



Projet DataCity

Documentation Technique

Version 4



Cyril Moralès	morale_c
Marc Soufflet	souffl_m
Lionel Hamsou	hamsou_l
Cédric Merouani	meroua_c
Cyntia Marquis	marqui_c
Ryan Legasal	legasa_r
Guillaume de Jabrun	d-eima_g

Résumé

Ce document a pour objectif de fournir à de futurs développeurs une documentation technique détaillée des différentes parties du projet DataCity.

La première partie concerne la configuration des environnements nécessaires au bon fonctionnement du web service et le site web. La configuration décrite dans ce document est surtout liée au système d'exploitation Microsoft Windows car la procédure est plus complexe que sur un système basé sur Linux. Globalement, une installation des paquets via les scripts fournis suffisent à faire tourner le projet.

La seconde partie contient une documentation de l'API et décrit le fonctionnement du site web.

Cette version du document décrit le fonctionnement de la réécriture de la nouvelle version de l'API qui est plus allégée car elle se concentre sur les fonctions liées à la gestion des données.

Métadonnées

Description du document

Titre	[2015][TD4][FR] Documentation Technique
Date	04/01/2015
Auteurs	souffl_m, hamsou_l, meroua_c, morale_c, marchi_c, legasa_r, d-eima_g
Responsable	Cyril Moralès
Email	datacity_2015@labeip.epitech.eu
Sujet	Documentation Technique
Mots clés	Datacity, EIP, Epitech, Documentation Technique
Version du modèle	5.0

Tableau des révisions

Date	Auteur	Section(s)	Commentaire
04/01/2015	Cyril Moralès	Toutes	Revue du document
05/10/2014	Cyril Moralès	Résumé	Modification du résumé
14/09/2014	Cyril Moralès	API	Complétion de l'installation de l'environnement
13/06/2014	Cyril Moralès	Métas	Modification des métadonnées et styles
16/03/2014	Marc Soufflet	Toutes	Finition du document
09/03/2014	Cyril Moralès	Toutes	Document initial

Table des matières

1 - Configuration de l'environnement.....	1
1.1 - Importation des projets.....	1
1.2 - Téléchargement et installation des outils	1
1.2.1 - Configuration de Nginx.....	2
1.2.2 - Configuration de PHP	4
1.2.3 - Configuration de MariaDB.....	5
1.2.4 - Configuration de Symfony	6
1.2.5 - Informations.....	6
1.3 - Outils de développement (optionnel)	7
1.4 - Exécution de l'API	7
1.4.1 - Premier lancement.....	7
1.4.2 - Lancement de l'API.....	7
1.5 - Exécution du site web.....	8
2 - API	10
2.1 - Fonctionnement.....	10
2.2 - Routes.....	10
2.3 - Table des droits d'accès.....	11
2.4 - Identification	12
2.5 - L'enregistrement de données	12
2.6 - Fonctions	13
2.6.1 - Explication du processus complet	13
2.7 - Tests unitaires	14
3 - Site Web	15
3.1 - La couche logicielle	15
3.2 - Le Framework.....	15
3.3 - Symfony et DataCity	15
3.3.1 - Bundle faisant partie du cœur de Symfony	15
3.3.2 - Bundle Tiers.....	16

3.3.3 - Bundle de Datacity	16
3.4 - La partie publique du site web.....	17
3.5 - La partie privée du site web	17
3.6 - Les jeux de tests.....	18

1 - Configuration de l'environnement

DataCity est composé en plusieurs parties :

- Le web-service REST propulsé par Node.js
- La base de données ElasticSearch servant à stocker les fichiers et jeux de données
- Le site web Symfony
- La base de données relationnelle (MariaDB/PostgreSQL)
- Projets liés aux partenariats
 - Application mobile ERP pour la Ville de Montpellier

1.1 - Importation des projets

Le web-service et le site web utilisent le gestionnaire de versions Git¹ et sont hébergés sur la plateforme [GitHub](#). Pour récupérer les projets, vous devez posséder un logiciel permettant d'exploiter Git comme [Cygwin](#), [Tortoise](#), [git-scm](#), ..., ou utiliser le module Git intégré dans les IDE² d'[Eclipse](#), [Visual Studio](#), ou votre environnement de développement préféré.

Localisation des projets :

- Module de traitement (API) : <https://github.com/palmsnipe/datacity-parser.git>
- Web service : <https://github.com/palmsnipe/datacity-api.git>
- Site web : <https://github.com/Wykks/Datacity.git>

1.2 - Téléchargement et installation des outils

Vous pouvez télécharger et installer la dernière version des serveurs Node.js et ElasticSearch nécessaires au fonctionnement de l'API ici :

- Node.js : <http://nodejs.org>
- ElasticSearch : <http://www.elasticsearch.org/overview/elkdownloads>

Certains modules de l'API node nécessitent d'être recompilés, **il faut impérativement installer Visual Studio si vous êtes sur Windows ainsi que Python 2.**

- Visual Studio : Télécharger depuis l'intranet Bocal Epitech la version Pro.
- Python 2 : <https://www.python.org/downloads>

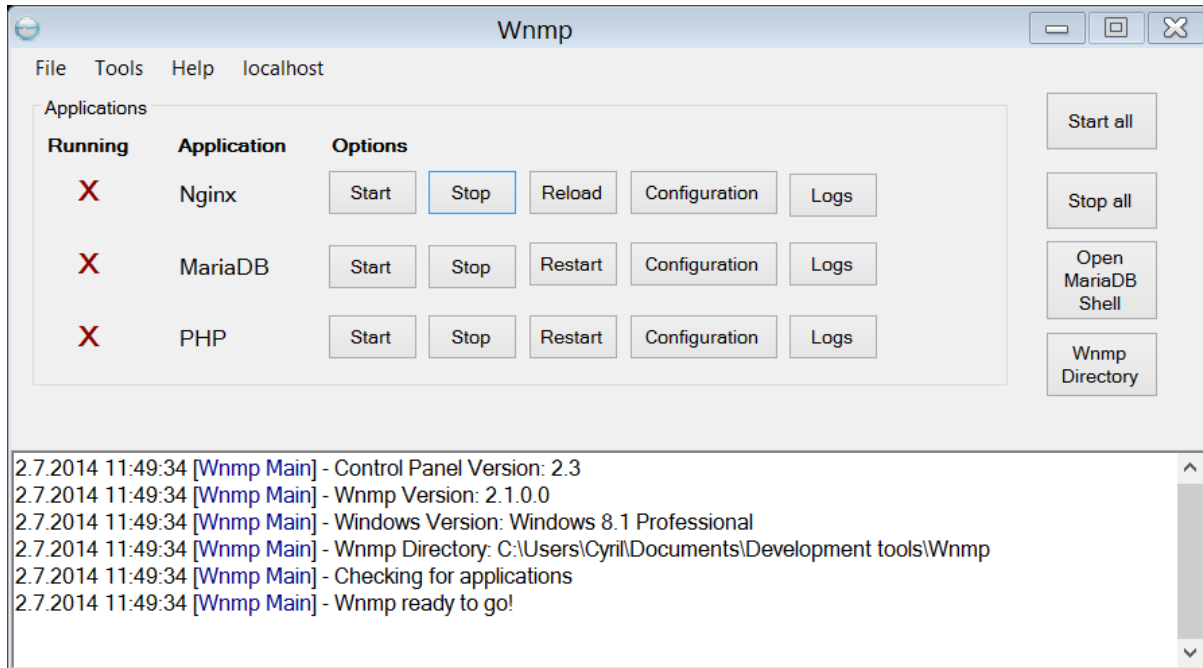
Le site web est configuré pour être lancé sur un serveur Nginx, avec MariaDB et PHP. Sur Windows, il est possible d'installer l'environnement en utilisant Wnmp. Installez les outils ci-dessous :

¹ Gestionnaire de versions Git - [Wikipédia](#)

² Environnement de développement - [Wikipédia](#)

- Wnmp : <http://www.getwnmp.org> (Attention, le répertoire d'installation ne doit pas contenir d'espace)
- jpegtran : <http://gnuwin32.sourceforge.net/downloadlinks/jpeg.php>
- Composer : <https://getcomposer.org/download>

Pour tester le bon fonctionnement du serveur, lancer Wnmp et démarrer tous les services.



1.2.1 - Configuration de Nginx

Remplacer le fichier de configuration de Nginx NGINX.CONF par ce contenu :

```
worker_processes 1;

error_log logs/error.log;
pid logs/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include mime.types;
    default_type application/octet-stream;

    access_log logs/access.log;

    sendfile on;
    #tcp_nopush on;

    #keepalive_timeout 0;
    keepalive_timeout 65;
    ssl_session_timeout 10m;
    ssl_protocols TLSv1.2 TLSv1.1 TLSv1 SSLv3;
    ssl_ciphers
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:ECDH+3DES:DH+3DES:RS
A+AES:RSA+3DES:!ADH:!AECDH:!MD5:!DSS;
    ssl_prefer_server_ciphers on;
```

```

gzip on;
# http server
server {
    listen 80; # IPv4
    server_name localhost;

    ## Parameterization using hostname of access and log filenames.
    access_log logs/localhost_access.log;
    error_log logs/localhost_error.log;

    ## Root and index files.
    root C:/[Path DataCity]/web;
    rewrite ^/app_dev\.php/?(.*)$ /$1 permanent;

    ## If no favicon exists return a 204 (no content error).
    location = /favicon.ico {
        try_files $uri =204;
        log_not_found off;
        access_log off;
    }

    ## Don't log robots.txt requests.
    location = /robots.txt {
        allow all;
        log_not_found off;
        access_log off;
    }

    location @rewriteapp {
        rewrite ^(.*)$ /app_dev.php/$1 last;
    }

    location ~ ^/(app|app_dev|config)\.php(/|$) {
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_split_path_info ^(.+\.(php))(/.*)$;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param HTTPS off;
        fastcgi_buffers 8 16k;
        fastcgi_buffer_size 32k;
    }

    # Try the requested URI as files before handling it to PHP.
    location / {
        index app_dev.php;
        try_files $uri @rewriteapp;
    } # / location
} # end http server

# https server
server {
    listen 443 spdy ssl;
    server_name localhost;
    ssl_certificate cert.pem;
    ssl_certificate_key key.pem;

    ## Parameterization using hostname of access and log filenames.
    access_log logs/localhost_access.log;
    error_log logs/localhost_error.log;

    ## Root and index files.
    root C:/[Path DataCity]/web;
    rewrite ^/app_dev\.php/?(.*)$ /$1 permanent;

    ## If no favicon exists return a 204 (no content error).
    location = /favicon.ico {
        try_files $uri =204;
        log_not_found off;
        access_log off;
    }

    ## Don't log robots.txt requests.
    location = /robots.txt {
        allow all;
        log_not_found off;
        access_log off;
    }

```



```

}

location @rewriteapp {
    rewrite ^(.*)$ /app_dev.php/$1 last;
}

location ~ ^/(app|app_dev|config)\.php(/|$) {
    fastcgi_pass 127.0.0.1:9000;
    fastcgi_split_path_info ^(.+\.(php|php5))(/.*)$;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param HTTPS on;
    fastcgi_buffers 8 16k;
    fastcgi_buffer_size 32k;
}

# Try the requested URI as files before handling it to PHP.
location / {
    index app_dev.php;
    try_files $uri @rewriteapp;
} # / location

} # end http server

server {
    listen 80;
    server_name phpmyadmin.localhost;

    access_log logs/localhost_access.log;
    error_log logs/localhost_error.log;

    root html/phpmyadmin/;
    index index.php;

    location ~* \.php$ {
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_split_path_info ^(.+\.(php|php5))(/.*)$;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param HTTPS off;
        fastcgi_buffers 8 16k;
        fastcgi_buffer_size 32k;
    }
}
}

```

Attention, il faut adapter les directives root (ligne 38 & 87).

1.2.2 - Configuration de PHP

Il est aussi nécessaire de modifier le fichier de configuration PHP `PHP.INI`.

Modifier la limite de la mémoire (ligne 405) :

```
memory_limit = 256M
```


Et commenter la ligne pour désactiver opcache pour php cli (ligne 1918) :

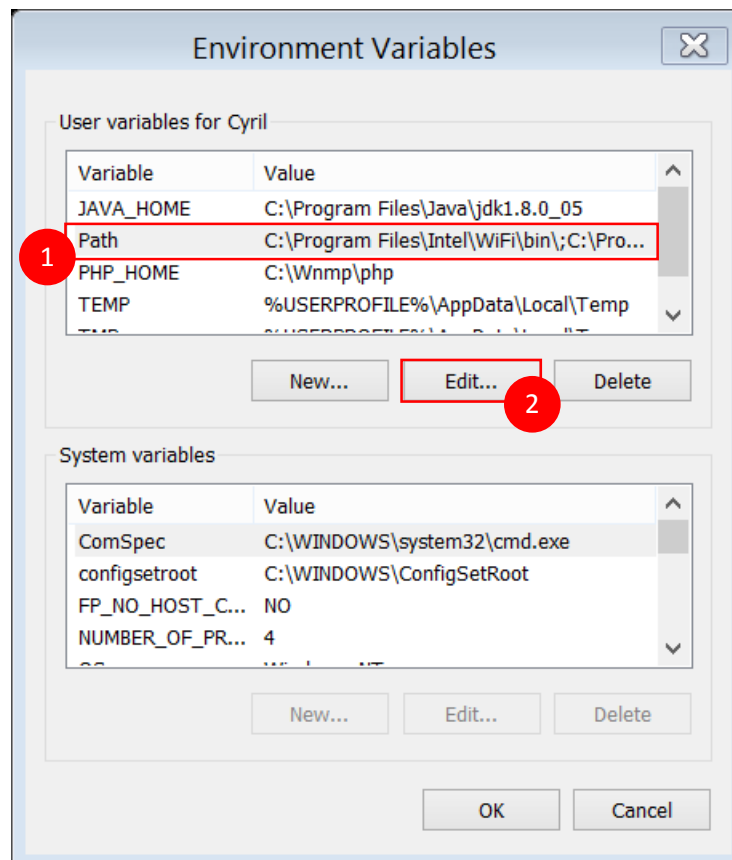
```
;opcache.enable_cli=1
```

Ajouter à la fin du fichier :

```
suhosin.executor.include.whitelist = phar
```

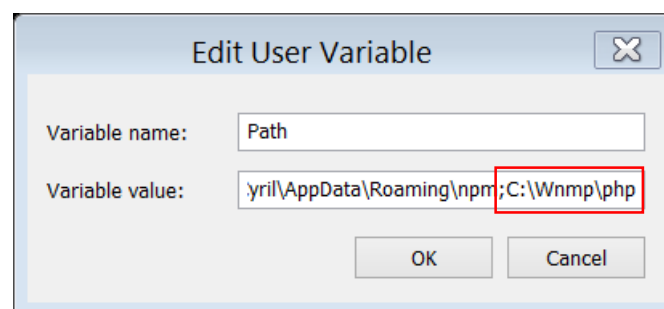
Vous pouvez redémarrer les services.

Par la suite, nous aurons besoin d'utiliser PHP en ligne de commande. Pour les utilisateurs Windows, il faut modifier les variables d'environnement. Pour cela, ouvrez la fenêtre ci-dessous en faisant une recherche « variables d'environnement » (raccourci :  + S).



Cliquez sur la ligne Path et sur le bouton Editor.

Mettez ensuite le chemin de votre installation PHP dans Wnmp et valider :



```
;C:\wnmp\php
```

1.2.3 - Configuration de MariaDB

Sur Wnmp : cliquer sur « Open MariaDBShell »

(mot de passe : password)

```
CREATE DATABASE datacity;
```

```
GRANT ALL PRIVILEGES ON datacity.* TO 'datacity'@'localhost' IDENTIFIED BY
'datacity' WITH GRANT OPTION;
FLUSH PRIVILEGES;
```

1.2.4 - Configuration de Symfony

Attention pour Skype : il faut désactiver le port 80/443 : Options → Avancées → Connexion → Décocher « Utiliser les ports... »

Attention pour VMware si utilisation des Shared Vms : Edit → Preferences... → Shared VMs → Change Settings → Disable Sharing → Mettre 8443 par exemple → Enable Sharing (si besoin).

Rentrer les informations suivantes :

```
database_driver (pdo_mysql):
database_host (127.0.0.1):
database_port (null):
database_name (symfony):datacity
database_user (root):datacity
database_password (null):datacity
mailer_transport (smtp):
mailer_host (127.0.0.1):
mailer_user (null):
mailer_password (null):
locale (en):fr
secret (ThisTokenIsNotSoSecretChangeIt):b92e0b5a484b36457682ccba4334b1b6
jpegtran_path (/usr/bin/jpegtran):'C:\\Program Files
(x86)\\Gnuwin32\\bin\\jpegtran.exe'
```

A partir de maintenant le site devrait fonctionner : <http://localhost/>

En cas de problème de rendu, régénérer le cache et réinstaller les ressources :

```
php app/console ca:c && php app/console assets:i --symlink && php app/console
asseti:d
```

1.2.5 - Informations

Liste des commandes symfony :

```
php app/console
```

Un problème ? Essayer de régénérer le cache :

```
php app/console cache:clear
```

Si vous travailler sur les ressources de type CSS/JS il faut indiquer a assetic de redump les fichiers :

```
php app/console assetic:dump --watch
```

Ne pas hésiter à faire des alias bash pour ces commandes.

Par exemple dans un fichier .profile (à mettre dans le home (C:\\Users\\USER sous Windows)) :

```
alias sfclean='php app/console ca:c && php app/console assets:i --symlink && php
app/console asseti:d'
alias sfdump='php app/console asseti:d --watch'
```

A faire après chaque « git pull » (ou « sync » pour les clients graphique)

```
composer install
php app/console doc:sche:up --force
php app/console doc:fix:lo
```

1.3 - Outils de développement (optionnel)

Pour faciliter le développement sur le site web, il est conseillé d'utiliser des modules pour Symfony2 :

- Pour Eclipse : <http://symfony.dubture.com/installation>
- Pour SublimeText : <http://www.pd-digital.de/symfonycommander>

1.4 - Exécution de l'API

1.4.1 - Premier lancement

Pour le premier lancement, il faut installer les modules node, et installer le service ElasticSearch.

Il faut aller dans le répertoire du projet de l'API, puis exécuter cette commande :

```
[Répertoire DataCity API]\npm install
```

Il faut ensuite installer le service ElasticSearch :

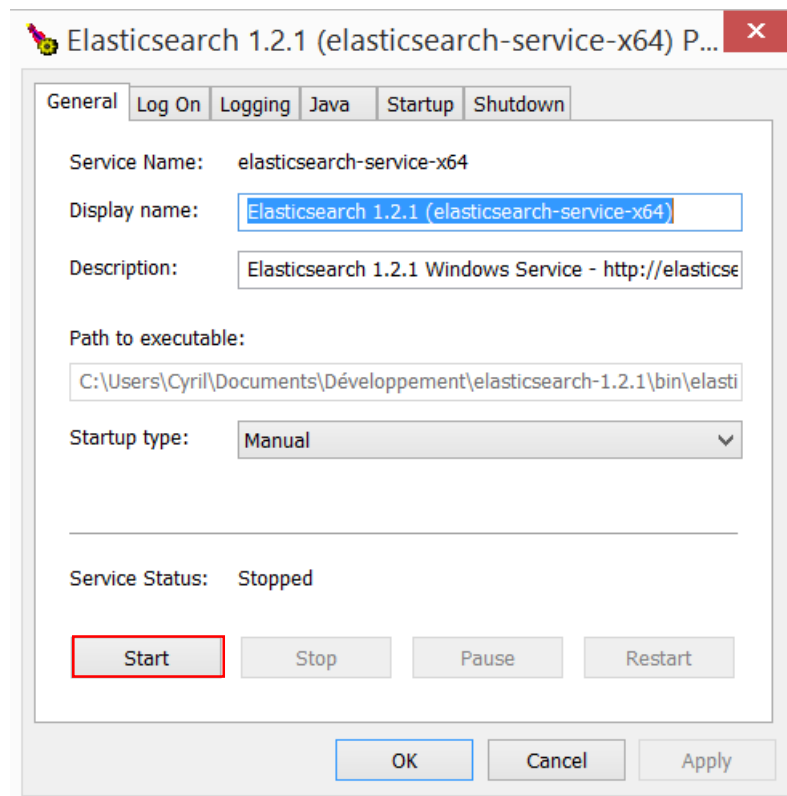
```
[Répertoire ElasticSearch]\bin\service.bat install
```

Vous pouvez maintenant lancer l'API comme cela est décrit sur le point suivant.

1.4.2 - Lancement de l'API

Il faut absolument que la base de données ElasticSearch soit lancée. Pour se faire, allez dans son répertoire puis exécuter le service :

```
[Répertoire ElasticSearch]\bin\service.bat manager
```



Une fois les modules installés et ElasticSearch lancé, vous pouvez lancer l'API depuis son répertoire :

```
[Répertoire DataCity API]node app.js
```

Pour simuler des requêtes HTTP, utilisez un module REST sur votre navigateur tel que POSTMAN

- <https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcm>

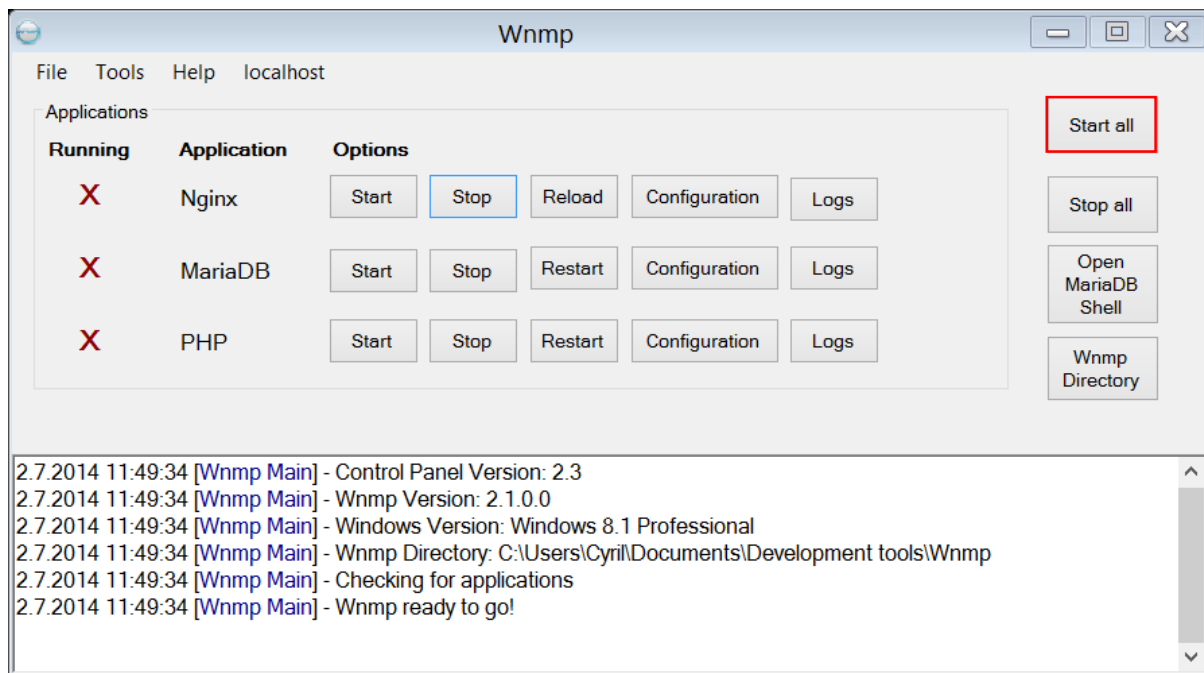
L'adresse devrait être de ce type avec pour {route}, une des routes disponibles plus basses.

```
http://localhost:4567/{ROUTE}
```

Ne pas oublier de mettre les informations nécessaires à la route associée tel que la *public_key* et la *private_key* dans le header.

1.5 - Exécution du site web

Pour le site web, il faut lancer les serveurs MariaDB et Nginx.



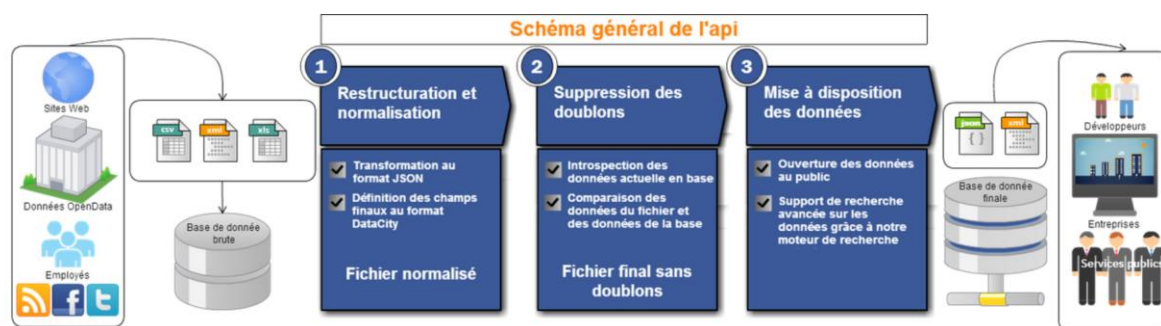
2 - API

2.1 - Fonctionnement

L'API DataCity est organisé sur l'architecture REST. Notre API est pensée pour avoir des URL prévisibles axées sur les ressources et pour utiliser les codes HTTP en réponse pour indiquer les erreurs API.

Nous utilisons les fonctions HTTP intégrées telles que l'authentification HTTP et les méthodes HTTP, qui peuvent donc être interprétées par des clients HTTP imprévus. Nous supportons le partage de ressources de plusieurs origines pour vous permettre d'interagir de manière sécurisée avec notre API depuis une application web coté client (sans oublier que vous ne devriez jamais exposer votre clé API secrète dans le code d'un site internet public coté client)

Du JSON sera retourné en réponse depuis l'API, incluant les codes d'erreurs.



2.2 - Routes

Route	Type	Paramètres	HEAD	Retour success	Retour erreur
/parse	POST	"file":UPLOADED_FILE	"public_key":PUBLIC KEY	TYPE: json {status:"success" data:JSON_FILE}	TYPE: json {status:"error", data:ERR_INFO}
[slugDataset]/source	POST	"file":UPLOADED_FILE"model": [{name: string, type: string, mandatory:	"public_key":PUBLIC KEY "private_key":PRIVATE KEY	TYPE: json {status:"success" slugDataset: string}	TYPE: json {status:"error", data:ERR_INFO}

		bool, unique: bool}, ...]			
[slugDataset]/[slugSource]/model	GET		"public_key":PUBLIC KEY	TYPE: json {status:"success" model: [{name: string, type: string, mandatory: bool, unique: bool}, ...]}	TYPE: json {status:"error", data:ERR_INFO}
[slugDataset]/download	GET		"public_key":PUBLIC KEY "file_format(accept)":FILEFORMAT	TYPE: json {status:"success" data:JSON_CONTENT}	TYPE: json {status:"error", data:ERR_INFO}
[slugDataset]/[slugSource]	DELETE		"public_key":PUBLIC KEY "private_key":PRIVATEKEY	TYPE: json {status:"success" data:SUCCESS_MSG}	TYPE: json {status:"error", data:ERR_INFO}
[slugDataset]/	DELETE		"public_key":PUBLIC KEY "private_key":PRIVATEKEY	TYPE: json {status:"success" data:SUCCESS_MSG}	TYPE: json {status:"error", data:ERR_INFO}

2.3 - Table des droits d'accès

Route	Type	Anon.	User	Admin
/parse	POST	oui	oui	oui
/upload	POST	non	oui	oui
/download	POST	oui	oui	oui
/delete	POST	non	limité*	oui

*limité : Peut seulement supprimer les fichiers dont l'utilisateur est propriétaire.

2.4 - Identification

L'identification à l'API DataCity se fait en fournissant votre clé API dans la requête. Vous avez accès à celle-ci depuis votre compte sur le site. Votre clé API renferme vos droits d'accès, gardez là en lieu sûr.

L'identification à l'API se fait via le header en utilisant la clé API comme nom d'utilisateur. Vous n'avez pas besoin de fournir de mots de passe.

Exemple de requête jQuery :

```
var publicKey = x456q168z4s51sss;
var parameters = {"paramPostExample": "1"}
$.ajax({
  url: "http://api.datacity.fr/parse/",
  type: 'POST',
  data: parameters,
  headers: {"public_key": publicKey}
  contentType: false,
  processData: false,
  success: function(data, textStatus, jqXHR) {
    console.log(data);
  },
  error: function(err) {
    console.error(err);
  }
});
```

2.5 - L'enregistrement de données

Les données sont stockées en sous la forme de documents dans la base de données Elasticsearch. Ce n'est pas une base de données SQL orientée fichiers.

Voici l'architecture de la base de données :

- index: dans un contexte relationnel, cela représente la base de données.
- type: dans un contexte relationnel, ceci représente une table.
- id: l'ID d'un document

Notre architecture pour les données : `/sources/:category/:id`

Cela signifie que toutes les données vont être représentées en tant que document. Pour le cas suivant :

```
[
  {
    "ex1": "1",
    "ex2": "2"
  },
  {
    "ex3": "3",
    "ex4": "4"
  }
]
```

Deviendra :

```
/sources/example/1
{
  "ex1": "1",
```

```
    "ex2": "2"  
  }  
  /sources/example/2  
  {  
    "ex3": "3",  
    "ex4": "4"  
  }  
}
```

2.6 - Fonctions

Les fonctions sont regroupées par sections (utilisateur, fichier, sources) et certaines d'entre elles sont limitées en fonction des droits utilisateur.

Si vous lisez (Accès administrateur uniquement), cela signifie que cette fonction est uniquement disponible pour les administrateurs. Nous décrivons cela dans le but de donner à nos partenaires et aux personnes contribuant au projet une vue globale des fonctions en accès limité.

2.6.1 - Explication du processus complet

Prenons l'exemple d'un développeur souhaitant créer une nouvelle application mobile. Dans un premier temps, il doit trouver un fichier de données libres lié à l'utilisation qu'il souhaite donner à son application, dans notre exemple les services publics. Ensuite il aura à corriger les erreurs éventuelles de nommage des catégories du fichier et aussi trouver une base de données sur laquelle stocker le fichier. Il sera peut être nécessaire d'utiliser ses propres outils pour rendre le fichier exploitable.

Avec notre API, nous facilitons tout ce long processus. L'utilisateur utilisera notre API dans son application pour faciliter toutes ces étapes. Il le pourra en utilisant notre API :

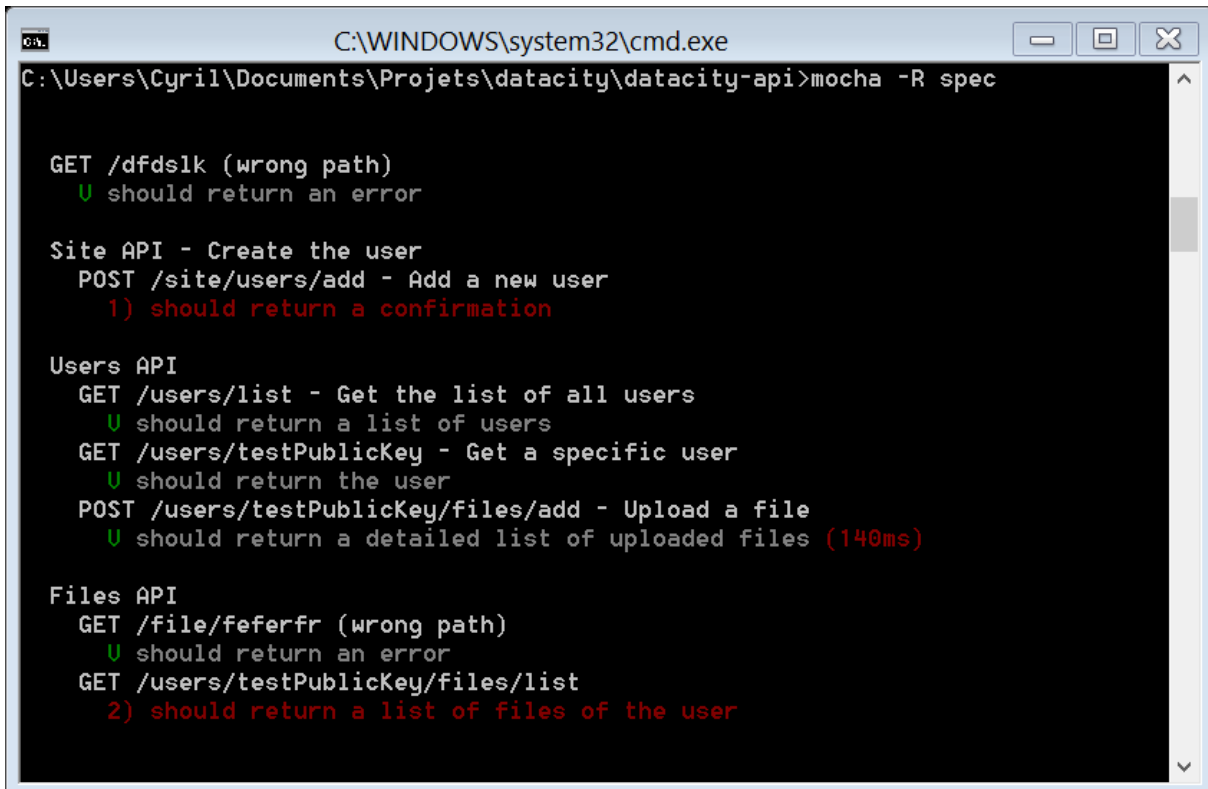
- analyser le fichier
- créer la source
- exploiter la source en utilisant de nombreuses fonctionnalités telles que la recherche par géolocalisation, recherche avancée ou le filtrage par résultat... (à venir).

2.7 - Tests unitaires

Pour les tests unitaires, nous avons choisi [Mocha](#), qui est adapté pour la génération de tests unitaires pour les projets node.js.

Pour lancer les tests unitaires, il faut que le serveur Elasticsearch soit lancé ainsi que l'API DataCity.

Afin d'avoir une meilleure visibilité, il est conseillé d'exécuter Mocha avec le *Reporter spec* comme sur la capture d'écran ci-dessous.



```
C:\WINDOWS\system32\cmd.exe
C:\Users\Cyril\Documents\Projets\datacity\datacity-api>mocha -R spec

GET /dfdslk (wrong path)
  U should return an error

Site API - Create the user
  POST /site/users/add - Add a new user
    1) should return a confirmation

Users API
  GET /users/list - Get the list of all users
    U should return a list of users
  GET /users/testPublicKey - Get a specific user
    U should return the user
  POST /users/testPublicKey/files/add - Upload a file
    U should return a detailed list of uploaded files (140ms)

Files API
  GET /file/feferfr (wrong path)
    U should return an error
  GET /users/testPublicKey/files/list
    2) should return a list of files of the user
```

3 - Site Web

3.1 - La couche logicielle

Le site web est composé d'une suite de logiciels :

NGINX : Le serveur http, permettant de servir toutes les pages. Pour le moment sa version n'a pas trop d'importance puisque nous utilisons les fonctionnalités simples de nginx. Il est prévu dans le futur d'utiliser le protocole SPDY, donc nginx 1.5.11 minimum. Apache est également supporté.

PHP FPM : C'est l'interpréteur PHP qui va générer les pages. Il communique directement avec nginx via socket. Pour le moment nous utilisons sa version 5.4, bientôt en 5.5 notamment dû au changement d'opcache.

MariaDB : La base de données relationnelle, permettant de stocker notamment tous les comptes utilisateurs. La version utilisée est la série 5.5.x. MySQL et postgresSQL sont également supportés.

Prévu :

Varnish : Un reverse proxy permettant de mettre en cache les pages générées par Symfony. Actuellement le reverse proxy de Symfony est utilisé.

3.2 - Le Framework

Le site web de DataCity est conçu à l'aide du Framework Symfony 2.

Symfony 2 est un Framework PHP de type MVC³, c'est-à-dire Modèle Vue Contrôleur.

3.3 - Symfony et DataCity

Symfony est découpé sous forme de « bundle ». Un bundle est une sorte de plugin comme on peut en trouver dans d'autres logiciels.

Hormis les bundles spécifiques au projet, ils ne sont pas localisés dans le dépôt de DataCity. Ils sont récupérés via le gestionnaire de dépendance PHP « Composer ».

La liste des bundles utilisés dans DataCity se divise en bundle existant dans le cœur de Symfony, en bundles tiers et aussi en bundles personnalisés.

3.3.1 - Bundle faisant partie du cœur de Symfony

FrameworkBundle

Le bundle contenant les fonctions de base de Symfony.

SecurityBundle

³ Modèle Vue Contrôleur – [Wikipédia](#)

Permet de définir les droits d'accès

TwigBundle

Le moteur de modèle TWIG. Toute les pages de DataCity sont rendue a par ce moteur.

MonologBundle

La gestion des logs.

AsseticBundle

Permet une meilleure utilisation des ressources (image, CSS, JS). Il est très utilise dans DataCity notamment pour fusionner les fichiers JavaScript et CSS.

DoctrineBundle

Un ORM⁴ (Object Relationnal Mapping), très utilise dans DataCity pour gérer tout ce qui concerne la base de données. Doctrine ajoute une couche d'abstraction permettant à DataCity de fonctionner sur différents SGBD. Tous les modèles de Datacity sont représentés sous forme de class, ce qui permet (entre autre) de versionner le schéma de la base de donnée.

3.3.2 - Bundle Tiers

BcBootstrapBundle

Intègre le Framework CSS Bootstrap dans Symfony 2. Il fournit notamment directement les templates twig adaptés à bootstrap pour les formulaires Symfony.

FOSUserBundle

Fourni un grand nombre de fonctionnalités pour la gestion d'utilisateurs, tels que l'inscription, la vérification via email, le changement de mot de passe, etc...

DoctrineFixturesBundle

Permet de créer des jeux de données nommés « Fixture » dans la base de données

MisdGuzzleBundle

Intègre le Framework Guzzle dans Symfony 2. Guzzle permet d'effectuer des requêtes http simplement en PHP. Il est utilisé pour communiquer avec l'API de Datacity.

FOSJsRoutingBundle

Ce bundle permet d'utiliser le système de routing de Symfony 2 dans les fichiers javascript.

3.3.3 - Bundle de Datacity

DatacityPublicBundle

⁴ Mapping objet-relationnel – [Wikipédia](https://fr.wikipedia.org/wiki/Mod%C3%A8le_objet-relationnel)

Gère toutes les pages publiques du site web.

DatacityPrivateBundle

Gère toutes les pages privées du site web ; c'est-à-dire toutes les pages nécessitant un compte DataCity.

DatacityUserBundle

Surcouche au *FOSUserBundle*, permettant de le personnaliser notamment pour l'agencement des différentes pages utilisateurs fournie.

3.4 - La partie publique du site web

Tout ce qui concerne la partie public du site web se situe ici :

- `src\Datacity\PublicBundle`
- `src\Datacity\UserBundle`

La liste des routes de la partie publique est récupérable via la console Symfony :

```
php app/console router:debug | grep -e datacity_public -e fos_user | grep -v private
```

Actuellement un seul contrôleur (*DefaultControler*) gère toutes les pages publiques.

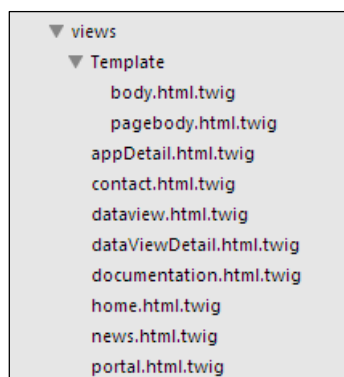
Un contrôleur supplémentaire et prévu pour la partie de visualisation de données.

Les vues sont organisées de la manière suivante :

La page *body.html.twig* contient l'entête et le pied de page.

La page *pagebody.html.twig* contient l'agencement d'une page classique.

A l'exception de la page d'accueil, toutes les pages héritent directement la page *pagebody.html.twig*.



3.5 - La partie privée du site web

Tout ce qui concerne la partie public du site web se situe ici :

```
src\Datacity\PrivateBundle
```

```
src\Datacity\UserBundle
```

La liste des routes de la partie privée est récupérable via la console Symfony :

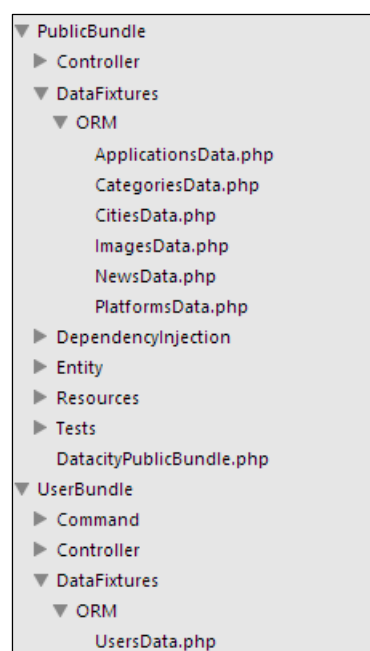
```
php app/console router:debug | grep private
```

Actuellement il y a 2 contrôleurs :

Le *DashboardController* pour le tableau de bord. Et un *ApiController* gérant les requêtes vers l'API Datacity.

L'*ApiController* utilise un service *PrivateApi* gérant la communication avec l'API Datacity.

Les vues sont organisées de la même manière que pour la partie publique. Il existe un dossier « include » dans les vues, contenant une partie de page utilisée dans la partie publique : le menu en haut à droite, affiché une fois connecté au site.



3.6 - Les jeux de tests

Les jeux de test sont placés dans les dossiers « DataFixtures » de chaque bundle.

Le bundle « PrivateBundle » ne possède pas de fixture puisqu'il ne définit pas d'entité (table de base de données).

Les jeux de données sont chargés depuis la console Symfony :

```
php app/console doctrine:fixtures:load
```

Des utilisateurs sont créés via la fixture « UsersData », cependant il est possible d'en créer rapidement sans utiliser le formulaire du site via la console :

```
php app/console datacity:user:create
```

