

# Rapport d'alternance développeur full stack chez Finalgo

Réalisé par : Julien Giraud

Diplôme préparé : Master 2 Informatique

Tuteur entreprise : Bertrand Héllion

Tuteur universitaire : Lionel Médini

Durée : du 07/09/2020 au 07/09/2022 (2 ans)



Université Claude Bernard



Lyon 1

## Remerciements

Pour commencer je tiens à remercier toute l'équipe de Finalgo ainsi que les dernières générations de stagiaires et alternants pour m'avoir intégré, accompagné et partagé leur bonne humeur au cours de ces deux dernières années.

En particulier je remercie Bertrand Hellion de m'avoir accepté à Finalgo et encadré tout au long de mon alternance. À ses côtés j'ai acquis de nombreuses compétences comme écrire du code robuste, utiliser le plus possible la généricité et avoir un esprit critique sur les missions à réaliser.

Enfin, je remercie Lionel Médini mon tuteur universitaire pour m'avoir aidé à la rédaction de ce rapport.

# Table des matières

<b>Remerciements</b>	<b>2</b>
<b>Glossaire</b>	<b>4</b>
<b>Introduction</b>	<b>6</b>
<b>1 Présentation de Finalgo</b>	<b>7</b>
1.1 L'équipe	7
1.2 Les projets	8
<b>2 Environnement de travail</b>	<b>8</b>
2.1 Matériel et lieu	8
2.2 Outils et logiciels	9
a. Slack	9
b. Asana	9
c. Google Workspace	10
d. Gather	11
<b>3 Environnement technique</b>	<b>12</b>
3.1 Côté serveur	12
3.2 Côté client	14
3.3 Outils de développement	14
<b>4 Travail réalisé</b>	<b>16</b>
4.1 Notre modèle de données	16
4.2 Problèmes de l'ancienne implémentation	17
4.3 Procédure de refactoring et migration des données	17
a. Mise en place d'une base propre	17
b. Préparation de la migration des objets	19
c. Migration des objets sur le nouveau système de données	19
d. Retours sur la mission	21
<b>Conclusion</b>	<b>22</b>
<b>Annexes</b>	<b>23</b>
Nettoyage des tables en SQL	23
Mise à jour des getters et setters en Java	23
Migration des données en SQL	23
Mise à jour des identifiants et nettoyage des tables en SQL	24
Script de mise à jour des identifiants pour le système de fichiers	24
Minimisation des appels à la base de données dans une méthode Java	24

# Glossaire

Terme	Définition
Angular	Angular est un framework MVC basé sur TypeScript, développé par Google et utilisé pour la création d'applications web en Single Page Application.
API	Une API (Application Programming Interface) est un programme qui permet à des applications différentes de communiquer ensemble afin d'échanger des données.
Back-End	Désigne l'ensemble des traitements effectués côté serveur afin de permettre le bon fonctionnement d'une application.
Composant	Dans Angular, un composant est un ensemble formé d'une page HTML, d'un fichier CSS et d'une classe TypeScript. Plus précisément, il s'agit d'une entité réutilisable et les pages sont des composants, eux-mêmes formés de plusieurs composants.
DAO	Abréviation de Data Access Object, il s'agit d'un design pattern structurel qui permet d'abstraire les manipulations liées à la persistance des données à travers à travers des services aussi appelés les DAO.
Design pattern	Arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel (patron de conception).
Dette technique	Désigne les différents surcoûts liés à des choix technologiques non adaptés.
Enum	Un enum est un type de donnée contenant un nombre fixe de valeurs constantes.
Financement alternatif	Système de financement qui ne repose pas sur les systèmes financiers traditionnels comme les banques réglementées et les marchés de capitaux. Par exemple les prêts à la consommation en ligne, les fonds de prêt aux entreprises en ligne et affacturage ou le financement participatif (crowdfunding).
FinTech	Une FinTech est un mot formé par les termes « finance » et « technologie ». Il désigne des entreprises innovantes qui proposent des services financiers à l'aide des nouvelles technologies.
Framework	Ensemble d'outils et de composants applicatifs utilisés comme structure logicielle. Contrairement aux bibliothèques, le framework est une couche d'abstraction qui impose le flot d'exécution de l'application.
Front-End	Désigne le visuel et l'ensemble des traitements réalisés par une application web côté client, c'est à dire depuis un navigateur web.
Getter	Méthode permettant d'accéder à un attribut d'un objet.
Hotfix	Correction de bug mise en production sans forcément suivre la procédure habituelle par soucis de rapidité

Terme	Définition
Map	Une map est un type de données qui relie un ensemble de clés à un ensemble de valeurs.
Mise en production	Une mise en production est un procédé permettant de déployer une nouvelle version d'une application.
Refactoring	Étape de réécriture de code sans ajout ou modification des fonctionnalités.
SaaS	Le SaaS « software as a service » est un logiciel hébergé par un tiers et accessible à distance. Généralement, cette solution est facturée sous la forme d'un abonnement mensuel.
Service	Un service est une instance d'une classe TypeScript injectable dans tous les composants d'une application pour les aider à communiquer entre eux. Cette classe centralise les données et facilite la réutilisation de code.
Setter	Méthode permettant de modifier un attribut d'un objet.
Single Page Application	Une SPA est un type d'application web qui possède la particularité de mettre à jour le visuel du site sans recharger la page.
TypeScript	Langage de programmation basé sur JavaScript avec un système de typage, de classe et d'héritage similaire à celui de Java. On peut le convertir en JavaScript ce qui rend son exécution possible sur tous les navigateurs web ou moteurs JS.

# Introduction

Dans le cadre de mon Master Informatique à l'UCBL<sup>1</sup> j'ai opté pour un cursus en alternance au sein de Finalgo, une startup locale spécialisée dans les applications web en lien avec les financements.

J'ai pu effectuer cette alternance suite à mon stage optionnel de Licence 3 réalisé au même endroit. Pour celle-ci, un contrat d'apprentissage de deux ans a été signé. Cette période correspond à un Master qui se termine par un diplôme dans les Technologies de l'Information et Web (TIW).

Les notions vues en cours lors de mon DUT, de ma Licence et tout au long du Master ainsi que mes précédentes expériences professionnelles m'ont beaucoup aidé à réaliser mes missions en entreprise.

Certaines missions ont renforcé mes connaissances, d'autres m'ont permis de comprendre des éléments abordés au cours de l'année dans un cadre professionnel.

Tout au long de l'année j'ai réalisé des missions diverses au sein de Finalgo, principalement dans le but de **réduire la dette technique et améliorer les performances de nos applications**.

---

1 Université Claude Bernard Lyon

# 1 Présentation de Finalgo

Finalgo est une FinTech familiale de 8 collaborateurs spécialisée dans la recherche de financements 100 % digitale ainsi que la construction et la gestion de dossiers de financement.

Notre vocation est de faciliter l'accès au financement pour les entrepreneurs, artisans, commerçants et plus généralement aux dirigeants de TPE / PME.

Nous proposons à nos clients trois plateformes web en SaaS qui répondent à ces besoins. La première est un outil de gestion pour les professionnels de la finance, elle permet de construire et gérer des dossiers de financement et fonctionne sous forme d'abonnement payant. La deuxième est à destination des dirigeants de TPE / PME, elle permet de rechercher gratuitement des financements alternatifs sur lesquels nous prenons une commission lorsqu'un partenaire finance le projet. La dernière permet de remplir et de suivre les demandes de financements, elle sert d'intermédiaire entre les dirigeants et nos partenaires financiers qui accordent les financements.

## NOTRE EXPERTISE DU FINANCEMENT PROFESSIONNEL



Figure 1 : Extrait du site finalgo.fr

## 1.1 L'équipe

À Finalgo la hiérarchie est très horizontale. Les cofondateurs font partie intégrante des équipes en plus d'avoir un rôle de manager.

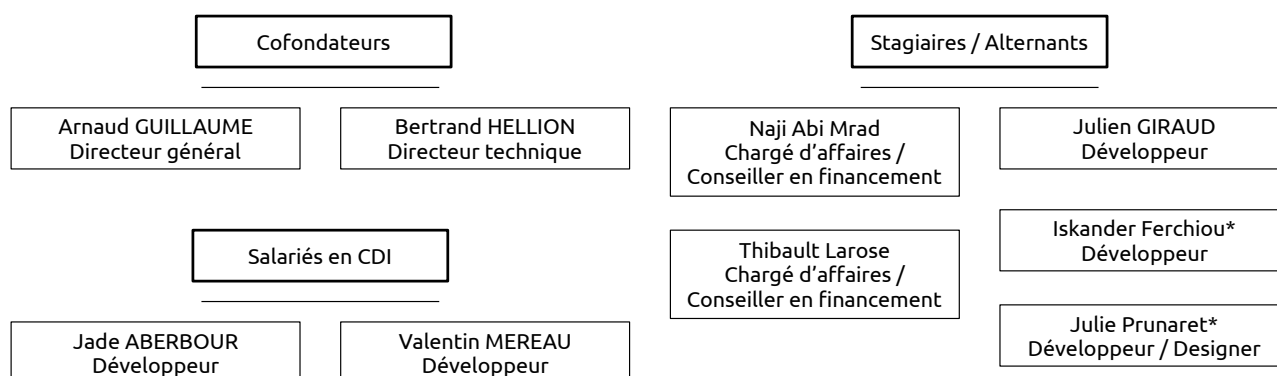


Figure 2 : Organigramme de Finalgo

\* Non présents en même temps.

## 1.2 Les projets

Cette année mes missions ont porté sur les trois principaux projets de Finalgo,

- **Main** notre application de construction et de gestion de dossiers de financement pour les professionnels de la finance,
- **Automate** notre application de recherche de financements alternatifs,
- **Advisor** notre application de gestion, de suivi et d'envoi de demandes de financements alternatifs pour les dirigeants de TPE / PME.

En terme de projets informatiques, toutes les plateformes possèdent un Front-End Angular et elles communiquent avec le même Back-End. La description détaillée de l'architecture se trouve dans la partie Environnement technique.

## 2 Environnement de travail

### 2.1 Matériel et lieu

À Finalgo nous travaillons beaucoup en télétravail. Lorsque nous allons dans les locaux, deux fois par semaine pour les membres de l'équipe à Lyon, nous allons au HUB612. Il s'agit d'un incubateur, une sorte d'open space avec une équipe qui accompagne les entreprises. Le HUB est spécialisé dans les entreprises qui travaillent sur la finance, les assurances et le marketing à l'aide d'outils numériques modernes. C'est un lieu agréable pour travailler et échanger avec des personnes dans des domaines similaires aux nôtres.

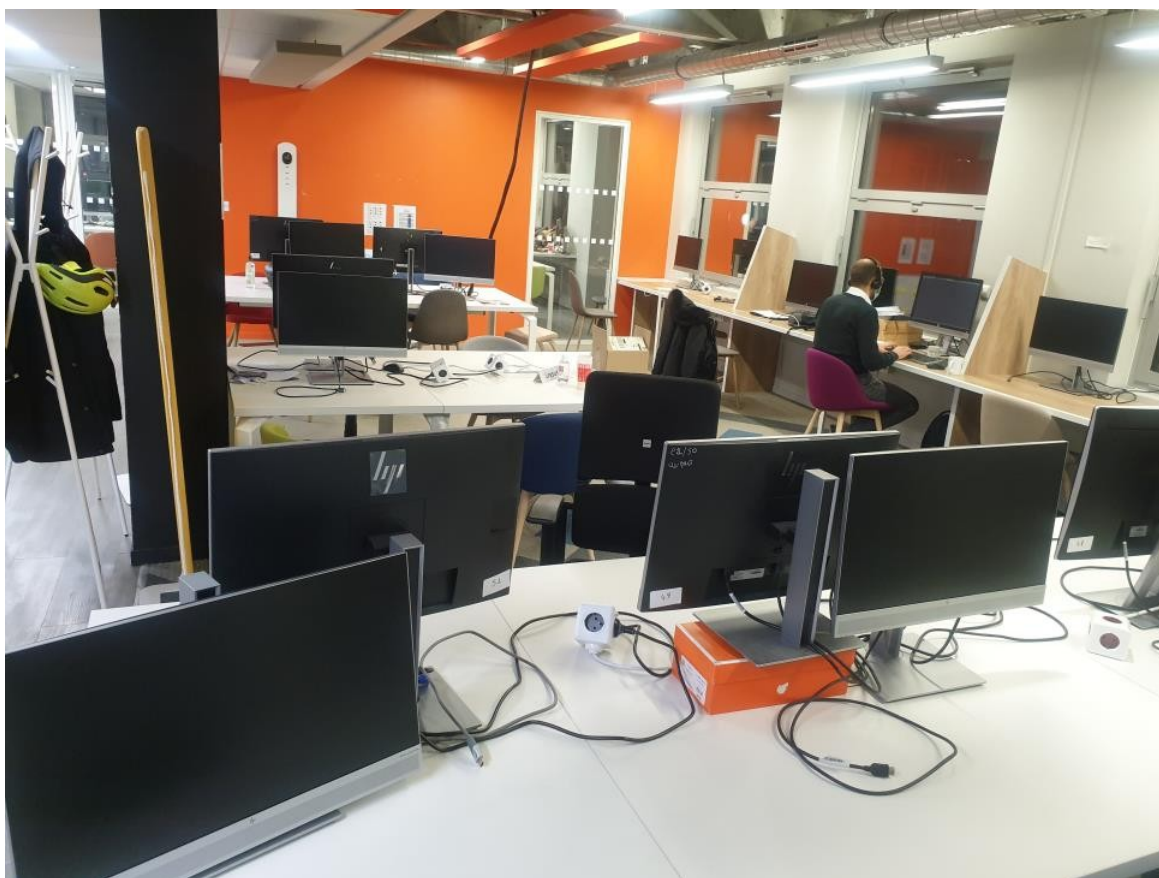


Figure 3 : Locaux du HUB612



Pour travailler on a mis à ma disposition un ordinateur portable avec une très bonne configuration, ce qui me permet d'utiliser efficacement tous les logiciels dont j'ai besoin. J'utilise généralement un deuxième écran pour des questions de confort, il y en a sur les bureaux du HUB et j'utilise mon écran personnel en télétravail.

## 2.2 Outils et logiciels

Depuis le début de la crise sanitaire, toute notre organisation est basée sur du travail en distanciel. Nous sommes donc équipés en outils de communication, visioconférence, gestion des tâches et suite de bureautique.

### a. Slack



Slack est notre outil de messagerie interne.

Nous avons un espace de travail « Finalgo » avec des **canaux de discussion** pour tous les sujets, ce qui permet de configurer les notifications que l'on souhaite recevoir pour chaque type d'information.

Il y a un canal pour :

- chaque **projet** : Back-End, Automate, Advisor
- chaque **domaine** : informatique, marketing, communication, design, recrutement, teambuilding...
- chaque **type de logs** : bugs utilisateurs, boutons « demander de l'aide », actions utilisateurs, actions nécessitant une intervention de notre part, traces des différents serveurs de production et de développement...
- la **détente** : on y trouve des blagues ou liens en tout genre pour partager de la bonne humeur avec l'équipe.

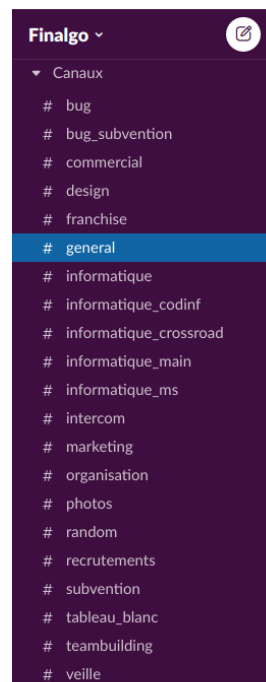


Figure 4 : Aperçu des canaux Slack

### b. Asana



Asana est notre plateforme de gestion des tâches.

La plateforme permet de créer des projets qui fonctionnent comme les tableaux sur Trello. Il est possible d'y créer des colonnes et d'y ajouter des tâches avec des attributions, des images, des sous-tâches.

Nous utilisons ces tableaux pour remplacer le **Scrum Board**, un tableau de post-it utilisé par la méthode SCRUM qui est à l'origine de notre méthode de travail. Grâce à ces tableaux nous pouvons voir qui travaille sur une tâche, connaître son avancement, en ajouter nous-même et écrire les spécifications. Il y a plus de fonctionnalités sur le site mais nous n'en avons pas encore l'utilité.

### c. Google Workspace



*Gmail*



*Drive*



*Meet*



*Calendar*



*Sites*



*Sheets*

Finalgo utilise un système de comptes Google pour les entreprises. Nos comptes nous permettent d'accéder à la suite de bureautique comme avec une adresse **Gmail** classique, même s'ils terminent par `@finalgo.fr`. Toutes ces applications sont liées à un espace partagé sur **Google Drive**, auquel nous avons accès avec notre compte.

**Meet** est une plateforme de réunions numériques très simple à utiliser. Il est possible de se connecter à un salon grâce à un lien, on peut ensuite participer avec sa caméra, son micro ou en faisant des partages d'écran. Nous utilisons Meet pour les réunions clients.

**Calendar** est un agenda, il sert surtout à planifier des rendez-vous clients ou des points d'équipe importants. Un salon Meet est associé à chaque événement de l'agenda. Ce système évite de générer un lien et de l'envoyer aux invités, il suffit de les ajouter sur un événement.

**Google Sites** sert exclusivement à voir et alimenter notre wiki interne. Il s'agit d'un site accessible uniquement avec notre compte, sur lequel nous répertorions toutes sortes d'informations utiles pour Finalgo. On y trouve les procédures comme les mises en production, les installations, les plateformes et il y a des explications sur les technologies inhabituelles. C'est un outil que nous utilisons et alimentons beaucoup, il entre dans la philosophie SCRUM en facilitant les formations mutuelles au sein de l'équipe.

**Sheets** est un tableur comme Excel, nous l'utilisons en interne et en externe. Il y a des tableaux administratifs pour gérer des choses comme les congés, certains servent d'outils de maintenance ou de tickets, d'autres permettent de synthétiser des informations afin de les exporter dans le code.

## d. Gather



*Figure 5 : Locaux virtuels sur Gather*

Gather est notre outil de communication interne n°1. Comme Slack, nous sommes connectés toute la journée sur cette plateforme qui permet de se déplacer dans un monde virtuel 2D et d'effectuer des visioconférences. Étant donné que tous les membres de l'équipe ne résident pas à Lyon, cet outil permet de rester en constante communication avec eux malgré la distance. Chaque personne a son propre bureau virtuel et des pièces spécifiques ont été créées pour remplir différentes fonctions : cafétéria pour discuter en arrivant au travail, salle de réunion ou jardin pour jouer à des mini-jeux.

## 3 Environnement technique

### 3.1 Côté serveur

Pour les serveurs nous avons un compte sur la plateforme OVH avec plusieurs VPS sous Debian.

Sur ces VPS nous avons un serveur Apache pour servir le front-end compilé par Angular et un serveur Tomcat pour le back-end Java Spring Boot.

Nos projets Spring Boot utilisent Java version 11 avec divers dépendances. Voici un résumé de celles que j'ai le plus utilisées.

Dépendance	Description
Auth0	Librairie qui permet de manipuler les JWT.
Hibernate / HQL	Framework qui permet de manipuler la base de données à travers des interfaces. Il implémente le HQL, un langage de requête de base de données relationnelles similaire à SQL avec une approche orientée objet.
Swagger	Une interface qui permet de visualiser l'architecture d'une API et d'y envoyer des requêtes.
JSONObject	Une librairie qui permet de manipuler facilement des objets JSON

Il y a d'autres programmes installés sur les serveurs, les plus importants sont les suivants.

Programme	Description
MySQL	Système de gestion de bases de données relationnelles.
Cron	Programme qui permet de programmer l'exécution de scripts ou de logiciels sur Linux.

Il existe une réplique de cette architecture sans les plateformes Advisor et Automate pour l'un de nos clients, nous utilisons un profil Spring pour effectuer la distinction dans le Back-End.

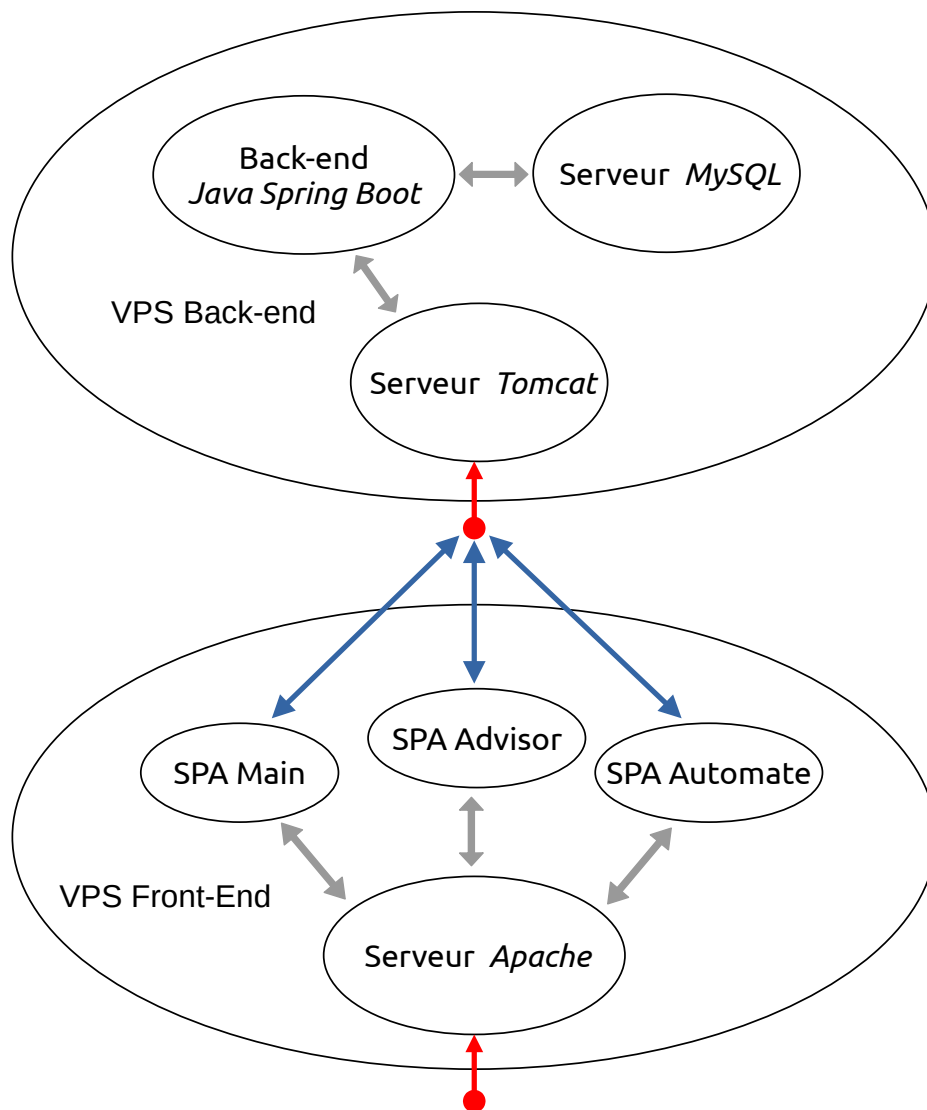





Figure 6 : Architecture des serveurs et services

-  Échanges internes au serveur.
-  Échanges HTTP entre la SPA de la plateforme et le Back-End.
-  Point d'entrée du serveur (VPS) via les requêtes HTTP.

## 3.2 Côté client

En Front-End nous utilisons Angular 11 qui simplifie grandement le développement web.

Notre projet utilise divers dépendances, voici un résumé de celles que j'ai le plus utilisées.

Dépendance	Description
Forms	Ensemble de Classes et composants Angular qui permettent de gérer les formulaires.
Material	Ensemble de classes et composants Angular qui permettent de créer des interfaces graphiques d'application très facilement.
Router	Ensemble de classes et composants Angular qui permettent de gérer les routes d'une application et de manipuler l'URL.
SweetAlert2	Librairie qui permet d'afficher toutes sortes de pop-up personnalisables.

## 3.3 Outils de développement

De façon générale il y a beaucoup d'informations utiles pour le développement sur notre wiki. On y trouve notamment :

- les liens des swaggers,
- les procédures d'installation des serveurs,
- les commandes pour installer MySQL sur notre machine et y charger des données de test,
- la procédure à suivre pour mettre en production le Back-End ou le Front-End.

Pour développer en Front-End j'utilise Visual Studio Code comme IDE. Il s'agit d'un éditeur de code sur lequel on peut ajouter diverses extensions pour simplifier le développement. J'utilise aussi le débogueur de Google Chrome qui permet de suivre l'exécution du code TypeScript.

Pour le Back-End j'utilise l'IDE IntelliJ qui est très pratique pour le Java, en particulier pour la qualité de son débogueur.

```
# z_cafpi_activite_client
# z_cafpi_logs
# z_dev_activite
# z_dev_activite_client
# z_dev_logs
# z_dev_need_action
# z_main_activite_client
# z_main_important
# z_main_logs
```

Nous utilisons un système de logs interne appelé « user action ». Il permet de stocker les logs en base de données et de les envoyer sur Slack en fonction de leur catégorie d'importance.

De cette façon nous avons un accès rapide à l'activité des utilisateurs et aux comportements anormaux, ce qui facilite le debug.

Figure 7 : Aperçu des canaux de « user action » sur Slack.

Nous utilisons GIT comme gestionnaire de versions et le code est stocké dans des projets privés sur GitHub auxquels nous avons accès. Pour gérer les commits et les branches j'utilise une extension de Visual Studio Code, elle permet d'effectuer la plupart des opérations très facilement et propose de visualiser les différences entre les commits.

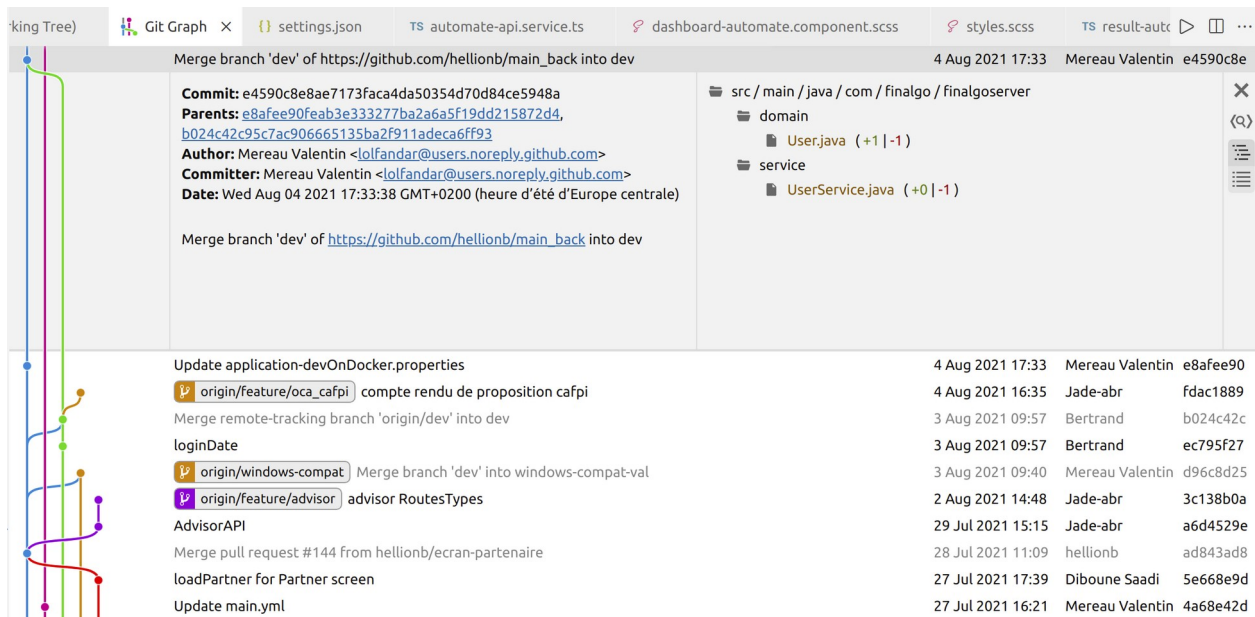


Figure 8 : Aperçu de l'extension Git Graph sur Visual Studio Code

Enfin, pour travailler sur les bases de données nous utilisons DBeaver, un logiciel libre avec une interface plutôt intuitive.

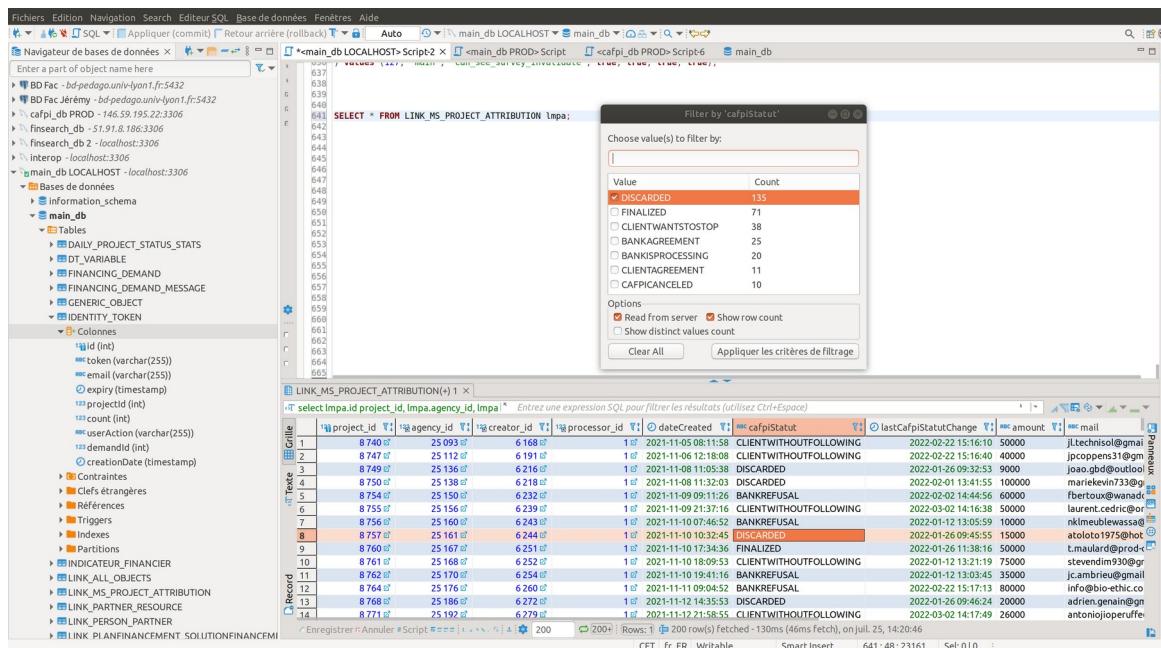


Figure 9 : Aperçu de l'interface de DBeaver



## 4 Travail réalisé

Cette année j'ai surtout travaillé sur l'amélioration de notre système de données, en plus de diverses missions d'ajout ou d'évolution des fonctionnalités sur nos applications.

### 4.1 Notre modèle de données

La plupart de nos objets métier ont les mêmes caractéristiques. Que ce soit un projet, un utilisateur ou une entreprise ; ils ont normalement un identifiant, un nom, un état de suppression, divers autres champs et une collection de propriétés sous une forme de clé-valeurs à deux niveaux. Nous appelons ces propriétés les « OCA variables », il en existe des centaines pour toute sorte d'informations communes à plusieurs objets.

Cette structure permet de stocker des propriétés comme le chiffre d'affaires d'une entreprise pour chaque année, ou son code NAF<sup>2</sup>.

```
id, #entreprise_id,      code_propriété, clé_propriété,  valeur
1,                  1,      "code_NAF",          -1, "96.04Z"
2,                  1, "chiffre_affaires",      2018, "427456"
3,                  1, "chiffre_affaires",      2019, "541471"
```

Figure 10 : Exemple du stockage de 3 OCA variables associées à une entreprise

Du point de vue des développeurs, cette structure se manipule comme une Map de propriété de Map de clé-valeur, ou bien simplement comme une Map de propriété-valeur pour les propriétés uniques. Il existe un Enum qui renseigne les propriétés et une classe abstraite qui permet de simplifier les opérations, il s'agit du design pattern *Façade*.

```
ocaVariablesEntreprise.addValue(Oca.code_NAF, "96.04Z");
... (Oca.chiffre_affaires, "2018", 427456);
... (Oca.chiffre_affaires, "2019", 541471);
```

Figure 11 : Exemple d'assignation de valeurs via la façade Java

```
{
  "code_NAF": {
    "-1": "96.04Z"
  },
  "chiffre_affaires": {
    "2018": 427456,
    "2019": 541471
  }
}
```

Figure 12 : Structure de données correspondante en syntaxe JSON

---

<sup>2</sup> Code à cinq chiffres permettant de qualifier l'activité professionnelle d'une entreprise



Ce modèle est efficace pour stocker nos centaines de propriétés sans rendre les tables ou les requêtes illisibles du côté de la base de données. Son implémentation possède cependant une forte dette technique que j'ai passé plusieurs mois à rembourser.

## 4.2 Problèmes de l'ancienne implémentation

En base de données, pour chaque objet métier il y avait une table pour l'objet et une table pour ses OCA variables soit une quinzaine de tables pour quelques centaines de milliers de lignes. Plusieurs tables d'OCA n'avaient pas de clé étrangère et il n'y avait aucun index, ce qui causait une latence sur de nombreuses requêtes. Aussi, certains objets avaient des clés étrangères dans leur table qui étaient en doublon avec nos tables de jointure.

Dans les méthodes de service, tous les mécanismes liés aux OCA variables n'étaient pas gérés par une façade ce qui était compensé par de la duplication de code avec des erreurs de mise à jour lors des évolutions.

Ensuite la création, la modification et la suppression des OCA variables étaient gérées par un service qui effectuait **un appel** à la base de données pour **chaque valeur** à modifier, en plus d'appeler des traitements spécifiques lors de l'enregistrement de certaines valeurs, ce qui brisait la chaîne de responsabilité.

Plusieurs services effectuaient également ce type d'appels **en boucle** pour divers traitements sur des collections d'objets. Cette erreur est liée à un manque de connaissance sur le fonctionnement des services en jeu et menait à des centaines voire des milliers d'échanges entre Spring et MySQL. Avec ces appels, nos API prenaient quelques secondes voire quelques minutes dans les pires cas.

Enfin, les clés étrangères en doublon étaient aléatoirement utilisées dans le code, ce qui portait à confusion sur la source des liens entre les objets.

## 4.3 Procédure de refactoring et migration des données

Pendant plusieurs mois j'ai mis en place un nouveau système de gestion de ces propriétés et j'ai passé les objets métier un à un sur ce nouveau système. Nous avons mis les applications en production après chaque nouvelle migration d'objet.

### a. Mise en place d'une base propre

L'un des objectifs de ma mission était de rassembler les différents objets métier dans une même table, ainsi que les différentes OCA variables dans une autre table. Cette nouvelle base a aussi la propriété de rendre l'identifiant de chaque objet unique, peu importe son type.

Dans un premier temps j'ai créé ces tables et ajouté le code permettant de les manipuler.

#### a.1 Création des tables

Pour la création des tables je me suis basé sur les tables existantes en ajoutant un champ pour la suppression, un pour l'ancien identifiant et un pour le type afin de différencier les différents objets.

J'ai également ajouté différents indexes sur les champs utilisés lors des requêtes pour améliorer les performances de celles-ci, en plus d'une contrainte d'unicité afin de garantir que les couples propriété-clé des OCA variables sont uniques pour un objet donné. Ce modèle ne respecte pas le concept de forme normale mais il a l'avantage d'être très facile à manipuler en SQL et lorsque nous effectuons la jointure sur les OCA variables nous avons systématiquement besoin de récupérer toutes les valeurs associées. C'est pourquoi nous avons décidé de garder ce modèle.

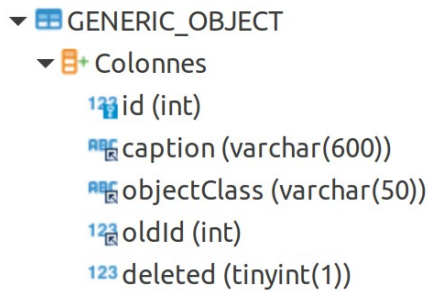


Figure 13 : Modèle de la table des objets métier

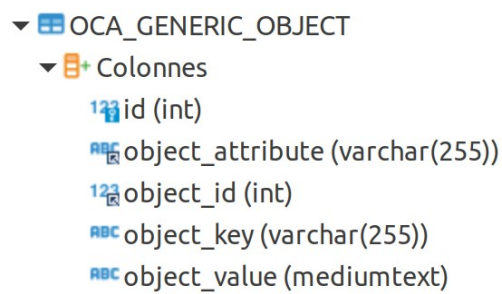


Figure 14 : Modèle de la table des OCA variables

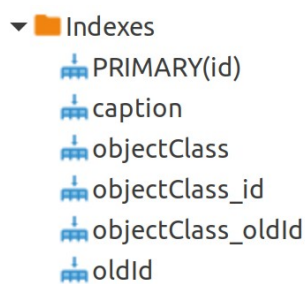


Figure 16 : Indexes de la table objet métier

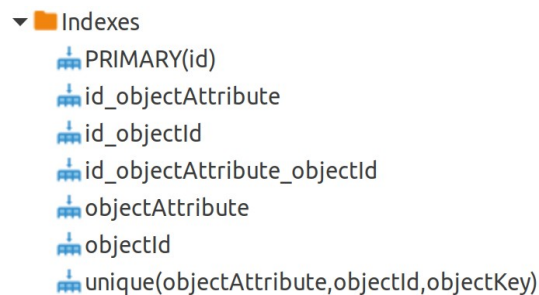


Figure 15 : Indexes de la table d'OCA variables

## a.2 Création des classes

Pour manipuler ces tables j'ai ajouté leur équivalent dans le code que j'ai lié à la base de données grâce aux annotations Spring / Hibernate. En particulier j'ai décrit que plusieurs types d'objets sont stockés dans la table GENERIC\_OBJECT via `@Inheritance` et que la colonne `objectClass` permet de les distinguer via `@DiscriminatorColumn`.

```

38  @Entity
39  @Table(name = "GENERIC_OBJECT")
40  @Inheritance(strategy= InheritanceType.SINGLE_TABLE)
41  @DiscriminatorColumn(
42      name = "objectClass",
43      discriminatorType = DiscriminatorType.STRING
44  )
45  public class GenericObject {
  
```

Figure 17 : Description du lien entre la classe Java et la base de données via Hibernate

J'ai également précisé que la mise à jour d'un objet devait être propagée aux données liées par la clé étrangère via l'attribut *cascade* ce qui permet d'automatiser la mise à jour des OCA variables.

```
58 @OneToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
59 @JoinColumn(name = "object_id")
60 protected List<OcaGenericObject> ocaGenericObjects;
```

Figure 18 : Propagation des mises à jour sur la clé étrangère

Enfin j'ai ajouté les DAO et Services liés aux nouvelles tables. Dans une optique de généralité, j'ai répertorié toutes les méthodes communes aux différents objets métiers ou à leurs services et je les ai ajoutés judicieusement dans les nouvelles classes.

## b. Préparation de la migration des objets

Afin d'effectuer la migration des données, il fallait d'abord uniformiser les objets de façon à ce qu'ils ne contiennent que les champs de la table `GENERIC_OBJECT`, c'est à dire un identifiant *id*, un nom *caption* et un état de suppression *deleted*. Les autres informations doivent être dans les OCA variables s'il s'agit de propriétés ou dans les tables de jointure s'il s'agit de clés étrangères.

En **base de données** j'ai déplacé plusieurs propriétés des tables d'objet métier vers leur table d'OCA variables, puis j'ai supprimé les colonnes devenues obsolètes ainsi que les colonnes de clés étrangères en doublons avec les tables de jointure.

Voir annexe [Nettoyage des tables en SQL](#).

Dans le **code Java** j'ai supprimé les mêmes attributs que dans la base de données, j'ai mis à jour les getters et les setters de ces attributs pour qu'ils atteignent la donnée depuis les OCA variables et j'ai mis à jour les requêtes nécessaires dans les DAO ainsi que les traitements de certains services suite au nettoyage des clés étrangères en doublon.

Voir annexe [Mise à jour des getters et setters en Java](#).

## c. Migration des objets sur le nouveau système de données

Cette étape était la plus sensible car elle a provoqué le changement de la plupart des identifiants des objets métiers dont beaucoup étaient stockés en dur dans différents systèmes.

### c.1 Migration des données

En **base de données** j'ai déplacé les données de chaque objet métier dans la nouvelle table ainsi que les OCA variables associées. J'ai ensuite procédé à la mise à jour de tous les identifiants et à rétablir les contraintes de clés étrangères avec les nouveaux identifiants. Enfin j'ai supprimé les anciennes tables d'objet métier et d'OCA variables.

Voir annexes [Migration des données en SQL](#) et [Mise à jour des identifiants et nettoyage des tables en SQL](#).

### c.2 Mise à jour du système de fichiers

Lorsque des utilisateurs envoient des documents sur nos plateformes, nous les stockons dans le système de fichiers du serveur en utilisant les identifiants des objets associés dans les noms des dossiers.

```
dossier_principal/entreprise/42/liasse_fiscale/liasse-2019.pdf
```

Figure 19 : Exemple de l'adresse d'une liasse fiscale pour l'entreprise d'identifiant 42

Étant donné qu'il n'est pas possible de prévoir les nouveaux identifiants avant la migration des données, pour chaque objet j'ai écrit un programme en langage bash pour récupérer automatiquement les nouveaux identifiants et renommer les dossiers concernés. À chaque mise en production nous avons exécuté ce programme après avoir effectué les migrations de données en SQL.

Voir annexe *Script de mise à jour des identifiants pour le système de fichiers*.

### c.3 Branchement des classes sur le nouveau système de données

Afin d'utiliser la nouvelle implémentation pour les objets métier j'ai dû effectuer de nombreuses opérations de refactoring.

Dans un premier temps j'ai supprimé les références aux anciennes tables de la base de données, j'ai modifié les déclarations des classes d'objet pour qu'elles étendent le nouveau type *GenericObject* et j'ai précisé la valeur à utiliser pour déterminer le type de l'objet depuis la table, celle du champ *objectClass*.

```
10 @Entity
11 @DiscriminatorValue("CUSTOMER_ENTITY")
12 public class CustomerEntity extends GenericObject {
```

Figure 20 : Nouvelle déclaration de classe pour l'objet métier d'un Cabinet

Toujours dans les objets métier, j'ai supprimé les liens restants avec les anciennes tables et tout le code devenu redondant avec l'héritage de *GenericObject*, c'est à dire les champs *id*, *caption*, *deleted*, *ocaVariables* ainsi que leurs accesseurs et diverses méthodes.

Ensuite j'ai modifié de nombreuses méthodes de service pour utiliser les DAO traditionnelles des objets métier lors de la persistance. Avant ça, la sauvegarde des OCA variables était gérée manuellement par un autre service qui effectuait divers traitements lors de la sauvegarde de certaines propriétés. J'ai dû déplacer ces traitements dans les services ou DAO appropriés dans la chaîne de responsabilité.

Durant cette étape j'ai réécrit beaucoup de méthodes de façon à minimiser les appels à la base de données, c'est à dire utiliser des requêtes pour traiter des groupes d'objets au lieu de boucler sur des traitements qui effectue des requêtes pour un seul objet. Le nombre d'appels de ces méthodes à la base de données était généralement de l'ordre de  $k*n$  avec  $k$  le nombre de requêtes différentes utilisées par la méthode et  $n$  le nombre d'objets dans la collection, entre 1 et plusieurs milliers. À chaque fois il était possible de faire le même traitement avec seulement  $k$  appels. Ce type de refactoring a parfois nécessité la réécriture de blocs de traitement entiers et l'ajout de requêtes spécifiques dans les DAO. Le temps de réponse des API concernées a diminué d'un facteur

de plusieurs dizaines, ce qui permet un temps de réponse compris entre quelques dixièmes et maximum quelques secondes pour les API les plus longues.

Voir annexe Minimisation des appels à la base de données dans une méthode Java.

Suite à ces opérations l'ancien service des OCA variables et diverses classes liées à l'ancienne implémentation sont devenues obsolètes, j'ai pu les supprimer.

## **d. Retours sur la mission**

Cette mission était la plus complexe que j'ai réalisée à Finalgo et je suis fier de l'avoir réussie malgré quelques incidents.

Je suis satisfait du résultat. Nos applications sont beaucoup plus performantes depuis le refactoring, certains bugs étranges ont disparu et les deux nouvelles tables sont très pratiques. En particulier nous pouvons facilement effectuer des requêtes sur les données des OCA variables dont différents objets sont concernés, et nous pouvons retrouver le type des objets dans les résultats. Ces requêtes sont particulièrement utiles lors des débuts.

Notre phase de conception nous a permis d'avoir une vision claire de la modélisation à obtenir ainsi que des grandes étapes à suivre. Ces étapes m'ont beaucoup guidé mais nous n'avons pas poussé la réflexion suffisamment loin en ce qui concerne les changements d'identifiants. Lors des premières mises en production nous avons oublié divers mécanismes qui utilisaient des identifiants non récupérés depuis la base de données, ce qui a nécessité plusieurs hotfix. Une procédure de tests automatisés aurait permis d'éviter ce type de problèmes, il s'agit d'une tâche qui sera réalisée au cours des prochains mois.

La plupart des difficultés que j'ai rencontrées étaient liées à une mauvaise implémentation des design patterns en jeu. Les responsabilités n'étaient pas réparties correctement au sein des différents services, plusieurs traitements manquaient dans la façade et il existait un fort couplage entre divers services et objets qui n'aurait pas dû exister.

## Conclusion

Mon ressenti sur l'alternance est très positif. J'ai pu approfondir mes connaissances en Java, en algorithmique, en bases de données et en toutes sortes de technologies du web liées à mes autres missions. Les différentes missions que j'ai réalisées m'ont aidé à mieux comprendre le contenu de plusieurs de mes cours, et inversement.

J'ai également appris de nombreuses bonnes pratiques. Au delà des technologies j'ai mis en place une démarche quotidienne de recherches, d'étude des documentations, de tests, de débogage et de travail d'équipe pour avancer dans mes tâches. Mon code est devenu plus facile à lire, mieux organisé et j'ai pris l'habitude de correctement documenter mon travail avec les outils à ma disposition.

Mon travail de refactoring m'a appris beaucoup sur les possibilités de connexions entre la base de données et les objets dans Spring, sur l'importance de la phase de conception, sur la pertinence des logs ou messages d'erreurs et sur ma capacité à réfléchir sur des algorithmes. Suite à cette mission nous allons réaliser d'autres étapes de refactoring dans le but de réduire notre dette tout en continuant le développement de nos applications.

Au cours des prochains mois je vais travailler sur une mission dont le but est de mettre en place Redux sur nos plateformes. Il s'agit d'une librairie de stockage de données basée sur le design pattern Observer, elle permettra d'alléger fortement la sollicitation de notre back-end et de corriger divers problèmes de cohérence des données en front-end. Je vais également travailler sur l'ajout de tests automatisés dans notre procédure d'intégration continue, ce qui permettra d'économiser du temps et de limiter fortement les régressions.

Enfin, avec l'accord de Finalgo j'envisage de réaliser des vacances en tant que professeur d'algorithmique et de Java à l'IUT informatique de la Doua. Il s'agit d'un projet de longue date que j'aimerais commencer à partir de septembre 2022.

# Annexes

## Nettoyage des tables en SQL

```
460 -- DÉBUT PARTNER
461 -- Suppression des colonnes déjà obsolètes
462 alter table PARTNER
463     drop column logo,
464     drop column config_file,
465     drop column object_creation_date,
466     drop column configuration_file,
467     drop column object_modification_date,
468     drop column contact;
469 -- Déplacement de description vers les OCA
470 insert into OCA_PARTNER (id, object_attribute, object_id, object_key, object_value)
471     select null, 'description', id, -1, description from PARTNER where description is not null and description <> '';
472 -- Déplacement de archaic vers les OCA
473 insert into OCA_PARTNER (id, object_attribute, object_id, object_key, object_value)
474     select null, 'archaic', id, -1, archaic from PARTNER where archaic is not null;
475 -- Déplacement de priority vers les OCA
476 insert into OCA_PARTNER (id, object_attribute, object_id, object_key, object_value)
477     select null, 'priority', id, -1, priority from PARTNER where priority is not null;
478 -- Suppression des nouvelles colonnes obsolètes
479 alter table PARTNER
480     drop column description,
481     drop column archaic,
482     drop column priority;
```

## Mise à jour des getters et setters en Java

```
24 public String getPostalCode() {
25     return getOcaVariables().getValueTypedString(Oca.postal_code);
26 }
27
28 public void setPostalCode(String postalCode) {
29     getOcaVariables().addValue(Oca.postal_code, postalCode);
30 }
```

## Migration des données en SQL

```
483 -- Remplissage des partenaires de GENERIC_OBJECT
484 insert into GENERIC_OBJECT (id, caption, objectClass, oldId, deleted)
485     select null, nom, 'PARTNER', id, deleted from PARTNER;
486 -- Remplissage des oca des partenaires de OCA_GENERIC_OBJECT
487 insert into OCA_GENERIC_OBJECT (id, object_attribute, object_id, object_key, object_value)
488     select null, op.object_attribute, (
489         -- Sélection du nouvel object_id
490         select go2.id from GENERIC_OBJECT go2 where objectClass = 'PARTNER' and oldId = op.object_id
491     ) object_id, op.object_key, op.object_value
492     from OCA_PARTNER op;
```



## Mise à jour des identifiants et nettoyage des tables en SQL

```
493 -- Mise à jour du partner_id dans FINANCING_DEMAND
494 alter table FINANCING_DEMAND drop foreign key FKkgm6kbjowse4dq5y75433dj57;
495 update FINANCING_DEMAND fd set fd.partner_id = (
496     select go2.id
497     from GENERIC_OBJECT go2
498     where go2.objectClass = 'PARTNER'
499     and fd.partner_id = go2.oldId
500 );
501 alter table FINANCING_DEMAND add constraint fk_financing_demand_partner foreign key (partner_id) references GENERIC_OBJECT(id);
502 -- Mise à jour du partner_id dans TYPEFINANCEMENT
503 alter table TYPEFINANCEMENT drop foreign key FK9cvq8bloi3furexqdwdp6cv4l;
504 update TYPEFINANCEMENT tf set tf.partner_id = (
505     select go2.id
506     from GENERIC_OBJECT go2
507     where go2.objectClass = 'PARTNER'
508     and tf.partner_id = go2.oldId
509 );
510 alter table TYPEFINANCEMENT
511 drop foreign key FKt9eq42n25wlrmbjobs8ah23x,
512 drop foreign key FKmbykv2c5gophm12h7r4mtos5i,
513 drop foreign key FKml6mrwl3cy4reiv5x533vd2rf,
514 add constraint fk_typefinancement_partner foreign key (partner_id) references GENERIC_OBJECT(id);
515 -- Mise à jour du partner_id dans LINK_PERSON_PARTNER
516 alter table LINK_PERSON_PARTNER drop foreign key FK20muf0wh4fmfg3yqdfs1jrv6j;
517 update LINK_PERSON_PARTNER lpp set lpp.partner_id = (
518     select go2.id
519     from GENERIC_OBJECT go2
520     where go2.objectClass = 'PARTNER'
521     and lpp.partner_id = go2.oldId
522 );
523 alter table LINK_PERSON_PARTNER add constraint fk_link_person_partner_partner foreign key (partner_id) references GENERIC_OBJECT(id);
524 -- Mise à jour du partner_id dans LINK_PERSON_PARTNER
525 alter table LINK_PARTNER_RESOURCE drop foreign key FK0mwbd75671bc1qyko068vod7y;
526 update LINK_PARTNER_RESOURCE lpr set lpr.partner_id = (
527     select go2.id
528     from GENERIC_OBJECT go2
529     where go2.objectClass = 'PARTNER'
530     and lpr.partner_id = go2.oldId
531 );
532 alter table LINK_PARTNER_RESOURCE add constraint fk_link_partner_resource_partner foreign key (partner_id) references GENERIC_OBJECT(id);
533 -- Suppression des tables
534 drop table PARTNER, OCA_PARTNER;
535 -- FIN PARTNER
```

## Script de mise à jour des identifiants pour le système de fichiers

```
main_back > src > main > resources > shell_updates > $ 2022-03-16-migration_company.sh
1  #!/bin/bash
2
3  # MIGRATION ENTREPRISE FILESYSTEM
4
5  cd /opt/finalgo_files/main
6  cp -r company company_before_migration_backup
7  cd company
8  echo "select 'mv', oldId, id from GENERIC_OBJECT where objectClass = 'COMPANY';" > migration_company.sql
9  mysql -u main_user -p main_db --protocol=tcp < migration_company.sql > migration_company.sh
10 chmod u+x migration_company.sh
11 ./migration_company.sh
12
```

## Minimisation des appels à la base de données dans une méthode Java

```
207 public List<ProjectBean> loadMsProjects(List<Integer> projectIds) {
208     // **** INTERDIT : performances catastrophiques ****
209     // List<LinkMsProjectAttribution> links = new ArrayList<>();
210     // for (Integer projectId : projectIds) {
211     //     links.add(linkMsProjectAttributionDAO.findOne(projectId));
212     // }
213     List<LinkMsProjectAttribution> links = linkMsProjectAttributionDAO.findAllByIds(projectIds);
```