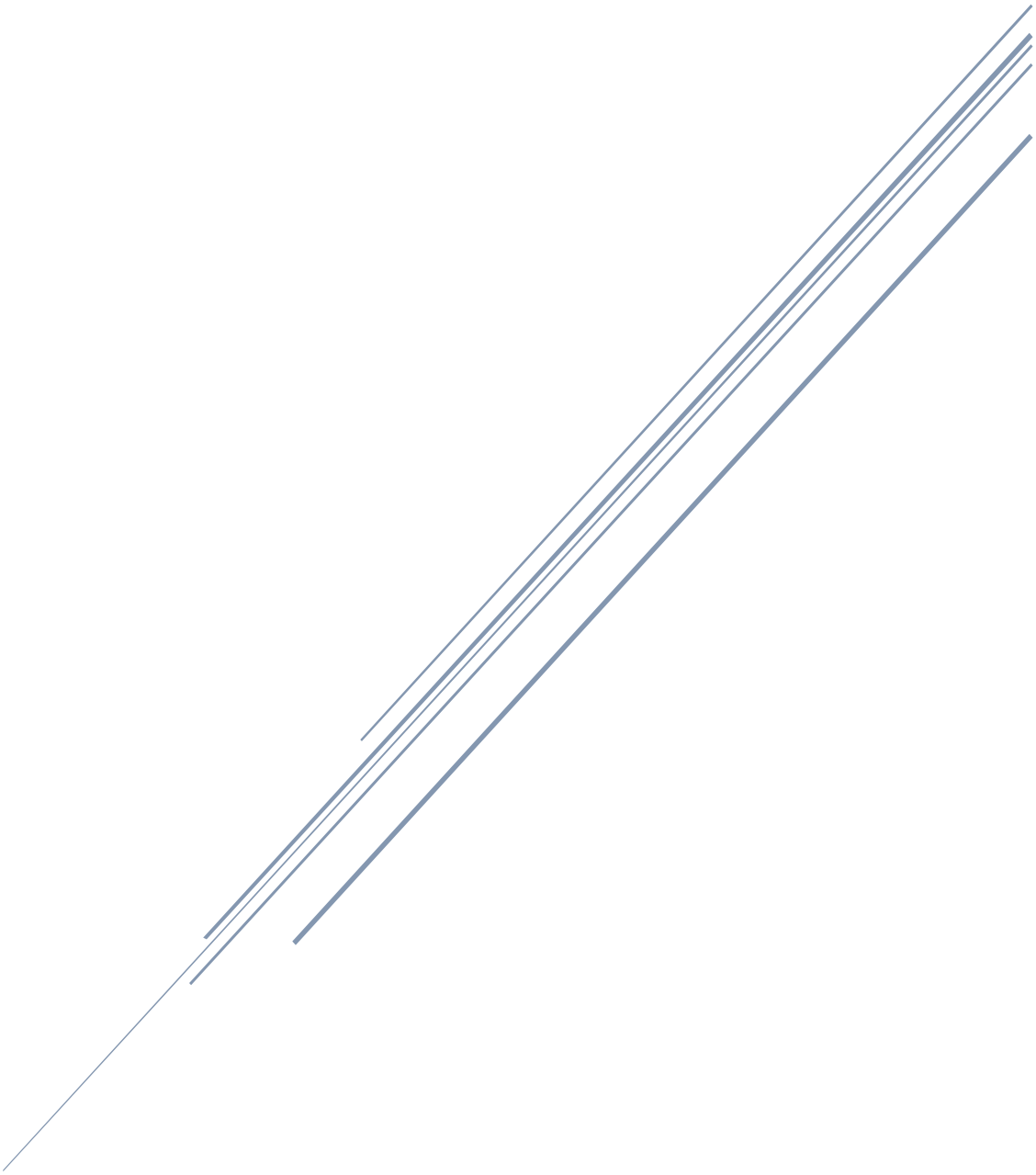
DUCHNETSERVER: A WEB SERVER FOR STORING FILE METADATA

Joel Aumedes and Pau Escolà



Universitat de Lleida
Distributed Computing

Index

Introduction

DuchnetServer is a web service designed to be used together with a series of Duchnet nodes, to manage metadata of files in the network. This server allows information to be stored in a data base and not in memory, so that they can be retrieved. It has been built using Spring Boot in Java.

The source code for the server can be found in:

https://github.com/Galahad3x/DuchnetServer

The source code for the nodes can be found in:

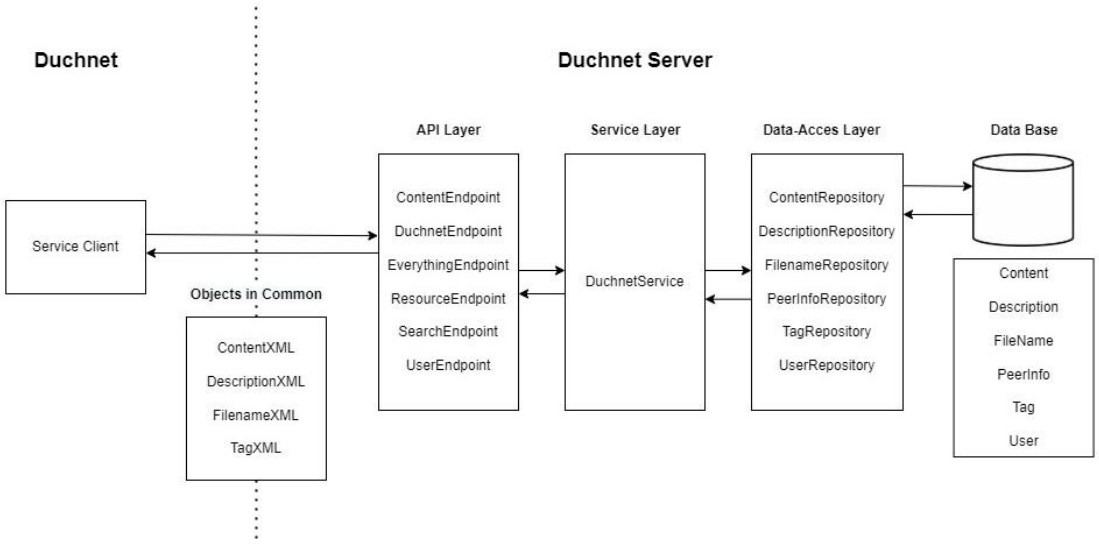
https://github.com/Galahad3x/Duchnet

Keep in mind that since the code can fail during the demonstration and quick fixes may be needed, which may cause the code sent in the deliverable activity to be deprecated. When wanting to execute a Node or the Server, make sure to have the latest release downloaded from the GitHub repositories.

How to use

The instructions for the DuchnetServer are in the DuchnetServer project's README, and the instructions for the Duchnet node are in the Duchnet project's README.

Architecture



DuchnetServer follows a 3-layered architecture in which the components are divided in these 3 different layers: API layer, Service layer and Data access layer.

API Layer

The API layer is built with all the endpoints of the service, in the *endpoints* package. These endpoints return data using the objects in the *common* package, which is present in the server and in the client, so that the client can rebuild the data using these classes. The definition of all the functions is explained later in the API Documentation section.

Service layer

DuchnetService is the only class in the service layer. It executes all the logic of the server, and ties together the endpoints and the data using the repository objects.

Data access layer

The data access layer is built in the *repository* package. These interfaces use class hierarchy with pre-defined ones so that us developers can access the data while knowing little to no SQL. These repositories, along with the models in the *models* package, are used to access a postgres database.

API Documentation

Before starting to code the server and all the functions, we defined what URI's would be available, which service would they provide and how would they respond to different cases, such as a correct use, or a use missing some data, etcetera.

This is a simple table of all the URI's available in the now deprecated first version of the service.

/v1	/resources/{resource-plural}	/	/contents/{hash}	/contents/{hash}/{resource-plural}	/contents/search/{resource-plural}
GET	Get all of that resource	Get everything	Get all resources related to that hash	Get all specified resources related to that hash	Search for contents with their resources that have the {resource-plural} in the request body
POST	Nothing	Nothing	Nothing	Post a new specified resource to that hash. If it doesn't exist, create new content.	Nothing
PUT	Nothing	Nothing	Nothing	Post a new specified resource to that hash. If it doesn't exist don't do anything.	Nothing
DELETE	Delete all that resource	Delete everything	Delete all resources related to that hash	Delete all specified resources related to that hash	Delete resources that have the resource in the body

This first version did not have users, which were added in the second and final version. To authenticate in the service, the request must have a "username" header with the username, and a "password" header with the password.

Below is the table for the second version and an individual explanation of each function.

/v2	/resources/{resource-plural}	/	/contents/{hash}	/contents/{hash}/{resource-plural}	/contents/search/{resource-plural}	/auth
GET	Get all of that resource	Get everything	Get all resources related to that hash	Get all specified resources related to that hash	Search for contents with their resources that have the {resource-plural} in the request body	Nothing
POST	Nothing	Nothing	Nothing	Post a new specified resource to that hash. If it doesn't exist, create new content.	Nothing	Create user
PUT	Nothing	Nothing	Nothing	Post a new specified resource to that hash. If it doesn't exist don't do anything.	Nothing	If password is correct, change password
DELETE	Delete all of that resource owned by the user	Delete everything owned by the user but not the user	Delete all resources related to that hash owned by the user	Delete all specified resources related to that hash owned by the user	Delete resources owned by the user that have the resource in the body	Delete user

Resources endpoint

GET /v2/resources/{resource-plural}

Get all resources of a certain type

Get all resources of a certain accepted type (descriptions, tags, or filenames), owned or not by the user, in the form of List<DescriptionXML>, List<TagXML> or List<FilenameXML>. If {resource-plural} is blank, all the data is returned in the form of List<ContentXML>.

Status codes

- 200 (OK): Correct use case
- 405 (METHOD NOT ALLOWED): Resource type is not correct

DELETE /v2/resources/{resource-plural}

Get all resources of a certain type owned by the user

Delete all resources of a certain accepted type (descriptions, tags, or filenames), owned by the user. If {resource-plural} is blank, all the user data is deleted, but not the user itself.

Status codes

- 200 (OK): Correct use case
- 403 (FORBIDDEN): Authentication failed
- 405 (METHOD NOT ALLOWED): Resource type is not correct

Everything endpoint

GET /v2/

Get everything

Get all resources in the form of List<ContentXML>.

Status codes

- 200 (OK): Correct use case

DELETE /v2/

Delete everything owned by the user

Delete all resources owned by the user, but not the user itself.

Status codes

- 200 (OK): Correct use case
- 403 (FORBIDDEN): Authentication failed

Content endpoint

GET /v2/contents/{hash}

Get all resources of a certain hash

Get all resources of a certain hash, owned or not by the user, in the form of a ContentXML object.

Status codes

- 200 (OK): Correct use case
- 204 (NO CONTENT): The hash is not in the database

DELETE /v2/resources/{resource-plural}

Delete all resources of a certain hash owned by the user

Delete all resources of a certain hash, owned by the user. If the user does not own the content, it is not deleted.

Status codes

- 200 (OK): Correct use case
- 403 (FORBIDDEN): Authentication failed, or user does not own the content.
- 405 (METHOD NOT ALLOWED): Resource type is not correct

GET /v2/contents/{hash}/{resource-plural}

Get all certain resources of a certain hash

Get all certain accepted resources (descriptions, tags, filenames, or peers) of a certain hash, owned or not by the user, in the form of a List<{type}XML> object.

Status codes

- 200 (OK): Correct use case
- 204 (NO CONTENT): The hash is not in the database
- 405 (METHOD NOT ALLOWED): The resource type is incorrect

POST /v2/contents/{hash}/{resource-plural}

Post a resource to a certain hash

Post an accepted type of resource to a hash in the database. If the hash does not exist, it is created.

Status codes

- 201 (CREATED): Correct use case
- 400 (BAD REQUEST): Request body is missing (blank resource)
- 403 (FORBIDDEN): Authentication failed
- 405 (METHOD NOT ALLOWED): The resource type is incorrect

PUT /v2/contents/{hash}/{resource-plural}

Post a resource to a certain hash

Post an accepted type of resource to a hash in the database. If the hash does not exist, it is not created.

Status codes

- 201 (CREATED): Correct use case
- 400 (BAD REQUEST): Request body is missing (blank resource)
- 403 (FORBIDDEN): Authentication failed
- 404 (NOT FOUND): Hash doesn't exist in the database
- 405 (METHOD NOT ALLOWED): The resource type is incorrect

DELETE /v2/contents/{hash}/{resource-plural}

Delete all certain resources of a certain hash owned by the user

Delete all certain accepted resources of a certain hash, owned by the user.

Status codes

- 200 (OK): Correct use case
- 403 (FORBIDDEN): Authentication failed, or user does not own the content.
- 404 (NOT FOUND): Hash does not exist in the database
- 405 (METHOD NOT ALLOWED): Resource type is not correct

Search endpoint

GET /v2/contents/search/{resource-plural}

Search for contents using a certain resource

Search for contents using partial search on a certain resource. All matching contents will be returned, not only those owned by the user. They will be return as a List<ContentXML>. If the search is blank, all contents will match.

Status codes

- 200 (OK): Correct use case
- 204 (NO CONTENT): The search resulted in no results
- 405 (METHOD NOT ALLOWED): The resource type is incorrect

DELETE /v2/contents/{hash}/{resource-plural}

Delete all contents with a certain resource

Search for contents using partial search on a certain resource, then delete only those owned by the user

Status codes

- 200 (OK): Correct use case
- 400 (BAD REQUEST): The search resulted in no results
- 403 (FORBIDDEN): Authentication failed
- 405 (METHOD NOT ALLOWED): The resource type is incorrect

User endpoint

POST /v2/auth

Register in the service

Register the user in the headers in the service.

Status codes

- 201 (CREATED): Correct use case
- 403 (FORBIDDEN): Username is already taken

PUT /v2/auth

Change password in the service

Change the password in the service. The user must still authenticate and set the password in a new header “new-password”.

Status codes

- 200 (OK): Correct use case
- 403 (FORBIDDEN): Authentication failed, or username does not exist.

DELETE /v2/auth

Delete user in the service

Delete a user in the service.

Status codes

- 200 (OK): Correct use case
- 403 (FORBIDDEN): Authentication failed, or username does not exist.