File Repository Documentation

Release 2

Wolnosciowiec Team

Nov 16, 2019

Contents:

File Repository is a modern API application dedicated for storing files. It is able to use various storage backends including AWS S3, Dropbox, Google Drive and just filesystem. Lightweight, requires just PHP7 and at least SQLite3 or MySQL (other databases can be also supported in future due to using ORM).

Main functionality:

- Strict access control, you can **generate a token** that will have access to specific actions on specific items
- Store files where you need; on **AWS S3, Minio.io, FTP, local storage and others...**
- **Deduplication for non-grouped files**. There will be no duplicated files stored on your disk
- **Backups management**, you can define a collection of file versions that can **rotate on adding a new version**
- API + lightweight frontend
- Ready to integrate upload forms for your applications. Only generate token and redirect a user to an url

CHAPTER 1

First steps

To start using the application you need to install PHP 7.3 with extensions listed in *composer.json* file (see entries ext-{name}), composer.

You can also use a ready-to-use docker container instead of using host installation of PHP, **if you have a possibility always use a docker container**.

Summary of application requirements:

- PHP 7.3 or newer
- SQLite3, MySQL 5.7+ or PostgreSQL 10+
- Composer (PHP package manager, see packagist.org)
- make (GNU Make)

Notice: For PostgreSQL configuration please check the configuration reference at *PostgreSQL support* page

CHAPTER 2

Manual installation

At first you need to create your own customized *.env* file with application configuration. You can create it from a template *.env.dist*.

Make sure the **APP_ENV** is set to **prod**, and that the database settings are correct. On default settings the application should be connecting to a SQLite3 database placed in local file, but this is not optimal for production usage.

```
cd server
cp .env.dist .env
edit .env
```

To install the application - download dependencies, install database schema use the make task **install**.

```
make install
```

All right! The application should be ready to go. To check the application you can launch a **development web server**.

```
make run_dev
```

CHAPTER 3

Installation with docker

There are at least three choices:

- Use *quay.io/riotkit/file-repository* container by your own (advanced)
- Generate a docker-compose.yaml using *make print VARIANT="s3 postgres persistent"* in *server/env* directory
- Create your own environment basing on provided example docker-compose

Proposed way to choose is the prepared docker-compose environment that is placed in *server/env* directory.

Starting the example environment:

```
cd ./server/env
make up VARIANT="s3 postgres persistent"
```

Generating a docker-compose example file:

```
cd ./server/env
make print VARIANT="s3 postgres persistent"
```

Production tips:

- Use external database, do backups
- Do not use SQLite3 for production. Use PostgreSQL or MySQL instead.
- Mount data as volumes. Use bind-mounts to have files placed on host filesystem (volumes can be deleted, bind-mounted files stays anyway)

CHAPTER 4

Post-installation

At this point you have the application, but you do not have access to it. **You will need to generate an administrative access token** to be able to create new tokens, manage backups, upload files to storage. To achieve this goal you need to execute a simple command.

Given you use docker you can do eg. **sudo docker exec some-container-name ./bin/console auth:generate-admin-token**, for bare metal installation it would be just **./bin/console auth:generate-admin-token** in the project directory.

So, when you have an administrative token, then you need a token to upload backups. It's not recommended to use administrative token on your servers. **Recommended way is to generate a separate token, that is allowed to upload a backup to specified collection**

To do so, check all available roles in the application:

```
GET /auth/roles?_token=YOUR-ADMIN-TOKEN-HERE
```

Note: If you DO NOT KNOW HOW to perform a request, then please check the postman section

You should see something like this:

```
{
    "roles": {
        "upload.images": "Allows to upload images",
        "upload.documents": "Allows to upload documents",
        "upload.backup": "Allows to submit backups",
        "upload.all": "Allows to upload ALL types of files regardless of mime type",
        "security.authentication_lookup": "User can check information about ANY token
↪",
        "security.overwrite": "User can overwrite files",
        "security.generate_tokens": "User can generate tokens with ANY roles",
        "security.use_technical_endpoints": "User can use technical endpoints to␣
↪manage the application",
        "deletion.all_files_including_protected_and_unprotected": "Delete files that␣
↪do not have a password, and password protected without a password",
        "view.any_file": "Allows to download ANY file, even if a file is password␣
↪protected",
```

(continues on next page)

(continued from previous page)

```
        "view.files_from_all_tags": "List files from ANY tag that was requested, else␣
↪the user can list only files by tags allowed in token",
        "view.can_use_listing_endpoint_at_all": "Define that the user can use the␣
↪listing endpoint (basic usage)",
        "collections.create_new": "Allow person creating a new backup collection",
        "collections.allow_infinite_limits": "Allow creating backup collections that␣
↪have no limits on size and length",
        "collections.modify_any_collection_regardless_if_token_was_allowed_by_
↪collection": "Allow to modify ALL collections. Collection don't have to allow such␣
↪token which has this role",
        "collections.view_all_collections": "Allow to browse any collection␣
↪regardless of if the user token was allowed by it or not",
        "collections.can_use_listing_endpoint": "Can use an endpoint that will allow␣
↪to browse and search collections?",
        "collections.manage_tokens_in_allowed_collections": "Manage tokens in the␣
↪collections where our current token is already added as allowed",
        "collections.upload_to_allowed_collections": "Upload to allowed collections",
        "collections.list_versions_for_allowed_collections": "List versions for␣
↪collections where the token was added as allowed",
        "collections.delete_versions_for_allowed_collections": "Delete versions only␣
↪from collections where the token was added as allowed"
    }
}
```

To allow only uploading and browsing versions for assigned collections you may choose:

```
POST /auth/token/generate?_token=YOUR-ADMIN-TOKEN-THERE
{
    "roles": ["upload.backup", "collections.upload_to_allowed_collections",
↪"collections.list_versions_for_allowed_collections"],
    "data": {
        "tags": [],
        "allowedMimeTypes": [],
        "maxAllowedFileSize": 0
    }
}
```

As the response you should get the token id that you need.

```
{
    "tokenId": "34A77B0D-8E6F-40EF-8E70-C73A3F2B3AF8",
    "expires": null
}
```

Remember the tokenId, now you can create collections and grant access for this token to your collections. Generated token will be able to upload to collections you allow it to.

Check next steps:

1. *Collection creation*
2. *Assigning a token to the collection*

That's all.

CHAPTER 5

Configuration reference

5.1 Application configuration

When setting up application without a docker a .env file needs to be created in the root directory of the application. The .env.dist is a template with example, reference values. If you use a docker image, then you may use those variables as environment variables for the container.

5.2 Permissions list

You can get a permissions list by accessing an endpoint in your application:

```
GET /auth/roles?_token=test-token-full-permissions
```

There is also an always up-to-date permissions list, taken directly from the recent version of the application.

How to read the list by example:

```
/** Allows to upload images */
public const ROLE_UPLOAD_IMAGES            = 'upload.images';
```

Legend:

- Between /** and */ is the description
- **upload.images** is the role name

5.3 Docker container extra parameters

Parameters passed to docker container are mostly application configuration parameters, but not only. There exists extra parameters that are implemented by the docker container itself, they are listed below:

Name and example	Description
WAIT_FOR_HOST=db_mysql:3306	(optional) Waits up to 2 minutes for host to be up when starting a container
SENTRY_DSN=url-here	(optional) Enables integration with sentry.io, so all failures will be logged there

5.4 PostgreSQL support

1. Required extensions: - uuid-ossp (*CREATE EXTENSION "uuid-ossp";*)

2. Due to lack of Unix sockets support in Doctrine Dbal library we created a custom PostgreSQL adapter.

UNIX Socket example:

```
DATABASE_URL=
POSTGRES_DB_PDO_ROLE=... (in most cases same as username)
POSTGRES_DB_PDO_DSN="pgsql:host=/var/run/postgresql;user=...;dbname=...;password=...;"
DATABASE_CHARSET=UTF8
DATABASE_COLLATE=pl_PL.UTF8
DATABASE_DRIVER=pdo_pgsql
DATABASE_VERSION=10.10
```

IPv4 example:

```
DATABASE_URL=
POSTGRES_DB_PDO_ROLE=... (in most cases same as username)
POSTGRES_DB_PDO_DSN="pgsql:host=my_db_host;user=...;dbname=...;password=...;"
DATABASE_CHARSET=UTF8
DATABASE_COLLATE=pl_PL.UTF8
DATABASE_DRIVER=pdo_pgsql
DATABASE_VERSION=10.10
```

CHAPTER 6

Docker, releases and versioning

Images are hosted on both hub.docker.com and quay.io

The versions are created from tags, when a code is considered stable, then it is tagged.

Please see https://semver.org/ for how we version the application.

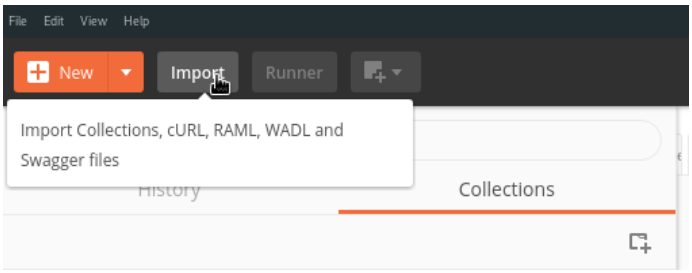
```
# quay.io/riotkit
quay.io/riotkit/file-repository
quay.io/riotkit/bahub
quay.io/riotkit/file-repository-sentry

# https://hub.docker.com/r/wolnosciowiec/file-repository
wolnosciowiec/file-repository
```

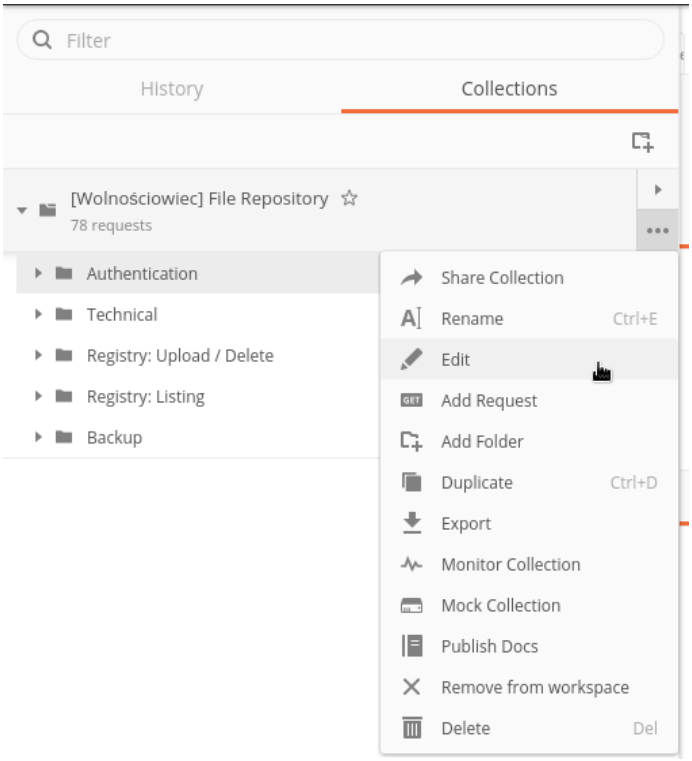
CHAPTER 7

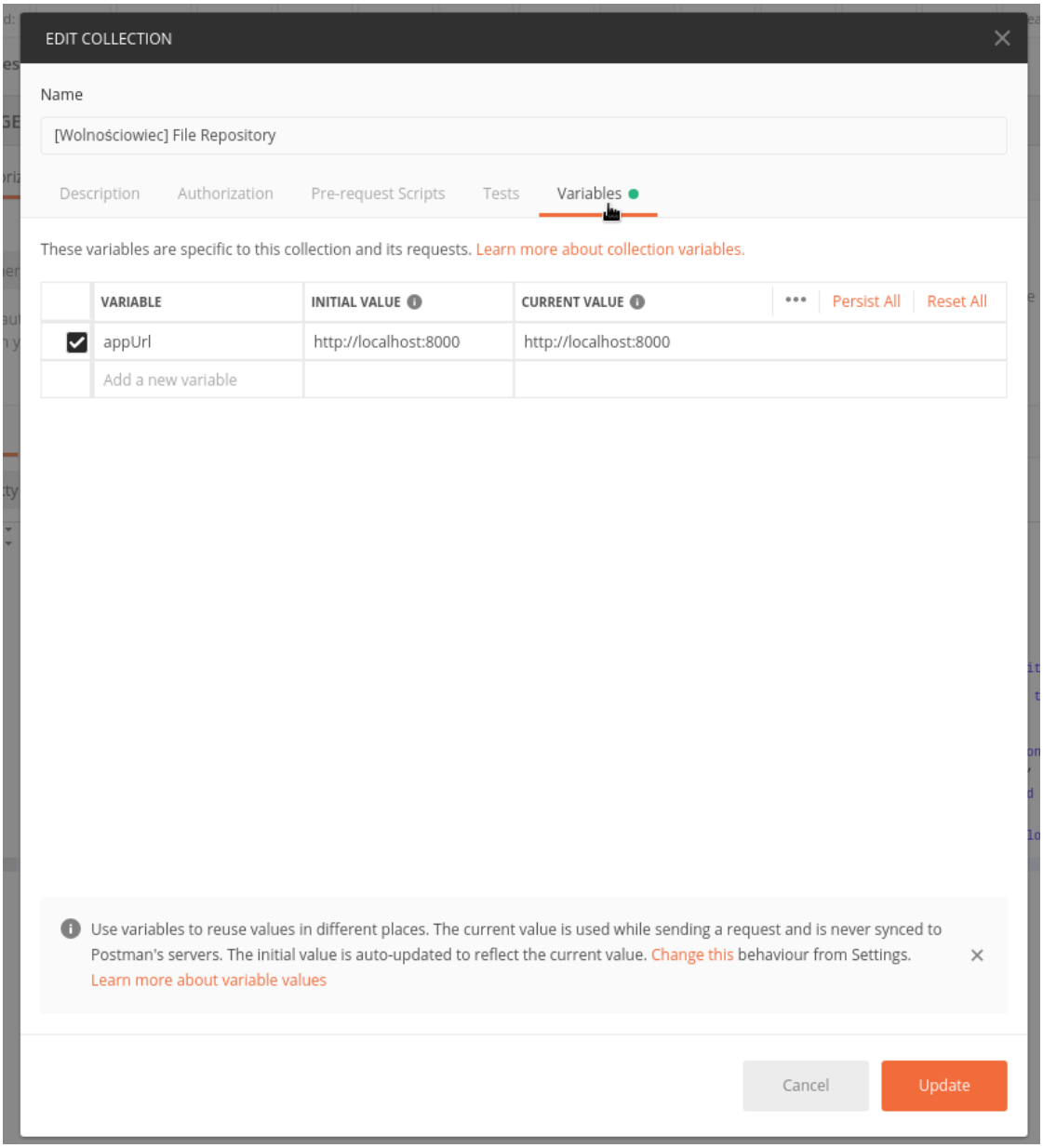
Using postman to manage the application

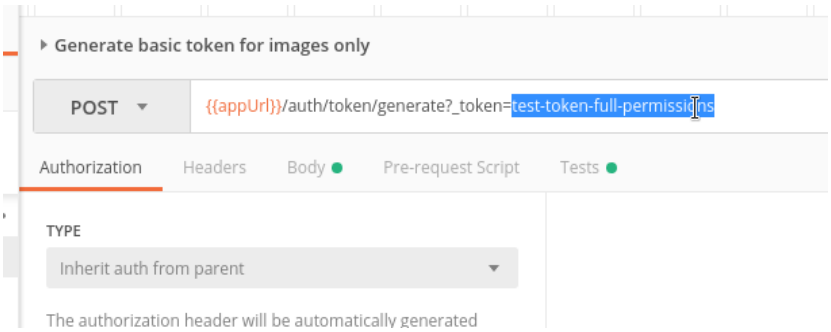
Postman is an API client that allowing to send HTTP requests. You can use it, when you do not have any other graphical application, that could be acting as a client of the File Repository.

At first you can create your own collection, then you can import our test-collection to have some examples.




Filter
History
Collections
[Wolnościowiec] File Repository
78 requests
Authentication
Technical
Registry: Upload / Delete
Registry: Listing
Backup
Share Collection
Rename Ctrl+E
Edit
Add Request
Add Folder
Duplicate Ctrl+D
Export
Monitor Collection
Mock Collection
Publish Docs
Remove from workspace
Delete Del


EDIT COLLECTION
Name
[Wolnościowiec] File Repository
Description
Authorization
Pre-request Scripts
Tests
Variables
These variables are specific to this collection and its requests. Learn more about collection variables.
VARIABLE
INITIAL VALUE
CURRENT VALUE
Persist All
Reset All
appUrl
http://localhost:8000
http://localhost:8000
Add a new variable
Use variables to reuse values in different places. The current value is used while sending a request and is never synced to Postman's servers. The initial value is auto-updated to reflect the current value. Change this behaviour from Settings.
Learn more about variable values
Cancel
Update


Generate basic token for images only
POST
{{appUrl}}/auth/token/generate?_token=test-token-full-permissions
Authorization
Headers
Body
Pre-request Script
Tests
TYPE
Inherit auth from parent
The authorization header will be automatically generated

CHAPTER 8

Authorization

File Repository is an API application, so there is no user account identified by login and password, there are **ACCESS TOKENS**.

An access token is identified by long UUIDv4, and has assigned information about the access, such as:

- List of actions that are allowed (eg. file uploads could be allowed, but browsing the list of files not)
- Allowed tags that could be used when uploading (optional)
- Allowed file types (mime types) when uploading (optional)
- List of allowed IP addresses that could use this token (optional)
- List of allowed User-Agent strings (optional)
- Maximum allowed file size (optional)
- Token expiration date

To authorize in the API you need to provide the token in one of those methods: - Using a query parameter "_token" eg. /some/url?_token=123 - Using a HTTP header "X-Auth-Token"

8.1 Creating a token

Check out the *Permissions list* for a complete list of permissions.

Parameters	
name	description
roles	A list of roles allowed for user. See permissions/configuration reference page
data.tags	List of allowed tags to use in upload endpoints (OPTIONAL)
data.allowedMimeTypes	List of allowed mime types (OPTIONAL)
data.maxAllowedFileSize	Number of bytes of maximum file size (OPTIONAL)
data.allowedUserAgents	List of allowed User-Agent header values (ex. to restrict token to single browser) (OPTIONAL)
data.allowedIpAddresses	List of allowed IP addresses (ex. to restrict one-time-token to single person/session) (OPTIONAL)
expires	Expiration date, or "auto", "automatic", "never". Empty value means same as "auto"

```
POST /auth/token/generate?_token=your-admin-token-there

{
    "roles": ["collections.create_new", "collections.add_tokens_to_allowed_collections
↪"],
    "data": {
        "tags": [],
        "allowedMimeTypes": ["image/jpeg", "image/png", "image/gif"],
        "maxAllowedFileSize": 14579,
        "allowedUserAgents": ["Mozilla/5.0 (X11; Linux x86_64; rv:57.0) Gecko/
↪20100101 Firefox/57.0"],
        "allowedIpAddresses": ["192.168.1.10"]
    },
    "expires": "2020-05-05 08:00:00"
}
```

Example response:

```
{
    "tokenId": "D0D12FFF-DD04-4514-8E5D-D51542DEBCFA",
    "expires": "2020-05-05 08:00:00"
}
```

Required roles:

- security.generate_tokens

8.2 Looking up a token

```
GET /auth/token/D0D12FFF-DD04-4514-8E5D-D51542DEBCFA?_token=your-admin-token-there
```

Example response:

```
{
    "tokenId": "34A77B0D-8E6F-40EF-8E70-C73A3F2B3AF8",
    "expires": "2019-01-06 09:20:16",
    "roles": [
        "upload.images"
    ],
    "tags": [
        "user_uploads.u123",
```

(continues on next page)

(continued from previous page)

```
        "user_uploads"
    ],
    "mimes": [
        "image/jpeg",
        "image/png",
        "image/gif"
    ],
    "max_file_size": 14579
}
```

Required roles:

- security.authentication_lookup

8.3 Revoking a token

```
DELETE /auth/token/D0D12FFF-DD04-4514-8E5D-D51542DEBCFA?_token=your-admin-token-there
```

Example response:

```
{
    "tokenId": "D0D12FFF-DD04-4514-8E5D-D51542DEBCFA",
    "expires": "2019-01-06 09:20:16"
}
```

Required roles:

- security.revoke_tokens

CHAPTER 9

Files storage

The file storage is like a bag of files, there are no directories, it's more like an object storage. When you put some file it is written down on the disk, and it's metadata is stored in the database.

Files could be tagged with some names, it's useful if the repository is shared between multiple usage types. The listing endpoint can search by tag, phrase, mime type - the external application could use listing endpoint to show a gallery of pictures for example, uploaded documents, attachments lists.

In short words the **File Storage** is a specialized group of functionality that allows to manage files, group them, upload new, delete and list them.

9.1 Security

9.1.1 Access

File can be PUBLIC or PRIVATE, the **public** attribute of input data that is sent together with file means the file will not be listed by listing endpoint (unless the token is not an administrative token).

Password protection could be used to protect from downloading the file content by not authorized person, and also it will anonymize the file in public listing if the person who lists the files will not know the password.

9.1.2 Uploading restrictions

When you give user a temporary token to allow to upload eg. avatar, then you may require that the file will not have a **password**, and possibly enforce to select some tags as mandatory.

Extra roles, that can restrict the token	
name	description
upload.enforce_no_password	Enforce files uploaded with this token to not have a password
upload.enforce_tags_selected_in_token	Regardless of tags that user could choose, the tags from token will be copied into each uploaded file

9.2 Uploading

Files could be uploaded in three ways - as RAW BODY, as POST form field and as URL from existing resource in the internet.

Common parameters for all endpoints	
name	description
tags	List of tags where the file will be listed
public	Should be listed/searched? (true/false)
password	Optionally allows to protect access to the file and it's metadata
encoding	Allows to upload encoded file, example values: base64, '' (helpful for frontend implementation)

9.2.1 From external resource by URL

Endpoint specific parameters	
name	description
fileUrl	URL address to the file from the internet

```
POST /repository/image/add-by-url?_token=some-token-there

{
    "fileUrl": "http://zsp.net.pl/files/barroness_logo.png",
    "tags": [],
    "public": true
}
```

9.2.2 In RAW BODY

Endpoint specific parameters	
name	description
filename	Filename that will be used to access the file later

```
POST /repository/file/upload?_token=some-token-here&fileName=heart.png

< some file content there instead of this text >
```

Notes:

- Filename will have added automatically the content hash code to make the record associated with file content (eg. heart.png -> 5Dgds3dqheart.png)
- Filename is unique, same as file
- If file already exists under other name, then it's name will be returned (deduplication mechanism)

9.2.3 In a POST form field

Endpoint specific parameters	
name	description
filename	Filename that will be used to access the file later

```
POST /repository/file/upload?_token=some-token-here&fileName=heart.png

Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="file"; filename=""
Content-Type: image/png


------WebKitFormBoundary7MA4YWxkTrZu0gW--

... file content some where ...
```

9.3 Downloading

When you upload your file you will always get an URL address in the JSON response, but the download endpoints has more to offer than it looks on first view. Let's explain additional things you can do with the download endpoint.

Features:

- Bytes range support, files could be downloaded partially, videos can be rewinded while streamed
- Big files support
- Content type is sent, so the browser knows the file size and can show the progress bar
- Optional password protection

Common parameters for all endpoints	
name	description
password	Password to access the file, optionally if the file is password protected

9.3.1 Regular downloading

It's very simple.

```
GET /repository/file/d3beb8a9f0some-file-name-there.txt?password=optional-password-
↪there-if-any
```

9.3.2 Downloading using alias defined in ids_mapping.yaml

Aliases are allowing to access files by other names, they can be defined in *./config/ids_mapping.yaml* file. It's very helpful feature when you migrate from other storage application to File Repository.

Example ids_mapping.yaml file:

```
"oh-my-alias-there": "d3beb8a9f0some-file-name-there.txt"
```

Example request:

```
GET /repository/file/oh-my-alias-there
```

9.3.3 Downloading using hotlink protection

Hotlink protection is allowing to generate personalized download urls by combining eg. user's IP address, some salt, file name and timestamp. Such link cannot be shared with other users.

Note: Hotlink protection endpoint also supports aliasing.

Example request:

```
GET /stream/531ce1f1d5d242cd5005b3758d3b5435/2219788800/d3beb8a9f0some-file-name-
↪there.txt
```

The format of the URL is defined in the environment variables:

```
ANTI_HOTLINK_PROTECTION_ENABLED=true
ANTI_HOTLINK_RESTRICT_REGULAR_URLS=false
ANTI_HOTLINK_URL=/stream/{accessToken}/{expirationTime}/{fileId}
ANTI_HOTLINK_CRYPTO=md5
ANTI_HOTLINK_SECRET_METHOD="\$http_x_expiration_time\$http_test_header MY-AWESOME-
↪SUFFIX"
```

It means you can change it, so the URL will be different. {expirationTime} is optional, but very helpful.

Short explanation:

> The {accessToken} is generated by hashing with eg. md5 the filled-up ANTI_HOTLINK_SECRET_METHOD.
>
> Example: Given ANTI_HOTLINK_SECRET_METHOD is "$http_x_expiration_time$http_test_header MY-AWESOME-SUFFIX" We send a request with {expirationTime} = 123 and a header Test-Header = HELLO
>
> So, the secret would be "123HELLO MY-AWESOME-SUFFIX", now we have to hash using selected crypto - md5. md5(23HELLO MY-AWESOME-SUFFIX) = 531ce1f1d5d242cd5005b3758d3b5435
>
> It means that we have URL: /stream/531ce1f1d5d242cd5005b3758d3b5435/123/d3beb8a9f0some-file-name-there.txt

9.4 Aliasing filenames (migrating existing files to File Repository)

Filename in **File Repository** is created based on file contents hash + name submitted by user. To allow **easier migration of your existing files**, the **File Repository** allows to create *aliases* to files you upload.

9.4.1 Scenario

Let's assume that you have a file named "Accidential-Anarchist.mp4", and your website shows a player that points to https://static.iwa-ait.org/Accidential-Anarchist.mp4 Now you want to migrate your storage to use **File Repository**, so the **File Repository** will store and serve the files with help of your webserver.

To keep old links still working you need to:

- Set up a URL rewrite in your webserver (eg. NGINX or Apache 2) to rewrite the FORMAT OF THE URL, example: /education/movies/watch?v=... to /repository/file/...
- You have a file "Accidential-Anarchist.mp4", after uploading to File Repository it will have different name eg. "59dce00bcAccidential-Anarchist.mp4", you can create an alias that will point from "Accidential-Anarchist.mp4" to "59dce00bcAccidential-Anarchist.mp4"

9.4.2 Practice, defining aliases

To start you need to create a file *config/ids_mapping.yaml*, where you will list all of the aliases in YAML syntax.

Example:

Notice: You need to restart the application (or execute ./bin/console cache:clear –env=prod) after applying changes to this file

9.5 Hotlink protection - personalizing URLs for your visitors

If for any reason you need to secure your content from being distributed outside of your website, then you need a hotlink protection. Hotlink protection gives your website a **control over who can see the video, image or any other resource that is kept on File Repository**.

9.5.1 Preparing your website and File Repository configuration

A website that is displaying eg. a video player that would play a video from **File Repository** need to point to a personalized URL address especially generated for your page visitor.

At first let's look at the URL format, you need to define a URL format that will point to protected files. Below there are multiple examples, you can configure the URL however you want, **this you need to adjust in your .env file or in environment variables in Docker container**.

```
# example 1
ANTI_HOTLINK_URL=/stream/{accessToken}/{expirationTime}/{fileId}

# example 2
ANTI_HOTLINK_URL=/video/{accessToken}/{expirationTime}/{fileId}

# example 3
ANTI_HOTLINK_URL=/watch/{fileId},{accessToken},{expirationTime}

# example 4
ANTI_HOTLINK_URL=/watch/{accessToken}/{fileId}
```

So, let's take a look at the most interesting part - the access token generation.

Each visitor on your page needs to get a unique access token that will allow to see the file content **only for him/her**. To generate such access token we need to **DEFINE A COMMON FORMAT that your application will use and File Repository will understand**.

```
ANTI_HOTLINK_SECRET_METHOD="\$http_x_expiration_time\$http_x_real_uri\$http_x_remote_
↪addr MY-AWESOME-SUFFIX"
```

Following example is combining most important variables, why?

- $http_x_real_uri - to restrict this token only to single file (this header may be required to be set on NGINX/Apache level)
- $http_x_remote_addr - to restrict access to single IP address
- MY-AWESOME-SUFFIX - this one definitely you should change to a SECRET you only know. It will prevent anybody from generating a token
- $http_x_expiration_time - optionally validate the passed input data in the url

Generally the rule with the variables is simple as in NGINX, but a little bit more extended to give better possibilities.

Variable templates	
name	description
$http_xxx	In place of xxx put your normalized header name eg. Content-Type would be content_type
$server_xxx	Everything what is in PHP's $_SERVER, including environment variables
$query_xxx	Everything what is in query string (query string in URL is everything after question mark)

9.5.2 Practical example of generating access token on your website

Assuming that you have following configuration:

```
ANTI_HOTLINK_PROTECTION_ENABLED=true
ANTI_HOTLINK_RESTRICT_REGULAR_URLS=false
ANTI_HOTLINK_CRYPTO=md5
ANTI_HOTLINK_SECRET_METHOD="\$http_x_expiration_time\$filename\$http_remote_addr MY-
↪AWESOME-SUFFIX"
ANTI_HOTLINK_URL=/stream/{accessToken}/{expirationTime}/{fileId}
```

That would be an example code that could generate URL addresses in your application:

```
<?php
$fileId = 'Accidential-Anarchist.mp4';
$expirationTime = time() + (3600 * 4); // +4 hours
$rawToken = $expirationTime . $fileId . ($_SERVER['REMOTE_ADDR'] ?? '') . ' MY-
↪AWESOME-SUFFIX';

$hash = hash('md5', $rawToken);
echo 'URL: /stream/' . $hash . '/' . $expirationTime . '/' . $fileId;
```

9.6 Listing and searching

Each file can be found by using a search endpoint. Password protected files are censored, if the correct password was not entered in the search field.

Note: Files can be named and tagged, marked as public/private, password protected.

Parameters	
name	description
page	Page number
limit	Limit results on single page
password	Password for password-protected files
searchQuery	Search phrase, a word, multiple words to be searched for in the file name
tags	List of tags to filter by (array)
mimes	List of mimes to filter by (array)

Example request:

```
GET /repository?_token=your-auth-token&page=1&limit=20
```

CHAPTER 10

Backup

Backup *collections* allows to store multiple *versions* of the same file.

Each submitted *version* has **automatically incremented version number by one**.

Example scenario with strategy "delete_oldest_when_adding_new":

```
Given we have DATABASE dumps of iwa-ait.org website
And our backup collection can contain only 3 versions (maximum)

When we upload a sql dump file THEN IT'S a v1 version
When we upload a next sql dump file THEN IT'S a v2 version
When we upload a next sql dump file THEN IT'S a v3 version

Then we have v1, v2, v3

When we upload a sql dump file THEN IT'S a v4 version
But v1 gets deleted because collection is full

Then we have v2, v3, v4
```

From security point of view there is a possibility to attach multiple tokens with different access rights to view and/or manage the collection.

10.1 Getting started

The workflow is following:

1. You need to have an access token that allows you to create collections
2. **Create a collection**, remember it's ID (we will call it **collection_id** later)
3. (Optional) Allow some other token or tokens to access the collection (all actions or only some selected actions on the collection)
4. **Store backups** under a collection of given **collection_id**

5. List and download stored backups when you need

10.1.1 Versioning

Each uploaded version is added as last and have a version number incremented by one, and a **ID** string generated.

For example: There is a **v1** version, **we upload a new version and a new version is getting a number v2**

Later any version could be accessed by generated **ID** string or **version number** (in combination with the **collection ID**)

10.1.2 Collection limits

Each collection could either be a infinite collection or a finite collection.

Below are listed limits for finite collections:

Limits	
limit	description
maxBackupsCount	Maximum count of versions that could be stored
maxOneVersionSize	Maximum disk space that could be allocated for single version
maxCollectionSize	Maximum disk space for whole collection (summary of all files)

10.1.3 Permissions

There could be multiple tokens with different permissions assigned to the collection.

Example use case: Generated **"Guest token"** with download-only permissions could be safe to share between administrators. The **"Upload token"** could be used by the server to automatically upload new versions without permissions to delete other versions and without need to modify collections limits. **"Management token"** with all of the permissions for managing a collection.

10.2 Managing collections

To start creating backups you need a collection that will handle ONE FILE. The file may be a zipped directory, a text file, SQL dump or anything you need.

10.2.1 Collection creation

To add any backup you need a **collection** at first. Collection is a container that keeps multiple versions of same file (for example your database dump from each day). Collection additionally can define limits on length, size, type of uploaded file, and tokens which have access to it at all.

Example request:

```
POST {{appUrl}}/repository/collection?_token=test-token-full-permissions

{
    "maxBackupsCount": 5,
    "maxOneVersionSize": 0,
```

(continues on next page)

(continued from previous page)

```
    "maxCollectionSize": "250MB",
    "strategy": "delete_oldest_when_adding_new",
    "description": "iwa-ait.org database backup",
    "filename": "iwa-ait-org.sql.gz"
}
```

In the response you will receive a collection **ID** that will be required for editing collection information, assigning tokens and uploading files.

There are two strategies. **delete_oldest_when_adding_new** is automatically deleting older backup versions when a `maxBackupsCount` is reached and a new backup is submitted. **alert_when_backup_limit_reached** will raise an error when submitting a new version to already full backup collection.

Notes:

- Put zero values to disable the limit
- *Supports "simulate=true" parameter that allows to send a request that will not create any data, but only validate submitted data*
- **You'r token will be automatically added as token allowed to access and modify the collection**

Required permissions:

- collections.create_new

Optional permissions:

- collections.allow_infinite_limits (allows to create an infinite collection, it means that you can eg. upload as much files as you like to, and/or the disk space is unlimited)

10.2.2 Collection editing

```
PUT {{appUrl}}/repository/collection?_token=test-token-full-permissions

{
    "collection": "SOME-COLLECTION-ID-YOU-RECEIVED-WHEN-CREATING-THE-COLLECTION",
    "maxBackupsCount": 5,
    "maxOneVersionSize": 0,
    "maxCollectionSize": "250MB",
    "strategy": "delete_oldest_when_adding_new",
    "description": "iwa-ait.org database backup (modified)",
    "filename": "iwa-ait-org.sql.gz"
}
```

Notes:

- The collection size cannot be lower than it is actual in the storage (sum of existing files in the collection)
- You need to have global permissions for managing any collection or to **have token listed as allowed in collection you want to edit**

Required permissions:

- collections.modify_details_of_allowed_collections

Optional permissions:

- collections.allow_infinite_limits (allows to edit an infinite collection, it means that you can eg. upload as much files as you like to, and/or the disk space is unlimited)

- collections.modify_any_collection_regardless_if_token_was_allowed_by_collection (gives a possibility to edit a collection even if token is not attached to it)

10.2.3 Deleting

To delete a collection you need to at first make sure, that there are no backup versions attached to it. Before deleting a collection you need to manually delete all backups. It's for safety reasons.

```
DELETE {{appUrl}}/repository/collection/SOME-COLLECTION-ID?_token=test-token-full-
↪permissions
```

Required permissions:

- collections.delete_allowed_collections

Optional permissions:

- collections.modify_any_collection_regardless_if_token_was_allowed_by_collection (gives a possibility to edit a collection even if token is not attached to it)

10.2.4 Fetching collection information

You can fetch information about collection limits, strategy, description and more to be able to edit it using other endpoints.

```
GET {{appUrl}}/repository/collection/SOME-COLLECTION-ID?_token=test-token-full-
↪permissions
```

Notes:

- You need to have global permissions for managing any collection or to **have token listed as allowed in collection you want to fetch**

Required permissions:

- (just the token added as allowed for given collection)

Optional permissions:

- collections.modify_any_collection_regardless_if_token_was_allowed_by_collection (gives a possibility to edit a collection even if token is not attached to it)

10.3 Authorization

Multiple tokens with different permissions could be assigned to the single collection. You may create a token for uploading backups, deleting backups and for managing collection limits separately.

10.3.1 Assigning a token to the collection

```
POST /repository/collection/{{collection_id}}/token?_token={{collection_management_
↪token}}

{
```

(continues on next page)

(continued from previous page)

```
    "token": "SO-ME-TO-KEN-TO-ADD"
}
```

Legend:

- {{collection_management_token}} is your token that has access rights to fully manage collection
- {{collection_id}} is an identifier that you will receive on collection creation (see collection creation endpoint)

Required permissions:

- collections.manage_tokens_in_allowed_collections

10.3.2 Revoking access to the collection for given token

```
DELETE /repository/collection/{{collection_id}}/token/{{token_id}}?_token={
↪{collection_management_token}}
```

Legend:

- {{token_id}} identifier of a token that we want to disallow access to the collection
- {{collection_management_token}} is your token that has access rights to fully manage collection
- {{collection_id}} is an identifier that you will receive on collection creation (see collection creation endpoint)

Required permissions:

- collections.manage_tokens_in_allowed_collections

10.4 Backups: Upload, deletion and versioning

Assuming that you have already a collection and an access token, then we can start uploading files that will be versioned and stored under selected collection.

10.4.1 Uploading a new version to the collection

You need to submit **file content in the HTTP request body**. The rest of the parameters such as token you need to pass as GET parameters.

```
POST /repository/collection/{{collection_id}}/backup?_token={{token_that_allows_to_
↪upload_to_allowed_collections}}

.... FILE CONTENT THERE ....
```

Pretty simple, huh? As the result you will get the version number and the filename, something like this:

```
{
    "status": "OK",
    "error_code": null,
    "exit_code": 200,
    "field": null,
    "errors": null,
    "version": {
```

(continues on next page)

(continued from previous page)


```
        "id": "69283AC3-559C-43FE-BFCC-ECB932BD57ED",
        "version": 1,
        "creation_date": {
            "date": "2019-01-03 11:40:14.669724",
            "timezone_type": 3,
            "timezone": "UTC"
        },
        "file": {
            "id": 175,
            "filename": "ef61338f0dsolidarity-with-postal-workers-article-v1"
        }
    },
    "collection": {
        "id": "430F66C3-E4D9-46AA-9E58-D97B2788BEF7",
        "max_backups_count": 2,
        "max_one_backup_version_size": 1000000,
        "max_collection_size": 5000000,
        "created_at": {
            "date": "2019-01-03 11:40:11.000000",
            "timezone_type": 3,
            "timezone": "UTC"
        },
        "strategy": "delete_oldest_when_adding_new",
        "description": "Title: Solidarity with Postal Workers, Against State␣
↪Repression!",
        "filename": "solidarity-with-postal-workers-article"
    }
}
```


Required permissions:

- collections.upload_to_allowed_collections

10.4.2 Deleting a version

A simple DELETE type request will delete a version from collection and from storage.

```
DELETE /repository/collection/{{collection_id}}/backup/BACKUP-ID?_token={{token}}
```

Example response:


```
{
    "status": "OK, object deleted",
    "error_code": 200,
    "exit_code": 200
}
```


Parameters		
type	name	description
bool	simulate	Simulate the request, do not delete in real. Could be used as pre-validation
string	_token	Standard access token parameter (optional, header can be used instead)

Required permissions:

- collections.delete_versions_for_allowed_collections

10.4.3 Getting the list of uploaded versions

To list all existing backups under a collection you need just a collection id, and the permissions.

```
GET /repository/collection/{{collection_id}}/backup?_token={{token}}
```

Example response:


```
{
    "status": "OK",
    "error_code": null,
    "exit_code": 200,
    "versions": {
        "3": {
            "details": {
                "id": "A9DAB651-3A6F-440D-8C6D-477F1F796F13",
                "version": 3,
                "creation_date": {
                    "date": "2019-01-03 11:40:24.000000",
                    "timezone_type": 3,
                    "timezone": "UTC"
                },
                "file": {
                    "id": 178,
                    "filename": "343b39f56csolidarity-with-postal-workers-article-v3"
                }
            },
            "url": "https://my-anarchist-initiative/public/download/
↪343b39f56csolidarity-with-postal-workers-article-v3"
        },
        "4": {
            "details": {
                "id": "95F12DAD-3F03-49B0-BAEA-C5AC3E8E2A30",
                "version": 4,
                "creation_date": {
                    "date": "2019-01-03 11:47:34.000000",
                    "timezone_type": 3,
                    "timezone": "UTC"
                },
                "file": {
                    "id": 179,
                    "filename": "41ea3dcca9solidarity-with-postal-workers-article-v4"
                }
            },
            "url": "https://my-anarchist-initiative/public/download/
↪41ea3dcca9solidarity-with-postal-workers-article-v4"
        }
    }
}
```


Required permissions:

- collections.list_versions_for_allowed_collections

10.4.4 Downloading uploaded versions

Given we upload eg. 53 versions of a SQL dump, one each month and we want to download latest version, then we need to call the fetch endpoint with the **"latest"** keyword as the identifier.

```
GET /repository/collection/{{collection_id}}/backup/latest?password={{collection_
↪password_to_access_file}}&_token={{token}}
```

If there is a need to download an older version of the file, a **version number should be used, eg. v49**

```
GET /repository/collection/{{collection_id}}/backup/v49?password={{collection_
↪password_to_access_file}}&_token={{token}}
```

There is also a possibility to download a last copy from the bottom, the oldest version available using keyword **first**.

```
GET /repository/collection/{{collection_id}}/backup/first?password={{collection_
↪password_to_access_file}}&_token={{token}}
```

In case we have an **ID of the version**, then it could be inserted directly replacing the alias keyword.

```
GET /repository/collection/{{collection_id}}/backup/69283AC3-559C-43FE-BFCC-
↪ECB932BD57ED?password=thats-a-secret&_token={{token}}
```

Parameters		
type	name	description
bool	redirect	Allows to disable HTTP redirection and return JSON with the url address instead
string	password	Password required for requested FILE (please read about passwords in notes section)
string	_token	Standard access token parameter (optional, header can be used instead)

Required permissions:

- collections.list_versions_for_allowed_collections
- (knowing the password for the collection file)

Notes:

- *The password for the file is inherited from collection, but it may be different in case when the collection would have changed the password, old files would not be updated!*

10.5 Data replication

File Repository does not support replication itself. The replication could be enabled on storage backend level.

You may want to check Minio.io that has a possibility to configure multiple nodes in primary-replica model.

10.6 Managing collections from shell

To allow automating things there are shell commands, those do not require authorization and have the same parameters as API endpoints.

10.6.1 Creating collections

```
→  file-repository git:(master) ✗ ./bin/console backup:create-collection --help
Description:
  Creates a backup collection, where the versions could be uploaded

Usage:
  backup:create-collection [options]

Options:
  -b, --max-backups-count=MAX-BACKUPS-COUNT
  -o, --max-one-version-size=MAX-ONE-VERSION-SIZE
  -c, --max-collection-size=MAX-COLLECTION-SIZE
  -s, --strategy[=STRATEGY]                        Backup rotation strategy (available: delete_oldest_when_adding_new, alert_when_backup_limit_reached) [default: "delete_oldest_when_adding_new"]
  -d, --description=DESCRIPTION
  -p, --password[=PASSWORD]
  -f, --filename=FILENAME
  -h, --help                                       Display this help message
  -q, --quiet                                      Do not output any message
  -V, --version                                    Display this application version
      --ansi                                       Force ANSI output
      --no-ansi                                    Disable ANSI output
  -n, --no-interaction                             Do not ask any interactive question
  -e, --env=ENV                                    The Environment name. [default: "dev"]
      --no-debug                                   Switches off debug mode.
  -v|vv|vvv, --verbose                             Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Help:
  Specify all limits and parameters to create a collection
```

The command will return just a collection id on success. On failure a json is returned.

Example success output:

```
 ./bin/console backup:create-collection -d "Some test collection" -f "backup.tar.gz" -
↪b 4 -o 3GB -c 15GB
48449389-E267-497E-A6F4-EAC91C063708
```

Example failure output:

```
 ./bin/console backup:create-collection -d "Some test collection" -f "backup.tar.gz" -
↪b 4 -o 3GB -c 1GB
{
    "status": "Logic validation error",
    "error_code": 4003,
    "http_code": 400,
    "errors": {
        "maxCollectionSize": "max_collection_size_is_lower_than_single_element_size"
    },
    "collection": null,
    "context": []
}
```

CHAPTER 11

MinimumUI

Although that **File Repository** is an API project, it has a few HTML endpoints which are allowing to upload files. MinimumUI idea is to allow to use *File Repository* as a fully standalone microservice, with easy to use embeddable upload forms on any website.

11.1 Quick start in steps

1. Your application needs to have a possibility to *create tokens* in **File Repository** on backend side (no one should see your administrative token).
2. For each user you need to generate a temporary token with minimal permissions (eg. upload only, with restrictions for password, mime types, tags etc.)
3. On your website you need to redirect user to the file repository upload form (MinimumUI endpoint) with specifying the "back" parameter in query string, so the user will go back on your website again and pass the uploaded file URL
4. You need to validate the URL from your user, if it comes eg. from proper domain where File Repository runs

11.2 Endpoints

Following endpoints are just displaying a static HTML page, that acts as a client to the API. No any endpoint is implementing any additional access rights, if the user does not have access to perform some action, then the page would display, but the backend will respond with an error.

If you need to restrict the file size, mime type, allowed tags or others, then you need to specify it in the access token that will be used in the UI.

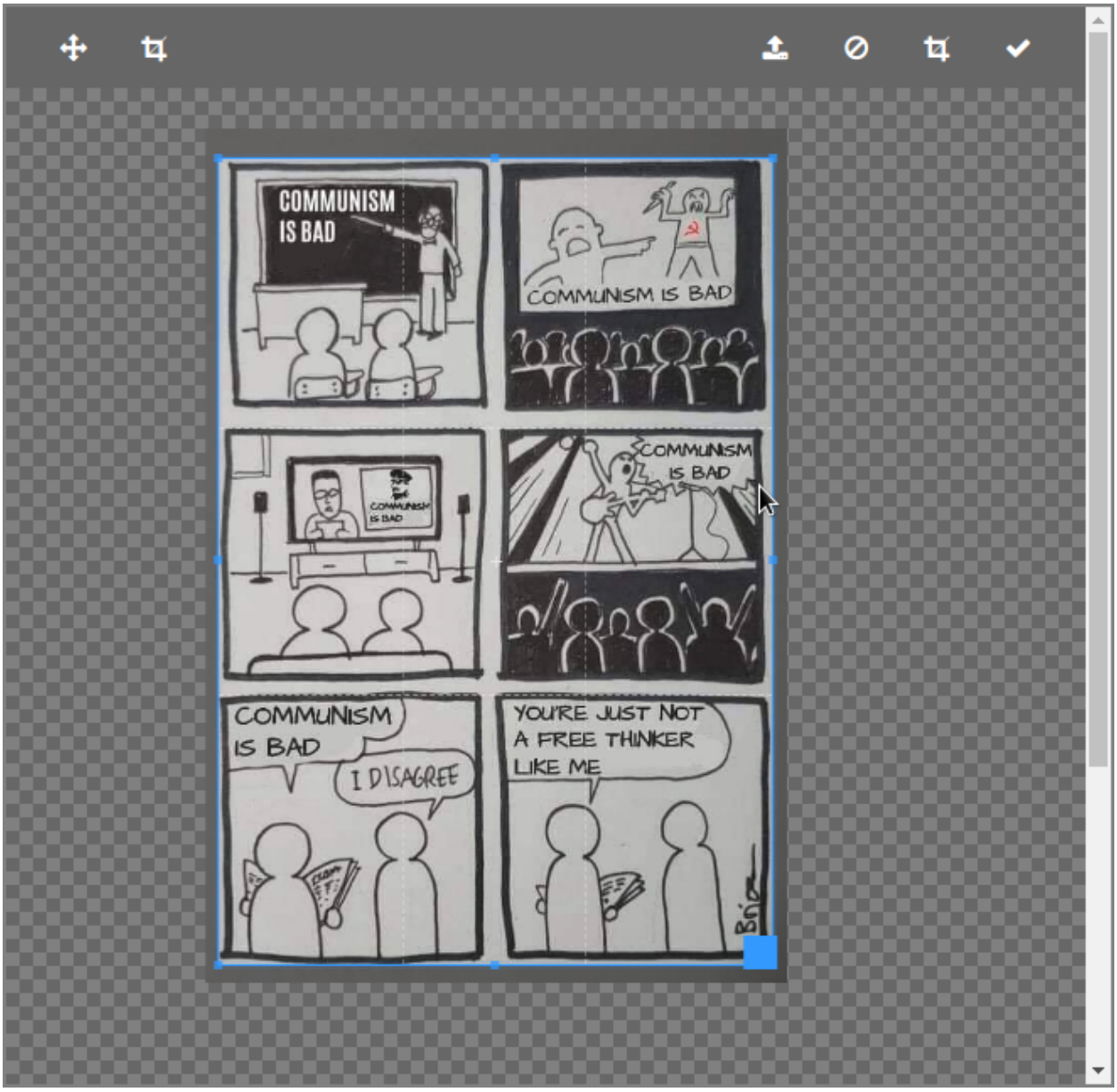
Roles used by the endpoints	
name	description
upload.enforce_no_password	Enforce the file to be uploaded without a password
upload.enforce_tags_selected_in_token	Tag uploaded file with tags specified in the token, regardless of user choice
upload.images	Upload images

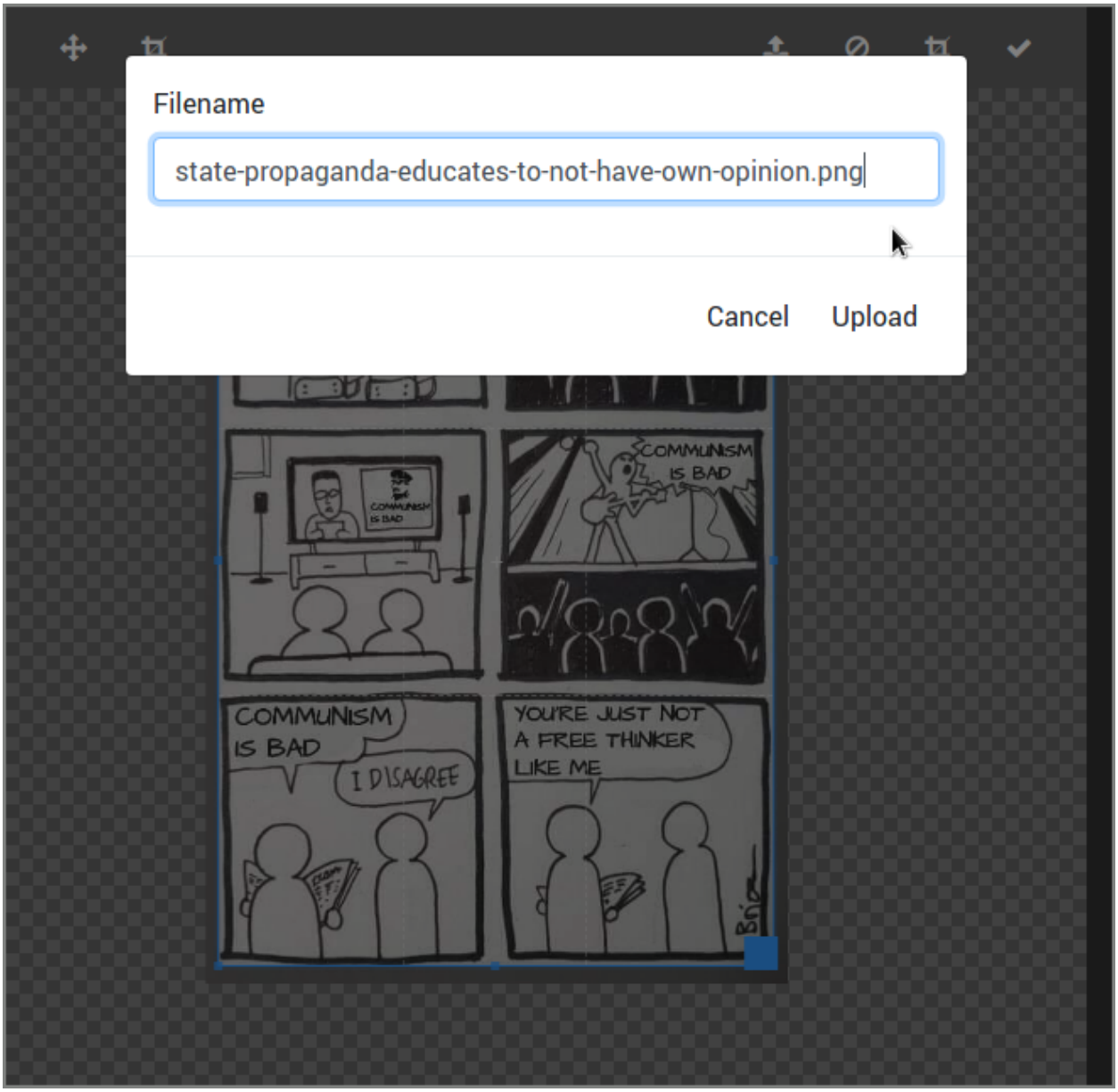
11.2.1 Image Upload

The image upload endpoint allows to upload whole file as is, or with cropping it. Cropper supports an aspect ratio, that could be specified in the query string.

Extra parameters in **query string**	
name	description
ratio	Aspect ratio for the images eg. 16/9 is 1.77, so it would be ?ratio=1.77
back	URL address to redirect the user on success. FILE_REPOSITORY_URL phrase will be replaced with the uploaded file URL
_token	Access token

```
In the browser access URL: /minimum.ui/upload/image?_token=TOKEN-THERE
```

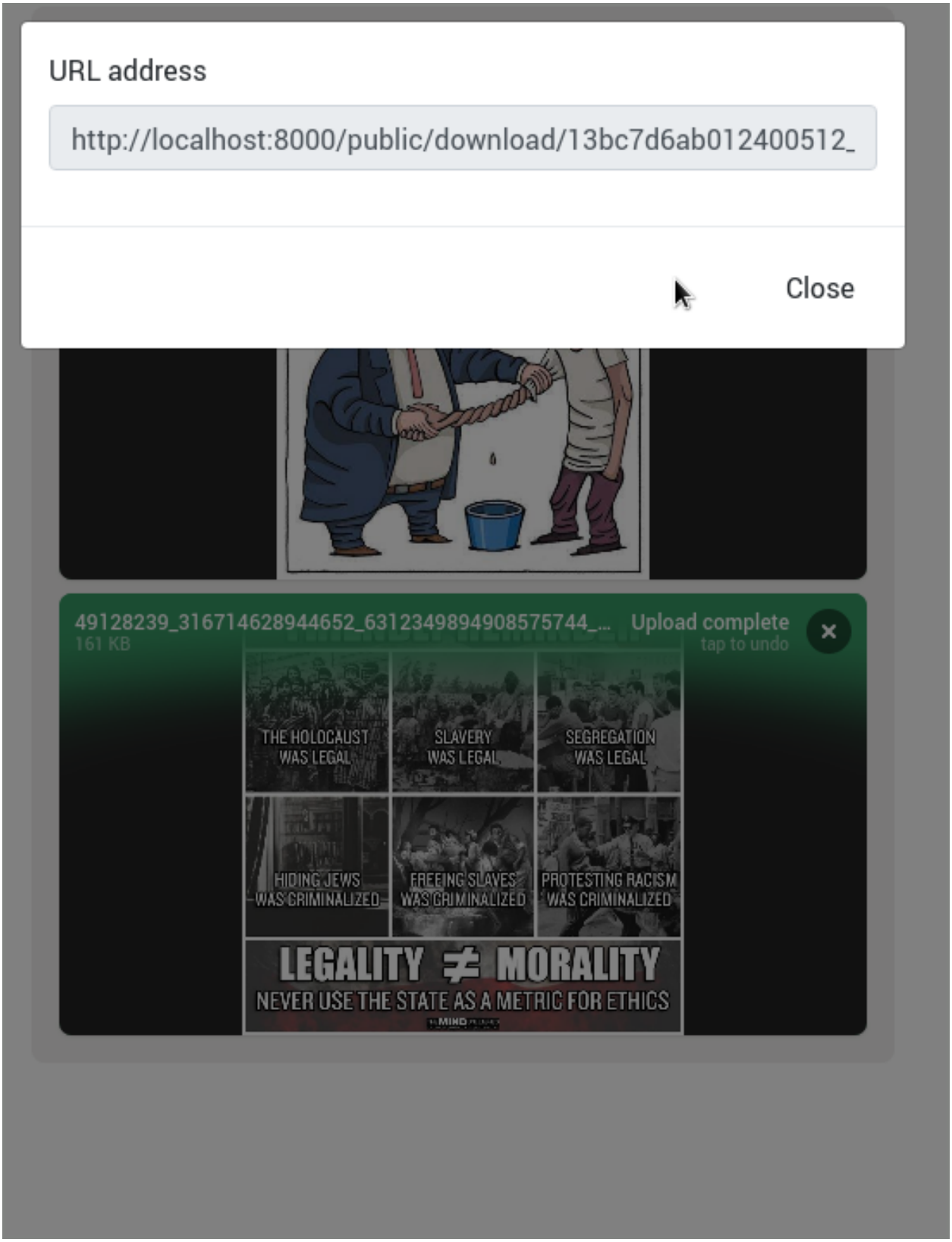

COMMUNISM
IS BAD
COMMUNISM IS BAD
COMMUNISM
IS BAD
COMMUNISM
IS BAD
I DISAGREE
YOU'RE JUST NOT
A FREE THINKER
LIKE ME

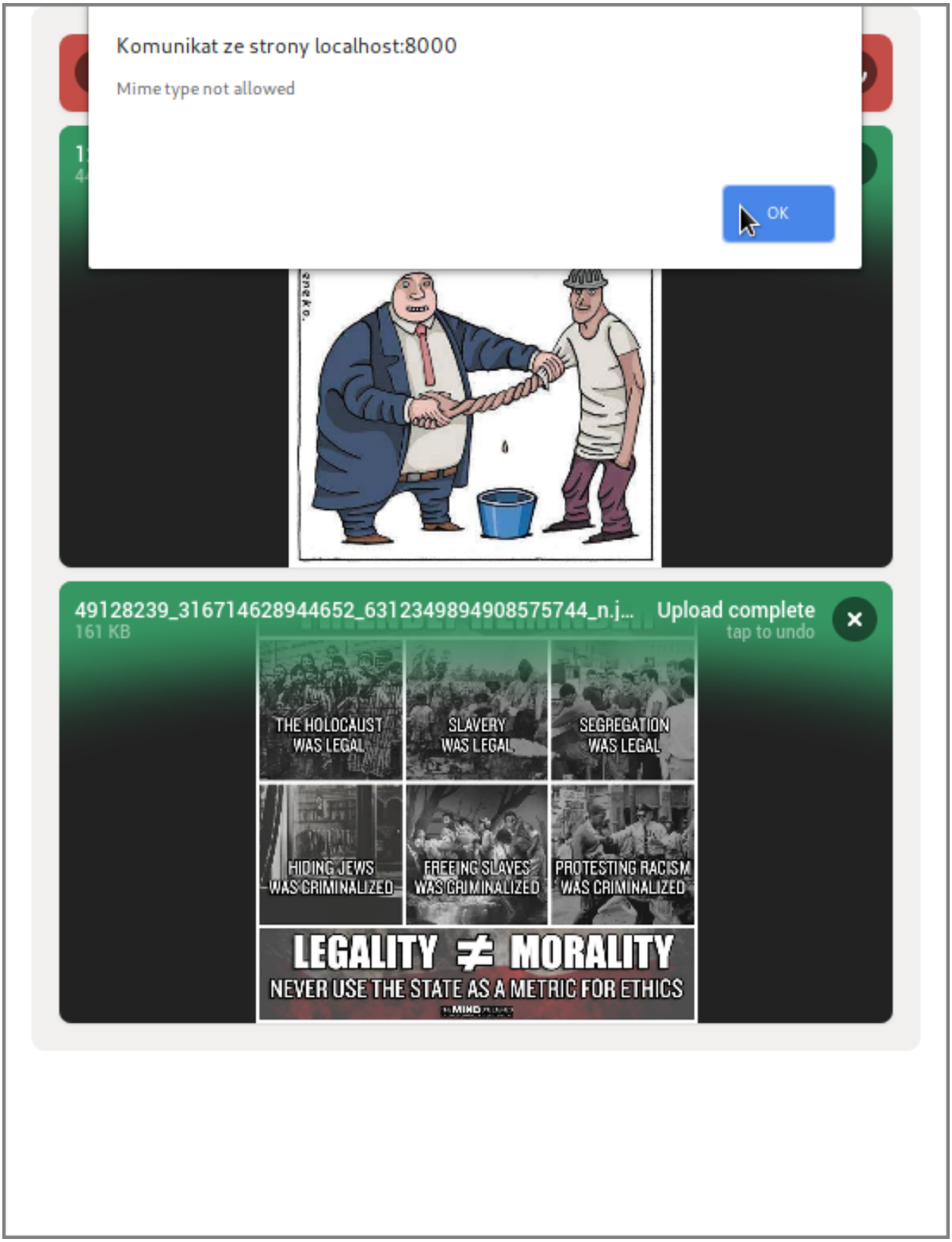


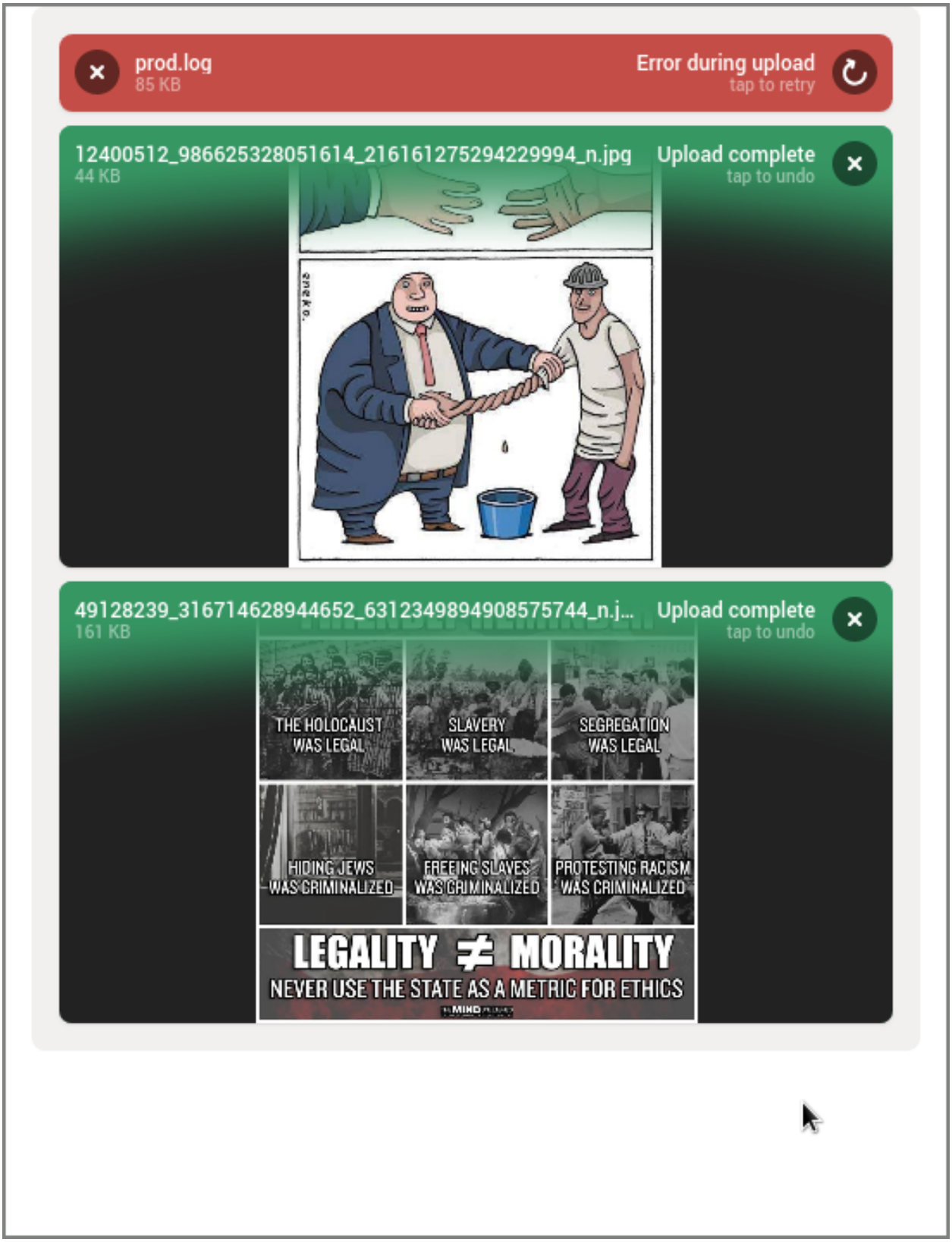
11.2.2 File upload

File upload offers a multiple file upload, with drag & drop and fancy animations.

```
In the browser access URL: /minimum.ui/upload/file?_token=TOKEN-THERE
```


URL address
http://localhost:8000/public/download/13bc7d6ab012400512_
Close
49128239_316714628944652_6312349894908575744_...
161 KB
Upload complete
tap to undo
THE HOLOCAUST WAS LEGAL
SLAVERY WAS LEGAL
SEGREGATION WAS LEGAL
HIDING JEWS WAS CRIMINALIZED
FREEING SLAVES WAS CRIMINALIZED
PROTESTING RACISM WAS CRIMINALIZED
LEGALITY ≠ MORALITY
NEVER USE THE STATE AS A METRIC FOR ETHICS

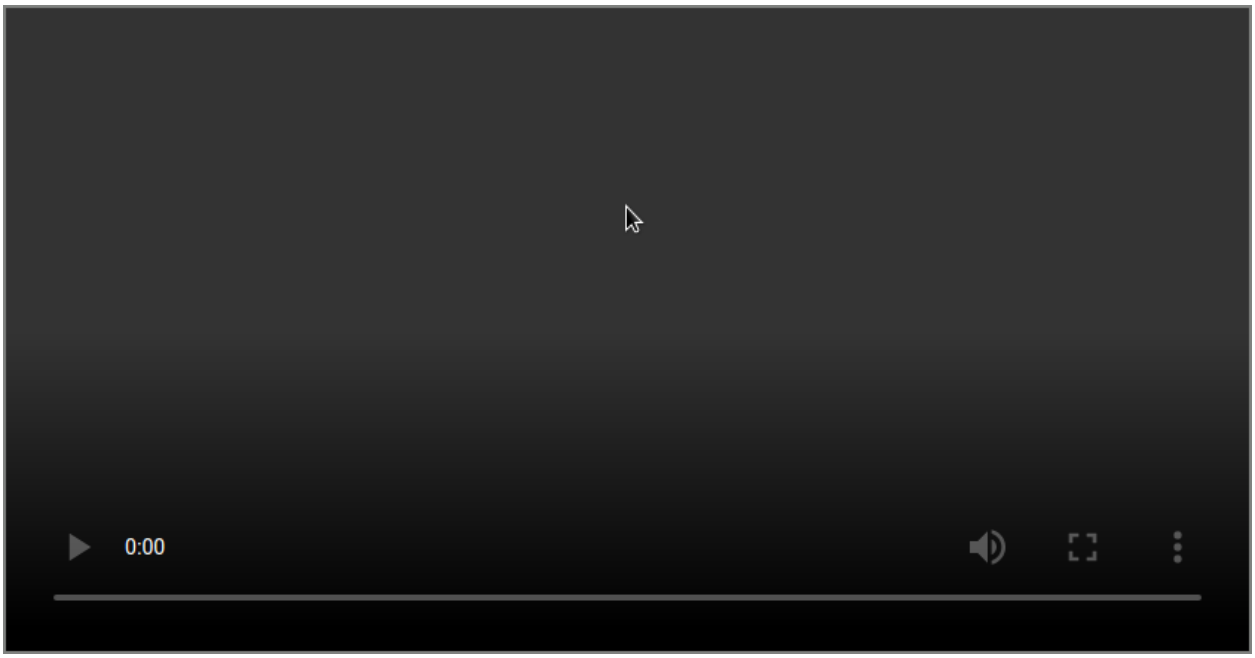

Komunikat ze strony localhost:8000
Mime type not allowed
OK
49128239_316714628944652_6312349894908575744_n.j...
161 KB
Upload complete
tap to undo
THE HOLOCAUST WAS LEGAL
SLAVERY WAS LEGAL
SEGREGATION WAS LEGAL
HIDING JEWS WAS CRIMINALIZED
FREEING SLAVES WAS CRIMINALIZED
PROTESTING RACISM WAS CRIMINALIZED
LEGALITY ≠ MORALITY
NEVER USE THE STATE AS A METRIC FOR ETHICS


prod.log
85 KB
Error during upload
tap to retry
12400512_986625328051614_216161275294229994_n.jpg
44 KB
Upload complete
tap to undo
49128239_316714628944652_6312349894908575744_n.j...
161 KB
Upload complete
tap to undo
THE HOLOCAUST WAS LEGAL
SLAVERY WAS LEGAL
SEGREGATION WAS LEGAL
HIDING JEWS WAS CRIMINALIZED
FREEING SLAVES WAS CRIMINALIZED
PROTESTING RACISM WAS CRIMINALIZED
LEGALITY ≠ MORALITY
NEVER USE THE STATE AS A METRIC FOR ETHICS

11.2.3 Video watching

File Repository is able to serve video files with possibility to rewind them, that's the responsibility of the download endpoint. MinimumUI exposes additional endpoint with a HTML5 <video> tag, so the video could be embedded easily on other website.

```
In the browser access URL: /minimum.ui/watch/video/some-file-name.mp4
```



CHAPTER 12

Bahub API client

Bahub is an automation tool for uploading and restoring backups. Works in shell, can work as a docker container in the same network with scheduled automatic backups of other containers, or can work as an UNIX daemon on the server without containerization.

```
→  bahub git:(master) ✗ python3 ./__init__.py --config ../configuration.yaml.dist backup my_logs
{'version': 59, 'file_id': 'BD15C9B8-66D6-424F-AD80-73B4BEC5D732', 'file_name': 'a625a578dfbackup.tar-v59.gz'}
→  bahub git:(master) ✗ python3 ./__init__.py --config ../configuration.yaml.dist list my_logs
{
    "v58": {
        "created": "2019-02-09 22:10:52.000000",
        "id": "752A1660-493F-4677-B0FF-7EB6FC7C3739"
    },
    "v59": {
        "created": "2019-02-09 22:14:38.000000",
        "id": "BD15C9B8-66D6-424F-AD80-73B4BEC5D732"
    }
}
→  bahub git:(master) ✗ python3 ./__init__.py --config ../configuration.yaml.dist restore my_logs v58
{"status": "OK"}
→  bahub git:(master) ✗ python3 ./__init__.py --config ../configuration.yaml.dist restore my_logs latest
{"status": "OK"}
```

12.1 Configuration reference

There are **3 sections**:

- Access: Describes authorization details, name it eg. server1 and put url and token
- Encryption: Encryption type and password (if any) to encrypt your files stored on **File Repository**
- Backups: Describes where is your data, how to access it and under which COLLECTION to send it to **File Repository**
- Recoveries: Recovery plans. A policy + list of "backups" to restore within a single command

Example scenario:

1. You have a server under https://backups.iwa-ait.org and token "XXX-YYY-ZZZ-123", you name it "ait_backups" under **access** section
2. You want to have encrypted backups using AES 256 CBC, then you add "ait_secret" under **encryption** with passphrase "something-secret" and type "aes-256-cbc"
3. Next you want to define where is the data, in our example it's in a docker container under /var/lib/mysql and we want to send this data to collection "123-456-789-000". You should reference "ait_backups" access and "ait_secret" as the encryption method for your backup there.

12.1.1 Environment variables

If you want to use environment variables, use bash-like syntax *${SOME_ENV_NAME}*.

NOTE: In case you will not set a variable in the shell, then application will not start, it will throw a configuration error.

12.1.2 Application configuration

Notice: Below example uses environment variables eg. ${DB_HOST}, you may want to replace them with values like localhost or others

12.2 Basic usage

Bahub is offering basic operations required to automate backup sending and receiving, not managing the server.

12.2.1 Sending a backup

```
$ bahub --config ~/.bahub.yaml backup some_local_dir
{'version': 72, 'file_id': 'E9D7103D-1789-475E-A3EE-9CF18F51ACA4', 'file_name':
↪'2b2e269541backup.tar-v72.gz'}
```

12.2.2 Listing stored backups

```
$ bahub --config ~/.bahub.yaml list some_local_dir
{
    "v71": {
        "created": "2019-02-10 14:27:52.000000",
        "id": "1684C60D-28B0-4818-A3EC-1F0C47981592"
    },
    "v72": {
        "created": "2019-02-11 07:54:52.000000",
        "id": "E9D7103D-1789-475E-A3EE-9CF18F51ACA4"
    }
}
```

12.2.3 Restoring a backup

Restoring latest version:

```
$ bahub --config ~/.bahub.yaml restore some_local_dir latest
{"status": "OK"}
```

Restoring version by number:

```
$ bahub --config ~/.bahub.yaml restore some_local_dir v71
{"status": "OK"}
```

Restoring version by id:

```
$ bahub --config ~/.bahub.yaml restore some_local_dir 1684C60D-28B0-4818-A3EC-
↪1F0C47981592
{"status": "OK"}
```

12.2.4 Recovery from disaster

In case you need to quickly recover whole server/environment from backup - there is a **RECOVERY PLAN**. A recovery plan is:

- List of backups to restore (names from section "backups")
- Policy of recovery (eg. recover everything, or stop on failure)

```
#
# Recovery plans
#   Restores multiple backups in order, using single command
#
#   Possible values:
#     policy:
#       - restore-whats-possible: Ignore things that cannot be restored, restore what␣
↪is posible
#       - stop-on-first-error: Restore until first error, then stay as it is
#
recoveries:
    default:
        policy: restore-whats-possible
        definitions: all

    plan_2:
        policy: stop-on-first-error
        definitions:
            - local_command_output
```

```
$ bahub --config ~/.bahub.yaml recover default
```

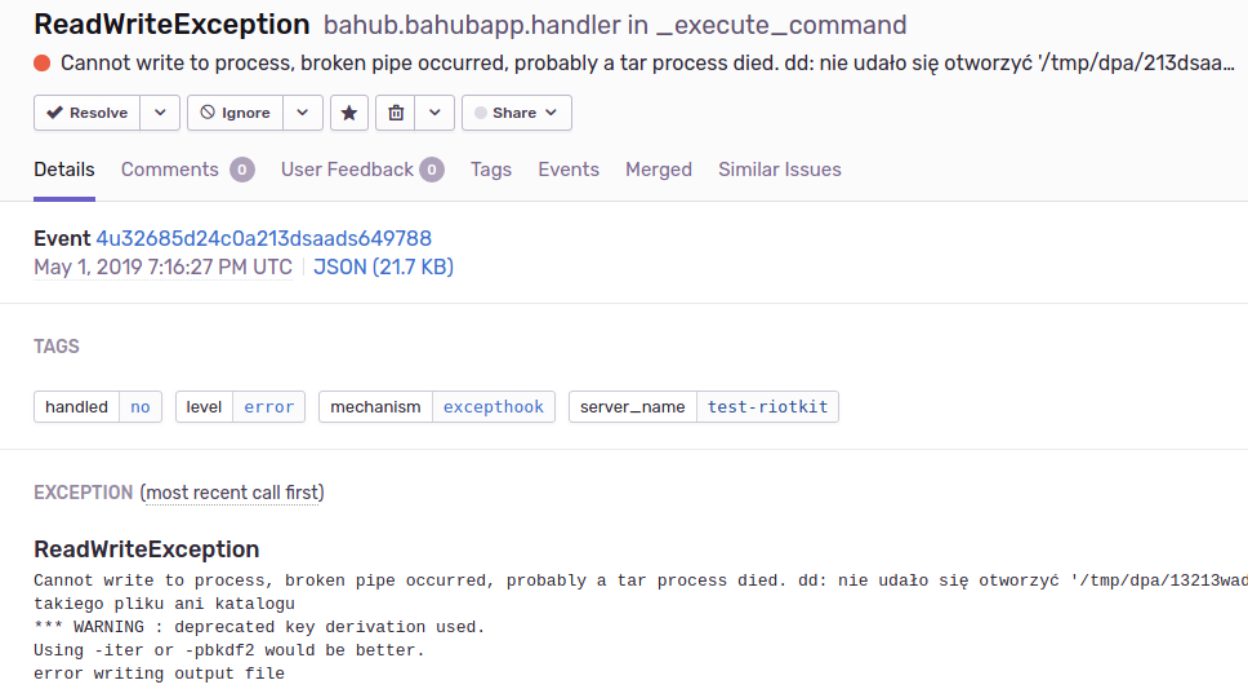
12.2.5 Making a snapshot of multiple services at once

Snapshot works exactly in the same way as **recovery from diaster**, but it's inverted. Instead of downloading a copy, it is actually uploading.

NOTICE: Be very careful, as this is a single command to backup everything, remember about the backups rotation

```
$ bahub --config ~/.bahub.yaml snapshot default

[2019-04-01 07:17:42,818][bahub][INFO]: Performing snapshot
[2019-04-01 07:17:42,819][bahub][INFO]: Performing a snapshot using "default" plan
[2019-04-01 07:17:42,819][bahub][DEBUG]: shell(sudo docker ps | grep "test_1")
[2019-04-01 07:17:42,870][bahub][DEBUG]: shell(set -o pipefail; sudo docker exec ␣
↪test_1 /bin/sh -c "[ -e /etc ] || echo does-not-exist"; exit $?)
[2019-04-01 07:17:42,967][bahub][DEBUG]: shell(set -o pipefail; sudo docker exec ␣
↪test_1 /bin/sh -c "tar -czf - \"/etc\" "| openssl enc -aes-128-cbc -pass␣
↪pass:Q**********************************************W; exit $?)
[2019-04-01 07:17:43,052][bahub][DEBUG]: shell(set -o pipefail; sudo docker exec ␣
↪test_1 /bin/sh -c "tar -czf - \"/etc\" "| openssl enc -aes-128-cbc -pass␣
↪pass:Q**********************************************W; exit $?)
[2019-04-01 07:17:45,672][bahub][DEBUG]: Request: https://api.backups.riotkit.org/
↪repository/collection/d**********************************9/backup?_
↪token=a********************************6
[2019-04-01 07:17:45,672][bahub][DEBUG]: response({"status":"OK","error_code":null,
↪"exit_code":200,"field":null,"errors":null,"version":{"id":"***************",
↪"version":1,"creation_date":{"date":"2019-04-01 05:17:45.492490","timezone_type":3,
↪"timezone":"UTC"},"file":{"id":110,"filename":"cd06f449fdtest-v2"}},"collection":{
↪"id":"d**********************************9","max_backups_count":1,"max_one_
↪backup_version_size":2000000000,"max_collection_size":8000000000,"created_at":{"date
↪":"2019-03-24 21:29:14.000000","timezone_type":3,"timezone":"UTC"},"strategy":
↪"delete_oldest_when_adding_new","description":"TEST","filename":"test"}})
[2019-04-01 07:17:45,673][bahub][INFO]: Finishing the process

{
    "failure": [],
    "success": [
        "test"
    ]
}
```

12.3 Monitoring errors with Sentry



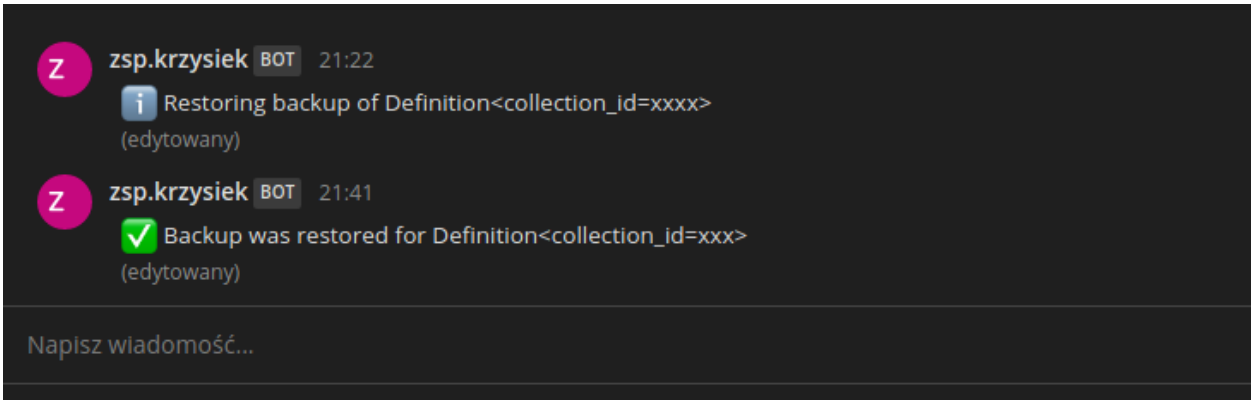
Bahub uses shell commands to take some data, pack it and encrypt. What if any of those commands will fail? What if there are no enough permissions? What if the directory does not exist? All of those are good reasons to have set up a monitoring.

Almost each application failure can be catched and sent to analysis. Don't worry about the privacy, you can use your own Sentry instance.

To enable the monitoring you need to have a ready-to-use Sentry instance/account and a error_handler configured in Bahub.

```
error_handlers:
    remote_sentry:   # name it as you want
        type: sentry
        url: "https://some-url"
```

12.4 Notifications



Each event such as upload success, restore success, or a failure can emit a notification.

```
notifiers:
    mattermost:     # name it as you want
        type: slack # compatible with Slack and Mattermost
        url: "https://xxxxx"
```

12.5 Setup

Bahub can be running as a separate container attached to docker containers network or manually as a regular process. The recommended way is to use a docker container, which provides a working job scheduling, installed dependencies and preconfigured most of the things.

12.6 Using docker container

There exists a bahub tag on the docker hub container, **wolnosciowiec/file-repository:bahub** You can find an example in "examples/client" directory in the repository.

docker-compose.yml

```
version: "2"
services:
    #
    # Our container that is running all the time, can run scheduled backups and␣
↪manually triggered backups
    #
    backup:
        image: quay.io/riotkit/bahub:dev
        volumes:
            - "./cron:/cron:ro"
            - "./config.yaml:/bahub.conf.yaml:ro"
            - "/var/run/docker.sock:/var/run/docker.sock"
        environment:
            - BACKUPS_ENCRYPTION_PASSPHRASE=some-very-long-passphrase-good-to-have-
↪there-64-characters-for-example
            - BACKUPS_TOKEN=111111-2222-3333-4444-55555555555555
```

(continues on next page)

(continued from previous page)

```
            - BACKUPS_REDIS_COLLECTION_ID=12345678-cccc-bbb-aaa-1232313213123
            - COMPOSE_PROJECT_NAME=test_client

    #
    # Test container for backup & restore
    #
    redis:
        image: redis:3-alpine
        volumes:
            - ./redis:/data
        command: "redis-server --appendonly yes"
```

/cron

```
# schedule REDIS server backup on every Monday, 02:00 AM
0 2 * * MON bahub backup some_redis_storage
```

/bahub.conf.yaml (see: *Configuration reference*)

```
accesses:
    some_server:
        url: http://api.some-domain.org
        token: "${BACKUPS_TOKEN}"

encryption:
    my_aes:
        passphrase: "${BACKUPS_ENCRYPTION_PASSPHRASE}"
        method: "aes-128-cbc"

backups:
    some_redis_storage:
        type: docker_volumes
        container: "${COMPOSE_PROJECT_NAME}_redis_1"
        access: some_server
        encryption: my_aes
        collection_id: "${BACKUPS_REDIS_COLLECTION_ID}"
        paths:
            - "/data"
```

*Note: It's very important to specify the project name in docker-compose with "-p", so it will have same value as "COMPOSE_PROJECT_NAME". You may want to add it to .env file and reuse in Makefile and in docker-compose.yml for automation**

12.7 Using bare metal

Use Python's PIP to install the package, and run it.

```
pip install bahub
bahub --help
```

CHAPTER 13

Shell access

File Repository usage can be automated using shell commands. There are not so many commands, but basic usage could be automated using scripts.

13.1 Introduction

Application is using *Symfony Console*, which is accessible in the main directory under **./bin/console** In our prepared docker compose environment you may use it differently.

Usage examples depending on how application is set up	
type	example
our docker env.	make console OPTS="backup:create-collection -d "Some test collection" -f "backup.tar.gz" -b 4 -o 3GB -c 15GB"
docker standalone	sudo docker exec -it some_container_name ./bin/console backup:create-collection -d "Some test collection" -f "backup.tar.gz" -b 4 -o 3GB -c 15GB
standalone/manual	./bin/console backup:create-collection -d "Some test collection" -f "backup.tar.gz" -b 4 -o 3GB -c 15GB

If something is not working as expected, there is an error and you would like to inspect it, then please add a "-vvv" switch to increase verbosity.

13.1.1 Managing authentication using console commands

Tokens can be easily generated without touching the cURL or browser or any API client. Just use the console.

Generating an unlimited administrative token

Probably first time when you set up the **File Repository** you may want to create a token, that will allow you to fully manage everything. We already knew about such case and we're prepared for it! ;-)

```
 ./bin/console auth:generate-admin-token
Generating admin token...
=========================
Form:
 [Role] -> security.administrator

Response:
=========================
{
    "tokenId": "1B3B15EC-18E9-45DD-846B-42C5006E872A",
    "expires": "2029-02-11 07:24:42"
}
```

In this case "1B3B15EC-18E9-45DD-846B-42C5006E872A" is your administrative token, pssst... keep it safe!

Generating a normal token

It is considered a very good practice to minimize access to the resources. For example the server which will be storing backups on the **File Repository** should only be allowed to send backups, not deleting for example.

For such cases you can generate a token that will allow access to specified collections and limit actions on them.

```
 ./bin/console auth:create-token --help
Description:
  Creates an authentication token

Usage:
  auth:create-token [options]

Options:
      --roles=ROLES
      --tags=TAGS
      --mimes=MIMES
      --max-file-size=MAX-FILE-SIZE
      --expires=EXPIRES             Example: 2020-05-01 or +10 years
  -h, --help                        Display this help message
  -q, --quiet                       Do not output any message
  -V, --version                     Display this application version
      --ansi                        Force ANSI output
      --no-ansi                     Disable ANSI output
  -n, --no-interaction              Do not ask any interactive question
  -e, --env=ENV                     The Environment name. [default: "dev"]
      --no-debug                    Switches off debug mode.
  -v|vv|vvv, --verbose              Increase the verbosity of messages: 1 for normal␣
↪output, 2 for more verbose output and 3 for debug

Help:
  Allows to generate a token you can use later to authenticate in application for a␣
↪specific thing
```

Example of generating a token with specified roles:

```
 ./bin/console auth:create-token --roles upload.images,upload.enforce_no_password --
↪expires="+30 minutes"
=========================
Form:
```

(continues on next page)

(continued from previous page)

```
 [Role] -> upload.images
 [Role] -> upload.enforce_no_password

Response:
=========================
{
    "tokenId": "A757A8CB-964F-4F7B-BB70-9DB2CF524BB9",
    "expires": "2019-02-11 08:01:00"
}
```

Deleting expired tokens

This should be a scheduled periodic job in a cronjob, that would delete tokens that already are expired.

```
 ./bin/console auth:clear-expired-tokens
[2019-02-05 08:07:01] Removing token 276CCE10-00C5-4CB6-9F9A-87934101BACE
```

CHAPTER 14

General guide for Administrators, DevOps and Developers

There is a general guide on how to maintain a backup server, what is the common approach to setup a server from Riotkit template and more.

The guide is on a separate repository: https://github.com/riotkit-org-education/guide

Check also our RiotKit Education organization at https://github.com/riotkit-org-education , where we teach basic and mid-advanced things.

CHAPTER 15

From authors

Project was started as a part of RiotKit initiative, for the needs of grassroot organizations such as:

- Fighting for better working conditions syndicalist (International Workers Association for example)
- Tenants rights organizations
- Various grassroot organizations that are helping people to organize themselves without authority

Technical description:

Project was created in *Domain Driven* like design in PHP 7, with Symfony 4 framework. There are API tests written in *Postman* and unit tests written in *PhpUnit*. Feel free to submit pull requests, report issues, or join our team. The project is licensed with a MIT license.

RiotKit Collective