Course Access Groups

Release 0.3.0

Omar Al-Ithawi <omar@appsembler.com>

Mar 27, 2020

Contents

An Open edX plugin to customize courses access by grouping learners and assigning different permissions to groups.

Contents:

CHAPTER 1

Course Access Groups

1.1 Overview

This is a plugin for the Open edX Platform that provides the Course Access Group functionality. It can be installed via pip with minimal configuration to provide an admin panel to allow site administrators to create access groups and assign courses to them.

Learners upon registration will be automatically assigned to a specific group, from which it'll be possible to see which courses they'll be able to see and enroll in.

The classic example is that you'd want to offer different courses to your `customers`, `employees` and offer some courses for everyone. Hence you'd need to make two groups and assign courses to only learners within those groups while mark some courses as public ones.

1.2 Documentation

The full documentation is at https://course-access-groups.readthedocs.org.

1.3 Supported Open edX Version

The bad news, there's none. The good news is that there's a plan to make it work with the upstream Open edX versions out of the box.

The even better news, is that you can get this plugin to work by cherry-picking the following pull requests:

- The Access Control Backends pull request.
- The USER_ACCOUNT_ACTIVATED signal.

- The edx-search integration with the "has_access" function.

1.4 License

The code in this repository is licensed under the MIT License unless otherwise noted.

Please see `LICENSE.txt` for details.

1.5 How To Contribute

Contributions are very welcome.

Even though they were written with `edx-platform` in mind, the guidelines should be followed for Open edX code in general.

The pull request description template should be automatically applied if you are creating a pull request from GitHub. Otherwise you can find it at PULL_REQUEST_TEMPLATE.md.

The issue report template should be automatically applied if you are creating an issue on GitHub as well. Otherwise you can find it at ISSUE_TEMPLATE.md.

1.6 Reporting Security Issues

Please do not report security issues in public. Please email security@appsembler.com.

1.7 Getting Help

Have a question about this repository, or about Open edX in general? Please refer to this list of resources if you need any assistance.

CHAPTER 2

Getting Started

This guide provides minimal guide to start working on this app on devstack, production and virtualenv environment.

2.1 Quickstart Instructions for Production

Install this plugin via `pip`. Then configure your Ansible `server-vars.yml` with the following:

```
ACCESS_CONTROL_BACKENDS:
  course.enroll:
    NAME: course_access_groups.acl_backends:user_has_access
  course.see_in_catalog:
    NAME: course_access_groups.acl_backends:user_has_access
  course.see_about_page:
    NAME: course_access_groups.acl_backends:user_has_access
  course.see_exists:
    NAME: course_access_groups.acl_backends:user_has_access
```

2.2 Quickstart Instructions for Devstack

Set `FEATURES["ORGANIZATIONS_APP"] = true` in both `lms.env.json` and `cms.env.json` of your Docker devstack. Set `"ENABLE_COURSE_ACCESS_GROUPS": true` in Site Configuration under: http://localhost:18000/admin/site_configuration/siteconfiguration/ .

Then run the following commands on your machine:

```
$ cd ~/work/tahoe-hawthorn/src/
$ git clone git@github.com:appsembler/course-access-groups.git cag
$ git clone https://github.com/appsembler/edx-search.git search
$ cd search && git checkout appsembler-beta-release-2020-01-07_4
$ cd ../../devstack
$ make COMMAND='pip install -e /edx/src/cag -e /edx/src/search' tahoe.exec.edxapp
```

(continues on next page)

(continued from previous page)

```
$ make COMMAND='python manage.py lms --settings=devstack_docker migrate' SERVICE=lms␣
↪tahoe.exec.single
$ make lms-restart studio-restart
```

You should be able to control the CAG model from within: http://localhost:18000/admin/course_access_groups/

Good luck fiddling with it.

If something doesn't work for you, make sure you have all the required changes for your Open edX for. See the :ref:`supported_open_edx_version` section for information about those required changes.

2.3 Install Dependencies for Contributing to This App

If you have not already done so, create or activate a virtualenv. Unless otherwise stated, assume all terminal code below is executed within the virtualenv.

Dependencies can be installed via the command below.

```
$ make requirements
$ pytest
```

CHAPTER 3

REST API Endpoints

This plugin adds a couple of REST API endpoints to make it possible to integrate with other systems.

The API endpoints follows a Django ViewSet conventions, so this documentation will expand in details on one of the endpoints and be somewhat succinct on the rest given that there's a shared pattern.

See also:

The *User Stories* section explain in details what the overall use cases and features of this plugin which will help to understand how to use the APIs better.

3.1 Course Access Groups

These endpoints lets us to create, edit and delete Course Access Group.

3.1.1 List Groups

This endpoint returns a paginated list of JSON objects in "results". Each object represents a single Course Access Group.

Note: In this version there's no filtering mechanism.

```
GET /course_access_groups/api/v1/course-access-groups/

{
  "count": 50,
  "next": "http://mydomain.com/course_access_groups/api/v1/course-access-groups/?
↪limit=20&offset=20",
  "previous": null,
  "results": [
    {
```

(continues on next page)

(continued from previous page)

```
      "id": 1,
      "name": "Customers",
      "description": "Any customer should be enrolled here"
    },
    {
      "id": 2,
      "name": "Sales Employees",
      "description": "All team members from the sales team"
    }
  ]
}
```

3.1.2 Group Details

This endpoint provides a detailed view of a single Course Access Group when providing an group identifier (`id` for short).

```
GET /course_access_groups/api/v1/course-access-groups/2/

{
  "id": 2,
  "name": "Sales Employees",
  "description": "All team members from the sales team"
}
```

3.1.3 Add a New Group

Performing `POST` request to this endpoint allows to add a new group. Both of the `name` and the `description` `POST` parameters are required.

```
POST /course_access_groups/api/v1/course-access-groups/
{"name": "New Group", "description": "My new group"}
```

3.1.4 Modify a Group

To modify a group, `PATCH` request can be used. Both `name` and `description` can be changed.

Note: This endpoint requires a `Content-Type: application/json` and the request payload to be a properly formatted JSON object as shown below.

```
PATCH /course_access_groups/api/v1/course-access-groups/2/
{"name": "Awesome Group"}
```

3.1.5 Delete a Group

`DELETE` request can be used for deletion, albeit one group at a time.

Note: In this version, `DELETE` is not idempotent, in which deleting an object twice will result in a 404 code for the next request. This is not really a problem as much as it of an issue of not conforming to the HTTP standards.

```
DELETE /course_access_groups/api/v1/course-access-groups/2/
```

3.2 Course-Focused Course Access Group API

This API is used to retrieve course information with their Course Access Group associations and their public status. This API is read-only.

This ViewSet is provide only the minimal course information like id and course name. For more detailed course information or modify the courses information other specialised APIs should be used.

3.2.1 List Courses

This endpoint returns a paginated list of JSON objects in "results". Each object represents a single course. The course JSON also has two sub-objects `public_status` and `group_links`. The inline comments will explain more the properties in more details:

Query Parameters

This endpoint supports the following query parameters e.g. `/course_access_groups/api/v1/courses/?search=python+course`

Name	Type	Description
search	string	Search for any text within the ID and name of the course.
is_public	boolean	Whether the course is set to public via the `/public-courses` APIs.
group	number	Filter by Course Access Group ID. A course could be association to multiple course access groups.
no_group	boolean	Use `True` to filter courses with no group association. On the other hand `False` would filter all courses with *any* group association.

```
GET /course_access_groups/api/v1/courses/

{
  "count": 50,
  "next": "http://mydomain.com/course_access_groups/api/v1/courses/?limit=20&offset=20
↪",
  "previous": null,
  "results": [
    {
      "id": "course-v1:Blue+Python+2020",  // Course ID
      "name": "Introduction to Python",  // Course Name
      "public_status": {
        "is_public": false  // Either `true` or `false`
      },
      "group_links": []
    },
    {
```

(continues on next page)

(continued from previous page)

```
      "id": "course-v1:Blue+SQL+2020",
      "name": "Advanced Postgres Deployments",
      "public_status": {
        "is_public": false
      },
      "group_links": [
        {
          "id": 1,  // GroupCourse linking ID to be used with the `/group-courses`␣
↪API for deletion.
          "group": {
            "id": 1,  // Course Access Group ID
            "name": "Employees"  // Course Access Group name
          }
        }
      ]
    },
    {
      "id": "course-v1:Blue+Coding+101",
      "name": "Coding 101",
      "public_status": {
        "id": 1,  // PublicCourse status ID. Can be deleted via `/public-courses/`
        "is_public": true
      },
      "group_links": []
    }
  ]
}
```

3.3 Linking Courses to Course Access Groups

These endpoints lets us to add and remove courses from Course Access Groups.

3.3.1 List Links

This endpoint returns a paginated list of JSON objects in "results". Each object represents a single a course link to a Course Access Group. The term "link" is only used for documentation purposes instead of the technical name `Group Course`. Each link JSON has a single property `id` which can be used to delete the link. The link JSON also has two sub-objects representing a course and a Course Access Group.

```
GET /course_access_groups/api/v1/group-courses/

{
  "count": 50,
  "next": "http://mydomain.com/course_access_groups/api/v1/group-courses/?limit=20&
↪offset=20",
  "previous": null,
  "results": [
    {
      "id": 1,
      "course": {
        "id": "course-v1:Red+Python+2020",
        "name": "Introduction to Python"
```

(continues on next page)

(continued from previous page)

```
      },
      "group": {
        "id": 1,
        "name": "Customers"
      }
    },
    {
      "id": 2,
      "course": {
        "id": "course-v1:Blue+SQL+2020",
        "name": "Advanced Postgres Deployments"
      },
      "group": {
        "id": 2,
        "name": "Employees"
      }
    }
  ]
}
```

3.3.2 Adding, Modifying and Deleting Links

The link (`Group Course`) endpoints lets us to add, modify and delete the links in a similar way to the Course Access Groups API endpoints.

To add a new link make `POST` request with a JSON payload:

Note: The `group` parameter is the Course Access Group `id` property which can be obtained from the Course Access Groups list API endpoint. Similarly the `course` parameter is the course identifier.

```
POST /course_access_groups/api/v1/group-courses/
{"course": "course-v1:Red+Python+2020", "group": 2}
```

To modify a link `PATCH` request should be used:

POST /course_access_groups/api/v1/group-courses/2/ {“course”: “course-v1:Blue+Python+2020_Fall”}

To delete a link:

```
DELETE /course_access_groups/api/v1/group-courses/2/
```

3.4 Setting Courses as Public

When the course access group feature is enabled, by default all courses are private and only accessible to learners with memberships to groups that have those courses.

However, some users could have no group membership so they won't have access to the private courses. To make a course available to both learners with and without a group membership a course should be made public.

This endpoint sets the course to be public.

3.4.1 List Public Courses

This endpoint returns a paginated list of JSON objects in "results". Each JSON object represents a course being public. Each JSON object has a single property `id` which can be used to make the course private. The JSON object also has a sub-object representing a course.

```
GET /course_access_groups/api/v1/public-courses/

{
  "count": 50,
  "next": "http://mydomain.com/course_access_groups/api/v1/public-courses/?limit=20&
↪offset=20",
  "previous": null,
  "results": [
    {
      "id": 9,
      "course": {
        "id": "course-v1:Red+Python+2020",
        "name": "Introduction to Python"
      }
    },
    {
      "id": 10,
      "course": {
        "id": "course-v1:Blue+SQL+2020",
        "name": "Advanced Postgres Deployments"
      }
    }
  ]
}
```

3.4.2 Making a Course Public or Private

`POST` request can be used to make a course public with the following JSON payload format:

Note: The `course` parameter is the Course `id` property which can be obtained from the Course API endpoint.

```
POST /course_access_groups/api/v1/public-courses/
{"course": "course-v1:Red+Python+2020"}
```

To make a course private:

```
DELETE /course_access_groups/api/v1/public-courses/10/
```

3.5 Membership in Course Access Groups

This endpoint lets us to add and remove users from Course Access Groups.

3.5.1 List Memberships

This endpoint returns a paginated list of JSON objects in "results". Each object represents a single user membership in a Course Access Group. Each membership JSON has a single property `id` which can be used to delete the membership. The membership JSON also has two sub-objects representing a user and a Course Access Group.

```
GET /course_access_groups/api/v1/memberships/

{
  "count": 50,
  "next": "http://mydomain.com/course_access_groups/api/v1/memberships/?limit=20&
↪offset=20",
  "previous": null,
  "results": [
    {
      "id": 5,
      "user": {
        "id": 2,
        "username": "ali",
        "email": "ali@corp.com"
      },
      "group": {
        "id": 1,
        "name": "Employees"
      }
    },
    {
      "id": 6,
      "user": {
        "id": 3,
        "username": "Mike",
        "email": "mike@customer.com"
      },
      "group": {
        "id": 2,
        "name": "Customers"
      }
    }
  ]
}
```

3.5.2 Adding, Modifying and Deleting Memberships

The membership endpoints lets us add, modify and delete the memberships in a similar way to the Course Access Groups API endpoints.

To add a new membership make `POST` request with a JSON payload:

Note: The `group` parameter is the Course Access Group `id` property which can be obtained from the Course Access Groups list API endpoint. Similarly the `user` parameter is the user identifier.

```
POST /course_access_groups/api/v1/memberships/
{"user": 857, "group": 2}
```

To modify a membership `PATCH` request should be used.

Note: A user can have a membership to a single group.

POST /course_access_groups/api/v1/memberships/5/ {"group": 3}

To delete a membership:

```
DELETE /course_access_groups/api/v1/memberships/5/
```

3.6 User-Focused Course Access Group API

This API is used to retrieve user information with their Course Access Group associations. This API is a read-only API.

This ViewSet is provide only the minimal user information like email and username. For more detailed user information or modify the membership information other specialised APIs should be used.

3.6.1 List Users

This endpoint returns a paginated list of JSON objects in "results". Each object represents a single user in your organization. Each user JSON has a few personal information like `email` and `username`. The user JSON also has a sub-object `membership` in a Course Access Group. The inline comments will explain in more details:

Query Parameters

This endpoint supports the following query parameters e.g. `/course_access_groups/api/v1/users/?search=corp.com`

Name	Type	Description
search	string	Search for any text within the name, username and email of the user data.
email_exact	string	Search for case-insensitive exact matches of a user email.
group	number	Filter by Course Access Group ID. A course can be a member of a single course access group.
no_group	boolean	Use `True` to filter users with no group association. On the other hand `False` would filter all users with *any* group association.

```
GET /course_access_groups/api/v1/users/

{
  "count": 50,
  "next": "http://mydomain.com/course_access_groups/api/v1/users/?limit=20&offset=20",
  "previous": null,
  "results": [
    {
      "id": 2,  // The User ID that can be used in the `/memberships/` endpoint
      "username": "ali",  // The short public username used in forums
      "name": "Ali Al-Ithawi",  // The full name used in certificates
      "email": "ali@corp.com",
      "membership": {  // Membership information
        "id": 5,  // Use this `membership` ID to delete the membership via the `/
↪memberships/` endpoint.
```

(continues on next page)

(continued from previous page)

```
        "group": {
          "id": 1,  // The Course Access Group ID
          "name": "Employees"  // The Course Access Group name
        }
      }
    },
    {
      "id": 2,
      "username": "johnb",
      "name": "John Baldwin",
      "email": "john@community.org",
      "membership": null  // This user has no membership
    }
}
```

3.7 Rules for Automatic User Membership

These endpoints lets us to manage rules for automatic membership based on email address.

Note: The membership rules are only activated after the learner (user) activates their email address. Before that, the learner will be considered as without a group.

3.7.1 List Membership Rules

This endpoint returns a paginated list of JSON objects in "results". Each object represents a single membership rule. Besides the `id` and the `name` properties, each rule JSON has a `domain` which is the email domain name to match the users for.

The membership rule JSON also has a sub-object representing a Course Access Group.

```
GET /course_access_groups/api/v1/membership-rules/

{
  "count": 50,
  "next": "http://mydomain.com/course_access_groups/api/v1/membership-rules/?limit=20&
↪offset=20",
  "previous": null,
  "results": [
    {
      "id": 8,
      "name": "Assign customers",
      "domain": "customer1.xyz",
      "group": {
        "id": 1,
        "name": "Customers"
      }
    },
    {
      "id": 9,
      "name": "Assign another customer",
      "domain": "company2.xyz.uk",
```

(continues on next page)

(continued from previous page)

```
      "group": {
        "id": 1,
        "name": "Customers"
      }
    }
  ]
}
```

3.7.2 Adding, Modifying and Deleting Memberships

The membership rule endpoints lets us to add, modify and delete the membership rules in a similar way to the Course Access Groups API endpoints.

To add a new membership rule make `POST` request with a JSON payload:

Note: The `group` parameter is the Course Access Group `id` property which can be obtained from the Course Access Groups list API endpoint.

```
POST /course_access_groups/api/v1/membership-rules/
{"name": "XYZ Customers", "domain": "company.xyz", "group": 2}
```

To modify a membership rule `PATCH` request should be used.

POST /course_access_groups/api/v1/membership-rules/5/ {“group”: 3}

To delete a membership rule:

```
DELETE /course_access_groups/api/v1/membership-rules/5/
```

CHAPTER 4

User Stories

Note: Current user stories are being drafted in our Confluence page. Please ping me at omar@appsembler.com to share it with you for early access.

CHAPTER 5

Engineering Design

This section describes the design of the Course Access Groups Open edX app. It aims to be succinct and focuses on three topics:

- Hard-coded assumptions
- Related extension points in Open edX
- The internal models and their interaction

Besides a section to denote future plans for the app.

5.1 Hard-coded Assumptions

Ideally, this app would blend into Open edX seamlessly without requiring special configuration or dependencies. However, this level of seamless integration may come at the cost of both complexity and reliability of the app.

Therefore the app was designed to be somewhat simple with a couple of hard-coded assumptions about the platform that would uses it.

While these assumptions are hard-coded in the initial release, some of them will be revisited in future release to be either modified or completely removed.

- **Multi-tenancy:** One of the main assumptions is that the platform uses the multi-tenancy (Django Site) features without having a main site of its own.
- **Dependency on edx-organizations:** In order to implement a full-featured multi-tenancy app, the `edx-organizations` is the go-to app to implement this feature.
- **Access Control Backends:** Which allows plugins like the CAG app to modify the behaviour of `courseware.access.has_access` function in Open edX platform. More info is one the Access Control Backends pull request. pull request.
- **Other app changes:** In order for the app to work it requires three hard-coded changes into the platform that would eventually be upstreamed. For more information see :ref:`supported_open_edx_version`.

5.2 Course Access Groups Database Modules

Naturally, `course_access_groups.models` module has the most up-to date information and documentation about the database models.

Nevertheless, this section aims to provide a summary of the models design. As of writing this docs, there are five distinct new models in this app. However, when looking at the logical design we would find only three models like the following:

5.2.1 The `CourseAccessGroup` Model

This is the main model which specify which learners should have access to which courses.

This model has two Many to Many relationships:

- **Course:** Courses that belongs to this group. To simplify the use relation with the Django REST Framewor it has been moved to its own model named `GroupCourse`.
- **User:** Users who are member of this group. For the same reasons above it has it's own model named `Membership`.

This model is the only essential model for this app to run. The others can be thought of as complementary.

5.2.2 The `PublicCourse` Model

This model is used to mark courses as public to exempt from the Course Access Group rules.

Instead of modifying `CourseOverview` to have an additional boolean field e.g. `CourseOverview.cag_is_public`, a new model has been created.

This necessary to avoid having complex database migration design and minimize the maintenance impact of this app.

5.2.3 The `MembershipRule` Model

This model is used to automatically assign learners to Course Access Groups. It utilizes the USER_ACCOUNT_ACTIVATED Django signal to match learners into groups based on their email domain name.

5.3 Future Plans

This application works on a *modified* fork of the Hawthorn release. This section denotes plans to support future releases the impact on the architecture of this app.

5.3.1 Supporting Bridgekeeper in Juniper

As of April, 2019 the Open edX team started to use Bridgekeeper for Access Control which eventually would deprecate the `courseware.has_access` function.

The plan to support Bridgekeeper is to remove the hooks for `has_access` and replace them with `bridgekeeper.perms[]` rules.

For more information about Bridgekeeper check the project documentation: https://bridgekeeper.readthedocs.io/.

This is probably the change with the most impact. So far there's no concrete plan to adapt to it.

In addition to this documentation please consider taking a look at the Access Control Backends thread on Open edX Discuss. It touches on a couple of related topics regarding Bridgekeeper and some recommendation from the edX engineers.

5.3.2 Single-site Setups

The majority of the Open edX Platform installations are single-site setups in which Site Configurations isn't used. This application doesn't support such installations at the moment. Several modifications needs to be done to support this installation. Here are few that we are already aware of:

- A new flag to select the mode of the app e.g. `FEATURES['COURSE_ACCESS_GROUPS_IS_MULTISITE']`.
- `CourseAccessGroup.organization` to be optional: This can be done in two methods: A) Make the field `null=True` or make a new `SiteWideCourseAccessGroup` to avoid having null values. My (Omar) preference is having a second module.
- Some queries checks either of `UserOrganizationMapping` and `OrganizationCourse`. Those queries won't work on the single-site setups so it needs to be refactored.

Open Question: Do we want to support both site-wide and organization-specific mode at the same time? The initial assumption that it would be very costly and would complicate the app. Anyway, that's still an open question.

CHAPTER 6

Testing

course-access-groups has an assortment of test cases and code quality checks to catch potential problems during development. To run them all in the version of Python you chose for your virtualenv:

```
$ make validate
```

To run just the unit tests:

```
$ make test
```

To run just the unit tests and check diff coverage

```
$ make diff_cover
```

To run just the code quality checks:

```
$ make quality
```

To run the unit tests under every supported Python version and the code quality checks:

```
$ make test-all
```

To generate and open an HTML report of how much of the code is covered by test cases:

```
$ make coverage
```

CHAPTER 7

Internationalization

All user-facing text content should be marked for translation. Even if this application is only run in English, our open source users may choose to use another language. Marking content for translation ensures our users have this choice.

Follow the internationalization coding guidelines in the edX Developer's Guide when developing new features.

7.1 Updating Translations

This project uses Transifex to translate content. After new features are developed the translation source files should be pushed to Transifex. Our translation community will translate the content, after which we can retrieve the translations.

Pushing source translation files to Transifex requires access to the edx-platform. Request access from the Open Source Team if you will be pushing translation files. You should also configure the Transifex client if you have not done so already.

The *make* targets listed below can be used to push or pull translations.

Target	Description
pull_translations	Pull translations from Transifex
push_translations	Push source translation files to Transifex

7.2 Fake Translations

As you develop features it may be helpful to know which strings have been marked for translation, and which are not. Use the *fake_translations* make target for this purpose. This target will extract all strings marked for translation, generate fake translations in the Esperanto (eo) language directory, and compile the translations.

You can trigger the display of the translations by setting your browser's language to Esperanto (eo), and navigating to a page on the site. Instead of plain English strings, you should see specially-accented English strings that look like this:

> Thé Fütüré øf Ønlïné Édüçätïøn σ ι# Før änýøné, änýwhéré, änýtïmé σ #

CHAPTER 8

Change Log

8.1 Unreleased

-

8.2 [0.1.0] - 2019-11-26

8.2.1 Added

- First release on PyPI.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search