AudioVideoCast

UNIVERSITÉ DE STRASBOURG

# UNISTRA - 2013
# Documentation développeur

**Dispositif de diffusion de cours en ligne AudioVideoCast**
**(anciennement AudioVideoCours)**

http://audiovideocast.unistra.fr

# Sommaire

# I. Présentation du projet AudioVideoCast

## 1. Description fonctionnelle du projet

Audiovideocast 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.
    - 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. Pour plus de détails, voir le fichier « licence.txt » présent dans les sources du projet.

## 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
- Être 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
- 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

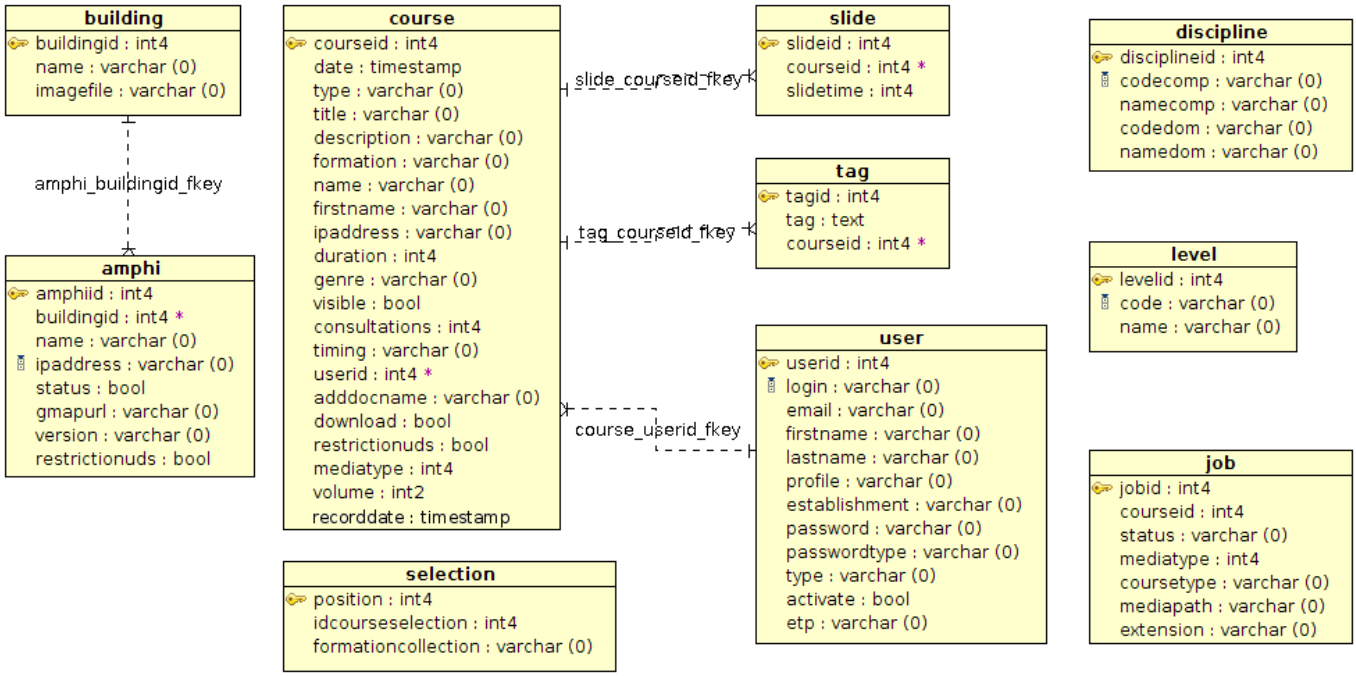
- Les servlets (JAVA) et les pages « jsp » ont été choisies comme solution pour la création de l'application web dynamique afin de garder une certaine homogénéité au sein de l'environnement technique des applications pédagogiques de l'UDS. La gratuité de cette solution est un avantage par rapport à une 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'UDS et profiter de l'avantage de la gratuité.

- L'application web dynamique et la base de données sont hébergées sur des serveurs Linux.

# II. Eléments techniques

## 1. La base de données

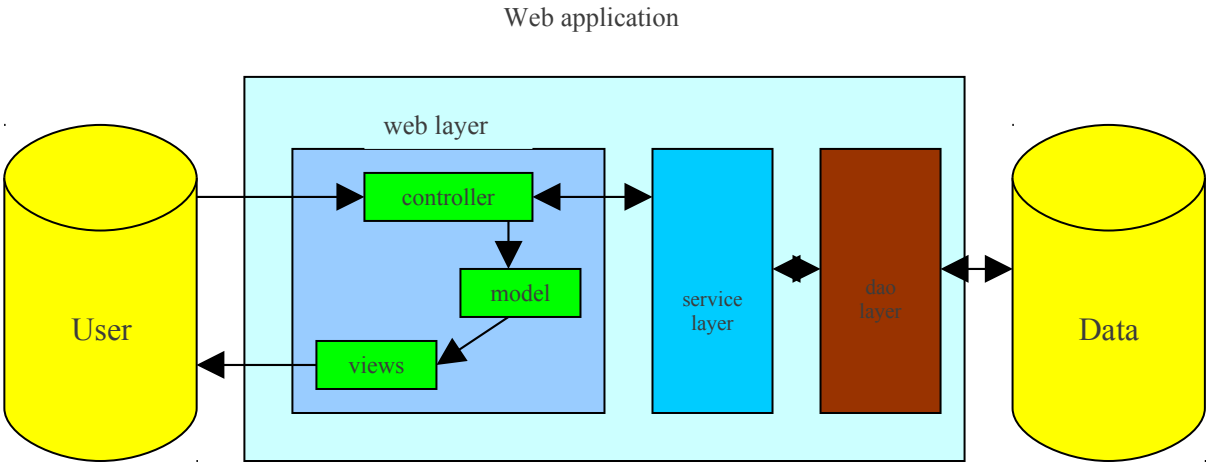



La base de données comprend donc 10 tables :

- 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.
- La table [discipline] contient les différentes composantes possibles (fac de math par ex). Son champ « codecomp » est indirectement lié au champ « formation » de la table [course].
- La table [level] contient les différents niveaux possibles (licence 1ère année par ex). Son champ «code» est indirectement lié au champ « formation » de la table [course].
- 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.
- la table [selection] correspond aux cours affichés dans les listes « Sélection » et « Collection » de la page d'accueil.
- La table [job] contient la liste des enregistrements à traiter. Elle est indirectement lié à la table course via « courseid ».

# 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 du LDAP.

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

### 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.
- *ILdapAccess* : c'est l'interface qui permet d'accéder aux données de l'annuaire LDAP.

Ces interfaces sont respectivement implémentées par les classes suivantes :

- *DatabaseImpl*
- *FileSystemImpl*
- *LdapAccessImpl*

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 *LdapFactory* est une classe utilisée par la classe *LdapAccessImpl*, permettant la récupération des paramètres du Ldap provenant du fichier de configuration de tomcat.

### 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, MediaUpload* et MediaRetag sont trois classes utilisées par la classe *ServiceImpl* et qui héritent de la classe *Thread*. Elles sont chargées de lancer les opérations de modification ou 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, le player, 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 *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. Il

contient également le code source pour le fonctionnement de google analytics.
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. Les liens de ce type seront traités par le contrôleur à chaque requête. Les url se terminant par « */audiocours_v2/servlet/** » sont aussi traitées dans un souci de compatibilité avec les anciennes versions des clients.
  - 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 librairie 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 aux pages de « mon espace » y sont définies grâce à une authentification CAS, en configurant un filtre.

### 3.3. Le répertoire *javadoc*

Le répertoire *javadoc* contient l'ensemble de la documentation du code Java.

# 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 *lib* de tomcat6 et une ressource de données doit être configurée dans le fichier *server.xml (voir doc exploitation)*. Le script de création de la base de données postgresql se trouve dans le fichier *WebContent/scripts/script_creation_database.sql* de l'application Audiovideocast.

## 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, Python.
Ils se trouvent dans le répertoire « WebContent/scripts/jobs_encodage »

- JobEnc.sh : il s'agit du script principal d'encodage des médias. C'est lui qui va se charger d'appeler tous les autres scripts. Il a besoin en entrée du répertoire contenant les scripts d'encodage, de l'url du serveur Audiovideocast pour valider la fin de l'encodage, du répertoire des cours, et

de la ligne « job » dans le cas de l'encodage non séparé (sinon c'est accessbase.py qui le récupère).
- accessbase.py : script utile uniquement lors de l'encodage séparé des médias (voir readme.txt). Il permet de récupérer les jobs en base.
- convertAll2Flv.sh : script permettant la conversion de n'importe quel vidéo en flv.
- convertAll2Mp3.sh : script permettant la conversion de n'importe quel média en mp3
- convertAll2Ogg.sh : script permettant la conversion de n'importe quel média en ogg
- convertAll2Mp4.sh : script permettant la conversion de n'importe quel video en mp3
- convertAll2Webm.sh : script permettant la conversion de n'importe quel video en webm
- calculate_padding.sh : script calculant le padding (bande noire) d'une video. Il est utilisé par « ConvertAll2Mp4.sh »
- CreatePDF.py : script générant un fichier pdf avec les screenshots du cours.
- videoslide.sh : script générant un videoslide à partir des screenshots du cours et du mp3.
- injectMetada.sh : script injectant les metadonnées dans le fichiers flv pour le streaming.

# 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 Audiovideocast. Quand celui-ci est fini, il valide l'envoi vers le serveur via le bouton « Publier ».
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 et l'envoie vers le serveur via FTP.
3. L'enseignant voit alors apparaître dans son navigateur web un formulaire concernant les informations relatives au cours (le titre, l'auteur, la formation, la description, et le code d'accès éventuel servant à rendre un cours privé, etc ...). Il le remplit et le valide.
4. Une tâche en attente de traitement est créé dans la table « Job ».
5. L'archive de cours est extraite dans le répertoire stockant les cours sur le serveur. Cette archive est également déplacée dans ce répertoire pour être téléchargeable.
6. Le type de cours est détecté en fonction de la présence d'un fichier .mp3 ou .flv dans le répertoire du cours.
7. Si le cours est un cours audio, le fichier .mp3 est renommé avec l'id du cours. Si le cours est un cours vidéo, le fichier .flv est renommé avec l'id du cours.
8. La durée du cours est récupérée à partir du fichier .mp3 ou .flv du cours.

9. Le cours est ajouté dans la base de données.
10. Un fichier .pdf contenant toutes les diapositives de cours est créé par un script Python.
11. Le fichier .mp3 est converti en .ogg (oggEnc) et les fichiers .mp4 sont crées selon les cas. Ces médias sont également taggés.
12. Les flux RSS sont régénérés.
12. La base de données est mise à jour pour diffuser les médias et valider la tâche « Job ».

## 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 cadre affichant les diapositives de cours dans la page web se recharge toutes les 5 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 « *testulpmm* » sont masqués.

# III. Configuration de l'environnement de développement

Ce chapitre va vous expliquer comment configurer correctement votre IDE avec les sources du projet.

Pour la mise en place de la solution technique, l'installation des paquets nécessaires et autres, je vous invite à consulter la documentation « exploitation » mise à disposition sur le svn du projet.

## 1. Récupération des sources du projets

Vous pouvez récupérer les sources du projet sur le svn via la commande :
svn co http://subversion.cru.fr/audiovideocours/trunk/server

Il vous faut également récupérer le player flash JW player, en acceptant la licence « Creative Commons : Attribution-Noncommercial-Share Alike 3.0 Unported » (voir http://creativecommons.org/licenses/by-nc-sa/3.0/) :

```
wget http://audiovideocast.unistra.fr/releases/player.swf
mv player.swf server/univ-r_av/WebContent/files/jwflvplayer/
```

## 2. Installation de la base de données

Installez *postgresql* et exécutez le script de création de la base qui se trouve dans le fichier *WebContent/scripts/script_creation_database.sql* de l'application (voir les sources du projet).

Pour plus de détails, veuillez vous référer à la documentation « exploitation ».

## 3. Installation de tomcat

- Téléchargez tomcat 6 via : http://tomcat.apache.org/download-60.cgi
- Dezippez l'archive à l'endroit voulu.
- Installez les librairies nécessaires à Audiovideocast dans Tomcat (voir doc « exploitation »)

## 4. Installation d'Eclipse

L'IDE Eclipse est téléchargeable sur leur site officiel: http://www.eclipse.org/
Nous vous recommandons de prendre la version Eclipse IDE for Java EE Developers.

Pour l'installer, il suffit d'extraire l'archive téléchargée à l'endroit voulu et de l'executer. Eclipse vous demandera alors de choisir un répertoire de travail (workpsace).

# 5. Configuration d'eclipse

## 5.1. Installation du plugin Tomcat

Vous pouvez télécharger le plugin « Tomcat » pour Eclipse via :
http://www.eclipsetotale.com/tomcatPlugin/tomcatPluginV321.zip

Pour l'installer, il suffit d'extraire l'archive téléchargée dans le répertoire « plugins » d'Eclipse, et de redémarrer Eclipse.

Pour le configurer dans Eclipse, il faut compléter les informations se trouvant dans: Window->Preferences->Tomcat

Vous pouvez alors utiliser les boutons présents dans la barre d'outils d'Eclipse pour le démarrer et l'arrêter.

## 5.2. Création du projet et configuration

Pour créer le projet *univ-r_av* :

- Allez dans *File->New->Project->Java->Projet Tomcat->Next*
- Remplissez le champ project name: *univ-r_av*
- Cliquez sur *Finish*
- Remplacez le répertoire *univ-r_av* présent dans votre workspace par le répertoire *univ-r_av* récupéré des sources du projet.
- Créez le répertoire « work » à la racine du répertoire « univ-r_av »
- Actualisez le projet (F5)
- Configurez correctement votre *build path: Clic droit sur le projet->Properties->Java Build Path->Librairies*
- *Clic droit sur le projet->Projet Tomcat->Mise à jour du contexte*
- Modifiez le fichier *univ-r_av.xml* dans le répertoire *conf/Catalina/localhost* créé par Eclipse de la manière suivante:

```
<Context path="/univ-r_av" reloadable="true" docBase="/home/mon_workspace/univ-r_av/WebContent" workDir="/home/mon_workspace/univ-r_av/work">

      <Resource name="jdbc/postgres" auth="Container"
              type="javax.sql.DataSource" driverClassName="org.postgresql.Driver"
              url="jdbc:postgresql://localhost:5432/univrav"
                 username="sqluser" password="secr3t" maxActive="20" maxIdle="10" maxWait="-1"/>

</Context>
```

- Configurez le fichier web.xml du répertoire *WEB-INF* du projet. Il faut juste modifier le filtre CAS.
- Configurez le fichier *univrav.properties* du répertoire *conf* du projet :

Détail du fichier « univrav.properties » :

```
# The Url of the server
serverUrl = http://localhost/univ-r_av
Il s'agit de l'url du serveur. Elle doit être changée.

# The Url to access to the course on internet Prod : http://[HOSTNAME].u-strasbg.fr/
coursesUrl = http://localhost/univ-r_av/coursv2/
Il s'agit de l'url permettant d'accéder aux enregistrements. Elle doit être changée.
Il est possible d'utiliser une fonction RAND[1-10] pour faire du load balancing sur plusieurs serveurs.
Ex: streamRAND[1-10].example.fr va load balancer sur stream1,stream2,...,stream10
Il est également possible d'utiliser une fonction [HOSTNAME] pour récupérer le nom du serveur.

# Folders on the file system Prod : /audiovideocours/cours/ /audiovideocours/ftp/
/audiovideocours/ftp/live/
coursesFolder = /audiovideocours/cours/
Dossier contenant les cours de l'application.
ftpFolder = /audiovideocours/ftp/
Dossier contenant les cours zippés envoyés par le client.
liveFolder = /audiovideocours/ftp/live/
Dossier servant à recevoir la capture d'écran envoyé par le client pendant le Live.

# Default media filenames in the archive sent by the client
defaultMp3File = enregistrement-micro.mp3
defaultFlashFile = enregistrement-video.flv
Ces deux paramètres ne doivent pas être changés pour le bon fonctionnement avec le client actuel.

# Copyright comment
comment = Owned by the author
Commentaire pour les tags des médias.

# IP address of the Flash Server for the video and audio live
flashServerIp = serveurflash.example.fr
Adresse du serveur flash pour la diffusion en live. Doit être changée.

# The settings of the RSS files, of the permalien (interface flash) and of emails
rssTitle = Audiovideocast
rssName = Audiovideocast
rssDescription = Université de Strasbourg
rssImageUrl = http://localhost/univ-r_av/files/img/univr-av-logo-rss.png
rssCategory = Enseignement
language = fr
Paramètres pour les flux RSS. Peut être changé.
recordedInterfaceUrl = http://localhost/univ-r_av/avc/courseaccess
Lien utilisé pour les accès directs aux enregistrements. Doit être changé.

# The setting of the RSS files for iTunes
itunesAuthor = Université de Strasbourg
itunesSubtitle = Enregistrement culturel
itunesSummary = Retrouvez les supports synchronisés sur audiovideocast.unistra.fr
itunesImage = http://localhost/univ-r_av/releases/Illustration_itunes_AVC.jpg
itunesCategory = Enseignement
itunesKeywords = science,culture,société,technologies,éducation
Paramètres pour les flux RSS sous iTunes. Peut être changé.

# University parameters
univName = Université de Strasbourg
univAcronym = UDS
univLink = http://www.unistra.fr/
Informations concernant l'université sur la page d'acceuil
```

```
# Publication free
pubFree = true
pubTest = true
Autoriser ou non la publication libre de contenu

# The numbers of courses to display at the same time
lastCourseNumber = 10
selectionCourseNumber = 10
collectionCourseNumber = 10
recordedCourseNumber = 10
Permet de définir le nombre de cours à afficher dans la page d'accueil et dans la page des
enregistrements.
# The default style
defaultStyle = orange-theme
Thème du site par défaut.

# The keyword to identify the tests to delete (genre is equal to this keyword)
testKeyWord1 = Suppression

# The keyword to identify the tests to hide (title begins with this keyword)
testKeyWord2 = Testulpmm
# testKeyWord3 = Essai
Mots clé pour le code d'accès d'un enregistrement permettant de le masquer.
#Admin email for notification
#AdminEmail1 is used for "Project leader contact"
adminEmail1=admin1@fai.fr
adminEmail2=admin2@fai.fr
adminEmail3=admin3@fai.fr
Adresse email des administrateurs. Doit être changé.

#CAS Logout
casLogoutUrl=https://cas.example.fr:443/cas/logout
Adresse de déconnexion du serveur cas. Doit être changé.

#Additional document formats
addDocFormats=pdf html swf ppt pptx odp docx doc odt xls xlsx ods rtf txt jpg jpeg png gif bmp zip bz bz2
ark rar 7z ac3 avi divx flv m3u m4a mov movie mp2 mp3 mp4 mpg mpeg ogg ra rm rv wav wma wmv aac
Formats de fichiers autorisés pour l'upload de documents additionnels.

#Upload media formats
uploadFormats=mp3 ogg wav wma avi divx mp4 mpg mpeg mov wmv mkv flv ogv webm m4v
Formats de fichiers autorisés pour l'upload de fichiers audio et video.

#Link for support (help page)
supportLink=http://support.unistra.fr
helpLink=https://sites.google.com/site/wikiaudiovideocast/
clientLink=http://sites.google.com/site/audiovideocours/client-logiciel-fr
tracLink=http://sourcesup.cru.fr/projects/audiovideocours
docLink=http://documentationlogicielle.u-strasbg.fr/DUN/IPM/AudiovideoCours

Lien d'aide pour l'application (page help). Peut être changé.

#LDAP search properties
ldapBaseDn=ou=uds,ou=people,o=annuaire
ldapSearchFilter=uid
ldapMail=mail
ldapFirstname=givenName
ldapLastname=sn
ldapProfile=eduPersonPrimaryAffiliation
ldapAffectation=supannetablissement
ldapEtpPrimaryCode=udsPrimaryEtpCode
Propriétés de recherche du ldap. A modifier en fonction du ldap.

# To separate medias encodage
sepEnc=false
Si false, l'application web encode les médias (par défaut). Si true, l'encodage des medias doit se faire
séparément (via tâche cron qui appelle les scripts).
```

Détail du filtre CAS du fichier « web.xml» :

```
<filter>
	<filter-name>CAS filter</filter-name>
	<filter-class>edu.yale.its.tp.cas.client.filter.CASFilter</filter-class>
	<init-param>
		<param-name>edu.yale.its.tp.cas.client.filter.loginUrl</param-name>
		<param-value>https://cas.example.fr:443/cas/login</param-value>
		Url de login du serveur CAS. A changer.
	</init-param>
	<init-param>
		<param-name>edu.yale.its.tp.cas.client.filter.validateUrl</param-name>
		<param-value>https://cas.example.fr:443/cas/serviceValidate</param-value>
		Url de validation du serveur CAS. A changer
	</init-param>
	<init-param>
		<param-name>edu.yale.its.tp.cas.client.filter.serverName</param-name>
		<param-value>localhost</param-value>
		Url de retour après connexion au serveur CAS. Il s'agit de l'adresse de votre site. A
		changer
	</init-param>
</filter>
```

### 5.3. Lancement du projet

Vous pouvez maintenant démarrer tomcat via eclipse, et accéder à l'application via l'url : http://localhost:8080/univ-r_av

NB: Pour configurer Apache et le FTP, veuillez vous reporter à la documentation exploitation.

### 5.4. Variante: utilisation de NetBeans

- Téléchargez la version complète ou java de Netbeans.
  http://www.netbeans.org/downloads/index.html
- Installez l'IDE avec l'option tomcat puis le lancer.
- Créez un nouveau projet « Java Web »
  File->New project->Java web->Web application with existing sources
  Project name : univ-r_av
  Location : Le dossier « univ-r_av » contenant les sources du projet.
  Server: Apache Tomcat 6
- Dans le dossier « configuration files », éditez le fichier context.xml pour définir la connection à la base de données
- Dans le dossier lib du tomcat installé par netbeans, n'oubliez pas d'ajouter les librairies complémentaires (voir doc exploitation)
- Configurez les fichiers web.xml et univrav.properties (voir plus haut)
- Cliquez sur run main project. Firefox doit s'ouvrir avec le site.