COOLS Aurélie

GAUTHIER Inès

GENIN Émilie

LERAT Jean-Sébastien

ROBBERTS François

ROMBAUX Michaël

VANDEN DRIES Virginie

Master ingénieur civil en informatique et gestion, à finalité spécialisée en web et stratégies d'entreprises

UI-M2-IRIGWE-805-C : Applications web et multimédia
UI-M2-IRIGIG-804-C : Expertises digitale et logicielle

Enseignants : MAHMOUDI Saïd et MAHMOUDI Sidi

Année académique 2018-2019

Table des matières

Tables des illustrations

1. Introduction et présentation du projet

1.1. Introduction

L'objectif de ce présent travail est de permettre aux étudiants que nous sommes d'acquérir des compétences en développement web adaptés aux appareils mobiles. C'est aussi une bonne occasion pour nous de développer nos compétences organisationnelles et notre capacité à travailler efficacement en équipe grâce à l'application de méthodologies vues notamment au cours d'expertises digitale et logicielle.

Le fait de travailler sur un projet réel pour un client tout aussi réel ajoute une pression supplémentaire qui simule adéquatement les conditions dans lesquelles nous serons amenés à évoluer professionnellement.

Ce rapport débutera par la présentation générale du projet WALLeSMART.

Ensuite nous aborderons la gestion du projet comprenant la méthodologie de gestion, la justification des choix technologiques ainsi que l'organisation et la répartition des tâches.

Le chapitre suivant concernera le développement du projet à proprement dit. Nous discuterons de l'architecture, des nombreuses facettes du développement ainsi que de la préparation des tests unitaires et fonctionnels. La fin de ce chapitre présentera le résultat final de notre travail par l'illustration de quelques fonctionnalités de notre plateforme.

Ensuite nous présenterons le bilan de WALLeSMART grâce à l'application de la méthode COCOMO II et nous évaluerons la qualité du code grâce à SONARQUBE avant d'arriver à la conclusion du rapport.

1.2. Présentation du projet WALLeSMART

Le présent projet est un sous-projet (projet d'amorçage) d'un projet interinstitutionnel de 6 ans demandé par l'Association Wallonne de l'Elevage (AWE). Cette dernière est représentée par François RIGA et les responsables UMONS qui sont Saïd MAHMOUDI et Sidi MAHMOUDI. Les personnes en charge du développement sont Fabrice Bellarmin NOLACK FOTE (doctorant) et Amine ROUKH (post-doctorant).

Au terme des 6 années, l'AWE souhaiterait obtenir une plateforme qui regroupe les différentes applications (proposées par différentes entités) qu'ils utilisent. La plateforme devrait permettre de gérer automatiquement toutes les données des agriculteurs (en respectant la loi sur le RGPD[1]) afin d'automatiser certaines tâches administratives, de proposer un suivi des données personnelles via une présentation des données (cartographie, graphique, tableau, ...), de proposer une visualisation des données à un conseiller (comptable, ...) ou encore de fournir des données à des partenaires de recherche.

[1] https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/principles-gdpr_fr

Cadre de travail du projet

En tant que *proof of concept*, il est demandé de réaliser une infrastructure interconnectant différentes sources de données qui peuvent être hébergées chez des agriculteurs, aux entreprises qui fournissent des services à l'AWE ou à l'AWE elle-même. Le travail consiste principalement à proposer ladite infrastructure permettant d'interconnecter et de visualiser des données dont les accès sont contrôlés par un unique point central : un portail Web simple et ergonomique. Celui-ci permet d'authentifier les utilisateurs et de présenter les données auxquelles ils ont accès sous forme de tableaux ou de graphiques.

Contraintes et exclusions

Les données proviennent de sources hétérogènes. Il est impossible de démontrer, à cette étape de *proof of concept*, l'interopérabilité avec toutes sources de données. C'est pourquoi le projet va restreindre le champ des possibles en sélectionnant 2 types de fichiers, un type de Web service et 2 systèmes distincts de gestion de bases de données relationnelles.

La solution proposée doit permettre l'extension de données de type "cartographie" mais ne doit pas l'intégrer actuellement.

Les données exploitées ne sont pas représentatives de la réalité car il est impossible de les obtenir dans le temps imparti.

L'interaction avec d'autres applications de l'AWE n'est pas possible étant donné que nous ne disposerons pas des accès dans le temps imparti. Cela est simulé via une communication avec une technologie de Web service.

L'authentification « utilisateur » se fera exclusivement via le portail Web qui exploite des données dans la base de données de la plateforme Web car il est impossible d'obtenir l'accès à un serveur LDAP existant et cohérent pour le projet. Toutefois, la conception du module d'authentification doit prévoir l'extensibilité dans ce sens.

Objectifs

L'objectif principal est de permettre la reconnaissance de 6 rôles utilisateurs au sein du portail Web. Un utilisateur peut avoir plusieurs rôles :

- Gestionnaire : peut créer des comptes (ou inviter quelqu'un à créer un compte), peut fournir des droits à une entité (attention aux droits sur les données personnelles : seul l'agriculteur gère ses droits).
- Agriculteur : peut importer des données (excel, access, fichier text (format csv), ...) et autoriser des personnes à accéder à ses données. Il peut également supprimer toutes ses données personnelles ou bien les modifier. Attention, certaines données peuvent être envoyées à une autre application et ne plus être modifiable sur cette autre application.
- Conseiller : peut accéder à des données générales et des données propres à un ou plusieurs agriculteurs (si ceux-ci le permettent).
- Scientifique : peut accéder à des données thématiques générales (définies par le gestionnaire) et accéder à des données privées d'agriculteurs (si ceux-ci le permettent).
- Citoyen : peut visualiser les données générales (exemple : quelle est la quantité de méthane produite en région wallonne en rapport avec le nombre de vaches).
- Développeur : un développeur peut avoir un accès particulier qui lui permet de publier une nouvelle application, obtenir un échantillon de données de test, environnement de test, ...

Les données générales doivent également contenir des informations (statistiques) à propos de l'utilisation du portail Web (exemple : nombre moyen de visiteurs, nombre d'agriculteurs inscrits, …).

Les données peuvent être visualisées sous forme de tableaux ou de graphiques.

2. Gestion du projet

2.1. Méthodologie de gestion de projet

Afin de mener à bien le projet, nous avons opté pour l'utilisation de méthodologies agiles, en particulier scrum combiné avec XP. Ces méthodologies permettent de fournir une solution de qualité, et une écoute attentive des clients. En effet, le prototypage régulier permet de suivre les attentes du client.

De plus, nous appliquons le principe de *pair programming* issu de la méthodologie XP qui œuvre dans le but d'éviter les erreurs en amont et le *tests driven developpement* où le principe est de créer les tests avant le développement du programme afin de définir clairement les attentes de la solution logicielle.

Nous avons également utilisé Prince II, en particulier le Project Initiation Document qui était mis à jour à chaque point de contrôle. Nous y avions identifié les risques potentiels et envisagé des solutions. Lorsque nous avons été confrontés à des problèmes, en particulier des retards, nous avons appliqué les solutions que nous avions envisagées. Dans ce même document, nous avons planifié le déroulement du projet ainsi que les étapes clefs et les dépendances. Cela nous a permis d'identifier clairement comment paralléliser au mieux les tâches afin de rattraper le retard.

Pour la communication, nous avons principalement utilisé les SMS, Facebook et les emails. En effet, il était difficile de travailler tous en même temps simultanément et d'être joignable par voie orale. C'est pourquoi nous avons utilisé ces outils moins conventionnels dans un projet logiciel mais plus adapté à des étudiants à horaires décalés.

Pour la gestion du code, nous avons utilisé Github. Toutefois, nous utilisons uniquement la branche principale étant donné que l'infrastructure de développement ne nous permettait pas de tester en local la solution développée. En effet, le serveur Web récupérait directement la branche principale du dépôt toutes les 5 minutes et l'utilisation d'une autre branche en locale n'aurait pas permis de tester le code étant donné les ressources matérielles nécessaires.

2.2. Outils et choix technologiques

Dans cette partie, nous traiterons des différents choix technologiques auxquels l'équipe a été confrontée. Nous commencerons par aborder le choix de l'infrastructure dédiée à la Business Intelligence (BI). Le choix d'un système de gestion de base de données sera traité au point suivant. S'en suivra les choix technologiques liés au développement web et au gestionnaire de code. Nous aborderons ensuite le choix des systèmes pour le déploiement des tests unitaires et fonctionnels. Nous terminerons par motiver le choix d'un IDE commun.

a. Infrastructure Business Intelligence

Les technologies qui ont été initialement envisagées sont les suivantes : outils de BI, cloud, cluster de calcul, VM/machines physiques, implémentation personnalisée, utilisation de logiciels libres.

Les outils de BI ont été exclus pour deux raisons majeures : premièrement, ils sont trop coûteux et deuxièmement, ils ne sont pas modulaires (par exemple : on ne peut pas imposer à Microsoft d'ajouter une fonctionnalité à PowerBI).

En ce qui concerne le cloud, celui-ci n'est pas envisageable pour des raisons de sécurité des données, RGPD et de coût.

Ensuite, le cluster de calcul (slurm) avec docker (container) est trop complexe à mettre en place et est très chronophage à prendre en main pour le client.

Nous avons finalement opté pour l'utilisation et la configuration de machines (virtuelles) pour les avantages suivants : elles sont dimensionnables et des backups sont possibles via simple snapshot.

b. Logiciel de Business Intelligence

Nous avons comparé les logiciels libres d'intégration et d'exploitation de données que sont Zeppelin et Jupyter. Nous avons préféré Zeppelin à Jupyter pour plusieurs raisons :

- il est soutenu par la fondation Apache ;
- il est plus extensible ;
- contrairement à Jupyter, il gère des données cartographiques ;
- il est plus récent et la communauté de Jupyter est en déclin.

c. Système de gestion de la base de données centrale

Parmi les systèmes de gestion de base de données (SGBD) existants, nous n'avons retenus que 3 candidats : MariaDB, MySQL et PostgreSQL. Tous les autres systèmes ont été écartés principalement pour des questions de licence (Oracle et SQL Server) ou par le simple fait qu'ils sont non-relationnels (Cassandra et MongoDB).

Des 3 candidats, pour le développement d'un portail web, il aurait été plus naturel de choisir MySQL car il est le SGBD le plus proposé par les solutions d'hébergement mais nous avons préféré utiliser MariaDB car il présente les avantages suivants :

- C'est un fork de MySQL et il peut se substituer à ce dernier ce qui est un avantage pour les applications qui seraient initialement développées pour du MySQL.
- La communauté est plus active et il ne dépend pas de l'entreprise Oracle
- De nombreux plugins jouissent d'un support officiel ce qui permet d'étendre son champ d'action
- Le moteur de stockage CONNECT pourrait se révéler être une fonctionnalité fort utile pour les prochaines évolutions de la plateforme. Ce moteur permet d'interconnecter MariaDB à

des sources de données externes (JDBC, .xlsx, .csv, etc.) et de les interroger au travers de son API avec de simples requêtes SQL.

Il nous a également été demandé d'utiliser un second SGBD dans le cadre de ce projet. PostgreSQL fut préféré à MySQL, car il offre plus de fonctionnalités tout en étant plus véloce. Par exemple : avec MySQL les vues n'exploitent pas toujours correctement les index des tables contrairement à PostgreSQL qui est capable d'optimiser ses plans de requêtes dynamiquement, en fonction des filtres appliqués aux vues.

d. FrameWork

En ce qui concerne le choix du Framework php, il en existe beaucoup. Parmi les plus utilisés à l'heure actuelle se trouvent Symphony 2 (S2), Zend Framework 2 (ZF2), Cake PHP, Lavarel et CodeIgniter (CI). L'image[2] ci-dessous donne une indication de l'évolution de l'intérêt pour ces Framework sur les cinq dernières années.

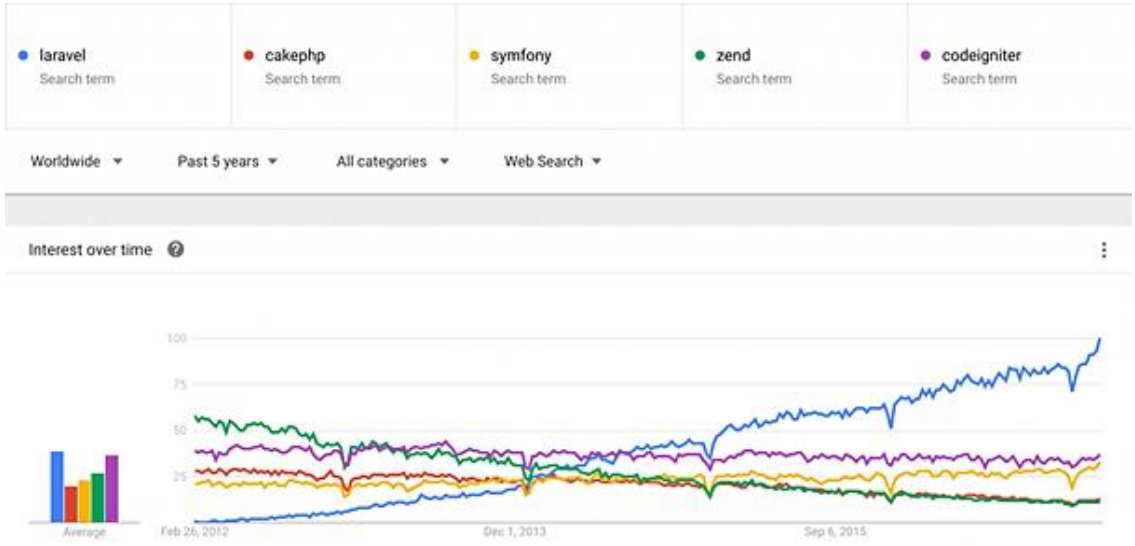

FIGURE 1: POPULARITÉ DE 5 FRAMEWORKS CES 5 DERNIÈRES ANNÉES

Etant tous étudiants en cours du soir et occupés par nos professions respectives en journée, il était vital de privilégier un framework rapide à prendre en main. Nous nous sommes donc, en toute logique, tournés vers Codeigniter qui est léger, simple, rapide à apprendre et modulable. En outre, le développement est actuellement géré par l'université British Columbia Institute of Technology (sous licence MIT – source wikipedia) et possède une communauté de développeurs importante et active[3], ce qui assure une certaine pérennité.

[2] https://www.oxiwiz.fr/meilleur-framework-php-2018-pour-developper-un-site-internet/
[3] https://github.com/codeigniter4/CodeIgniter4 , https://github.com/bcit-ci/CodeIgniter/tree/master

e. Design du site

Pour le design du site l'utilisation de Bootstrap a été confrontée à l'implémentation manuelle du CSS.

En effet, Bootstrap est une collection d'outils qui utilise une approche tout en un. Cela implique que l'on considère que dans un fichier HTML, les balises doivent représenter un visuel et être complétées de manière à laisser transparaître rapidement le résultat visuel. Pour obtenir le visuel attendu, cela implique de produire une grande quantité de balises imbriquées et la multiplication du nombre de classes dans l'attribut class.

Par contre, le W3C améliore les normes de structures HTML dans un but de séparation de la structure et du rendu. Dans ce but, le HTML5 introduit de nouvelles balises telles que <header>, <footer>, <section>, <article>, <aside>, <figure>, etc. Donc, la structure de Bootstrap va à l'encontre de la philosophie du W3C avec HTML5 et CSS3.

Enfin, Bootstrap est assez lourd (il génère plus de code) et peu adéquat pour des systèmes amenés à évoluer. Il n'est donc pas optimal du point de vue de la maintenance du code et des changements de design.

Pour les raisons citées ci-avant, la présentation de l'interface web a été développée manuellement ce qui a permis de délivrer un code mieux structuré et épuré.

f. Environnement de tests

Les tests unitaires ont été développés à l'aide de la bibliothèque PHPUnit. Cette bibliothèque a été configurée afin de charger un fichier PHP qui s'occupe de simuler le chargement du *framework* CodeIgniter. Cela est nécessaire car CodeIgniter a été conçu afin de répondre à des requêtes HTTP lorsque le serveur Web exécute un script contrairement aux tests unitaires qui s'utilisent en console et ne génèrent pas de requête HTTP.

L'outil d'intégration continue Jenkins a été utilisé et est lié au compte Github où nous développons le projet. Ainsi, il est possible d'automatiser une batterie de tests unitaires lorsqu'une modification du code a été validée.

L'outil sélénium est utilisé afin de valider chacune des fonctionnalités du portail Web. En effet, cet outil permet d'automatiser des tâches dans le navigateur Web Firefox.

g. Gestion du code (et documentaire)

Pour la gestion du code, l'outils de versioning GitHub a été utilisé. Cet outil est le plus utilisé et n'est plus à présenter. L'internet regorge de documentation à ce sujet, de nombreux tutoriels sont disponibles pour apprendre à l'utiliser et une quantité non négligeable de plugins existe pour l'intégrer à tout environnement de travail.

Afin de centraliser les informations, la documentation a aussi été enregistrée sur le github du projet dans un dossier dédié.

Le logiciel de génération de documentation doxygen a été utilisé car il est multi plateforme et il peut être utilisé par quiconque, sans nécessairement utiliser un terminal en ligne de commande. De plus, ce système est compatible avec de nombreux langages et l'utilisateur peut générer des documentations en pdf, fichier texte et même sous forme de site web.

h. Environnement de développement

L'IDE (integrated development environment) utilisé pour le projet était Eclipse. Il a été choisi car il est gratuit et possède une importante communauté d'utilisateurs, ce qui permet de trouver facilement de la documentation disponible sur internet. De plus, cet éditeur est multi plateforme et peut être utilisé pour programmer des applications dans différents langages.

2.3. Organisation et répartition du travail

Les tâches au sein de l'équipe de développement ont été réparties par *pair programming* comme le prévoit la méthodologie XP.

La figure ci-dessous schématise les différentes parties prenantes du projet ainsi que leur rôle.

La répartition initiale a été préservée mais Aurélie Cools a également contribué à la partie responsive et Jean-Sébastien Lerat à la partie contrôleurs.

Le descriptif de chaque rôle est repris ci-dessous :

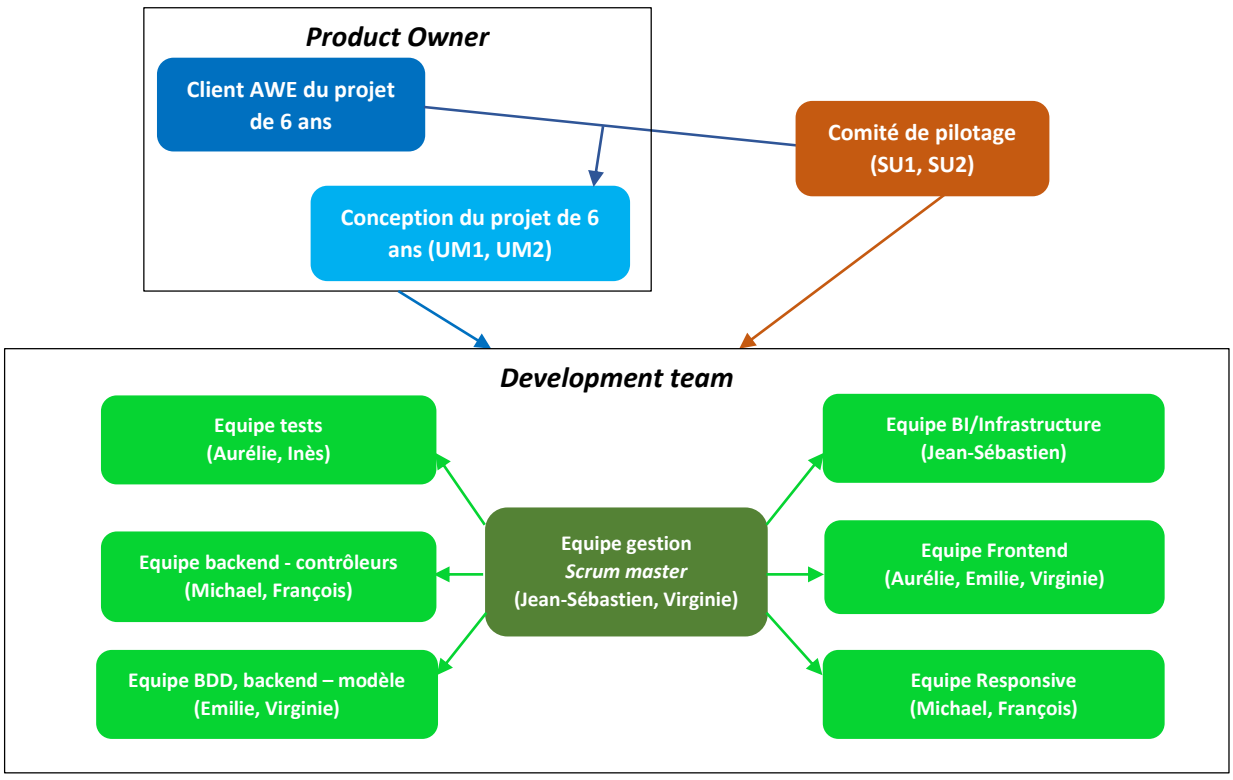

FIGURE 2 : PARTIES PRENANTES DU PROJET ET RÔLES RESPECTIFS

Conformément aux évolutions décrites dans le Project Initiation Document de la méthodologie PRINCE 2, quelques modifications ont été apportées dans la répartition des tâches. En effet, JSL s'est occupé d'une partie des contrôleurs (en particulier l'interaction avec Zeppelin) tandis que ACO s'est chargée d'améliorer la partie responsive.

Équipe	***Description***
Client AWE	Représente l'AWE, client du projet de 6 ans. Décrit les besoins à terme.
Conception projet 6 ans	Développement de la solution au terme des 6 ans. Décrit les besoins initiaux afin de démarrer le projet du groupe 1.
Comité de pilotage	Supervise le présent projet et le projet de 6 ans. Support en cas de problèmes.
Gestion	Coordonne l'organisation du groupe et s'assure que le cadre de travail *scrum* est respecté.
Tests	Développe les tests unitaires et les tests fonctionnels dans un environnement d'intégration continue (*tests driven*).
BDD	Conçoit la base de données qui sera exploitée par le site Web (gestion des utilisateurs, des droits d'accès aux différentes sources de données).
BI/Infrastructure	Conçoit et prépare l'infrastructure de développement/tests et met en place l'interfaçage des données vers l'exploitation des données.
Site *frontend*	Développe l'affichage graphique du site Web.
Site *backend*	Développe le noyau du site (PHP) en suivant le motif MVC.
Site *responsive*	Met en place une adaptation graphique du site Web afin de rendre le site *responsive*.

FIGURE 3 : DESCRIPTIF DES RÔLES

3. Développement du projet

Dans ce chapitre, nous débuterons par décrire de manière globale l'architecture du projet. Ensuite, nous détaillerons la plateforme web qui a été réalisée dans le cadre de ce projet. Par la suite, nous parlerons de la partie business intelligence, transiterons par les tests unitaires et terminerons par les résultats obtenus.

3.1. Architecture

L'architecture du portail Web suit le patron de conception MVC mis en œuvre par CodeIgniter. Nous avons donc des vues dans le dossier *application/views*, des contrôleurs dans le dossier *application/controllers* et des modèles dans le dossier *application/models*. De plus, il y a un fichier dit *helper* qui permet d'interagir avec Zeppelin, l'outil de centralisation des données.

Concernant l'infrastructure de la *proof of concept*, nous avons adopté deux machines virtuelles sous Ubuntu[4] : *data-sources* et *zeppelin*. La machine data-sources possède le serveur Web Apache ainsi que le système de gestion de base de données relationnelles (SGBDR) MariaDB afin d'héberger le portail Web de WalleSmart. Elle héberge également PostgreSQL, un autre SGBDR, un Web service qui utilise la technologie REST et un espace de stockage de fichiers. Ceci afin de démontrer la faisabilité d'une telle solution qui permet l'interopérabilité des sources de données également appelées applications. En parallèle, nous utilisons Jenkins qui est un outil d'intégration continue et qui est également exécuté sur cette machine afin de valider chaque version du portail Web. La machine *zeppelin* quant à elle, n'héberge que zeppelin et se connecte via le réseau aux autres sources de données. De plus, cette machine utilise le protocole *Network File System* (NFS) afin de lire les fichiers stockés sur la machine *data-sources* comme s'ils étaient stockés sur la machine *zeppelin*. Cette infrastructure est hébergée au campus technique de la Haute École en Hainaut (HEH). Afin de pouvoir y accéder en toute sécurité, une demande d'accès VPN a été faite. Ainsi chaque membre a pu se connecter à un sous-réseau distant de la HEH pour manipuler les machines. La figure suivante illustre le schéma de principe de l'architecture précédemment décrite :

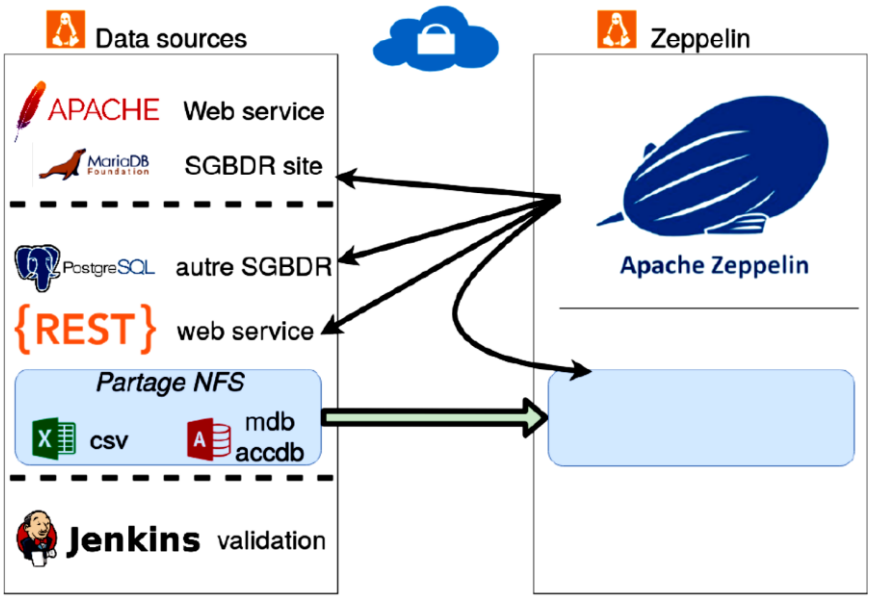

FIGURE 4 : ARCHITECTURE DE LA SOLUTION

3.2. Développement du projet

Dans cette section, nous commencerons par présenter le modèle conceptuel de la base de données. Comme le portail web a été développé avec le framework CodeIgniter et en utilisant une architecture modèle-vue-contrôleur (MVC), nous poursuivrons avec le backend du développement web, à savoir le modèle et le contrôleur. Nous présenterons ensuite la vue que nous nommerons frontend. La présentation des méthodes utilisées pour rendre l'interface responsive clôturera cette section.

[4] Une distribution du système d'exploitation GNU/Linux.

3.2.1. Base de données

Pour rappel, le SGBDR utilisé pour la gestion centralisée des informations est MariaDB et l'équipe BDD a porté une attention toute particulière au bon respect des 3 premières formes normales durant la conception de la base de données.

Afin de répondre aux besoins spécifiés (voir chapitre 1.2), le modèle conceptuel des données a été réalisé (voir figure ci-dessous).

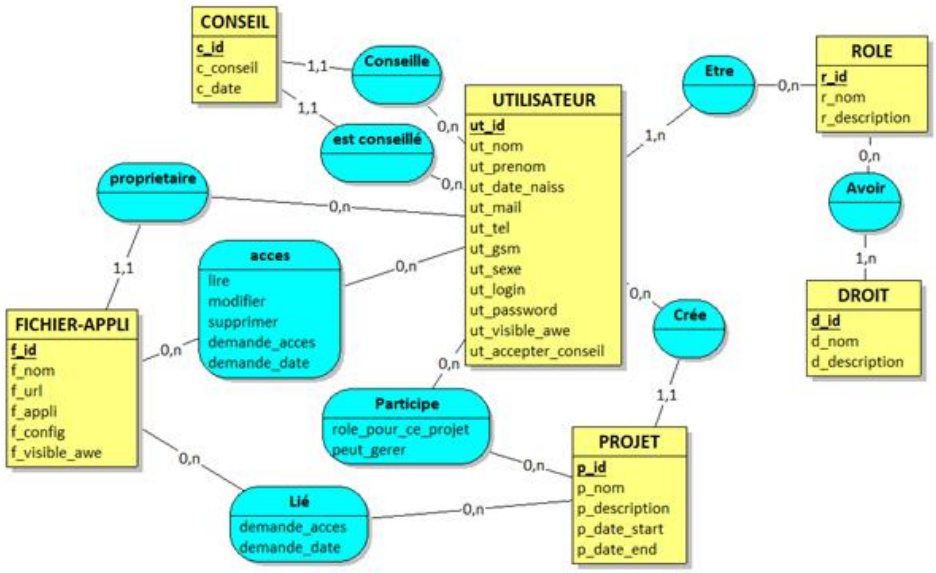

FIGURE 5 : MCD DE LA BASE DE DONNÉES CENTRALE

On y trouve les entités suivantes :

- **Utilisateur :** informations relatives à chaque utilisateur ainsi que sa visibilité pour l'AWE et son choix de vouloir recevoir des conseils ou non.
- **Rôle :** scientifique, développeur, admin, etc. Un utilisateur peut avoir plusieurs rôles.
- **Droit :** droits d'accès aux fonctionnalités de la plateforme (quel rôle peut créer des comptes, quel rôle peut ajouter des applis, etc.)
- **Conseil :** cette entité contient un conseil (texte), une date de conseil et quel utilisateur à rédiger le conseil pour quel autre utilisateur.
- **Fichier-appli :** afin de simplifier la BD, la gestion des fichiers et applications est fusionnée dans une seule entité. Au niveau des attributs, il y a :
    - le nom du fichier/appli ;
    - l'url d'accès ;
    - f_appli est un booléen permettant de savoir si c'est un fichier ou une appli ;
    - f_config pourrait être un JSON de configuration d'appli ;
    - f_visible_awe est utilisé pour les fichiers, vaut 1 si l'agriculteur veut bien rendre visible son fichier pour l'AWE.

  Indépendamment de la valeur du champs f_visible, un agriculteur doit pouvoir décider d'autoriser l'accès à d'autres utilisateurs (un collègue, un scientifique, etc.) ce qui justifie l'association « accès ».

- **Projet :** les scientifiques doivent pouvoir créer un projet, ajouter des collègues à ce projet (en fournissant éventuellement un petit texte pour attribuer un role_pour_ce_projet à leur collègue) et en définissant si leur(s) collègue(s) peut(vent) gérer aussi le projet. Pour cette BD, gérer un projet se limite à y associer des fichiers et/ou applis.

3.2.2. Backend

3.2.2.1. Modèle

Pour le modèle, l'organisation a été choisie pour modulariser le modèle en restant cohérent avec le concept suivi pour la base de données. Le modèle exploite donc l'implémentation physique de la base de données (cf. figure ci-dessous)

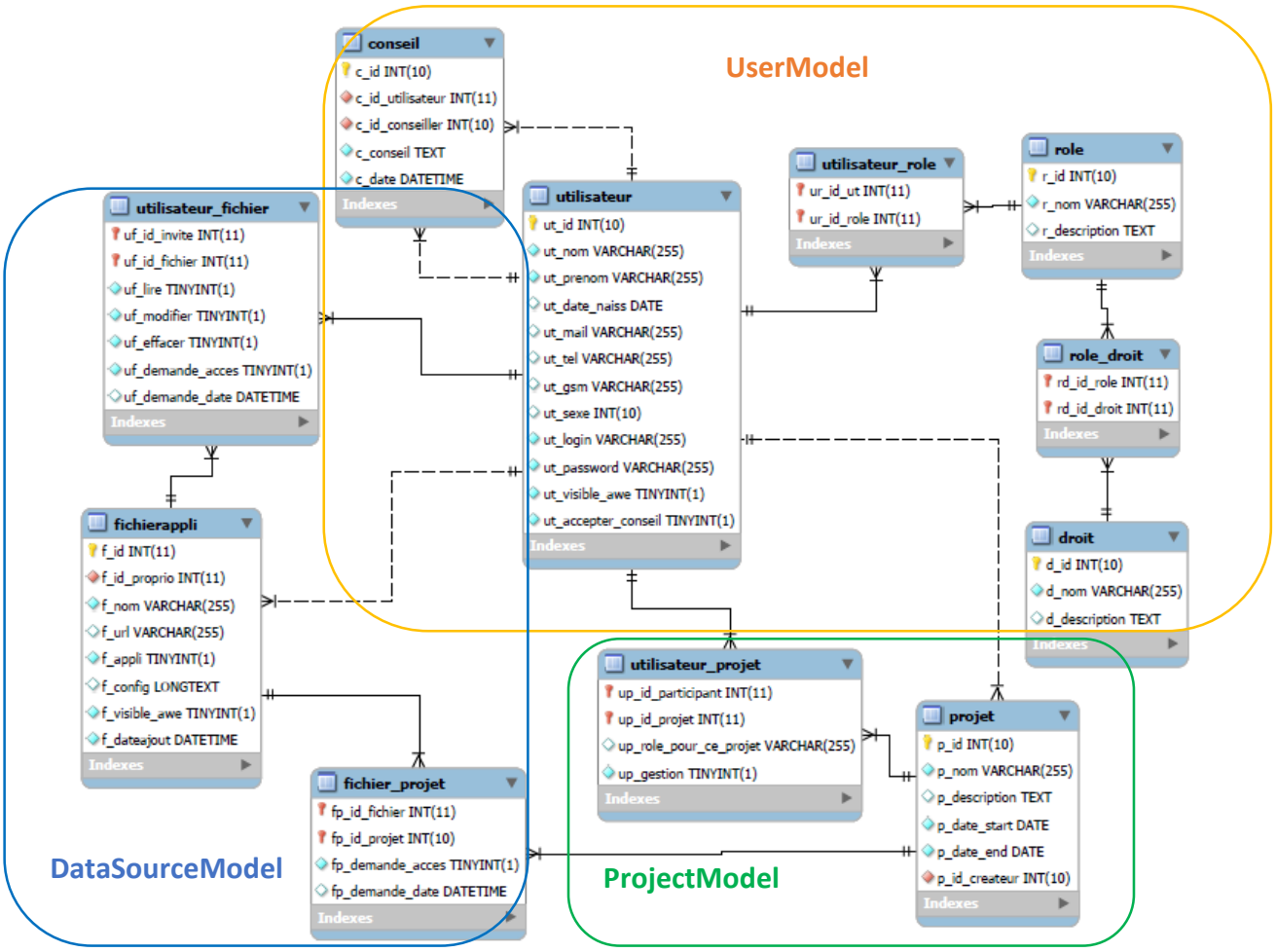

FIGURE 6 : RÉPARTITION DES CLASSES DANS LE MODÈLE DU BACKEND PAR RAPPORT AU MODÈLE DE LA BASE DE DONNÉESS

Nous avons préféré ne pas utiliser d'ORM afin de fournir un modèle adaptable et optimisé. Par adaptable, nous entendons que les ORM exigent généralement un lien fort avec le SGBD utilisé. Dans

notre cas, seul le changement du driver de connexion est requis en cas de changement de SGBD ce qui offre une plus grande flexibilité. Par optimisé, nous entendons 2 caractéristiques importantes :

- Les requêtes SQL ont été implémentées dans le modèle manuellement en utilisant toute la puissance du langage SQL (jointures, indexation, etc.) afin d'accéder à la base de données de manière efficace. Les ORM, avec leur génération de SQL automatique ne peuvent rivaliser en termes de performances[5].
- Cette approche a également permis de restreindre le modèle à seulement 3 classes, qui sont, de plus, de tailles contenues ce qui rend l'utilisation, la maintenance et l'évolutivité aisé. Les 3 classes sont :
    - **UserModel** : qui gère les données des utilisateurs ainsi que leurs rôles et droits. Cette classe comprend aussi la partie « conseil » ;
    - **DataSourceModel** : pour la gestion des informations relatives aux sources de données et à leur accessibilité pour l'AWE. Cette classe prend aussi en compte les accès des invités et des projets qui utilisent ces sources de données ;
    - **ProjectModel** : qui gère les informations relatives aux projets et aux membres participants à ces projets.

Ces classes n'ont donc pas d'attributs puisqu'elles sont exclusivement composées de méthodes qui interagissent avec la base de données par des opérations de type CRUD (create, read, update, delete).

3.2.2.2. Contrôleur

Nous avons implémenté neuf contrôleurs qui peuvent regrouper plusieurs pages, chacun étant responsable d'une fonctionnalité :

- **Administration** permet de gérer les projets de recherche ainsi que les utilisateurs associés à ces projets.
- **Connection** permet d'authentifier un utilisateur et de lui créer une session qui l'identifie de manière unique sur le serveur Web.
- **Datasource** permet à n'importe quel utilisateur de manipuler ses propres sources de données et les droits qui y sont associés. C'est-à-dire quel utilisateur ou quel projet a l'autorisation d'accéder à quelle source de données détenue par l'utilisateur.
- **Help** affiche la liste des contributeurs au projet WalleSmart.
- **Home** affiche la page d'accueil avec le compteur de visite.
- **Profil** permet à l'utilisateur de gérer son profil, c'est-à-dire ses données personnelles d'identification ainsi que de supprimer son compte.
- **Register** permet à un utilisateur de s'enregistrer sur le site Web afin d'y gérer ses données et de contribuer au projet.
- **Rest** fournit un ensemble de Webservice RESTful qui nécessite des informations d'identification en méthode POST du protocole HTTP.
- **Search** permet de rechercher un utilisateur ou un projet.

[5] Toon Koppelaars, *"NoPLSQL and Thick Database Approaches"*, Oracle Learning Library, 2016
url : https://www.youtube.com/watch?v=8jiJDflpw4Y - visité le 22/05/2019

3.2.3. Frontend

Nous allons désormais vous présenter l'interface utilisateur du site Wallesmart. Pour travailler sur l'interface, nous sommes partis d'un point de base qui a été la création d'un logo et une icône. En effet, aucun logo n'était disponible, nous avons donc créé celui-ci pour qu'il soit la base couleur de notre site.

FIGURE 7 : LOGO ET ICÔNE DE WALLESMART

Nous avons donc choisi comme base de couleurs pour le site le bleu rappelant l'eau, le jaune rappelant le blé et le vert rappelant la couleur des feuilles ou encore de l'herbe. Ces 3 couleurs rappellent donc de manière générale la nature qui correspond bien au thème du sujet Wallesmart.

Une fois connecté, nous avons la page suivante. Celle-ci se constitue d'une bannière comprenant le logo qui sera commune à toutes les pages du site ainsi que d'une barre de menu qui sera également présente sur chaque page.

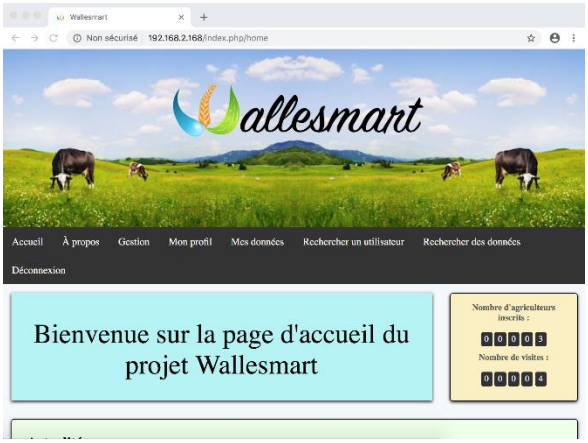

FIGURE 8 : PAGE D'ACCUEIL LORS DE LA CONNEXION

Hormis la zone comprenant les compteurs, chaque page aura la même structure. Ainsi, nous retrouverons la bannière avec le logo et le menu, ensuite le titre de la page où nous nous trouvons dans un cadre bleu et ensuite les informations ou formulaires de la page en question. Vous pourrez également retrouver dans l'onglet du navigateur l'icône Wallesmart que nous avons créé.

Pour aider l'utilisateur au mieux dans sa navigation, le menu sera toujours présent sur la page. En effet, nous avons fait en sorte que l'utilisateur, une fois arrivé en bas d'une page, n'ait pas besoin de remonter toute la page pour pouvoir se rediriger. Une fois que l'utilisateur descend sur la page, le menu reste dans le haut de la page, comme vous pouvez le voir sur la seconde image.

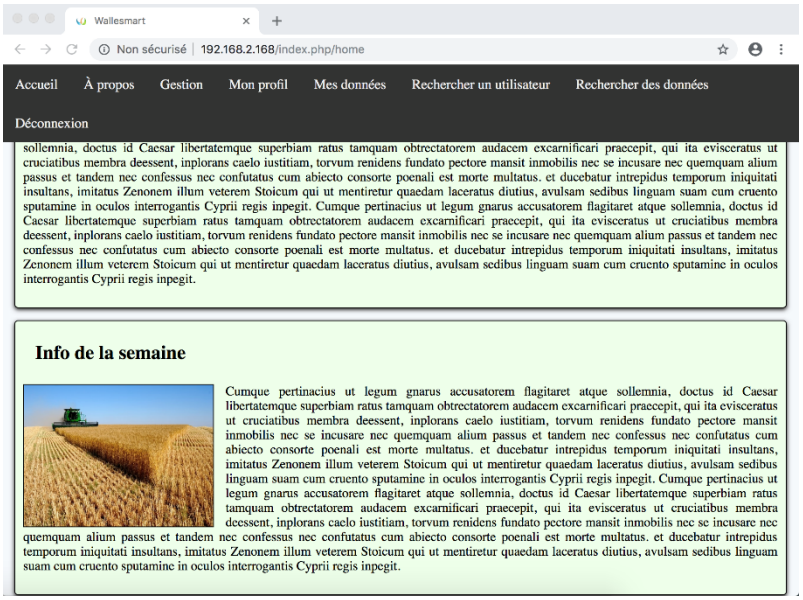

FIGURE 9 : VISUALISATION DU MENU LORS DE LA NAVIGATION

Toujours dans le but d'aider au mieux l'utilisateur et que notre site soit user-friendly, l'utilisateur peut retrouver relativement aisément le nom d'un utilisateur en entrant seulement le début, comme le montre l'image ci-dessous.

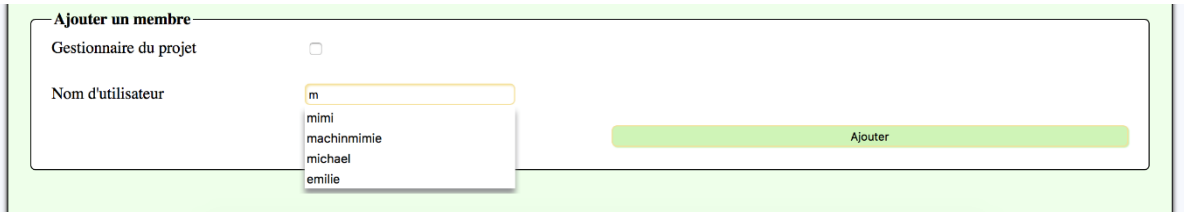

FIGURE 10 : VISUALISATION DE LA RECHERCHE SUR BASE D'ENTRÉE CLAVIER

3.2.4. Responsive

Nous avons développé un site qui est responsive et qui est donc accessible sur tous les écrans et tout type de smartphone ou tablette. Le responsive peut se voir à différents niveaux dont nous allons détailler et illustrer certains ci-après. Tout d'abord, le menu s'adapte à la taille de l'écran comme vous pourrez le constater à la figure suivante.

FIGURE 11 : VISUELS DE LA PAGE D'ACCUEIL EN MODE DESKTOP ET EN MODE MOBILE

Vous pouvez constater que le menu qui est un menu en bar, se réduit en un unique bouton avec un bouton de liste déroulante pour afficher la suite.

FIGURE 12 : VISUEL DU MENU DÉROULANT

Vous aurez également constaté que l'image de présentation avec le logo s'adapte également à la taille de l'écran. Enfin, nous pouvons toujours, sur ces mêmes figures, voir que les compteurs affichant le nombre d'agriculteurs inscrits et le nombre de visites journalières s'affichent à des positions différentes selon la taille de l'écran sur lequel nous visitons le site.

3.2.5. Business Intelligence infrastructure

L'infrastructure Business Intelligence (BI) se résume à l'utilisation de Zeppelin, ce qui nous a fait économiser du temps, qui assure une pérennité logicielle ainsi qu'une stabilité plus élevée que ce que nous aurions pu implémenter en trois mois en parallèle du portail Web. Zeppelin s'exécute sur une machine et permet d'interconnecter diverses sources de données locales ou distantes. De plus, Zeppelin permet d'exécuter des requêtes soit sur la machine locale soit en envoyant la charge de travail via Hadoop (Spark).

3.3. Tests

Les tests unitaires ont été réalisés à l'aide de l'environnement Jenkins. Jenkins est open-source. Il a été configuré de sorte à pouvoir tester des fichiers présents sur le dépôt Github utilisé pour le traçage de l'évolution du code. Le code des tests est mis à jour et tous les 3 jours, on réalise automatiquement l'ensemble des tests à cette occasion. Les tests ont été écrits en PHP et testent les méthodes des modèles de contrôleur. Ces tests ont été automatisés de sorte à pouvoir être indépendant de la base de données.

Nous avons travaillé cycliquement de la manière suivante :

- On prépare un ensemble de tests.
- On lance la compilation des tests sur Jenkins
- On corrige l'ensemble des erreurs liés à la programmation des tests

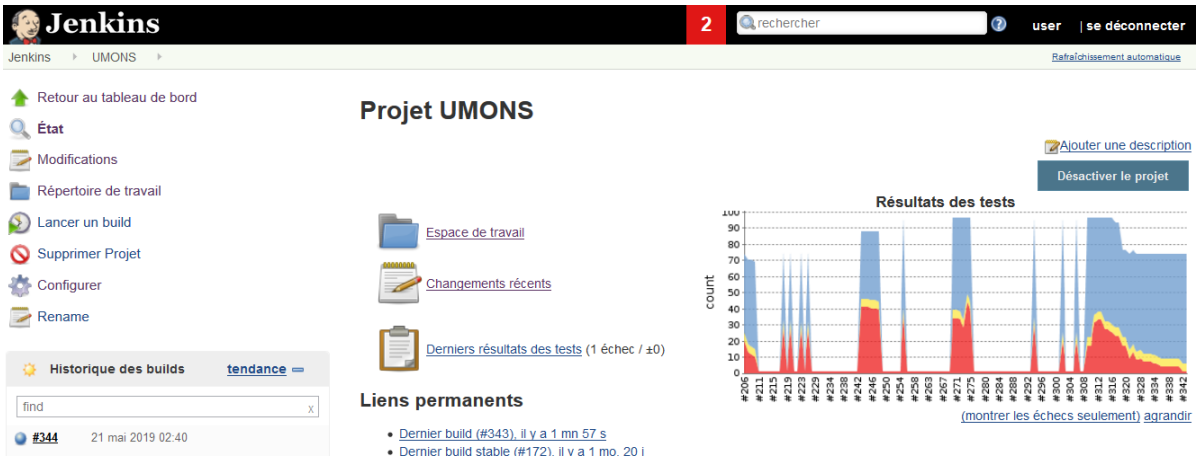

FIGURE 13 : ENVIRONNEMENT DE TEST JENKINS

Le graphique qui illustre le résultat des tests correspond se compose de trois codes couleurs : bleu signifie un test réussi, orange un test échoué et en rouge une erreur. L'abscisse correspond à l'évolution temporelle et l'ordonnée correspond au nombre de tests. Nous pouvons voir que la proportion de tests augmente avec le temps et que le nombre de succès augmente contrairement au nombre d'erreur et d'échec.

De plus des tests fonctionnels ont été réalisés grâce à la librairie "selenium" qui permet d'automatiser les tests sur un explorateur web. Ces tests ont donc été codés en langage Python et permettent de tester l'interface du projet. On teste donc l'ensemble des niveaux du projet c'est à dire les données, les contrôleurs et l'interface graphique. Les tests fonctionnels sont donc plus complets que les tests unitaires car on teste l'interface en plus.

3.4. Présentation des résultats

Les résultats de notre travail ont été très fructueux. En effet, nous avons créé un site internet qui prouve qu'il est possible de gérer des utilisateurs différents en leur permettant d'ajouter des données provenant de sources diverses.

Notre site gère donc les utilisateurs, en permettant l'inscription, la connexion/déconnexion, la suppression du compte. Mais également, la modification des données personnelles en incluant la possibilité de rendre ses données visibles ou non selon le règlement général de la protection des données. La figure n°11 reprend une synthèse des fonctionnalités du site.

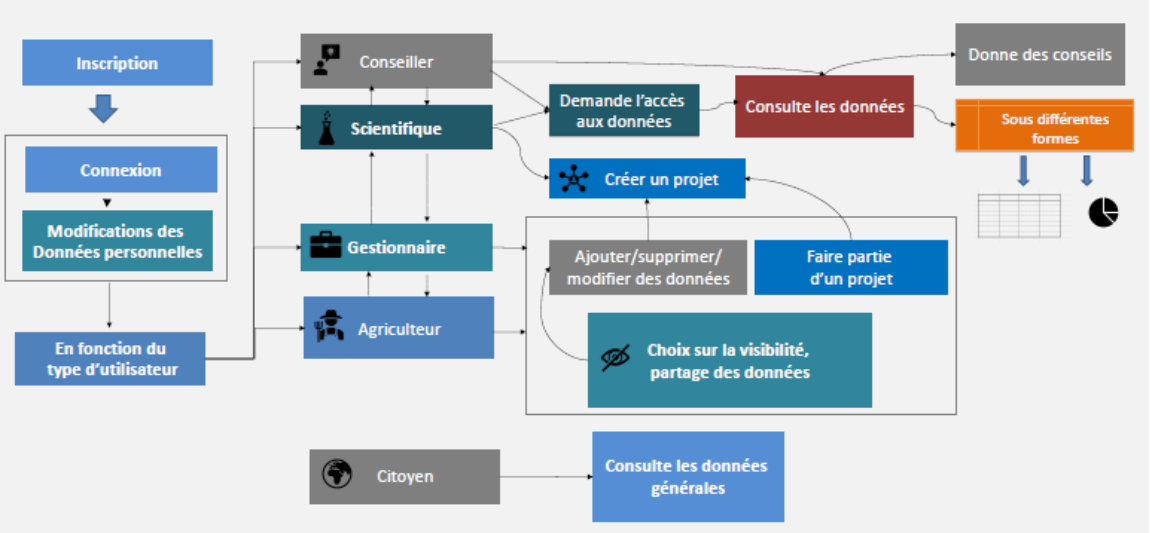

FIGURE 14 : FONCTIONS GÉNÉRALES DU SITE

Nous allons illustrer ici la fonction la plus spécifique qui est l'affichage des données provenant de sources différentes. Pour le reste des fonctions, nous vous invitons à consulter le manuel d'utilisation qui liste toutes les fonctions avec une aide pour pouvoir utiliser le site.

Une fois connecté, l'utilisateur peut gérer ses données grâce à l'onglet "Mes données". Il peut alors ajouter des données, les consulter, les modifier, les afficher ou encore les supprimer.

FIGURE 15 : PAGE DE L'ONGLET "MES DONNÉES"

A partir cette page, l'utilisateur peut afficher les données qui sont déjà en ligne via le menu déroulant et en cliquant ensuite sur "Charger".

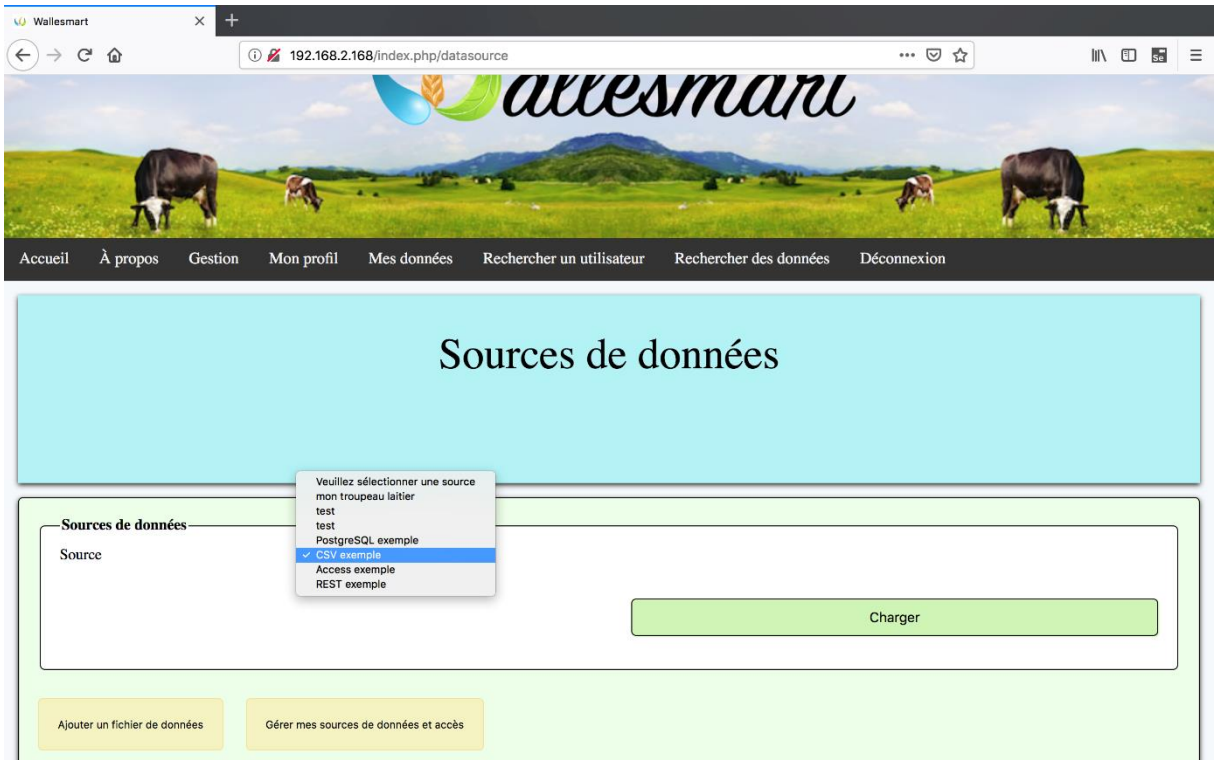

FIGURE 16 : CHOIX DES DONNÉES À AFFICHER

Vous trouverez ci-dessous les différents exemples des fichiers qu'il est possible d'afficher. Nous pouvons afficher des fichiers de type REST, Access, POSTGRESQL et CSV. Vous trouverez pour chaque type de source de donnée, un exemple en image ci-après.

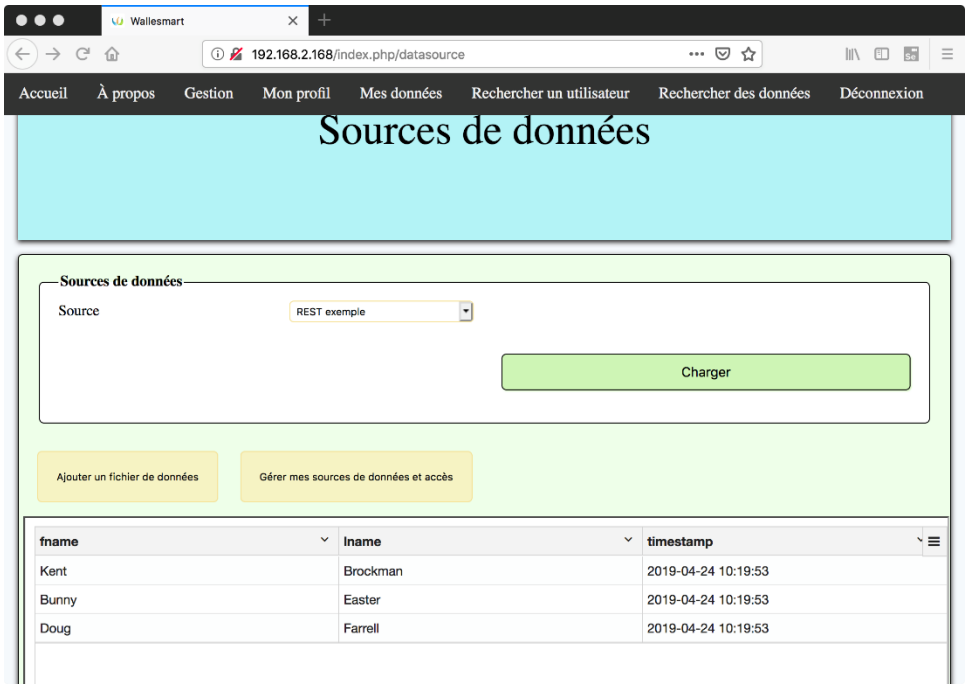

FIGURE 17 : EXEMPLE DU FICHIER REST

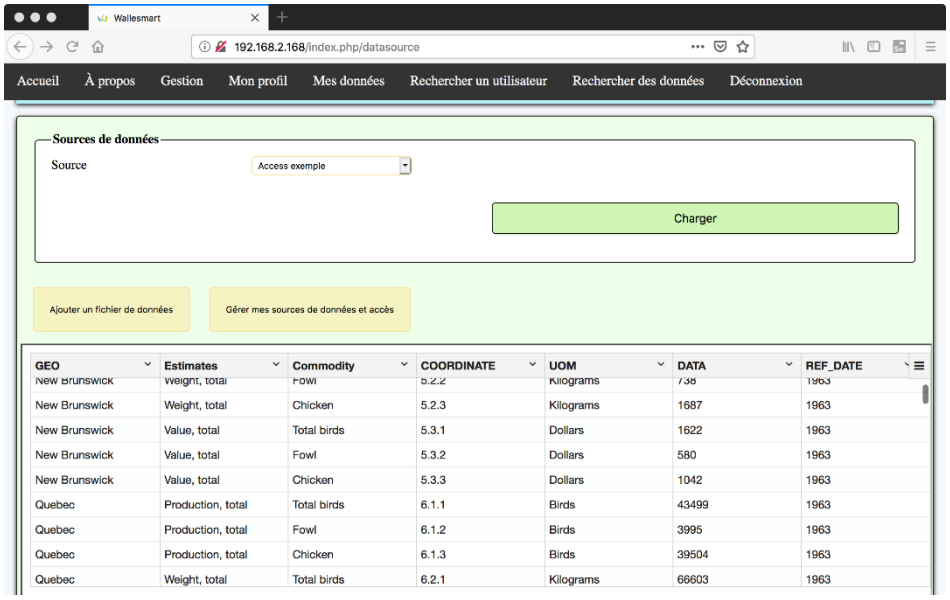

FIGURE 18 : EXEMPLE DE FICHIER ACCESS

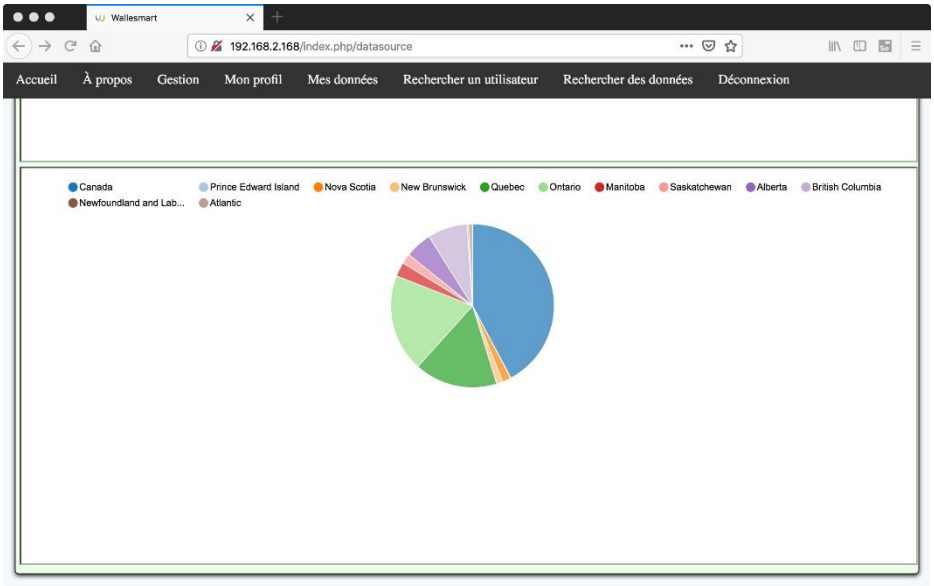

FIGURE 19 : AUTRE EXEMPLE DE FICHIER ACCESS

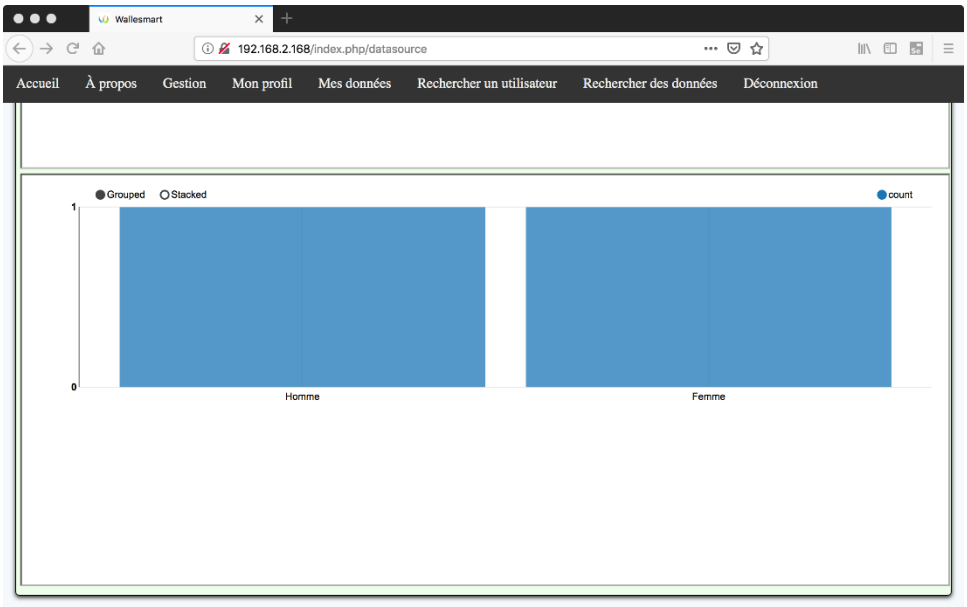

FIGURE 20 : EXEMPLE DU FICHIER POSTGRESQL

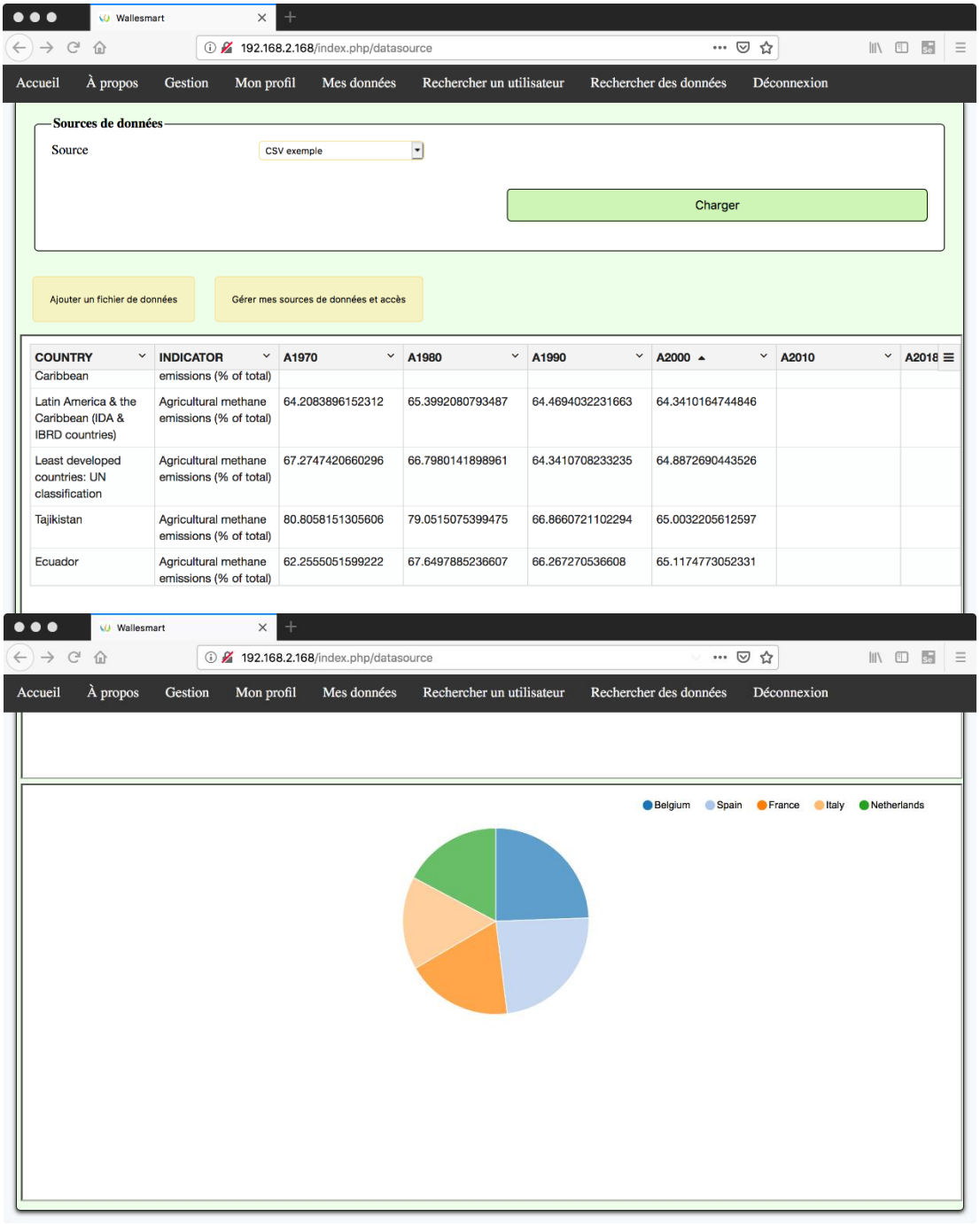

FIGURE 21 : EXEMPLE DE FICHIER CSV (2 IMAGES)

4. Bilan et performances

4.1. Bilan du projet

En appliquant la méthode COCOMO II uniquement sur les lignes de codes qui ont été produites au terme du projet, on obtient 2567 lignes logiques de code source sans les tests. Ces lignes prennent en compte l'implémentation du MVC dans CodeIgniter, les fichiers *helpers* et le code HTML-CSS-Javascript.

COCOMO II a été configuré à l'aide des paramètres suivants :

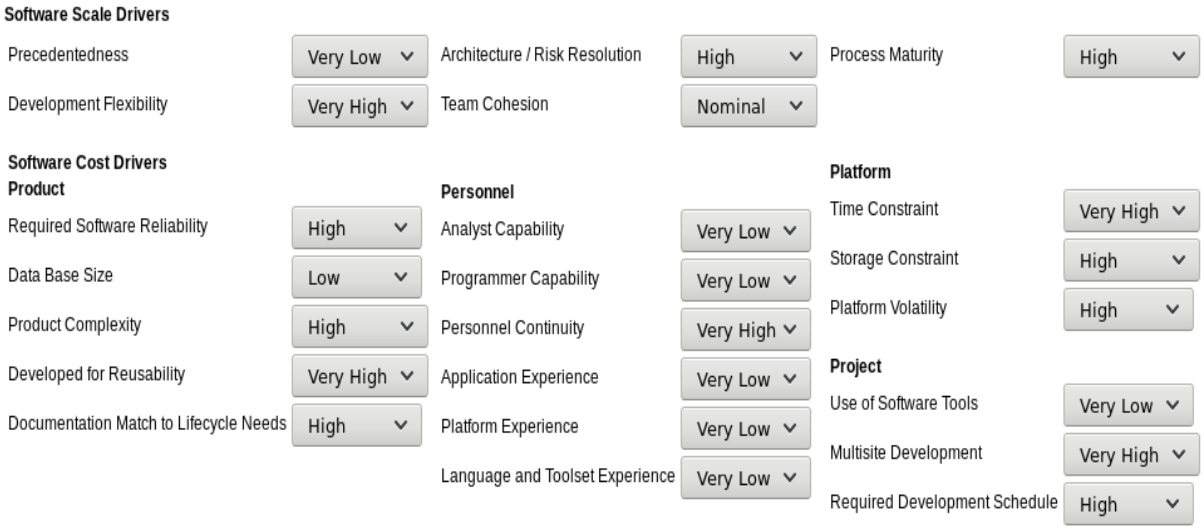

FIGURE 22 : PARAMÈTRES DE COCOMO II

Les paramètres principaux sont la fiabilité (haute) et l'expérience des développeurs (faible) qui ont un impact significatif sur le coût du projet WalleSmart. Avec ces paramètres, à raison de 3000$ brut par mois par développeur, l'application a un coût total de développement de 151 135$. En effet, le nombre de personnes-mois est de 50,4. Soit 7,2 mois par personne réalisé en 3 mois. Toutefois, si on s'intéresse au *scheduling*, le temps de développement est moins élevé et réalisable dans les temps. En effet, l'homme-mois représente l'effort à réaliser.

4.2. Respect des délais

Pour gérer notre projet, nous avons élaboré un planning se basant sur 8 histoires/sprints. Nous nous sommes donnés un délai de 15 jours pour la réalisation de chaque histoire. En principe, il aurait été plus confortable de faire des sprints d'un mois mais cela était impossible vu le temps imparti à l'ensemble du projet.

Le délai de la première histoire n'a pas été respecté, à cause d'une part la prise en main des logiciels et d'autre part la réception du matériel. Cela a eu des répercussions sur la suite du projet mais nous avons pu l'anticiper ce qui a pu réduit le retard par la suite.

A la moitié du délai du projet, le retard avait été rattrapé. La première raison était que le projet était assimilé et que chaque membre connaissait ses taches. Le rendement en a été augmenté. Les différents outils de planification, de communication instantanée et d'échanges d'informations ont apporté une grande aide à l'avancement du projet. Une bonne élaboration initiale des histoires a aussi profité au projet.

Respecter les délais n'est pas une chose facile à la vue des différents imprévus mais nous avons remarqué qu'une bonne équipe soudée et le suivi du planning peuvent faire aboutir un projet conséquent.

4.3. Qualité du code

Une rapide analyse via sonarqube a été réalisée sur les fichiers applicatifs (modèles, contrôleurs et vues) et les assets afin de vérifier la qualité du code fourni.

Les résultats obtenus pour cette analyse sont évalués principalement sur base des sources bibliographiques suivantes :

- Dr ir R. Viseur, « Evaluation technologique des logiciels », 2018 ;
- P. Mengal, « Métriques et critères d'évaluation de la qualité du code source d'un logiciel », 2013.

La figure ci-dessous reprend un résumé du résultat d'analyse fourni par SonarQube.

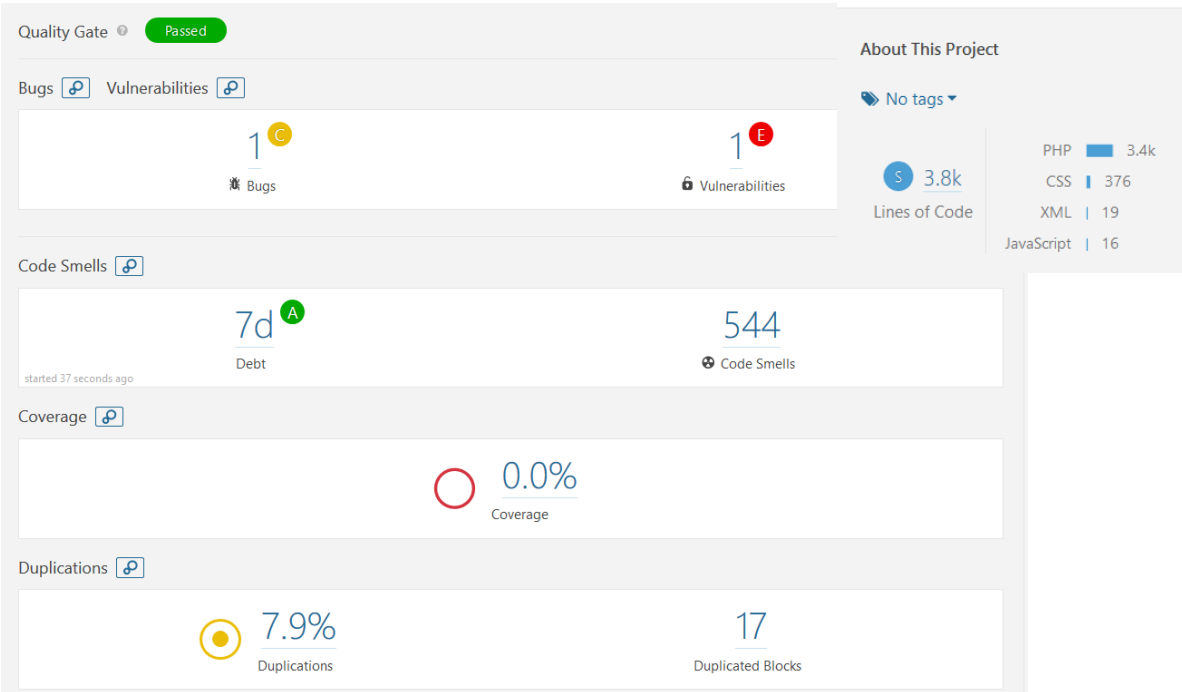

FIGURE 23 : RÉSUMÉ D'ANALYSE FOURNI PAR SONARQUBE

L'analyse montre environ 3800 lignes de codes, pour 16 classes et 164 méthodes.

Un bug a été détecté qui, après analyse, est lié à une mauvaise interprétation de sonarQube. Pour la vulnérabilité, elle est liée à la dénomination de la variable « $password ».

Les code smell impliquent un ratio de dette technique de 2.8%, ce qui est très bon (niveau de maintenabilité de A). Un bon nombre d'entre eux sont liés à des commentaires ou à des accolades qui n'ont pas été introduites car pas nécessaires.

Le pourcentage de commentaires est de 17.4%. La valeur minimale recommandée est de 20% ce qui est assez proche de la valeur mesurée.

Le taux de duplications est de 7.9%, ce qui est supérieur à 3% maximum recommandé par la littérature. Malgré tout, le taux de duplication est assez faible.

Enfin, la complexité cyclomatique globale est de 661. Cela donne une complexité cyclomatique moyenne par méthode très bonne, d'environ 4.

Il semble que pour pouvoir afficher l'information sur la couverture de code, un rapport de couverture doit être généré avant l'analyse par sonar-scanner. Cette valeur n'a pas pu être évaluée et donc, le taux de couverture de code fourni est erroné.

5. Conclusions

À l'issu de ces trois mois de développement, nous pouvons affirmer que la plateforme créée est en adéquation avec les besoins et attentes de notre client.

Nous avons particulièrement veillé à ce que le site soit de manipulation intuitive pour l'utilisateur tout en offrant les fonctionnalités demandées.

La réalisation de ce projet a nécessité la prise en main de bons nombres de technologies ce qui a généré du retard au début du développement. Mais ce retard a été progressivement résorbé et finalement nous avons réussi à achever ce projet dans les temps.

Bien que nous étions chacun responsable d'une partie du projet, nous devions fréquemment interagir ensemble afin que tout un chacun partage une même vision globale des objectifs à atteindre.

La réalisation du projet fut chronophage et il fut difficile d'assurer sa réalisation malgré nos obligations professionnelles.

Cependant nous sommes fiers du niveau d'aboutissement dans les délais impartis.