Course authoring with Open edX and SCORM XBlock

SCORM packages are course assets, why should we treat it any differently in Open edX®?

SCORM is a widely used format in the e-learning world. Open edX® has had an integration called edx_xblock_scorm, that allows to include SCORM packages into an Open edX® course.

At Abstract, we've used the package countless times to empower our customers with the best authoring tools. It has worked very well for us, but we started to see some shortcomings and we wanted to fix them.

_ THE PROBLEM

• It didn't provide a Studio preview
• It wasn't possible to import/export a course together with the SCORM packages in OLX format: the SCORM data would get lost in the import/export journey
• It couldn't deal with HTTP/HTTPS environments without additional configuration

_ THE IDEA

We already have the interface needed under the "Files & Uploads" section of the course thus avoiding the need to handle the file upload on the Studio XBlock edit modal. This way we can store SCORM compressed files in the MongoDB database and unpack them on the default storage when needed.

This allows us to effectively bundle the SCORM packages into the course allowing for export and import of courses between different Open edX® installations.

_ THE CHALLENGES

The first challenge was to be able to interact with the course files in the MongoDB database and extract the SCORM packages into the platform configured file storage: it's good for the SCORM file to stay compressed when imported/exported, but it needs to be extracted somewhere to be served to the browser. We used the Django default storage (that for Open edX® installations usually is Amazon S3) for that.

The real challenge was to be able to load contents coming from a different domain, since S3 files are not published on the LMS/CMS domain. This was a tough problem to solve and required greater attention to be solved in a practical way.

All attempts to configure CORS headers or Content Security Policy (particularly the frame-src and frame-ancestors) proved futile.

Indeed, the reason for the inability of the SCORM package to contact the LMS API was caused by browser cross-origin JavaScript restrictions (same-origin policy). There is no way modern browsers will allow some JavaScript code running in an iframe to access frames on another domain. And if you are wondering if subdomain.domain.com and domain.com are considered different domains, well, yes, they are.

Browser architects designed the `Window.PostMessage` method specifically for this job:

The window.postMessage() method safely enables cross-origin communication between Window objects; e.g., between a page and a pop-up that it spawned, or between a page and an iframe embedded within it.

That is, if we want frames to communicate with each other on different domains we need some piece of JavaScript running on both frames handling the communication, or we need both frames to have the same domain.

So what do we have in our case?

Currently, the SCORM XBlock render this structure:

• An iframe which will control whether content should be shown inline or in a new pop-up window. We'll call this "Outer Frame"
• An iframe which will load the SCORM index page. This is the "Inner frame"
• The SCORM index page to embed some content through iframes

Unfortunately the SCORM standard does not mandate a postMessage hook.

Some more useful resources on the subject:

• This article well explains this kind of SCORM run-time environment with nested frames,together with how the SCORM package discovers the LMS JavaScript API.
• This article shows how to bypass the JavaScript same-origin policy. Well, actually they just explain the problem and the possible solutions and redirect to their proprietary solution.

_ A FIRST ATTEMPT

The first approach revolved around dynamically inserting a script block in the extracted SCORM package index HTML file which would have set the JavaScript document object domain to a common top-level domain.

This approach had many disadvantages:

• Forced the file storage to be on the same top-level domain (e.g. lms.example.com and storage.example.com).
• Required the SCORM package to be edited. This operation can be tricky and can possibly generate corner cases difficult to prevent and debug.

The solution was to avoid entirely cross-domain JavaScript restrictions by serving the SCORM package contents from the platform itself.

_ THE SOLUTION THAT WORKED

We needed to find a way to serve the files from the SCORM package from the same domain as the LMS/CMS (depending on where it's being shown). We settled for a solution that stores the zipped file alongside the course data, and its extracted contents on the configured Django Storage. We then serve the extracted files through a Django view, so that they can be seen from the browser as coming from the same domain as the rest of the LMS/CMS. Open edX® from Hawthorn onward features a plugin system which gives us the possibility to easily add Django apps, thus providing us with everything we needed to seamlessly integrate our code.

Interacting with the MongoDB database files was made easy by the usage of the Open edX® contentstore abstraction. It provided a nice and easy interface to filter and fetch course-related files in the GridFS MongoDB storage. GridFS also provides md5 hashes of the uploaded packages for free, allowing us to easily check if the SCORM package needs to be extracted on the file storage (if the same SCORM file is uploaded to two courses we only extract it once).

_ ON CODE TESTING

We highly value automated testing and continuous integration: they help developers keep their software bug-free and maintainable. abstract_scorm_xblock runs its tests on every push and every pull request, to make sure that a contributor is not introducing any bugs and find problems when a new Open edX® version is released.

View the code here:
https://github.com/Abstract-Tech/abstract-scorm-xblock

We finally were able to publish a 1.0.0 version of the Scorm XBlock on PyPI.

Lesen Sie die deutsche Version dieses Artikels: https://abstract-technology.de/lab/artikel/working-with-scorm-xblock