
File Repository Documentation

Release 2

Wolnosciewicz Team

Nov 16, 2019

Contents:

1	First steps	3
2	Manual installation	5
3	Installation with docker	7
4	Post-installation	11
5	Configuration reference	13
5.1	Application configuration	13
5.2	Permissions list	16
5.3	Docker container extra parameters	19
6	Using postman to manage the application	21
7	Authorization	25
7.1	Creating a token	25
7.2	Looking up a token	26
7.3	Revoking a token	26
8	Files storage	29
8.1	Security	29
8.2	Uploading	30
8.3	Downloading	31
8.4	Aliasing filenames (migrating existing files to File Repository)	32
8.5	Hotlink protection - personalizing URLs for your visitors	33
9	Backup	37
9.1	Getting started	37
9.2	Managing collections	38
9.3	Authorization	40
9.4	Backups: Upload, deletion and versioning	41
9.5	Data replication	44
9.6	Managing collections from shell	44
10	MinimumUI	47
10.1	Quick start in steps	47
10.2	Endpoints	47

11 Bahub API client	55
11.1 Configuration reference	55
11.2 Basic usage	60
11.3 Monitoring errors with Sentry	63
11.4 Notifications	64
11.5 Setup	64
11.6 Using docker container	64
11.7 Using bare metal	65
12 Shell access	67
12.1 Introduction	67
13 General guide for Administrators, DevOps and Developers	71
14 From authors	73

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, Dropbox, Google Drive, FTP, SFTP, 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**
- Pure API, you can choose any frontend, use it internally in your application, or create your own frontend

CHAPTER 1

First steps

To start using the application you need to install PHP 7 with extensions listed in *composer.json* file (see entries ext-{name}), composer, and a database engine - SQLite3 or MySQL server.

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:

- PHP7.2 or newer
- SQLite3 or MySQL database
- Composer (PHP package manager, see packagist.org)
- make (GNU Make)

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**.

```
cp .env .env.dist
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

You have at least three choices:

- Use *wolnosciewicz/file-repository* container by your own (advanced)
- Use a prepared docker-compose environment placed in *examples/docker* directory
- Create your own environment based on docker-compose

Proposed way to choose is the prepared docker-compose environment that is placed in *examples/docker* directory. Here are instructions how to start with it:

```
# go to the environment directory and copy template file
cd ./examples/docker
cp .env.dist .env

# adjust application settings
edit .env
```

Now adjust the environment variables to your need - you might want to see the configuration reference. If you think the configuration is finished, start the environment. To stop it - type CTRL+C.

```
# start the environment
make start
```

Example docker-compose.yml file:

```
1 version: '2'
2   services:
3       #
4       # Optional: You can use AWS S3, Minio, but actually you don't have to.
5       # "local" adapter may be also used, then files would be stored_
6       #
7       # S3 gives possibility to replicate the data and/or redistribute it,
8       # relocate
9       #
```

(continues on next page)

(continued from previous page)

```

9      app_minio:
10         image: minio/minio
11         volumes:
12             - ./data/minio:/data
13         expose:
14             - "9000"
15         command: server /data
16         restart: on-failure
17         mem_limit: 367001600 # 350M
18         environment:
19             - MINIO_DOMAIN=app_minio
20             - MINIO_ACCESS_KEY=hello
21             - MINIO_SECRET_KEY=hello123
22
23     #
24     # File Repository API application
25     #
26     app_storage:
27         image: wolnosciewicz/file-repository:latest
28         environment:
29             - DATABASE_URL=mysql://file_repository:playground@db_mysql:3306/file_
↪ repository
30             - DATABASE_DRIVER=pdo_mysql
31
32         # SQLite3 example
33         #- DATABASE_URL=sqlite:///kernel.project_dir%/var/data.db
34         #- DATABASE_DRIVER=pdo_sqlite
35
36         # minio adapter setup
37         - FS_AWSS3V3_CLIENT=s3_client
38         - FS_AWSS3V3_BUCKET=misc
39         - FS_AWSS3V3_PREFIX=
40         - FS_AWSS3V3_OPTIONS_ENDPOINT=http://app_minio:9000
41         - FS_AWSS3V3_VERSION=latest
42         - FS_AWSS3V3_KEY=hello
43         - FS_AWSS3V3_SECRET=hello123
44         - BACKUP_ONE_VERSION_MAX_SIZE=25GB
45         - BACKUP_COLLECTION_MAX_SIZE=150GB
46         - WAIT_FOR_HOST=db_mysql:3306
47
48         # local adapter setup:
49         #- FS_ADAPTER=local
50         #- FS_LOCAL_DIRECTORY=%kernel.root_dir%/../var/uploads
51     env_file:
52         - .env
53     expose:
54         - "80"
55     depends_on:
56         - db_mysql
57     restart: on-failure
58     mem_limit: 524288000 # 500M
59
60     #
61     # Optional: A SQLite3 can be used instead MySQL
62     #             MySQL gives a possibility to replicate the data
63     #
64     db_mysql:

```

(continues on next page)

(continued from previous page)

```
65     image: mysql:8
66     environment:
67         - MYSQL_USER=file_repository
68         - MYSQL_PASSWORD=playground
69         - MYSQL_DATABASE=file_repository
70         - MYSQL_ROOT_PASSWORD=root
71     volumes:
72         - "./data/mysql:/var/lib/mysql"
73     command: mysqld --innodb_file_per_table=1
74
75     #
76     # Test: Our test application we will make a backup for
77     #       Access it via localhost:8100, modify it's files in ./www
78     #
79     app_webserver:
80         image: nginx
81         volumes:
82             - "./www:/usr/share/nginx/html"
83         ports:
84             - "8100:80"
85
86     #
87     # Test: Backup client, normally it should be on a separate machine.
88     #       For testing we have it in same docker network.
89     #
90     backup_client:
91         image: wolnosciowiec/file-repository:bahub
92         volumes:
93             - "./data/backup/cron:/cron:ro"
94             - "./data/backup/bahub.conf.yaml:/bahub.conf.yaml:ro"
95             - "/var/run/docker.sock:/var/run/docker.sock"
```


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.override": "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.

Configuration reference

5.1 Application configuration

```
# This file is a "template" of which env vars need to be defined for your application
# Copy this file to .env file for development, create environment variables when
↳ deploying to production
# https://symfony.com/doc/current/best_practices/configuration.html#infrastructure-
↳ related-configuration

#
# Symfony framework specific configuration
#
APP_ENV=prod # possible values: prod, test, dev. Use "prod" for public instances.
↳ Use "test" installing the application and changing its configuration
APP_SECRET=faec5e5fcf0ff499f53ecc30bdfc653
#TRUSTED_PROXIES=127.0.0.1,127.0.0.2
#TRUSTED_HOSTS=localhost,example.com

# how long can take request if it is expected, that it will take long (eg. file
↳ upload)
# please consider there if you have request buffering enabled in webserver, nginx has
↳ this by default
LONG_EXECUTION_TIME=300
# application domain (if empty, then it will be taken from Host header. If Host
↳ header is invalid, then fill this variable)
APP_DOMAIN=
# default expiration time for generated tokens
TOKEN_EXPIRATION_TIME="+30 minutes"
# used for tokens encryption (tokens can be generated by external applications)
ENC_TOKEN_PHRASE=test123
TEMP_DIRECTORY=/tmp

#
# Application connection details
```

(continues on next page)

(continued from previous page)

```

#

# SQLite3 example
DATABASE_URL=sqlite:///kernel.project_dir%/var/data.db
DATABASE_DRIVER=pdo_sqlite

# MySQL example
#DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name
DATASAE_VERSION=5.7

#DATABASE_DRIVER=pdo_mysql
# Possibly you can use also PostgreSQL and others which are supported by Symfony 4_
↪and Doctrine ORM
# please take a look there: https://symfony.com/doc/current/reference/configuration/
↪doctrine.html

BACKUP_ONE_VERSION_MAX_SIZE=4GB
BACKUP_COLLECTION_MAX_SIZE=15GB

#
# Anti-Hotlink protection
# (NOTICE: Requires application to be restarted or ./bin/console cache:clear to be_
↪executed)
#
ANTI_HOTLINK_PROTECTION_ENABLED=true
ANTI_HOTLINK_RESTRICT_REGULAR_URLS=false
ANTI_HOTLINK_URL=/stream/{accessToken}/{expirationTime}/{fileId}
ANTI_HOTLINK_CRYPT=md5
ANTI_HOTLINK_SECRET_METHOD="\$http_x_expiration_time\$filename\$http_remote_addr MY-
↪AWESOME-SUFFIX"

#
# Storage type
#
# local: local filesystem
# awss3: AWS S3, Minio.io or other compatible with AWS S3 interface
# ftp: FTP/FTPS
#
FS_ADAPTER=local

#
# === Local filesystem ===
#

# path, where for local filesystem the files should be stored. %kernel.root_dir% is_
↪the application main directory
FS_LOCAL_DIRECTORY="%kernel.root_dir%/../var/uploads"

# (?) see: http://flysystem.thephpleague.com/docs/adapters/local/
FS_LOCAL_LAZY=0

# file locking; 1 - LOCK_SH, 2 - LOCK_EX, 3 - LOCK_UN, 4 - LOCK_NB
FS_LOCAL_WRITEFLAGS=

# how to handle symbolic links: 1 - skip links, 2 - disallow links
FS_LOCAL_LINKHANDLING=

```

(continues on next page)

(continued from previous page)

```
# permissions chmod eg. 0755
FS_LOCAL_PERMISSIONS=

#
# === S3 ===
#

# set to enable S3 interface as adapter (supports Minio, and possibly Amazon AWS S3)
#FS_ADAPTER=awss3v3
# (advanced) Symfony service name of an S3 Client
FS_AWSS3V3_CLIENT=s3_client
FS_AWSS3V3_BUCKET=misc
FS_AWSS3V3_PREFIX=
FS_AWSS3V3_OPTIONS_ENDPOINT=http://localhost:9000
FS_AWSS3V3_VERSION=latest
FS_AWSS3V3_REGION=eu-central-1
FS_AWSS3V3_KEY=some-key
FS_AWSS3V3_SECRET=some-secret

#
# === FTP ===
#

# there you can enable FTP adapter
#FS_ADAPTER=ftp

# FTP host where to connect to, IP address or hostname
FS_FTP_HOST=localhost

# FTP port to connect to
FS_FTP_PORT=21

# FTP authorization: username
FS_FTP_USERNAME=user

# FTP authorization: user's password
FS_FTP_PASSWORD=password

# FTP directory where files will be stored in
FS_FTP_ROOT=/

# Use SSL? (FTPS instead of FTP)
FS_FTP_SSL=1

# Timeout
FS_FTP_TIMEOUT=120

# Permissions for owner eg. 0755
FS_FTP_PERMPRIVATE=

# Permissions for public (others) eg. 0400
FS_FTP_PERMPUBLIC=

# Passive mode?
FS_FTP_PASSIVE=0
```

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

```
<?php declare(strict_types=1);

namespace App\Domain;

/**
 * List of roles which could be required for a temporary token
 *
 * @codeCoverageIgnore
 */
final class Roles
{
    //
    //
    // upload/creation
    //
    //

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

    /** Allows to upload video files */
    public const ROLE_UPLOAD_VIDEOS = 'upload.videos';

    /** Allows to upload documents */
    public const ROLE_UPLOAD_DOCS = 'upload.documents';

    /** Allows to submit backups */
    public const ROLE_UPLOAD_BACKUP = 'upload.backup';

    /** Allows to upload ALL types of files regardless of mime type */
    public const ROLE_UPLOAD = 'upload.all';

    /** Enforce no password for all uploads for this token */
    public const ROLE_UPLOAD_ENFORCE_NO_PASSWORD = 'upload.enforce_no_password';

    /** Enforce token tags. In result every uploaded file will have tags specified in_
    ↪token regardless if they were sent in request */
    public const ROLE_UPLOAD_ENFORCE_TOKEN_TAGS = 'upload.enforce_tags_selected_in_
    ↪token';
```

(continues on next page)

(continued from previous page)

```

//
//
// authentication and tokens
//
//

/** User can check information about ANY token */
public const ROLE_LOOKUP_TOKENS = 'security.authentication_lookup';

/** User can overwrite files */
public const ROLE_ALLOW_OVERWRITE_FILES = 'security.overwrite';

/** User can generate tokens with ANY roles */
public const ROLE_GENERATE_TOKENS = 'security.generate_tokens';

/** User can use technical endpoints to manage the application */
public const ROLE_USE_TECHNICAL_ENDPOINTS = 'security.use_technical_endpoints';

/** User can expire other token, so it will be not valid anymore */
public const ROLE_REVOKE_TOKENS = 'security.revoke_tokens';

/** Special: Marking - tokens with this marking will not be able to be revoked by
↳non-administrators */
public const ROLE_ADMINISTRATOR = 'security.administrator';

//
//
// deletion
//
//

/** Delete files that do not have a password, and password protected without a
↳password */
public const ROLE_DELETE_ALL_FILES = 'deletion.all_files_including_
↳protected_and_unprotected';

//
//
// browsing
//
//

/** Allows to download ANY file, even if a file is password protected*/
public const ROLE_VIEW_ALL_PROTECTED_FILES = 'view.any_file';

/** List files from ANY tag that was requested, else the user can list only files
↳by tags allowed in token */
public const ROLE_BROWSE_LIST_OF_FILES_BY_ANY_TAG = 'view.files_from_all_tags';

/** Define that the user can use the listing endpoint (basic usage) */
public const ROLE_ACCESS_LISTING_ENDPOINT = 'view.can_use_listing_endpoint_at_all
↳';

//
//
// collections

```

(continues on next page)

(continued from previous page)

```

//
//

/** Allow person creating a new backup collection */
public const ROLE_COLLECTION_ADD = 'collections.create_new';

/** Allow creating backup collections that have no limits on size and length */
public const ROLE_COLLECTION_ADD_WITH_INFINITE_LIMITS = 'collections.allow_
↳infinite_limits';

/** Edit collections where token is added as allowed */
public const ROLE_MODIFY_ALLOWED_COLLECTIONS = 'collections.modify_details_of_
↳allowed_collections';

/** Allow to modify ALL collections. Collection don't have to allow such token_
↳which has this role */
public const ROLE_COLLECTION_MODIFY_ANY_COLLECTION = 'collections.modify_any_
↳collection_regardless_if_token_was_allowed_by_collection';

/** Allow to browse any collection regardless of if the user token was allowed by_
↳it or not */
public const ROLE_COLLECTION_VIEW_ANY_COLLECTION = 'collections.view_all_
↳collections';

/** Can use an endpoint that will allow to browse and search collections? */
public const ROLE_CAN_USE_LISTING_COLLECTION_ENDPOINT = 'collections.can_use_
↳listing_endpoint';

/** Manage tokens in the collections where our current token is already added as_
↳allowed */
public const ROLE_CAN_MANAGE_TOKENS_IN_ALLOWED_COLLECTIONS = 'collections.manage_
↳tokens_in_allowed_collections';

/** Delete collections where token is added as allowed */
public const ROLE_CAN_DELETE_ALLOWED_COLLECTIONS = 'collections.delete_allowed_
↳collections';

/** Upload to allowed collections */
public const ROLE_CAN_UPLOAD_TO_ALLOWED_COLLECTIONS = 'collections.upload_to_
↳allowed_collections';

/** List versions for collections where the token was added as allowed */
public const ROLE_LIST_VERSIONS_FOR_ALLOWED_COLLECTIONS = 'collections.list_
↳versions_for_allowed_collections';

/** Delete versions only from collections where the token was added as allowed */
public const ROLE_DELETE_VERSIONS_IN_ALLOWED_COLLECTIONS = 'collections.delete_
↳versions_for_allowed_collections';

/** Collection manager: Create, edit, delete collections */
public const GROUP_COLLECTION_MANAGER = [
    self::ROLE_COLLECTION_ADD,
    self::ROLE_COLLECTION_ADD_WITH_INFINITE_LIMITS,
    self::ROLE_COLLECTION_MODIFY_ANY_COLLECTION,
    self::ROLE_COLLECTION_VIEW_ANY_COLLECTION,
    self::ROLE_CAN_USE_LISTING_COLLECTION_ENDPOINT,
    self::ROLE_CAN_MANAGE_TOKENS_IN_ALLOWED_COLLECTIONS,

```

(continues on next page)

(continued from previous page)

```

        self::ROLE_CAN_UPLOAD_TO_ALLOWED_COLLECTIONS,
        self::ROLE_LIST_VERSIONS_FOR_ALLOWED_COLLECTIONS,
        self::ROLE_DELETE_VERSIONS_IN_ALLOWED_COLLECTIONS
    ];

    public const GRANTS_LIST = [
        self::ROLE_UPLOAD_IMAGES,
        self::ROLE_UPLOAD_DOCS,
        self::ROLE_UPLOAD_BACKUP,
        self::ROLE_UPLOAD,
        self::ROLE_LOOKUP_TOKENS,
        self::ROLE_ALLOW_OVERWRITE_FILES,
        self::ROLE_GENERATE_TOKENS,
        self::ROLE_USE_TECHNICAL_ENDPOINTS,
        self::ROLE_DELETE_ALL_FILES,
        self::ROLE_VIEW_ALL_PROTECTED_FILES,
        self::ROLE_BROWSE_LIST_OF_FILES_BY_ANY_TAG,
        self::ROLE_ACCESS_LISTING_ENDPOINT,
        self::ROLE_REVOKE_TOKENS,
        self::ROLE_ADMINISTRATOR,

        self::ROLE_COLLECTION_ADD,
        self::ROLE_COLLECTION_ADD_WITH_INFINITE_LIMITS,
        self::ROLE_CAN_DELETE_ALLOWED_COLLECTIONS,
        self::ROLE_COLLECTION_MODIFY_ANY_COLLECTION,
        self::ROLE_COLLECTION_VIEW_ANY_COLLECTION,
        self::ROLE_CAN_USE_LISTING_COLLECTION_ENDPOINT,
        self::ROLE_CAN_MANAGE_TOKENS_IN_ALLOWED_COLLECTIONS,
        self::ROLE_CAN_UPLOAD_TO_ALLOWED_COLLECTIONS,
        self::ROLE_LIST_VERSIONS_FOR_ALLOWED_COLLECTIONS,
        self::ROLE_DELETE_VERSIONS_IN_ALLOWED_COLLECTIONS
    ];

    public const RESTRICTIONS_LIST = [
        self::ROLE_UPLOAD_ENFORCE_NO_PASSWORD,
        self::ROLE_UPLOAD_ENFORCE_TOKEN_TAGS
    ];

    public static function getRolesList(): array
    {
        return \array_merge(self::GRANTS_LIST, self::RESTRICTIONS_LIST);
    }
}

```

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:

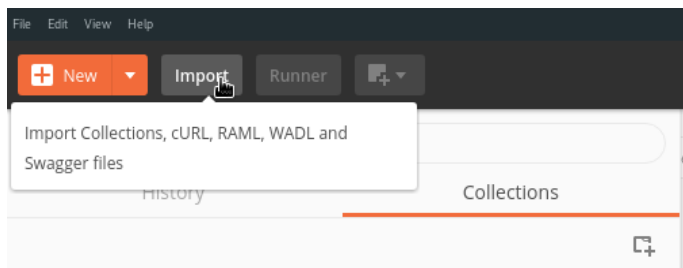
Name and example	Description
WAIT_FOR_HOST=db_mysql:3306	(optional) Waits up to 2 minutes for host to be up when starting a container

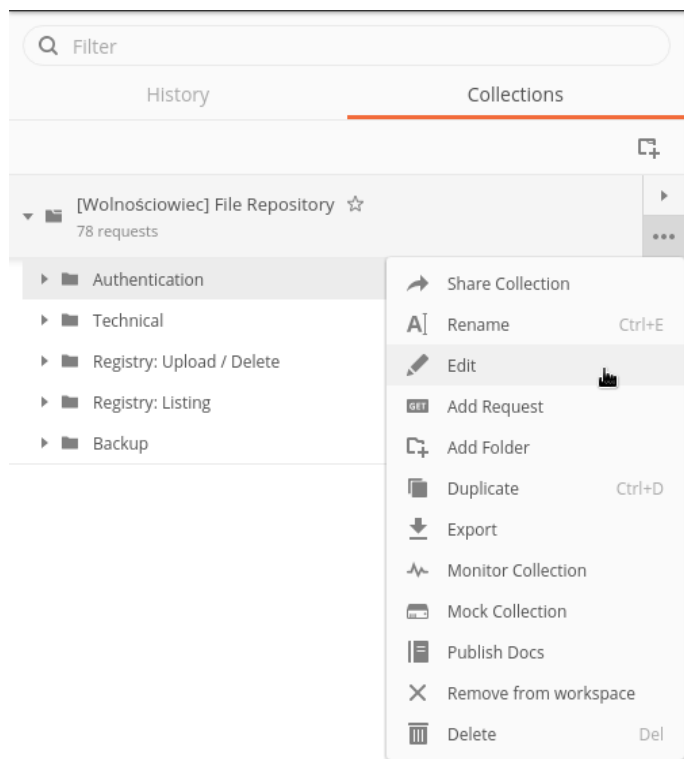
CHAPTER 6

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.





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
<input checked="" type="checkbox"/>	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

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
- Allowed file types (mime types) when uploading
- Maximum allowed file size
- Token expiration date

7.1 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
data.allowedMimeTypes	List of allowed mime types
data.maxAllowedFileSize	Number of bytes of maximum file size
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
↪"],
```

(continues on next page)

(continued from previous page)

```
"data": {
  "tags": [],
  "allowedMimeTypes": ["image/jpeg", "image/png", "image/gif"],
  "maxAllowedFileSize": 14579
},
"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

7.2 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.ul23",
    "user_uploads"
  ],
  "mimes": [
    "image/jpeg",
    "image/png",
    "image/gif"
  ],
  "max_file_size": 14579
}
```

Required roles:

- security.authentication_lookup

7.3 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 8

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.

8.1 Security

8.1.1 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.

8.1.2 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

8.2 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)

8.2.1 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
}
```

8.2.2 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)

8.2.3 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 ...
```

8.3 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

8.3.1 Regular downloading

It's very simple.

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

8.3.2 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
```

8.3.3 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

8.4 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.

8.4.1 Scenario

Let's assume that you have a file named "Accidental-Anarchist.mp4", and your website shows a player that points to <https://static.iwa-ait.org/Accidental-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 “Accidental-Anarchist.mp4”, after uploading to File Repository it will have different name eg. “59dce00bcAccidental-Anarchist.mp4”, you can create an alias that will point from “Accidental-Anarchist.mp4” to “59dce00bcAccidental-Anarchist.mp4”

8.4.2 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:

```
#
# File name mapping for file downloading
# -----
# Purpose: Allows to create single or multiple aliases for a file name
# Example usage: When you migrate from other file storage application to the File_
↳Repository
#
#           then you probably have files with names without eg. hash
#           After you import all files to the File Repository and rewrite urls_
↳on eg. NGINX
#
#           still the identifiers are different. File Repository uses hash +_
↳filename and other storage
#
#           may use other naming convention.
#
#
# To enable this file simply rename it removing the ".example" from the filename.
#

some_name: 3fcfca2d27d1681e2b72b07709174ec8a8bb618666b5d4c4a18618f974350deb_image.jpg
"example-alias": "c7beb8a9f0mapped.txt"
Accidental-Anarchist.mp4: "59dce00bcAccidental-Anarchist.mp4"
```

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

8.5 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**.

8.5.1 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
<code>\$http_xxx</code>	In place of xxx put your normalized header name eg. Content-Type would be <code>content_type</code>
<code>\$server_xxx</code>	Everything what is in PHP's <code>\$_SERVER</code> , including environment variables
<code>\$query_xxx</code>	Everything what is in query string (query string in URL is everything after question mark)

8.5.2 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 = 'Accidental-Anarchist.mp4';
```

(continues on next page)

(continued from previous page)

```
$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;
```


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.

9.1 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

9.1.1 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**)

9.1.2 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)

9.1.3 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.

9.2 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.

9.2.1 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)

9.2.2 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)

9.2.3 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)

9.2.4 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)

9.3 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.

9.3.1 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`

9.3.2 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`

9.4 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.

9.4.1 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

9.4.2 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

9.4.3 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

9.4.4 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!*

9.5 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](https://min.io/docs/en/concepts/replication.html) that has a possibility to configure multiple nodes in primary-replica model.

9.6 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.

9.6.1 Creating collections

```

➤ file-repository git:(master) x ./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": []
}

```


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.

10.1 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

10.2 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.

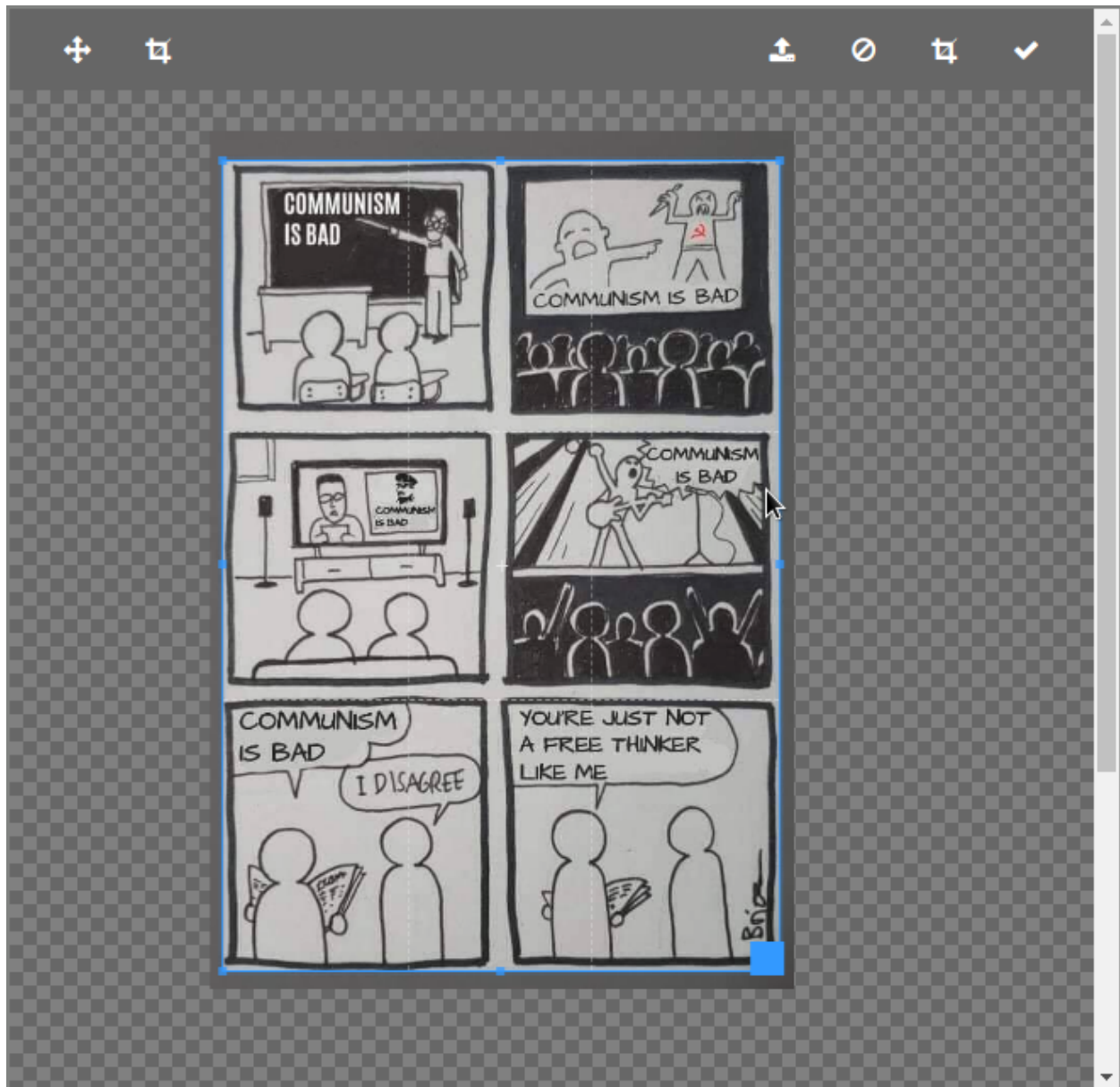
Roles used by the endpoints	
name	description
upload.enforce_no_password	Enforce the file to be uploaded without a password
upload.enforce_tags_selected_in_token	Tag uploaded file with tags specified in the token, regardless of user choice
upload.images	Upload images

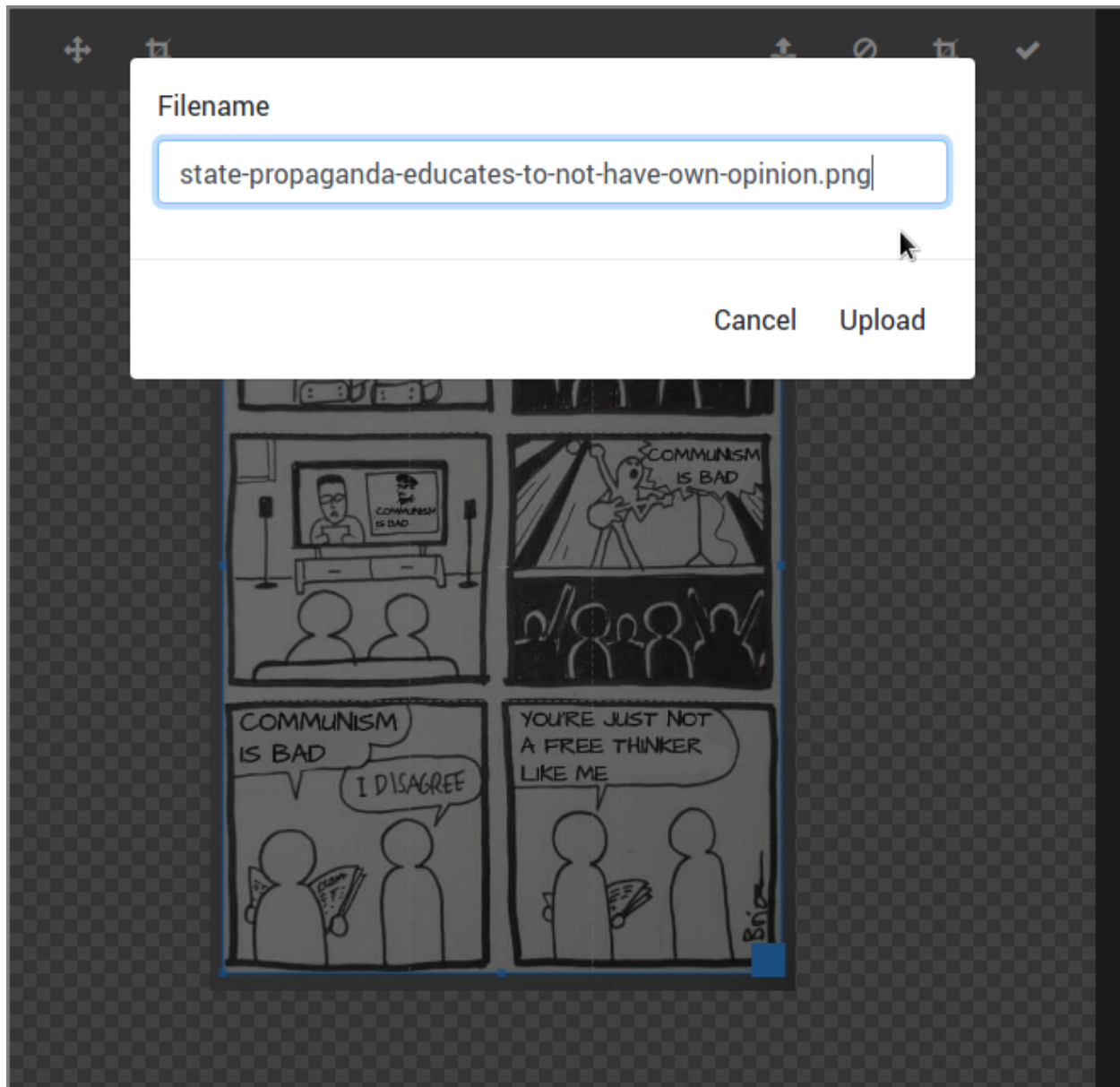
10.2.1 Image Upload

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
--





10.2.2 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

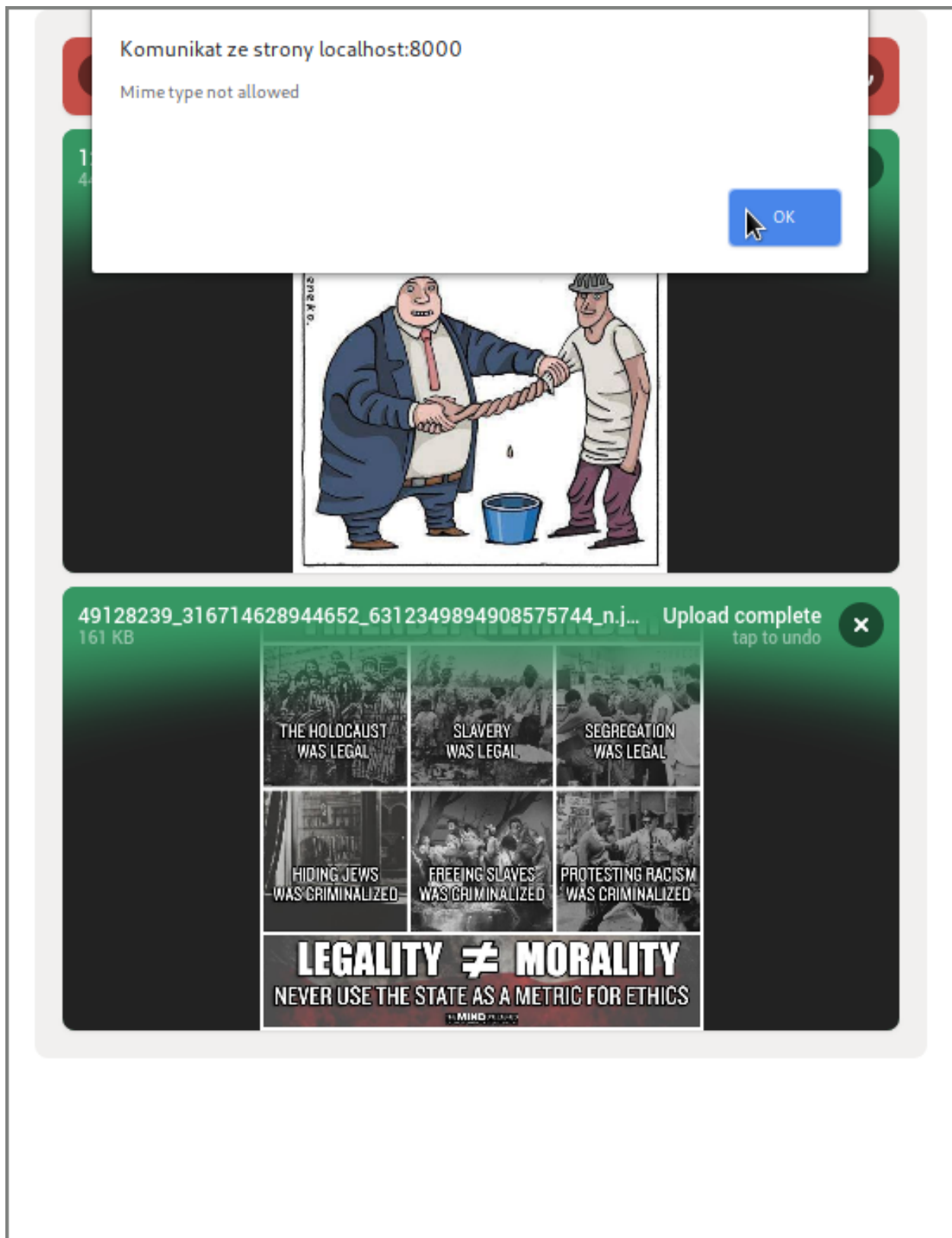
http://localhost:8000/public/download/13bc7d6ab012400512_

Close



49128239_316714628944652_6312349894908575744_... Upload complete
161 KB tap to undo



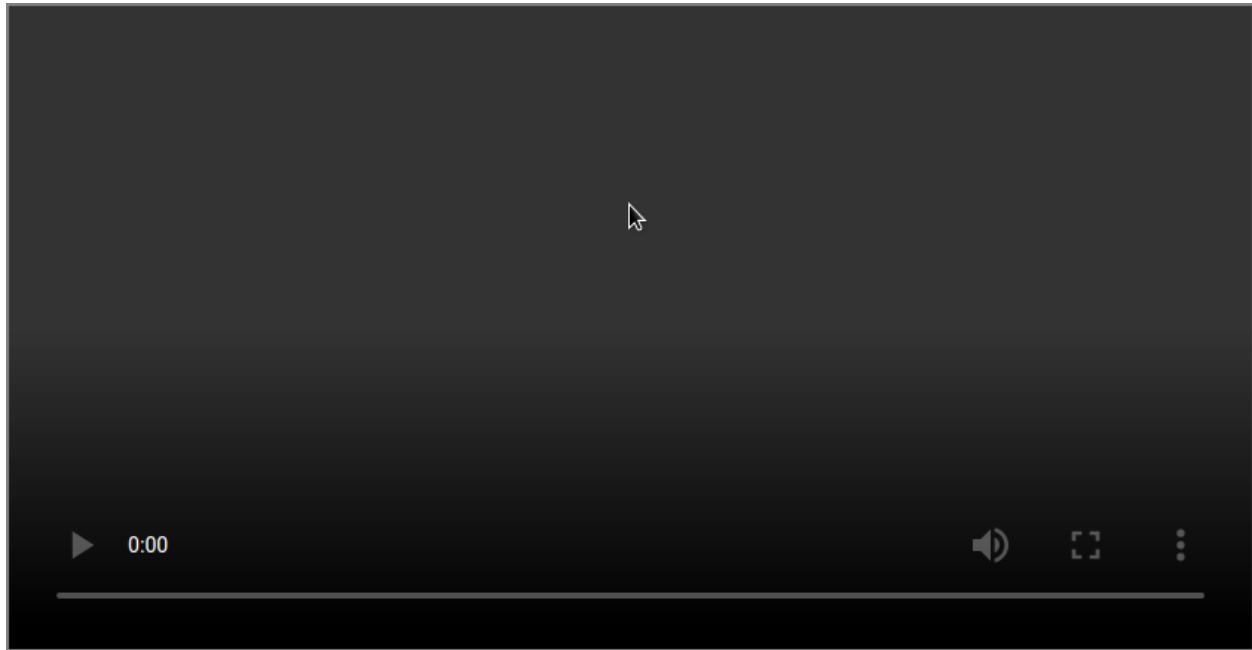




10.2.3 Video watching

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 11

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) x 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) x 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) x python3 ./__init__.py --config ../configuration.yaml.dist restore my_logs v58
{"status": "OK"}
+ bahub git:(master) x python3 ./__init__.py --config ../configuration.yaml.dist restore my_logs latest
{"status": "OK"}
```

11.1 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.

11.1.1 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.

11.1.2 Application configuration

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

```
#
# Access information about the storage server (File Repository), the URL, token
# In "backups" section you should use name of an access as a reference in field
# ↪ "access"
#
accesses:
  backup_one:
    url: http://localhost:8000
    token: test-token-full-permissions

#
# Allows to define an encryption and identify it with name
# In "backups" section you should use just the name as reference in "encryption" key_
# ↪ of a backup
#
encryption:

  #
  # Example of AES-256-CBC encryption (uses OpenSSL)
  #
  # Decryption command example:
  #   cat 0ad166b4cfbackup.tar-v5.gz| openssl enc -aes-256-cbc -pbkdf2 -d -pass_
  # ↪ pass:some-encryption-key-here > ./backup.tar.gz
  #
  enc1:
    passphrase: some-encryption-key-here
    method: aes-256-cbc      # possible values: aes-256-cbc, aes-256-ecb, aes-128-
  # ↪ cbc, aes-128-ecb, des-ecb
    #encrypt_cmd: openssl enc -%method% -pass pass:%pass%
    #decrypt_cmd: openssl enc -d -%method% -pass pass:%pass%

  none:
    passphrase: ""
    method: ""
```

(continues on next page)

(continued from previous page)

```

#
# Backups - list of backups, each backup = single archive = single collection.
#           To backup invoke the shell command with a backup name eg. bahub backup_
↪www_uploads
#           To restore invoke eg. bahub restore www_uploads latest
#
backups:

#
# Online / Hot-backup of docker container example
#
# Copies data from running container.
# NOTICE: May be dangerous to your data, please make sure you know what you do.
#           In most cases please choose "docker_volumes" method for safe, offline_
↪backup.
#
docker_hot_volumes_example:
  type: docker_hot_volumes
  container: "test_www"
  access: backup_one
  encryption: none
  collection_id: "${COLLECTION_ID}"
  paths:
    - /var/www
    - /var/log

# optional
#tar_pack_cmd: "tar -czf %stdin% %paths%"
#tar_unpack_cmd: "tar -xzf %stdin% %target%"
#docker_bin: "sudo docker"

#
# Offline backup of docker container
#
# Stops the container, copies the data, then starts it again
# Fully safe method of backup for all types of services. Makes a downtime, but_
↪guarantees a non-corrupted
# backup data.
#
www_docker_offline:
  type: docker_volumes
  container: "test_www"
  access: backup_one
  encryption: encl
  collection_id: "${COLLECTION_ID}"
  paths:
    - /etc
    - /var/lib/mysql
    - /var/log

# optional
docker_bin: "sudo docker"
tar_pack_cmd: "tar -czf %stdin% %paths%"
tar_unpack_cmd: "tar -xzf %stdin% %target%"
temp_image_name: "alpine:3.9"

```

(continues on next page)

(continued from previous page)

```

temp_image_cmd: "apk add --update xz bzip2 && sleep 3600"

#
# MySQL online backup using mysqldump
#
# Can backup a single database or all databases from any MySQL server, local,
↳ in-docker or remote.
# Notice: Restoring backups is only possible for SINGLE DATABASES
#
mysql_native_single_database:
  type: mysql
  host: "${DB_HOST}"
  port: 3306
  user: root
  password: root
  database: "${DB_DATABASE}"
  access: backup_one
  encryption: none
  collection_id: "${COLLECTION_ID}"

#
# MySQL online backup using mysqldump (DOCKER)
#
# Can backup a single database or all databases from any MySQL server, local,
↳ in-docker or remote.
# Notice: Restoring backups is only possible for SINGLE DATABASES
#
mysql_docker_single_database:
  type: mysql
  host: localhost
  port: 3306
  user: root
  password: root
  database: "${DB_DATABASE}"
  container: "test_mysql" # this one is required to use a docker container
  access: backup_one
  encryption: none
  collection_id: "${COLLECTION_ID}"

# optional:
#tar_pack_cmd: "tar -czf %stdin% %paths%"
#tar_unpack_cmd: "tar -xzf %stdin% %target%"

#
# Docker command output online backup
#
# Executes command in a Docker container and captures output
#
docker_command_output:
  type: docker_output
  container: "test_www"
  command: "cat /bin/sh"
  restore_command: "cat - > /tmp/sh.restored"
  access: backup_one
  encryption: encl
  collection_id: "${COLLECTION_ID}"

```

(continues on next page)

(continued from previous page)

```

# optional
#docker_bin: "sudo docker"

#
# Execute a local command
#
# Executes command on this machine/environment on local shell and captures
↪output
#
local_command_output:
  type: command_output
  command: "cat /bin/bash"
  restore_command: "cat - > /tmp/bash.restored"
  access: backup_one
  encryption: enc1
  collection_id: "${COLLECTION_ID}"

#
# Local file/directory backup
#
# Packs a local directory or file
#
some_local_dir:
  type: directory
  paths:
    - /tmp/test
  access: backup_one
  encryption: enc1
  collection_id: "${COLLECTION_ID}"

# optional
#tar_pack_cmd: "tar -czf %stdin% %paths%"
#tar_unpack_cmd: "tar -xzf %stdin% %target%"

#
# 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 possible
#     - stop-on-first-error: Restore until first error, then stay as it is
#
recoveries:
  default:
    policy: restore-whats-possible
    definitions: all

  some_selected_only:
    policy: stop-on-first-error
    definitions:
      - some_local_dir
      - local_command_output

#
# Optional: Error handlers, very helpful to track backup errors

```

(continues on next page)

(continued from previous page)

```
#      Be careful! Your shell commands for backup/restore may fail some time, it
→ 's worth to have a notification.
#      You can set up a free account on sentry.io, install a Sentry instance on
→ your server, or use eg. Mattermost/Slack notifications
#
#error_handlers:
#   remote_sentry:
#       type: sentry
#       url: "${SENTRY_IO}"

#
# Optional: Notifications. Can notify when backup starts, ends, there is an error etc.
#
#notifiers:
#   mattermost:
#       type: slack
#       url: "https://xxxxx"
```

11.2 Basic usage

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

11.2.1 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'}
```

11.2.2 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"
  }
}
```

11.2.3 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"}
```

11.2.4 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 possible
#     - 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
```

11.2.5 Making a snapshot of multiple services at once

Snapshot works exactly in the same way as **recovery from disaster**, 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"
  ]
}
```

11.3 Monitoring errors with Sentry

ReadWriteException bahub.bahubapp.handler in `_execute_command`

● Cannot write to process, broken pipe occurred, probably a tar process died. dd: nie udało się otworzyć '/tmp/dpa/213dsaa...

✓ Resolve ▼ ⌂ Ignore ▼ ★ 🗑️ ▼ ● Share ▼

Details Comments 0 User Feedback 0 Tags Events Merged Similar Issues

Event [4u32685d24c0a213dsaads649788](#)
May 1, 2019 7:16:27 PM UTC | [JSON \(21.7 KB\)](#)

TAGS

handled no level error mechanism excepthook server_name test-riotkit

EXCEPTION (most recent call first)

ReadWriteException

```
Cannot write to process, broken pipe occurred, probably a tar process died. dd: nie udało się otworzyć '/tmp/dpa/13213wac
takiego pliku ani katalogu
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
error writing output file
```

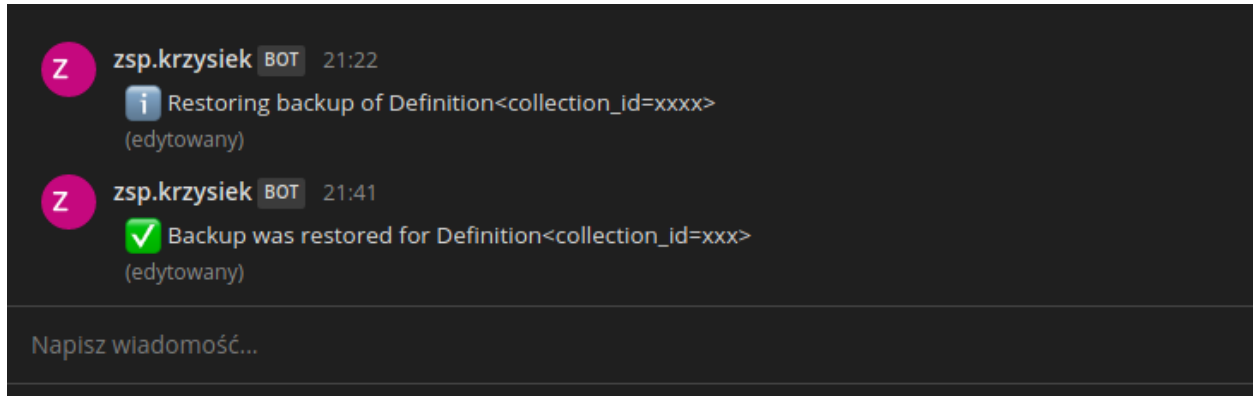
Bahub uses shell commands to take some data, pack it and encrypt. What if any of those commands will fail? What if there are no enough permissions? What if the directory does not exist? All of those are good reasons to have set up a monitoring.

Almost each application failure can be caught 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"
```

11.4 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"
```

11.5 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.

11.6 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:
  backup:
    image: wolnosciowiec/file-repository:bahub
    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=111111-2222-3333-4444-5555555555555555
      - BACKUPS_REDIS_COLLECTION_ID=12345678-cccc-bbb-aaa-1232313213123
      - COMPOSE_PROJECT_NAME=test_client

  redis:
```

(continues on next page)

(continued from previous page)

```

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**

11.7 Using bare metal

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

```

pip install bahub
bahub --help

```


CHAPTER 12

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.

12.1 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.

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

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
```

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 14

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 PHP7, 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