

Contents

Hack'edX

Introduction

Edx est une plateforme web qui permet de délivrer des cours en ligne ouverts à tous, les MOOCs - Massive Online Open Courses.

Les cours sont édités dans le « Studio ». Le Studio est une section réservée aux enseignants et aux personnes responsables de maintenir les cours. Il s'agit d'une interface web d'où l'on peut éditer les contenus de cours, gérer les vidéos et les autres ressources à destination des apprenants. Depuis le Studio, on peut gérer les calendriers de cours, gérer les barèmes de notation des apprenants, concevoir les quiz, etc.

Le LMS - Learning Management System, est la section publique de la plateforme Edx. Les cours qui sont édités dans le Studio sont publiés dans le LMS et disponibles aux apprenants. Le LMS est la partie la plus exposée et permet notamment aux apprenants de se connecter et de suivre leur cours. Il s'agit de <https://www.france-universite-numerique-mooc.fr> par exemple. Les apprenants peuvent s'inscrire, consulter les cours, répondre aux quiz, accéder aux résultats, etc.

Références

Comment lire cette documentation ?

Installation de OpenFun / Edx

Introduction aux XBlocks

LMS et Studio

Lorsque vous installez OpenFUN ou OpenEdX, vous aurez les deux “sites” LMS et Studio sur votre machine de développement.

Les moocs ne sont pas composés uniquement de vidéos de cours. Ils s'accompagnent aussi de nombreuses activités, jeux ...

Les étudiants suivent des moocs depuis le LMS (Learning Management System):

Et les professeurs conçoivent ces moocs depuis un CMS (Content Management System), appelé aussi Studio:

Comment lire cette documentation ?

La version pdf de cette documentation peut être téléchargée ici.

Vous pouvez également générer cette documentation au format html et markdown vous-même en clonant le dépôt :

Installez les dépendances nécessaires :

Générez la documentation au format markdown et html :

Installation d'une machine virtuelle OpenFUN / Edx

Les composants nécessaires à l'installation de FUN ou d'edx sont nombreux et relativement complexes ; c'est pourquoi il existe des machines virtuelles (VM) disponibles en simple téléchargement qui permettent de commencer rapidement à tester ces applications. Dans la suite de cette section, nous allons voir les étapes à suivre pour obtenir un environnement de développement fonctionnel.

Téléchargement

Les VM OpenFUN sont disponibles au téléchargement via bittorrent. Si vous ne disposez pas d'un client bittorrent (tel que Transmission, Azure ou Deluge), vous devrez télécharger les VM en HTTP, ce qui risque d'être plus lent et de saturer les serveurs de FUN.

Les fichiers .torrent correspondant aux différentes versions d'OpenFUN sont disponibles ici : <http://files.alt.openfun.fr/vagrant-images/fun/>

Vous pouvez télécharger le fichier openfun-*.torrent correspondant à la version la plus récente d'OpenFUN dans votre client bittorrent favori.

Lancement de la machine virtuelle

Une fois que l'image d'OpenFUN est téléchargée, vous pouvez procéder à son lancement. Pour cela, clonez le dépôt fun-boxes et consultez le README :

Suivez les instructions pour l'installation des dépendances et le lancement de la release que vous avez téléchargée. Vous devrez notamment définir les variables d'environnement suivantes :

Lancement d'un serveur web

Si vous avez correctement lancé votre machine virtuelle, vous pouvez maintenant vous y connecter via ssh et lancer un serveur web local :

Ouvrez maintenant votre navigateur (de votre machine hôte) à l'adresse <http://127.0.0.1:8000> : vous devriez voir apparaître la page d'accueil de FUN. Win!

Le Studio/CMS est alors visible à l'adresse <http://127.0.0.1:8001>.

Vous pouvez également lancer les tests associés à FUN :

Sous le capot, 'fun' est un raccourci permettant d'exécuter une variété de commandes. Pour plus d'informations, consultez la documentation de fun-cmd : <https://github.com/openfun/fun-cmd>

Mon premier Xblock 'Hello Student!'

Les XBlocks enrichissent le contenu de cours : il existe des XBlock pour afficher des vidéos dans le cours, pour y insérer des quiz, pour permettre des discussions de forum, ou même pour exécuter des lignes de code. Edx met à disposition un SDK qui aide à la création de XBlocks. Ainsi, il est possible de commencer le développement de vos modules XBlock sans avoir à installer la plateforme Edx.

Vous trouverez dans ce guide, les instructions pour installer le SDK et pour créer votre premier XBlock.

Installer Python

Installer Pip

Pip est un gestionnaire de dépendance python.

Pour une installation sous Debian/Ubuntu

Une autre façon d'installer pip en téléchargeant le script <https://bootstrap.pypa.io/get-pip.py>:

Installer le xblock sdk depuis le dépôt Github

Lancer le serveur de développement

Maintenant depuis votre navigateur allez à cette adresse 127.0.0.1:8001. Si tout va bien la page suivante devrait apparaître :

Créons la structure de notre xblock

Le script demande d'abord un nom court pour notre xblock, choisissons 'hellostudent'. Ensuite rentrons le nom de classe 'HelloStudentXBlock'

Nous avons maintenant un dossier 'hellostudent' contenant la structure du XBlock.

Afficher 'Hello student'

Ouvrons le fichier `hellostudent/static/html/hellostudent.html` et remplaçons son contenu par :

Enregistrer notre xblock dans le workbench.

Pour afficher notre xblock il est nécessaire de l'installer dans l'environnement de travail, le 'workbench'. L'installation est contrôlée par fichier `setup.py` qu'il faudra modifier pour l'adapter à nos besoins.

Ici, nous travaillons dans le contexte du SDK, mais sachez que ce même principe utilisant pip install est utilisé pour installer un XBlock dans la plateforme Edx.

Vous devriez maintenant avoir un environnement minimal complet.

Pour rappel, les commandes pour démarrer le serveur:

Voici ce que vous devriez voir:

JS-Input

Introduction

Une activité OpenedX peut aller de la simple page HTML, aux quiz et évaluations par les pairs. Le JS-Input est une spécificité d'OpenedX permettant d'étendre les types d'activités disponibles sur la plateforme.

Bien qu'OpenedX offre beaucoup de types d'activités différentes aux créateurs de cours, "beaucoup" n'est souvent pas assez pour l'ensemble des acteurs de la

plateforme. Pour répondre à la demande des MOOC, ils faut souvent créer de multiple types d'activités afin d'éviter l'aspect répétitif et permettre au cours d'être suivi avec plus d'engouement. Il ne faut pas être limité à un choix toujours trop réduit de types de quiz.

OpenedX propose de résoudre ce problème en mettant à portée du développeur/concepteur deux technologies: - Le Xblock (voir documentation spécifique): une extension en python qui doit être installée sur le serveur qui héberge la plateforme. C'est probablement la meilleure solution si vous avez accès au serveur. - Le JS-Input: probablement moins flexible en terme de possibilité, mais probablement la réponse à de nombreux besoins. Le JS-Input a l'avantage d'être une extension dont l'installation se fait directement dans un cours sans nécessiter un accès au serveur.

En gros une activité JS-Input c'est : une page HTML avec un peu de Javascript !

Dans ce document nous allons expliquer comment construire une simple application JS-Input assez générique pour que vous compreniez les mécanismes de base.

Les mécanismes de base

Tout d'abord, voici à quoi ressemble un problème de ce type dans studio:

Exemple d'activité dans studio

Les paramètres de l'activité dans studio sont les suivants:

On peut en déduire que les étapes clés dans l'instanciation d'une activité JS-Input sont: - Le chargement de l'activité et restauration de l'état initial: `set_statefn` - Les actions de vérification du problème côté Open edX: `grade_fn` - Les actions de changement : de note ou d'état `get_statefn` et `grade_fn`

Chargement de l'activité et initialisation de l'état

L'activité se charge dans la page de cours et utilise différents modules internes à Open edX.

Le module principal est "Custom Response Problem" qui est le module générique dans Open edX, permettant d'évaluer une réponse de manière programmatique. L'autre module est appelé JSChannel et permet à l'application JS-Input de communiquer avec Open edX. Nous allons revenir en détail vers ces deux modules dans un autre chapitre.

Pour l'instant occupons nous du processus décrit sur ce schéma:

Sequence de chargement

Lorsque la page se charge, Open edX retrouve le dernier état de l'application pour un utilisateur donné. Cet état se présente sous la forme d'une information codée en JSON. Le format de cette information est particulière à l'application

JS-Input (seule elle la comprend en réalité). Sa signification est définie par le créateur de l'activité.

Si aucun "état" (JSON) pour l'utilisateur n'est trouvé et que l'on a spécifié un état initial, celui-ci est chargé et présenté à l'application JS-Input par un appel à la fonction "setState".

Vérification du problème coté Open edX

La routine de vérification d'un problème est activée par l'appui de l'utilisateur sur le bouton Vérifier (ou "Check" en Anglais). C'est seulement cette action qui déclenchera la séquence de vérification.

Ce qui se passe: * Le conteneur JS Input Problem envoie un "Get grade" pour récupérer une l'information d'état de l'activité encodée en JSON. Il appellera aussi la fonction "Get State" si elle existe pour stocker l'état actuel de l'utilisateur. * L'information passe à travers toutes les couches logicielles (JSChannel, JS Input Problem) et vers edX * Le script python intégré à l'activité JS-Input dans Open edX est lancé pour vérifier le résultat, et renvoie une information sous forme de note. * Le résultat est renvoyé vers le serveur edX

Séquence de chargement

Ensuite le résultat est stocké dans la base de donnée Open edX avec des information sur le temps exact de soumission et ce que l'on appelle un 'correct_map' qui permet de stocker le status (correct ou non) de la réponse après analyse par le script python de l'exercice.

Vous pouvez voir l'historique des soumissions grâce au bouton "Historique des soumissions" situé au dessous de l'activité (seulement accessible par l'enseignant).

Séquence de chargement

Cet historique va donner des résultats comme ceux-ci (application d'exemple Javascript):

Mécanismes de retour d'information

Il existe un troisième mécanisme de retour d'information appelé get_statefn. Dans la pratique, on peut se baser sur le retour de la note (qui peut donner bien plus qu'un état de note, mais aussi une idée de l'état de l'application). Dans ce cas, on va pouvoir définir une fonction de l'application qui est appelée lorsque l'on requiers un status sur l'application. Dans ce cas la réponse de l'application devra comporter deux champs: answer et state.

Exemple:

Les modules

Custom Response Problem et JS Input Problem

Ces deux types de problèmes sont des modules permettant de vérifier la réponse utilisateur par un petit script python avant l'enregistrement réel sur Open edX.

Ceci permet de faire pas mal de choses notamment de noter de manière plus souple tout en restant automatique.

Documentation ici: <https://github.com/Stanford-Online/js-input-samples>

JSChannel

JSChannel est un wrapper créé par Mozilla pour faciliter la communication entre pages et iframes (voir `window.postMessage`: <https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>). La bibliothèque JS Channel facilite ce travail: (<https://github.com/mozilla/jschannel>)

Trucs et astuces

Intégrer du JS Input directement de github

Lorsque l'on développe une extension, il est assez pratique de pouvoir avoir une version de l'application externe au cours sur un site externe. Sinon on est obligé de recharger les fichiers correspondants à chaque mise à jour.

Pour cela il est pratique d'utiliser le lien provenant directement de github sur les ressources:

<https://rawgit.com/>

Faire une activité qui retourne une note différente de 0 ou 1

Liens utiles

Documentation de l'activité JS Input: http://edx-partner-course-staff.readthedocs.org/en/latest/exercises_tools/

Documentation identique mais orientée développeur: http://edxpdrlab.readthedocs.org/en/latest/course_data_f

Custom Python evaluated problem (Version générique du JS Input): http://edx-partner-course-staff.readthedocs.org/en/latest/exercises_tools/custom_python.html

Stanford JS Input Samples: <https://github.com/Stanford-Online/js-input-samples>

Analytics

Introduction

FUN met à la disposition des participants au hackathon une quantité de logs extraits de ses machines de productions à fins d'analyse.

Téléchargement des fichiers de logs

TODO

Source des logs

Les logs proviennent des appels à `tracker.emit` qui parsèment le code d'edX et de FUN : <https://github.com/edx/event-tracking/blob/0.2.0/eventtracking/tracker.py#L65>

Chaque évènement loggé se présente sous la forme d'un blob JSON contenant au moins un champ `time`.

Format des logs

Les logs fournis par FUN sont anonymisés, ce qui signifie que les champs email, address, etc. ont été retirés des blobs JSON. Par ailleurs, le champs username a été chiffré à l'aide d'une méthode chiffrage à sens unique :

Analyse des logs à l'aide de ElasticSearch

Les logs fournis par FUN se prêtent particulièrement bien à l'analyse via ElasticSearch. Si vous décidez de charger les logs fournis dans un cluster ElasticSearch, nous vous recommandons d'installer la pile ELK : ElasticSearch + Logstash + Kibana.

ElasticSearch est le moteur d'indexation et de recherche de vos données

Kibana est le frontend qui vous permettra de visualiser vos données dans le navigateur

Logstash permet d'envoyer vos logs à ElasticSearch en les convertissant en événements au format ad-hoc.

Installation

L'installation des trois composants de la stack ELK est bien documentée :

<https://www.elastic.co/downloads/elasticsearch>

<https://www.elastic.co/downloads/logstash>

<https://www.elastic.co/downloads/kibana>

Envoi des logs vers ElasticSearch

Une fois que vous avez correctement installé Logstash et ElasticSearch, vous pouvez insérer les logs de FUN dans ElasticSearch à l'aide du fichier de configuration logstash.conf fourni dans ce dépôt :

Visualisation des résultats dans Kibana

Après avoir inséré quelques événements dans ElasticSearch, vous pouvez lancer Kibana et observer ces événements en ouvrant <http://localhost:5601> dans votre navigateur. N'oubliez pas de sélectionner un intervalle de temps couvert par les logs (en haut à droite).

Réaliser des requêtes manuelles sur ElasticSearch

Vous pouvez souhaiter réaliser des requêtes complexes sur ElasticSearch et en récupérer le résultat brut au format JSON sans passer par Kibana. Pour ça, le mieux est de :

créer une requête via Kibana, dans l'onglet "Discover".

récupérer cette requête au format JSON, en repliant le graphe de résultats, puis sous l'onglet "Request". Par exemple :

Copier-coller cette requête dans un fichier query.json, puis réaliser la requête à l'aide du script fourni dans ce dépôt :

```
static/es.py query.json > result.json
```