OLCMS Documentation

Release 2018-07-03

Stem4youth consortium

Jul 25, 2018

Contents

CHAPTER 1

Intended audience

This guide is mainly for technical people that want to re-use our solution.

CHAPTER 2

Contents:

2.1 Glossary

- `Content Item` — is a smallest piece of information that is usable by teacher on it's own. Some partners call it `Lesson`. Content items contain mostly metadata, actual pieces of content are stored as `Resource` items.
- `Resource` — single material (e.g. document) usable by teacher.
- `Domain` — A categorisation system for Content Items with predefined categories.
- `Tag` — User provided categories (tags) for Content Items.

2.2 Contributing Guide

2.2.1 How to configure developer environment

System configuration

We assume you have a linux based system. Technically everything should be doable using docker on Windows, but we did not try it.

Hosts

You'll need to set up mapping in: `/etc/hosts` from `olcms.dev.local` to localhost IP, for example:

```
127.0.2.1       olcms.dev.local
```

Note: It is required for local swift datastore, due to swift internal details container must be resolvable under the same domain/ip on both localhost (so your browser can access it) and inside django container (so django can access it), since

`localhost` won't work, as well as using raw IP, best way I could thought of was just using a domain.

Docker

You'll need to install `docker` (any recent version) and `docker-compose` (any recent version, tested on `1.17.1`).

Also make sure you have rights to call docker command, to check it see if `docker info` exits successfully

Setting the system up

```
touch .local.dev.env
./dev.sh build
./dev.sh init
./dev.sh up
```

Sanity check

Run `./dev.sh ps` and see all services are up.

Basic set-up

- See "Creating user accounts"

2.2.2 Developer environment features

Email Server

In development, it is often nice to be able to see emails that are being sent from your application. For that reason local SMTP server *MailHog* with a web interface is available as docker container.

With MailHog running, to view messages that are sent by your application, open your browser and go to `http://127.0.0.1:8025`

Youtube

To work with the upload of videos to youtube service developer needs to create locally .local.dev.env file, next to .dev.env.

In this file add two variables: YOUTUBE_SECRET_JSON and YOUTUBE_OAUTH2_JSON. As values for this variables use data from olcms-secrets project under yoututbe group. Management of the youtube account is done in cellery tasks. To see logs from this operations open logs for celleryworker process in the docker compose. Also remember to set correctly environment variable YT_PLAYLIST_ID with the correct id.

Note: See "Obtaining youtube credentials" on how to obtain youtube credentials.

Run elasticsearch locally

We use different indexing service locally than in production. Locally we use whoosh, and elasticsearch on production. Main reason for this difference is that Elasitc search eats a lot of RAM.

To use elastic search locally follow instructions below:

To start a local cluster using elastic search you can use: `docker-compose -f docker-compose-elasticsearch.yml` and then usuall `up`, `run django bash` (and so on).

You can access elastic search gui, by using: `http://localhost:9200/_plugin/gui/#/inspect`.

2.2.3 Common developer tasks

Creating users

- To create normal user account just go create account normally, to verify emails go to `http://localhost:8025` where all emails sent from the system are presented.
- To create superuser just call: `./dev.sh manage createsuperuser`.

Running tests

To run all tests call: `./dev.sh test` (this takes some time), once you isolate failing tests you may re-run just them: `./dev.sh test www.content_item`.

Running linter

Run `./dev.sh lint`.

Starting bash shell in the container

Run `./dev.sh bash`.

Generate translation files

Start shell in the container `./dev.sh bash` and then do following:

- `cd /app`
- `./manage.py makemessages` — this will extract strings from all applications.
- `cd www/templates`
- `./manage.py makemessages` — this will extract strings from templates

2.2.4 Not so common developement tasks

Debugging selenium tests

If your selenium tests start to fail, here is what you can do:

1. When you run tests screenshots and page HTML dumps are stored in the `screenshots`, usually there are more than one screen-shoot per test. File names have following format: `test_name-specifier.extension`, where specifier describes the screen-shots/page dump. At the end of each test (both for successes and failures) final screen-shots is stored it has `final` specifier, so the name looks like: `www.integration_tests.test_content_item.ContentItemTest.test_add_content_item_with_default_author.final.png`.

   Screenshots are easiest way of spotting errors in your tests.

2. Try adding more screenshots `self.save_screenshot("before-submit")` at critical moments of failing test.

3. If the tests are failing locally skip to the next (`Running selenium tests on a local browse`) section.

   If tests fail only on CI and not on your computer you might try running `stress-ng` locally this will simulate noisy-neighbours on CI and hopefully enable you to reproduce.

4. If tests fail on CI, things get harder. Start with running ssh console on our CI instance: Click launch SSH console on the failing job.

5. Now you can start tests manually (starting with `docker-compose -f dev.yaml run django bash`).

6. To pinpoint the issue I usually used pudb (console debugger).

Running selenium tests on a local browser

Easiest way to pinpoint selenium failures is to run selenium tests on your local browser.

You'll need to:

1. Download geckodriver and add it to path. You can use this handy script (taken from `Dockerfile-dev`.

```
curl https://ocs-pl.oktawave.com/v1/AUTH_385fff76-290b-43da-b2fc-96b1c08bce24/
↪tools/geckodriver.xz | xzcat > /usr/local/bin/geckodriver  && \
# Verify file integrity (did checksumming myself)
sha256sum /usr/local/bin/geckodriver | grep --quiet␣
↪469052c6bf2c4f6e0b07f32d50ba0ff21b20e83d132468ec389050734dd23142 && \
chmod +x /usr/local/bin/geckodriver
```

2. There **might** be some way to run tests in docker, on the local browser but I have no idea how to configure it, so in this case we'll just run django outside of container.

3. Create a virtualenv on python 3.5

4. Install all requirements: `pip install -r requirements/local.txt && pip install -r requirements/test.txt`

5. Source proper variables: `source compose/django/entrypoint-local.sh`.

6. If you want to use search: `./manage.py rebuild_index`.

7. `./manage.py test`. Browser should start, and you'll see what selenium does.

8. It is useful to run `./manage.py test` in debugging mode and then manually stepping through the code.

Debugging indexes

When you encounter incorrect search results in development enviorment, inspecting index contents can help:

```
from whoosh.index import open_dir
ix = open_dir("whoosh_index_dir")
print(list(ix.searcher().documents()))
```

Debugging hanging tests

If selenium hangs you can try to debug it by sending USR1 signal to `./manage.py test`, eg. by execing to container and then running: `pkill -USR1 -f test`. This will dump current stack-trace to `stacktrace.log` log in your OLCMS folder.

Obtaining youtube credentials

ref:*See here <obtain-youtube-credentials>*.

2.2.5 How to add a new feature

1. First get in touch with us e.g. by creating an issue: https://bitbucket.org/olcms/www/issues?status=new&status=open.
2. Then write the feature.
3. Make sure to add enough tests too keep coverage 100%.

2.2.6 What to test

For now I want to have 100% coverage on Python code, and good coverage on in the functional Selenium UI tests.

1. Create unit tests that cover everything, except UI templating stuff.
2. Explicitly test for permissions — check that users without permission have no access.
3. Test rest framework. Test permissions there as well.
4. Create automatic selenium tests for UI, Focus here on pure frontend aspects of the system.

How to test

Django unittest facilities are explained here: https://docs.djangoproject.com/en/1.10/topics/testing/tools/

Integration tests live in `www.integration_tests`, while unit tests are inside each application.

Selenium

Selenium is slightly harder to use. Start with extending: `LiveServerBrowserTestCase`, it runs firefox browser using a virtual framebuffer. Then you have access to selenium driver using (unsurprisingly) `self.driver`.

There are some helper methods in `SeleniumUtilsMixin`, most notably you'll use `self.login` (`LiveServerBrowserTestCase` extends that mixin) that logs in `self.user` to OLCMS using the browser.

2.3 Deploy

2.3.1 Third party system you'll need to configure

email This application sends emails to it's users, please set-up SMTP account for it (or better yet) sent emails using some dedicated service (mailgun, mailinator*).

Email volume should be reasonably small.

- Keep in mind GDPR requirements with overseas providers, in wake of

possible Privacy Shield revocation. Also — I'm not a lawyer.

youtube Video resources upload videos to youtube.

You'll need to obtain valid youtube credentials. See *how to obtain youtube credentials*

Swift file storage Files uploaded to our repository are hosted in OpenStack Swift Service, which is free/open-source system similar to S3, that is it delivers easy-to use file storage service.

You can find providers here: https://www.openstack.org/marketplace/.

Database Postgres database. You can either deploy your own PostrgreSQL server, add docker image to the `docker-compose` file you use or just use database from your hosting provider (e.g. RDS).

Configuring swift

Rationale for swift configuration is here

We use SWIFT such way that users can upload files directly there.

To do this we use Swift cluster using two kinds of middleware:

1. `temp url` middleware. This middleware allows us to generate temporary urls, that allow user to perform a single HTTP verb (GET or PUT). After timeout these urls become invalid.
2. `CORS` middleware, enables Swift to server proper `CORS` headers.

To configure swift container you'll need:

- Container name
- Authorization information.

In this tutorial I'll assume that you have swift client software installed, also I'll refer as `swift-authorized` to swift client with proper authorization flags e.g. `swift -A https://my.swift.service.org/auth/v1.0 -U username -P password`.

To mark container as publicly readable you'll need to issue following command: `swift-authorized post container_name -r '.r:*`,

To set temp url key you'll need: `swift-authorized post container_name -H "X-Container-Meta-Temp-URL-Key:{temp_url_key}"`. Temp url key should be a long random string that matches `SWIFT_TEMP_URL_KEY` setting in the settings.

You'll also need to update CORS headers:

```
swift-authorized post container_name \
      "-H 'X-Container-Meta-Access-Control-Allow-Origin:*' " \
      "-H 'X-Container-Meta-Access-Control-Expose-Headers:*' " \
      "-H 'X-Container-Meta-Access-Control-Allow-Headers: *' "
```

Note that since upload rights will be guarded by temporary urls permissive CORS are not a big problem.

2.3.2 VM requirements

We use `2 VCPU, 2 GB RAM` server with `40GB` disk drive, and it is enough.

2.3.3 Building images

To build production docker images use following command: `./dev.sh push-prod`.

Proper tags are also build whenever someone updates `master` and `preprod` branches in our repository.

2.3.4 Prepare environment variables

Meta variables

Following are meta variables used in many different variables:

- **olcms domain** — we will use `olcms.example.com` thought the examples.
- **discourse domains** — we will use `discourse.example.com` thought the examples.

  See here for forum integration here.

Django environment

Variables for django images (in `.dj_env` file using docker compose deployment below.

Generic settings

- `USE_HTTPS` set to 1 if you want to force https (which is good thing)
- `DJANGO_ADMIN_URL: "/admin"` — url for administrative interface
- `DJANGO_SETTINGS_MODULE: "config.settings.production"` — settings module used
- `DJANGO_SECRET_KEY` — set this to random string.
- `DJANGO_ALLOWED_HOSTS: "*"` — if you set this to hostname, eg: “olcms.my.project.eu” OLCMS will deny HTTP requests with other host names
- `DJANGO_SECURE_SSL_REDIRECT` if true django redirects http to https. In our case nginx also does that.
- `DJANGO_ACCOUNT_ALLOW_REGISTRATION` set to 1 allow users to register themselves (and create new content).
- `OLCMS_DOMAIN: olcms.example.com` domain for your instance.
- `CELERY_BROKER_URL: redis://redis/0` — in all cases we use built-in redis docker image.

Database settings

- `POSTGRES_HOST` — hostname of postgres database
- `POSTGRES_USER` — username for postgres user, as well as name of database to use
- `POSTGRES_PASSWORD` — postgresql connection password

Swift settings

- `SWIFT_AUTH_URL: https://swift.example.com/auth/v1.0` — Swift authorization url. Consult your provider documentation for details.

  Our settings support (rather old) authorization version `1.0` (as our provider supports only that. Adapting to never auth should be straightforward.
- `SWIFT_USERNAME`: Username for swift administrator
- `SWIFT_KEY`: Key (password) for swift administrator user
- `SWIFT_CONTAINER_NAME`: Swift container to store data in
- `SWIFT_TEMP_URL_KEY`: Key to generate temporary upload urls, see *Rationale for swift configuration here*. This value *needs to match swift configuration*
- `SWIFT_TEMP_URL_DURATION` duration for which temporary urls generated by file upload api will be valid. **in seconds**.

SMTP settings

SMTP service used to send emails. Configuring OLCMS to use e.g. mailgun API endpoint will require changes to `DJANGO_SETTINGS_MODULE`.

- `DJANGO_SERVER_EMAIL` — `From:` email for all sent mail.
- `DJANGO_EMAIL_USER` — login user for SMTP server
- `DJANGO_EMAIL_HOST` — hostname for SMTP server
- `DJANGO_EMAIL_PORT` — port for SMTP connection (unsurprisingly)
- `DJANGO_EMAIL_USE_SSL` — set to 1 if you want to use "SSL" for SMTP connection
- `DJANGO_EMAIL_PASS` — password for `DJANGO_EMAIL_USER`

Youtube settings

See this section on how to obtain youtube settings.

- `YOUTUBE_OAUTH2_JSON`
- `YOUTUBE_SECRET_JSON`
- `YT_PLAYLIST_ID`

Google analytics settings

- `GOOGLE_ANALYTICS_TRACKING_CODE: "UA-0000001-1"` — we support google analytics user tracking, if you wish to use it, obtain google analytics and put it here. If missing user tracking won't be used.

Forum integration

- `ENABLE_DISCOURSE_FORUM_TRIGGERS` — if you set this to `1` olcms integration with discourse. *See here for forum integration here*.

  If you set to `0` you can omit rest of these variables.

- `DJANGO_DISCOURSE_SSO_URL: https://discourse.example.com/session/sso_login` — discourse sso url it will be used to redirect users after single sign on login to discourse forum.
- `DJANGO_DISCOURSE_ROOT_URL: https://discourse.example.com/` — root url for discourse installation, used to install forum "comments" under materials.
- `COMPOSE_DISCOURSE_SSO_SECRET`: Discourse SSO secret. Set this to a long random string. Used both by django and discourse image.

Nginx configuration

Please note that the same cert used for olcms service as well as discourse service. We just use wildcard certificate.

- `OLCMS_DOMAIN: olcms.example.com` domain for your instance. Please note that this setting is used both in django and nginx configurations
- `NGINX_CERT` base64 encoded certificate file in PEM format. To obtain this value use following command (assuming certificate is named `my.crt`): `cat my.crt | base64 -w0`. You should get long string starting with `LS0tLS1CRUdJTi`.
- `NGINX_KEY` base64 encoded certificate key in PEM format. To obtain this value use following command (abusing key file is named `my.key`): `cat my.key | base64 -w0`. You should get long string starting with `LS0tLS1CRUdJTi`.

Discourse configuration

We use 'unsupported' (by discourse team) method of deploying Discourse — we want:

- To treat discourse as "more or less" stateless service
- Be able to deploy discourse automatically
- To automatically configure discourse using 7 factor app.

Feel free to use our deployment scripts, feel free also to use official docker image for discourse (and of course, feel free to not use discourse at all). In any case integrating stock discourse service should be relatively straightforward.

See forum integration for full information

Discourse misc settings

Discourse settings that should just be left without changes, used by discourse/ruby configuration.

They should be

- RAILS_ENV: production
- RUBY_GLOBAL_METHOD_CACHE_SIZE: 131072
- RUBY_GC_HEAP_GROWTH_MAX_SLOTS: 40000
- RUBY_GC_HEAP_INIT_SLOTS: 400000
- RUBY_GC_HEAP_OLDOBJECT_LIMIT_FACTOR: 1.5
- UNICORN_WORKERS: 3
- UNICORN_SIDEKIQS: 1

Configuration

- DISCOURSE_HOSTNAME: discourse.example.com
- DISCOURSE_REDIS_HOST: redis_disco — redis instance configured for discourse in our compose file.

SMTP configuration, consult discourse documentation if anything is not obvious.

- DISCOURSE_SMTP_ADDRESS
- DISCOURSE_SMTP_PORT
- DISCOURSE_SMTP_USER_NAME
- DISCOURSE_SMTP_PASSWORD
- DISCOURSE_SMTP_ENABLE_START_TLS: false
- DISCOURSE_DB_USERNAME
- DISCOURSE_DB_PASSWORD
- DISCOURSE_DB_HOST
- DISCOURSE_DB_NAME
- DISCOURSE_REDIS_HOST

Integration with Django

- COMPOSE_DISCOURSE_SSO_SECRET: Discourse SSO secret. Set this to a long random string. Used both by django and discourse image.
- COMPOSE_DISCOURSE_SSO_PROVIDER_URL: https://olcms.example.com/discourse/sso sso endpoint
- DISCOURSE_ENABLE_CORS: true — needed for SSO
- DISCOURSE_CORS_ORIGIN: "*" — you probably should limit this to prevent security issues.

Misc configuration

Admin details, admin user with following email data will be created, to login via SSO you'll need to create user with the same e-mail in OLCMS.

- COMPOSE_DISCOURSE_ADMIN_EMAIL: admin@ourdomain
- COMPOSE_DISCOURSE_ADMIN_USERNAME: admin

- `COMPOSE_DISCOURSE_ADMIN_PASSWORD: password`

Misc configuration:

- `COMPOSE_DISCOURSE_INVITE_ONLY: t` — disable self registration (extra caution)
- `COMPOSE_DISCOURSE_LOGIN_REQUIRED: f` — don't require login to read posts — required for comment integration
- `COMPOSE_DISCOURSE_BOOTSTRAP_MODE: f` — Disable bootstrap mode.
- `COMPOSE_DISCOURSE_TITLE: Discussion forum for OLCMS` — forum title
- `COMPOSE_DISCOURSE_SITE_DESCRIPTION: A description` — forum description
- `COMPOSE_DISCOURSE_CONTACT_EMAIL: contact@example.com` — contact e-mail

2.3.5 Deployment

1. Obtain all credentials third-party services described above.
2. Install docker on a linux server.
3. Install docker compose.
4. *Install docker compose file from here*.

   Please note that images related to `discourse forum` are optional.
5. Create environment file
6. Start compose service.

2.3.6 Docker compose file we use

```
version: '2'

volumes:
  elastic_data: {}
  videotmp: {}
  discourse_assets: {}
  discourse_shared: {}
  discourse_logs: {}

services:

  elasticsearch:
    image: elasticsearch:2.4
    volumes:
      - elastic_data:/usr/share/elasticsearch/data
    environment:
      - ES_JAVA_OPTS=-Xms350m -Xmx350m
    logging:
      driver: 'json-file'
      options:
        "max-size": "10mb"
        "max-file": "1"

  # Runs migrations during startup.
  django_migrations:
```

(continues on next page)

(continued from previous page)

```
    image: olcms/www:latest
    user: django
    command: /scripts/prod-init.sh
    env_file: .env
    volumes:
      - videotmp:/videotmp
    logging:
      driver: 'json-file'
      options:
        "max-size": "10mb"
        "max-file": "1"

  # Updates elastic search index.
  update_index:
    image: olcms/www:latest
    user: django
    command: /scripts/update-index.sh
    env_file: .env
    logging:
      driver: 'json-file'
      options:
        "max-size": "10mb"
        "max-file": "1"

  # Django application server
  django:
    image: olcms/www:latest
    user: django
    depends_on:
      - redis
      - django_migrations
      - elasticsearch
    volumes:
      - videotmp:/videotmp
    command: /scripts/gunicorn.sh
    env_file: .env
    logging:
      driver: 'json-file'
      options:
        "max-size": "100mb"
        "max-file": "5"

  # Nginx webserver
  nginx:
    image: olcms/nginx:latest
    env_file: .env
    entrypoint: /entrypoint.sh
    command: /start.sh
    depends_on:
      - django
      - discourse
    ports:
      - "0.0.0.0:80:80"
      - "0.0.0.0:443:443"

    logging:
      driver: 'json-file'
```

(continues on next page)

(continued from previous page)

```
    options:
      "max-size": "100mb"
      "max-file": "3"

#Redis cache
redis:
  image: redis:3.0-alpine
  logging:
    driver: 'json-file'
    options:
      "max-size": "100mb"
      "max-file": "1"

# Celery task queue
celeryworker:
  image: olcms/www:latest
  user: django
  env_file: .env
  ports: []
  depends_on:
   - redis
   - django_migrations
  command: celery -A www.taskapp worker -l INFO --concurrency 2
  volumes:
    - videotmp:/videotmp
  logging:
    driver: 'json-file'
    options:
      "max-size": "100mb"
      "max-file": "3"

# Celery task queue
celerybeat:
  image: olcms/www:latest
  user: django
  env_file: .env
  ports: []
  depends_on:
    - redis
    - django_migrations
  command: celery -A www.taskapp beat -l INFO
  volumes:
    - videotmp:/videotmp
  logging:
    driver: 'json-file'
    options:
      "max-size": "100mb"
      "max-file": "2"

# Separated worker for converting documents to other formats.
document_converter:
  image: olcms/documentconverter
  links:
    - redis
  environment:
    - WORKER_QUEUE=converter
    - CELERY_BROKER_URL=redis://redis/0
```

(continues on next page)

(continued from previous page)

```
    logging:
      driver: 'json-file'
      options:
        "max-size": "100mb"
        "max-file": "2"

  ################################################################################
  ## OPTIONAL DISCOURSE IMAGES
  ################################################################################

  redis_disco:
    image: redis:3.0-alpine
    logging:
      driver: 'json-file'
      options:
        "max-size": "100mb"
        "max-file": "1"

  discourse:
    ports:
      - "8081:80"
    volumes:
      - discourse_assets:/var/www/discourse/public/
      - discourse_shared:/shared/
      - discourse_logs:/var/log
    command: /discourse-composite/run/scripts/discourse.start.sh
    image: olcms/discourse-composite:2018-01-24
    depends_on:
      - redis_disco
      - discourse_migrate
    links:
      - redis_disco
      - discourse_migrate
    env_file: .env
    logging:
      driver: 'json-file'
      options:
        "max-size": "100mb"
        "max-file": "2"

  discourse_migrate:
    volumes:
      - discourse_assets:/var/www/discourse/public/
      - discourse_shared:/shared/
      - discourse_logs:/var/log
    command: /discourse-composite/run/scripts/discourse.migrate.sh
    image: olcms/discourse-composite:2018-01-24
    depends_on:
      - redis_disco
    links:
      - redis_disco
    env_file: .env
    logging:
      driver: "json-file"
      options:
        max-size: "100kb"
        max-file: "5"
```

2.4 Content management system

By Content Management System (or CMS) I refer to a functionality that allows you to add text (and/or other content snippets) to many different ContentItems or just to normal static pages.

2.4.1 Markdown

Markdown is a lightweight markup language that has very readable source text, and renders to plain simple HTML.

In most cases where we wanted users to provide simple text with some optional formatting we ask them to provide it in markdown format, which forces user not to complicate things too much.

Here is useful markdown primer

2.4.2 CMS Pages

Cms pages functionality allow you to create static (that is not changing) pages at designated urls in the interface.

These pages may (and should!) be translated to languages for your users.

To create a page:

- Go to admin page
- Add new Translatable page.

  Here just fill page `slug`, if your site is at `olcms.example.com` and you add slug equal to `mycmspage` your page will be visible at `http://olcms.example.com/mycmspage`
- Add page translations:
  - `parent` — translatable page (created in previous step)
  - `language` — translation language. *See here on how to tweak available languages*.
  - `title` — page title
  - `default` — if true we will present this page if we don't have "proper" language for user.
  - `content` here you add content :)

Required CMS pages:

- `about` — contains generic about information
- `terms` — contains terms and conditions

2.4.3 Page Blocks

Page blocks are, *hackish* way to include content in different parts of page.

Page blocks use markdown formatting.

Adding page blocks is very similar to adding CMS Pages:

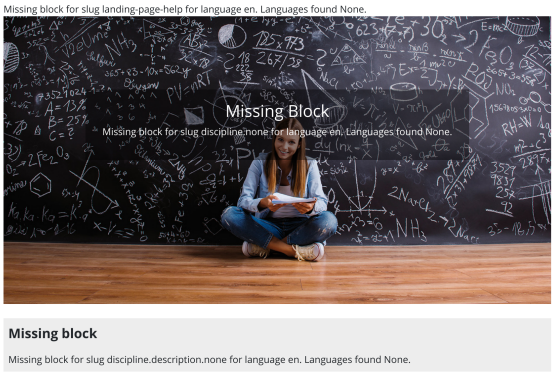
- Go to admin page
- Add new `Page Block`. For a page block to be displayed you need to add it a known slug (known: means one that OLCMS understands — list is below)
- Add new `Page Block Translation` which works pretty much like `Page`

List of page blocks:

- `landing-page-help` / `landing-page-help-nomenu` first paragraph of text visible when user enters landing page. Should explain what is in the menu (and probably how to use the system)
- `discipline.<slug>` / `discipline.description.<slug>` where `<slug>` refers to a `<slug>` of a discipline (*See tagging system for explanation*)

  This adds descriptions for disciplines on discipline landing page, that is: when user enters `discipline/medicine`, they'll see descriptions from `discipline.medicine` and `discipline.description.medicine`.
- `wofkforce.<slug>` — displays workforce information on all content items that belong to a given discipline. That is you'll see `workforce.medicine` when visiting any Content Item.
- `<slug>`, used to display small tooltip near any domain that explains meaning of this discipline.

When you start application in `DEBUG` mode you'll see missing blocks like on below screenshots:



Featured:

2.4.4 Main page links

You can programmatically add links to `about` menu and main page sidebar, by:

- Go to admin page
- Add new `Menu link`
- Add some translations for the link label.

2.5 CSS stylesheets

We use bootstrap 4, so most of the styling is done using this library, all our "project specific" styles are stored in `project.scss` file (which is in slightly better CSS variant).

2.5.1 Per-discipline styling

We use per-discipline styling, where each discipline gets it's own color.

Disciplines and its colors are defined in `$stem_colors` variable:

```
$stem_colors: (
  "medicine": #ea5284,
  "chemistry": #97BF0D,
  "engineering": #009DD1,
  "mathematics": darken(#FFDD00, 10%),
  "astronomy": #A5027D,
  "physics": #E7511E,
  "citizen-science": #9BBDBD
);
```

Which maps discipline slug to discipline color. To add new discipline just add it to above variable in `project.scss`.

2.6 Uploading files how does it work

Uploading a file to a swift server is a little bit more complex than normally.

From javascript you need to do the following:

- Obtain the temporary url to which end-user will be able upload files.
- Upload the file to before mentioned url
- Notify OLCMS about upload.

2.6.1 Rationale

Generally speaking normal django storage backends should work, so you can use FileFields and so on.

However we'd like end-users to upload files directly to SWIFT, which in case of big files could put some strain on the server.

To do this we use Swift cluster using two middlewares:

1. `temp url` middleware. This middleware allows us to generate temporary urls, that allow user to perform a single HTTP verb (GET or PUT). After timeout these urls become invalid.
2. `CORS` middleware, enables Swift to server proper `CORS` headers.

So upload flow is as follows:

1. End user does `POST` on `/api/fileupload` endpoint and gets a temporary url in return.
2. User uploads a file to swift.
3. User sends file path to OLCMS to associate uploaded file with Resource or something else.

For auditory purposes when user obtains a temp url for upload we store user id and uploaded file name in a table.

2.6.2 Python api

Public python API for file upload is in: `www.fileupload.api`, generally method to use is: `register_and_prepare_upload`.

2.6.3 HTTP API

There is a single endpoint `/api/fileupload` to use it user needs to have `filestorage.add_fileupload` permission.

2.6.4 How to handle file updates

Just upload a new file, it will get new path, and update resource field with new file.

2.7 Forum integration

2.7.1 Our deployment practices

We use 'unsupported' (by discourse team) method of deploying Discourse — we want:

- To treat discourse as "more or less" stateless service
- Be able to deploy discourse automatically
- To automatically configure discourse using 7 factor app.

Feel free to use our deployment scripts, feel free also to use official docker image for discourse (and of course, feel free to not use discourse at all). In any case integrating stock discourse service should be relatively straightforward.

2.7.2 Integration points

OLCMS is integrated with Discourse in two points:

- OLCMS serves as login provider for Discourse;

  Discourse SSO is documented here: https://meta.discourse.org/t/official-single-sign-on-for-discourse-sso/13045

  Django provider (developed in this team) is here: https://bitbucket.org/olcms/django-discourse-sso/src/master/

- OLCMS loads comments for every content item;

  Integration is purely in JavaScript: https://meta.discourse.org/t/embedding-discourse-comments-via-javascript/31963

2.8 Language switching

2.8.1 Making this application translatable

Wherever you add a new user visible string mark it for translation,

- In Python code user ugettext function — as described in the documentation
- In HTML templates user {% trans %} tag — as described in the documenation

Instructions for generating translation files linked here.

2.8.2 How to translate this application

1. Generate new translation files;
2. Notice which `*po` files changed;
3. Send them to translators;

   Any translation program that can handle `gettext` files, eg: https://poedit.net/.

4. Update any *po file.

2.8.3 How to add new languages

To add new language add it to the LANGUAGE setting in config/settings/common.py file.

And then generate new translation file. Start shell in the container ./dev.sh bash and then do following:

- cd /app
- ./manage.py makemessages --locale <locale code> — this will extract strings from all applications.
- cd www/templates
- ./manage.py makemessages --locale <locale code> — this will extract strings from templa

And then translate new files.

2.8.4 How does language detection work

Language is detected using standard HTTP headers.

2.9 Permissions

We extensively use django permissions, we use following permissions:

- filestorage.add_fileupload — allows user to attach files to content items; given to all users by default;
- content_item.is_autoverified — when user has this permission his uploads are automatically verified, see *Verification documentation*.
- content_item.can_verify — when user has this permission they can mark other content items as verified/unverified, see: *Verification documentation*.

Also we use Django admin panel for some administrative work, you might want give some of your high-powered users following permissions:

- content_item.change_contentitemverifier — allows admin users to edit verification status from admin page.
- page_block.*_pageblock, page_block.*_pageblocktranslation — allows admin users to edit snippets shared between content items. See *CMS documentation*.
- cms.*_translateablepage, cms.*_pagetranslation, cms.*mediafile*, cms.*mediafile*, cms.*richtext* — allows users to add CMS pages in different languages. See *CMS documentation*
- landing_page.*menulink*, landing_page.*_menulinktranslation — allows you to add links to menu on the side of main page and in the /about/ menu.

2.10 How does indexing work

We use haystack, this is an API that allows one to index Django models using various indexing engines.

Indexing is controlled by SearchIndex instances, we use single index named: ContentIndex.

2.10.1 Indexing in dev environment

In dev environment we use Whoosh index, it is completely inside docker container. Whoosh is a python only library.

To index your data you'll need to call: `./manage.py rebuild_index`.

2.10.2 Indexing in prod environment

First you'll need to set up elastic search index service. Indexing should be done outside of request context, so you'll need to call `./manage.py update_index --age HOURS`, it will update index for ContentItems updated less than `HOURS` ago.

2.11 How to add new fields

2.11.1 How to add field searchable by full text search

Haystack uses document indexing, so you'll need to add this field to `contentitem_text.txt` template. This template will be rendered and indexed.

2.11.2 How to add field searchable as an extra field

If you want to add filtering capabilities, like filter by license, you'll need to add additional field to `ContentIndex`.

2.11.3 How to add ranked field

Ranks are a way of saying: "this field is more important". To add a ranked field you'll need to just add a "rank" attribute to field.

If this field is also a part of "full text search" (or is indexed inside document) you'll need to add some magic inside search form, like here: http://django-haystack.readthedocs.io/en/v2.4.1/boost.html#field-boost

2.12 Tagging system

We have elaborate tagging system for content items.

There are two kinds of `tags` in OLCMS:

- `domains` — this is categorization system that contains predefined cathegories;
- `tags` — user defined tagging system;

2.12.1 Domains

Domains are our predefined tagging system, that is domains are provided by the system, and user can't add new domain.

Domains are stored in the database, so feel free to add new domains from the `admin` interface (feel free also to delete unused ones).

When deleting/adding disciplines some care needs to be taken as they are referenced in *the css system*.

Each domain belong to a domain type, example of domain type is: `Dicipline` or `Education Level`.

Domain Type

Domain type has following properties:

- `name` — Human readable domain name
- **`slug` — unique short name of domain type. Used to reference to domain type** inside OLCMS code.

Domain

Domain has following properties

- `name` — Human readable domain name
- **`slug` — unique short name of domain. Used to reference to domain type** inside OLCMS code.
- `type` — domain type.

Translating domains

Only domain names need to be translated, since they are well integrated into the system it was decided that translations should be stored not in the database, but in django `*po` files along with the rest of internal strings (*see django translations*).

How to add new domains

Best way to add new domains (and or delete domains) is to:

- Add new django migration;
- Generate domains there, see `0002_create_domains.py` migration for a good way to do this.

Adding new disciplines

Add them just like other domains, however please note that right now we have `per-discipline` styling, so whenever you add new discipline you'll need to update `projects.scss` file. See *css documentation for details*.

2.12.2 Tags

End users can add any tags they want to content items.

2.13 ContentItem Verification

2.13.1 How does content verification work

Whenever user creates a `Content Item` it starts in `unverified` state. Admins might switch `Content Item` between verified and unverified states. `Content Items` added by users given special permission are automatically verified (*See permission documentation*).

2.13.2 What changes with unverified content

Unverified content:

- has red “unverified” label;
- is not searchable;

2.13.3 How to verify content

If you have proper permissions you will see big “Update” button near “unverified” marker.

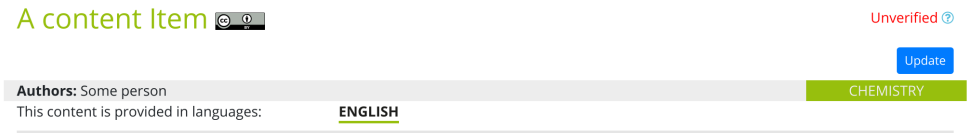


Fig. 1: Update button location

Also if you have `admin` access you can edit verified state on the “Content Item Verifier” list, where you can see verification state for all content items and then update it.

2.14 `www-base` repository

To speed up build times we have created `www-base` repository that contains Docker image with *most* of our python dependencies already installed, so when running tests or building image on CI most of lengthy install process is done.

This is purely a performance enhancement for the CI.

2.15 Obtaining youtube credentials

2.15.1 Introduction

To integrate with youtube you need to obtain Oauth2 credentials.

We use Youtube API in a not typical way (that is we upload videos from OLCMS on behalf of OLCMS Consortium), this upload method is largely undocumented.

So we ended up with with roundabout solution.

2.15.2 What credentials do you need

First obtain a Google Account for your service.

OLCMS uses Youtube Data API v3.

You’ll need to obtain two types of youtube credentials:

- Youtube Oauth2 Key for logging in to Youtube data API v3.
- Oauth2 Refresh token

2.15.3 Obtaining youtube oauth2 key

Follow this google guide, which boils down to:

- Obtain Google Account.
- Go to Google Developer Console to Credentials Page.
- Create "OAuth client ID credentials"
- Download `client_secret.json` file.

2.15.4 Obtain refresh token

Current method

We currently use `video_upload.py` script from youtube-api samples repository, to upload videos to youtube.

This script stores refresh token in a json file generated on first use, so you'll need to:

1. Download our fork of api-samples-repository.
2. Copy client secrets to `python` directory
3. Run `video_upload.py` with dummy file.
4. Perform authentication.
5. Keep `upload-oauth2.json` file.

Desired method

Note: This method is not implemented but I decided to outline it here as I want to move from deprecated google API to something more future-proof, since Google does not document our use case need to document upgrade plan here.

Here I will outline how this **should be done**, if you have some time feel free to implement it.

Changes to OLCMS code

Instead of using old `upload_video` sample either use recent one or just rewrite it in pure python using sample as a sample.

New code doesn't use `Storage` type to store refresh token, so you'll need to create credentials by hand.

To obtain authenticated service you might use following snippet (that was tested to work):

```
# Authorize the request and store authorization credentials.
def get_authenticated_service():
  # flow = InstalledAppFlow.from_client_secrets_file(CLIENT_SECRETS_FILE, SCOPES)
  # credentials = flow.run_console()
  credentials = Credentials.from_authorized_user_info(
    scopes=SCOPES,
    info={
    'refresh_token': u'REFRESH_TOKEN',
    'client_id': u'CLIENT_ID',
    'client_secret': u'CLIENT_SECRET',
```

(continues on next page)

(continued from previous page)

```
    }
  )
  return build(API_SERVICE_NAME, API_VERSION, credentials = credentials)
```

To obtain `client_id` and `client_secret` open `client_secret.json` and read `client_id` and `client_secret` properties there.

Obtaining refresh token

You can follow this tutorial, but here is the gist of it:

1. Go to google oauth playground.
2. Select all youtube scopes.
3. Login.
4. Click "Exchange Authorization Code for Tokens"
5. You should get something that looks like:

```
HTTP/1.1 200 OK
Content-length: 287
(...)
{
  "access_token": "token",
  "token_type": "Bearer",
  "expires_in": 3600,
  "refresh_token": "REFRESH TOKEN abdceffff"
}
```

6. Copy value of *refresh_token*, in your case this will be `REFRESH TOKEN abdceffff`.

2.15.5 Create a playlist and obtain playlist ID

1. Login to youtube, create new playlist.
2. Open said playlist.
3. You should get url that looks like: `https://www.youtube.com/playlist?list=PLrwbUZAqoi2gYedn9wbjqxXsRCMI_HOZb`
4. So your playlistId is value of `list` parameter, in our example it's `PLrwbUZAqoi2gYedn9wbjqxXsRCMI_HOZb`.

2.15.6 Store permissions in environment variables

To store development permissions save them to `.local.dev.env`, in case of deployment see: *deployment docs*.

Store contents of `client_secret.json` as `YOUTUBE_SECRET_JSON` variable and contents of `upload-oauth2.json` as `YOUTUBE_OAUTH2_JSON`. Playlist id should be stored as `YT_PLAYLIST_ID`

2.16 CMS End user documentation

This is documentation for consortium partners

2.16.1 Obtaining permissions

If you cant log-in to the Admin interface at `https://olcms.stem4youth.pl/admin/` ask the OLCMS administrator for permissions.

2.16.2 Adding new CMS page

To add new CMS page:

1. Go to admin interface: `https://olcms.stem4youth.pl/admin/`;
2. Select: `Translatable Pages`
3. Click `Add new Translatable page`
4. Translatable page has only a single property `slug`, which will form the `URL` of the page in our system.

   So when you set slug to `framework` your page will be visible as `https://olcms.stem4youth.pl/framework`.
5. Now you'll want to add some content to the translatable page, to do this go to `Page Translations` and click `Add new Page Translation`.
6. Then fill following fields:
   - `parent` — this is reference to Translatable page;
   - `title` — page title
   - `language` — page language

   Then click combo-box that says: `Insert New Content` and select `Rich Text`.

   When you are done click save.

Fig. 2: Admin options for page translations

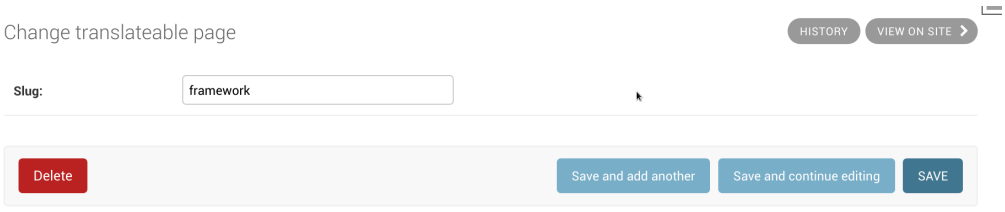


Fig. 3: Adding a new page.

Fig. 4: Adding a new page translation

2.16.3 Add new link to menu

To add new link in the menu you need:

1. Go to admin interface: `https://olcms.stem4youth.pl/admin/`;
2. Select `Menu Link` and then `Add Menu Link`.
3. Set url to which this link will be pointing, eg. you want to put link to `framework` page you were creating, then set url to `/framework/`.

   If you want to link to external resource (which might **not** be a good idea) put fully qualified url here, eg: `http://google.com/?q=why+i+am+doing+this`),
4. Add translation of link label by selecting language and link text.

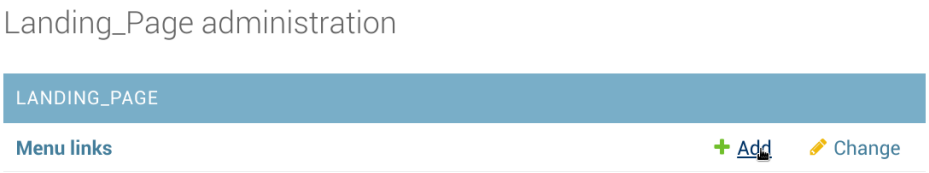


Fig. 5: Menu link location

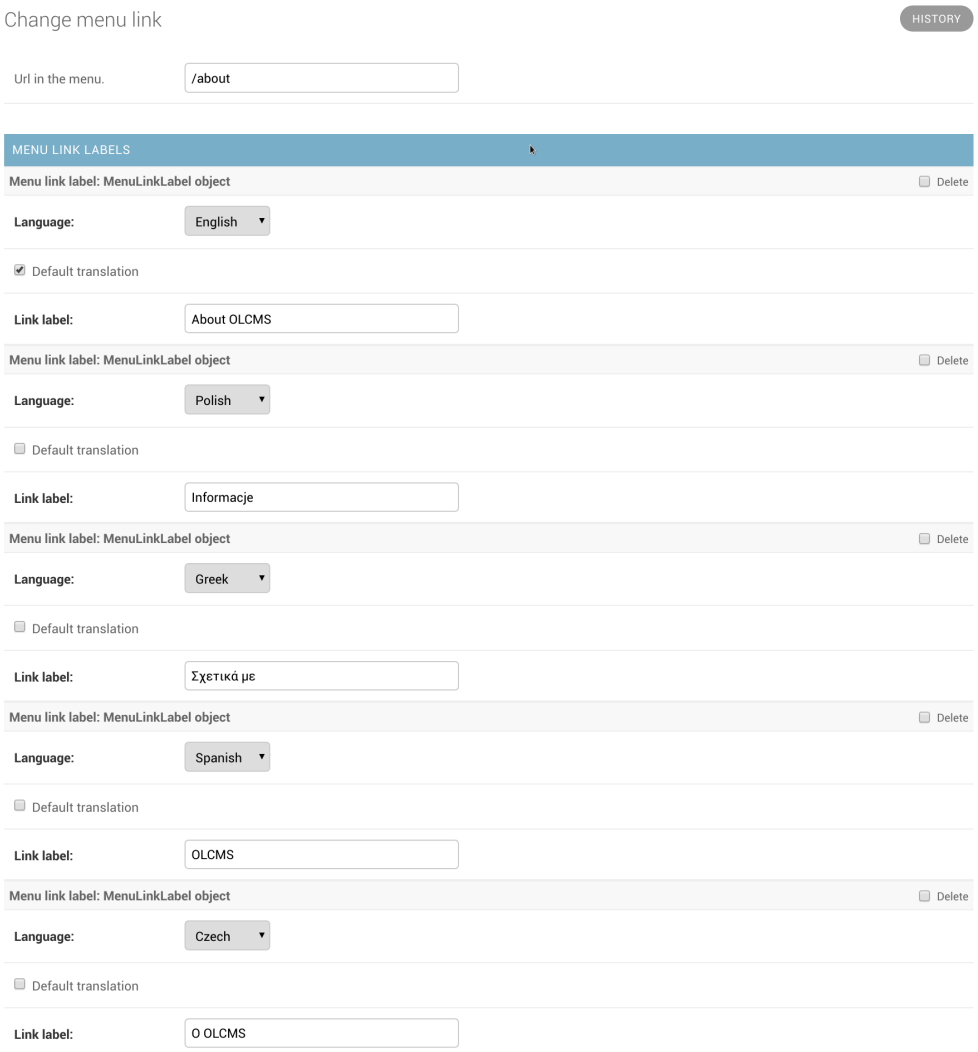


Fig. 6: Fully filled menu item

CHAPTER 3

Indices and tables

- genindex
- modindex
- search