Orcoursetrion Documentation

Release 0.1.1

MIT Office of Digital Learning

August 10, 2015

Contents

Automatic course provisioning for the edx-platform using github and zendesk.

CHAPTER 1

Quick Start

To install the latest release, run `pip install orcoursetrion`.

If you want to be on the development edge (generally stable), clone the repository, and run `pip install .` or just install directory from github.com with `pip install git+https://github.com/mitodl/orcoursetrion`.

Once installed, create or acquire an OAUTH2 token from github. That at least has the `repo`, `write:repo_hook`, `admin:org`, and `write:org` permissions.

Add the environment variable `ORC_GH_OAUTH2_TOKEN=<your token>` to your environment, and run `orcoursetrion --help` for available commands and actions.

If you are adding an XML course, you will also need to define `ORC_STAGING_GITRELOAD` in your environment for where Web hooks should be sent for push events.

Optional

There are a few other environment variables to add if you want to use the release command, or if you would like orcoursetrion's commits to be from on particular user.

`ORC_PRODUCTION_GITRELOAD` for where Web hooks related to a production run of your course should be sent for push events.

`ORC_GH_NAME` for how you want commits from orcoursetrion to be identified.

`ORC_GH_EMAIL` for what email address you want associated with commits from orcoursetrion.

CHAPTER 3

Table of Contents

3.1 Command Line

There is an exposed command line interface that is available upon installation of the repository. It can be run with `orcoursetrion`, and `orcoursetrion --help` will provide the most up to date help information.

The command allows you to run commands that correspond to actions, currently the only supported action is `create_export_repo`, and if your configuration is setup correctly (see *Configuration*), and at least minimally have set *`ORC_GH_OAUTH2_TOKEN`* and you run `orcoursetrion create_export_repo -t Spring_2030 -c DevOps.001 -d 'My awesome class repo'` you should see it respond with the URL of the repo that it just created for you.

3.1.1 Available Actions

create_export_repo This will create a new repository with the content deployment team from *`ORC_STUDIO_DEPLOY_TEAM`* added to the repository.

rerun_studio This will remove all Web hooks from the course specified by `term` and then create a new repo with the `new_term`, along with the *`ORC_STUDIO_DEPLOY_TEAM`* added.

release_studio This will add the production git Web hook to the course specified with *`ORC_PRODUCTION_GITRELOAD`*.

create_xml_repo This will create a new repository with the *`ORC_XML_DEPLOY_TEAM`* and a command line specified team added to repository. It will also set up a git hook to the URL specified with *`ORC_STAGING_GITRELOAD`*. The membership of the team can also be specified, and will replace the existing membership of the team if it already exists.

rerun_xml This will rerun an XML course. Currently this will just remove any Web hooks that are currently attached to the repository.

release_xml This will add the production git Web hook to the course specified with *`ORC_PRODUCTION_GITRELOAD`*.

put_team This will create or update a team specified in the specified organization. If the team doesn't exist, there is an option to give the team either push or pull access, otherwise the `read_only` flag is ignored. It optionally takes a list of members of the team that should replace the existing team.

3.2 Design

3.2.1 Workflow, Design, And Architecture

Quite some time is lost to course provisioning and request management. This document both defines what course provisioning is, and is a design for how to create software to assist in automating this process.

There are three major stories/workflows that occur in our course production/publishing architecture. While all three occur for residential MITx, some of them also apply and could be used for use with MOOC/Open Education.

The three major process flows are:

- Creating a brand new course
- Publishing a course to our student facing LMS
- Rerunning an already existing course

3.2.2 Creating a Brand New Course

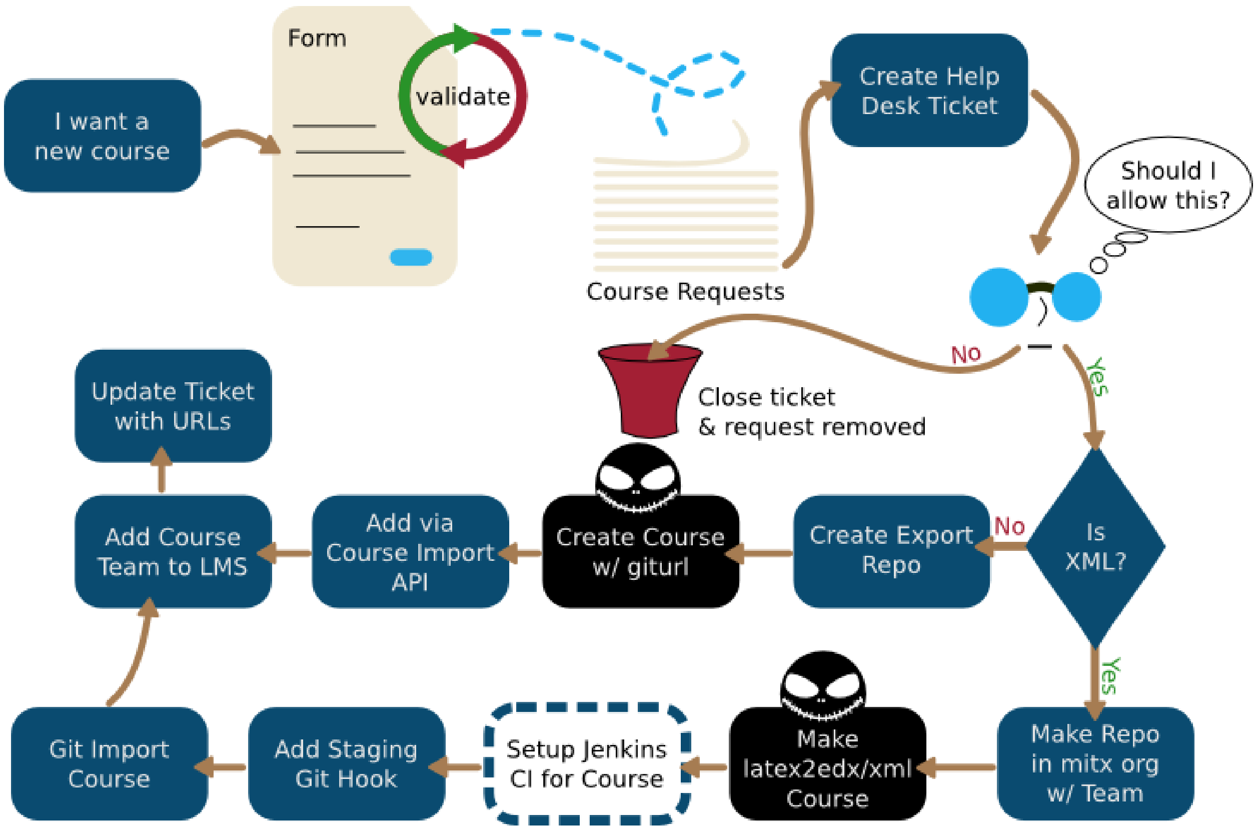

Fig. 3.1: New Course

Areas of note

- Setting up Jenkins CI for the course is doable, but is left in cutout in the diagram as it is a quite complex and not required for course provisioning.

- The form needs to both programmatically validate course team members, and the `course id` entered (i.e. 8.01r). It is intended that the validation for `course id` will either use the edX-Platform course listing API directly to check for duplicates, or maintain a (mostly) complete list of all current courses internally
- A fully automatic mode could be supported without the help desk ticket approval step.

3.2.3 Moving a Course from Staging to Production

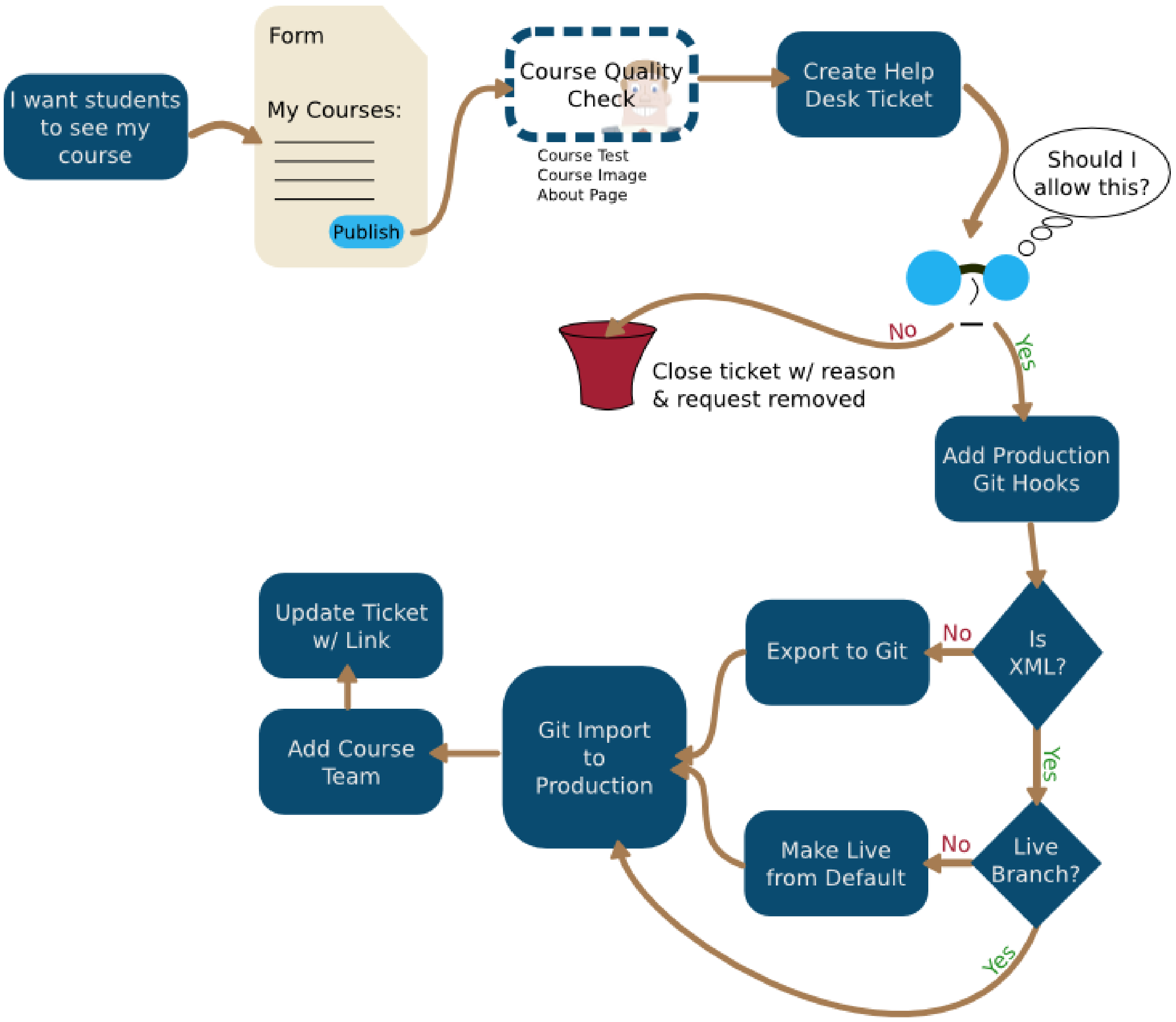

Fig. 3.2: Moving a Course

Areas of note

- The course quality check could potentially be done by a jenkins job, or ACPx could provide that feature itself since it will already have to be somewhat courseware capable.
- Similar to a new course, the approval process can be skipped here

3.2.4 Rerunning a Course

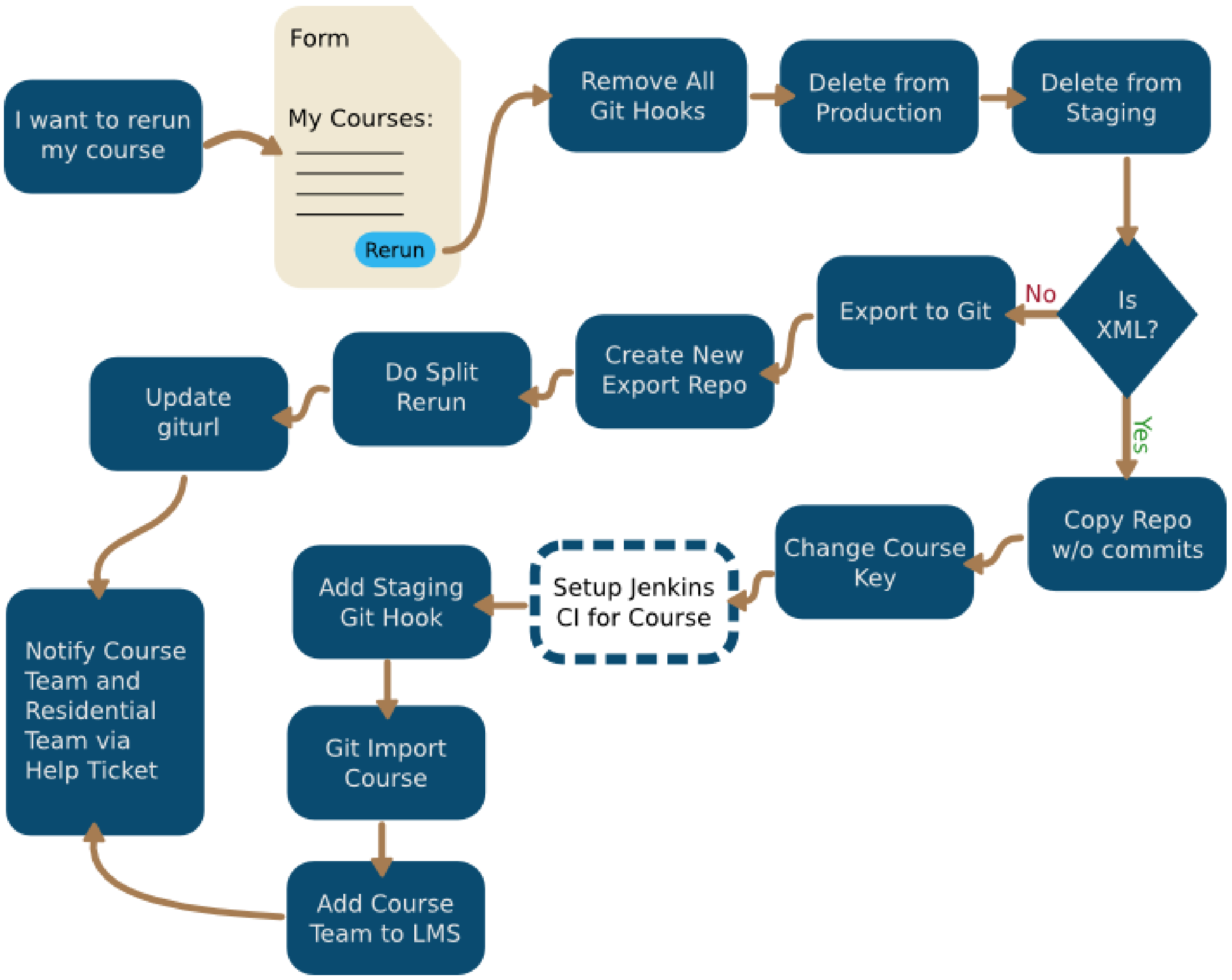

Fig. 3.3: Rerunning a Course

Areas of note

- This is pretty much a blend of the move and rerun flows, and only adds the rerun edX platform call, the deletions, and the giturl update.
- As an option, instead of doing the split rerun function above, we could potentially use the same flow as the git course by deleting it, modifying the course XML (and using an XML cleanup script i.e. Piotr's Cleanup Script, and re-importing.

3.2.5 System Architecture

Areas of note

- None of the APIs for edx-platform currently exist and will likely have to be replaced with instruction updates to the helpdesk tickets instructing the agent to perform the requested task and provide instructions that make it as quickj and easy as possible for the agent to do so, and mark them complete.

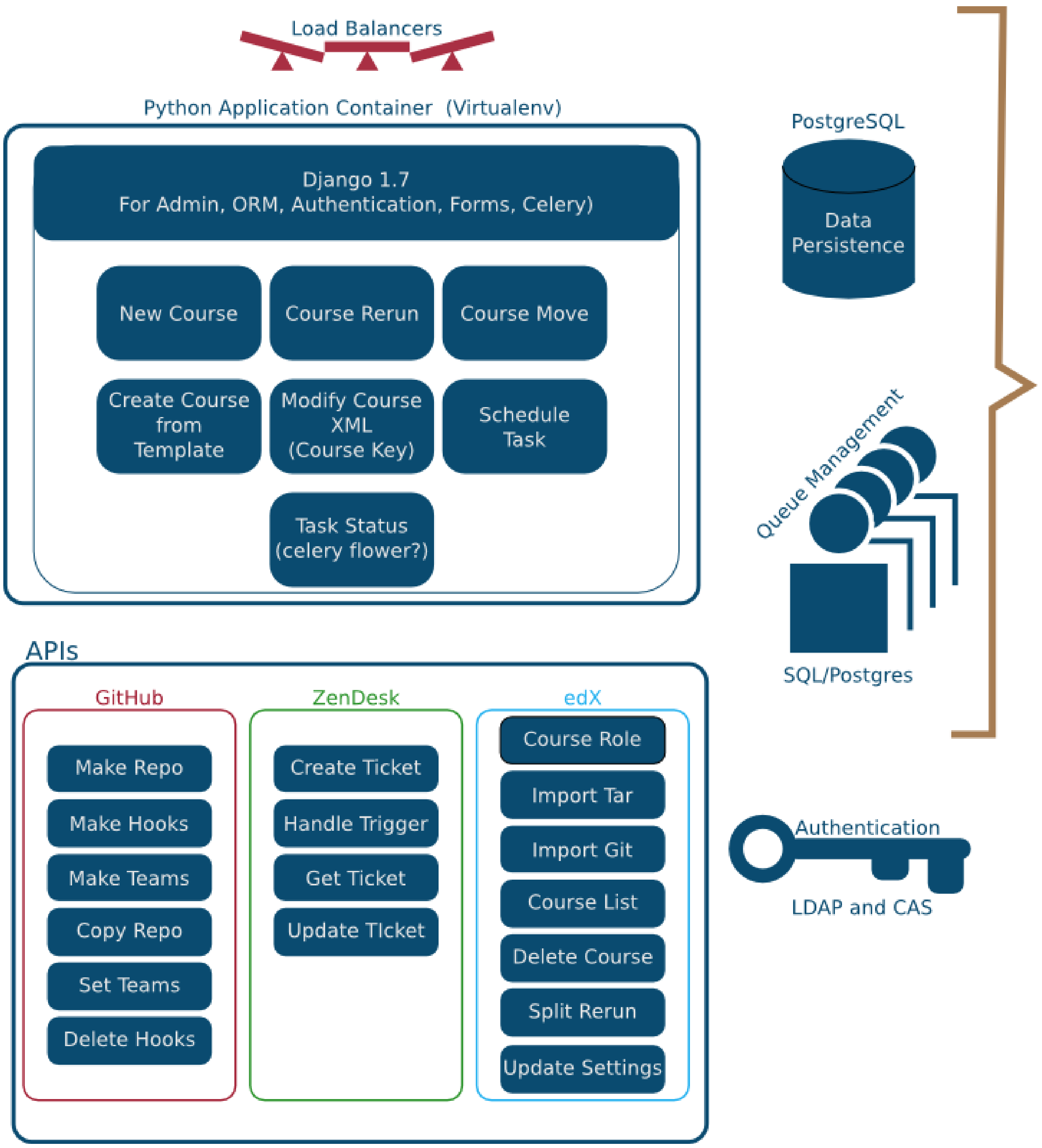

Fig. 3.4: System Architecture diagram

- We have two of the eight needed API calls implemented in a feature branch:
  - Import Tar
  - Course List and we can likely easily add the rest if that feature branch is merged.

3.2.6 Data Structure/Entity Relationship Diagram

Below is an Oracle style ERD diagram, but I do not presuppose the database choice, and do not think it is neccessary to use a traditional RDBMS as the persistance layer in the application. The persistence area is a flexible area within the architecture since even a simple document store would likely be adequate.

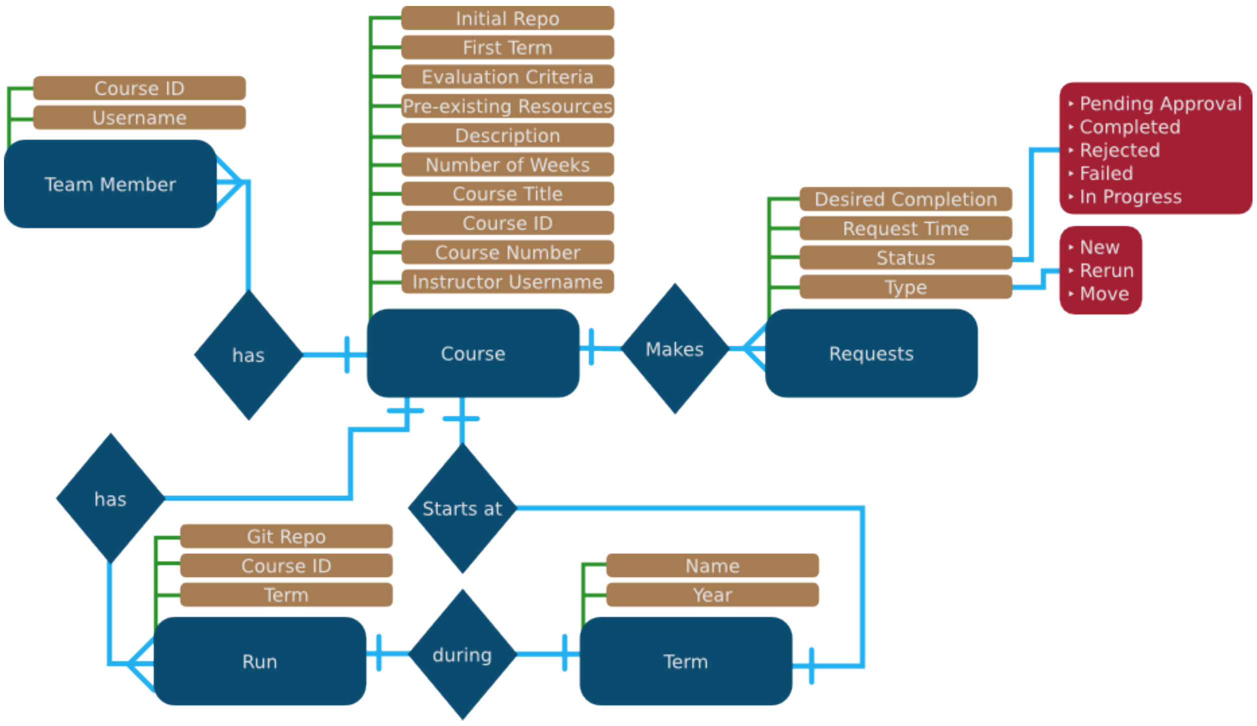

Fig. 3.5: Entity Relationship Diagram

3.3 Orcoursetrion API Docs

For convenient reference in development, here are the Orcoursetrion API docs.

3.3.1 Actions

The actions that are available to use. Action library access

`orcoursetrion.actions.`**`create_export_repo`**(*course*, *term*, *description=None*)

Creates a studio based course repo at *`ORC_GH_API_URL`* with key *`ORC_GH_OAUTH2_TOKEN`*, at organization *`ORC_STUDIO_ORG`*, and with collabarator *`ORC_STUDIO_DEPLOY_TEAM`*

Raises

- `requests.RequestException`

- *orcoursetrion.lib.GitHubUnknownError*
- *orcoursetrion.lib.GitHubNoTeamFound*
- *orcoursetrion.lib.GitHubRepoExists*

Parameters

- **course** (*str*) – Course name to be used to name repo (i.e. 6.004r)
- **term** (*str*) – Term the course is expected to run (i.e. 2015_Spring)
- **description** (*str*) – Optional description for repo to show up on github

Returns Github dictionary of a repo (https://developer.github.com/v3/repos/#create)

Return type dict

`orcoursetrion.actions.`**`rerun_studio`**(*course*, *term*, *new_term*, *description=None*)

Run any actions needed to re-run a Studio course.

This will remove the hooks from the specified term, and then create a new export repo for the `new_term`. It finds the repo based on *ORC_STUDIO_ORG* as the organization, *ORC_COURSE_PREFIX* as the prefix, replacing all the dots in `course` and appending "-`term`".

Parameters

- **course** (*str*) – Course name to be used to name repo (i.e. 6.004r)
- **term** (*str*) – Term the course is last run (i.e. 2015_Spring)
- **new_term** (*str*) – Term the course is expected to run again (i.e. 2018_Spring)
- **description** (*str*) – Optional description for repo to show up on github

Raises

- `requests.RequestException`
- *orcoursetrion.lib.GitHubRepoDoesNotExist*
- *orcoursetrion.lib.GitHubUnknownError*

Returns Github dictionary of the newly created repo (https://developer.github.com/v3/repos/#create)

Return type dict

`orcoursetrion.actions.`**`release_studio`**(*course*, *term*)

Moves a studio course to be ready for production.

Currently this will just add a hook to the production server, but it will eventually take care of everything else needed for a transfer as well.

Parameters

- **course** (*str*) – Course name of the repo to release to production (i.e. 6.001)
- **term** (*str*) – Term the course is currently running in (i.e. 2015_Spring)

Raises

- `requests.RequestException`
- *orcoursetrion.lib.GitHubUnknownError*

Returns Nothing returned, raises on failure

Return type None

`orcoursetrion.actions.`**`create_xml_repo`**(*course*, *term*, *team=None*, *members=None*, *description=None*)

Creates a course repo at *`ORC_GH_API_URL`* with key *`ORC_GH_OAUTH2_TOKEN`* and at organization *`ORC_XML_ORG`*, and with `team` as a collaborator (Along with *`ORC_XML_DEPLOY_TEAM`*).

If `team` is not provided, then it will be generated with *`ORC_COURSE_PREFIX`*, `course`, and `term`

If `members` is provided, the `team` membership will be *replaced* with the members listed. It will also create the team if it doesn't already exist regardless of the value of `members`.

This also adds a github Web hook to the course development environment gitreload server via *`ORC_STAGING_GITRELOAD`*.

Raises

- `requests.RequestException`
- *`orcoursetrion.lib.GitHubUnknownError`*
- *`orcoursetrion.lib.GitHubNoTeamFound`*
- *`orcoursetrion.lib.GitHubRepoExists`*

Parameters

- **`course`** (*str*) – Course name to be used to name repo (i.e. 6.004r)
- **`term`** (*str*) – Term the course is expected to run (i.e. 2015_Spring)
- **`team`** (*str*) – Name of an organizational team that already exists to add read/write access to this repo.
- **`members`** (*list*) – Exclusive list of usernames that should be on the team.
- **`description`** (*str*) – Optional description for repo to show up on github

Returns Github dictionary of a repo (https://developer.github.com/v3/repos/#create)

Return type dict

`orcoursetrion.actions.`**`rerun_xml`**(*course*, *term*)

Run any actions needed to re-run an XML course.

Currently this only deletes the Web hooks, but eventually it will also copy the repo to a history clean one, and setup up that new one with hooks. It finds the repo based on *`ORC_XML_ORG`* as the organization, *`ORC_COURSE_PREFIX`* as the prefix, replacing all the dots in `course` and appending "-`term`".

Parameters

- **`course`** (*str*) – Course name to be used to name repo (i.e. 6.004r)
- **`term`** (*str*) – Term the course is expected to run (i.e. 2015_Spring)

Raises

- `requests.RequestException`
- *`orcoursetrion.lib.GitHubUnknownError`*

Returns Number of hooks removed

Return type int

`orcoursetrion.actions.`**`release_xml`**(*course*, *term*)

Moves an XML course to be ready for production.

Currently this will just add a hook to the production server, but it will eventually take care of everything else needed for a transfer as well (i.e. making live branch, import it to lms...).

Parameters

- **`course`** (*str*) – Course name of the repo to release to production (i.e. 6.001)
- **`term`** (*str*) – Term the course is currently running in (i.e. 2015_Spring)

Raises

- `requests.RequestException`
- *`orcoursetrion.lib.GitHubUnknownError`*

Returns Nothing returned, raises on failure

Return type None

`orcoursetrion.actions.`**`put_team`**(*org*, *team*, *read_only*, *members*)

Create or update a team with the list of `members`.

If `members` is None, the team will be created if it doesn't exist, but membership will not be changed.

Parameters

- **`org`** (*str*) – Organization that owns/should own the team.
- **`team`** (*str*) – Name of the team.
- **`read_only`** (*bool*) – True if pull access, False if push access
- **`members`** (*list*) – Exclusive list of usernames that should be on the team.

Raises

- `requests.RequestException`
- *`orcoursetrion.lib.GitHubUnknownError`*
- *`orcoursetrion.lib.GitHubRepoExists`*

Returns Github team dictionary (https://developer.github.com/v3/orgs/teams/#response-1)

Return type dict

3.3.2 Library

API libraries. Orchestrion library

class `orcoursetrion.lib.`**`GitHub`**(*api_url*, *oauth2_token*)

Bases: `object`

API class for handling calls to github

Initialize a requests session for use with this class by specifying the base API endpoint and key.

Parameters

- **`api_url`** (*str*) – Github API URL such as https://api.github.com/
- **`oauth2_token`** (*str*) – Github OAUTH2 token for v3

`add_repo_file`(*org*, *repo*, *committer*, *message*, *path*, *contents*)

Adds the `contents` provided to the `path` in the repo specified and committed by the `commiter` parameters provided.

https://developer.github.com/v3/repos/contents/#create-a-file

Note: This commits directly to the default branch of the repo.

Parameters

- **`org`** (*str*) – Organization the repo lives in.
- **`repo`** (*str*) – The name of the repo.
- **`committer`** (*dict*) – {'name': ..., 'email': ...} for the name and e-mail to use in the initial commit of the destination repo.
- **`message`** (*str*) – Commit message to use for the addition.
- **`path`** (*str*) – The content path, i.e. `docs/.gitignore`
- **`contents`** (*str*) – The actual string Contents of the file.

Raises

- `requests.exceptions.RequestException`
- *`GitHubRepoDoesNotExist`*
- *`GitHubUnknownError`*

Returns None

`add_team_repo`(*org*, *repo*, *team*)

Add a repo to an existing team (by name) in the specified org.

We first look up the team to get its ID (https://developer.github.com/v3/orgs/teams/#list-teams), and then add the repo to that team (https://developer.github.com/v3/orgs/teams/#add-team-repo).

Parameters

- **`org`** (*str*) – Organization to create the repo in.
- **`repo`** (*str*) – Name of the repo to create.
- **`team`** (*str*) – Name of team to add.

Raises

- *`GitHubNoTeamFound`*
- *`GitHubUnknownError`*
- `requests.exceptions.RequestException`

`add_web_hook`(*org*, *repo*, *url*)

Adds an active hook to a github repository.

This utilizes https://developer.github.com/v3/repos/hooks/#create-a-hook to create a form type Web hook that responds to push events (basically all the defaults).

Parameters

- **`org`** (*str*) – Organization to create the repo in.
- **`repo`** (*str*) – Name of the repo the hook will live in.
- **`url`** (*str*) – URL of the hook to add.

Raises

- *`GitHubUnknownError`*
- `requests.exceptions.RequestException`

Returns Github dictionary of a hook (https://developer.github.com/v3/repos/hooks/#response-2)

Return type dict

create_repo (*org*, *repo*, *description*)

Creates a new github repository or raises exceptions

Parameters

- **org** (*str*) – Organization to create the repo in.
- **repo** (*str*) – Name of the repo to create.
- **description** (*str*) – Description of repo to use.

Raises

- *GitHubRepoExists*
- *GitHubUnknownError*
- requests.exceptions.RequestException

Returns Github dictionary of a repo (https://developer.github.com/v3/repos/#create)

Return type dict

delete_web_hooks (*org*, *repo*)

Delete all the Web hooks for a repository

Uses https://developer.github.com/v3/repos/hooks/#list-hooks to get a list of all hooks, and then runs https://developer.github.com/v3/repos/hooks/#delete-a-hook to remove each of them. :param org: Organization to create the repo in. :type org: str :param repo: Name of the repo to remove hooks from. :type repo: str

Raises

- *GitHubUnknownError*
- *GitHubRepoDoesNotExist*
- requests.exceptions.RequestException

Returns Number of hooks removed

Return type int

put_team (*org*, *team_name*, *read_only*, *members*)

Create a team in a github organization.

Utilize https://developer.github.com/v3/orgs/teams/#list-teams, https://developer.github.com/v3/orgs/teams/#create-team, https://developer.github.com/v3/orgs/teams/#list-team-members, https://developer.github.com/v3/orgs/teams/#add-team-membership, and https://developer.github.com/v3/orgs/teams/#remove-team-membership. to create a team and/or replace an existing team's membership with the `members` list.

Parameters

- **org** (*str*) – Organization to create the repo in.
- **team_name** (*str*) – Name of team to create.
- **read_only** (*bool*) – If false, read/write, if true read_only.
- **members** (*list*) – List of github usernames to add to the team. If none, membership changes won't occur

Raises

- `GitHubUnknownError`
- `requests.RequestException`

Returns The team dictionary (https://developer.github.com/v3/orgs/teams/#response-1)

Return type dict

static shallow_copy_repo(*src_repo*, *dst_repo*, *committer*, *branch=None*)

Copies one branch repo's contents to a new repo in the same organization without history.

Danger: This will overwrite the destination repo's default branch and rewrite its history.

The basic workflow is:

- Clone source repo
- Remove source repo `.git` folder
- Initialize as new git repo
- Set identity
- Add everything and commit
- Force push to destination repo

Parameters

- **src_repo** (*str*) – Full git url to source repo.
- **dst_repo** (*str*) – Full git url to destination repo.
- **committer** (*dict*) – {'name': ..., 'email': ...} for the name and e-mail to use in the initial commit of the destination repo.
- **branch** (*str*) – Option branch, if not specified default is used.

Raises `sh.ErrorReturnCode`

Returns None

exception `orcoursetrion.lib.`**GitHubException**

Bases: `exceptions.Exception`

Base exception class others inherit.

exception `orcoursetrion.lib.`**GitHubRepoExists**

Bases: `orcoursetrion.lib.github.GitHubException`

Repo exists, and thus cannot be created.

exception `orcoursetrion.lib.`**GitHubRepoDoesNotExist**

Bases: `orcoursetrion.lib.github.GitHubException`

Repo does not exist, and therefore actions can't be taken on it.

exception `orcoursetrion.lib.`**GitHubUnknownError**

Bases: `orcoursetrion.lib.github.GitHubException`

Unexpected status code exception

exception `orcoursetrion.lib.`**GitHubNoTeamFound**

Bases: `orcoursetrion.lib.github.GitHubException`

Name team not found in list

3.3.3 Configuration

Configuration options Configuration needed for Orchestrion to function (i.e. API keys)

config.**ORC_GH_OAUTH2_TOKEN = GitHub OAUTH2 Token**

config.**ORC_GH_API_URL = GitHub API URL**

config.**ORC_GH_NAME = Git committer name to use.**

config.**ORC_GH_EMAIL = Git committer e-mail to use**

config.**ORC_COURSE_PREFIX = Prefix to use in repository name**

config.**ORC_STUDIO_ORG = Organization to use for Studio export repos**

config.**ORC_STUDIO_DEPLOY_TEAM = Deployment team for Studio Export repos**

config.**ORC_XML_ORG = Organization to use for XML/latex2edx courses**

config.**ORC_XML_DEPLOY_TEAM = Deployment team for XML/latex2edx courses**

config.**ORC_STAGING_GITRELOAD = ‘gitreload <https://github.com/mitodl/gitreload>‘_ server URL (including username a**

config.**ORC_PRODUCTION_GITRELOAD = ‘gitreload <https://github.com/mitodl/gitreload>‘_ server URL (including userna**

CHAPTER 4

Indices and Search

- genindex
- modindex
- search

Index