Département Informatique
IUT Bordeaux1
15 rue Naudet
33175 Gradignan Cedex

Ergolis
17 CRS Xavier Arnozan
33000 BORDEAUX

Création d'une application Mobile pour la traçabilité agricole

STAGE DE DUT REALISE PAR

Julien LEVEAU

Du 10 Avril au 13 Juin 2014

Maître de stage : M. Brice TEXIER
Enseignant responsable : Mme Laure CURVALE

Résumé :

La société Ergolis souhaite permettre aux utilisateur de leur ERP libre Ekylibre de réduire leur saisie clavier. Il a donc été décidé d'apporter une application mobile de tracking à Ekylibre.

L'objectif de ce stage est de développer cette application depuis le début, en s'appuyant sur des méthodes agiles. Le rapport présente le contexte du stage, et les besoins d'Ergolis. Il détaille le projet Ekylibre dans lequel il s'inscrit, ainsi que l'environnement et les outils de développement utilisés.

Abstract :

The company Ergolis aims to allow their FOSS[1] Ekylibre users to reduce keyboarding. For that purpose they wish to provide a tracking mobile application for Ekylibre.

The objective of this internship to develop this application from scratch, using agile approach. This report presents this internship context and the requirements of Ergolis. It details Ekylibre to which it belongs, the tools and the environement used.

[1] Free Open Software

Remerciements

Avant tout, je souhaite remercier toutes les personnes qui m’ont permis de réaliser mon stage dans les meilleures conditions et je tiens tout particulièrement à remercier :

Messieurs Brice TEXIER responsable informatique et David JOULIN chef de projet pour m’avoir permis d’effectuer mon stage au sein d'Ergolis et ce dans d’excellentes conditions.

Je souhaite également remercier les autres membres du SACEA du service juridique fiscal-rural et employeurs pour leur accueil et leur bonne humeur.

Sommaire

INTRODUCTION :

1

Le stage que j'ai effectué s'est déroulé du 10 avril au 13 juin 2014 chez Ergolis à Bordeaux sous la responsabilité de Monsieur Brice Texier.

Ergolis est une société détenue par le SACEA (Service Assistance Conseil des Exploitants Agricoles), constituée de Monsieur Brice Texier responsable informatique et Monsieur David Joulin chef de projet MOA (maîtrise d'ouvrage), elle contribue au développement du logiciel libre Ekylibre. Basé sur le framework Ruby on Rails, Ekylibre est un progiciel de gestion intégré destiné aux TPE[1] du monde agricole.

Les principales fonctionnalités du progiciel sont : une gestion des relations clients/fournisseurs, de la comptabilité générale, de l'aspect commercial, de la production, et de la traçabilité. Le tout avec une gestion de droits de l'utilisateur.

De base Ekylibre permet de réduire la ressaisie d'information. L'objectif est de s'orienter vers un « zéro-saisie » et donc de réduire au maximum, voir supprimer la saisie au clavier pour l'utilisateur grâce a un smartphone, des objets connectés, et des capteurs embarqués. Le sujet de stage qui m'a été confié est de réaliser une application mobile de tracking permettant l'acquisition de coordonnées géolocalisées et horodatées avec la possibilité d'y associer des valeurs numériques et une unité (population, litre, kg, m). L'application devant fonctionner pour le plus grand nombre de smartphones possible.

Ce rapport se présente en trois parties. La première concernera Ekylibre le progiciel d'Ergolis, en détaillant son histoire, les personnes qui participent au projet, et leurs objectifs pour 2014. Dans une deuxième partie je décrirai le sujet du stage, et les outils et méthodes à utiliser. Et enfin dans une dernière partie, nous verrons quel à été l'environnement de travail mis en place, et les méthodes de gestion de projet utilisée, et pour se rendre compte plus en détail nous étudierons l'approche sur point précis du projet.

[1] (Très petites entreprises)

I. Présentation de l'entreprise

A) Ekylibre

1. Historique

En 2005, Michel Gil-Antoli, président de l'ABUL et céréalier, établit avec l'ENSEIRB un avant-projet de SIG[1] adapté aux TPE agricoles, sept étudiants ont travaillé sur l'avant projet. L'un d'eux, Brice Texier, a pu continuer à travailler sur ce projet.

Fin 2007 avec l'aide du Conseil Régional d'Aquitaine, un dossier FEDER, [Fonds Européen de Développement Régional] a permis d'obtenir les ressources permettant d'obtenir un logiciel via création d'une cellule informatique au SACEA, Service d'Assistance et de Conseil aux Exploitants Agricole de la Gironde.

Entre 2010 et 2012, Brice Texier continue à maintenir seul Ekylibre au sein du SACEA, dont il est devenu le responsable informatique.

En 2012, Julien Capdeville, étudiant de l'École nationale supérieure des sciences agronomiques de Bordeaux Aquitaine travaille sur la thématique des systèmes d'informations agricoles et recherche notamment les outils adaptés à la problématique de centralisation et traitement des données agricoles.

Il recense Ekylibre comme étant adapté mais pas assez abouti. David Joulin qui est, à l'époque, enseignant et ingénieur d'étude au laboratoire informatique de l'ENITA suit le travail de Julien de près. David rencontre Brice et ils décident de commencer à collaborer sur le projet pour réaliser un outil adapté aux demandes du terrain (notamment via la dynamique du CETA de Guyenne).

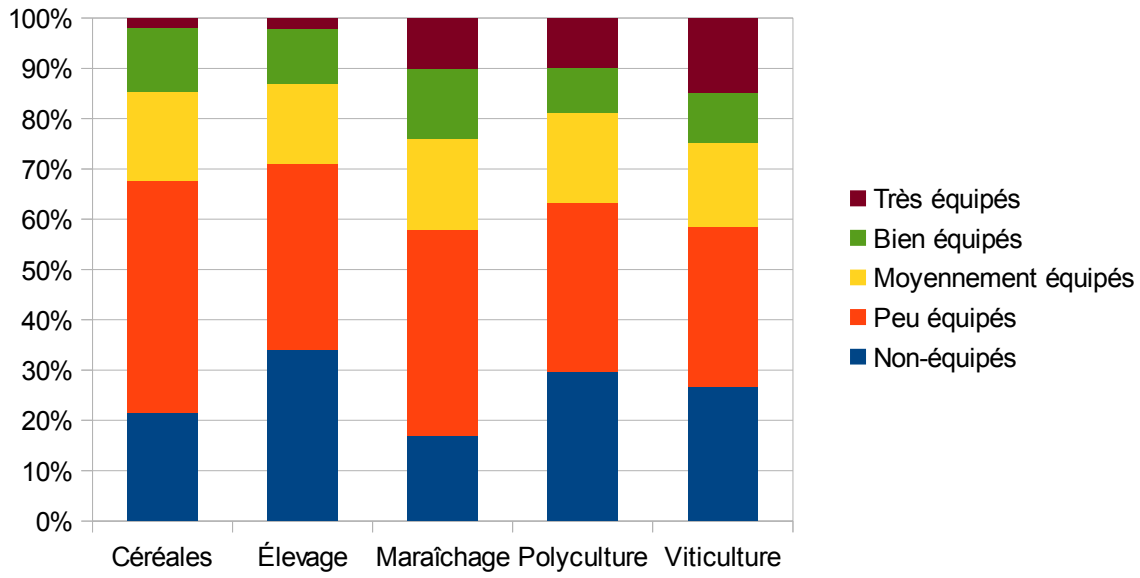
En mai 2013, David Joulin rejoint officiellement le projet avec la volonté de développer l'outil et combler les fonctionnalités métiers manquantes (traçabilité, gestion des interventions, production agricole, cartographie...) afin d'arriver à un outil simple et complet.

2. Raison d'être

Afin de comprendre pourquoi un tel projet, spécifique à une secteur de travail bien particulier (l'agriculture) à vu le jour, nous allons étudier les résultats d'une enquête réalisée par l'OAT (Observatoire Agriculture et TIC) entre le 1er Novembre 2011 et le 8 mai 2012, sur une population de 504 gérants d'exploitation aquitains divisés en 5 sous-populations définies selon la production agricole dominante de l'interviewé : élevage, maraîchage, viticulture, céréales et polyculture.

De plus 5 classes ont été créées pour le taux d'équipement : les « non-équipés » (absence totale d'équipement), les « peu équipés » (un ordinateur et/ou Internet), les « moyennement équipés » (un ordinateur, Internet et 1 outil numérique), les « bien équipés » (un ordinateur, Internet et 2 outils numériques), les « très équipés » (un ordinateur, Internet et au moins 3 outils numériques).

[1] Système d'information géographique



Cette enquête est divisée en 4 parties ayant chacune un sujet différent :

Équipement des agriculteurs en TIC

Cette enquête montre que plus de 60 % des exploitants agricoles sont non-équipés ou peu équipés, que les secteurs les plus en pointe sont le maraîchage et la viticulture et que les facteurs contribuant positivement au taux d'équipement sont la jeunesse, un niveau d'étude et un nombre d'enfants élevé ainsi qu'une taille salariale de l'exploitation importante.

Les exploitants non équipés en TIC

A l'inverse des exploitants à fort taux d'équipement, les personnes non-équipés sont majoritairement des hommes d'âge avancé, ayant reçu une formation courte et travaillant dans de petite structures. Si la majorité d'entre eux n'évoquent pas de raisons particulières à leur refus de s'équiper, celles qui sont avancées sont principalement l'absence de besoin, la complexité et le coût d'investissement.

Représentations des technologies numériques

Les utilisateurs y voient de nombreux avantages principalement un gain d'autonomie et une meilleure communication auprès des consommateurs. Les non-utilisateurs perçoivent eux aussi le gain de temps possible, mais sont pour la plupart rebutés par l'aspect complexe des TIC, et ne souhaitent jamais s'équiper de ce type d'outils dans leur majorité. Enfin 3 agriculteurs sur 4 pensent que l'introduction des TIC augmente la part du travail administratif même s'ils admettent que ceux-ci sont un meilleur moyen pour répondre aux réglementations gouvernementales et pour organiser leurs activités.

La formation au numérique des agriculteurs

Les répondants à l'enquête voient comme trois principaux avantages des outils numériques leur rapidité, leur gain de temps, la précision des informations et leur grande ressource et disponibilité, et comme trois principaux inconvénients leur coût (à l'achat comme à l'entretien), leur complexité et leur évolution trop rapide. Pour la formation initiale, 47 % des sondés sont titulaires du diplôme

BTSA, et plus de la moitié estiment que celui-ci est suffisant pour gérer leur exploitation. Toutes classes d'âge confondues, 65 % des agriculteurs ont suivi une formation continue en informatique, avec un taux de satisfaction qui dépasse les 90 %.

On peut donc tirer comme conclusion de cette enquête que malgré une inertie certaine par rapport aux nouvelles technologies, notamment dans les exploitations agricoles traditionnelles et moins en contact avec les clients (éleveurs, céréaliers), une partie des agriculteurs d'aujourd'hui, plus ouverte et plus formés à l'informatique, est prête à les utiliser au quotidien dans leur profession.

Cependant les rares agriculteurs qui utilisent des progiciels utilisent des progiciels génériques. En effet, il existe très peu de progiciels spécifiques à l'agriculture, et les projets de développement de ce genre de produits par des société informatique ont souvent échoués, notamment à cause du manque de connaissances métiers des développeurs et de la difficulté de communication entre les clients et les développeurs, les logiciels produits étant jugés décevants par les agriculteurs, car trop compliqué à utiliser, et ne correspondants pas à leurs attentes.

La raison d'être d'Ekylibre est donc cette demande qui n'a jusqu'à présent pas été comblé par l'offre des grandes sociétés d'édition logiciel. Le produit se positionne donc dans une niche et se différencie des autres produits par sa démarche de qualité, en effet Ekylibre est développé en s'appuyant sur des connaissances métier, et avec un cycle agile, et non un cycle en V, afin que le logiciel soit ergonomique, facile à prendre en main, et au plus proche des attentes des clients, et par son aspect libre, le progiciel étant disponible gratuitement.

En effet, seuls les services liés au logiciel (installation de l'équipement nécessaire, intégration, hébergement sur un serveur, apport de modification,etc) sont payants.

3. Fonctionnalités

Ekylibre est un logiciel de gestion intégré multi-devises, multi-utilisateurs, multi-dépôts et multi-sites pour les TPE agricoles. C'est une application web (donc multi-postes) basée sur le framework Ruby on Rails et sur la base de données Postgresql.

Le logiciel est distribué sous la licence GNU GPL 3. Il comprend les modules suivants :

- Gestion des relations clients/fournisseurs ;
- Gestion comptable : comptabilité générale et agricole, analytique, emprunts et immobilisations ;
- Gestion commerciale : Facturation des ventes, achats, réglements, remise en banque ;
- Gestion des stocks : Approvisionnement, apports, livraisons, transports ;
- Gestion de la production : Activités, productions, animaux, cultures, interventions, contrôles qualité ;
- Outils : Imports, extractions, reporting, GED, génération des documents réglementaires agricoles et comptables, numérisation, archivage automatique ;

- Le tout avec une gestion de droits à l'utilisateur

Ainsi, lors de la gestion d'une culture, un agriculteur pourra utiliser Ekylibre pour :

- Évaluer ses stocks et calculer la quantité de grains qu'il doit acquérir selon la surface de sa parcelle ;
- Éditer la vente et le transport avec impression des factures en fonction du fournisseur de la date, du lieu, du moyen de transport, etc ;
- Modifier facilement les données concernant ses stocks ;
- Produire automatiquement les écritures comptables en fonction des activités commerciales(ventes, achats, etc).

B) L'entreprise, les personnes

La FDSEA (Fédération des Syndicats d'Exploitants Agricoles) se défini comme la fédération de tous les syndicats communaux de Gironde.

Elle tire son existence de la loi du 21 mars 1884, qui lui assigne l'objet suivant : "Les syndicats professionnels ont exclusivement pour objet l'étude et la défense des droits ainsi que des intérêts matériels et moraux, tant collectifs qu'individuels, des personnes visées par leurs statuts."(art. L.41 1 -1 du Code du Travail).

La FDSEA regroupe aujourd'hui autour de 3000 adhérents, et 146 syndicats communaux ou intercommunaux et se compose d'un conseil d'administration regroupant des délégués cantonaux et des représentants issus des sections sociales, Elle est dirigée par un bureau exécutif issu du conseil d'administration. A Bordeaux, cours Xavier Arnozan nous sommes au SACEA qui fait partie de la FDSEA, et regroupe une direction générale, un service juridique employeurs et un service juridique fiscal-rural. Le SACEA ne pouvant traiter qu'avec des exploitants agricoles.

Ergolis à été créée pour jouer le rôle de service informatique, sa position lui permet de se détacher du syndicat et s'occupe du développement de logiciels libres. Le président du bureau de la FDSEA est donc président du SACEA, et président d'Ergolis. Dans les faits David Joulin et Brice Texier sont les principaux acteurs d'Ergolis, mais d'un point de vue administratif ne sont pas salariés d'Ergolis.

Service Informatique

Brice Texier, David Joulin

- Conseils et formation à l'utilisation de logiciels libres,
- Conception de sites Internet,
- Installation, formation et accompagnement du logiciel libre de gestion d'entreprises Ekylibre.

➔ **tél. 05 56 00 73 69, fax 05 56 81 66 40, e-mail informatique@fdsea33.fr**

Service Employeurs

Marie-France Chauvet-Raynaud, Christelle Capdeboscq-Degrave et Sylvie Conte-Lallet

- Vente et mises à jour de la convention collective des exploitations agricoles de la Gironde,
- Assistance téléphonique et consultations juridiques en droit du travail,
- Rédaction d'actes juridiques (contrats de travail, avenants, transactions...),
- Procédures de licenciement,
- Défense des employeurs devant le Conseil des Prud'hommes
- Défense des exploitants agricoles devant le Tribunal des Affaires de la Sécurité Sociale,
- Audits sociaux de conformité,
- Aide à la constitution de Groupements d'Employeurs Agricoles (en lien avec le Service de Remplacement Gironde, tél. 05 56 81 49 06),
- Tenue de permanences décentralisées (en partenariat avec certains syndicats viticoles et ADAR).

➔ **Tél. 05 56 00 73 67, fax 05 56 81 66 40, e-mail service.employeurs@fdsea33.fr**

Service Fiscal-Rural

Céline Gentile-Roy et Anne Pélisson

- Droits des sociétés et droit fiscal (aide au choix de la forme sociétaire –GAEC, EARL, SCEA...– création, transformation, assemblées générales –approbation des comptes, changement au niveau de la gérance, augmentation de capital...– et dissolution-liquidation),
- Droit rural (choix du contrat le plus adapté - bail à ferme, à métayage, mise à disposition, prêt gratuit..., conseil et rédaction des contrats, avenants et résiliations, défense devant les Tribunaux Paritaires des Baux Ruraux –en cas de litige bailleurs-preneurs–, conseil en transmission du patrimoine,
- Droit de la vie de l'entreprise (accompagnement dans différentes matières du droit –contrats commerciaux, relations avec les administrations, avec la MSA, aide à la négociation d'échéanciers, dégrèvements fiscaux et sociaux–, dépôt et gestion des marques, réglementation étiquetage).

➔ **Tél. 05 56 00 73 65, fax 05 56 81 66 40, e-mail service.fiscal-rural@fdsea33.fr**

C) Les projets 2014

Soucieux de compléter Ekylibre et rendre le logiciel facilement accessible, Ergolis souhaite ajouter des services, voici ceux prévus pour l'année 2014 :

Pour faciliter l'accès à Ekylibre, Ergolis prévoit le lancement du SaaS (Software as a Service) Ekylibre as a Service pour cette année. C'est à dire qu'Ekylibre sera installé sur un serveur distant, et les clients pourront l'utiliser en ligne. Ce procédé apporte différents avantages, aucune installation n'est nécessaire pour l'utilisateur, il suffit de se connecter avec un identifiant via un smartphone, un ordinateur ou une tablette. Il n'y a pas de risque de perte de données de la part de l'utilisateur. Ergolis s'engage également à assurer la sécurité des données et à permettre à l'utilisateur de mettre à jour sa version lorsqu'il le désire. Un accès gratuit limité en ressources sera proposé, avec la possibilité de quitter à tout moment. Pour la version pro, seul le premier mois ne sera pas remboursable, trois forfais payant sont prévu allant de 29,99€ HT a 49,99€ HT offrant chacun plus d'espace de stockage, et un traitement plus rapide.

Dans le soucis de réduire au maximum la saisie de données des agriculteurs, Ergolis développe une application mobile de tracking nommée Rei. Elle doit permettre d'enregistrer les informations relatives aux actions de l'utilisateur, et de les envoyer sur les serveurs d'Ekylibre. Ce sera le sujet de ce stage, et les fonctionnalités seront détaillées plus en avant.

Pour permettre à l'utilisateur de visualiser les données enregistrées sur les serveurs d'Ekylibre, Ergolis développe une interface publique permettant d'afficher une carte selon des critères que défini l'utilisateur. Tout comme Rei, ce projet à également été débuté lors d'un stage, par Hugo Poilvé.

II. Développement d'une application mobile de tracking

Cette partie présente les détails du sujet de stage et la place qu'il prend dans Ekylibre, ainsi que les outils et méthodes utilisés. Le stage s'est décliné en une première période de prise en mains des outils d'une durée de 2-3 semaines. Puis de trois sprints de 2 semaines chacun. Le temps restant pouvant être consacré à la rédaction du présent rapport.

A) Rei l'application zéro saisie

Sujet

Illustration 1: logo de Rei

Ekylibre permet d'enregistrer les interventions qui ont été effectuées, et d'en déduire des informations. Par exemple si un agriculteur déplace ses animaux, leurs nouvelles positions seront enregistrées, ou s'il sème, les champs concernés seront enregistré et les stocks mis à jour.

Dans un soucis de simplifier cette étape au maximum. Ekylibre souhaite proposer une application mobile de tracking. Lorsque l'utilisateur démarre une intervention (faire un amendement, semer, …) il lance l'application, qui va dès lors récupérer des données géolocalisées, et horodatées régulièrement. Le nom du projet vient du japonais, et signifie zéro. En effet elle à pour objectif de ramener la saisie des informations par l'utilisateur à quelque chose de presque nul.

Afin de permettre à l'utilisateur d'utiliser cette fonction, même s'il n'est pas connecté à internet durant l'intervention, les données sont recueillies en mode asynchrone puis sont envoyées sur les serveurs lorsqu'une connexion est détectée.

Il doit également être possible d'acquérir des QRCode et code barres géolocalisés et horodatés placés a proximité des outils. Ces codes ont pour but d'identifier les outils utilisés, l'utilisateur pourra également ajouter une valeur numérique, et une unité associée à la saisie du code. Ceci permet à la fois d'identifier l'intervention effectuée, mais également de gérer les stocks.

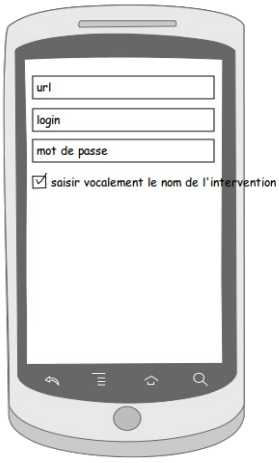
L'utilisateur pourra donc par exemple flasher un QRCode se trouvant à proximité de ses sacs de semence, indiquer la quantité qu'il souhaite utiliser. Puis grâce au GPS, localiser dans quelles parties de son exploitation l'intervention à eu lieu.

Mock-u:

Voici les mock-up ou prototypes qui m'ont été fournis pour réaliser l'application

- L'écran principal permet le lancement d'une intervention. En appuyant sur Play, des points géolocalisés et horodatés sont pris. Un compteur indique la durée de l'intervention. Il est possible de mettre l'intervention en pause. Une liste des interventions réalisées s'affiche. Un bouton permet d'indiquer que l'on passe en mode actif, et un autre de saisir un QRCode

Illustration 2: mock-up intervention en cours



- Cet écran permet à l'utilisateur de s'identifier, et d'enregistrer une URL, celle ci permettra d'envoyer ses données vers son compte Ekylibre. L'identifiant et l'URL[1] ne sont pas directement liés, afin de permettre à une même personne de travailler pour différents comptes. Par exemple si j'utilise Ekylibre pour mon exploitation et que je souhaite travailler avec un autre agriculteur sur son exploitation. Je peux utiliser les mêmes identifiants.

Illustration 3: mock-up identification

- Cet écran permet de flasher un QRCode et code barre, et d'y associer une unité et une valeur. Cette action provoque la création d'un point géolocalisé contenant le code, et les informations rentrées.

[1] u*niform resource locator*

Illustration 4: mock-up saisie QR-Code

B) PhoneGap

1) Utilisation

Afin de permettre à Rei de fonctionner sur le plus grand nombre de plates-formes, l'application a été développée en utilisant Phonegap.

Phonegap est une distribution open source de Cordova d'Apache Software Foundation. Il s'agit d'une plate-forme permettant de construire des applications en utilisant le HTML, CSS et JavaScript. Une fois l'application écrite, Phonegap permet de les déployer sur les principales plates-formes (Andoid, Windows-phone, iOS …) sans perte de fonctionnalités.

Phonegap permet une utilisation en ligne de commande pour créer un projet pour l'application, et d'installer les plateformes. Les SDK des plates-formes doivent cependant être installées sur l'ordinateur. Il est alors possible de lancer un émulateur directement en ligne de commande.

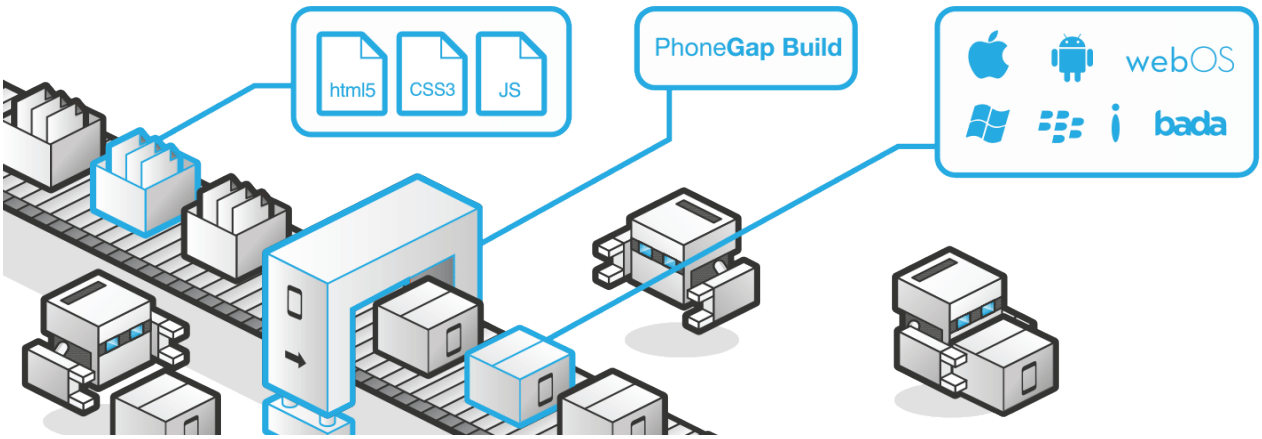
Un projet nouvellement crée contient un Hello-World. Pour le voir il suffira donc d'installer la SDK android sur l'ordinateur puis :

$ phonegap create hello

$ cd hello

$ phonegap run android

Cet exemple aura pour effet de construire l'application, en générant un .apk, le format utilisé par les application Android, puis de le lancer dans un émulateur. L'émulateur doit démarrer puis installer le fichier APK[1]. C'est un procédé long, d'autant plus que l'émulateur est très lent. C'est pourquoi j'ai cherché une solution altérnative.



[1] Android Package

2) Phonegap Build

Conjointement à Phonegap, j'ai utilisé PhoneGap Build, un service basé sur la concept cloud. Ce service intègre les SDK des différentes plates-formes, et propose de construire les applications, puis de nous les retourner dans les format utilisés par les smartphones. Ceci évite d'avoir à installer et mettre à jour les SDK des différentes plates-formes sur son ordinateur. Il permet également d'intégrer des plugins, comme la lecture des QRCode, pour cela il suffit de le préciser dans le fichier config.xml du dossier Phonegap. Les plugins seront rajoutés lors de la construction du projet.

La documentation de Phonegap Build docs.build.phonegap.com permet de bien comprendre comment fonctionne le framework. Ceci mérite d'être précisé car la documentation présente sur le site de phonegap correspond à celle de Cordova. Et même si Phonegap utilise Cordova, ils ne sont pas totalement semblable, notamment pour les actions en ligne de commande, et l'installation des plugins.

Une application Phonegap Build se présente de la façon suivante :

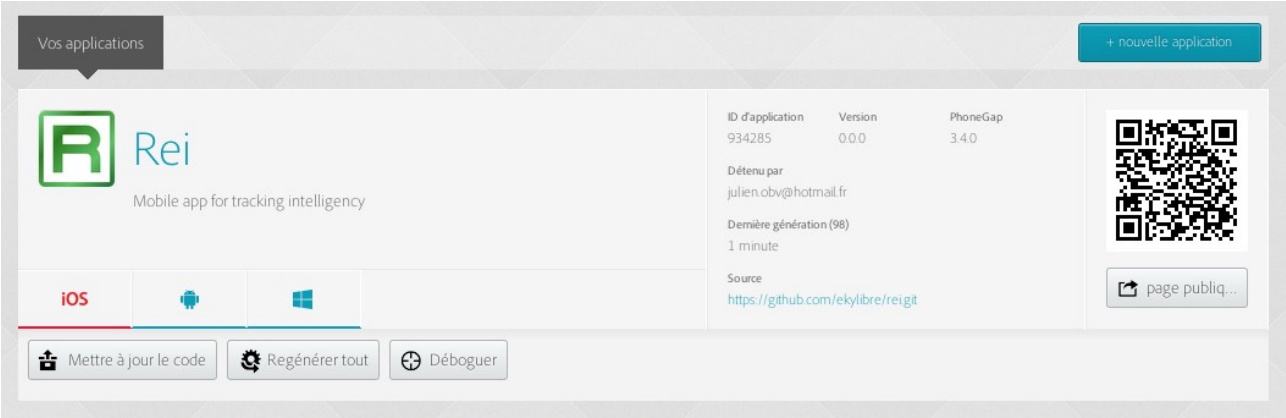


Illustration 5: Rei sur Phonegap Build

- Le titre, le logo et la description sont définies dans le fichier config.xml du projet
- La génération de l'application se fait soit à partir d'un dépôt GitHub, soit en donnant le projet sous la forme d'un fichier zip. Ici Rei se trouve sur le dépôt git d'Ekylibre. Les applications sont générées pour Android et Windows Phone 8. Le compte que j'utilise dans l'image ne possédant pas de licence pour iOS, je ne peut pas générer l'application pour cette plate-forme.

- Un QR code permet d'installer directement l'application sur le smartphone. Il retourne le bon fichier suivant la plate-forme qu'utilise le smartphone. Le QRCode est généré automatiquement lorsque l'application est mise à jour.

C) Les méthodes utilisées

1) Données géolocalisées

PhoneGap propose un objet /géolocation/, qui permet d'accéder aux informations GPS du téléphone. Cette API est basée sur la spécification W3C. Dans le cas où une plate-forme implémente déjà cette spécification, c'est celle ci qui est appelée, elle n'est pas remplacée par celle de PhoneGap. Pour les autres, la documentation nous assure que l'implémentation fournie par PhoneGap devrait être conforme à la spécification du W3C.

La fonction /géolocation.getCurrentPosition()/ prend en argument une fonction qui sera appelée en cas de succès. La fonction possède alors un objet position passé en paramètre. C'est un fonctionnement fréquent en JavaScript, mais qui a pour inconvénient que tous les traitement sur le résultat doivent être fait dans la fonction de succès.

Les données récupérées sont utilisées pour créer un objet point au format JSON, puis le stock dans le téléphone

2) Stockage des données

Les points récupérés doivent être stockés. Le premier objectif du sujet prévoyait que je stock les données soit grâce au /localStorage/, permettant de stocker de donnée persistantes. Il fonctionne comme des cookies et utilise un tableau associatif avec un système clé/valeur. Soit grâce à SQLite le moteur open source de base de donnée.

J'ai choisie en premier lieu d'utiliser /localStorage/, car il me semblait un peu lourd de créer une base de données contenant une seule table. Son API est disponible sur les spécifications W3C. Mais la solution à été abandonnée car trop coûteuse en performance. Un grand nombre de points étant enregistré, et chaque lecture demandant d'effectuer une modification '{objet}' → 'objet' au préalable. C'est à dire parcourir l'objet passé en chaîne de caractères, et supprimer le premier et dernier caractère.

J'ai donc utilisé WebSQL dont l'API est également disponible sur les spécification W3C, Et permettant d'utiliser une base de donnée SQLite. Il s'agit du moteur de base de donnée le plus utilisé dans le monde. Son développement est sponsorisé par NOKIA, ORACLE, Bloomberg, Bentley, Mozilla et Adobe. La base n'est pas séparé de l'application client, et utilise un langage SQL très simple.

J'ai donc créé une table permettant de stocker les informations rattachées à un point géolocalisé (date, latitude, longitude, précision, type, code, quantité, nom de l'intervention), ainsi qu'une fonctionnalité de migration permettant de maintenir la base de donnée à jour.

Illustration 6: logo SQLite

3) Envoi des données :

Une fois les données enregistrées, lorsque la connexion à internet est disponible, et si un nombre important de points à été enregistré (actuellement indéterminé, mais ce devrait être aux environs de 10000), les informations sont envoyées sur un web service d'Ekylibre.

Pour cela l'utilisateur doit se connecter, c'est à dire entrer son identifiant, son mot de passe et URL correspondant à son exploitation. Le web service lui envoi alors un token au format JSON. Il s'agit d'un jeton permettant d'identifier l'utilisateur. Ce principe utilisera le protocole Oauth à therme.

Une fois l'utilisateur enregistré, il peut désormais envoyer ses données sur Ekylibre, soit automatiquement lorsqu'un certain nombre de point à été enregistré, soit en appuyant sur un bouton.

L'application utilise alors AJAX[1] pour envoyer une requête POST. Les données sont envoyées au format JSON.

Illustration 7: logo OAuth

4) La lecture des QRCode :

Afin de lire les QRCode/code barre j'ai utilisé un plugin proposé sur le site de Phonegap Build. En effet, parmi les services proposés, il est possible d'indiquer des plugins qui seront installés lors de la construction de l'application par Phonegap Build.

Cette information doit apparaître dans le fichier config.xml, et le html doit appeler le javascript. Ici barcodeScanner.js .

Le plugin est sous MIT[2] License, une licence open-source, qui permet d'inclure des modifications sous d'autres licences, y compris non libre, et offre un droit illimité d'utilisation, modification, copie,

[1] Asynchronous JavaScript and XML

[2] licence originaire du *Massachusetts Institute of Technology*

publication, distribution et vente. Ce plugin est disponible sur GitHub à l'adresse suivante :

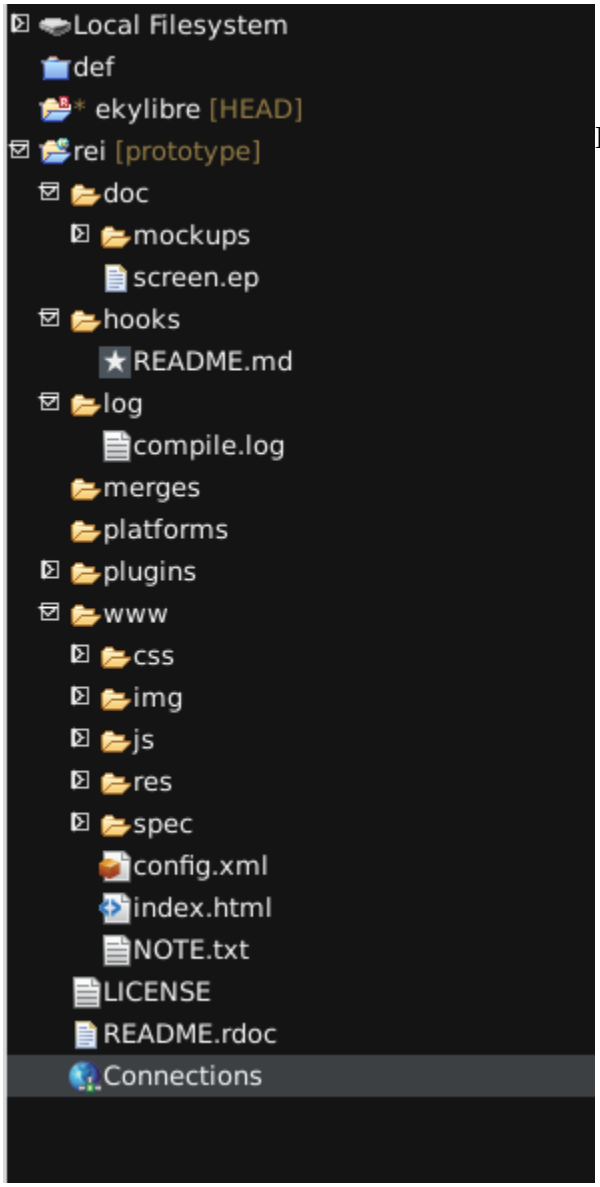
https://github.com/phonegap/phonegap-plugins/tree/master/Android/BarcodeScanner

Il ne permet pas de récupérer les mêmes types de code suivant les plates-formes sur lesquelles il est utilisé. Mais toutes fonctionnent avec les QRCodes.

III. Environnement de travail

A) Framework et IDE [1]

Rei étant réalisée suivant le framework Phonegap, voici a quoi elle ressemble :



Les principaux dossiers d'une application PhoneGap :

- doc : Il contient les documents annexe, par exemple les mock-ups illustrant ce que doit donner l'application visuellement.
- platforms : Ce sont les fichiers permettant de construire l'application sur différentes plates-formes (iOS, Android, Windows phone …) ici il est vide car l'application est construite par PhoneGap Build, qui s'occupe de fournir les fichiers nécessaires.
- www : Il s'agit du dossier de travail, c'est lui qui contient les fichier HTML, Javascript et CSS.

  Au lancement de l'application, c'est le fichier index.html qui est lu. Le fichier config.xml permet de régler certaines informations, comme la version de PhoneGap, le nom et les icônes de l'application, les permissions, les plugins, ...

[1] *Integrated Development Environment*

1) Les bibliothèques

Jquery Mobile

Les applications PhoneGap étant réalisées en HTML5 et JavaScript, j'ai choisi d'utiliser JQuery mobile. Il s'agit d'un framework fonctionnant avec JQuery (un framework JavaScript) et HTML5. Il permet d'obtenir une interface fonctionnant sur toutes les plates-formes mobiles, et d'utiliser JQuery, qui offre des fonctionnalités très pratiques dans le développement JavaScript.

Il propose d'utiliser des « data-role » aux élements HTML. Ici l'utilisation du rôle « page » permet de créer de séparer le code HTML en différentes pages, et de choisir laquelle est affichée.

```
<div data-role="page" id="interventions" data-theme="a">
    contenu de la page
</div>
```

J'ai donc crée 3 pages :

- /home/, la page d'accueil,
- /intervention/, la page principale, c'est ici qu'on lance les interventions
- /configuration/, elle permet de s'identifier, et de régler le type de saisie pour le nom des interventions

Ce système permet de naviguer entre les pages, tous en laissant le javascript tourner en fond. Évitant que l'enregistrement de s’arrête lorsque l'on change de page.

Jquery mobile permet également d'utiliser le système de navigation d'AJAX pour apporter des transitions animées, ainsi que des widgets tels que des formulaires, des icônes, ou encore un système de pages, qui permet de ne pas changer de fichier pour gérer les différentes pages de l'application.

2) Plugins

Jquery countdown

Pour compteur indiquant le temps écoulé depuis le début de l'intervention en cours, j'ai utilisé JQuery countdown.

JQuery countdown est disponible sur le site suivant : http://keith-wood.name/countdown.html

Comme /BarcodeScanner/, ce plugin est sous licence MIT, ne posant donc aucun soucis d'utilisation et de modification.

Illustration 8: mock-up intervention en cour

Il permet d'utiliser un compteur javascript. On place donc une balise dans le code html :

```
<div id="clock" ></div>
```

puis dans le javascript en utilisant Jquery pour accéder à l'élément clock, on instancie une horloge en lui donnant des options au format JSON :

```
$(clock).countdown({
    since : new Date(),
    format : 'HMS',
    description :'durée de l'intervention en cour',
    compact : true
}) ;
```

Ce code permet d'afficher une horloge en fixant la date de départ à maintenant (c'est à dire qu'elle part de 0), au format HH:MM:SS. L'option compact indique que les caractères sont collés les uns aux autres, sinon ils sont séparés par des espaces

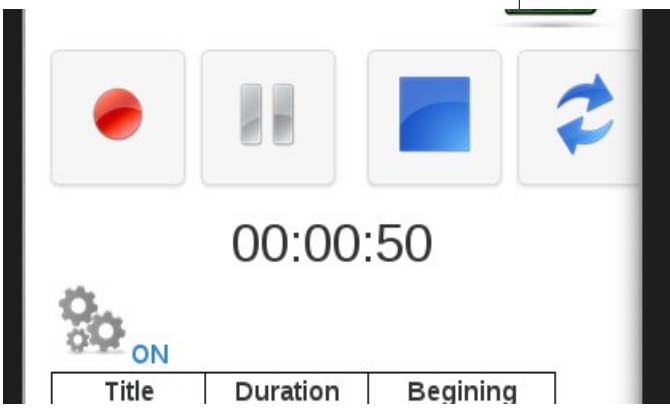


Illustration 9: Résultat obtenu

BarcodeScanner

BarcodeScanner est un plugin disponible directement via PhonegapBuild. C'est à dire qu'il n'est pas nécessaire de le télécharger et de l'inclure au projet Phonegap. Il suffit d'ajouter la ligne correspondante dans le fichier config.xml, et Phonegap build ajoutera lui même les fichiers, en utilisant la version correspondant à chacune des plates-formes.

En ajoutant la ligne suivant :

```
<gap:plugin name="com.phonegap.plugins.barcodescanner" />
```

Un récapitulatifs des plugins installés de cette manière est disponible sur le site de Phonegap Build. Il est également possible d'installer les plugins en ligne de commandes si on utilise pas Phonegap Build. Dans ce cas il existe plugman, un géstionnaire de plugin pour cordova.

Il est possible d'appeler un objet `BarcodeScanner` comme ceci :

```
var scanner = cordova.require("cordova/plugin/BarcodeScanner") ;
```

On appelle ensuite la fonction scan, qui va lancé le lecteur de QRCode sur le smartphone.

```
scanner.scan(functionSuccess,functionError) ;
```

Suivant si la lecture à réussie sera appelé la fonction *functionSuccess* avec un objet JSON *result* en paramètre, ou *functionError* avec en paramètre un objet *error.*

3) Les outils de développement

GitHub:

GitHub est un service d'hébergement et de géstion de développement de logiciel. Il permet d'héberger des projets avec git, le logiciel libre de gestion de version crée par Linus Torvalds. Github offre des fonctionnalités que l'on retrouve généralement sur les réseaux sociaux, comme les flux et la possibilité de suivre des projets et personnes.

Pour travailler sur Rei, j'ai rejoin le projet Ekylibre sur GitHub. Un dépôt Rei à été créé pour l'application. Les /commit/ sur ce dépôt devaient être réguliers et rédigés en anglais.

GitHub à également été utilisé pour récupérer du code issu d'autres logiciels libre, comme pour BarcodeScanner et JqueryCountdown.

Illustration 10: octocat la mascotte de git

Aptana :

Pour écrire le code de l'application j'ai utilisé l'IDE Aptana. Aptana est open-source, et très utile pour le développement de site web. Il fournit des aides de saisies pour HTML, CSS et JavaScript. De plus Aptana est issu d'Eclipse, IDE que j'ai déjà utilisé au cour de ma formation à l'IUT, il m'a donc été simple de le prendre en main.

Aptana n'est plus libre depuis fin 2007 puisqu'il est passé de licence GPL à une double licence APL(Aptana Public Licence)/GPL provoquant la déception du monde libre. Malgré cette défection, il reste très pratique dans mon cas. Phonegap se basant exclusivement sur du HTML/CSS/JS.

Illustration 11: logo d'Aptana

Ripple

Bien que Phonegap Build permette de construire les application facilement, il reste lent de l'utiliser pour chaque test. Ainsi pour accélérer les choses, j'ai choisi d'utiliser le plugin de Chrome : Chrome-Ripple disponible sur le web store de Google Chrome.

Il permet d'émuler le rendu du HTML/CSS/JS sur un smartphone. En ayant l'avantage de ne pas demander de créer un fichier APK grâce à Phonegap Build, et de ne pas avoir a faire booter un émulateur. Il permet de plus, de régler des paramètres de géolocalisation et de faire des tests sous différents téléphones.

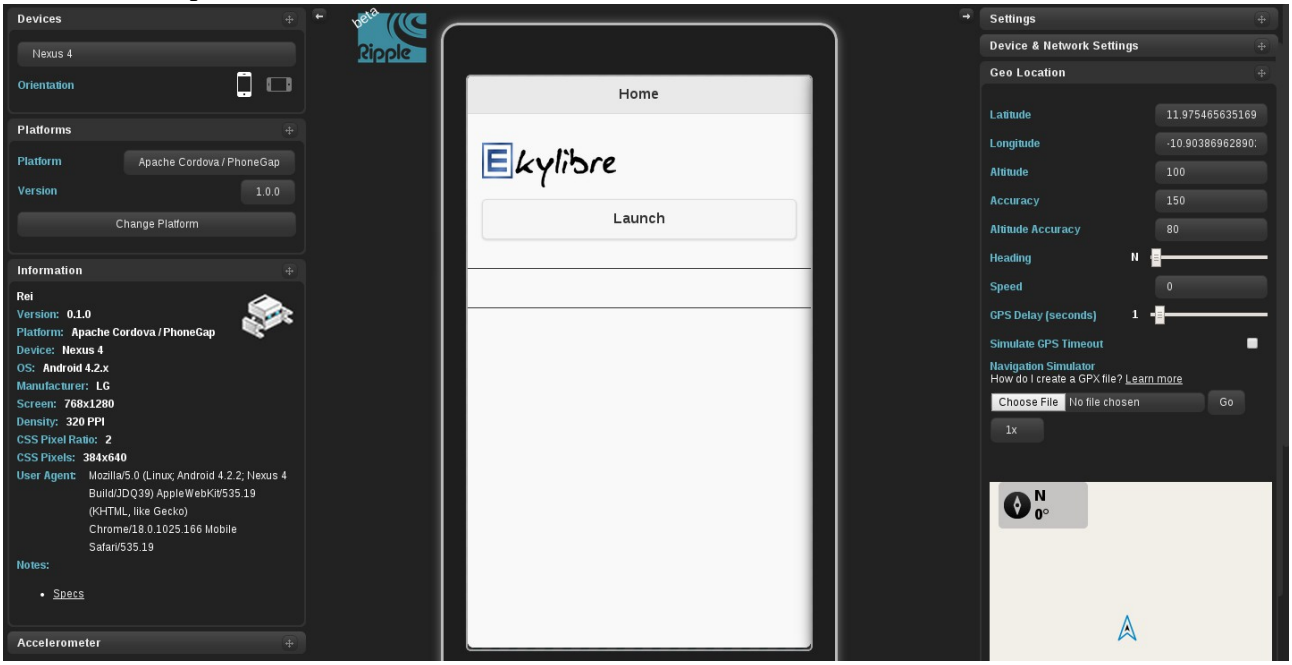


Illustration 12: Rei lancé avec Ripple

Ripple est un outil intéressant pour le début du développement et permet d'avoir un aperçu rapidement sur la mise en forme. Il fonctionne avec WebSQL et localStorage, j'ai donc pu développer les fonctionnalité de stockage des points géolocalisés dessus.

Cependant je n'avais pas de véritable smartphone pour réaliser les tests. Or Ripple ne fonctionne pas correctement avec la version 3,4 de PhoneGap. Il a par exemple été impossible de faire fonctionner des requêtes POST avec AJAX. Il n'était pas possible de faire fonctionner un plugin de lecture des QR code, et la gestion des permissions pour la géolocalisation ne fonctionnait pas comme sur un véritable smartphone.

Wiko

Afin de tester l'application, un smartphone a été mis à ma disposition à partir du 26 mai. Il s'agit d'un Wiko Rainbow, une marque française, avec la configuration suivante : processeur quad core 1,3GH, 4GB ROM, 1GB RAM, écran 5 pouces HD.

L'utilisation d'un smartphone a permis de ne pas avoir a passer par Ripple, qui ne convient pas a Phonegap 3.4, ni un émulateur, long à démarrer, et de tester le rendu dans les conditions d'utilisation. Un cycle de travail se déroule donc de la façon suivante :

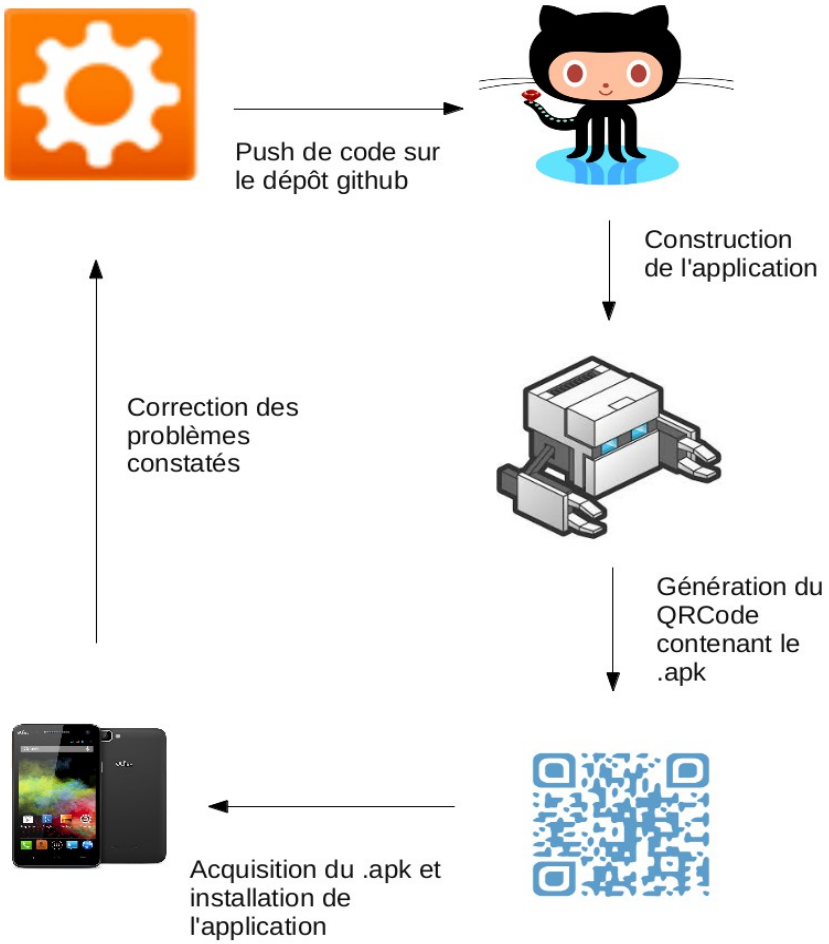


Illustration 13: Cycle de développement

B) Sprints et gestion de projet

Méthode SCRUM

Le projet à été réalisé en appliquant la méthode SCRUM. Comme outil nous avons utilisé Trello, une application permettant de définir des tâches à faire, en cours et terminée. Cet outil permet à tous les membre de mettre à jour l'état des taches, de plus il évite d'utiliser des post-it pour représenter les objectifs.

Les Sprints prévus à l'origine :

Sprint 1 :

Réaliser une application mobile avec acquisition du signal GPS, et stocker des valeurs via localStore ou WebSQL en fonction des possibilités. Les données sont au format :

1 read_at TIMESTAMP

2 latitude INT

3 longitude INT

4 type STRING (point, QRCode, start, finish ...)

5 accuracy DECIMAL

6 code STRING (correspond au QRCode enregistré si il y en a un)

7 unit STRING (population, litre, kg, m …)

Sprint 2 :

Intégrer le workflow avec les différents modes, et la synchronisation des données.

Sprint 3 :

Intégrer la lecture de QRCodes (et codes barres) dans l'appli et la synchronisation

Des feed-back réguliers nous on fait modifier les objectifs des sprints voici les principaux changements :

- une colonne intervention_name a été ajoutée. Elle n'est remplie que lors de l'acquisition d''un point de type « start », et permet de renseigner sur le contenu de l'intervention. Une option a été ajoutée dans la menu pour choisir si le nom est généré automatiquement, ou si c'est l'utilisateur qui écrit le nom. Sur ce point il a été choisi de ne pas ajouter de plugins pour permettre la saisi vocale. La plupart des smartphones intégrant déjà cette fonctionnalité.
- Un système d'authentification utilisant OAuth à également été ajouté aux fonctionnalités à implémenter.
- Il a été choisi d'utiliser SQLite pour le stockage des données, car localStorage demande trop d'opérations en lecture, et l'application peut être amenée à stocker de très nombreux points. LocalStorage est toutefois utilisé pour stocker l'identifiant et le mot de passe de l'utilisateur, et les options de saisie s'il est connecté.

- Un système de migration à été ajouté pour maintenir la base WebSQL à jour. Les migrations SQL à effectuer pour passer d'une version à l'autre sont enregistrées. Au démarrage de l'application, s'il existe des requêtes pour passer à une nouvelle version elles sont effectuées.
- Des boutons pour naviguer entre le menu, et l'écran des interventions ont été ajoutés. Il existe des boutons /menu/ et *back* sur les smartphone. Mais il a été décidé de rajouter des boutons de navigation dans le cas où un smartphone ne posséderait pas ces boutons.

C) Présentation du travail réalisé

1) Fonctionnalités et rendu de l'application

Pour rendre compte du travail effectué, voici des copie d'écran commentées montrant le rendu de l'application. Puis une attention particulière sera portée sur l'acquisition et le stockage des données géolocalisées.

L'application n'a volontairement que peu de texte, afin de faciliter la traduction.

Illustration 14: écran home

L'écran Home

Il affiche le logo d'Ekylibre. Le logo de Rei apparaît pendant le chargement de l'application. La base de donnée est chargée, et les autres pages sont chargées.

Appuyer sur *launch* lance l'application.

L'écran Interventions

Cet écran s'occupe de l'affichage des interventions déjà réalisées.

Il permet de lancer une nouvelle intervention en appuyant sur le bouton d'enregistrement (rond rouge).

Le bouton de synchronisation permet d'envoyer les données sur le serveur Ekylibre. C'est un bouton temporaire, l'envoi des données devant se faire automatiquement par la suite.

La flèche verte permet d'aller sur l'écran de menu, le bouton menu du smartphone remplit la même fonction. La flèche est présente par sécurité, dans le cas où un smartphone ne posséderait pas de bouton menu, ou ne soit plus désigné de la même façon.

Le bouton avec un QRCode lance le lecteur de QRCode

Le bouton *login* en haut à droite envoie vers le menu pour que l'utilisateur s'authentifie, s'il est déjà authentifié, c'est un bouton *logout*, qui déconnecte l'utilisateur.

Le bouton avec les rouages dans l'écran de l'intervention lancée est un bouton mode actif. Il permet d'indiquer les moments où l'intervention est réellement entrain de se dérouler. Si l'agriculteur doit s’arrêter momentanément dans son travail, et que l'intervention n'est pas terminée, il arrête le mode actif.

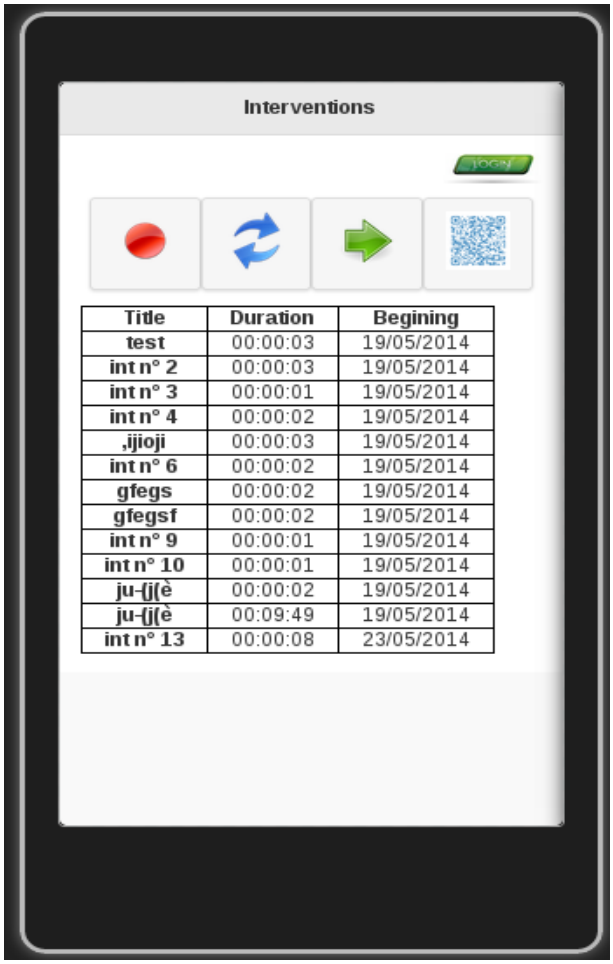


Illustration 15: écran intervention arret

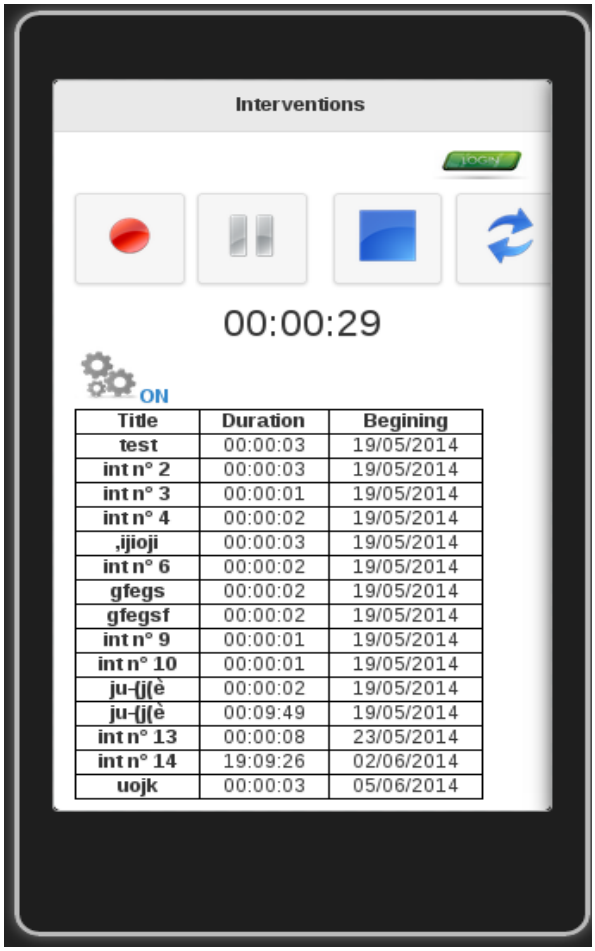


Illustration 16: écran intervention lancée

L'écran de menu :

La flèche verte permet de retourner sur l'écran d'intervention, le bouton *back* du téléphone permet la mêle chose.

L'utilisateur peut s'authentifier, et rentrer une URL qui correspond à son exploitation sur Ekylibre. Si l'utilisateur est déjà authentifié, le formulaire est remplacé par un bouton de déconnexion.

Il est possible de choisir si l'utilisateur doit entrer un titre pour ses intervention grâce à la section « intervention name » . Le choix est alors enregistré dans le *localStorage*.

Illustration 17: écran menu

2) Focus sur l'enregistrement et l'affichage des interventions.

Pour lancer une intervention l'utilisateur appui sur le bouton d'enregistrement. Si le nom n'est pas choisi automatiquement, un prompt s'ouvre pour le saisir.

Pour récupérer l'événement, j'utilise JQuery, qui permet d'utiliser l'attribut *id* d'un élément HTML, et la fonction *onClick()*. Il suffit ensuite de définir le listener, la fonction *onClick*.

Le bouton est défini par la balise html suivante :

```
<div id="buttonStart">
<a data-role="button" href="#"> <img src="img/Record-Normal-Red-icon.png"
alt="start" width="50px" height="50px" /> </a> </div>
```

La balise *div* possède l'attribut *id* qui permet à JQuery d'identifier l'élément. On peut voir l'attribut *data-role= 'button*' dans la balise /<a>/, il s'agit d'un attribut pour JQuery mobile, JQuery mobile propose des widgets utilisable très simplement en définissant un data-role.

La fonction JavaScript associée au bouton

```
$("#buttonStart").click(function() {
   switch (window.localStorage.getItem("saisie")) {
         case "Write" :
               window.location.href = "#nameChoice";
               break;
         case "Auto" :
               $('#clock').countdown('destroy');
               createClock('#clock');
               //affichage de boutons
               $("#buttonFinish").show();
               $("#buttonPause").show();
               $("#buttonActiveOn").show();
               $("#buttonActiveOff").hide();
               app.startIntervention();
               $('#clockActiveMode').countdown('destroy');
               break;
         default :
```

```
                $('#clock').countdown('destroy');
                createClock('#clock');
                //affichage de boutons
                $("#buttonFinish").show();
                $("#buttonPause").show();
                $("#buttonActiveOn").show();
                $("#buttonActiveOff").hide();
                app.startIntervention();
                $('#clockActiveMode').countdown('destroy');
                break;
    }
})
```

Pour enregistrer la façon dont le nom de l'intervention est défini, j'utilise le *localStorage*. Cela fonctionne comme un tableau associatif, le nom de la clé est 'saisie', et les valeurs possibles sont 'Write', et 'Auto'. Une option vocale était prévue, mais Phonegap ne proposait pas de plugin, et ceux que j'ai trouvé ne fonctionnait que pour une seule plate-forme. Il a donc été décidé que pour enregistrer en vocal, l'utilisateur choisirait 'Write', et utiliserait la fonctionnalité native de son téléphone, s'il en dispose.

La fonction *window.location.href* envoie vers un popup JQuery mobile. Ici c'est un prompt permettant de saisir le nom de l'intervention. Un fois fait, les même fonctions que dans le cas d'un nom automatique sont appelées, mais avec le nom de l'intervention en paramètre.

```
createClock('#clock');   //   Lance   l'horloge   indiquant   la   durée   de
l'intervention en cours
$("#buttonFinish").show(); //Indique quels boutons sont visibles ou cachés
$("#buttonPause").show();
$("#buttonActiveOn").show();
$("#buttonActiveOff").hide();
app.startIntervention(int_name); // On lance l'intervention
```

La fonction startIntervention va créer un premier point avec pour attribut sa position, la précision, la date, un type start, et le nom de l'intervention s'il est défini.

Puis la fonction lance la prise de points géolocalisés à intervalle régulier. Pour cela j'utilise la fonction javascript *setInterval(location, REFRESH_TIME);*

La fonction *location()* sera appelée suivant un intervalle de temps défini par *REFRESH_TIME*, ici

toutes les secondes.

Afin de récupérer la latitude, longitude et précision, j'utilise la fonction :

```
navigator.geolocation.getCurrentPosition(successLocation,          errorLoc,
optionsLoc);
```

Cette fonction appellera de manière asynchrone *successLocation* avec un objet *pos* en paramètres, ou *errorLoc* en cas d'erreur avec un objet *error.* La fonction de succès lance la création et le stockage du point avec la fonction *createLocPoint(crd, typePoint, name).*

Les points sont crées à partir de *crd* obtenu avec *pos.crd* de *succesloc*.

```
function createLocPoint(crd, typePoint, name) {
          var point = {
                latitude : crd.latitude,
                longitude : crd.longitude,
                accuracy : crd.accuracy, //accuracy in meters
                date : new Date(),
                type : typePoint,
                name : name
          };
          if (typePoint == 'end' || typePoint == 'start')
                db.storePoint(point, db.writeInterventions);
          else
                db.storePoint(point);
    }
```

L'objet *db* correspond à la base de donnée. La date est obtenue grâce à l'objet javascript Date, qui par défaut correspond à la date actuelle.

La méthode *storePoint* est ensuite appelée pour enregistrer le point. S'il s'agit d'un point 'start' ou 'end' cela signifie qu'il faut actualiser le tableau contenant le récapitulatifs des interventions, et la fonction *writeInterventions* est passée en paramètre.

La base de donnée correspond à une base WebSQL déclarée de cette façon :

```
db = openDatabase("Rey_Database", "", "database", 65536);
          ////////Migration
```

```
        var M = new Migrator(db);
        M.migration(1, function(t) {
            t.executeSql("CREATE TABLE IF NOT EXISTS points (id INTEGER PRIMARY KEY AUTOINCREMENT,name, latitude FLOAT, longitude FLOAT , date DATE, accuracy NUMERIC, type TEXT, code TEXT, quantity NUMERIC, unit TEXT)");
        });
    M.doIt();
```

openDatabase ouvre la base *Rey_Database*, si elle n'existe pas elle est créée.

L'objet *Migrator* permet de mettre à jour la base de données. Il enregistre les modifications à faire dans la base de donnée grâce à sa méthode *migration(numéro_version, fonction)*. Puis la fonction *doIt()* le fait exécuter toutes les mises a jour.

Ici, si une nouvelle base est créée, elle sera de version 0, et donc fera la mise à jour pour la version 1, et créera une table /points/. Si la base existait déjà, et qu'elle est de version 1, aucune mise a jour ne sera faite. Ce système permet de facilement maintenir à jour les bases de données des utilisateurs

Le stockage des points se fait ensuite simplement avec la fonction *storePoint*.

```
this.storePoint = function(point, write) {
        if (point != undefined) {
            var query = "INSERT INTO points (name,latitude,longitude,date,accuracy,type,code,quantity,unit) VALUES (?,?,?,?,?,?,?,?,?);";
            db.transaction(function(tx) {
                tx.executeSql(query, [point.name, point.latitude, point.longitude, point.date, point.accuracy, point.type, point.code, point.quantity, point.unit], function(tx, result) {
                    console.log("Query Success");
                });
                if (point.type == 'end' || point.type == 'start')
                    write();
            }, function(error) {
                console.log("Transaction Error: " + error.message);
            }, function() {
                console.log("Transaction Success : ajout Point");
            });
        }
```

```
    };
```

La fonction /executeSql/ exécute une requête d’insertion, en utilisant les attributs de l'objet point. La fonction de succès lance l’exécution de la fonction de mise à jour de l'affichage, la fonction d'erreur permet d'afficher le contenu des erreurs SQL.

CONCLUSION

Pendant ce stage, j'ai réalisé une application mobile libre Rei pour Ekylibre, l'1er ERP libre du monde agricole. Le développement de l'application va se poursuivre, mais elle à déjà pu être présentée à la foire de Bordeaux et au SIAD[1] à Agen.

Grâce à ce travail, j'ai découvert une mise en pratique des méthodes agiles avec SCRUM, et son fonctionnement avec un nombre réduit d'acteurs. J'ai ainsi pu avoir des retours très réguliers sur l'analyse du projet que ce soit d'un point de vue informatique ou métier, permettant grâce à de simples explications de lever rapidement les ambiguïtés .

De plus Ekylibre m'a permis de familiariser avec le monde libre, et de découvrir concrètement les possibilités d'évolution qu'apporte ces licences. C'est à dire l'idée de partage qui outre l'aspect idéologique de permettre à d'autre d'utiliser son travail, nous permet d'intégrer des projets libres aux notres, apportant un gain en rapidité considérable.

Enfin ce stage m'a conforté dans l'idée de poursuivre mes études, car travailler dans un projet libre tel qu'Ekylibre dans un cadre comme celui ci, avec peu de personnes, mais une communication permanente et efficace me semble être le plus proche de ce que je souhaite. Si je pense que le DUT est une bonne base, je souhaite tout de même ouvrir d'avantage mes possibilités afin d'avoir une plus grande liberté dans mes choix professionnels.

[1] Semaine des Initiatives pour l'Agriculture de Demain

Annexes

Biblio

W3C

Web SQL

http://www.w3.org/TR/webdatabase/

HTML 5

http://www.w3.org/TR/html5/

ECMAScript (JavaScript)

http://www.w3.org/TR/DOM-Level-2-Core/ecma-script-binding.html

LocalStorage

http://www.w3.org/TR/webstorage/

,

PhoneGap

http://phonegap.com/

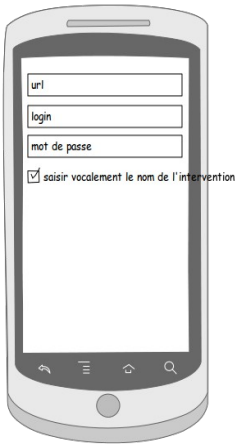
http://docs.phonegap.com/en/3.5.0/index.html

PhoneGap Build

https://build.phonegap.com/

Images

Mock-up



Code Source

Migrateur

```
function Migrator(db) {
	var migrations = [];
	this.migration = function(number, func) {
		migrations[number] = func;
	};
	var doMigration = function(number) {
		if (migrations[number]) {
			db.changeVersion(db.version, String(number), function(t) {
				migrations[number](t);
			}, function(err) {
				if (console.error)
					console.error("Error!: %o", err);
			}, function() {
				doMigration(number + 1);
			});
		}
	};

	this.doIt = function() {
		var initialVersion = parseInt(db.version) || 0;
		try {
			doMigration(initialVersion + 1);
		} catch(e) {
			if (console.error)
				console.error(e);
		}
	};
}
```