

P12 : aider la communauté en tant que Développeur
d'Application Java

**Développement d'un outil de gestion de
métadonnées d'œuvres, d'artistes
et d'un drive de documents**

Contexte du projet

Le M.U.R. (Modulable Urbain Réactif) est une association culturelle type 1901 promouvant le street art, déployée au national via un réseau d'antennes régionales, notamment dans la ville dans laquelle je réside : Grenoble.

Jusqu'en 2019, ses membres disposaient d'un site dédié, mis à jour, et constituant une vitrine de leurs activités artistiques et événementielles, et d'une boutique en ligne.

Depuis 2019, suite à des aléas liés à la vie associative, le site web et la boutique ne peuvent plus être maintenus (accès aux DB impossible), mais l'association dispose d'un back-up d'une partie des ressources (articles Wordpress).

L'association a besoin d'aide pour redéployer un nouveau site Internet et m'a fait part d'une demande de développements spécifiques pour laquelle je me suis porté volontaire.

Sommaire

1. Besoins, spécifications et choix techniques
2. Démonstration du logiciel
3. Structure du projet
4. Quelques détails d'implémentation
5. Pistes d'amélioration

1. Besoins, spécifications et choix techniques

1. Besoins, spécifications et choix techniques

2 axes de travail :

- Réalisé par l'association : déployer un nouveau site Internet :
 - basé sur le système de gestion de contenu (CMS) Wordpress (bien maîtrisé par l'association),
 - avec l'extension de e-commerce **WooCommerce**.
- Réalisé par moi-même : développer un système open-source en **Java** :
 - pour une gestion spécifique aux besoins du bureau de l'association,
 - en mesure d'échanger avec le shop au travers d'une **interface REST**,
 - déployé sur les serveurs de l'association.

1. Besoins, spécifications et choix techniques

A noter :

- Le site Wordpress est dédié aux visiteurs.
- L'application Java de métadonnées et de documents est exclusivement dédiée au bureau de l'association.
- Les ressources (nom de domaine, hébergement et DB) seront mis à disposition par l'association.
- Je m'engage à assurer le support, ainsi que, si nécessaire, le déploiement de nouvelles mises à jour du programme sur Docker Hub.

1. Besoins, spécifications et choix techniques

Fonctionnalités du logiciel :

- Une base de métadonnées d'artistes et d'œuvres comprenant :
 - Micro dashboard de l'état instantané du shop,
 - Ajout, modification, consultation, suppression, tagging et recherche d'œuvres et d'artistes par critères de métadonnées,
 - Synchronisation des œuvres du shop et de la base de métadonnées (mini batch).
- Un drive de documents (texte, PDF et tableur) embarqué dans l'application
 - Sans DB, ni abonnement
 - Sécurisation des documents,
 - Historique et hashing (SHA1) des modifications,
 - Utilisation d'un repository Git privé pour persister les documents via l'API REST de GitHub.
- Croiser les différentes informations entre :
 - Le shop WooCommerce,
 - Les métadonnées d'artistes, d'œuvres et de documents.
- Une interface de configuration en ligne

1. Besoins, spécifications et choix techniques

- **Architecture à microservices :**

- Un client Java pour l'API REST du shop WooCommerce
- Un client Java pour l'API REST de GitHub.

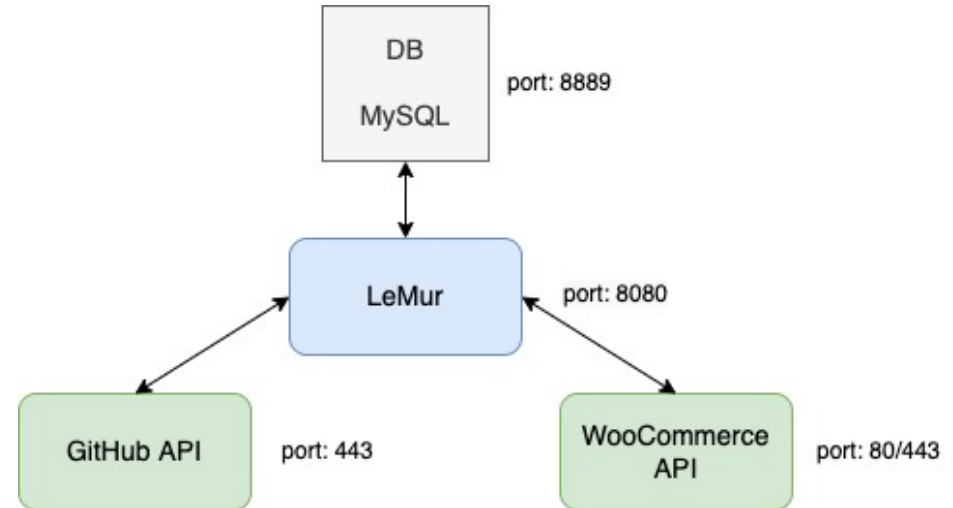
- **Patron logiciel MVC**

- **Stack technique :**

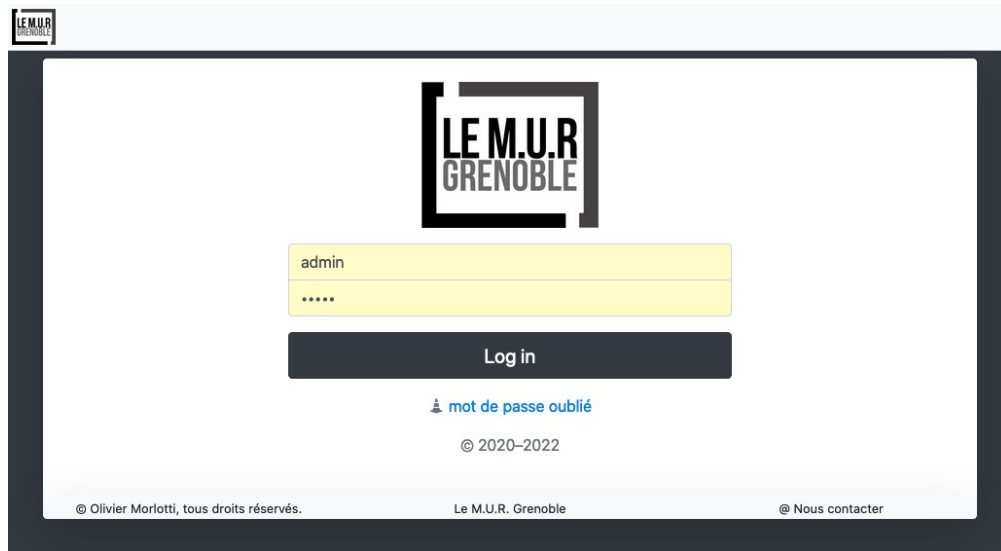
- Partie *Back* :
 - Java 12, Maven 3
 - Spring Boot, Spring Security (avec JWT), Spring Feign et Hibernate
 - Pebble (moteur de templates Twig pour Java)
 - MySQL/MariaDB
- Partie *Front* :
 - HTML5, CSS6
 - Twitter Bootstrap 4
 - JavaScript ES6 et JQuery
 - DataTables.js (bibliothèque qui permet de faire des tables HTML en consommant une API REST)

- **Déploiement :**

- Code source sous GitHub
- Image déployée sur Docker Hub à chaque commit via l'intégration continue de GitHub : <https://hub.docker.com/repository/docker/morlotti/le-mur>



2. Démonstration du logiciel



The screenshot displays the login interface for LEM.U.R. Grenoble. At the top left, there is a small logo. The main content area features a large, stylized logo for 'LEM.U.R. GRENOBLE'. Below the logo, there are two yellow input fields: the first contains the text 'admin', and the second contains six dots representing a password. A dark grey 'Log in' button is positioned below the password field. Underneath the button is a blue link with a key icon and the text 'mot de passe oublié'. Below this link is the copyright notice '© 2020-2022'. The footer of the page contains three items: '© Olivier Moriotti, tous droits réservés.', 'Le M.U.R. Grenoble', and '@ Nous contacter'.

LEM.U.R. GRENOBLE

admin

.....

Log in

[🔑 mot de passe oublié](#)

© 2020-2022

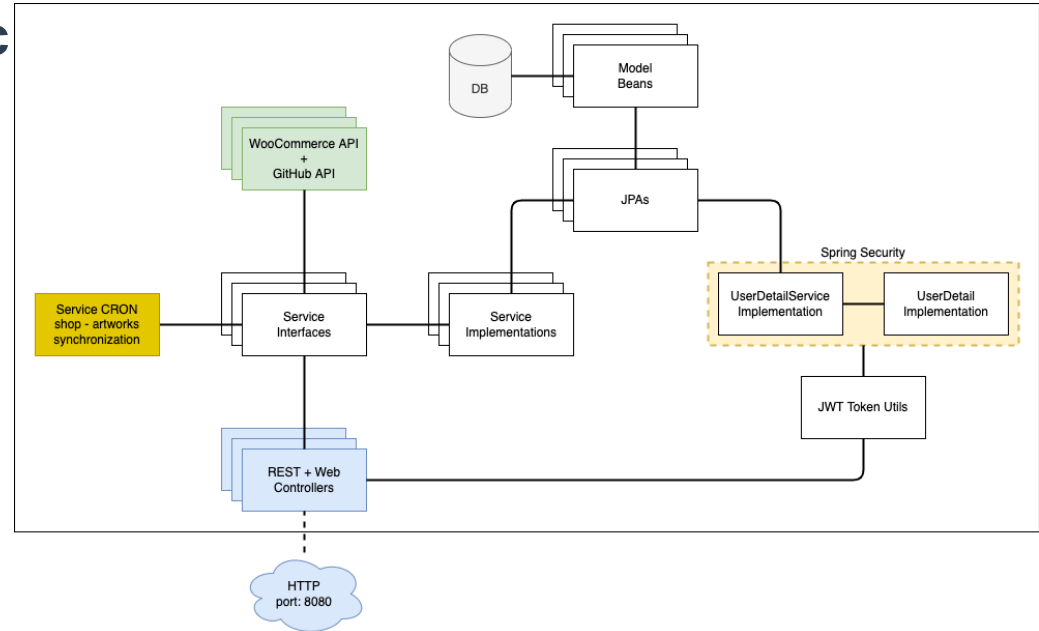
© Olivier Moriotti, tous droits réservés. Le M.U.R. Grenoble @ Nous contacter

3. Structure du projet

3. Structure du projet

Structure conventionnelle d'application Spring Boot avec

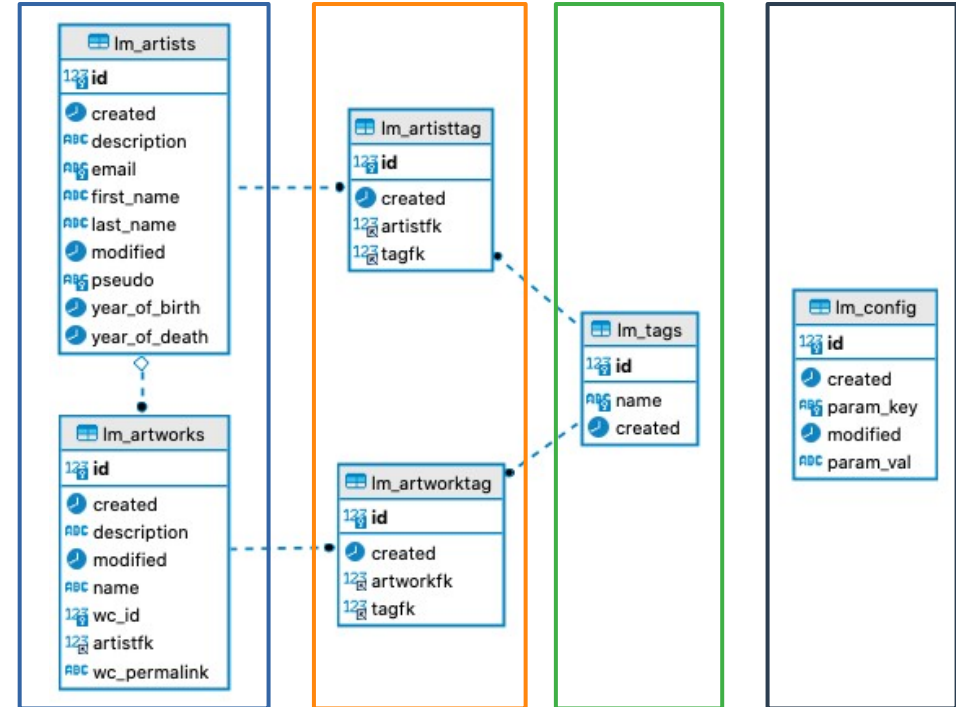
- Modèles de données Hibernate
- Repositories (JPA)
- Proxies Feign
- Services de type CRON
- Services
- Controllers REST (pour JS / Ajax)
- Controllers Web (pour l'interface)



3. Structure du projet

Modèle Physique de Données :

- Tables d'artistes et d'oeuvres d'art
- Tables d'association ManyToMany
- Table de tags
- Table de configuration de l'application



Remarque : les documents sont entièrement gérés via l'API REST de GitHub (pas de table).
Une feature future pourrait être de rajouter une table liant les tags aux chemins de fichiers.

3. Structure du projet : descr. des modèles Hibernate

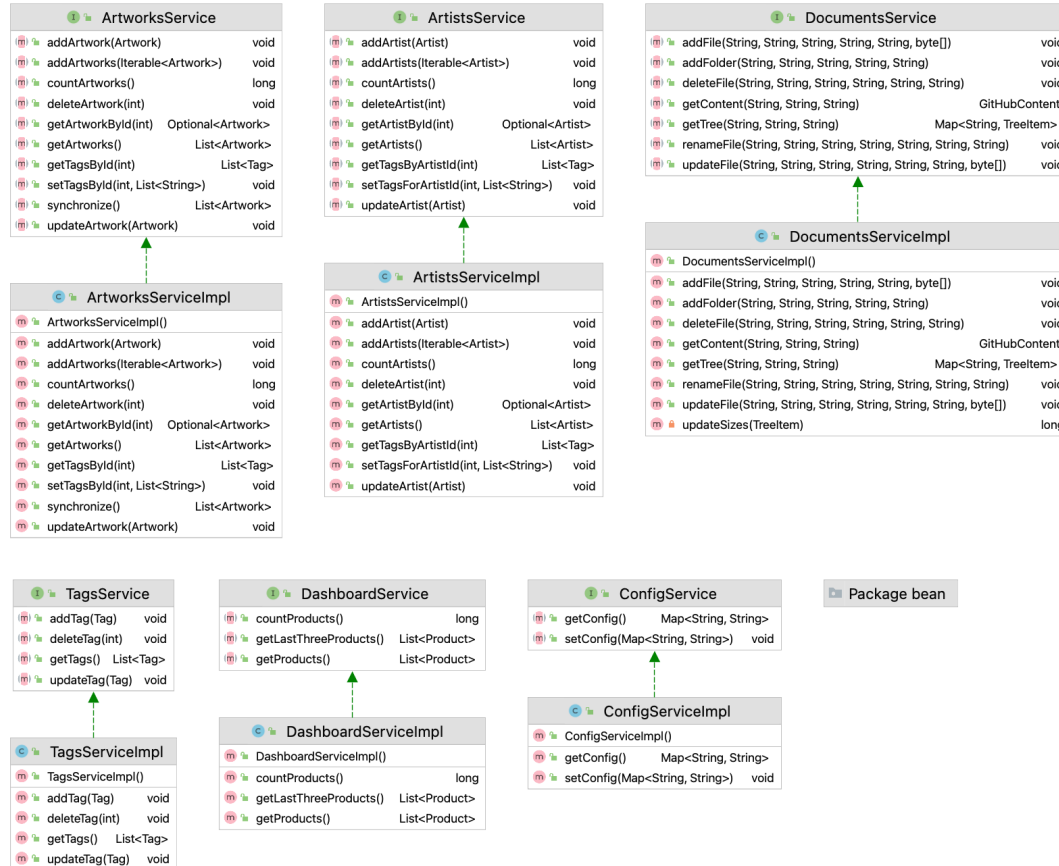
MPD

Artist	Artwork
<pre>Artist() Artist(Integer, String, String, String, String, Integer, Integer, String, LocalDate, LocalDate, Set<ArtworkTag>) getArtistTags() Set<ArtworkTag> getCreated() LocalDate getDescription() String getEmail() String getFirstName() String getId() Integer getLastName() String getModified() LocalDate getPseudo() String getTagString() String getYearOfBirth() Integer getYearOfDeath() Integer setArtistTags(Set<ArtworkTag>) void setCreated(LocalDate) void setDescription(String) void setEmail(String) void setFirstName(String) void setId(Integer) void setLastName(String) void setModified(LocalDate) void setPseudo(String) void setYearOfBirth(Integer) void setYearOfDeath(Integer) void toString() String</pre>	<pre>Artwork() Artwork(Integer, Integer, String, String, Artist, String, LocalDate, LocalDate, Set<ArtworkTag>) getArtist() Artist getArtistId() Integer getArtistPseudo() String getArtworkTag() Set<ArtworkTag> getCreated() LocalDate getDescription() String getId() Integer getModified() LocalDate getName() String getTagString() String getWcId() Integer getWcPermalink() String setArtist(Artist) void setArtistId(Integer) void setArtworkTag(Set<ArtworkTag>) void setCreated(LocalDate) void setDescription(String) void setId(Integer) void setModified(LocalDate) void setName(String) void setWcId(Integer) void setWcPermalink(String) void toString() String</pre>
Config	Tag
<pre>Config() Config(Integer, String, String, LocalDate) getId() Integer getKey() String getModified() LocalDate getVal() String setId(Integer) void setKey(String) void setModified(LocalDate) void setVal(String) void toString() String</pre>	<pre>Tag() Tag(Integer, String, LocalDate) getCreated() LocalDate getId() Integer getName() String setCreated(LocalDate) void setId(Integer) void setName(String) void toString() String</pre>
ArtistTag	ArtworkTag
<pre>ArtistTag() ArtistTag(Integer, Artist, Tag, LocalDate) getArtist() Artist getArtistPseudo() String getCreated() LocalDate getId() Integer getTag() Tag getTagName() String setArtist(Artist) void setCreated(LocalDate) void setId(Integer) void setTag(Tag) void toString() String</pre>	<pre>ArtworkTag() ArtworkTag(Integer, Artwork, Tag, LocalDate) getArtwork() Artwork getArtworkName() String getCreated() LocalDate getId() Integer getTag() Tag getTagName() String setArtwork(Artwork) void setCreated(LocalDate) void setId(Integer) void setTag(Tag) void toString() String</pre>

Bridges

3. Structure du projet : description des services

Code
métier



3. Structure du projet : description des clients (Open Feign) WooCommerce et GitHub

I 📁 WooCommerce	
m 📁 countOrders()	long
m 📁 countProducts()	long
m 📁 getOrders()	List<Object>
m 📁 getProducts()	List<Product>

C 📁 WooCommerceImpl	
m 📁 WooCommerceImpl()	
m 📁 countOrders()	long
m 📁 countProducts()	long
m 📁 getOrders()	List<Object>
m 📁 getProducts()	List<Product>

Services qui
s'appuient sur
des proxies
Feign

I 📁 GitHub	
m 📁 addFile(String, String, String, String, String, byte[])	void
m 📁 addFolder(String, String, String, String, String)	void
m 📁 deleteFile(String, String, String, String, String, String)	void
m 📁 fileTree(String, String, String)	GitHubTree
m 📁 getContent(String, String, String)	GitHubContent
m 📁 me(String)	GitHubUser
m 📁 renameFile(String, String, String, String, String, String, String)	void
m 📁 updateFile(String, String, String, String, String, String, byte[])	void
m 📁 versions(String, String, String)	List<GitHubVersion>

C 📁 GitHubImpl	
m 📁 GitHubImpl()	
m 📁 addFile(String, String, String, String, String, byte[])	void
m 📁 addFolder(String, String, String, String, String)	void
m 📁 deleteFile(String, String, String, String, String, String)	void
m 📁 fileTree(String, String, String)	GitHubTree
m 📁 getContent(String, String, String)	GitHubContent
m 📁 me(String)	GitHubUser
m 📁 renameFile(String, String, String, String, String, String, String)	void
m 📁 toHex(byte[])	String
m 📁 updateFile(String, String, String, String, String, String, byte[])	void
m 📁 versions(String, String, String)	List<GitHubVersion>

4. Quelques détails d'implémentation

4. Quelques détails d'implémentation API WooCommerce pour Le M.U.R.

- **Récupération de produits et d'achats :**
 - <https://woocommerce.github.io/woocommerce-rest-api-docs/#orders>
 - <https://woocommerce.github.io/woocommerce-rest-api-docs/#products>
 - Beans dans xyz.morlotti.lemur.woocommerce.bean
- **Authentification (avec https et http *(pratique pour développer)*) :**
 - OAuth 1.0a "one-legged" authentication
 - <https://woocommerce.github.io/woocommerce-rest-api-docs/#authentication-over-http>
 - *Implémentation décrite dans les slides suivantes*

4. Quelques détails d'implémentation WooCommerce Authentication

- **OAuth 1.0a "one-legged" authentication**
 - Par consumerKey et consumerSecret, définis dans la partie admin du site WooCommerce
 - A chaque requête, injection d'un header HTML contenant une signature
 - Solide/Secure pour https comme http
 - Hachage HmacSHA1 (« cryptage » non réversible) + nonce aléatoire pour générer une signature sécurisée
 - **La classe...**
 - `xyz.morlotti.lemur.clients.woocommerce.proxy.WooCommerceClientConfiguration`
- ... est utilisée comme classe de configuration du proxy...**
- `xyz.morlotti.lemur.clients.woocommerce.proxy.WooCommerceClient`

4. Quelques détails d'implémentation

Description de OAuth 1.0 « one legged »

- **Génération d'une signature HmacSHA1 (mot de passe = consumerKey) à partir d'une string URL-encoded des informations suivantes :**
 - Méthode HTTP (GET, POST, ...)
 - De l'URL de la ressource REST
 - Nonce aléatoire
 - Un timestamp
- **Injection, dans les headers HTTP, de la signature + toutes les informations nécessaires à son re-calcul SAUF la consumerKey**
- **WooCommerce se charge de la validation de la nouvelle signature à chaque requête à l'API REST**
- **<https://github.com/OMorlotti/P12/blob/master/src/main/java/xyz/morlotti/lemur/clients/woocommerce/proxy/WooCommerceClientConfiguration.java>**

4. Quelques détails d'implémentation

Treeltem : structure arborescente de fichiers 1/2

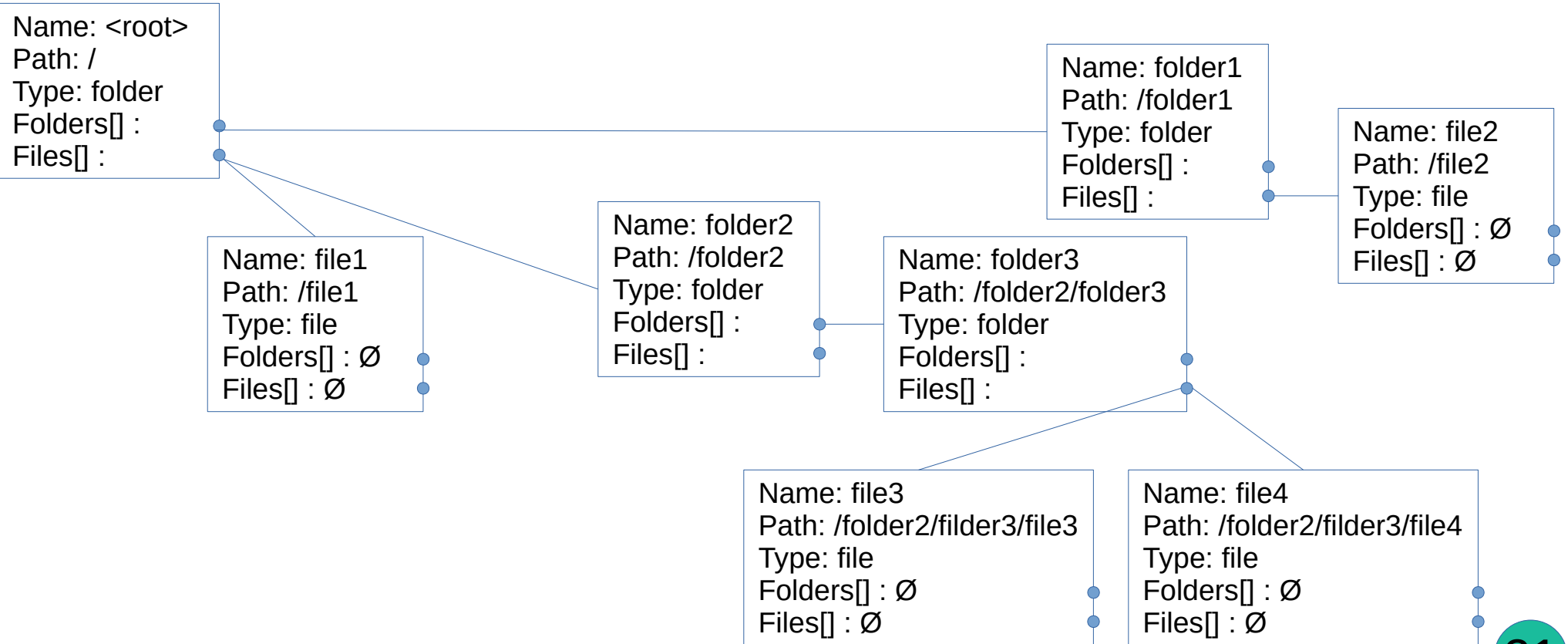
Treeltem	
<code>Treeltem()</code>	
<code>Treeltem(int, Type, String, String, long, String, String)</code>	
<code>getAbbreviatedHash()</code>	String
<code>getFiles()</code>	Map<String, Treeltem>
<code>getFolders()</code>	Map<String, Treeltem>
<code>getHash()</code>	String
<code>getId()</code>	int
<code>getName()</code>	String
<code>getPath()</code>	String
<code>getSize()</code>	long
<code>getType()</code>	Type
<code>setAbbreviatedHash(String)</code>	void
<code>setHash(String)</code>	void
<code>setId(int)</code>	void
<code>setName(String)</code>	void
<code>setPath(String)</code>	void
<code>setSize(long)</code>	void
<code>setType(Type)</code>	void
<code>toString()</code>	String

- La structure Treeltem permet de représenter un fichier ou un dossier.
- Dans le cas d'un dossier, les maps retournées par **getFiles()** et **getFolders()** contiennent respectivement les sous-fichiers et les sous-dossiers.
- Les clés des maps représentent les fichiers ou les dossiers.

Remarque : l'arborescence de structure Treeltem sera construite à partir de la méthode `fileTree()` du client GitHub (qui retourne la liste des fichiers à plat).

4. Quelques détails d'implémentation

Treeltem : structure arborescente de fichiers 2/2



4. Quelques détails d'implémentation

Création de la structure arborescente de fichiers

- La méthode `fileTree()` du client GitHub retourne une liste de fichiers (avec leurs métadonnées) identifiés par un nom et un path (= liste à plat).
- Pour construire l'arbre des fichiers, on itère sur la liste à plat et on fabrique la structure récursive en maintenant un pointeur sur le dossier courant
 - D'abord pour les dossiers (afin d'avoir la structure d'arbre vide)
 - Ensuite sur les fichiers (afin de remplir la structure d'arbre)
- <https://github.com/OMorlotti/P12/blob/master/src/main/java/xyz/morlotti/lemur/service/DocumentsServiceImpl.java#L36>

5. Pistes d'amélioration

5. Pistes d'amélioration

- **Pouvoir taguer des fichiers et/ou des dossiers**
- **Travailler la généricité du système :**
 - Entités définies arbitrairement
- **Explorer GraphQL en alternative aux interfaces REST**