Adding SCORM content to a Drupal LMS

How to include SCORM authored content into an LMS built on Drupal, and extract details of student progress

Posted on September 18, 2021 · 14 mins read

The role of SCORM content in Learning Management Systems

When an organization faces the challenge of setting up an **e-learning platform (LMS)** for their staff or clients, an important decision to make is what platform to build into. The options they have are:

- Using an existing cloud-based service.
- An open-source project, mainly Moodle.
- Developing their own.

Whatever their choice is, once they begin to generate content for their platform this content will be specific to the system and they won't be able to move to another platform. They will remain tied to it forever, will they?

No, they won't! SCORM comes to the rescue.

SCORM sets a standard of communication between the authored content and the container LMS, simplifying among other things the migration from an LMS platform to another. Actually, SCORM has been superseded by other more evolved standards, but the industry is stubborn, and it still remains as a de facto standard because it fills most organizations needs.

Opigno SCORM and Drupal

If your organization is familiar with Drupal, Opigno is the natural choice of an LMS, but SCORM content can also be integrated into a bare-bones Drupal site if preferred. Anyway, the Opigno team has done a great job in the integration of SCORM content so you will most probably end up using at least their Opigno SCORM contrib module. It is a component of the Opigno package but can also be used as a stand-alone module in your Drupal setup.

This great module deploys your SCORM content on the site, lets users interact with it and holds the completeness status of each user on each content. This will be enough in many situations, but Opigno is somewhat self-contained and you might require accessing user details on each specific content which are not provided by default.

Obtain additional information

For example, let's suppose a SCORM content has 5 questions. Your Drupal or Opigno site will know the student has completed it or not, will know the number of correct answers, but you may have other more fine-grained requirements:

- Which was the student response to each question.
- Student progress status before completion.
- The time it took to complete it.

Although the Opigno implementation does not directly provide this information, it exposes it via specific **Drupal hooks** to interact with, and that's what I'm going to show in this article.

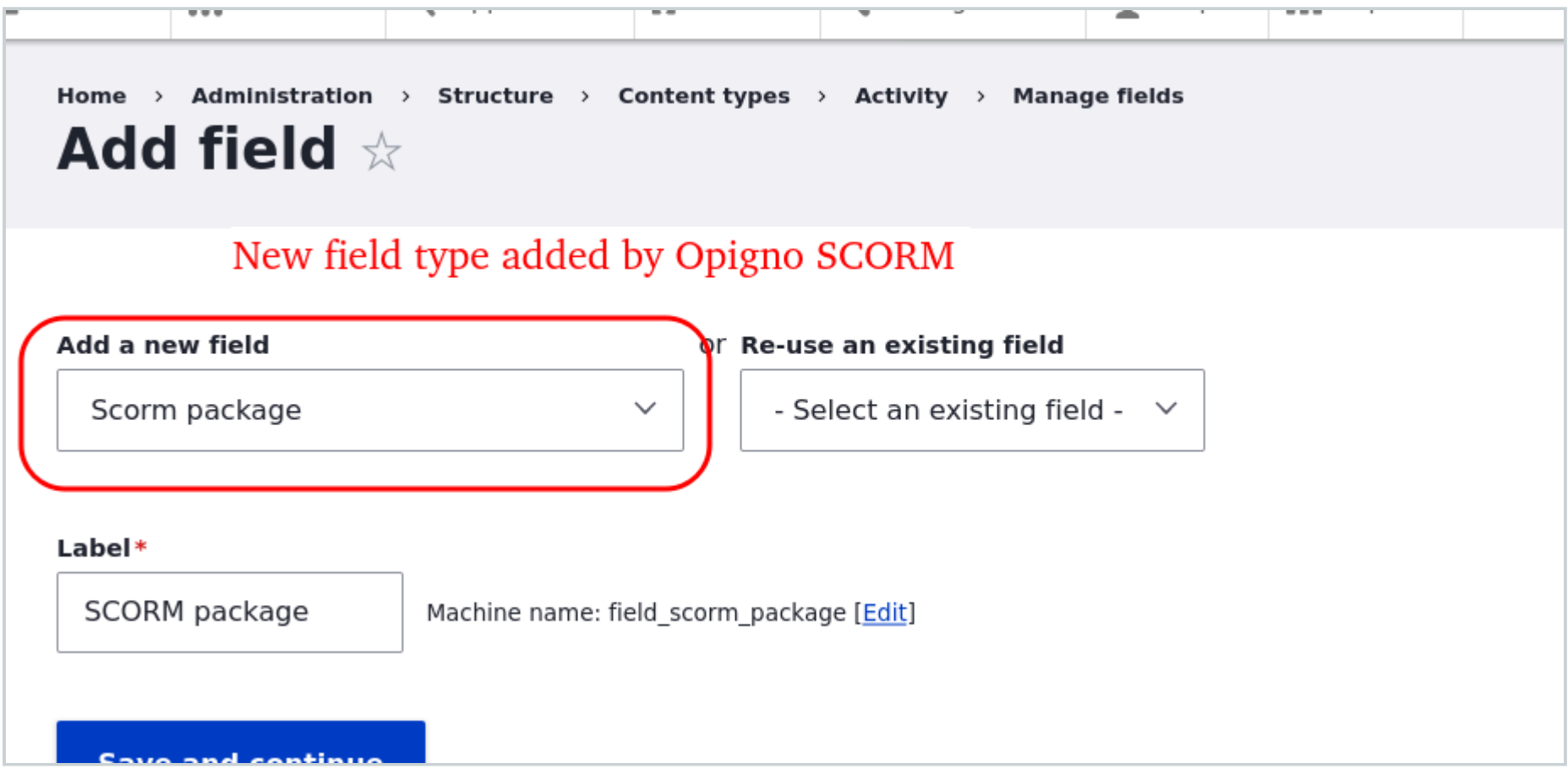
For our tests to work, we need some SCORM content to play with. My choice was the **Forced Sequential Order** example from the Sample SCORM packages at scorm.com. It will be very useful to us because it has multiple pages and there will be some user information to grab.

Use Opigno SCORM into our content type

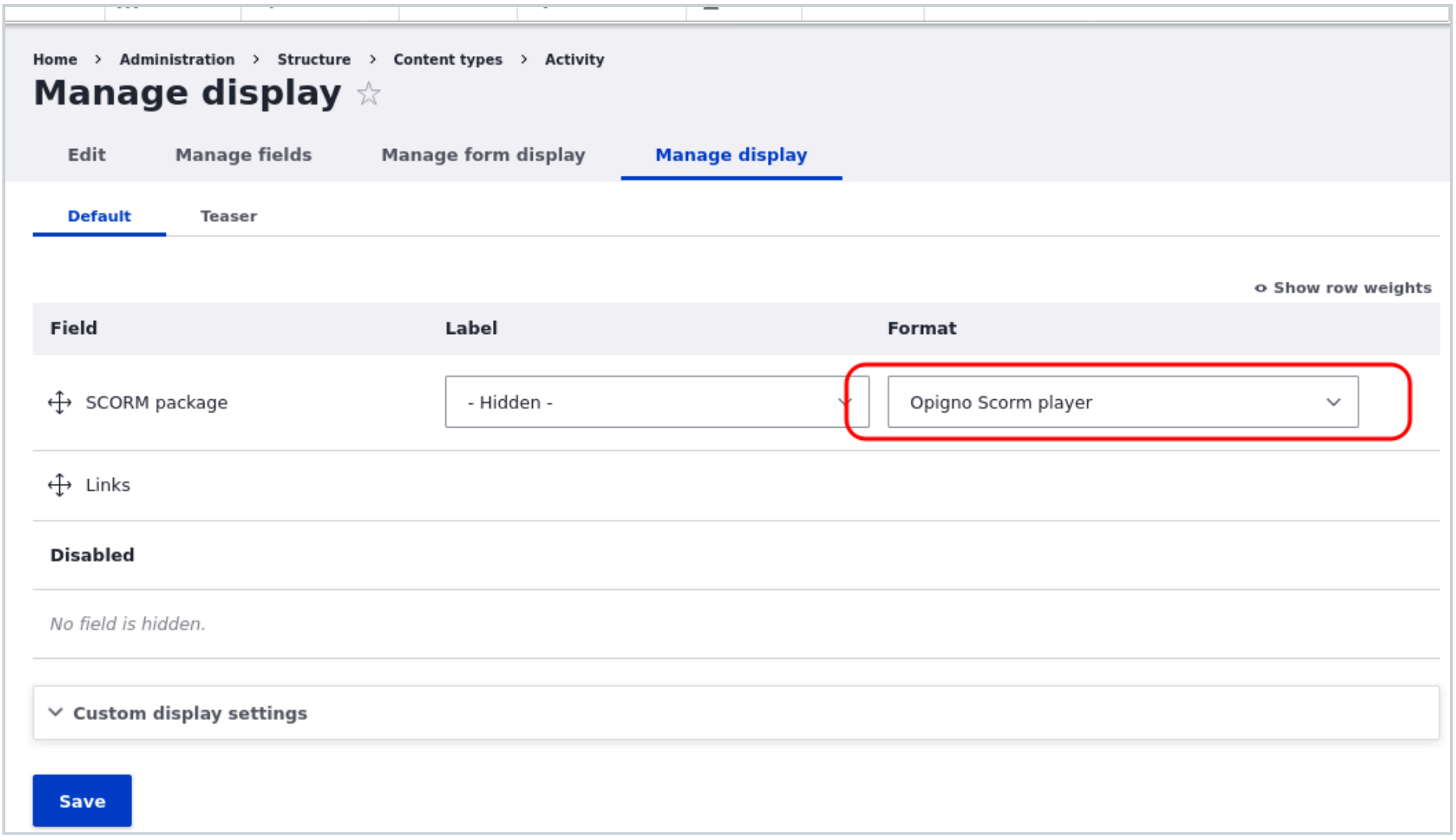
Once we have a Drupal 9 site installed we can add the **Opigno SCORM module** via composer. The latest stable release is only directly compatible with Drupal 8. In order to work in a Drupal 9 environment, we will stick with the new 3.0 branch. It is in beta stability but that won't be an issue in this case:

```
composer require drupal/opigno_scorm @dev
drush -y en opigno_scorm
```

This module provides us with a new **SCORM package** entity field type that we can add to our content types:



Now we can create a custom content type for the training activities, add a **SCORM package** field to it and customize its appearance in **Manage form display** and **Manage display** tabs. SCORM format is delivered as a zip file so that's the only extension you need to allow in the file upload widget settings.



Please note in the previous image the specific formatter provided by the **Opigno SCORM**. It will make our **SCORM activity** visible to students.

At this point, we are ready to create our first activity. After creating it you will find a new **opigno_scorm_extracted** directory in the public files directory of your Drupal installation probably at `/web/sites/default/files/`. This is where **Opigno SCORM** has magically extracted the SCORM content preparing it for student's visualization.

You can visit your new node page and hopefully view the SCORM content thanks to the SCORM player included in the module. It even stores user status so in a future visit with another browser session or even on a different device the user will be able to continue the SCORM content exactly in the point it was left.

A custom module to interact with SCORM committed data

We will go a step further just to demonstrate what we can do directly with the SCORM data. As an example, we will add information to the student user profile page about the completeness status of all the SCORM contents he has followed. In order to do this, a custom entity should be created to store the data, but for the sake of simplicity, we will use the user.data service provided by Drupal.

For a start, we create a custom module. Prepare a service to manage SCORM users status and show them on the user profile page. Our initial module could be something like this:

`modules/custom_scorm/custom_scorm.module`:

```php
/**
 * Implements hook_user_view().
 */
function custom_scorm_user_view(array &$build, \Drupal\Core\Entity\EntityInterface $entity,
\Drupal\Core\Entity\Display\EntityViewDisplayInterface $display, $view_mode) {

  /** @var \Drupal\custom_scorm\UserStatusHandler $scorm_status_service */
  $scorm_status_service = \Drupal::service('custom_scorm.user_status');

  $user_statuses = $scorm_status_service->getAllForCurrentUser();

  if (empty($user_statuses)) {
    $render_array = [
      '#type' => '#markup',
      '#markup' => '<p>You haven\'t started any activites yet.</p>',
    ];
  }
  else {
    $render_array = [
      '#theme' => 'table',
      '#header' => ['Scorm id', 'Location', 'Completion Status', 'Total items', 'Created', 'Last
updated'],
      '#rows' => array_map([$scorm_status_service, 'getRenderableRow'], $user_statuses),
    ];
  }

  $render_array['#prefix'] = '<h3>Status in your training activities:</h3>';

  $build[] = $render_array;
}
```

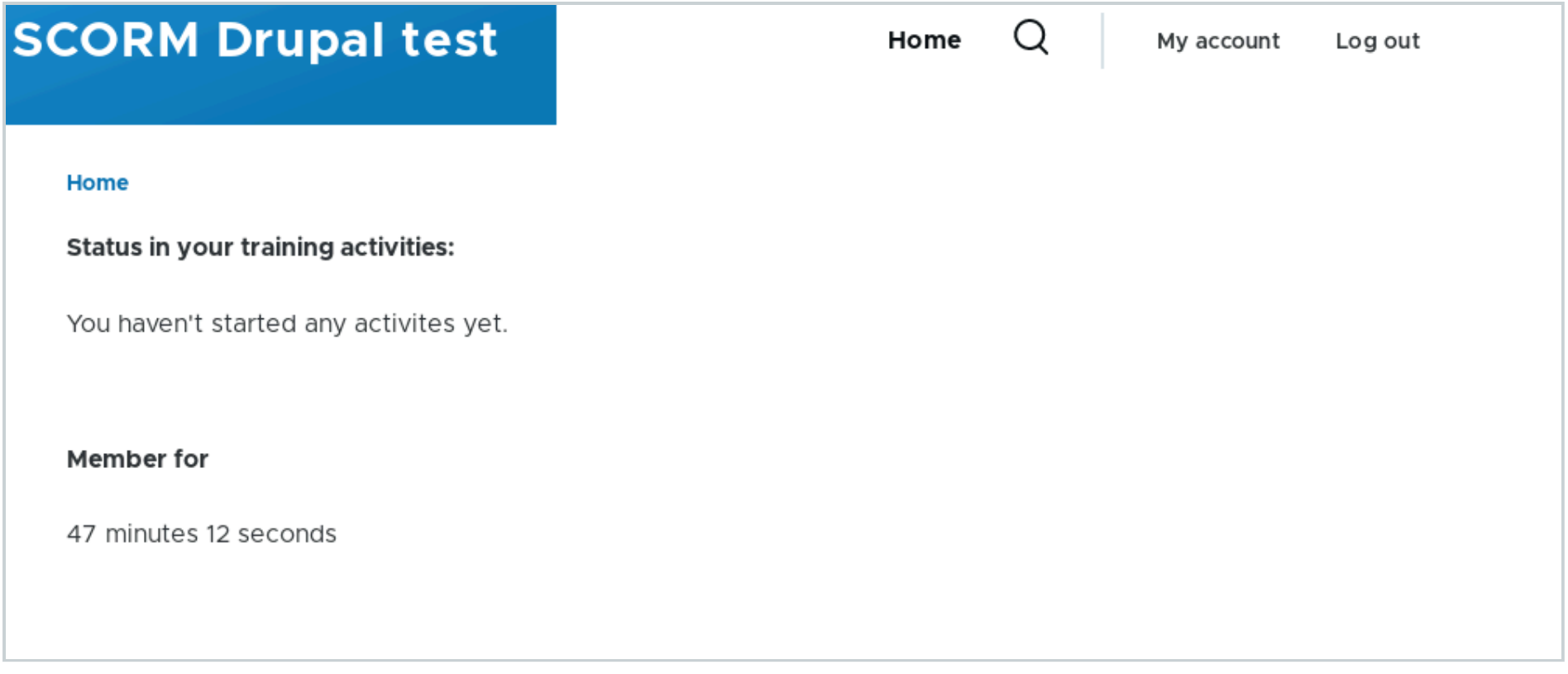
modules/custom_scorm/custom_scorm.services.yml:

```yaml
services:
  custom_scorm.user_status:
    class: Drupal\custom_scorm\UserStatusHandler
    arguments: ['@user.data', '@current_user']
```

modules/custom_scorm/src/custom_scorm.module

```php
namespace Drupal\custom_scorm;

use Drupal\Core\Session\AccountInterface;
use Drupal\user\UserDataInterface;

/**
 * UserStatusHandler service.
 */
class UserStatusHandler {

  /**
   * The user data service.
   *
   * @var \Drupal\user\UserDataInterface
   */
  protected $userData;

  /**
   * The current user.
   *
   * @var \Drupal\Core\Session\AccountInterface
   */
  protected $currentUser;

  /**
   * Constructs an UserStatusHandler object.
   *
   * @param \Drupal\user\UserDataInterface $user_data
   *   The user data service.
   * @param \Drupal\Core\Session\AccountInterface $current_user
   *   The current user.
   */
  public function __construct(UserDataInterface $user_data, AccountInterface $current_user) {
    $this->userData = $user_data;
    $this->currentUser = $current_user;
  }

  /**
   * Returns all SCORM statuses for current user.
   *
   * @return UserStatus[]
   */
  public function getAllForCurrentUser(): array {

    // No data stored for anonymous user
    if ($this->currentUser->isAnonymous()) {
      return [];
    }

    $user_statuses = $this->userData->get('custom_scorm', $this->currentUser->id(), 'user_status');

    return is_null($user_statuses) ? [] : $user_statuses;
  }
}
```

And the result:



Storing interactions our way

In the SCORM standard, the content is responsible for the submission to the LMS of the information regarding user status, each time it should be updated. This is done via an AJAX call usually named `LMSCommit` in SCORM slang. For this purpose, Opigno SCORM provides us with the `opigno_scorm_commit` undocumented hook. Please find in the following code block our implementation of this hook.

`custom_scorm.module`:

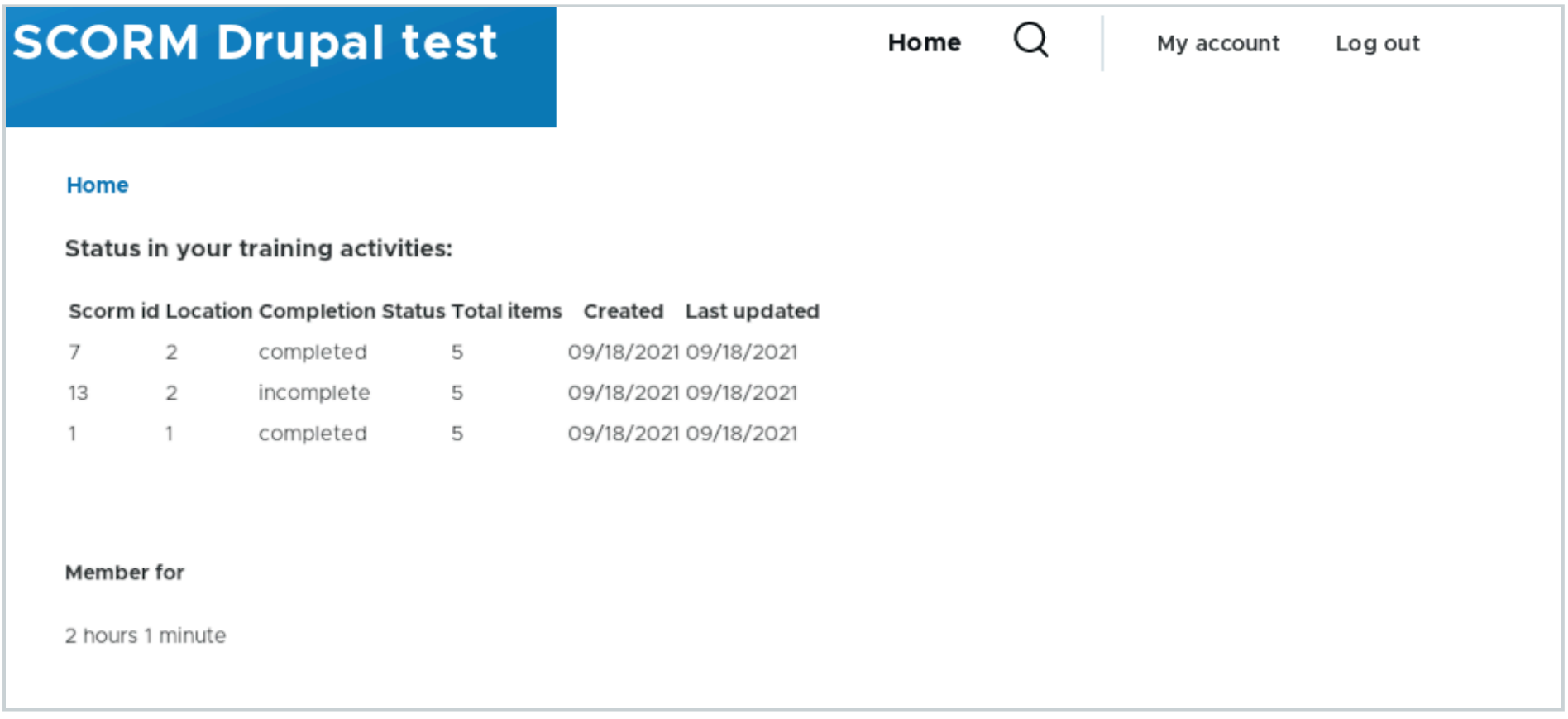
```
/**
 * Implements hook_opigno_scorm_commit().
 */
function custom_scorm_opigno_scorm_commit($scorm, $opigno_scorm_sco_id, $data) {

  /** @var \Drupal\custom_scorm\UserStatusHandler $scorm_user_status */
  $scorm_user_status = \Drupal::service('custom_scorm.user_status');

  $scorm_user_status->setUserStatusFromScormCommitData($opigno_scorm_sco_id, $data);
  \Drupal\Core\Cache\Cache::invalidateTags(['user:' . \Drupal::currentUser()->id()]);
}
```

And in the `setUserStatusFromScormCommitData` we add to `UserStatusHandler` you can see the general structure of the data SCORM exposes:

```php
/**
 * Set SCORM status for current user
 *
 * @param int $opigno_scorm_sco_id
 * @param object $data
 */
public function setUserStatusFromScormCommitData(int $opigno_scorm_sco_id, object $data): void {

  if (!isset($data->scorm_version) || $data->scorm_version != '2004') {
    throw new \Exception(sprintf('Unsupported SCORM version %s.', $data->scorm_version));
  }

  $current_date = date_create();

  if (!$scorm_user_status = $this->findByScoId($opigno_scorm_sco_id)) {
    $scorm_user_status = new UserStatus();
    $scorm_user_status->sco_id = $opigno_scorm_sco_id;
    $scorm_user_status->created = $current_date;
  }

  $scorm_user_status->location = $data->cmi->location;
  $scorm_user_status->completion_status = $data->cmi->completion_status;
  $scorm_user_status->total_items  = count(get_object_vars($data->cmi->suspend_items));
  $scorm_user_status->updated = $current_date;

  $this->saveUserStatus($scorm_user_status);
}
```

The code for the `findByScoId` and `saveUserStatus` is quite straightforward. You can check it out with the module complete code in the GitHub project referenced at the end.

If we advance some pages through the SCORM content and then visit our user profile page we'll find the information is being properly stored and retrieved:



Additional considerations

In order to convert this example into real-life code some additional things have to be taken into account:

Upload of big files

SCORM activity files may be quite big, in the range of 50-100MB so you will need to set `post_max_size` and `upload_max_filesize` PHP settings accordingly or use a tool as Plupload, which lets you upload files of size well above the defined PHP limits allowing you to keep them at reasonable values.

Privately store SCORM content

We can choose the location where the SCORM zip files will be stored, so we can easily make them private, but the Opigno SCORM module stores those files under your site's `public://` path. So a method to make them private is required. This might require modifying the original module's code. The Composer Patches composer plugin may be a good strategy to keep our changes under control.

Special care will also need to be taken regarding the MIME type of the delivered files. Testing our SCORM contents in all available devices (especially your lovely iPhones) is a must to avoid client issues.

Module repo

You can view the full code example in the project repository on GitHub.

← Previous

Images by all-free-download.com

Next →

COMMENT ON THIS POST

Name

Email Address

Comment

Send

LOCATION

C./ Sant Miquel 24
17003 Girona

AROUND THE WEB