



Rapport de stage
Concepteur Développeur Informatique
de Franck Canonne
(étudiant à l'ENI Quimper)

(Stage chez VETOSOFT du 18/06 au 10/08 2018)





Remerciements

Je souhaite remercier la société Vetosoft de m'avoir accueilli au sein de son service informatique pendant la durée de mon stage "Concepteur Développeur Informatique".

Je remercie Simon Perigault, mon tuteur dans l'entreprise, Florian Garcia (développeur PHP / JavaScript) ainsi que l'ensemble de l'équipe de Vetosoft pour m'avoir fait confiance et permis de vivre un stage passionnant et enrichissant.

Enfin, je remercie mon responsable de formation à l'ENI, Anthony Cosson, ainsi que les formateurs (tout particulièrement Adrien, Emmanuel, Jonathan et Guillaume) qui m'ont apporté beaucoup tant au niveau développement qu'au niveau "bonnes pratiques".

La formation "Concepteur Développeur Informatique" de l'ENI est épuisante, mais pour qui veut s'en donner la peine, la qualité des cours et les compétences des enseignants rendent cela passionnant et motivant.

Merci à tous.





Abstract

I did my internship at Vetosoft.

The creation of this startup follows an observation: veterinary medicine must adapt to the new economic context, by equipping itself with innovative tools. Vetosoft has therefore produced dairy cattle management software for use by veterinarians. This software (Milkup) has no competition in the European market.

I am an "old self-taught PHP developer" (that's a private joke from Emmanuel, one of our teachers at ENI school), and this training allowed me to correct a lot of bad development practices.

Vetosoft's request for this internship was the realization of a module to integrate with their Milkup software. This module, which I finalized during these two months, was intended to evaluate the opportunity to reform a cattle according to various criteria, so as not to reduce the turnover of the operator.

This internship and this training brought me a lot, both in computer development and humanly. I thank ENI, Anthony Cossen and my fellow trainers for that.



Table des matières

Remerciements.....	3
Table des matières.....	7
La startup Vetosoft.....	8
Le logiciel Milkup.....	9
Environnement de travail.....	10
Technologies utilisées.....	11
Le choix des frameworks et librairies.....	13
Descriptif de la demande.....	16
<i>La problématique de l'éleveur.....</i>	16
<i>La solution proposée.....</i>	17
<i>Planning prévisionnel.....</i>	18
Réalisation du projet.....	19
<i>Persistence des données et mobilité.....</i>	20
<i>Développement métier.....</i>	25
<i>Pourquoi utiliser JavaScript ?.....</i>	28
<i>POO (programmation orientée objet).....</i>	29
<i>L'internationalisation (i18n).....</i>	30
<i>L'interface homme/machine (IHM).....</i>	31
Tests.....	33
Retro planning.....	35
Conclusion.....	36

La startup Vетosoft

Vetosoft est une startup fondée en 2016 par trois personnes :

Karen Noteris (CEO)

Gaëtan Mabille (COO) - vétérinaire

Simon Perigault (CIO) - développeur informatique



Karen Noteris
CEO

Passionnée par l'échange et le partage, nourrie de sa formation en sciences humaines, après dix années d'enseignement, Karen Noteris élargit son horizon en embarquant dans l'aventure VETOSOFT.



Gaëtan Mabille
COO

Vétérinaire depuis une dizaine d'années, Gaëtan Mabille a investi la rurale dans toutes ses dimensions, explorant la reproduction, l'alimentation des bovins. Frustré, par les contraintes de terrain, toujours plus motivé par l'envie d'améliorer son offre de service, et face au constat du manque d'outils pratiques pour lui faciliter la tâche, il décide de se lancer dans l'aventure VETOSOFT afin de créer des outils efficaces.



Simon Perigault
CIO

Passionné d'informatique, Simon apprend à faire de la programmation informatique dès son plus jeune âge. Autodidacte, il enchaîne les projets de développement et crée sa propre entreprise. A sa rencontre avec Gaëtan, il découvre le monde vétérinaire, ils décident de se lancer dans la concrétisation d'un logiciel efficace, modulable et personnalisable, MilkUp !

La création de cette startup fait suite à un constat : la médecine vétérinaire doit s'adapter au nouveau contexte économique, en s'équipant d'outils innovants. Vetosoft a donc réalisé un logiciel de gestion d'élevage bovin laitier à l'usage des vétérinaires. Ce logiciel n'a pas de concurrence sur le marché européen.

Le fait que Gaëtan Mabille soit lui-même vétérinaire spécialiste des bovins laitiers apporte une expertise particulière permettant de penser le logiciel du côté de l'utilisateur pour lui apporter un outil performant et efficace (du Agile++ en quelque sorte).

Le logiciel MilkUp

Dans le cadre de ce stage de fin de formation "Concepteur Développeur Informatique", j'ai développé un module spécifique pour l'application propriétaire MilkUp afin de gérer statistiquement la pertinence de réformer des vaches afin de faire monter le chiffre d'affaire de l'exploitation.

MilkUp est une application de gestion d'élevage bovins laitiers destinée aux vétérinaires. Les technologies utilisées par MilkUp sont PHP7 à travers un framework propriétaire, MySQL avec un ORM maison, CSS avec Bootstrap, et JavaScript avec jQuery, Moment.js, DataTables...

Milkup est une application n-tiers (Simon Périgault le créateur du framework a délibérément choisi de ne pas utiliser une structure MVC pour simplifier l'architecture du logiciel). L'ORM gérant la partie DAO, les requêtes vers la base de données, sont intégrées dans la BLL. Ceci s'avère très confortable à l'usage. L'IHM est quand à lui géré en JavaScript et en Bootstrap.

Le principal inconvénient d'utiliser un framework propriétaire est qu'il faut tout réapprendre (principes d'utilisations, syntaxe, nommage...) et qu'il n'y a aucun support en ligne comme on peut l'avoir avec Symfony, Zend ou Laravel. En contre partie, le framework ayant été réalisé spécifiquement pour MilkUp, celui-ci est parfaitement adapté aux fonctionnalités voulues. En outre il est beaucoup plus léger que Symfony et l'ORM est assez efficace.



Environnement de travail

Pour ce stage, il a été convenu avec Vetosoft que je travaille sur mon ordinateur personnel (ça arrangeait tout le monde). Ceci apporte de nombreux avantages. Pas de temps d'adaptation, possibilité de travailler en dehors des horaires du stage. Et pour ma part, possibilité de travailler sous Linux.

J'ai donc utilisé mon ordinateur personnel sous Xubuntu 16.04 LTS, avec comme environnement de travail les logiciels Visual Studio code, Chrome, DataGrip, Discord, GanttProject, Git, JEdit, et Libre Office.



Cet ordinateur dispose en outre d'une machine virtuelle Windows 10 sous VMWare disposant de la suite Microsoft Office, de la suite Adobe Creative et de divers logiciels pour le développement ou le graphisme.

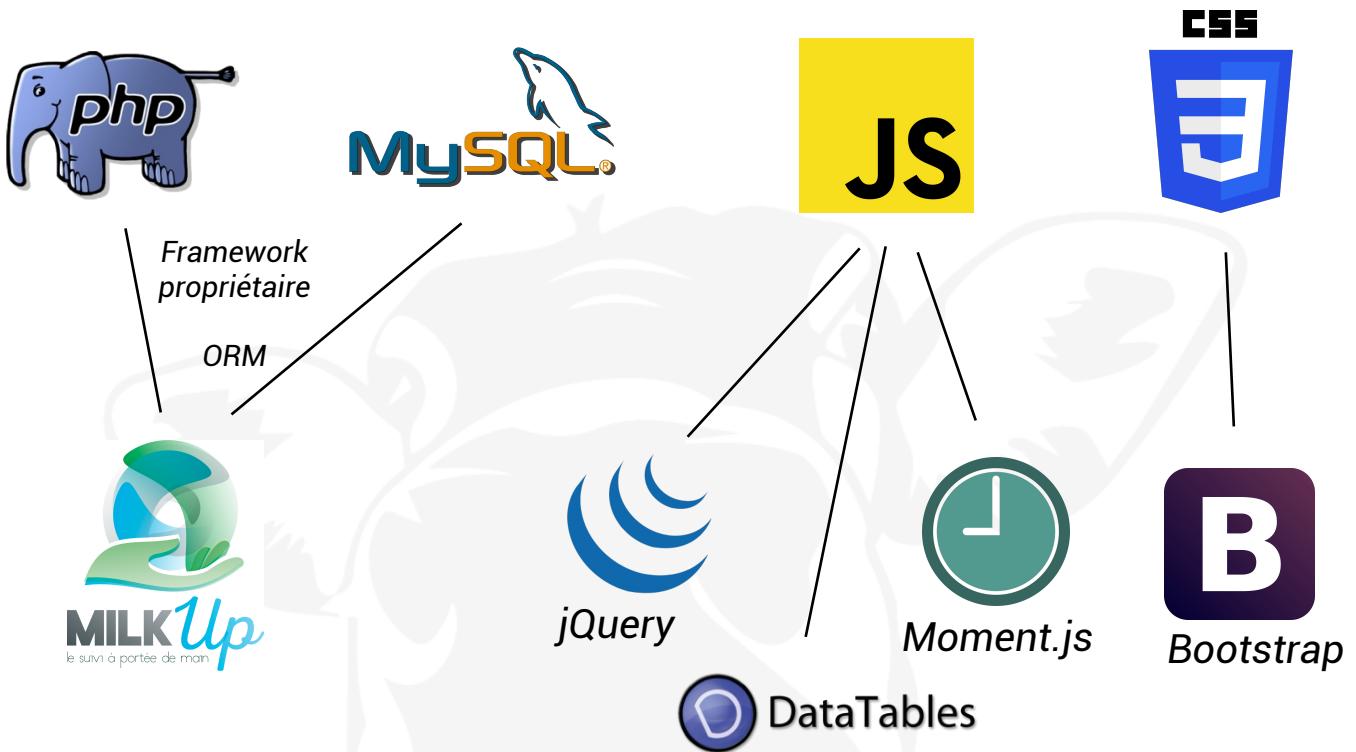
Il est évident que de travailler dans un environnement familier est un vrai plus, surtout pour un adepte des logiciels Libres (même si Visual Studio code, DataGrip, VMWare et Chrome ne sont pas libres).

L'IDE étant l'outil du développeur, il est important bien connaître celui que l'on utilise. J'utilisai pour ma part PHPStorm de JetBrains avant ce stage, mais comme les développeurs chez Vetosoft utilisent Visual Studio Code de Microsoft, j'ai fait le choix de m'adapter à ce très bon IDE dans un souci d'uniformité... Je ne suis pas déçu de ce choix, l'IDE étant peut être moins bon pour PHP, mais meilleur pour JavaScript, ce logiciel permet de n'utiliser qu'un seul IDE pour tout le développement.

Technologies utilisées

L'application Milkup est développée en PHP 7 / MySQL / JavaScript / CSS. Pour la partie IHM (interface homme / machine), le choix s'est porté sur l'utilisation du framework CSS Bootstrap 4.

Les frameworks JavaScript jQuery et jQuery UI sont aussi utilisés ainsi que quelques librairies (DataTables, Moment.js...).



Simon Perigault a délibérément fait le choix de créer son propre framework PHP propriétaire.

Il a pour ce faire développé un modèle de templating en JavaScript, un ORM complet ainsi que diverses librairies comme par exemple VText pour l'internationalisation, une librairie pour gérer les formulaires, une librairie pour envoyer des SMS...

Ce framework bien que très complet est relativement aisé à appréhender et plutôt confortable à l'usage. Il permet (et c'est bien là le but d'un framework) de gagner beaucoup de temps de développement ainsi qu'une maintenabilité du code efficace.

On peut se demander pourquoi se compliquer la tâche en créant un

framework maison plutôt que d'utiliser un framework qui a largement fait ses preuves, tel que Symfony, Laravel ou Lumen (sorte de Laravel light) avec Doctrine ORM. La réponse de Simon Perigault est qu'il avait besoin pour cette grosse application n-tiers qu'est MilkUp d'un outil simple, maîtrisé et sans un tas de fonctionnalités inutiles pour ce projet. Cela permet au final d'avoir un outil relativement simple à prendre en main, et facilement modifiable selon les besoins (j'ai par exemple formulé une demande spécifiques à propos de l'ORM, une modification de l'ORM a été faite très rapidement pour régler cette demande, ce qui aurait été possible mais certainement beaucoup plus complexe avec un framework non propriétaire).

De plus ce framework (il n'a pas de nom) permet de faire une application n-tiers à fenêtre unique, ce qui facilitera ensuite le passage de cette application sous Apache Cordova pour une expérience mobile optimisée.

L'utilisation systématique de Git pour le travail collaboratif (deux développeurs et deux développeurs stagiaires) s'avère très efficace et pratique (bien plus que SVN, utilisé à l'école lors de la formation).

Le choix des frameworks et librairies

Comme je l'ai déjà précisé, l'utilisation d'un framework maison est quelque peu déstabilisant au départ. En effet, lors de la formation, nous avons étudié Symfony 3 et j'ai en marge de la formation étudié Laravel. Néanmoins, dans le cas de Vetosoft, ce choix est très pertinent. Mais comme ceci était imposé (application existante), je tenais quand même à évoquer le choix d'un framework et de bibliothèques lors du lancement d'un nouveau projet.

L'intégration d'un framework ou d'une bibliothèque est un choix important et impactant pour une application. Rappelons que dans le développement logiciel, l'expérience s'accumule au fil des succès mais aussi des échecs. En effet, le développement logiciel reste un processus d'apprentissage et le code n'est qu'un effet de bord (traduction mot à mot de l'anglais « side effect », dont le sens est plus proche d'effet secondaire).

Un projet comporte très souvent des fonctionnalités déjà développées par vos pairs et il est intéressant de les réutiliser au lieu d'essayer de réinventer la roue (combien de fois un développeur, qui plus est en formation, peut entendre ceci ?). Que vous adoptiez un framework ou une bibliothèque, les deux formats sont du code maintenu par d'autres développeurs. Il est potentiellement bien écrit et mieux pensé car il couvrira des cas limites auxquels vous ne penseriez pas. Un bon framework ou une bonne bibliothèque devrait vous permettre de vous focaliser sur la vraie valeur attendue par votre métier.

Une bibliothèque est un ensemble de fonctionnalités qui se concentre sur un besoin bien précis et ne fera que ça. En général, le code d'une bibliothèque reste relativement petit et maîtrisable.

Un framework portera l'accent surtout sur la réutilisation de sa conception plutôt que sur la réutilisation d'une de ses fonctionnalités. Rappelons qu'un framework, par définition, est un "cadre de travail". Il se traduit par des règles, des conventions, des suppositions et des choix techniques. Cela va vous contraindre à développer d'une certaine manière au profit d'une productivité accrue, surtout lorsque vous démarrez un nouveau projet.

Néanmoins, dans les deux cas, la réutilisation de code via une bibliothèque ou un framework oblige les développeurs à se documenter sur la manière



dont on les utilise. C'est une complexité qui sera transmise aux autres développeurs. C'est pourquoi, il est important de bien analyser le besoin.

Au début, votre choix peut vous sembler être une bonne idée, mais gardez en tête que vous êtes rarement seul(e) à travailler sur une application et que le gain doit se calculer sur la durée (du développement

à la maintenance). Réutiliser du code peut booster le démarrage d'un projet, mais pourrait ralentir ceux qui reprendront le code derrière vous.

C'est notamment le cas quand il s'agit d'un framework maison, qui même s'il est de très bonne qualité (c'est le cas du framework créé pour Milkup par Simon Perigault) sera sans doute plus faiblement documenté mais bénéficiera de mises à jour plus pertinentes et quasi "sur mesure" (à la demande des développeurs). En outre les mises à jours seront moins contraignantes car elles apporteront un réel plus au framework, et ne l'encombreront pas de fonctionnalités inutiles au projet.

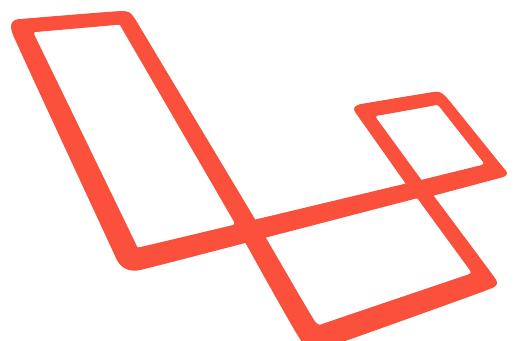
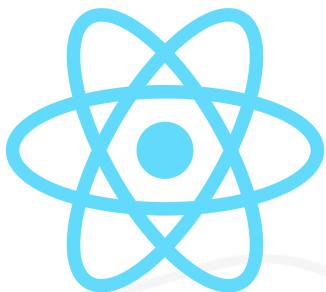
Le choix devra prendre en compte certaines contraintes au niveau du staff technique : taux de turn-over de l'équipe, expérience, taille de l'équipe...

En effet le choix sera très différent pour une équipe de quatre développeurs avec un faible turn-over que dans une équipe de quinze développeurs expérimentés avec un fort turn-over. Dans le premier cas, un framework maison semble une bonne solution, dans le second cas, l'utilisation d'un framework reconnu et très pratiqué (comme Symfony en PHP) semble plus approprié. En effet beaucoup de développeurs sur le marché du travail connaissent déjà un ou plusieurs framework.

La popularité d'une bibliothèque ou d'un framework est aussi un facteur à prendre en compte. Un framework "populaire" disposera d'une communauté active et d'une documentation exhaustive à même de vous aider et de répondre à vos questions. Sans compter que votre problème aura de fortes chances d'avoir été déjà soulevé et qu'une réponse est sans doute déjà sur le web. Attention toutefois à ne pas tomber dans l'effet de mode et focalisez vous plutôt sur le besoin.

Pour faire son choix, il ne faut pas non plus hésiter à tenir compte des retours d'expérience des autres développeurs.

L'un des points à prendre en compte avec beaucoup d'intérêt est la documentation. C'est d'ailleurs souvent le point d'entrée pour aborder une bibliothèque ou un framework. Un utilisateur final n'a pas besoin de



connaître tous les rouages internes des fonctionnalités qu'il utilise. Il est cependant préférable qu'il puisse accéder à une documentation complète en cas de besoin. De plus, la présence en ligne d'exemples et de tutoriels est un réel plus pour se familiariser à un nouvel outil.

Certains frameworks "populaires" disposent aussi d'une intégration avancée (sous forme d'extensions ou de plugins) aux IDE. Symfony et Laravel, par exemple sont très bien intégrés à PHPStorm ou à Visual Studio Code. Il y a fort à croire que ce ne sera pas le cas d'un framework propriétaire.

Je n'ai ici évoqué que les frameworks PHP, mais pour une application professionnelle, il y a de grandes chances que vous utilisiez aussi un framework JavaScript, un framework CSS, une multitude de bibliothèques, un ORM et que sais-je encore... Ces choix sont souvent définitifs, il est donc très important de passer du temps pour faire les bons choix en fonction de vos contraintes. Vos choix auront un impact sur les autres développeurs (et sur vous dans le futur), il est donc important d'intégrer l'équipe à ces décisions. De plus, dans une grosse structure, il faudra justifier ces choix à sa hiérarchie ou à son client, il est donc préférable d'y passer du temps en amont pour ne pas se tromper. Par professionnalisme, il faut avant tout défendre l'intérêt de notre client plutôt que le désir d'essayer des technologies afin de remplir notre CV.

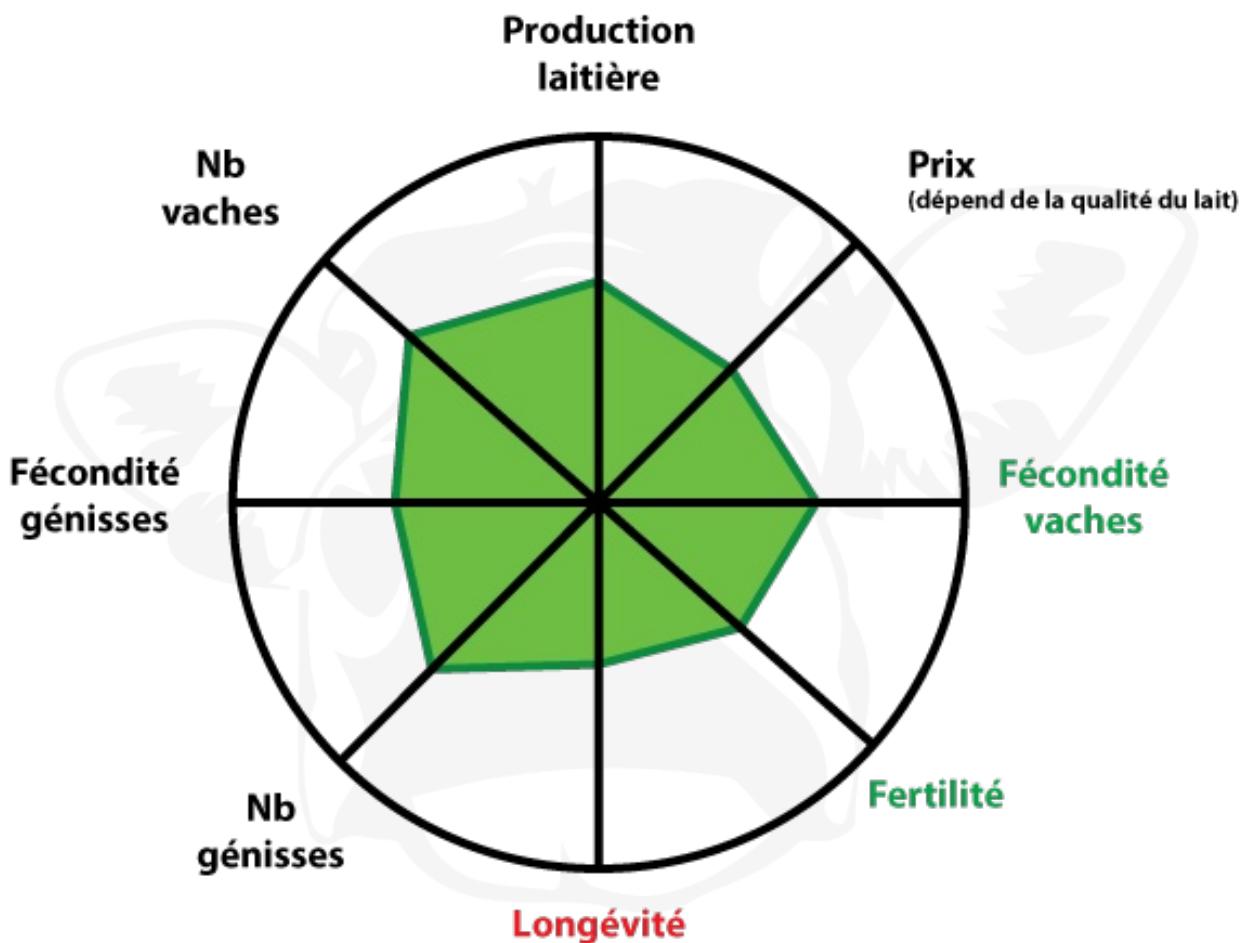


Il ne faut toutefois pas oublier qu'un développeur doit être à même de s'adapter à de nouveaux langages et donc à plus forte raison à de nouveaux frameworks.

Descriptif de la demande

Mon projet pour ce stage était d'ajouter à MilkUp un module de données et de calculs de statistiques en temps réel depuis l'élevage sur une tablette ou même un smartphone pour évaluer et expliquer à l'éleveur l'opportunité (ou pas) de réformer une vache en fonction de divers critères.

La problématique de l'éleveur



Le chiffre d'affaire de l'exploitation (représenté en vert sur schéma ci-dessus) dépend de 8 critères principaux. Si l'éleveur n'a quasiment pas de levier sur le prix du lait (hormis la qualité du lait, mais il s'agit là d'un autre module Milkup en développement) il peut par contre tenter d'ajuster les autres critères en fonction des statistiques de son troupeau.

Grace aux données recueillies sur l'élevage (production laitière, anomalies, taux butyreux, taux protéique, moyenne de lactation par velage...), le vétérinaire pourra conseiller l'éleveur quant à l'opportunité de se séparer

d'un bovin dont il considère le rendement insuffisant par rapport au reste du troupeau.

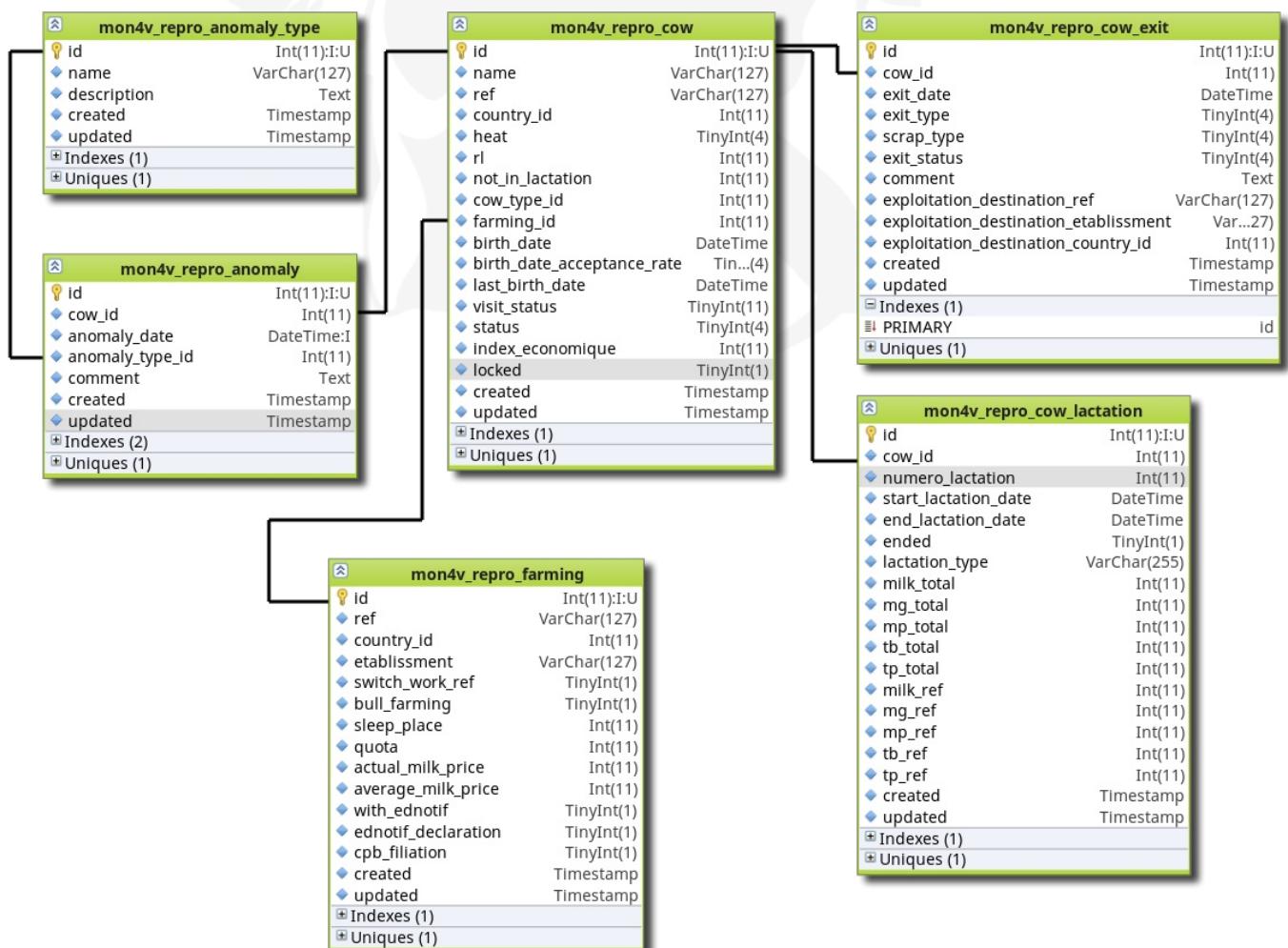
Il pourra aussi mettre le doigt sur des problèmes inhérents à l'élevage entier et sera ainsi à même de proposer des solutions.

A moi donc d'aller chercher dans la base de données Milkup tous les éléments nécessaires et de faire les calculs adéquats afin de sortir les chiffres et les statistiques demandées pour proposer un affichage le plus pertinent possible. Je suis pour cela briffé par Gaëtan Mabille, le vétérinaire de chez Vetosoft.

La solution proposée

La matière dont je dispose pour mener à bien ce défi est la base de données MySQL de Milkup. Pour le développement, nous travaillons en local sur nos machines et synchronisons régulièrement notre travail avec le serveur de pré-production grâce à Git. Nous disposons en local d'un jeu de test en BDD MySQL

La base de données comporte 98 tables sans aucune clé étrangère. Celles-ci sont gérées directement par l'ORM de Milkup.



Cette solution (pas de foreign key) a été choisie pour des questions d'optimisation. En effet il semblerait que la gestion par l'ORM soit plus rapide que la gestion par clés étrangères sur un moteur de bdd InnoDB.

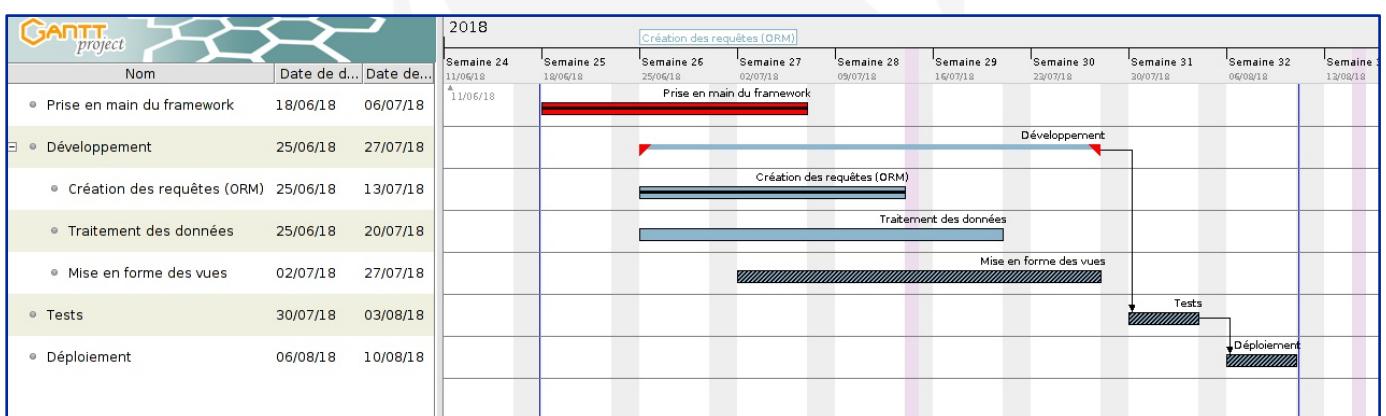
Pour mon projet, j'utilise essentiellement six tables de la base de données (voir diagramme ci-dessus disponible en annexe). Les jointures visibles sur le diagramme sont uniquement à titre indicatif (ce sont les jointures que nous précisons dans nos requêtes à l'ORM. L'ORM peut gérer un nombre de jointure illimité et jusqu'à trois niveaux jointures).

La partie BDD étant gérée par l'ORM du framework, la partie templating aussi (JavaScript, AJAX et Bootstrap), le gros du travail est la partie métier qui elle est en full PHP 7 (même si certains calculs peuvent être réalisés en JavaScript).

Ce module sera déployé sur sur Milkup lors de sa prochaine mise à jour majeure.

Milkup étant soumis aux contraintes du copyright, tous les fichiers ainsi que la base de données enregistrés sur mon ordinateur devront être supprimés suite à la soutenance de stage.

Planning prévisionnel



Voici le planning prévisionnel de cette mission : Beaucoup de tâches sont simultanées. En effet la prise en main du framework se fait au cours du projet et le développement étant Full Stack, les parties DAO, BLL et IHM se font en même temps. La phase de test fait suite au développement et le déploiement sur le serveur pré-production (sur le serveur de production lors de la prochaine mise à jour majeure) suite aux tests.

Réalisation du projet

Pour présenter mes réalisations et vous permettre d'évaluer au mieux mes compétences, j'ai choisi de présenter mon travail en trois phases :

- 1 : la partie persistance de données.*
- 2 : la partie purement métier.*
- 3 : la partie IHM (interface homme / machine).*

Il est bien entendu que je pars sur les bases d'une application existante avec une base de données existante. Ceci impose quelques contraintes et frustrations, et limite aussi la créativité. C'est pourquoi la partie métier, celle où j'ai pu m'exprimer totalement, sera plus complète que les deux autres.

Il est plus facile de concevoir une application en partant d'une page blanche, mais c'est très intéressant de s'imprégner du code d'autres développeurs, et de s'adapter aux pratiques d'une équipe.

Chaque équipe est différente et c'est très enrichissant de s'adapter. Chez Vetosoft, la spécificité est donc le framework et son ORM, les exigences sont essentiellement sur le nommage, le fait de tout commenter et de commenter uniquement en anglais et l'utilisation intensive de git.



Persistance de données et mobilité

La problématique de Milkup est qu'il utilise une grosse base de données, mais qu'il doit pouvoir fonctionner sur une tablette, et surtout éventuellement hors connexion (le vétérinaire saisit les données de chaque vache lors de sa consultation directement sur le lieu de consultation).

Simon Perigault a donc développé un système de cache qui enregistre temporairement les données sur la mémoire de la machine mobile et qui les synchronise avec la base de données dès que l'appareil se reconnecte. Ce procédé pouvant se révéler à risque, chaque enregistrement dans la base est inscrit dans les logs.

Pour ce qui est de la communication avec cette base de données, on utilise donc un ORM perso qui a ses avantages quant à l'écriture et qui est très complet au niveau des fonctions SQL disponibles (ISNULL, NVL, COALESCE, IFNULL, DATEDIFF, NOW, CURDATE, DATE_FORMAT, EXTRACT, DATE_ADD, DATE_SUB, CURTIME, 'UPPER', 'LOWER', 'LENGTH', 'ROUND', 'ABS', 'ACOS', 'ASIN', ATAN, ATAN2, CEIL, CEILING, CONV, COS, CRC32, DEGREES, EXP, FLOOR, LN, LOG, LOG10, LOG2, MOD, PI, POW, POWER, RADIANS, RAND, 'ROUND', 'SIGN', SIN, SORT, TAN, TRUNCATE, 'ASCII', BIN, BIT_LENGTH, CHAR, CHAR_LENGTH, CHARACTER_LENGTH, CONCAT, CONCAT_WS, ELT, EXPORT_SET, FIELD, FIND_IN_SET, FORMAT, FROM_BASE64, HEX, INSTR, LCASE, LEFT, LENGTH, LOAD_FILE, LOCATE, LOWER, LPAD, LTRIM, MAKE_SET, MID, OCT, OCTET_LENGTH, ORD, POSITION, QUOTE, REPEAT, REPLACE, REVERSE, RIGHT, RPAD, RTRIM, SOUNDEX, SPACE, STRCMP, SUBSTR, SUBSTRING, SUBSTRING_INDEX, TO_BASE64, TRIM, UCASE, UNHEX, UPPER, WEIGHT_STRING)... Ceci permet une souplesse énorme dans les requêtes et devient vite très puissant.



La base de données récupère en outre des données Edel. Edel est un web service (soap) qui centralise des données par élevage qui viennent de divers organismes d'état. Edel est géré par l'Institut de l'élevage.

La base de données ne contient aucune donnée calculée afin de ne pas la surcharger inutilement en requête.

Vu le nombre de tables dans la base (98) et malgré une bonne structure, les jointures peuvent très vite s'avérer "compliquées" en SQL. L'ORM est là pour nous simplifier la vie.

Exemple de requête avec l'ORM :

```
getObject('Cow')->findList(['id', 'name', 'birth_date' => 'birth',
'birth.birth_date' => 'velage', 'cow_exit.exit_type' => 'exit_type',
'cow_lactation.milk_total' => 'milk', 'cow_exit.exit_date' => 'exit_date',
'cow_exit.exit_status' => 'exit_status', 'farming.etablisment' =>
'etablisment'], [['farming_id', '=', $this->id]], 'birth_date DESC')
```

La même requête en SQL :

```
SELECT DISTINCT (`c`.`id`), `c`.`name`, `c`.`birth_date` AS `birth`,
`b`.`birth_date` AS `velage`, `co`.`exit_type` AS `exit_type`,
`co`.`exit_date` AS `exit_date`, `co`.`exit_status` AS `exit_status`,
`f`.`etablisment` AS `etablisment`, `c`.`farming_id`
FROM `mon4v_repro_cow` AS `c`
LEFT JOIN (SELECT `birth_date`, `cow_id` FROM
`mon4v_repro_birth` AS `tmp1_b`
JOIN (SELECT cow_id AS join_key, `id` as filter, MAX(`birth_date`)
as indexMax FROM `mon4v_repro_birth`
GROUP BY cow_id) AS `tmp2_b` ON `tmp1_b`.`cow_id` =
`tmp2_b`.`join_key` AND `tmp1_b`.`birth_date` =
`tmp2_b`.`indexMax`) AS `b` ON `b`.`cow_id` = `c`.`id`
LEFT JOIN (SELECT `exit_type`, `exit_date`, `exit_status`, `cow_id`
FROM `mon4v_repro_cow_exit` AS `tmp1_co`
JOIN (SELECT cow_id AS join_key, `id` as filter, MAX(`id`)
as indexMax FROM `mon4v_repro_cow_exit`
GROUP BY cow_id) AS `tmp2_co` ON `tmp1_co`.`cow_id` =
`tmp2_co`.`join_key` AND `tmp1_co`.`id` = `tmp2_co`.`indexMax`) AS
`co` ON `co`.`cow_id` = `c`.`id`
LEFT JOIN `mon4v_repro_farming` AS `f` ON `c`.`farming_id` =
`f`.`id`
WHERE `c`.`farming_id` = 57
ORDER BY `c`.birth_date DESC
```

Cet exemple très simple puisque ne comportant qu'une clause "WHERE" est édifiant. Il montre l'intérêt de l'usage de cet ORM. Toutefois, le framework n'interdit pas les requêtes MySQL classiques. En cas de requête particulièrement spécifique ou demandant des clauses SQL non gérées par l'ORM, cela est bien pratique et rassurant pour qui aime les requêtes SQL (j'en suis). Je n'ai pour ma part, et malgré mon intérêt pour SQL pas eu l'utilité de cette possibilité.

Il n'y a pas de couche DAO à proprement parler, la logique du framework étant de placer la base de données au centre du logiciel. Cette couche est donc directement intégrée dans la couche "métier".

Cette façon de structurer un projet peut dérouter un développeur JEE ou un développeur C#, mais cela procure une réelle simplicité tout en gardant un code source très structuré et donc facilement maintenable.

Je n'utilise pour ma part que 6 tables de la base de données. Cela peut sembler très peu, mais l'exigence en terme de performance veut que les requêtes ne demandent que ce dont elles ont besoin, et pas une donnée de plus. De même les requêtes dans des boucles sont bannies.

Sur ces 6 tables, je fais trois requêtes. Aucun "" dans mes requêtes (pour des raisons d'optimisation).*

```
//cow per farming request
```

```
$cows = getObject('Cow')->findList(['id', 'name', 'birth_date' => 'birth',
'birth.birth_date' => 'velage', 'cow_exit.exit_type' => 'exit_type',
'cow_lactation.milk_total' => 'milk', 'cow_exit.exit_date' => 'exit_date',
'cow_exit.exit_status' => 'exit_status', 'farming.etablisment' =>
'etablisment'], [[['farming_id', '=', $this->id]], 'birth_date DESC');
```

```
//total milk request
```

```
$cowlactations = getObject('CowLactation')-
>findRaw(['start_lactation_date', 'end_lactation_date', 'milk_total', 'cow_id',
'cow.farming_id', 'tp_total', 'tb_total'], [['cow.farming_id', '=', $this->id], 'AND',
['start_lactation_date', '>', $dt_start], 'AND', ['end_lactation_date', '<',
$dt_end]]);
```

```
//anomalies request
```

```
$anomalys = getObject('Anomaly')->findList(['cow_id', 'anomaly_date',
'anomaly_type.id', 'anomaly_type.name' => 'name'], [['cow.farming_id', '=',
$this->id], 'AND', ['anomaly_date', '<', $dt_end], 'AND', ['anomaly_date', '>',
$dt_start]], 'anomaly_date DESC');
```

Peu de requêtes, donc, mais sur une base de données conséquente. Si beaucoup d'utilisateurs utilisent simultanément l'application, le serveur risque d'être assez vite surchargé.

C'est pourquoi il est primordial d'optimiser en amont la base de données. C'est certes plutôt de la compétence d'un administrateur système et réseau,

mais il est intéressant de savoir comment ça marche et de pouvoir intervenir en cas de surcharge apparente et ce d'autant plus que la base de données est hébergée sur le même serveur que l'application elle-même.

Si le serveur MySQL (ou MariaDB) tourne depuis quelques temps, un petit coup de MySQLTuner (sous Linux mais également sous Windows mais je

----- Performance Metrics -----

```
[--] Up for: 1h 57m 35s (5K q [0.735 qps], 1K conn, TX: 2M, RX: 744K)
[--] Reads / Writes: 99% / 1%
[--] Binary logging is disabled
[--] Total buffers: 192.0M global + 1.1M per thread (151 max threads)
[OK] Maximum reached memory usage: 195.2M (0.61% of installed RAM)
[OK] Maximum possible memory usage: 352.4M (1.10% of installed RAM)
[OK] Slow queries: 0% (0/5K)
[OK] Highest usage of available connections: 1% (3/151)
[OK] Aborted connections: 0.74% (10/1346)
[!!] Query cache is disabled
[OK] Sorts requiring temporary tables: 4% (3 temp sorts / 70 sorts)
[OK] Temporary tables created on disk: 8% (222 on disk / 2K total)
[OK] Thread cache hit rate: 99% (3 created / 1K connections)
[!!] Table cache hit rate: 4% (415 open / 8K opened)
[OK] Open file limit used: 3% (40/1K)
[OK] Table locks acquired immediately: 100% (253 immediate / 253 locks)
```

----- MyISAM Metrics -----

```
[!!] Key buffer used: 18.3% (3M used / 16M cache)
[OK] Key buffer size / total MyISAM indexes: 16.0M/215.0K
[OK] Read Key buffer hit rate: 95.7% (279 cached / 12 reads)
[!!] Write Key buffer hit rate: 0.0% (1 cached / 1 writes)
```

----- InnoDB Metrics -----

```
[--] InnoDB is enabled.
[OK] InnoDB buffer pool / data size: 128.0M/32.6M
[OK] InnoDB buffer pool instances: 1
[!!] InnoDB Used buffer: 20.91% (1713 used/ 8191 total)
[OK] InnoDB Read buffer efficiency: 99.27% (220259 hits/ 221881 total)
[!!] InnoDB Write buffer efficiency: 0.00% (0 hits/ 1 total)
[OK] InnoDB log waits: 0.00% (0 waits / 36 writes)
```

----- AriaDB Metrics -----

```
[--] AriaDB is disabled.
```

----- Replication Metrics -----

```
[--] No replication slave(s) for this server.
[--] This is a standalone server..
```

----- Recommendations -----

General recommendations:

```
Run OPTIMIZE TABLE to defragment tables for better performance
Restrict Host for user@% to user@SpecificDNSorlp
MySQL started within last 24 hours - recommendations may be inaccurate
Increase table_open_cache gradually to avoid file descriptor limits
Read this before increasing table_open_cache over 64: http://bit.ly/1mi7c4C
Beware that open_files_limit (1024) variable
should be greater than table_open_cache ( 431)
```

Variables to adjust:

```
query_cache_type (=1)
table_open_cache (> 431)
```

n'ai pas testé) peut fournir un bon aperçu des réglages de base, de son usage et des axes d'amélioration potentiels. Cet exemple sur ma base de données MySQL locale montre très clairement quelques améliorations simples à effectuer pour gagner en performance et en stabilité.

Même si cela ne dispense pas d'optimiser les requêtes, on peut au préalable optimiser notre base en jouant sur divers paramètres tels que query_cache, query_cache_size, tmp_table_size, innodb_buffer_pool_size, innodb_thread_concurrency (dans mon exemple, on constate par exemple que le query_cache est désactivé).

Pour faire des tests sur les requêtes ou tout simplement chercher dans la base de données, j'utilise le logiciel DataGrip (JetBrains) qui me convient mieux que PHPMyAdmin. Je l'utilise ici en local, mais je l'utilise sur d'autres projets distants (tant avec MySQL que MariaDB ou SQL Server) depuis quelques temps.



Développement métier

Part la plus intéressante du développement, selon moi, la couche métier est aussi sur ce projet celle qui prend le plus de temps.

La problématique ici est de développer une application "one page". On utilise donc jQuery et Ajax pour un développement asynchrone. Une fois mes requêtes faites, je dois faire pas mal de calculs liés à des filtrages par dates.

Les calculs de dates sont toujours problématiques en développement informatique, les divers langages n'utilisant pas les formats de date SQL "standardisés". PHP ne fait pas exception à cette règle.

Mon expérience en PHP procédural m'a en outre poussé à utiliser certaines techniques que mes collègues ne connaissaient pas, notamment l'utilisation à plusieurs reprises des variables dynamiques. Cette technique permettant de créer des variable "à la volée" s'avère très pratique et très puissante à l'usage quand on utilise un grand nombre de variables. Quand j'ai expliqué le fonctionnement des variables dynamiques à Simon Perigault (mon tuteur de stage) et à Florian Garcia (second développeur chez Vetosoft), ils ont d'abord été sceptiques, puis intéressés. Simon m'a demandé pourquoi utiliser cette méthode : Je lui ai répondu que ça m'était plutôt familier et que très honnêtement, je ne voyais pas d'autre solution plus simple pour gérer mon problème.



Voici un exemple du fonctionnement :

```
// Ici je fais ma requête dans l'ORM.  
$cowlactations = getObject('CowLactation')-  
>findRaw(['start_lactation_date', 'end_lactation_date', 'milk_total', 'cow_id',  
'cow.farming_id', 'tp_total', 'tb_total'],[['cow.farming_id', '=', $this->id], 'AND',  
['start_lactation_date','>', $dt_start], 'AND', ['end_lactation_date','<',  
$dt_end]]);
```

```
//recup total_milk per cow...  
if(!empty($cowlactations)) {
```

// Ici je boucle sur ma réponse pour initialiser mes variables dynamiques. Mes variables porteront comme nom l'ID de la vache,

```

"nb_velage" suivi de l'ID et je crée aussi deux tableaux "tab_tp" et "tab_tb".
foreach ($cowlactations as $cowlactation) { //initialize variables
    $id = $cowlactation->cow_id;
    $$id = 0;
    ${'nb_velage'.'$id'} = 0;
    $tab_tp[intval($id)] = 0;
    $tab_tb[intval($id)] = 0;
}

// Ici je boucle une seconde fois pour traiter mes variables.
foreach ($cowlactations as $cowlactation) {
    $id = $cowlactation->cow_id;
    $$id = $$id + $cowlactation->milk_total; //add total_milk
    ${'nb_velage'.'$id'}++; //increment nb_velage
    $tab_milk[intval($id)] = $$id; //set this key/value in table
    $tab_velage[intval($id)] = ${'nb_velage'.'$id'}; //set this key/value
    in table
        $tab_tp[intval($id)] = $tab_tp[intval($id)] + $cowlactation-
        >tp_total;
        $tab_tb[intval($id)] = $tab_tb[intval($id)] + $cowlactation-
        >tb_total;
    }
}

```

Toutes ces données seront ensuite très faciles à récupérer simplement et donc à traiter grâce à l'ID de chaque vache.

Il faut aussi savoir qu'il n'y a pas de limite à l'utilisation de ce type de variables, nous pouvons donc faire d'une variable dynamique une autre variable dynamique et ceci à l'infini en cas de besoin (à utiliser avec parcimonie tout de même, ça peut vite devenir complexe à lire).

J'affectionne beaucoup cette méthode peu connue et elle m'a souvent permis d'éviter de taper énormément de lignes de code et aussi de limiter l'usage des tableaux à n dimensions.

On peut d'une façon similaire utiliser des fonctions dynamiques. C'est beaucoup plus fréquent que l'utilisation des variables dynamiques, mais l'utilisation est assez proche

Un exemple :

```
// Recuperation de l'action demandée par l'url  
$action = $_GET['action'];  
  
function ajoute($x, $y) {  
    return $x+$y;  
}  
  
function soustrait($x, $y) {  
    return $x-$y;  
}  
  
// execution de la fonction  
if($action == 'soustrait' or $action == 'ajoute'){  
    echo $action('4', '2');  
}
```

Dans cet exemple succin l'intérêt de l'utilisation des fonctions dynamiques n'est pas évident, mais en imaginant qu'on ait des dizaines d'actions possibles, passer par des ifs serait très long et donnerait un code très verbeux.

La modification de mes requêtes pour y inclure le tri par date (requête Ajax) m'a fait perdre un peu de temps. En effet, j'ai longtemps cherché pourquoi mes résultats étaient incohérents. En fait, mes résultats étaient bons, c'était juste mon jeu de données de test qui n'était pas assez complet.

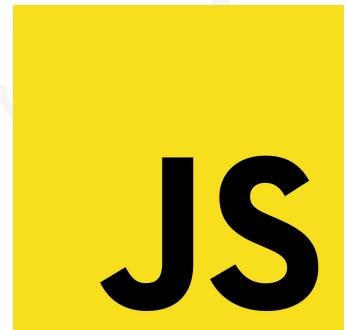
En outre Ajax (JavaScript Et XML Asynchrones) m'a posé quelques soucis. Je n'en ai pas fait depuis longtemps, et nous n'en avons pas fait en cours (nous n'avons pour ainsi dire pas fait de JavaScript, ce qui est fort dommage car c'est un langage qui a le vent en poupe malgré son âge). La documentation en ligne (OpenClassRooms, StackOverflow, W3School...) et l'aide de mon tuteur de stage m'ont permis de m'en sortir, mais j'ai en marge de ce stage commencé la formation JavaScript d'OpenClassRooms afin de me remettre à niveau dans ce langage indispensable. Je pense continuer ensuite avec la formation NodeJS (plutôt à la mode depuis quelque temps).



Une fois le code fonctionnel, j'ai du faire un gros nettoyage de celui-ci pour tout passer en Ajax. Auparavant, ma page (site "one page") se chargeait en premier lieu statiquement et les rechargements se faisaient en Ajax. Maintenant, il n'y a plus cette première requête, tout est fait en Ajax. Cela me permet d'avoir une seule (trois, en fait) et même requête pour tous les affichages (contre deux variantes, donc six requêtes, précédemment).

Cette optimisation du code est passée aussi par une factorisation du code PHP pour une vitesse de calcul améliorée (nettement) et une maintenabilité simplifiée.

Souvenons nous que l'optimisation est une priorité pour cette application qui doit rester la plus "légère" possible (c'est l'une des exigences principales demandées par Simon Perigault). On comprend aisément l'intérêt de ces optimisations en lisant quelques chiffres : la France compte 3,5 millions de vaches laitières, principalement Prim'Holstein réparties dans 92 000 exploitations soit en moyenne 40 à 50 vaches par exploitation laitière (avec un maximum de 150). La France est ainsi le deuxième pays producteur de lait en Europe après l'Allemagne. Donc à terme une base de données potentiellement très conséquente et un nombre de calculs effectués non négligeables.



Pourquoi utiliser JavaScript ?

Une petite blague de développeur web : "JavaScript : a language so flexible it even runs on Internet Explorer" ("un langage si flexible qu'il fonctionne même sur Internet Explorer").

Plus sérieusement, JavaScript a eu plusieurs vies. Il a eu mauvaise presse. On l'a considéré comme un "petit truc parfois bien pratique", longtemps pour "faire des effets dans sa page web". Or, JavaScript est avant tout un langage. Un langage au même titre que le C, Ruby, PHP, Java, C# et bien d'autres.

Aujourd'hui, JavaScript est de retour et il prend sa revanche.

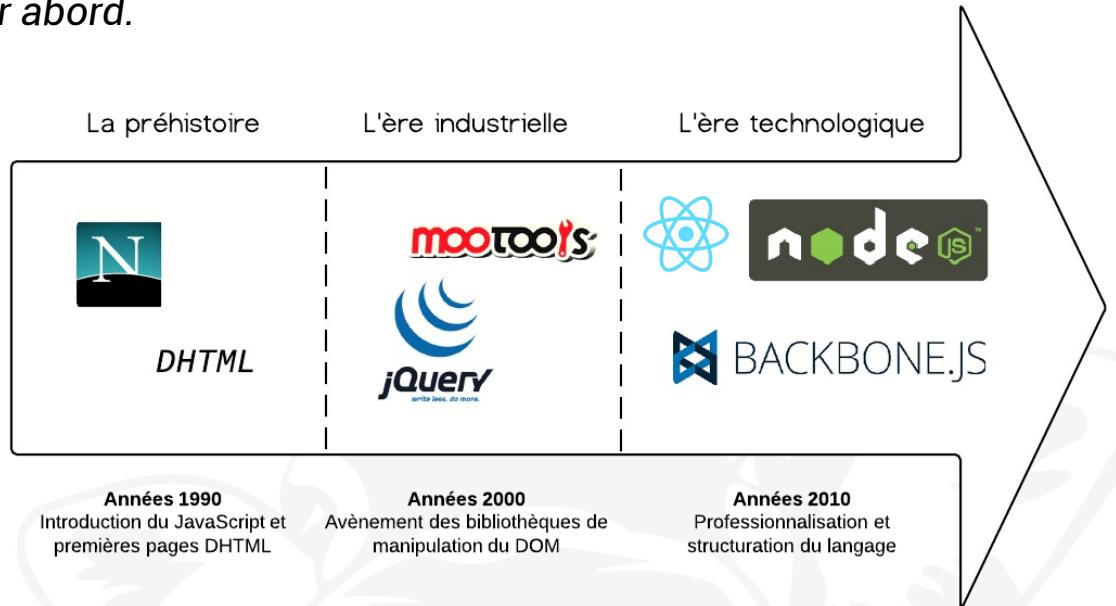
En quelque sorte, les développeurs (notamment les développeurs PHP) sont en train de découvrir que ce langage qu'ils ont longtemps ignoré, parfois même méprisé, cache en fait bien son jeu.

Non, JavaScript n'est pas juste un petit langage utilitaire.

Oui, JavaScript est un langage à part, qui s'utilise vraiment différemment de Java, du C et d'un tas d'autres langages.

Oui, JavaScript peut être compliqué à utiliser, mais recèle derrière ça une réelle puissance.

Google a commencé à rendre le langage beaucoup plus rapide avec l'apparition du navigateur Google Chrome. Avec ce navigateur est né le moteur d'exécution V8 qui a considérablement permis d'accélérer l'exécution de code JavaScript. Des outils comme Node.js sont ensuite apparus. Les bibliothèques dont le nom finit par .js se sont multipliées : Backbone.js, Ember.js, Meteor.js, Moment.js... JavaScript a l'air à nouveau cool... et semble en même temps plus compliqué qu'il n'y paraissait au premier abord.



Ce schéma bien qu'assez caricatural reflète plutôt bien la réalité. Toutefois jQuery n'est pas à jeter avec l'eau du bain, il (ainsi que jQueryUI et jQuery mobile) reste très utilisé et très efficace. Milkup utilise jQuery et jQueryUI. Les futures évolutions de l'application utiliseront aussi la librairie React (React est une bibliothèque qui ne gère que l'interface de l'application, considéré comme la vue dans le modèle MVC. Elle peut ainsi être utilisée avec une autre bibliothèque ou un framework MVC comme AngularJS. La bibliothèque se démarque de ses concurrents par sa flexibilité et ses performances, en travaillant avec un DOM virtuel et en ne mettant à jour le rendu dans le navigateur qu'en cas de nécessité).

POO (programmation orientée objet)

Milkup est comme tous les logiciels moderne développé en POO (programmation orientée objet). Chaque table de la base de données est associée à un objet. Pour ma part, je n'ai pas eu à créer d'objet spécifique à mon projet, mais j'utilise six objets existants. Encore une fois, cela interdit (tant que possible) de modifier les objets existants, il faut donc s'adapter et coder avec ce que l'on a (mais comme la base de données ne contient aucun élément calculé, cela ne pose pas vraiment de problème).

L'internationalisation (i18n)

Vetosoft ne distribue pour l'instant son logiciel Milkup qu'en France, mais dans un souci d'évolutivité, il a été prévu dès le départ un système d'internationalisation (i18n dans le jargon des développeurs). Aucun mot n'est écrit « en dur » dans le code. Pour écrire « Nom », par exemple, nous appelons en PHP `Vtext('COW_NAME')`. Si la constante `COW_NAME` n'existe pas dans la base de données, elle sera automatiquement créée sans traduction. Si une constante sans traduction est appelée dans le code, elle retournera le nom de la constante (ici `COW_NAME`). De cette façon, il est très simple d'ajouter une nouvelle langue au logiciel, seule la table contenant tous les textes est à traduire. Il n'y a rien à modifier dans le code. Ce système est notamment très utilisé dans les CMS (Wordpress et Drupal utilisent ce système avec des fichiers .po modifiables avec le logiciel poedit, par exemple).

Les traductions sont modifiables (on peut également en ajouter ici) dans l'interface de Milkup dans un tableau comme celui-ci :

Traductions				
			+ Ajouter une traduction	Milkup
1700	PLANED_TYPE	Planifié	fr-FR	04/07/2018 16:06:59
1699	PLANED_EXIT	Sortie planifiée	fr-FR	04/07/2018 15:59:34
1698	PLANED	Planifiés	fr-FR	04/07/2018 15:35:55
1697	COWS_NOT_OUT	Bovins non sortis	fr-FR	04/07/2018 14:20:54
1696	COWS_EXIST	Non sortis	fr-FR	04/07/2018 13:33:07
1695	PLANNED_COWS_OUT	Bovins sortis planifiés	fr-FR	04/07/2018 12:37:32
1694	COWS_EXIT	Sortis	fr-FR	04/07/2018 12:13:04
1693	PLANNED	Planifiés	fr-FR	04/07/2018 12:12:22

Ce système est plus une bonne habitude à prendre (bonne pratique pour tout développement logiciel) qu'une réelle contrainte. Si les constantes sont nommées correctement, cela ne pose pas de problème particulier quant à la lisibilité du code.

L'interface homme/machine (IHM)

Exemple de code PHP/HTML.

Envoie d'une valeur dans un tableau dans la vue :

On envoie du HTML (avec du Bootstrap et du Font Awesome) par PHP :

```
$global[$i]['aff_anomaly'] .= '<i class="icon-info fa fa-info-circle" role="tooltip" title="'.${'anomaly_aff'}.{$id}.'"></i>';//infobulle anomalies
```

On récupère ce HTML dans la vue comme ceci :

La variable (ici "aff_anomaly") sera affichée dans mon tableau dans le <TD> correspondant (l'attribut "column" porte le même nom que ma variable envoyée).

```
<td column="aff_anomaly" class="text-center text-md-left"><?php echo VText('ANOMALIES'); ?></td>
```

Pour les valeurs à afficher en dehors d'un tableau :

On envoie nos variables comme ceci :

```
$datas['iTotalRecords'] = count($datas['aaData']);
$datas['iTotalDisplayRecords'] = count($datas['aaData']);
foreach($datas['aaData'] as &$data){
    $data['DT_RowAttr'] = ['item-id'=>$data['id']];
}

return [
    'global' => $global,
    'listData' => $datas,
    'total' => $total,
]
```

Dans la vue, on récupère la variable en JavaScript comme ceci :

```
<?php echo VText('TOTAL'); ?> : <%=customData.total %>
```

Le framework rend encore une fois le code assez déroutant au départ mais très simple et efficace à l'usage. La vue est en HTML/CSS et les données dynamiques sont exportées en PHP et importées dans le HTML en JavaScript (avec jQuery notamment) tout comme les filtres (sélections de dates et de divers paramètres) qui sont eux aussi gérés en JavaScript.

L'utilisation des librairies Bootstrap et Font Awesome simplifie beaucoup le design, mais en contre partie, comme ces deux bibliothèques sont aujourd'hui très utilisées sur le web (tant pour des applications en ligne que pour des sites web), le logiciel ressemblera à beaucoup d'autres utilisants les mêmes technologies. Ceci peut être un énorme avantage pour ce qui concerne l'ergonomie (pour l'utilisateur) et la création (pour le développeur) de l'application (au détriment de l'originalité du design).

Tous	Sortis	Non sortis	Planifiés						
Bovins sortis Exploitation 22 Propre									
Sorties : 1576 Non précisé : 0 (0%) Réformé : 331 (21%) Vente : 1013 (64.28%) Equarrissage : 232 (14.72%) Sortie de la reproduction : 0 (0%) Prêt ou pension : 0 (0%)									
↓Filtres ↑									
Dates									
Date de début	Date de fin								
01/08/2008	03/08/2018								
Appliquer le(s) filtre(s)									
100 ▾ Lignes	Recherche								
Nom	Lait total	Nb velages	Moy lactation	Jours de vie	PL/V/JdV	Taux protéique	Taux butyreux	Anomalies	Raison
STEL DREAM i	79774	5	15955	3722	21	21	24	3 i	Equarrissage
STELVEREI i	62907	6	10485	4194	15	31	38	2 i	Vente
STEL DESKA i	63412	7	9059	1138	56	31	38		Vente
STELCASTE i	79558	7	11365	4047	20	29	41		Réformé
STEL DIOR i	73593	6	12266	3376	22	26	34		Réformé

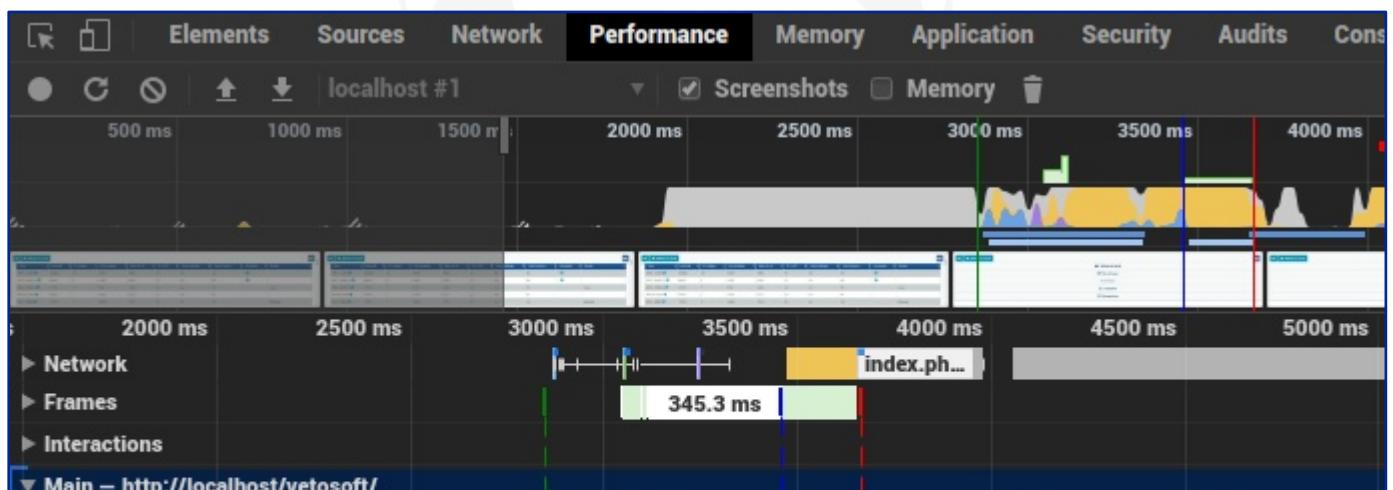
On reconnaît très bien ici Bootstrap. C'est fonctionnel, efficace, et pour une application comme celle ci qui ne nécessite pas de design particulier, hormis certains graphiques de statistiques (je n'ai pour ma part pas eu à utiliser la librairie de graphiques), c'est parfait.

Tests

Les tests effectués ont été de deux types distincts : les tests unitaires classiques pour vérifier la pertinence des résultats obtenus et les tests concernant l'optimisation du code, afin de gagner en temps de calcul pour l'utilisateur final, mais aussi pour le serveur (très important puisque l'évolution prévue du logiciel sera un passage à REACT, une bibliothèque JavaScript libre développée par Facebook depuis 2013. Le but principal de cette bibliothèque est de faciliter la création d'application web monopage, via la création de composants dépendant d'un état et générant une page HTML à chaque changement d'état., donc du JavaScript côté serveur et non plus côté client. Pour information, Netflix utilise REACT côté serveur uniquement depuis le 25 octobre 2017, ce qui lui a permis de gagner 50% de performance).

Les tests unitaires sont à part dans la grande famille des tests car ils se placent en amont, au tout début du processus, lors des développements. C'est ce qui a été réalisé (plutôt en fin de développement, en fait, mais généralement à chaque évolution du module). Pour ces tests, aucun outil particulier (comme le framework PHPUnit, par exemple) n'est utilisé chez Vetosoft (la taille de la structure et le fait que le développement est réalisé en interne ne le justifient pas).

Ces tests consistent essentiellement à une vérification manuelle des résultats obtenus et à une recherche de bugs (très peu de bugs de fonctionnement, du fait de la structure même du framework).



Pour les tests de performance, J'ai utilisé l'un des meilleurs amis du développeur d'applications web : la console JavaScript de Google Chrome. Afin de donner les bons outils aux développeurs web, les navigateurs (et

tout particulièrement Google Chrome) se sont peu à peu équipés de consoles de développement permettant d'entrer des instructions à la volée, avec bien souvent de l'auto-complétion, de consulter les données en mémoire ou d'explorer les fonctions et variables disponibles.

Des commandes plus avancées visent à définir des points d'arrêt et d'inspecter la pile des appels. La console est un outil indispensable lorsque l'on souhaite écrire quelques lignes de JavaScript, ou bien concevoir des scripts plus évolués notamment avec des frameworks tels que jQuery.

L'onglet NETWORK nous permet de voir tout ce qui se passe dans le niveau réseau : appel des requêtes vers le serveur selon les chargements de la page.

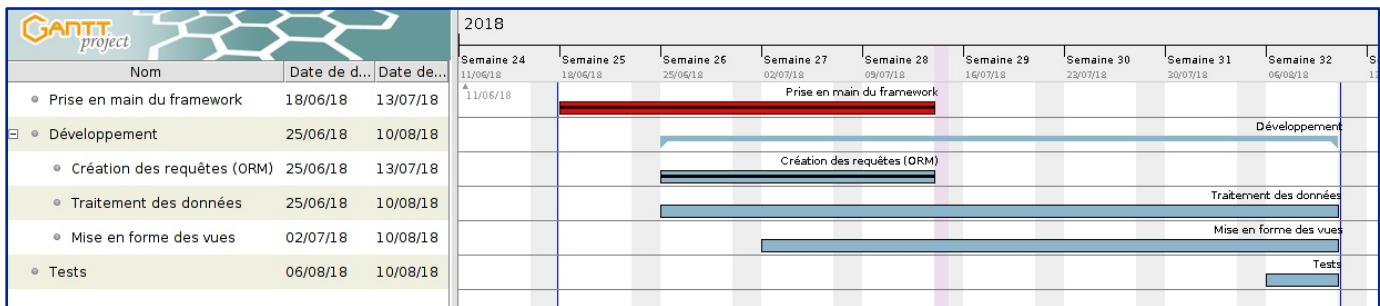
L'onglet PERFORMANCE (que j'ai beaucoup utilisé en fin de développement pour l'optimisation du code) permet de comprendre comment se déroule le chargement de la page et de voir les problèmes tels que les boucles redondantes, ou les requêtes dans des boucles (à éviter tant que possible, bien sur).

L'onglet SOURCE est quant à lui utile pour régler les problèmes graphiques.

En tout état de cause ces tests ont permis de mettre le doigt sur quelques problèmes de code qui ne gênaient pas le bon fonctionnement de l'application mais ralentissaient considérablement son exécution. J'ai pu ainsi refactoriser mon code et diminuer par deux le temps d'exécution de mes requêtes.

Retro planning

Il n'est jamais simple d'établir un planning, surtout quand on utilise des outils que l'on ne connaît pas et qu'il faut apprendre à utiliser ces outils tout en les utilisant. Je veux dire que huit semaines pour développer un projet concret, ça semble assez peu, mais quand il faut en parallèle apprendre l'outil (ici le framework, son ORM et son système de templating), ça devient



un vrai casse tête. En effet, on fait son planning prévisionnel en se basant sur notre expérience, et donc sur des outils que l'on maîtrise en se laissant une petite marge d'apprentissage au début. Par contre, même si j'ai du travailler en collaboration avec deux développeurs et un autre stagiaire, j'étais seul sur mon module, et donc je n'avais pas à attendre la fin d'un développement d'un de mes collègues pour pouvoir avancer dans mon propre développement. Cette pseudo autonomie m'a permis de mener à terme mon projet malgré une évaluation de départ erronée.

Le framework utilisé implique par exemple que l'IHM est réalisée en même temps que la partie métier. De ce fait, le développement de l'IHM est presque transparent et permet de passer plus de temps sur le métier, et donc au final de respecter les délais.

Les tests ont été réalisés au cours du développement à chaque nouvelle fonctionnalité. Ce n'est pas une pratique courante (quoique relativement fréquente en PHP pour les tests unitaires), mais la structure étant petite et ne disposant pas de testeur, chaque développeur teste son propre travail avec sa base de données plus ou moins complète. Les tests effectués ont été techniquement concluants mais pas crédibles pour le vétérinaire de l'entreprise. Cela s'explique par le fait que pour ces tests, des jeux de données fictives ont été enregistrés dans la base de données, mais ces données n'étaient pas crédibles d'un point de vue vétérinaire. Mais il s'avère que si on avait fait un dump d'une base de données réelle (trop compliqué à gérer sur 5 serveurs de pré-prod et 4 machines de développement), les résultats auraient été concluants.

Conclusion

J'ai eu la chance durant ce stage de travailler sur un projet bien défini et d'avoir une demande réelle et précise. J'ai eu la satisfaction de mener à terme ce projet. Je suis fier de savoir que le module que j'ai réalisé sera intégré au logiciel lors de sa prochaine mise à jour majeure et que l'équipe ai pu être contente de mon travail.

Lors de ce stage, j'ai eu la chance de travailler sur une technologie que je connaissais plutôt bien à savoir PHP (et un peu moins bien pour JavaScript) mais en utilisant un framework propriétaire. J'ai ainsi pu évaluer mon adaptabilité dans un environnement de travail spécifique.

Je considère ce stage plus comme la suite de l'apprentissage que comme un travail, et ces huit semaines chez Vetosoft m'ont appris beaucoup notamment pour ce qui est du travail collaboratif (mon expérience passée en tant que développeur PHP était plutôt individuelle, voire en binôme, mais pas plus).

j'ai pu constater au fur et à mesure du stage une nette amélioration dans mon utilisation des outils à ma disposition et des librairies utilisées par Milkup (Bootstrap, jQuery, jQuery UI, DataTables, Moment.js, entre autre), ce qui m'a permis de travailler de plus en plus efficacement au cours de mon développement.

Ce stage et ce travail collaboratif a en outre bien insisté sur ce que l'on nous a sans cesse répété tout au long de la formation (presqu'une obsession chez nos formateurs, je comprends maintenant pourquoi) : l'importance des BONNES PRATIQUES...

J'ai mis en pratique la méthode Agile, et j'ai pu constater la difficulté à établir un planning et surtout à s'y tenir. Mais certaines tâches étant sous évaluées quand d'autres sont sur évaluées permettent de retomber sur ses pieds quand il s'agit de développement full stack.

Un grand merci encore à toute l'équipe de Vetosoft qui m'a permis de passer deux mois passionnants et épanouissants.

Je rappelle qu'il m'a été demandé pour des questions de droits d'auteur de supprimer de mon ordinateur toutes les sources ainsi que le dump de la base de données à la suite de cette présentation.

Le push git final.

```
franck@franck:/var/www/html/vetosoft$ git commit -am "fin du stage"
[preprod-franck 143668d0] fin du stage
 1 file changed, 3 insertions(+), 1 deletion(-)
franck@franck:/var/www/html/vetosoft$ git push
root@preprod.milkup.fr's password:
Énumération des objets: 9, fait.
Décompte des objets: 100% (9/9), fait.
Delta compression using up to 8 threads.
Compression des objets: 100% (5/5), fait.
Écriture des objets: 100% (5/5), 441 bytes | 441.00 KiB/s, fait.
Total 5 (delta 4), reused 0 (delta 0)
remote: **** pulling changes from preprod-franck ****
remote: Depuis /var/www/preprod
remote:   399c968..143668d preprod-franck -> origin/preprod-franck
remote:   6dc8606..d56460d master      -> origin/master
remote:   a6a88c8..20ecf94 preprod-flo -> origin/preprod-flo
remote:   fffffcdc..e7f8a7c preprod-fred -> origin/preprod-fred
remote: Mise à jour 399c968..143668d
remote: Fast-forward
remote:   views/page/PageScrapView.php | 4 +---
remote: 1 file changed, 3 insertions(+), 1 deletion(-)
To ssh://preprod.milkup.fr:22222/var/www/preprod.git
  399c968b..143668d0 preprod-franck -> preprod-franck
franck@franck:/var/www/html/vetosoft$ █
```

Franck Canonne
www.le7.net