* est l'interface constituée par les pages web de l'application qui permet à l'utilisateur de la piloter et d'en recevoir les informations.

Ainsi, lorsque l'utilisateur fait une requête sur le serveur, il contacte l'interface utilisateur de la couche *[web]*. La demande est ensuite transmise de la couche *[web]* vers la couche *[service]*. La couche *[service]* demande ensuite les données dont elle a besoin à la couche *[dao]* et les retourne à la couche *[web]* pour former la réponse au client.

Les couches ont été conçues pour être au maximum indépendantes les unes des autres. Des interfaces Java ont été utilisées de sorte qu'une couche ne connaisse que les interfaces d'une autre couche et non les classes les implémentant. Ainsi si l'implémentation d'une couche change, elle n'aura pas d'incidence sur les autres couches, tant qu'on ne modifie pas la définition de l'interface.

2.2. Description de l'architecture MVC

C'est dans la couche *[web]* que l'architecture Modèle-Vue-Contrôleur prend place. Le Design Pattern MVC utilisé est appelé « Front Controller », c'est-à-dire un modèle à contrôleur unique.

Voici comment se déroule le traitement d'une demande du client dans cette architecture MVC :

1. Le client fait une demande au contrôleur qui constitue la porte d'entrée de l'application. Le contrôleur est assuré par une Servlet unique. Le paramètre identifiant l'action à réaliser est passé dans l'url de la Servlet, cela permet ensuite de lancer l'exécution d'une méthode interne.
2. Le contrôleur traite alors cette demande dans cette méthode interne en sollicitant l'aide de la couche *[service]*.
3. Le contrôleur choisit la vue à retourner au client. Il s'agit cette fois d'une page JSP qui va représenter les données au client. Le contrôleur va devoir fournir les données à afficher dans cette vue ; celles-ci forment le modèle de l'application. Ce modèle est placé dans le contexte de la requête (*request.setAttribute("clé", "valeur")*).
4. Le contrôleur demande à la vue de s'afficher. Une instruction *getServletContext().getRequestDispatcher("pageJSP").forward(request, response)* est alors utilisée.
5. Le générateur de vue utilise le modèle préparé par le contrôleur pour initialiser les parties dynamiques de la réponse.
6. Enfin, la réponse est envoyée au client.

L'application n'utilise pas de framework. L'architecture MVC 3-tiers est uniquement réalisée avec un simple JDK et les bibliothèques de bases du développement web.
Le choix de l'implémentation des couches *[service]* et *[dao]* doit toutefois être définie dans le contrôleur dans la couche *[web]*.

3. Organisation des sources

3.1. Le répertoire des sources Java

L'application contient 5 packages dans ses sources :

- *org.ulpmm.univrav.dao* : ce package correspond à la couche *[dao]* de l'application
- *org.ulpmm.univrav.entities* : ce package contient les objets métier partagés entre les différentes couches
- *org.ulpmm.univrav.language* : ce package contient les fichiers de langues utilisés par l'internationalisation
- *org.ulpmm.univrav.service* : ce package correspond à la couche *[service]* de l'application
- *org.ulpmm.univrav.web* : ce package correspond à la couche *[web]* de l'application

3.1.1. Les objets métier

Ils représentent les objets tels qu'ils sont stockés dans la base de données. Une classe a été créée pour chaque table :

- *Amphi*
- *Building*
- *Course*
- *Slide*
- *Univr*
- *Teacher (classe n'ayant pas de correspondance avec une table de la base mais utilisée pour simplifier l'affichage des enseignants)*
- *User*
- *Tag*
- *Selection*

3.1.2. La couche *[dao]*

La couche d'accès aux données tire partie de différentes sources de données. Pour chacune de ces sources une interface a été définie :

- *IDatabase* : c'est l'interface qui accède aux données du SGBD. L'utilisation d'une telle interface permet de changer l'implémentation de l'accès à la base de données sans modifier la couche *[service].*
- *IFileSystem* : c'est l'interface qui permet d'accéder aux données présentes sur le système de fichiers, donc de faire de la lecture-écriture sur celui-ci.
- *IUnivrDao* : c'est l'interface de l'accès aux données de la base Univ-R, grâce à l'utilisation d'une API utilisable par Univ-R AV.

Ces interfaces sont respectivement implémentées par les classes suivantes :

- *DatabaseImpl*
- *FileSystemImpl*
- *UnivrDaoImpl*

Des autres classes sont présentes dans la couche [dao] :

- La classe *DaoException* est une classe héritant de *RuntimeException* permettant d'identifier les erreurs qui se produisent dans la couche *[dao]*. Ce type d'exception est non contrôlé, ce qui évite de devoir signer les méthodes des interfaces avec des exceptions d'un type particulier.
- La classe *PgsqlAccess* est une classe utilisée par la classe *DatabaseImpl* et permettant de faire des requêtes sur une base de données PostgreSQL en fournissant un objet *DataSource*.
- L'interface *ISmil* est une interface utilisée par la classe *FileSystemImpl* définissant la création d'un fichier multimédia SMIL pour un cours.
- La classe *LocalAudioSmil1* est l'implémentation de l'interface *ISmil* permettant de créer des fichiers SMIL pour les cours audio, visualisables en local.
- La classe *LocalVideoSmil1* est l'implémentation de l'interface *ISmil* permettant de créer des fichiers SMIL pour les cours vidéo, visualisables en local.

- La classe *RemoteAudioSmil1* est l'implémentation de l'interface *ISmil* permettant de créer des fichiers SMIL pour les cours audio, visualisables à distance.
- La classe *RemoteVideoSmil1* est l'implémentation de l'interface *ISmil* permettant de créer des fichiers SMIL pour les cours vidéo, visualisables à distance.

3.1.3. La couche *[service]*

La couche *[service]* est constituée de plusieurs classes et interfaces :

- *IService* : il s'agit de l'interface de la couche *[service]*. Elle rassemble les méthodes des interfaces de la couche *[dao]*.
- *ServiceImpl* : cette implémentation de l'interface *IService* appelle les méthodes des interfaces de la couche *[dao]*. Les méthodes de cette classe sont synchronisées, de sorte qu'un seul thread à la fois puisse appeler une méthode.
- *CourseAddition, UnivrCourseCompletion* et *MediaUpload* sont trois classes utilisées par la classe *ServiceImpl* et qui héritent de la classe *Thread*. Elles sont chargées de lancer les opérations de création des nouveaux cours, qui peuvent être très longues. Cela permet de rendre la main à la couche *[web]* sans devoir attendre que tout le processus de création d'un cours soit fini.

3.1.4. La couche *[web]*

Le contrôleur : la classe *Application*

La méthode *[init]* de cette Servlet est appelée une seule fois lors de la première requête après le démarrage de l'application web. Un certain nombre d'initialisation sont effectuées lors de celle-ci :

- Chargement des paramètres de configuration de l'application présente dans le fichier de propriétés *univrav.properties*.
- Chargement des données du fichier *web.xml* permettant d'obtenir la source de données pour se connecter au SGBD.
- Création d'une instance de la couche *[service]*.
- Création des instances implémentant la couche *[dao]* et association à la couche *[service]* en lui donnant des références de ces instances.
- Génération des flux RSS des cours existants.

Après l'initialisation du contrôleur, les méthodes de celui-ci disposent d'une référence vers la couche *[service]* qu'elles vont utiliser pour exécuter les actions demandées par l'utilisateur.

Celles-ci sont toutes interceptées par la méthode *[doGet]* de la Servlet qui va ensuite les traiter en appelant la méthode privée correspondante, ou directement dans son propre corps si le traitement est succinct. Le paramètre permettant d'identifier cette action demandée par l'utilisateur est récupéré dans l'URL de la requête.
La méthode *[doGet]* est de plus utilisée pour gérer les sessions de l'utilisateur ainsi que les cookies : la méthode détermine si la session vient juste d'être créée ou non. Si c'est le cas, elle cherche les paramètres d'identification de la langue et du thème

du site dans les cookies. Si les cookies n'existent pas encore, elle en créé avec le thème par défaut et la langue du navigateur. Ces paramètres sont ensuite placés dans la session.

Dans les méthodes privées associées aux actions demandées par le client (ou directement dans la méthode *[doGet]*), le contrôleur va créer le modèle de l'application et le placer dans le contexte de la requête (*request.setAttribute("clé", "valeur")*). Le contrôleur va alors rediriger la requête vers la page JSP correspondante avec une instruction *getServletContext().getRequestDispatcher("pageJSP"). forward(request, response)*.

La classe *PaginationTag* définit un tag personnalisé pour gérer facilement la pagination au sein des pages JSP sans avoir à rajouter de nombreuses lignes de code dans celles-ci.

3.2. Le répertoire *WebContent* de l'application web

- le fichier *index.jsp* est le point d'entrée de l'utilisateur lorsqu'il demande la racine du site. Son rôle est de faire une redirection vers le contrôleur en lui demandant d'afficher la page d'accueil.
- le répertoire *admin* contient un unique fichier *index.jsp*. Celui-ci est le point d'entrée de l'interface d'administration. Il fait une redirection vers le contrôleur en lui demandant d'afficher la page d'accueil de l'interface d'administration.
- le répertoire *conf* contient le fichier .properties permettant de configurer l'application web.
- le répertoire *files* contient tous les fichiers autres que html nécessaires au site tels que les images, les codes javascripts, et les répertoires des thèmes contenant eux-même des fichiers image et des feuilles de style CSS.
- le répertoire *rss* est un répertoire qui sert à stocker les fichiers des flux RSS. Ceux-ci sont créés par l'application au cours de son fonctionnement, ce qui explique que le répertoire est vide.
- le répertoire *scripts* contient les scripts nécessaires à la création des cours.
- le répertoire *upload* contient un unique fichier *index.jsp*. Celui-ci est le point d'entrée de la page d'upload. Il fait une redirection vers le contrôleur en lui demandant d'afficher la page d'accueil de la page d'upload.
- le répertoire *WEB-INF* contient différents types de fichiers :
    - Le répertoire *lib* contient les fichiers .jar des librairies externes nécessaires à l'application.
    - Le répertoire *views* qui contient les pages JSP/JSTL de l'application. Le sous-répertoire *admin* contient les pages JSP de l'interface d'administration.
      Le sous-répertoire *myspace* contient les pages JSP de l'interface « mon espace ».
      Le sous-répertoire *include* contient des pages JSP représentant des éléments utilisés dans plusieurs pages et qui sont donc inclus dans d'autres pages, ou les pages JSP correspondant aux thickbox.

La page *exception.jsp* correspond à la page affichée lorsque l'application lance une exception non gérée.
La page *message.jsp* permet d'afficher des messages d'information ou d'erreurs contrôlées dans le site web.
Les autres pages JSP correspondent aux différentes fonctions de l'application.

- Les fichiers tld de définition des balises personnalisées
- Le fichier *web.xml* de l'application qui permet la configuration de la couche *[web].*
  - Dans ce fichier on définit que toutes les url se terminant par */avc/** seront traitées par la Servlet. C'est sous cette forme que les liens sur les pages JSP seront afin que le contrôleur puisse traiter chaque requête. Les url se terminant par */audiocours_v2/servlet/** sont aussi traitées dans un souci de compatibilité avec les anciennes versions des clients qui ne sont pas à jour.
  - On définit la page d'entrée par défaut (*index.jsp*) qui se trouve à la racine du dossier de l'application web et redirige vers la Servlet.
  - La page d'erreur par défaut *exception.jsp* qui est affichée lorsqu'une exception non gérée se produit est également définie.
  - Les paramètres d'accès à la source de données PostgreSQL définies dans le fichier *server.xml* de Tomcat y sont définies.
  - L'anglais y est défini comme langue par défaut de l'application pour la libraire d'internationalisation lorsque la langue de l'utilisateur n'a pas de fichier de traduction.
  - Les restrictions d'accès aux pages d'administration y sont définies, en relation avec les fichiers de configuration de Tomcat (*tomcat-users.xml*).
  - Les restrictions d'accès à la page d'upload y sont définies grâce à une authentification CAS, en configurant un filtre.

4. Technologies mises en œuvre et dépendances

4.1. Utilisation de taglibs dans les pages JSP

Les pages JSP contiennent un minimum de code Java pour ne pas le mélanger avec le code HTML. Différentes librairies de taglibs sont utilisées pour simplifier au maximum le code des vues :

- La bibliothèque de balises JSTL. La librairie *[core]* qui contient les actions de base d'une application web, ainsi que la librairie de formatage *[fmt]* qui permet l'internationalisation d'une application web sont utilisées.
- La bibliothèque de balises *[DateTime]* du projet Apache (Jakarta Taglibs). Celle-ci permet de formater facilement des dates.
- La bibliothèque de balises *[DisplayTag]* qui permet d'afficher des séries d'enregistrements ou des listes d'objets via des tableaux en html de façon très simple.

- La balise personnalisée *[PaginationTag]* a été créée afin de pouvoir gérer facilement la pagination au sein des pages JSP.

Les fichiers .jar de toutes ces librairies ainsi que leurs dépendances sont nécessaires au fonctionnement de l'application.

4.2. Connexion à la base de données

La connexion à la base de données se fait sous la forme d'un pool de connexions géré par Tomcat. Celui-ci a alors besoin du pilote JDBC de connexion au SGBD utilisé. Il doit être placé dans le dossier *common/lib* de l'application web et une ressource de données doit être configurée dans le fichier *server.xml*. Le script de création de la table se trouve dans le fichier *WebContent/scripts/script_creation_database* de l'application AudioVideoCours.

4.3. Génération des flux RSS

Pour créer les fichiers XML des flux RSS, l'API *DOM* incluse dans Java est utilisée.
Les fichiers RSS respectent les standards définis par le W3C pour les flux RSS.

4.4. Gestion des sessions et des cookies

Les sessions et les cookies sont utilisés pour conserver les paramètres des thèmes et de langue, ou de recherche d'un cours pour chaque client.
L'utilisation de l'URL rewriting permet de conserver la session d'une page à l'autre même si l'utilisateur a désactivé les cookies. L'id de session est alors passé dans l'url du contrôleur.

4.5. Internationalisation

De nouvelles langues peuvent être ajoutées aisément au site web.
L'internationalisation se fait grâce à la librairie *[fmt]* de la JSTL, qui utilise les mécanismes standards de Java pour la gestion de l'internationalisation : les classes *Locale* et *ResourceBundle* pour accéder aux données des fichiers de localisation.

Pour ajouter une nouvelle langue, il faut ajouter un fichier .properties pour cette langue dans le package *[org.ulpmm.univrav.language],* du nom du *ResourceBundle* suivi d'un '_' puis des deux lettres identifiant la locale de la langue.

4.6. Thèmes

De nouveaux thèmes peuvent facilement être ajoutés au site web.
Pour ajouter un thème, il faut copier le répertoire générique de thème, modifier les images et les CSS, et le placer dans le répertoire des thèmes sous un nouveau nom. Le thème sera détecté automatiquement.

4.7. Thickbox

La Thickbox est un widget écrit en Javascript qui permet d'afficher et de mettre en valeur le contenu d'une page web sous forme de pop-up avec un effet de présentation à la mode.
Pour fonctionner, les sources Javascript et CSS des projets Thickbox et JQuery doivent être présentes dans les sources de l'application (dans le répertoire *[WebContent/files/thickbox]*) et déclarées dans les pages HTML les utilisant.

4.8. Scripts

Différents scripts sont utilisés lors de l'ajout de cours. Ceux-ci sont de plusieurs langages différents : Shellscript, C, Python.

5. Description des principaux mécanismes de l'application

5.1. Processus de création et d'ajout de cours en différé

1. L'enseignant enregistre son cours grâce au client Audiovideocours. Quand celui-ci est fini, il remplit les informations relatives au cours telles que le titre, l'auteur, la formation, la description, et le code d'accès éventuel servant à rendre un cours privé. Il valide ensuite l'envoi vers le serveur.
2. Le client créé une archive du cours contenant l'enregistrement audio ou vidéo, les captures d'écran du support de cours, et un fichier .csv contenant la liste des « timecodes » de synchronisation des diapositives de cours avec la bande son ou la vidéo.
3. Le client envoie l'archive de cours au serveur via FTP et appelle le contrôleur en lui passant les informations sur le cours et le nom du fichier envoyé par FTP.
4. L'archive de cours est extraite dans le répertoire stockant les cours sur le serveur.
5. Le type de cours est détecté en fonction de la présence d'un fichier .mp3, .rm ou .flv dans le répertoire du cours.
6. Le script Python de vérification des miniatures des diapositives de cours est lancé. Celui-ci permet de créer les miniatures manquantes des diapositives de cours.
7. Si le cours est un cours audio, le fichier .mp3 est renommé avec l'id du cours et retraité grâce à un programme C qui permet d'éviter les problèmes de lecture dans RealPlayer. Si le cours est un cours vidéo, le fichier .flv est renommé avec l'id du cours et un fichier .mp3 de ce cours est créé grâce à un script shell.
8. Les métadonnées avec les informations sur le cours sont ajoutées au fichier .mp3 du cours.
9. La durée du cours est récupérée à partir du fichier .mp3 du cours.
10. Un fichier .pdf contenant toutes les diapositives de cours est créé par un script Python.
11. Le SMIL du cours est créé. Il utilise le fichier .csv pour réaliser la synchronisation entre les médias audio ou vidéo et l'affichage des diapositives de cours.
12. Le fichier .mp3 est converti en .ogg (oggEnc).
13. Un fichier .zip est créé contenant le média du cours (.mp3 ou .flv), le fichier .smil, et le fichier .pdf contenant l'ensemble des diapositives de cours.
14. Le cours est ajouté dans la base de données.
15. Les flux RSS sont régénérés.

5.2. Lancement d'un enregistrement en direct

1. Un enregistrement est lancé sur un client configuré pour la diffusion en direct.
2. Le client contacte le serveur en lui fournissant l'adresse IP de l'amphi et lui indiquant qu'il s'est mis à émettre
3. Le serveur modifie l'état de l'amphi dans sa base de données pour indiquer qu'un cours en direct est diffusé et donner l'accès à la page de visualisation de cours en direct pour cet amphi.
4. Le client envoie les diapositives de cours par FTP en écrasant successivement un fichier image dont le nom est formé à partir de l'adresse IP de l'amphi.
5. Lorsqu'un utilisateur désire visualiser un cours : si celui-ci est diffusé dans une salle permettant la diffusion en vidéo et en audio, un test est lancé pour voir si le client diffuse en audio ou en vidéo. Un cadre affichant les diapositives de cours dans la page web se recharge toutes les 10 secondes pour afficher la diapositive successivement modifiée par FTP. Le flux audio ou vidéo est lu dans la page via un player flash.
6. Lorsque le cours est terminé, Le client contacte le contrôleur en lui fournissant l'adresse IP de l'amphi et lui indiquant qu'il a cessé d'émettre.
7. Le serveur modifie l'état de l'amphi dans sa base de données pour indiquer qu'aucun cours n'est en diffusion et empêcher l'accès à la page de visualisation de cours en direct pour cet amphi.

5.3. Mécanisme de suppression des tests

Une tâche *Cron* est mise en place pour appeler chaque jour le contrôleur et lui donner l'ordre de supprimer les cours dont le code d'accès est « *suppression* ». Ce code d'accès est utilisé pour identifier les cours servant à faire des tests des clients dans les amphis. De plus, les cours contenant dans leur titre les mots « *test* » ou « *essai* » sont masqués.

5.4. Fonctionnement de l'Interfaçage Univ-R

1. Un enseignant se connecte sur la plate-forme WebUniv-R.
2. Il clique sur un lien pour enregistrer un cours. Une fenêtre lui demande d'entrer les informations relatives au cours telles que le titre, la description …
3. Lorsque l'enseignant valide sa saisie, WebUniv-R contacte le contrôleur de l'application AudioVideoCours et lui donne les informations sur le cours ainsi que l'Id de l'enseignant, le code du groupe, et l'identifiant de session Tomcat de WebUniv-R et le nom de l'établissement
4. Le contrôleur vérifie que l'utilisateur est bien connecté à WebUniv-R, et qu'il a le droit de diffuser des cours pour le groupe grâce à l'API Univ-R.
5. Le contrôleur ajoute alors les informations qui sont pour le moment disponibles sur le cours dans la base de données, mais masque ce cours car les fichiers de cours n'ont pas encore été envoyés.
6. Le contrôleur contacte le client (présent sur la machine sur laquelle l'enseignant s'est connecté à WebUniv-R) par socket et lui envoie l'Id du cours qu'il a généré.
7. Le client débute alors l'enregistrement du cours.

8. Lorsque l’enregistrement est terminé, le client renvoie l’Id du cours et le nom du fichier envoyé par FTP au serveur.
9. Le serveur lance alors le processus d’ajout de cours en différé.
10. Ensuite, il complète les informations manquantes dans la base de données et rend le cours visible.
11. Enfin, il fait un ajout dans la base de données d’Univ-R grâce à l’utilisation de l’API Univ-R.

Ensuite, pour la visualisation d’un cours créé depuis Univ-R, le contrôleur dispose d’une méthode permettant l’affichage de l’interface de visualisation depuis Univ-R en contrôlant si l’utilisateur est bien connecté et qu’il a accès au cours.

III. Mise en place de la solution technique

Pour la mise en place de la solution technique, je vous invite à consulter la documentation « exploitation » mise à disposition.