Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Travail de bachelor 2010

Filière Informatique de gestion

Développement d'un client iPhone pour le calcul du bilan énergétique des produits de grande consommation

Etudiant : Yann Métrailler

Professeur : Jean Hennebert

www.hevs.ch

Table des matières

1 Introduction

1.1 Projet Green-T

Depuis 2008, le projet RCSO Green-T (Ra&D) est mené par l'Institut Informatique de Gestion de la HES-SO // Valais à Sierre. Comme indiqué dans sa description[1], ce projet a pour but de répondre à deux questions :

- *« Est-il possible, avec les performances actuelles des systèmes d'information, d'étendre les analyses écologiques à des produits de grande consommation, typiquement des produits alimentaires ? »*
- *« Peut-on fournir ces informations au consommateur sur son lieu d'achat, à travers son téléphone mobile, au moment où il doit faire un choix parmi plusieurs produits ? »*

Actuellement, la conscience écologique de la population s'accroit face aux problèmes tels que la pollution et le réchauffement climatique qui prennent de l'ampleur. C'est pourquoi Green-T a un enjeu social important dans la mesure où son but est de guider le consommateur vers un comportement plus respectueux de l'environnement en lui permettant de prendre des décisions réfléchies lors de ses achats.

L'intérêt est aussi conséquent pour les producteurs, qui sont de plus en plus montrés du doigt dès lors que la fabrication et la distribution de leurs produits a un mauvais impact écologique. Ainsi, en adoptant une démarche « éco-responsable », ceux-ci seront mis en valeur par une évaluation favorable du bilan écologique.

De plus, grâce à l'évolution technologique actuelle, que ce soit au niveau des appareils mobiles (Smartphones), des méthodes d'identification des objets (code-barres, reconnaissance par image), ou encore des technologies web, nous pouvons imaginer des solutions à la fois conviviales, ergonomiques et performantes qui nous permettent de répondre positivement à ces deux questions.

Ce travail de bachelor s'inscrit dans le cadre du projet Green-T, puisqu'il doit permettre de démontrer la réalisation d'une telle solution.

Image 1 – Concept général du projet Green-T

[1] http://iig.hevs.ch/valais/green-t.html, Green-T - 2008-2010 - RCSO

1.2 Objectifs

Le but du projet est principalement le développement d'un client natif iPhone proposant au consommateur le bilan écologique des produits qu'il achète. Toutefois, il couvre aussi les aspects d'intégration et d'adaptation de l'architecture existante, mise en place en amont. Par conséquent, une bonne compréhension des fonctionnalités du serveur est à acquérir.

Le cahier des charges annexé définit de manière détaillée les fonctionnalités que l'application doit proposer. Il contient les fonctionnalités « de base », qui sont le cœur de l'application et sur lesquelles repose tout son intérêt :

- Géolocalisation (modes GPS et saisie manuelle)
- Identification du produit (modes scannage du code-barres et saisie manuelle)
- Consommation du web service
- Affichage du bilan environnemental pour un produit
- Affichage du bilan environnemental de chaque produit pour une même famille
- Historique des recherches

Des fonctionnalités optionnelles ont en outre été proposées, et doivent tirer parti des capacités de l'iPhone pour apporter une plus-value en matière de convivialité :

- Identification du produit par reconnaissance d'image
- Affichage des détails d'un produit et d'un producteur
- Affichage du parcours producteur → consommateur sur une carte pour un produit donné
- Partage d'une analyse avec un contact

Durant le temps imparti de 360 heures, il s'agit donc pour résumer de :

- Se familiariser avec l'architecture existante de Green-T
- S'initier au développement iPhone en Objective-C et sa plateforme Xcode
- Réaliser un client iPhone fonctionnel
- Elaborer un planning et effectuer le suivi du projet
- Rédiger ce rapport de projet

2 Environnement

Dans ce chapitre, nous allons aborder tout ce qui concerne l'environnement de travail nécessaire pour se lancer pleinement dans le développement pour iPhone. Il est en grande partie basé sur des morceaux choisis de l'excellent livre « Beginning iPhone Development : Exploring the SDK »[2].

2.1 Prérequis

Avant de pouvoir commencer à écrire des applications pour iPhone, il y a quelques prérequis à remplir. Je vous propose un rapide tour d'horizon de ceux-ci.

2.1.1 Matériel

Tout d'abord, le développement iPhone n'est possible actuellement qu'à partir d'une plateforme Mac. Cela veut dire qu'il faut disposer d'un Macintosh Intel avec le système d'exploitation Leopard (OS X 10.5.3 au minimum). Tous les ordinateurs fabriqués par Apple depuis 2006 devraient convenir parfaitement.

2.1.2 iPhone Dev Center

Nous allons aussi devoir nous inscrire en tant qu' « Apple Developer ». En effet, cette étape est indispensable pour avoir accès au SDK et son environnement de développement Xcode, sans oublier la documentation, des vidéos et des exemples de code.

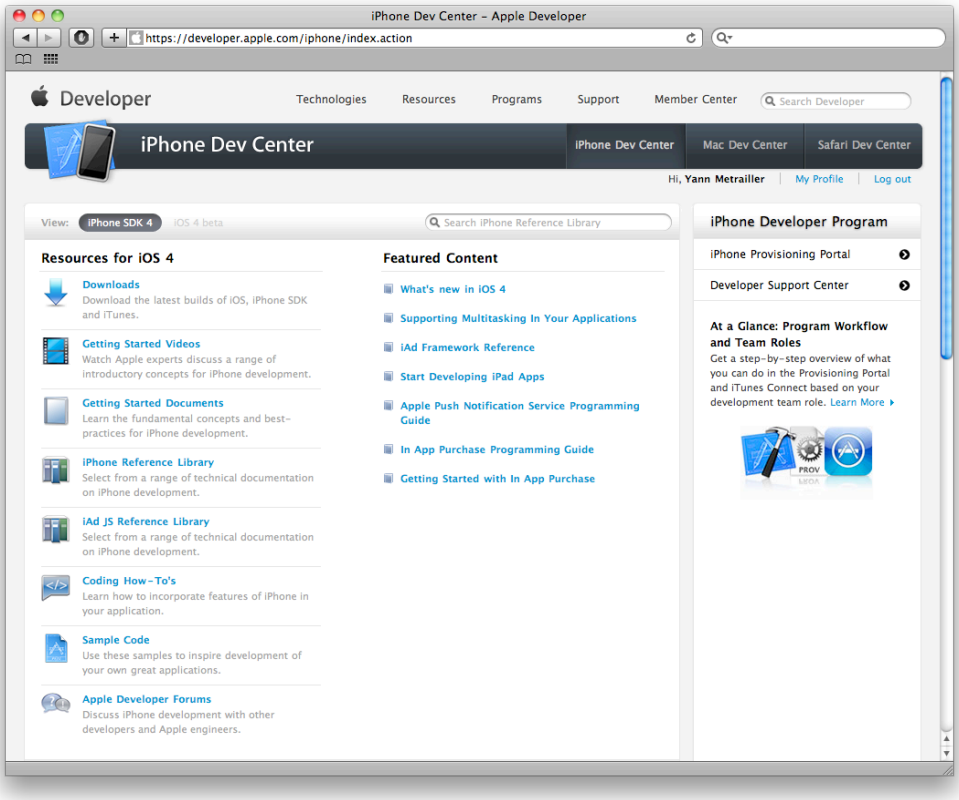


Image 2 - iPhone Dev Center

[2] MARK D. et LAMARCHE J., *Beginning iPhone Development : Exploring the iPhone SDK*, Apress, 2009

2.1.3 iPhone Developer Program

Une fois inscrit en tant qu' « Apple Developer », nous avons donc à notre disposition tous les outils nécessaires. Le kit de développement nous permet effectivement d'écrire des applications, et de les tester dans le simulateur inclus. A noter que ce simulateur ne supporte pas les fonctionnalités hardware de l'iPhone (accéléromètre, appareil photo).

Toutefois, il n'est pas encore possible d'installer l'application sur notre appareil. En effet, cela nécessite d'abord de faire partie du « iPhone Developer Program ». Ce programme est payant et offre trois options qui permettent d'installer l'application et de la distribuer sur l'App Store.

- **Programme standard**, $99/année
  - Pour un développeur individuel ou une équipe de développeurs qui souhaitent créer des applications libres ou commerciales
- **Programme entreprise**, $299/année
  - Pour les développeurs qui souhaitent créer des applications propriétaires
- **Programme éducation**, gratuit
  - Pour les institutions d'éducation supérieure qui souhaitent introduire le développement iOS dans leur cursus. Toutefois, ce programme ne permet pas de publier l'application sur l'App Store.

Pour la réalisation du projet Green-T, j'ai été intégré à l'équipe de la HES-SO // Valais, qui dispose d'un programme éducation.

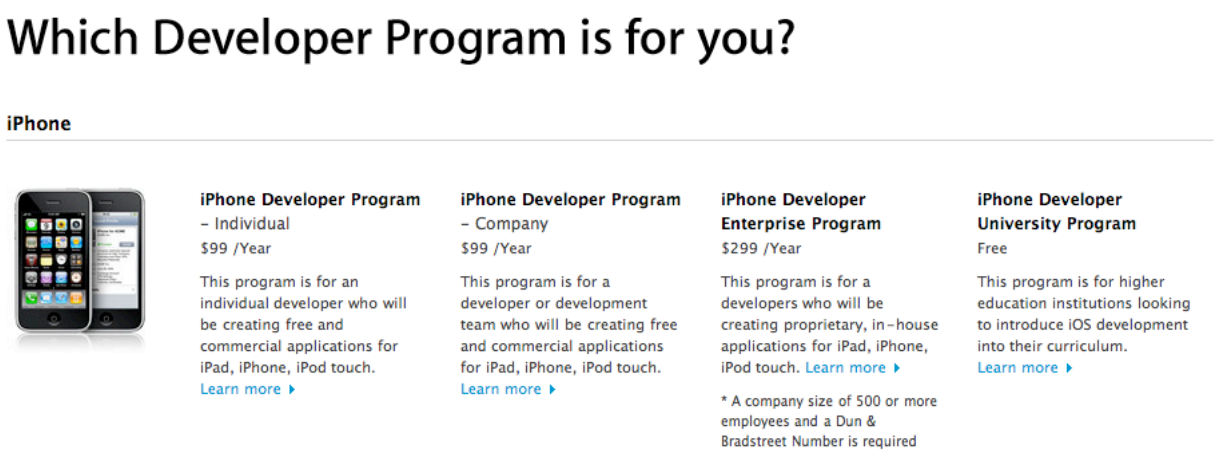


Image 3 - iPhone Developer Program[3]

2.2 iPhone SDK

Le SDK de l'iPhone contient, comme indiqué précédemment l'IDE Xcode, le simulateur, et un ensemble d'outils permettant le développement d'applications pour iPhone, iPad et iPod Touch, sans oublier les API qui sont les mêmes que celles utilisées par Apple.

[3] Source : http://developer.apple.com/programs/which-program, juillet 2010

Celui-ci a été lancé officiellement par Apple le 6 mars 2008, mais il aura fallu attendre jusqu'au 11 juillet de la même année, et une mise à jour du firmware, pour que les développeurs puissent distribuer leurs applications.

La version 4 du SDK vient tout juste d'être proposée par Apple en même temps que l'iPhone 4 et le nouvel iOS4, et offre bien entendu une quantité de nouveautés, dont la plus marquant est l'arrivée de multitâche. Néanmoins, c'est la version précédente (SDK 3.2) que nous avons utilisé pour Green-T car c'était la version courante au début du projet et que, pour des raisons évoquées plus loin, nous avons conservé pour la fin du projet.

2.3 Outils

Nous allons pas ici détailler précisément les outils du SDK, mais simplement en faire une présentation globale afin d'avoir une vue complète de l'environnement de travail. Ci-dessous les logos[4] des applications que nous allons aborder.

Image 4 - Logo Xcode

Image 5 - Logo Interface Builder

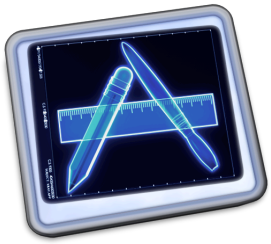


Image 6 - Logo Instruments

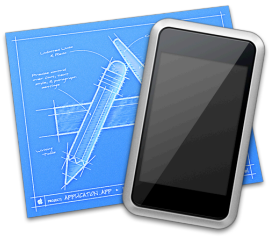


Image 7 - Logo iPhone Simulator

2.3.1 Xcode IDE

Xcode est donc l'environnement de développement par défaut sur Mac. Il intègre les frameworks Cocoa (Mac) et Cocoa Touch (iPhone, iPod Touch, iPad), et supporte une quantité de langages, bien que le principal soit l'Objective-C. Il est fourni en standard avec chaque Mac, dans les disques d'installation d'OS X.

En plus de l'édition du code source, Xcode gère bien entendu sa compilation et son débogage, mais se charge en outre de la gestion des appareils de test, des certificats associés, de l'installation des applications sur ces appareils, et j'en passe.

Nous avons travaillé avec la version 3.2.2 qui est la dernière avant l'apparition du SDK 4, qui a apporté avec lui la nouvelle version Xcode 3.2.3. Il n'y a pas grand-chose à redire sur cet IDE qui est très complet et performant, si ce n'est quelques lenteurs au niveau du débuggeur et des messages d'erreur liés à la mémoire qui ne sont pas toujours très parlants.

Parions qu'Apple aura amélioré cela dans sa prochaine version, Xcode 4 qui, selon le site spécialisé MacGeneration[5], fera son apparition bientôt et permettrait de développer du meilleur code plus rapidement.

[4] Source : icônes récupérées dans le paquet des applications, SDK iPhone 3.2, 2010
[5] LAPORTE C., *WWDC : Apple a dévoilé Xcode 4*, http://www.macgeneration.com, 08.06.2010

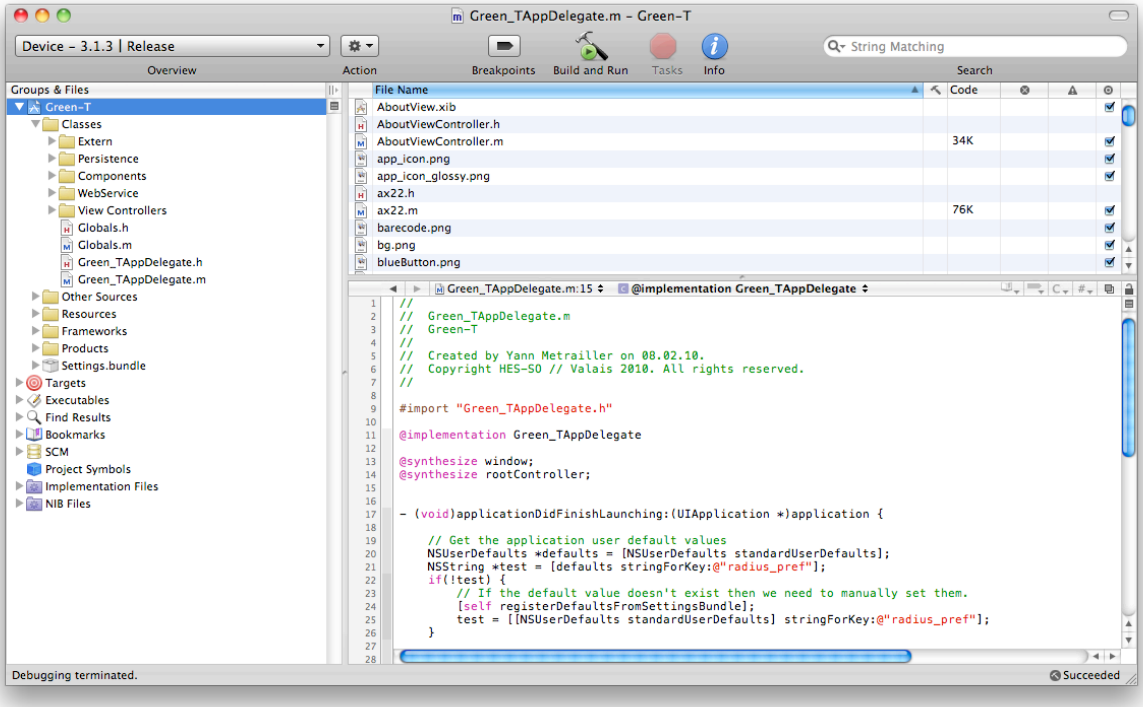


Image 8 - Interface de Xcode

2.3.2 Interface Builder

Interface Builder est l'éditeur graphique permettant de réaliser les interfaces des applications Mac ou iPhone. Concrètement, c'est dans Interface Builder que nous définissons le look de nos applications. C'est là aussi que sont définies les relations entre les objets graphiques et leur code, mais nous verrons cet aspect plus loin dans ce document.

Image 9 - Interface de Interface Builder

2.3.3 Instruments

Instruments est une application qui nous permet de tracer les performances des applications pour déceler les problèmes.

Cet outil rassemble des données comme l'utilisation du disque, de la mémoire ou encore du processeur et propose une visualisation graphique des résultats, pratique pour déterminer la source du problème et accéder aux lignes de code correspondantes.

2.3.4 iPhone Simulator

Le simulateur iPhone permet d'exécuter une application telle qu'elle le serait sur un véritable appareil, à peu de chose près. Bien que celui-ci ne possède pas les ressources hardware d'un iPhone (GPS, appareil photo, accéléromètre), il se lance rapidement et permet de gagner du temps pour tester que l'application fonctionne comme désiré, ou pour le débogage.

2.3.5 Organizer

Nous allons terminer ce chapitre en disant un mot sur l'Organizer. C'est cet outil qui se charge de la gestion des iPhone connectés et des certificats. Il fonctionne avec le trousseau d'accès et s'assure que l'application a bien été signée avec de permettre son installation sur l'appareil.

3 Développement

Nous entrons ici dans le vif du sujet, puisqu'il s'agit en effet du développement à proprement parler. Ce chapitre traitera de quelques principes à connaître pour se lancer dans le développement en Objective-C pour iPhone, de mon expérience personnelle lorsque je l'ai appréhendé, et de l'application Green-T en elle-même. Ce chapitre contient à nouveaux un certain nombre d'éléments tirés du livre « Beginning iPhone Development : Exploring the SDK ».

3.1 Principes

Nous allons ici introduire certains principes de base, ou en tout cas ceux qui me semblent importants à maîtriser, ou qui m'ont donné du fil à retordre.

3.1.1 MVC

Model-View-Controller, ou Modèle-Vue-Contrôleur, est un concept primordial dans Objective-C, comme dans beaucoup d'autres langages, et est un moyen de diviser une application en trois couches :

- Modèle : Couche qui contient les données
- Vue : Couche de présentation (fenêtres, éléments d'interface)
- Contrôleur : Contient les traitement métiers, et fait le lien entre le modèle et la vue

Le but du concept MVC est de créer des objets bien distincts qui répondent à une seule de ces couches, afin de maximiser leur réutilisabilité. Ainsi, une classe qui implémente un bouton ne devrait pas contenir de données, et une classe qui implémente un compte bancaire ne devrait pas dessiner elle-même une table pour afficher ses données.

L'article concernant MVC dans Wikipedia[6] résume très bien l'avantage du concept :

« Un avantage apporté par ce modèle est la clarté de l'architecture qu'il impose. Cela simplifie la tâche du développeur qui tenterait d'effectuer une maintenance ou une amélioration sur le projet. En effet, la modification des traitements ne change en rien la vue. Par exemple on peut passer d'une base de données de type SQL à XML en changeant simplement les traitements d'interaction avec la base, et les vues ne s'en trouvent pas affectées. »

Dans l'application Green-T, ce concept, qui me semble être d'une grande importance, a été respecté autant que possible pour faciliter la maintenance et l'évolution de l'application dans le futur.

3.1.2 Nib File et View Controller

Je traite ici un point qui m'a particulièrement dérouté au départ. En effet, il m'a fallu du temps pour maîtriser la façon de gérer les relations entre les éléments d'interface, créés dans Interface Builder, et les objets qui les implémentent, créés dans Xcode. De plus, ces explications permettront par la suite de mieux comprendre les explications concernant l'application Green-T.

[6] WIKIPEDIA, *Modèle-Vue-Contrôleur*, http://fr.wikipedia.org/wiki/Modèle-Vue-Contrôleur, juillet 2010

3.1.2.1 Nib File

Commençons par parler des « Nib Files ». Sous ce nom se cache en fait les fichiers créés par Interface Builder et qui portent l'extension «.xib ». « Nib File » est le nom hérité de l'ancien format des fichiers qui utilisait l'extension « .nib », et c'est pourquoi les développeurs continuent de les désigner ainsi. Les termes « nib » et « nib files » sont d'ailleurs aussi utilisés dans la documentation Apple.

Voyons donc ci-dessous à quoi ressemble un Nib File (voir Image 10). Il s'agit ici de la fenêtre permettant d'identifier un produit dans Green-T. Nous voyons que plusieurs éléments ont été créés, dont un champ texte pour le code-barres et un bouton pour scanner ce code-barres. Mais, comment manipuler ces éléments, comment leur définir des actions ?

Image 10 - Nib File IdentificationView.xib dans Interface Builder

3.1.2.2 File's Owner

C'est maintenant qu'intervient le « File's Owner », il s'agit toujours de la première icône dans tous les nib files (voir Image 11). C'est cet objet qui est le propriétaire et qui va charger le nib file. Il faut donc lui attribuer la classe créée à cet effet. Dans notre cas, il s'agit de la classe « IdentificationViewController » (voir Image 11).

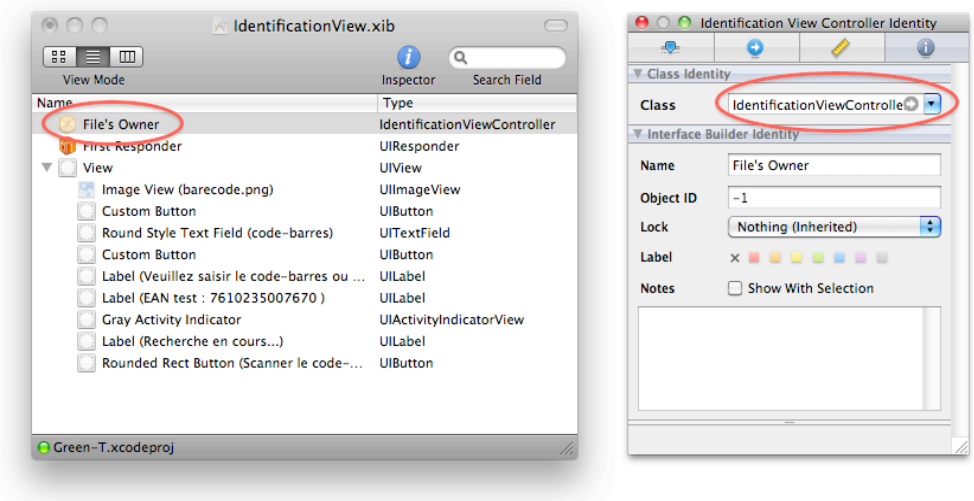


Image 11 - Affectation du File's Owner dans Interface Builder

3.1.2.3 View Controller

Nous avons vu comment créer notre vue (nib file), et comment lui définir quelle est sa classe responsable. Cette classe, qui hérite de « UIViewController », gère les évènements de la vue, les actions disponibles et les éléments d'interfaces. Ainsi, notre classe « IdentificationViewController » est définie comme suit :

```
@interface IdentificationViewController : UIViewController<CKCodePickerDelegate> {

    //Déclaration
    UITextField *codeTextField;
    UIButton *scanButton;
    UIButton *backgroundButton;
    UIActivityIndicatorView *indicatorView;
    UILabel *searchLabel;
}

//Propriétés
@property(nonatomic, retain) IBOutlet UITextField *codeTextField;
@property(nonatomic, retain) IBOutlet UIButton *scanButton;
@property(nonatomic, retain) IBOutlet UIButton *backgroundButton;
@property(nonatomic, retain) IBOutlet UIActivityIndicatorView *indicatorView;
@property(nonatomic, retain) IBOutlet UILabel *searchLabel;

//Actions
-(IBAction)scanCode;
-(IBAction)searchProduct;
-(IBAction)backgroundClick:(id)sender;
-(IBAction)cancel:(id)sender;

@end
```

Nous trouvons donc la déclaration des éléments d'interface, avec toujours notre champ texte (codeTextField), et notre bouton (scanButton) pour le code-barres. En dessous, nous définissons les propriétés pour ces éléments. Nous reviendrons sur ces propriétés, mais ce qui nous intéresse est le mot-clé « `IBOutlet` ». Ce mot clé permet de communiquer à Interface Builder que les variables qu'il précède sont celles que nous voulons connecter aux éléments d'interface afin de pouvoir les manipuler.

De même, nous déclarons des méthodes, comme celle qui effectue le scannage du code-barres (scanCode). Cette méthode est précédée du mot-clé « `IBAction` » qui de nouveau, communique à Interface Builder que cette méthode peut-être invoquée depuis un élément de l'interface.

Il reste ensuite à retourner dans Interface Builder et de faire les connections désirées. Pour connecter les outlets, il faut à l'aide de CTRL+Click faire un glisser-déposer depuis le File's Owner jusqu'à l'élément désiré (voir Image 12).

Pour les actions, il faut tout d'abord sélectionner l'élément qui doit appeler une action définie. Dans l'inspecteur, les différents évenements disponibles s'affichent dans le deuxième onglet. Pour un button, le click correspond à l'événement « Touch Up Inside ». Il faut alors faire un glisser déposer depuis cet événement jusqu'au File's Owner pour définir l'action à cet élément (voir Image 13).

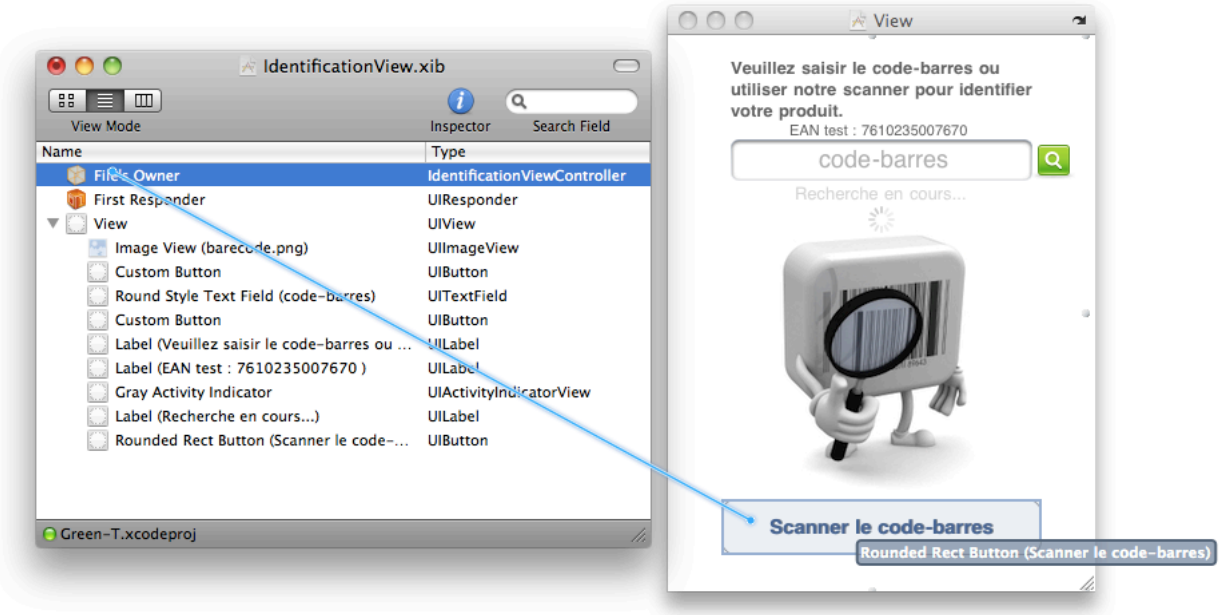


Image 12 - Connection d'un outlet dans Interface Builder

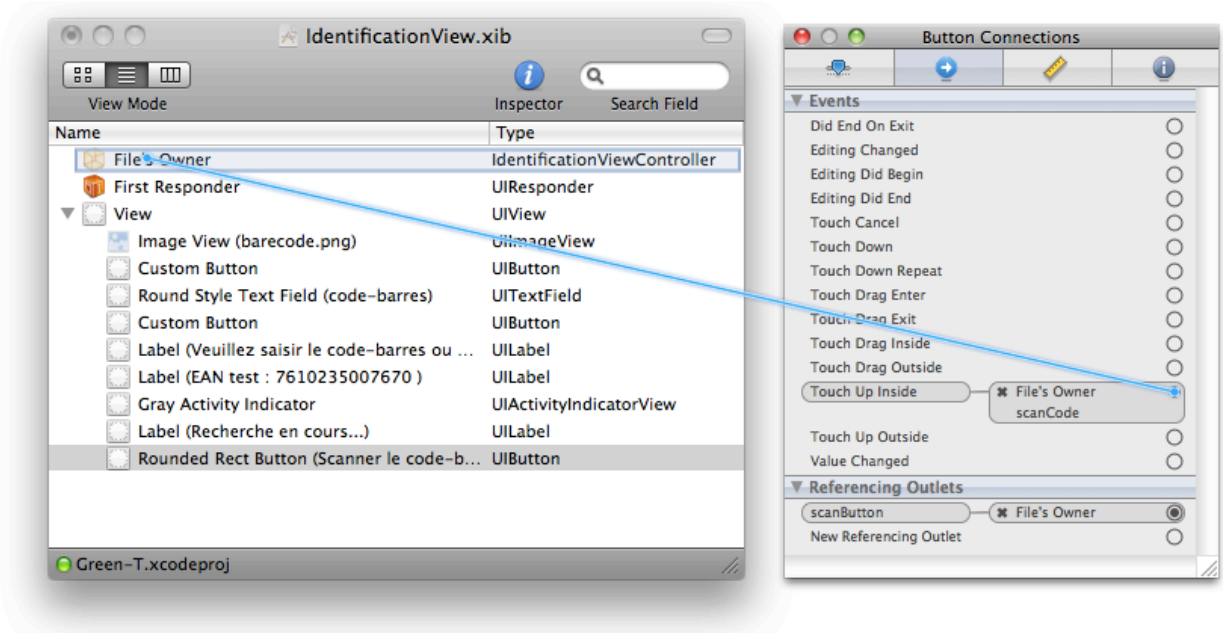


Image 13 - Connexion d'une action dans Interface Builder

3.1.3 Propriétés

Avant que les propriétés soient introduites dans Objective-C, les développeurs devaient écrire les traditionnels « getters » et « setters ». Bien que cette méthode reste toujours bien valide, les propriétés permettent de gagner du temps car il n'est plus nécessaire de définir ces méthodes.

En effet, l'utilisation du mot-clé `@property` dans le fichier de déclaration, combinée à l'utilisation du mot-clé `@synthesize` dans le fichier d'implémentation permet de dire au compilateur de créer les getters et setters au moment de la compilation.

Dans la déclaration, le mot-clé `@property` est suivi par des attributs optionnels entre parenthèses, qui définissent comment les accesseurs et mutateurs vont être crées. La plupart du temps, pour les applications iPhone, les arguments utilisés seront `nonatomic` et `retain`.

Par défaut, les méthodes contiennent du code utile lorsque l'on écrit des applications multitâches. Ce n'est pas le cas lorsque l'on déclare des pointeurs sur des éléments d'interface. En utilisant l'argument `nonatomic` cela évite ce surplus de code.

L'argument `retain` permet de garder la variable en mémoire tant que celle-ci est utilisée. C'est nécessaire, car par défaut l'argument `assign` est utilisé et fonctionne avec le « garbage collector », une fonctionnalité qui n'existe pas actuellement sur l'iPhone.

3.1.4 Gestion de la mémoire

Je ne vais pas détailler le fonctionnement de la mémoire en Objective-C, je n'en ai de toute façon pas les capacités, mais simplement rappeler le principe à assimiler pour s'éviter quelques problèmes lors de l'exécution des applications.

C'est en théorie plutôt simple, puisqu'il suffit de libérer (release) chaque objet qui a été alloué (alloc, retain, copy). Par exemple, un objet qui a été alloué, puis retenu, doit être libéré deux fois (voir Image 14).

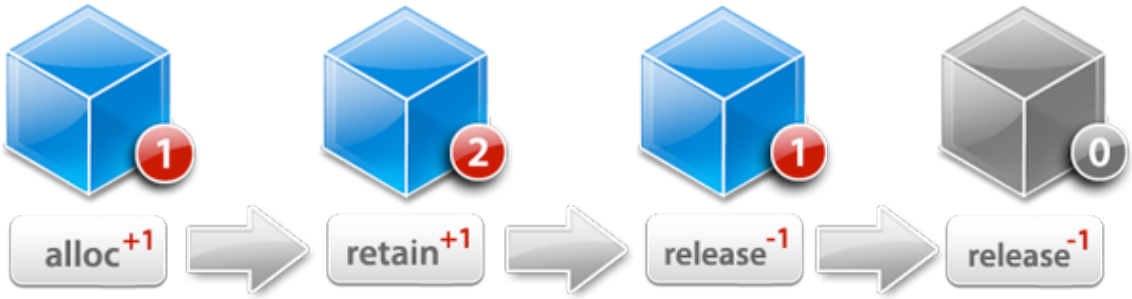


Image 14 - Principe de la gestion de la mémoire[7]

Dans la pratique, ce n'est pas toujours aussi aisé. Il faut être rigoureux et ne pas laisser trop d'objets en mémoire car sur l'iPhone, qui n'a pas de fichier swap, lorsqu'il n'y a plus de mémoire, l'application est simplement quittée.

3.2 Application

Maintenant que les bases ont été posées, nous pouvons à présent nous attaquer au sujet principal de ce travail de bachelor, l'application iPhone Green-T. Nous allons aborder ici concrètement la réalisation de l'application, son intégration dans l'architecture existante, les choix qu'il a fallu faire.

3.2.1 Architecture existante

L'application iPhone est une étape supplémentaire dans la concrétisation du projet Green-T mais, comme évoqué au début de ce document, l'architecture permettant de fournir les services de calcul environnemental existe déjà, et il s'agit de faire en sorte que le client iPhone s'y intègre au mieux.

[7] Source : http://www.cocoadevcentral.com/d/learn_objectivec, 2008

3.2.1.1 Serveur web

Le serveur TomCat mis en place utilise une base de données MySQL pour la persistance des données. Un moteur de persistance Hibernate a été utilisé pour mapper les données relationnelles aux objets de la couche métier réalisée en Java. Le schéma ci-dessous résume cette architecture.

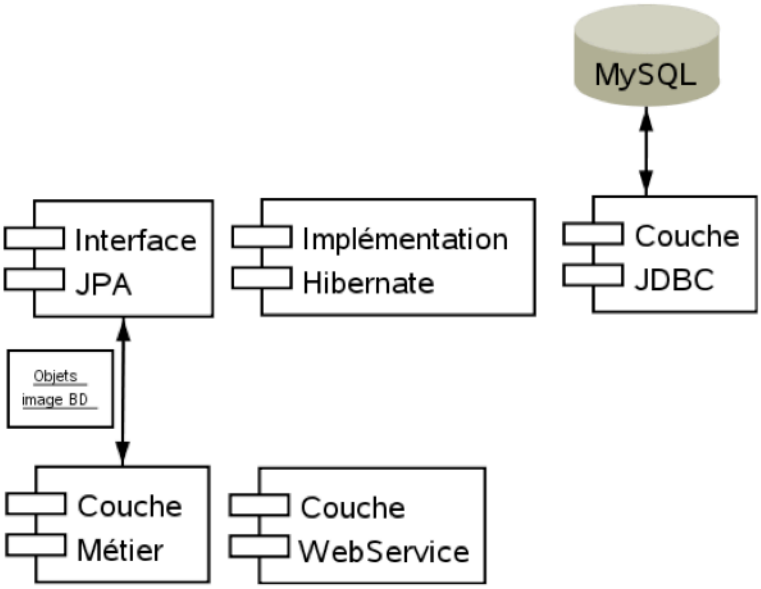


Image 15 - Architecture du serveur web[8]

3.2.1.2 Web Service

Les services web sont réalisés en Java, et déployés dans le serveur grâce à Axis. Les services proposés au début de mon travail ont par la suite été étendus, et d'autres ajoutés, afin de disposer des données nécessaires pour répondre aux besoins définis dans le cahier des charges. Nous aborderons ces modifications plus loin, dans les chapitres concernés. En attendant, voici un aperçu ci-dessous des services proposés au départ :

Magasins

```
public Store[] findStoresByCityName(String city, String name) {}
public Store[] findStoresByLocation(double latitude, double longitude) {}
```

Ces méthodes nous permettent de récupérer une liste de magasins. La première en passant en paramètres la ville et le magasin recherché (Ex : Coop, Sierre), la deuxième en passant en paramètres la latitude et la longitude.

Ecobilan

```
public EcoResult computeEcoResult(String ean13, double latitude, double longitude) {}
public EcoResult computeEcoResultByProductId(int id, double latitude, double longitude) {}
```

Ces méthodes renvoient l'écobilan pour un produit donné. Pour les deux, il faut donner comme paramètres la latitude et longitude du magasin pour lequel il faut calculer l'écobilan. Dans l'une, le

[8] Source : ZUFFEREY D., *Calcul de bilan énergétiques liés au transport des produits de grande consommation,* Travail de Bachleor, août 2008

premier paramètre est le numéro du code-barres (EAN13), dans l'autre il s'agit de l'identifiant du produit.

Famille

```
public EcoResult[] computeEcoResults(String ean13, double latitude, double longitude) {}
public EcoResult[] computeEcoResultsByProductId(int id, double latitude, double longitude) {}
```

Enfin, ces deux dernières méthodes retournent une liste d'écobilans, qui sont en fait les produits appartenant à la même famille que le produit passé en paramètre. Là encore, il faut donner les coordonnées du magasin, et le code-barres ou l'identifiant du produit comme paramètres.

3.2.2 Client iPhone

Notre client iPhone doit donc s'intégrer dans cet environnement. L'existence des services web nous facilite la tâche, puisqu'ils sont le meilleur moyen d'accéder aux données d'une base externe depuis un client mobile.

Les tâches de base de l'application Green-T sont, comme le résume le schéma ci-dessous (voir Image 16), la collecte des informations du consommateur (localisation) et du produit (identification), la communication avec le serveur à travers le web service, et l'affichage des données récupérées. Les capacités de l'iPhone nous permettent de proposer un affichage à la fois complet, pratique et convivial.

De plus, afin d'améliorer le confort de l'utilisateur, l'historique de ses recherches et la possibilité de définir des favoris lui sont proposés pour éviter qu'il doive à chaque fois reproduire le processus complet de recherche d'un produit. A noter que la gestion de favoris n'était pas prévue dans le cahier des charges initial.

Image 16 - Fonctionnement basique du client iPhone

3.2.2.1 Structure

Pour proposer ces fonctionnalités, l'application est composée de quatorze vues à travers lesquelles l'utilisateur peut aisément naviguer. Le point de départ est un menu convivial et clair qui permettra à l'utilisateur de se repérer rapidement. La structure de l'application est détaillée dans le schéma ci-dessous et présente les relations entre les différentes vues (voir Image 17).

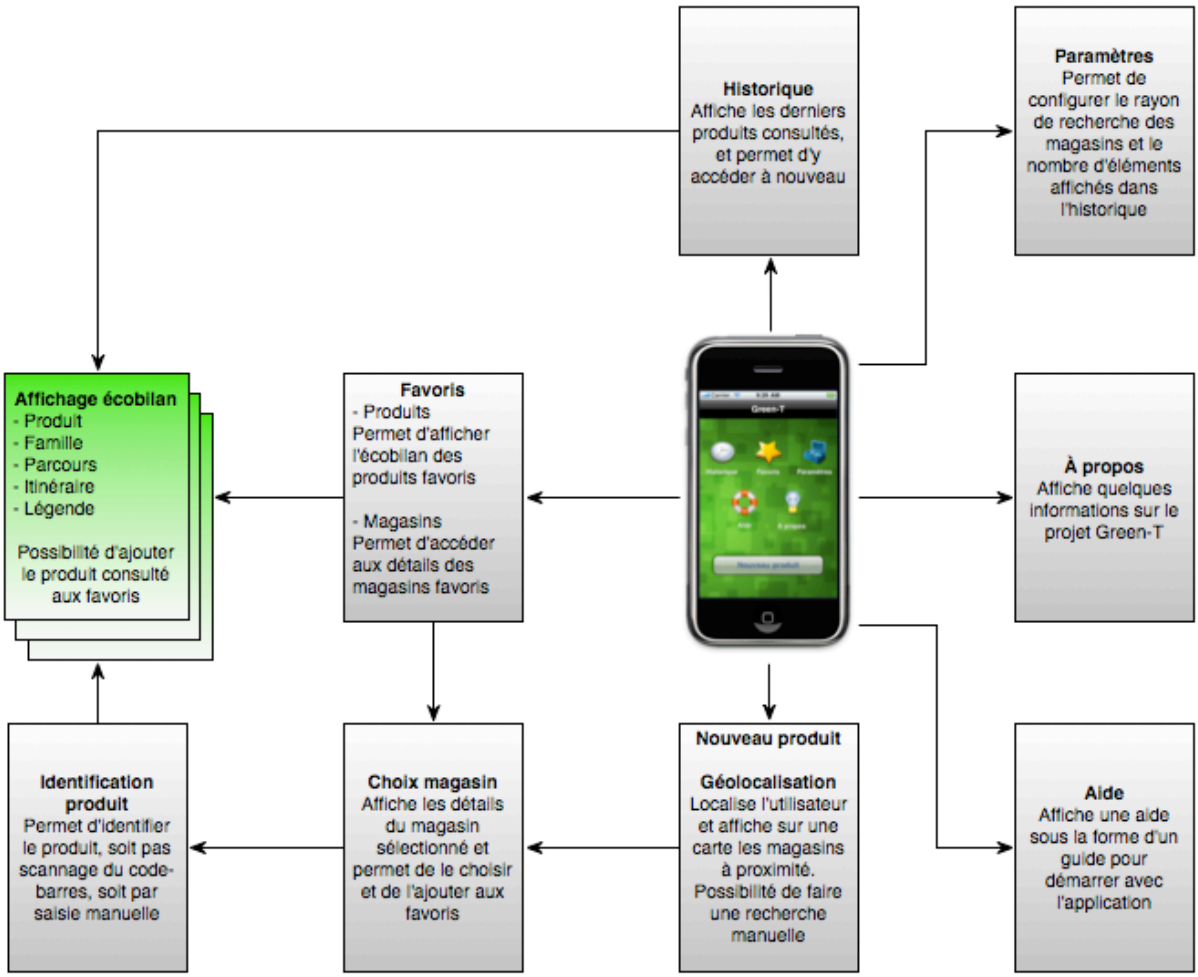


Image 17 - Structure du client iPhone

3.2.2.2 Navigation Controller

Quelques mots à présent pour expliquer le fonctionnement de la navigation entre les vues au sein de l'application. Le contrôleur qui a été utilisé est le « UINavigationController », il gère les vues grâce à une pile, dans laquelle nous trouvons au fond la vue principale (menu) et au sommet la vue qui est actuellement affichée.

Des méthodes sont à disposition pour « pousser » une nouvelle vue dans la pile, ou pour la retirer. A noter que le Navigation Controller ajoute automatiquement un bouton permettant à l'utilisateur de revenir à la vue précédente (Voir Image 18). Ainsi, le code suivant permet d'afficher une nouvelle vue (nextController) en animant la transition.

```
[self.navigationController pushViewController:nextController animated:YES];
```

Alors que ce code nous sert à revenir à la vue tout au fond de la pile, qui est chez nous notre menu.

```
[self.navigationController popToRootViewControllerAnimated:TRUE];
```

L'image ci-dessous, tirée du site développeur d'Apple, illustre parfaitement le fonctionnement de ce Navigation Controller.

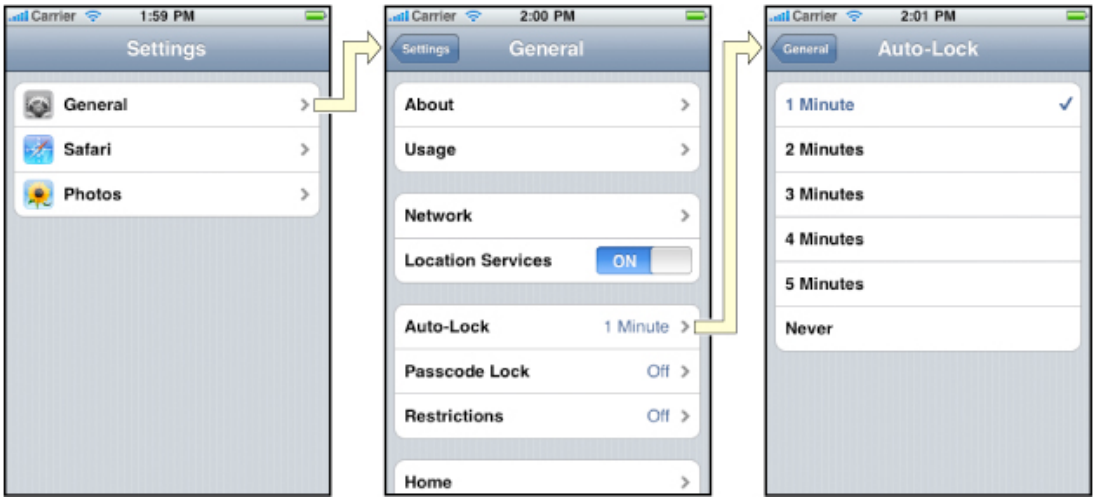


Image 18 - Principe du Navigation Controller

3.2.3 Traitements métiers

3.2.3.1 Variables globales

Il est indispensable de pouvoir stocker certaines variables afin qu'elle soient disponibles à partir de toutes les vues de l'application. Ainsi, nous avons fait en sorte de pouvoir sauvegarder le magasin et le produit en cours une fois qu'ils ont été choisis. Nous aurions pu simplement stocker ces variables dans le « AppDelegate » qui est accessible de partout mais cette solution n'est pas propre et peut vite amener du désordre. C'est pourquoi j'ai adopté la solution proposée par Johann "John" Dowa sur son blog[9]. Il s'agit de créer une classe qui utilise le Singleton design pattern. Dans notre cas, la classe « Globals » contient le code suivant :

```
+ (Globals *)sharedInstance
{
    // the instance of this class is stored here
    static Globals *myInstance = nil;

    // check to see if an instance already exists
    if (nil == myInstance) {
        myInstance  = [[[self class] alloc] init];

        // initialize variables here
        myInstance.currentStore = [[ax22_Store alloc] init];
        myInstance.currentEcoResult = [[ax22_EcoResult alloc] init];

        UIImage *buttonImageNormal = [UIImage imageNamed:@"btn_bg.png"];
        UIImage *buttonImagePressed = [UIImage imageNamed:@"btn_bg_on.png"];

        myInstance.stretchableButtonImageNormal = [buttonImageNormal
                                                   stretchableImageWithLeftCapWidth:12
                                                   topCapHeight:0];

        myInstance.stretchableButtonImagePressed = [buttonImagePressed
                                                    stretchableImageWithLeftCapWidth:12
                                                    topCapHeight:0];
    }
    // return the instance of this class
    return myInstance;
}
```

Cette classe contient nos variables et conserve les valeurs grâce au Singleton pattern qui initialise une seule fois l'instance. Dans notre programme, pour accéder ou définir notre variable, nous n'avons plus qu'à utiliser le code suivant :

[9] DOWA J., Global Variables in iPhone Objective-C, http://maniacdev.com/2009/07/global-variables-in-iphone-objective-c, juin 2009

```
[Globals sharedInstance].currentStore = store;

ax22_Store *store = [Globals sharedInstance].currentStore;
```

3.2.3.2 Consommation du service web

Pour accéder au service web depuis Objective-C, il nous fallait dans un premier temps générer les stubs. Nous aurions pu écrire nos propres requêtes http, mais cela aurait considérablement prolongé la durée du projet. Apple propose bien un outil, « WSMakeStubs », pour générer ces stubs, mais il ne fonctionne que pour les projets Mac et pas pour iPhone. Par conséquent, des solutions ont été cherchées, et deux ont été trouvées.

gSOAP

gSOAP est un outil en ligne de commande qui permet de générer des stubs en C++. Xcode gérant ce langage, ils peuvent être utilisés au sein de notre projet en Objective-C. C'est la première solution qui a été implémentée grâce aux explications fournies par Sébastien HOERNER dans son article[10], et qui a été utilisée au début du projet. Toutefois, la cohabitation des deux langages (C++ et Objective-C) n'était pas aisée, il fallait gérer la conversion des variables dans les différents types, ce qui était un travail fastidieux qui ralentissait l'avancement du développement. C'est pourquoi il a été décidé de se tourner vers la deuxième méthode décrite ci-dessous.

Wsdl2objc

Cet outil génère lui aussi les stubs, mais en Objective-C cette fois. C'est un grand avantage, surtout qu'il possède, contrairement à gSOAP, une interface graphique qui facilite son utilisation (voir Image 19). Les fichiers générés étant en Objective-C, leur intégration et leur utilisation sont bien plus simples. C'est pourquoi, en cours de projet, nous avons modifié ces stubs d'accès au service web.

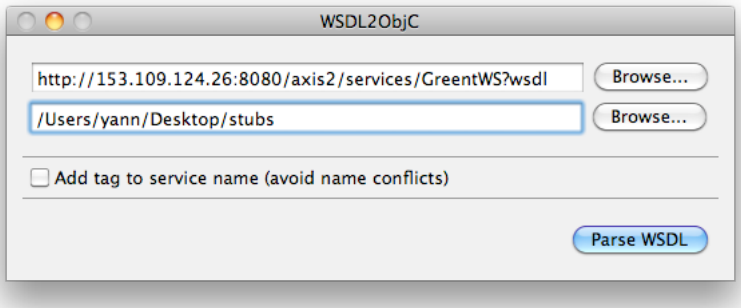


Image 19 - Interface de wsdl2objc

Le fait de changer ces stubs en cours de projet n'a eu que peu d'impact, car une couche intermédiaire avait été créée afin de gérer les appels au service web. Cette couche est une classe qui définit les méthodes disponibles et nous retourne les objets désirés. C'est cette classe, nommée « WSUtils » qui dialogue avec les stubs, ce qui permet d'être très souple, car il suffit simplement d'adapter cette classe si de nouveaux changements interviennent au niveau des stubs. Voici à quoi ressemble l'appel au service pour récupérer un écobilan.

[10] HOERNER S., *Appel de web services SOAP sur iPhone,* http://www.sebhoerner.fr/?p=127, mai 2009

```
+(ax22_EcoResult*)computeEcoResult:(NSString *)ean13Product
                          latitude:(double)latitude longitude:(double)longitude
{
    ax22_EcoResult *result = nil;

    GreentWSSoap12Binding *binding = [GreentWS GreentWSSoap12Binding];
    binding.logXMLInOut = NO;

    GreentWS_computeEcoResult *params = [[GreentWS_computeEcoResult alloc] init];
    params.ean13 = ean13Product;
    params.latitude = [NSNumber numberWithDouble:latitude];
    params.longitude = [NSNumber numberWithDouble:longitude];

    GreentWSSoap12BindingResponse *response = [binding computeEcoResultUsingParameters:params];

    NSArray *responseBodyParts = response.bodyParts;

    for(id bodyPart in responseBodyParts)
    {
        /****
         * SOAP Fault Error
         ****/
        if ([bodyPart isKindOfClass:[SOAPFault class]])
        {
            // You can get the error like this:
            NSLog(@"%@", ((SOAPFault *)bodyPart).simpleFaultString);
            continue;
        }

        /****
         * Get Eco result
         ****/
        if([bodyPart isKindOfClass:[GreentWS_computeEcoResultResponse class]])
        {
            GreentWS_computeEcoResultResponse *ecoResultResponse =
                                    (GreentWS_computeEcoResultResponse*)bodyPart;
            result = ecoResultResponse.return_;
            if(result.productEan13 == nil)
                result = nil;
        }
    }
    return result;
}
```

Ainsi, dans notre application, pour récupérer un écobilan, ce bout de code suffit, et n'aura pas à être modifié en cas de changement au niveau du service web, puisque les adaptations se feront dans la classe « WSUtils » :

```
ax22_EcoResult *result = nil;

result = [WSUtils computeEcoResult:codeTextField.text
                          latitude:[store.latitude doubleValue]
                         longitude:[store.longitude doubleValue]];
```

3.2.3.3 Persistance

Dans notre application, nous stockons l'historique et les favoris en local pour chaque utilisateur. Etant donné que cela ne représente pas une grosse quantité de données, nous aurions pu nous contenter de les sauvegarder dans des « Property Lists », dont le format est XML. Mais cela offrait l'opportunité de s'initier à SQLite, et comme le langage SQL est déjà connu, c'est ce choix qui s'est imposé.

Création de la base de données

Une couche de persistance a donc été réalisée pour gérer l'accès et l'écriture des données dans la base de données, qui est en fait un fichier qui a été préalablement créé par un outil ou par ligne de commande. Dans notre cas, nous l'avons fait en passant par le terminal (voir Image 20).

```
Terminal — sqlite3 — 133×9
Last login: Sun Jul  4 22:17:09 on ttys000
MacBook-Pro-de-Yann-M:~ yann$ sqlite3 greentDB.sql
SQLite version 3.6.12
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> CREATE TABLE greent_histo ("productId" INTEGER NOT NULL , "productEan13" VARCHAR NOT NULL , "productName" VARCHAR NOT NULL ,
"storeId" INTEGER NOT NULL , "storeName" VARCHAR NOT NULL , "storeLat" DOUBLE NOT NULL , "storeLong" DOUBLE NOT NULL , "creationDate"
 DATETIME NOT NULL , "type" INTEGER NOT NULL,PRIMARY KEY ("productId", "storeId", "creationDate"));
```

Image 20 - Création de la base de données SQL

Objet persistant

La deuxième étape consiste à créer une classe pour contenir les données récupérées de la base. La classe « History » fait donc son apparition. On y retrouve les mêmes membres que ceux créés dans la table « greent_histo » ci-dessus.

```
@interface History : NSObject {
    int productId;
    NSString *productEan13;
    NSString *productName;
    int storeId;
    NSString *storeName;
    double storeLat;
    double storeLong;
    NSDate *creationDate;
    int type;
}

@property int productId;
@property int storeId;
@property (retain, nonatomic) NSString *productEan13;
@property (retain, nonatomic) NSString *productName;
@property (retain, nonatomic) NSString *storeName;
@property double storeLat;
@property double storeLong;
@property (retain, nonatomic) NSDate *creationDate;
@property int type;
```

Interagir avec la base de données

Il reste maintenant à créer une classe qui interagit avec la base de données. Elle est nommée « DBManager » et propose les méthodes d'initialisation et de création de la base, de lecture et d'écriture des enregistrements. Elle déclare aussi un tableau NSArray pour récupérer les données.

```
@interface DBManager : NSObject {
    NSString *dbName;
    NSString *dbPath;

    NSMutableArray *historyArray;
}

@property (nonatomic, retain) NSMutableArray *historyArray;

-(id) initDatabase;
-(void) checkAndCreateDatabase;
-(void) readHistory:(int)type;
-(int)  insertHistory:(History*)newHistory;
-(int) deleteHistory:(History*)oldHistory;
```

Les premières méthodes de cette classe servent à initialiser la base de données. Pour cela, on vérifie si la base a été créée sur l'iPhone et, si ce n'est pas le cas, on copie sur l'iPhone la base de données (fichier greentDB.sql) créée plus haut qui se trouve dans les ressources de l'application.

```
-(id) initDatabase{

    // Define the databse name
    dbName = @"greentDB.sql";

    // Get the path tho the database file
    NSArray *documentPaths = NSSearchPathForDirectoriesInDomains(
                                NSDocumentDirectory, NSUserDomainMask, YES);
    NSString *documentsDir = [documentPaths objectAtIndex:0];
    dbPath = [documentsDir stringByAppendingPathComponent:dbName];

    [self checkAndCreateDatabase];

    return self;
}

-(void) checkAndCreateDatabase{

    BOOL success;

    // Check if the database is already created on iPhone
    NSFileManager *fileManager = [NSFileManager defaultManager];
    success = [fileManager fileExistsAtPath:dbPath];

    // Database exists
    if(success) return;

    // Database does not exists
    // Copy database from app to filesystem
    NSString *databasePathFromApp = [[[NSBundle mainBundle] resourcePath]
                                     stringByAppendingPathComponent:dbName];
    [fileManager copyItemAtPath:databasePathFromApp toPath:dbPath error:nil];

    [fileManager release];
}
```

Ci-dessous, voilà comment s'effectue l'appel au « DBManager » pour récupérer l'historique. Cette partie de code ne changera pas si l'on change de base de données ou si on la modifie. En effet, comme la communication avec la base de données s'effectue dans le « DBManager », il suffira d'adapter cette classe. Encore une fois, cette méthode de faire permet une grande souplesse pour la maintenance.

```
//Get data from database
DBManager *dbManager = [[DBManager alloc] initDatabase];
[dbManager readHistory:DB_TYPE_HISTO];
self.data = dbManager.historyArray;
[dbManager release];
```

Enfin, nous pouvons ajouter à ce « DBManager » les méthodes pour manipuler les données de la base. Nous y trouvons pour Green-T les fonctions qui permettent d'ajouter et de supprimer des entrées de l'historique, et celle qui sert à récupérer l'historique.

A noter que les favoris sont enregistrés en tant qu'élément de l'historique. En effet, les données à stocker étant les mêmes, il aurait été superflu de faire une table supplémentaire. Les enregistrements de l'historique et ceux des favoris sont distingués grâce au champ « type ».

Voyons comme exemple le code du « DBManager » qui permet de lire les données de la table historique. Dans ce code, on parcourt les enregistrements de la base, crée un objet « History » pour chacun, et l'ajoute au tableau de résultats.

Pour les autres opérations (insertion, suppression) le principe est sensiblement le même.

```
-(void) readHistory:(int) type {

    sqlite3 *database;

    // NSDateFormatter to format dates
    NSDateFormatter *inputFormatter = [[NSDateFormatter alloc] init];
    [inputFormatter setDateFormat:@"yyyyMMddHHmmss"];

    // Array to store the results
    historyArray = [[NSMutableArray alloc] init];

    // Open database from filesystem
    if(sqlite3_open([dbPath UTF8String], &database) == SQLITE_OK) {

        NSString *sqlStatement = [NSString stringWithFormat:
                    @"select * from greent_histo WHERE type = %i ORDER BY creationDate DESC", type];

        // Object to get the request execution status
        sqlite3_stmt *compiledStatement;

        // Execute request and read results
        if(sqlite3_prepare_v2(database, [sqlStatement UTF8String], -1, &compiledStatement, NULL) == SQLITE_OK) {

            while(sqlite3_step(compiledStatement) == SQLITE_ROW) {
                History *histo = [[History alloc] init];
                histo.productId = sqlite3_column_int(compiledStatement, 0);
                histo.productEan13 = [NSString stringWithUTF8String:(char *)sqlite3_column_text(compiledStatement, 1)];
                histo.productName = [NSString stringWithUTF8String:(char *)sqlite3_column_text(compiledStatement, 2)];
                histo.storeId = sqlite3_column_int(compiledStatement, 3);
                histo.storeName = [NSString stringWithUTF8String:(char *)sqlite3_column_text(compiledStatement, 4)];
                histo.storeLat = sqlite3_column_double(compiledStatement, 5);
                histo.storeLong = sqlite3_column_double(compiledStatement, 6);
                histo.creationDate = [inputFormatter dateFromString:
                                      [NSString stringWithUTF8String:(char *)sqlite3_column_text(compiledStatement, 7)]];
                histo.type = sqlite3_column_int(compiledStatement, 8);

                // Add result to results array
                [historyArray addObject:histo];
                [histo release];
            }
        }
        // Release compiledStamenent
        sqlite3_finalize(compiledStatement);
    }
    [inputFormatter release];

    // Close database
    sqlite3_close(database);
}
```

3.2.4 Fonctionnalités de base

3.2.4.1 Géolocalisation

Image 21 - Vue localisation

Lors de l'ajout d'un nouveau produit, la première étape est de localiser le magasin dans lequel le consommateur va faire ses achats.

Pour ce faire, la position de l'utilisateur sera déterminée grâce à la puce aGPS de l'iPhone et affichée sur une carte avec les magasins se trouvant à proximité de cette position et stockés dans la base de données (voir Image 21). Dans le cas où la localisation par GPS ne fonctionnerait pas, ou mal, il reste la possibilité de rechercher une adresse par saisie manuelle. Ainsi, dans tous les cas, le consommateur devrait pouvoir retrouver le magasin désiré.

Points importants

- Géolocalisation grâce à la classe « CLLocationManager »
- Récupération des magasins à travers le service web
- Affichage des informations sur une carte « MKMapView »
- « Forward Geocoding » permettant de récupérer les coordonnées d'une adresse

Utilisation de la classe CLLocationManager

La classe CLLocationManager nous permet donc de récupérer la position de l'utilisateur. Il s'agit de la première chose à faire puisque nous voulons lui proposer uniquement les magasins qui se trouvent proches de lui. Cette classe se trouve dans le Framework « CoreLocation », qu'il faut par conséquent ajouter à son projet (voir Image 22).

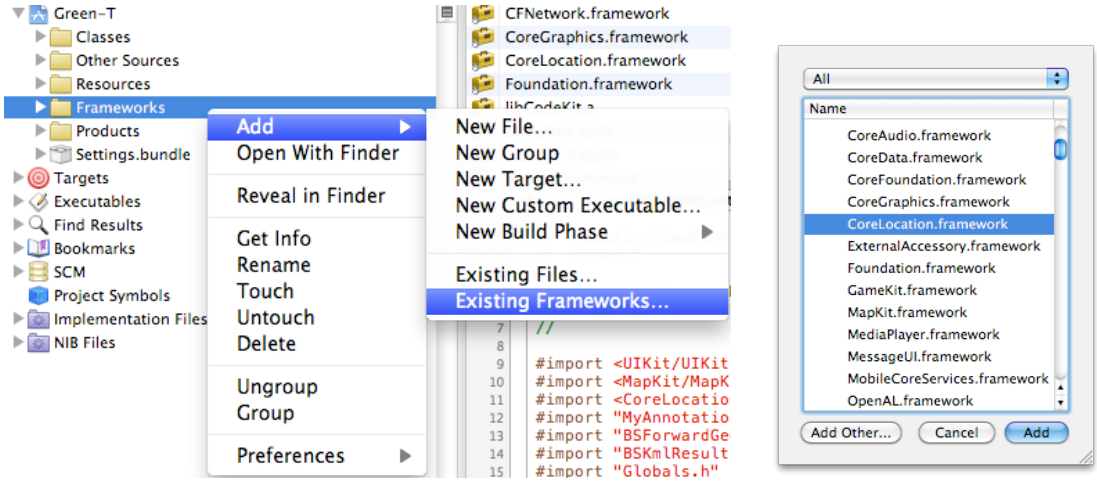


Image 22 - Ajout du Framework CoreLocation

Ensuite, en créant une instance de cette classe, nous pouvons lui définir certaines propriétés et lancer la recherche. Ces propriétés sont la précision à atteindre et la distance à parcourir entre les mises à jour de la localisation. Il faut aussi définir quel est le délégué, la classe qui recevra les évènements. Le « location manager » est finalement démarré grâce à la méthode « startUpdatingLocation ». A partir de là, la puce aGPS fait son travail pour localiser l'appareil et envoie un message au délégué lorsque la position est mise à jour. Il faut donc implémenter la méthode du délégué dans notre classe. Elle nous permet de récupérer l'instance de CLLocationManager en question, ainsi que la nouvelle et l'ancienne position :

```
// Implement the location update
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
           fromLocation:(CLLocation *)oldLocation { }
```

Grâce à la nouvelle position trouvée (newLocation), nous pouvons simplement centrer la carte sur celle-ci en récupérant ses coordonnées.

Récupération des magasins à travers le service web

Nous avons vu précédemment qu'un service était proposé pour récupérer les magasins. La recherche était effectuée selon un rayon défini au niveau du paramétrage du serveur. L'objectif était que l'utilisateur puisse choisir le périmètre dans lequel rechercher les magasins. C'est pourquoi une nouvelle procédure a été créée, nommée « findStoresByLocationRadius ». Elle est identique à la méthode existante, seulement qu'il y a un paramètre en plus, le rayon. Cela permet maintenant de chercher les magasins en définissant le rayon de recherche depuis l'application.

Nous avons implémenté ce nouveau service dans notre classe « WSUtils » qui nous permet alors de nous retourner un tableau de magasins, en lui donnant les coordonnées de la position trouvée grâce à la géolocalisation ainsi que le rayon choisi par l'utilisateur.

```
//Get the stores trough the web service
storesArray = [[NSMutableArray alloc] initWithArray:
               [WSUtils findStoresByLocationRadius:latitude longitude:longitude radius:radius]];
```

Nous pouvons alors parcourir ce tableau pour lire les magasins et les ajouter à notre carte.

Utilisation de la classe MKMapView pour afficher une carte

La classe permettant d'afficher une carte géographique est « MKMapView », elle se trouve cette fois dans le Framework « MapKit » qu'il faut aussi ajouter au projet. Dans Green-T, elle permet d'afficher la localisation de l'utilisateur (ou l'adresse recherchée), ainsi que les magasins autour de cette position. Cette carte a été ajoutée à notre vue dans Interface Builder, et reliée à notre variable de type MKMapView dans Xcode à travers un outlet. Le délégué de la carte est défini à la classe courante, qui doit implémenter le protocole « MKMapViewDelegate ».

Ces points affichés sur la carte sont des annotations, et sont gérés grâce à la classe « MKAnnotation ». Une classe « MyAnnotation » a été créée et hérite de « MKAnnotation », elle nous permet d'avoir des annotations personnalisées qui correspondent tout à fait à nos besoins.

```
@interface MyAnnotation : UIControl <MKAnnotation>
{
    CLLocationCoordinate2D coordinate;
    NSString *title;
    NSString *subtitle;
    int idStore;
    int step;
}
```

Il est alors possible de créer une nouvelle annotation et de l'ajouter sur la carte avec le simple code ci-dessous, ou « map » est la carte, et « ann » l'annotation.

```
[map addAnnotation:ann];
```

Le délégué doit implémenter la méthode qui prend en charge la visualisation de l'annotation sur la carte.

```
- (MKAnnotationView *) mapView:(MKMapView *)mapView
             viewForAnnotation:(id <MKAnnotation>) annotation { }
```

Forward Geocoding

Tout d'abord, il faut différencier le « Forward Geocoding », qui consiste à trouver les coordonnées à partir d'une adresse, au « Reverse Geocoding » qui lui associe une adresse à des coordonnées.

Apple offre une classe qui permet de faire du « Reverse Geocoding ». Il s'agit de « MKReverseGeocoder » qui retourne un objet dans lequel on retrouve quelques informations comme la localité, le code postal ou le pays.

Mais ce qui nous intéresse ici, c'est de pouvoir faire l'inverse. En effet, l'utilisateur doit pouvoir saisir une adresse qui sera affichée sur la carte. Et pour cela, Apple ne propose rien. Il a donc fallu trouver une alternative.

La première qui a été trouvée est « GMGeocoder » (http://code.google.com/p/gmgeocoder). Il s'agit d'une simple classe qui utilise http://tinygeocoder.com pour géocoder les adresses. Dans un premier temps, cette solution convenait très bien et a été utilisée pendant un certain temps. Mais suite à quelques tests, il s'est avéré que pour des recherches précises, avec un nom de rue par exemple, GMGeocoder était inefficace.

Suite à ce constat, une nouvelle alternative a été cherchée. Et elle a été trouvée sur le blog http://blog.sallarp.com. Son auteur propose en effet une API complète qui utilise les services de

Google, avec explications et code d'exemple. Il a donc fallu procéder à quelques adaptations pour implémenter cette API à la place de GMGeocoder. Mais la résultant est bluffant, même des petites rues de la ville de Sion sont trouvées !

Grâce à cette API, il est donc possible de récupérer les coordonnées géographiques de l'adresse saisie par l'utilisateur. Il ne reste plus qu'à afficher cette adresse sur la carte en procédant de la même manière que pour afficher un magasin, grâce à une annotation.

3.2.4.2 Affichage du détail d'un magasin

Image 23 - Vue détails magasin

Lorsque l'on choisit un magasin sur la carte, on accède à ses détails, comme l'adresse ou le n° de téléphone.

Cette vue est relativement simple et ne pose pas de problème particulier, dans le sens où elle ne fait qu'afficher les détails du magasin courant dans une table.

Un bouton permet d'ajouter le magasin à ses favoris. Nous verrons à quel point il est simple de le faire grâce à la classe « DBManager » qui a été décrite précédemment.

Points importants

- Modification du service web pour obtenir les détails d'un magasin
- Ajout d'un magasin aux favoris

Détails d'un magasin

L'objet « Store » retourné par le service web était au départ composé uniquement de l'id, de la latitude et de la longitude. Il manquait donc des données pour pouvoir afficher les détails de magasin sélectionné par l'utilisateur. C'est pourquoi la classe a été étendue afin d'avoir à disposition toutes les informations nécessaires. Les services ont eux aussi été adaptés afin de remplir correctement les objets.

```
public class Store {
    private int id;
    private double latitude;
    private double longitude;
    private String name;
    private String description;
    private String address;
    private String zipCode;
    private String city;
    private String country;
    private String emailContact;
    private String faxContact;
    private String phoneContact;
}
```

A présent, lorsque l'utilisateur choisit un magasin, il accède à tous ses détails. En arrière-plan, ce magasin a été sauvegardé dans les variables globales, ce qui permet à tout moment d'y accéder.

Ajout d'un magasin aux favoris

L'utilisateur a la possibilité d'ajouter ses magasins préférés aux favoris grâce à un bouton. La classe « DBManager » qui a été créée auparavant nous facilite la tâche, puisque la grande partie du travail est déjà faite. En effet, c'est cette classe qui gère les manipulations de la base de données, et contient donc une procédure pour insérer un nouvel enregistrement.

Le code qui se cache derrière ce bouton se contente alors de créer un nouvel objet « History » contenant les données du magasin en question, et d'initialiser le « DBManager » pour effectuer l'insertion. Notons ici que le champ « type » est initialisé avec une valeur permettant de distinguer cet enregistrement comme étant un magasin favori.

```
//Init database manager
DBManager *dbManager = [[DBManager alloc] initDatabase];

//Get current store
ax22_Store *store = [Globals sharedInstance].currentStore;

//New history
History *histo = [[History alloc] init];
histo.productId = 0;
histo.productEan13 = nil;
histo.productName = nil;
histo.storeId = [store.id_ intValue];
histo.storeName = store.name;
histo.storeLat = [store.latitude doubleValue];
histo.storeLong = [store.longitude doubleValue];
histo.creationDate = [NSDate date];
histo.type = DB_TYPE_STORE_FAV;

//Insert into history
int res;

res = [dbManager insertHistory:histo];
```

3.2.4.3 Identification du produit

Image 24 - Vue identification produit

Une fois que le magasin a été choisi, il faut identifier le produit pour lequel on veut calculer le bilan écologique.

Cette vue, qui paraît basique au premier coup d'œil cache bien son jeu puisque c'est le point de départ pour scanner un code-barres. Un champ texte permet tout de même de saisir manuellement le numéro d'un produit au cas où l'appareil utilisé n'aurait pas d'appareil photo, où si l'identification grâce au lecteur de code-barres ne fonctionnerait pas.

Points importants

- Choix de la méthode de lecture des codes-barres
- Utilisation de la librairie CodeKit
- Récupération d'un écobilan à travers le service web

Choix de la méthode de lecture des codes-barres

Une des fonctionnalités majeures de l'application Green-T est la lecture des codes-barres. Celle-ci permet à l'utilisateur de gagner du temps pour identifier un produit. Il a donc fallu trouver une solution existante à intégrer à notre projet, ou envisager le portage d'une librairie provenant d'un autre langage.

Le RFID Center, dont les locaux se trouvent au même endroit que l'institut informatique de l'HES-SO, avait déjà développé une librairie de lecture de codes-barres pour l'iPhone. Une application qui l'implémente est d'ailleurs disponible sur l'App Store. Une demande a donc été faite en vue d'obtenir une version de cette librairie, ainsi que l'autorisation de l'utiliser.

Dans l'intervalle, une autre alternative a été envisagée. « RedLaser » est une autre application disponible gratuitement sur l'AppStore. Celle-ci est réellement étonnante, la lecture des codes-barres et vraiment efficace et rapide, même sur un iPhone 3G ne disposant pas de l'autofocus. Ce n'est pas le cas de la librairie du RFID Center. La société qui a créé cette application, « Occipital », propose un SDK qui permet d'implémenter cette fonctionnalité dans nos applications. Depuis, « RedLaser » a été racheté par la eBay, ce qui démontre son potentiel certain.

Bien que cette dernière solution paraisse vraiment prometteuse, c'est la librairie du RFID Center qui a été utilisée, en raison de la proximité et de la collaboration entre les deux instituts.

Utilisation de la librairie CodeKit

C'est donc la librairie fournie par le RFID Center, « CodeKit » qui a été implémentée dans notre application. Celle-ci a été transmise accompagnée d'un guide d'installation ainsi qu'un exemple complet de sa mise en œuvre, ce qui nous a permis de l'installer sans encombres. En effet, en quelques étapes simples, clairement expliquées dans la documentation, la librairie était prête à fonctionner. Ci-dessous les méthodes permettant de récupérer les évènements, elles appartiennent au protocole « CKCodePickerDelegate » qui doit être implémenté dans la classe utilisant la librairie pour fonctionner.

Method	Description
-(void)pickerDidCancel:(CKCodePicker *)picker;	Informs you if the user canceled decoding
-(void)picker:(CKCodePicker *)picker didErrorOccur:(NSError *)error;	Informs you when a general error occurred
-(void)picker:(CKCodePicker *)picker didFindCode:(NSString *)code;	Informs you when the decoder finds a code and gives it to you. *Note that the decoding process is automatically stopped!*

Image 25 - Méthodes du délégué CodeKit[11]

A l'utilisation, la librairie se montre performante, et reconnaît en général parfaitement les codes-barres dans un délai tout à fait convenable. Le petit bémol est qu'il est nécessaire de disposer de l'autofocus. En effet, sur les iPhone 3G qui n'en disposent pas, la lecture ne fonctionne pas, à mois de recourir à l'utilisation d'une coque intégrant une lentille « macro ».

[11] Source : Documentation de la librairie CodeKit, 2010

Récupération d'un écobilan à travers le service web

Maintenant que le produit a été identifié, soit par le scannage du code-barres, soit par la saisie manuelle, nous avons toutes les informations pour récupérer le bilan écologique. Pour ce faire, il existe le service qui nous retourne cet écobilan en lui passant en paramètres le code du produit (EAN13), la latitude et la longitude du magasin.

L'EAN13 est récupéré dans le champ texte, et les coordonnées du magasin dans notre objet « Store » courant, qui est toujours stocké dans les variables globales.

```
//Get the eco result trough the web service
ax22_Store *store = [[Globals sharedInstance] currentStore];
ax22_EcoResult *result = nil;

result = [WSUtils computeEcoResult:codeTextField.text
                          latitude:[store.latitude doubleValue]
                         longitude:[store.longitude doubleValue]];
```

L'objet « EcoResult » retourné était lui aussi incomplet par rapport aux besoins exprimés dans le cahier des charges. En effet, il manquait certaines informations intéressantes, comme le producteur ou le pays du produit analysé. C'est pourquoi cette classe a aussi été étendue afin de disposer une nouvelle fois de toutes les données. Grâce à ces adaptations, nous disposons de toutes les informations pour afficher les résultats du bilan écologique.

```
public class EcoResult implements Comparable<EcoResult> {
    private int productId;
    private String productEan13;
    private String productName;
    private String productDescription;
    private int ranking;
    private double kgCO2;
    private String brand;
    private String family;
    private String coutry;
    private String producer;
}
```

3.2.4.4 Affichage étiquette énergétique

Image 26 - Vue affichage étiquette

Cette vue est très importante, puisqu'il s'agit de celle sur laquelle repose tout l'intérêt de l'application Green-T. Elle affiche toutes les informations d'un produit et son écobilan, et est accessible de plusieurs endroits, que ce soit par le flux normal, l'historique ou les favoris.

Cette vue, qui est en fait composée de 5 vues, affiche dans un premier temps le produit et le bilan écologique. Un sous-menu permet d'accéder aux autres vues (famille du produit, parcours, itinéraire, légende). Il est aussi possible d'ajouter un produit aux favoris depuis cette vue.

Il n'y a pas de problème particulier ici puisqu'il s'agit d'afficher les données récupérées depuis le service web dans des tables, chose qui a déjà été faite dans les vues précédentes.

Points importants

- Gestion des sous-vues
- Cellules de table customisées
- Ajout d'un produit aux favoris

Gestion des sous-vues

L'affichage de l'écobilan ne comporte pas que des informations sur le produit et son impact écologique. Il est aussi possible de consulter les produits appartenant à la même famille et le parcours du produit entre le producteur et le point de vente. Une légende est aussi proposée pour que le consommateur sache à quoi correspondent les labels affichés.

Bien que l'écran de l'iPhone permette d'afficher un certain nombre d'éléments, toutes ces informations ne pouvaient pas raisonnablement être disposées sur une seule vue. C'est la raison pour laquelle une barre de boutons a été insérée au fond de l'écran, et permet de passer d'une vue à une autre. Pour éviter des temps d'attente dans les changements de vues, toutes les données sont chargées en mémoire à l'initialisation, ce qui permet d'avoir des transitions instantanées.

La vue principale affiche quelques informations sur le produit et son étiquette énergétique. Il s'agit ici d'un affichage traditionnel dans une table, rien de bien compliqué. Elle affiche aussi un bouton pour ajouter ce produit aux favoris, nous y reviendrons. Mais surtout, cette vue gère l'affichage des autres vues, que nous appellerons les sous-vues. En effet, la barre de boutons est contenue dans cette vue et c'est elle qui intercepte l'évènement lorsque l'utilisateur clique sur un de ces boutons. A ce moment-là, le contrôleur de cette vue détermine quel bouton a été appuyé, initialise la vue correspondante, et l'affiche au-dessus d'elle-même, grâce à la méthode « addSubview ».

```
//Load the selected view
if(currentView != nil)
    [self.view addSubview:currentView.view];
```

Cellules de table customisées

La classe « UITableView » qui permet l'affichage de tables propose déjà 4 styles de cellules. Il s'agit des mises en forme que nous avons l'habitude de voir dans les applications iPhone. Mais lorsque l'on désire obtenir un affichage plus personnalisé, il est obligatoire de créer ses propres cellules. C'était le cas pour nous avec l'affichage de l'étiquette écologique et de la description du produit.

Le style des cellules utilisées dans notre affichage comprend un libellé à gauche, et un texte à droite. Or, l'étiquette écologique est une image, et la description est sur plusieurs lignes, sans libellé. Deux cellules customisées ont alors été créées. Une pour permettre d'afficher l'image du label à la place du contenu texte, et une pour afficher la description en s'adaptant au nombre de lignes. Pour ce faire, deux nouvelles classes ont été réalisées, « RankingCell » et « DescCell ». Celles-ci héritent de « UITableViewCelle » et redéfinissent les éléments d'une cellule et leur position. Cela nous permet d'obtenir le résultat suivant (voir image 27).

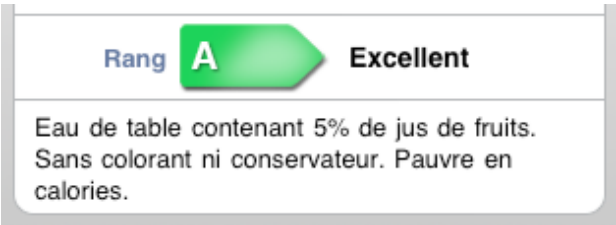


Image 27 - Cellules de table customisées

Ajout d'un produit aux favoris

Nous avons vu précédemment comment ajouter un magasin favori. Cette fois, pour ajouter un produit, la procédure est exactement la même ! En effet, il suffit de créer un objet « History » et d'appeler la méthode d'insertion de notre classe « DBManager », et c'est cette dernière qui gère la manipulation des données selon la propriété « type » de notre objet. C'est fois, cette propriété est initialisée à la valeur « DB_TYPE_RESULT_FAV » qui permet de dire que cet historique est un produit favori. Le magasin et le produit en cours sont toujours récupérés depuis nos variables globales.

```
//Add to history
DBManager *dbManager = [[DBManager alloc] initDatabase];
ax22_Store *store = [[Globals sharedInstance] currentStore];
ax22_EcoResult *ecoResult = [[Globals sharedInstance] currentEcoResult];

History *histo = [[History alloc] init];
histo.productId = [ecoResult.productId intValue];
histo.productEan13 = ecoResult.productEan13;
histo.productName = ecoResult.productName;
histo.storeId = [store.id_ intValue];
histo.storeName = store.name;
histo.storeLat = [store.latitude doubleValue];
histo.storeLong = [store.longitude doubleValue];
histo.creationDate = [NSDate date];
histo.type = DB_TYPE_RESULT_FAV;

int res;

res = [dbManager insertHistory:histo];
```

3.2.4.5 Affichage de la famille d'un produit

Image 28 - Vue famille

La vue famille est très utile puisqu'elle permet de connaître quels produits appartiennent à la même catégorie que le produit qui nous intéresse, et ainsi voir où celui-ci se situe par rapport aux autres. De plus, il est possible en sélectionnant un produit de la famille d'accéder à son détail.

Cette vue est très simple, elle ne fait qu'afficher les données qui ont été chargée au préalable par la vue principale (vue « produit ») dans une table. Ces données proviennent elles aussi du web service.

Points importants

- Récupération de la famille d'un produit
- Affichage du produit sélectionné

Récupération de la famille d'un produit

Bien que les données de la vue famille soient chargées à l'initialisation de la vue principale, nous allons voir ici comment sont récupérées les informations sur les produits de la même catégorie.

Le service web est une nouvelle fois mis à contribution pour accéder à ces données. En effet, il y a une méthode qui n'a pas encore été utilisée mais qui va nous servir à présent. Celle-ci prend comme paramètres l'id du produit dont on veut connaître la famille, ainsi que la latitude et la longitude du magasin. Elle va ensuite nous retourner une liste de produits de type « EcoResult ».

Voyons donc ci-dessous l'initialisation de cette vue famille, et l'affectation des données à afficher, récupérées par la méthode « computeEcoResultsByProductId ». Ici encore, le magasin et le produit courants proviennent des variables globales.

```
//Load family View
familyView = [[EcoResultFamilyViewController alloc] initWithNibName:@"EcoResultFamilyView" bundle:nil];
familyView.rootViewController = self;
familyView.ecoResultsData = [[NSMutableArray alloc] initWithArray:
                             [WSUtils computeEcoResultsByProductId:[result.productId intValue]
                                                          latitude:[store.latitude doubleValue]
                                                         longitude:[store.longitude doubleValue]]];
```

Affichage du produit sélectionné

Il est intéressant pour l'utilisateur de pouvoir afficher le détail d'un produit de la famille. Les données étant stockées dans un tableau, il suffit de déterminer l'indice de la ligne sélectionnée pour ensuite récupérer l'objet à cet indice, et le définir comme produit courant dans les variables globales.

Il faut maintenant revenir à la vue principale qui affiche le produit et la rafraîchir pour afficher les informations du produit sélectionné. Pour ce faire, la méthode « viewDidLoad » de la vue principale est appelée et la sous-vue famille, qui avait été affichée au-dessus, est supprimée par la fonction « removeFromSuperview ».

```
NSInteger row = [indexPath row];
ax22_EcoResult *result = [ecoResultsData objectAtIndex:row];
[Globals sharedInstance].currentEcoResult = result;

[rootViewController viewDidLoad];
[self.view removeFromSuperview];
```

3.2.4.6 Historique

Image 29 - Vue historique

L'historique est intéressant car il permet de revenir rapidement sur un produit que l'on a consulté dernièrement. Ainsi, chaque fois que l'utilisateur consulte un produit, les informations le concernant sont sauvegardées. Il fallait donc une nouvelle vue pour afficher cet historique et proposer de visualiser à nouveau les produits qui s'y trouvent.

Un affichage dans une table a une nouvelle fois été choisi. Les données proviennent de la base SQLite, et y sont insérées à chaque consultation d'un produit. Ces manipulations se font aisément grâce à la classe « DBManager ».

Point important

- Création et affichage de l'historique

Création et affichage de l'historique

L'ajout à l'historique est très simple, nous l'avons déjà abordé avec l'ajout de favoris pour les magasins et les produits. Là encore, il suffit de créer un objet « History » et de l'insérer grâce au « DBManager » avec la propriété qui définit le type. En l'occurrence, il s'agit ici du cas normal, une vraie entrée d'historique de type `DB_TYPE_HISTO`. Ca n'est pas plus compliqué que cela, en plaçant ce code à l'initialisation de la vue « Produit », nous aurons une entrée dans l'historique pour chaque produit consulté !

```
//Add to history (each time we read a product)
DBManager *dbManager = [[DBManager alloc] initDatabase];

History *histo = [[History alloc] init];
histo.productId = [result.productId intValue];
histo.productEan13 = result.productEan13;
histo.productName = result.productName;
histo.storeId = [store.id_ intValue];
histo.storeName = store.name;
histo.storeLat = [store.latitude doubleValue];
histo.storeLong = [store.longitude doubleValue];
histo.creationDate = [NSDate date];
histo.type = DB_TYPE_HISTO;

[dbManager insertHistory:histo];
```

Rien de très sorcier non plus pour afficher cet historique. Il est lu grâce à la classe « DBManager » et stocké dans une variable. Il n'y a plus qu'à lancer la méthode « reloadData » pour remplir la

table de notre vue historique. Un clic sur un ligne permet de visualiser le produit sélectionné, comme cela a été fait pour la vue famille.

```
//Get data from database
DBManager *dbManager = [[DBManager alloc] initDatabase];
[dbManager readHistory:DB_TYPE_HISTO];
self.data = dbManager.historyArray;
[dbManager release];

[self.dataTableView reloadData];
```

3.2.5 Fonctionnalités optionnelles

3.2.5.1 Affichage du parcours

Image 30 - Vue parcours

Cette vue parcours était optionnelle, mais elle apporte une réelle plus-value à l'application, en permettant à l'utilisateur de se rendre compte de manière visuelle du parcours effectué par un produit du producteur jusqu'à son point de vente. Elle peut permettre de sensibiliser le consommateur à acheter des produits de proximité, s'il se rend compte que le produit qu'il avait l'habitude d'acheter vient de loin ou fait de grands détours qui sont néfastes pour l'environnement.

Cette vue affiche les intermédiaires selon le même principe que les magasins sont affichés sur la carte pour la géolocalisation. Toutefois, il a fallu ajouter un service nous permettant de récupérer ces intermédiaires.

Point important :

- Récupérer les intermédiaires du trajet
- Afficher l'itinéraire

Récupérer les intermédiaires du trajet producteur - consommateur

Nous n'avions donc pas à disposition les données du trajet entre le producteur et le consommateur, bien que celles-ci existent dans la base de données. Il a donc été décidé d'étendre le service web afin de pouvoir extraire ces informations, d'autant que nous pouvons stocker ces intermédiaires dans la classe existante « Store ».

La nouvelle méthode « getTransportSteps » fait alors son apparition. Celle-ci prend comme paramètre l'id du produit en question, ainsi que l'id du magasin. Cela permet de rechercher tous les intermédiaires pour ce produit, menant à ce magasin. Chacun de ces intermédiaires est stocké dans un objet « Store » puisqu'ils partagent la même structure, un magasin étant soi-même un intermédiaire. Le résultat retourné est un tableau contenant ces objets. Ce tableau est ensuite parcouru afin de créer une annotation pour chaque intermédiaire et ainsi l'afficher sur la carte.

```
public static Store[] getTransportSteps(int productId, int retailerId) { }
```

Afficher l'itinéraire

Le but était aussi de pouvoir offrir à l'utilisateur l'itinéraire entre ces intermédiaires pour qu'il se rende compte de la durée des trajets, et leur impact environnemental. Plusieurs pistes ont été explorées pour afficher cet itinéraire.

La première était de l'afficher directement sur la carte. Malheureusement, aucune API ne propose pour l'instant une solution pour le faire facilement. Il aurait alors fallu trouver un moyen de récupérer tous les points GPS du parcours, et de dessiner une ligne qui traverserait tous ces points. Etant donné qu'il s'agissait ici d'une fonctionnalité optionnelle, et que le but n'était pas d'y consacrer trop de temps, cette première piste a rapidement été abandonnée.

La deuxième piste abordée est d'utiliser l'appel à l'application native Maps. En effet, iOS reconnaît certains types d'URL, et lance l'application qui y est associée. Ainsi, en lançant une URL comme « http://www.google.com/maps?g=London », Maps sera lancée et affichera la carte de Londres. Une adresse URL a donc été générée pour pouvoir afficher l'itinéraire passant par nos intermédiaires (voir les paramètres de Google Maps[12]). Si celle-ci s'affichait parfaitement dans le navigateur internet Safari, mais le résultat n'était vraiment pas satisfaisant dans l'application Maps dès que le parcours devenait quelque peu conséquent. Le trajet était en effet incomplet, avec des parties « à vol d'oiseau », des détours… Cette solution a par conséquent été aussi abandonnée.

Reste la dernière piste, celle qui a été finalement mise en place. Comme le trajet affiché dans Safari grâce à l'URL générée plus haut fonctionnait bien, il fallait trouver un moment de l'intégrer dans notre application. Et c'est à présent chose faite, grâce à une vue web (classe « UIWebView ») qui permet, comme son nom l'indique, d'afficher une page web. A présent, lorsque l'utilisateur clique sur le lien, une page internet Google Maps s'affiche avec l'itinéraire, tout en restant dans l'application !

3.2.5.2 Affichage des détails d'un produit et d'un producteur

Affichage des détails d'un produit

Un des points optionnels était d'afficher des détails sur le produit analysé, et pas seulement l'étiquette écologique. Mais comme nous l'avons vu plus haut, cette étape a été réalisée en même temps que l'affichage de l'écobilan. En effet, la vue produit affiche non seulement le label et l'impact écologique du produit, mais aussi les informations sur le produit comme son producteur, le pays, sa description. Cela a été rendu possible par la modification de l'objet « EcoResult ».

Affichage d'un producteur

L'idée de départ était, grâce à la librairie « CodeKit », de récupérer l'adresse internet du producteur du produit identifié. L'institut Icare[13] l'a fait dans leur application « CodeIcare » qui utilise aussi cette librairie. L'application aurait alors pu proposer d'accéder à cette adresse pour obtenir plus d'informations sur ce producteur.

Malheureusement, la version de la librairie qui nous avait été fournie ne contenait pas cette fonctionnalité. Vu les démarches administratives nécessaires pour obtenir l'accès à cette fonctionnalité, par rapport au gain qu'elle aurait rapporté, il a été convenu de ne pas aller plus loin dans cette direction. Toutefois, le producteur faisant partie des informations stockées dans la base

[12] Google Mapki, *Google Map Parameters,* http://mapki.com/wiki/Google_Map_Parameters, juin 2010
[13] Institut Icare, TechnoArk 3, 3960 Sierre, http://www.icare.ch/

de données pour un produit, il a été décidé d'afficher au moins de nom de ce producteur, à condition que celui-ci soit renseigné.

3.2.5.3 Favoris

Comme décrit précédemment, l'utilisateur à la possibilité d'ajouter des magasins ou des produits à une liste de favoris. Cette fonctionnalité n'a pas été évoquée dans le cahier des charges, c'est pourquoi elle se trouve dans les optionnelles.

L'idée de proposer des favoris a germé lors de la réalisation de l'historique. En effet, les données à stocker pour les deux fonctionnalités étant sensiblement les mêmes, il était facile de le mettre en œuvre. Cette gestion des favoris, qui a été détaillé plus haut, permet à l'utilisateur de sélectionner très rapidement les éléments qu'il préfère, et cela lui évite de répéter à chaque fois le même processus complet pour consulter l'étiquette écologique d'un produit.

3.2.5.4 Paramétrage

Image 31 - Vue paramètres

Il est important de permettre à l'utilisateur de paramétrer l'application selon ses désirs ou ses besoins. Ils peuvent le faire en passant par la partie « Réglages » de l'iPhone, ou directement par l'option « Paramètres » de l'application Green-T. Pour l'instant, les variables qu'il est possible d'éditer sont le rayon de recherche des magasins, qui détermine à quelle distance de notre position la recherche est effectuée, et le nombre d'éléments affichés dans l'historique.

Le SDK de l'iPhone permet de créer un « Settings Bundle » pour définir les paramètres à éditer. Une nouvelle vue a été créée pour y accéder et les modifier directement depuis l'application.

Points importants :

- Création des préférences avec un « Settings Bundle »
- Accéder et modifier ces préférences depuis l'application

Création des préférences avec « Settings Bundle »

Il est possible de créer un « Settings Bundle » pour une application. Celui-ci permet d'ajouter une entrée dans les réglages de l'iPhone pour l'application. Il se compose du fichier « Root.plist », qui est une liste de propriétés, et des fichiers de langues. La première étape consiste donc à définir les paramètres qui nous intéressent dans le fichier « Root.plist », ainsi que certaines propriétés comme les bornes maximale et minimale. Ci-dessous, un extrait où l'on voit comment est définir le slider pour paramétrer le rayon de recherche des magasins. Ce slider sera donc accessible depuis les réglages de l'iPhone. Nous avons fait de même pour le slider concernant le nombre d'éléments à afficher dans l'historique.

Key	Type	Value
Root	Dictionary	(2 items)
StringsTable	String	Root
PreferenceSpecifiers	Array	(4 items)
Item 0	Dictionary	(2 items)
Item 1	Dictionary	(7 items)
Type	String	PSSliderSpecifier
Key	String	radius_pref
DefaultValue	Number	2
MinimumValue	Number	0.1
MaximumValue	Number	20
MinimumValueImage	String	
MaximumValueImage	String	
Item 2	Dictionary	(2 items)
Item 3	Dictionary	(7 items)

Image 32 - Extrait du fichier de propriétés "Root.plist"

Accéder et modifier les préférences depuis l'application

Mais le but était que l'utilisateur n'ait pas besoin de quitter l'application pour modifier ces paramètres. Une nouvelle vue paramètres à alors été créée pour permettre d'y accéder depuis le menu de l'application. Cette vue contient les deux sliders permettant de modifier les paramètres. Ces deux sliders devaient ensuite être initialisés selon les bornes définies dans le fichier de propriétés. Nous pouvons récupérer la valeur de ces bornes dans un tableau « NSArray » de la manière suivante :

```
//Get the bundle path
NSString *bPath = [[NSBundle mainBundle] bundlePath];
NSString *settingsPath = [bPath stringByAppendingPathComponent:@"Settings.bundle"];
NSString *plistFile = [settingsPath stringByAppendingPathComponent:@"Root.plist"];

//Get the Preferences Array from the dictionary
NSDictionary *settingsDictionary = [NSDictionary dictionaryWithContentsOfFile:plistFile];
NSArray *preferencesArray = [settingsDictionary objectForKey:@"PreferenceSpecifiers"];
```

Une fois que les éléments sont initialisés, il faut récupérer les valeurs courantes de nos propriétés. Et celles-ci se trouvent dans les « NSUserDefaults », une classe implémentée selon le singleton pattern, et qui stocke les préférences utilisateurs. Ci-dessous, un extrait qui démontre comment récupérer ces préférences.

```
//Get defaults
defaults = [NSUserDefaults standardUserDefaults];

//Set values
float radiusValue = [defaults floatForKey:@"radius_pref"];
radiusSlider.value = radiusValue;
radiusLabel.text = [[[NSString alloc] initWithFormat:@"%.1f km", radiusValue] autorelease];

float nbElemValue = [defaults floatForKey:@"nb_elements_histo"];
nbElemSlider.value = nbElemValue;
nbElemLabel.text = [[[NSString alloc] initWithFormat:@"%.0f", nbElemValue] autorelease];
```

Une fois que la vue paramètres a été initialisée, il est donc possible pour l'utilisateur de modifier ses préférences, simplement en faisant glisser les sliders. Un évènement récupère la valeur lorsqu'elle est modifiée, et l'enregistre dans les préférences, toujours en utilisant la classe « NSUserDefaults ». La ligne ci-dessous suffit à le faire.

```
[defaults setFloat:radiusSlider.value forKey:@"radius_pref"];
```

La dernière remarque concerne la première initialisation des valeurs dans « NSUserDefaults ». En effet, celles-ci sont initialisées avec la valeur par défaut définie dans le fichier de propriétés, seulement une fois que l'utilisateur a été dans les réglages de l'iPhone. S'il ne le fait pas avant de

démarrer l'application, les valeurs de préférences seront donc nulles. Pour pallier à ce problème, une vérification est faite au démarrage, et si les valeurs ne sont pas initialisées, le fichier de propriétés est parcouru pour affecter la valeur par défaut à ces préférences.

3.2.6 Fonctionnalités abandonnées

3.2.6.1 Identification du produit par mode « Image »

Lors de l'élaboration du cahier des charges, il avait été discuté d'éventuellement intégrer l'identification du produit par reconnaissance d'image, sur laquelle l'institut informatique de l'HES-SO travaille. Bien que ce projet soit en bonne voie et continue d'avancer, il n'était pas suffisamment au point pour finalement envisager de l'intégrer.

3.2.6.2 Partage d'une analyse avec un contact

Cette fonctionnalité optionnelle devait permettre d'ajouter une plus-value sociale à l'application Green-T, en permettant de sensibiliser un de ses contact à tel ou tel produit en lui envoyant l'étiquette écologique d'un produit et ses détails.

Un certain nombre d'heures ayant été consacré à la réalisation des favoris, cette fonctionnalité a été abandonnée par manque de temps. Toutefois, elle fera certainement partie des améliorations futures, qui devraient mettre l'accent sur l'aspect social du projet, notamment par le partages des informations entre amis ou sur des réseaux sociaux.

4 Compte-rendu

4.1 Réalisé VS planifié

Il est temps de faire le point sur les fonctionnalités qui ont été réalisée par rapport à ce qui avait été planifié.

Au niveau des fonctionnalités de base, elles ont toutes été implémentées selon la description du cahier des charges. En effet, le consommateur peut procéder à toutes les étapes permettant de consulter l'étiquette écologique d'un produit qui sont, pour rappel, la géolocalisation, l'identification du produit, la consommation du service web, et l'affichage du bilan environnemental du produit et de sa famille complète. L'historique, qui était la dernière fonctionnalité obligatoire, est aussi proposé.

Pour ce qui est des fonctionnalités optionnelles, une grande partie a aussi été réalisée, et même plus puisque les favoris, qui n'étaient pas prévus dans le cahier des charges, sont disponibles. Les fonctionnalités qui ont été abandonnées sont l'identification du produit par le mode image, puisque celui-ci n'était pas encore assez élaboré, et le partage d'une analyse avec un contact par manque de temps. Par contre, les informations sur les produits et le consommateur sont fournies, et le parcours producteur - consommateur est aussi proposé.

Le bilan est donc positif car parmi toutes les fonctionnalités planifiées, seules deux optionnelles n'ont pas été implémentées, et une supplémentaire a fait son apparition.

4.2 Utilisabilité / Convivialité

L'application a été testée par 3 personnes dont les profils étaient très éloignés. La première est un utilisateur avancé de l'iPhone qui maîtrise aussi le développement sur ce mobile. La deuxième possède aussi un iPhone, mais en a une utilisation beaucoup plus basique, et n'est pas du tout confronté au domaine de l'informatique et des communications. La dernière n'utilise pas l'appareil d'Apple au quotidien et n'est donc pas une habituée de ses interfaces. Il leur était demandé de donner un avis sur l'utilisabilité et la convivialité de l'application, et de voir quel était leur comportement face à elle.

Tous trois, en prenant plus ou moins de temps, ont réussi à consulter le bilan environnemental d'un produit et on assimilé le concept général du projet, ce qui démontre que l'application est abordable pour tous types d'utilisateurs. De plus l'aide, qui fait office de guide de démarrage, est utile pour connaître les fonctionnalités qui sont à disposition de l'utilisateur.

L'interface leur a semblé agréable et claire, bien qu'un petit bémol ait été relevé concernant le menu. En effet, la remarque qui est revenue plusieurs fois concerne l'ajout d'un nouveau produit qui n'est pas suffisamment mis en valeur et qui a amené ces testeurs à se diriger d'abord vers l'historique ou les favoris, qui étaient bien entendu vides. Il faudrait donc guider l'utilisateur vers cette option de menu lors du premier démarrage, ou alors la mettre plus en évidence.

Au final, le bilan est prometteur avec des remarques positives de bon augure pour la suite du projet. Les quelques points négatifs, qui concernent plutôt l'aspect pratique et cosmétique, peuvent être rapidement améliorés.

4.3 Améliorations

4.3.1 Techniques

Une application en production est en perpétuelle évolution. Il y a toujours des points à améliorer, des demandes d'utilisateurs à combler, de la maintenance à effectuer. Dans l'état actuel de l'application Green-T, deux points à améliorer ressortent.

Le premier concerne la gestion des exceptions. L'application a été développée sur un iPhone 3GS, mais rien ne nous assure qu'il fonctionne de la même manière sur d'autres appareils, comme l'iPod Touch et l'iPad. Il serait nécessaire de faire des tests sur la présence de l'appareil photo, sans lequel il n'est bien sûr pas possible de scanner des codes-barres, ou de la connexion internet, obligatoire pour accéder aux données à travers le service web.

Concernant le deuxième, il s'agit des performances. La récupération des données peut prendre un certain temps lorsqu'il y en a une certaine quantité, comme lorsque l'on recherche les nombreux produits d'une même famille. Il y a certainement des actions à entreprendre pour améliorer ce temps de réponse.

4.3.2 Sociales

L'aspect social du projet Green-T peut apporter une plus-value non négligeable à l'application. En effet, en permettant au consommateur de partager les produits qui l'intéressent et qui sont écologiques, non seulement cela permet de sensibiliser ses contacts, mais en plus de mettre Green-T sur le devant de la scène.

Plusieurs moyens peuvent être envisagés pour le faire. Cela commence avec le partage d'une analyse avec un contact, comme cela était envisagé dans les fonctionnalités optionnelles. Mais cela ne touche qu'un minimum de gens. Le partage sur les réseaux sociaux est une option qui revient souvent dans tous types d'applications et qui remporte un joli succès. Ce serait une excellente opportunité pour faire la promotion du projet Green-T et de son concept. Enfin, un système de notifications qui permettrait aux consommateurs intéressés d'être tenus au courant des nouveautés respectant l'environnement où des promotions sur certains produits favorables ne pourrait qu'ajouter de l'intérêt à l'application pour ses utilisateurs.

5 Problèmes rencontrés

5.1 Accès à la base de données

Le serveur web sur lequel était hébergé la base de données du projet se trouvait dans les locaux du TechnoArk à Sierre. Comme je ne travaillais pas sur place, je n'avais pas d'accès direct à ce serveur. Dans un premier temps cela n'a pas été un problème puisque le développement de l'application iPhone ne nécessitait pas de modifier l'infrastructure mise en place.

Par contre, lorsqu'il a fallu adapter les services web ou apporter des modifications sur les données présentes dans la base, cet aspect est devenu gênant dans la mesure où il fallait pour chaque modification passer par un intermédiaire. Cela a ralenti l'avancement du développement, en particulier le week-end, où il n'y a personne pour répondre à mes requêtes. Toutefois, grâce à la précieuse collaboration de Simon Martin, cela s'est passé sans impact majeur sur la réalisation du projet.

5.2 Dépendance envers la librairie « CodeKit »

La librairie « CodeKit », fournie par le RFID Center de Sierre, nous a permis de pouvoir proposer le scannage des codes-barres dans l'application Green-T. Cela a été un grand avantage dans le sens où nous n'avons pas eu à le développer nous-mêmes, surtout que la librairie fonctionne très efficacement.

Malgré tout, l'application est totalement dépendante de cette librairie et du RFID Center. Cela peut devenir un vrai problème dans le cas où il y aurait des adaptations à apporter ou pour la mise à jour des versions. D'ailleurs, plusieurs semaines après la sortie d'iOS4, la librairie n'est toujours pas compatible, et le RFID Center n'a pas démontré une grande réactivité jusqu'à présent.

5.3 Gestion de la mémoire

Avant de me lancer dans le développement pour iPhone, je ne m'étais jamais inquiété de la gestion de la mémoire dans mes programmes. Les autres langages que j'utilise, en particulier Java, possèdent un garbage collector qui gère la libération des objets, donc pas besoin de s'en soucier, surtout qu'il ne s'agissait pas de plateformes mobiles.

Je me suis très vite rendu compte que ce n'était pas le cas avec Objective-C, et cela m'a tout de même causé quelques grandes prises de tête. Il m'est arrivé un bon nombre de fois de vouloir accéder à une variable qui avait déjà été libérée, ou que l'application plante sans avertissement à cause d'un problème de mémoire.

Il semblerait que tous les développeurs qui travaillent avec des pointeurs soient passés par là. Mais j'en retire une bonne expérience, puisqu'avec un peu de rigueur et d'habitude, ce problème s'est bien estompé tout au long du projet, pour devenir anecdotique au moment de la rédaction de ce rapport.

6 Gestion du projet

6.1 Organisation

Le travail de bachelor a début le 26 novembre 2009 suite à la remis des sujets pour se terminer le 19 juillet 2010, date à laquelle le travail doit être rendu. Dans cet intervalle, 360 heures doivent être effectuées, ce qui représente 8 semaines de travail à plein temps, mais qui sont étalées sur plusieurs mois pour les étudiants à temps partiels, étant donné qu'ils continuent leur activité professionnelle en parallèle.

6.2 Planification

6.2.1 Planification initiale

Le planning détaillé a été établi en tenant compte de la date imposée pour la fin du travail. Les heures à disposition ont été réparties en différentes tâches afin d'avoir une vision clair du temps à passer sur chacune des étapes de la réalisation. Cette répartition démontre que presque 60% du temps est consacré à l'implémentation des fonctionnalités, ce qui est réaliste pour un projet technique comme celui-ci. Le reste du temps comprend environ 20% d'initiation et de recherche, qui sont indispensables lorsque l'on démarre avec un langage et des outils inconnus, 15% de gestion de projet et 7% consacrés aux tests de fonctionnement de l'application.

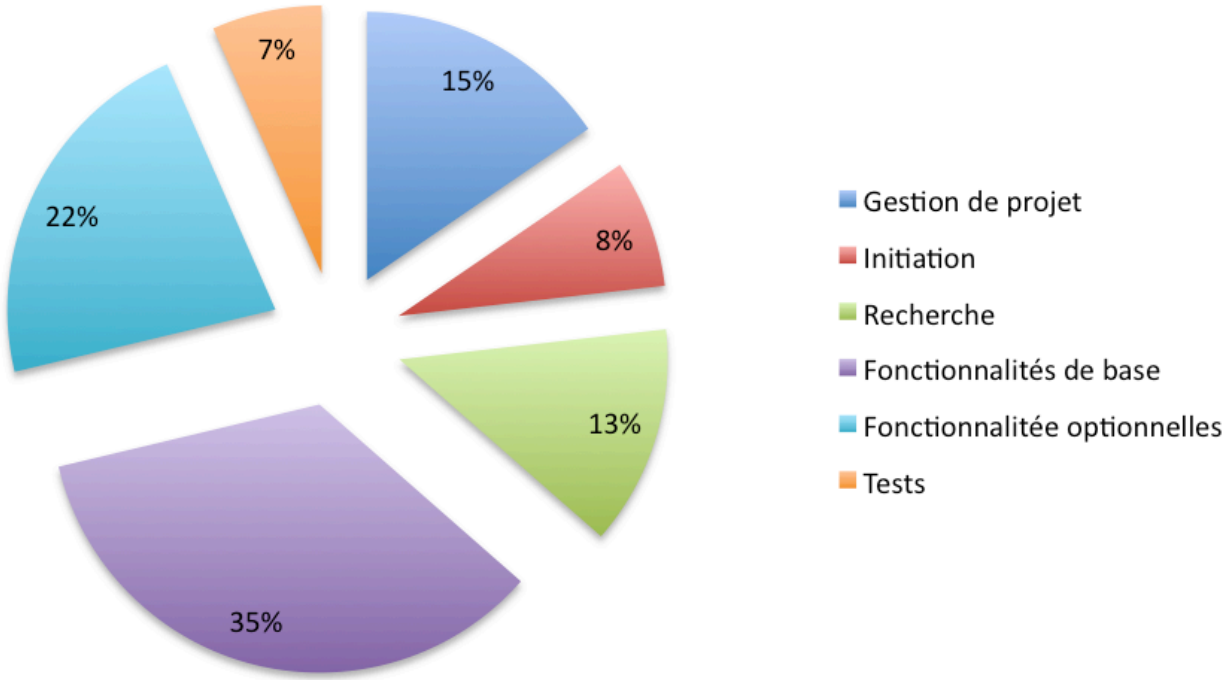


Image 33 - Répartition des heures de la planification initiale

Afin de respecter certaines échéances et ne pas prendre un retard qui pourrait être compromettant pour la réalisation du projet, des jalons ont été fixés. Ainsi, il avait été prévu de consacrer 3 semaines au début du projet pour s'initier aux outils et à l'Objective-C, suivies d'un peu plus de 2 semaines pour la recherche sur les fonctionnalités à implémenter. Ensuite, le développement à proprement parler était prévu sur 22 semaines, soit 13 pour les fonctionnalités

de base, et 9 pour les fonctionnalités supplémentaires. Restait alors 2 semaines pour la documentation et les tests. A noter que les activités de gestion de projet sont des tâches étalées sur toute la durée du projet. Le planning détaillé est fourni en annexe.

6.2.2 Gestion du temps

Le travail de bachelor a démarré très lentement en début d'année à cause de mon activité professionnelle extrêmement chargée qui m'a pris beaucoup de temps, notamment sur des heures planifiées pour ce travail de bachelor, mais aussi par une certaine difficulté à entrer pleinement dans le projet. Un retard relativement conséquent a été pris jusqu'à la fin mars, où ma situation s'est un peu calmée.

A partir d'avril, un effort a été fait pour rattraper ce retard, et il a été progressivement comblé jusqu'au mois de juin, où une semaine prise sur les heures supplémentaires du début d'année m'a permis de respecter le dernier jalon fixé au 2 juillet et qui désignait la fin du développement. Restait alors une semaine pour finaliser et rédiger ce rapport.

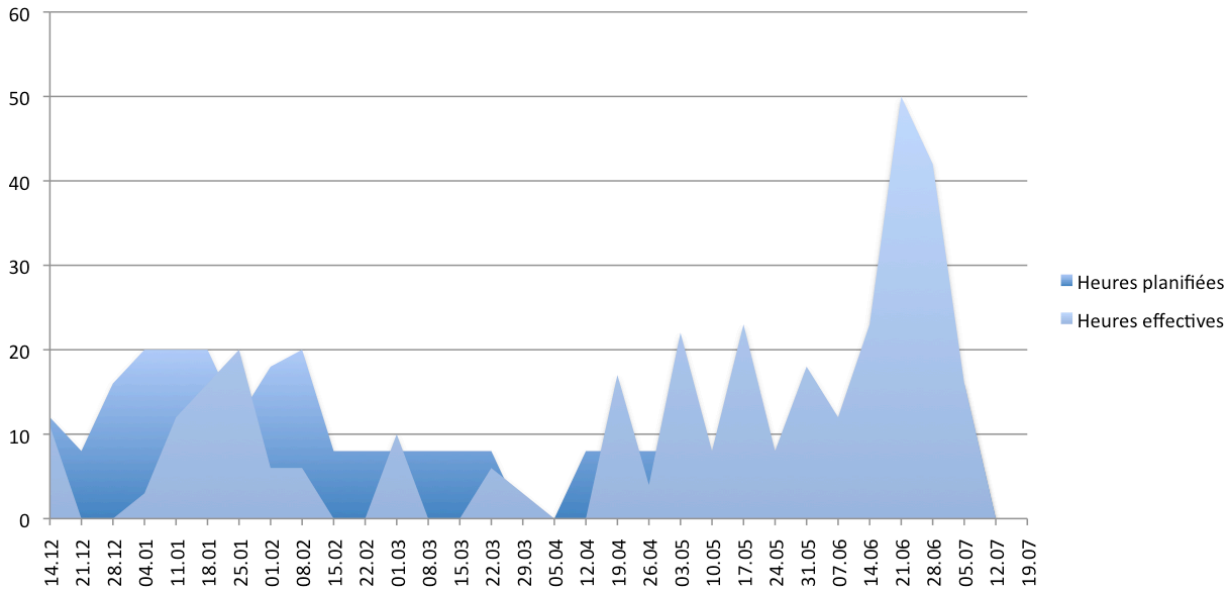


Image 34 - Heures effectuées VS heures planifiées

7 Conclusion

Avant de collaborer au projet Green-T, je ne m'étais jamais soucié de savoir quel était l'impact de mes achats lorsque j'allais au magasin. La réalisation de ce travail de bachelor m'a fait prendre conscience qu'au quotidien, il est possible de faire des gestes pour limiter la dégradation de notre environnement. Et ça commence par ce que l'on consomme. Le premier constat est clair, dorénavant l'étiquette environnementale s'ajoutera aux critères qui déterminent les produits que j'achète.

Découvrir et apprendre le langage et les outils nécessaires au développement pour les appareils d'Apple est un atout non négligeable, tant la demande pour cette plateforme est grandissante. Cette expérience est donc très enrichissante et pourrait avoir son importance pour ma future carrière dans l'informatique.

Le projet Green-T a de réelles opportunités d'imposer son concept dans un avenir proche. Le domaine de l'écologie est complètement dans la tendance actuelle, et de plus en plus de personnes s'y intéressent. Il y a fort à parier que Green-T peut son sortir son épingle du jeu, et notamment grâce à l'application iPhone réalisée dans ce travail de Bachelor. C'est ce sentiment de participer à un projet prometteur qui rend le travail excitant et motivant.

Toutefois, il faut souligner l'importance de trouver des partenaires afin de relever le défi qui consiste à peupler la base de données, sans quoi le projet ne pourra jamais atteindre son objectif final.

Déclaration sur l'honneur

Je déclare, par ce document, que j'ai effectué le travail de bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après :

- HENNEBERT Jean, Professeur HES, HES-SO // Valais
- MARTIN Simon, Adjoint Scientifique II, HES-SO // Valais
- EGGEL Ivan, Adjoint Scientifique I, HES-SO // Valais
- KENZELMANN Pierre, RFID Center
- BÜRGISSER Philippe, RFID Center

Lieu et date	Signature
_____	_____
	(Yann METRAILLER, Etudiant HES-SO)

8 Lexique

Terme	Définition
RCSO	Réseau de Compétences HES-SO
Ra&D	Recherche appliquée et Développement
SDK	Software Development Kit, ensemble d'outils pour le développement
IDE	Integrated Development Environment - Environnement de développement
Firmware	Micro logiciel, programme interne de l'iPhone
Framework	Ensemble de bibliothèques permettant le développement d'applications
iOS	iPhone Operating System
SQLite	Moteur de base de données relationnelles directement intégré aux programmes

9 Références

9.1 Sources Internet

Page Wikipedia de l'iPhone
http://en.wikipedia.org/wiki/IPhone

Page Wikipedia de iOS
http://fr.wikipedia.org/wiki/IOS_(Apple)

Hello There: A CoreLocation Tutorial
http://www.mobileorchard.com/hello-there-a-corelocation-tutorial/

iPhone SDK 3.0 – Playing with Map Kit
http://blog.objectgraph.com/index.php/2009/04/02/iphone-sdk-30-playing-with-map-kit/

Using iPhone SDK MapKit Framework – A tutorial
http://mithin.in/2009/06/22/using-iphone-sdk-mapkit-framework-a-tutorial/

Projet wsdl2objc
http://code.google.com/p/wsdl2objc/

Appel de web services SOAP sur Iphone
http://www.sebhoerner.fr/?p=127

gSOAP
http://www.cs.fsu.edu/~engelen/soap.html

how to debug EXC_BAD_ACCESS on iPhone
http://www.codza.com/how-to-debug-exc_bad_access-on-iphone

iPhone Up - La communauté des développeurs iPhone
http://www.ipup.fr/index.php

iPad/iPhone forward geocoding API using Google gecoding service
http://blog.sallarp.com/ipad-iphone-forward-geocoding-api-google/

GMGeocoder
http://code.google.com/p/gmgeocoder/

Google Map Parameters
http://mapki.com/wiki/Google_Map_Parameters

iPhone Default User Settings Null?
http://paulsolt.com/2009/06/iphone-default-user-settings-null/

Learn Objective-C
http://cocoadevcentral.com/d/learn_objectivec/

9.2 Bibliographies

Référence
MARK D. et LAMARCHE J., *Beginning iPhone Development : Exploring the iPhone SDK*, Apress, 2009

10 Tables des illustrations

11 Annexes

Annexe	Définition
Cahier des charges	Cahier des charges définissant les objectifs du projet
Planning détaillé	Planning détaillé des heures planifiées et effectives

11.1 Annexe 1 - Cahier des charges

Développement d'un client iPhone pour le calcul de bilan énergétiques de produits de grande consommation
- Cahier des charges -

Introduction

Contexte

Le travail de bachelor s'inscrit dans le cadre du projet RCSO Green-T (Ra&D) mené par l'institut d'informatique de gestion de la HES-SO // Valais. Le projet RCSO Green-T s'inscrit dans la tendance actuelle de l'écologie et du développement durable en proposant aux consommateurs le bilan écologique des produits qu'ils achètent. Concrètement, grâce au code-barres imprimé sur chaque produit, le consommateur à l'aide de son téléphone mobile récupère directement une notice lui indiquant clairement l'évaluation écologique du produit qu'il souhaite acheter (voir Figure 1).

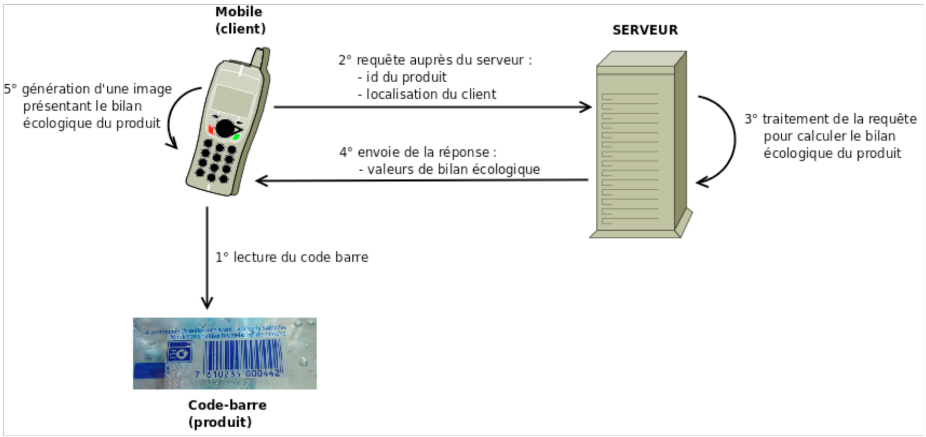


Figure 1. Architecture générale du projet Green-T[1]

[1] Source : rapport du travail de bachelor de Damien Zufferey, 2008

Cette phase du projet traite principalement du développement de la partie client sur iPhone, mais couvre aussi les aspects d'intégration et de configuration de l'architecture ayant déjà été mise en place auparavant.

Ce projet d'une durée de 360 heures s'effectue dans le cadre d'un travail de bachelor en informatique de gestion à la HES-SO // Valais.

Architecture existante

L'architecture est multi-tiers et comprend, outre le client mobile, un serveur web TomCat qui est lié à une base de donnée MySQL, et un service web permettant d'accéder aux informations. Voici en détail ci-dessous la solution mise en place.

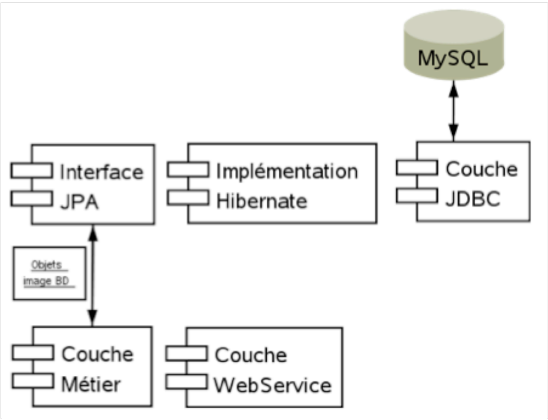


Figure 2. Architecture multi-tiers existante[2]

Client iPhone

L'iPhone est une véritable révolution sur le marché des smartphones grâce, entre autres, à son interface constituée d'un écran tactile multipoint remplaçant les boutons ou claviers traditionnels. L'objectif est d'utiliser les capacités de cet appareil novateur pour proposer une application à la fois conviviale et pratique au service du projet RCSO Green-T. De façon concrète, la pénétration de l'iPhone au niveau du marché suisse et mondiale est devenue très significative. Selon un récent article de la plate-forme informative ICTjournal,[3] le volume des ventes d'iPhone en Suisse a

[2] Source : rapport du travail de Bachelor de Damien Zufferey, 2008
[3] www.ictjournal.ch, *Ruée sur l'iPhone: la Suisse est sa plus grande fan!*, 17 décembre 2009

atteint 600'000 unités sur 2009. Cette même source précise encore que "la demande ne faiblit pas et notre pays détient le record mondial du nombre d'iPhones par habitant".

Fonctionnalités de base

1. **Géolocalisation**

Il sera possible de déterminer l'emplacement de l'utilisateur de **deux manières**. La première consiste à utiliser la fonctionnalité de l'iPhone qui permet de localiser l'appareil. En effet, grâce au framework « Core Location » il est possible de rechercher la position de l'iPhone.

L'iPhone 3G/3GS utilise le GPS assisté (ou A-GPS). Cela veut dire qu'il existe plusieurs moyens de localiser l'appareil, outre l'utilisation des satellites GPS. En fonction du modèle et des services à disposition, une combinaison entre les réseaux cellulaires, Wi-Fi et GPS est utilisée. Il faut noter que la méthode de détection utilisée par l'appareil est transparente pour l'utilisateur et le développeur. Le service de localisation est très bien détaillé sur la page de support d'Apple[4], ainsi que sur la page web "iPhone Secrets"[5].

Une fois que la latitude et la longitude ont été déterminées, nous pourrons utiliser un service mis à disposition par le web service et qui nous retourne la liste des lieux de vente à proximité.

Public Store[] findStoresByLocation(double latitude, double longitude)

Dans le cas ou cette première option serait indisponible ou ne fonctionnerait pas correctement (en intérieur par exemple), l'utilisateur aura la possibilité de faire une recherche textuelle du lieu de vente. L'application permettra la saisie du lieu et du magasin recherché, qui seront ensuite passés comme paramètres à un second service qui retourne une nouvelle fois les lieux de vente à proximité.

Public Store[] findStoresByCityName(String City, String Name)

[4] support.apple.com/kb/HT1975, *iPhone and iPod touch: Understanding Location Services*, décembre 2009
[5] www.edepot.com/iphone.html, *iPhone Networking / A-GPS*, 2009

2. Identification du produit

Là aussi, plusieurs méthodes seront à la disposition de l'utilisateur pour identifier le produit qu'il désire évaluer. Dans la première méthode, il s'agit de scanner le code-barre afin d'en extraire le numéro du produit pour l'identifier. Le centre de compétence Icare (http://www.icare.ch) a développé une application qui permet d'effectuer cette opération. Une demande a été fait à cet institut quant à la possibilité de nous fournir une librairie utilisable depuis l'iPhone. En cas de refus, il faudra alors se pencher vers une alternative, telle que le portage de la libraire existante Java ou C/C++ vers Objective-C. Il s'agit ici du point le plus délicat du projet, car cette fonctionnalité est un aspect primordial de l'application.

La deuxième solution réside dans une recherche textuelle. Il sera en effet possible de retrouver le produit en saisissant directement son numéro d'identification EAN-13. Optionnellement, nous envisagerons d'identifier le produit par mots-clés. Cette dernière option implique d'étendre le web service de manière à offrir une fonction qui retourne les produits correspondants aux mots-clés.

Le dernier procédé n'est pas encore confirmé, car il concerne une fonctionnalité qui n'est pas encore aboutie au niveau des développements actuels du RCSO Green-T. Cette fonctionnalité consiste à effectuer une reconnaissance du produit par image. Sur le principe, l'utilisateur prendrait une photographie du produit, et l'application tenterait de retrouver de quel produit il s'agit sur la base de ce cliché. Il sera décidé en cours de développement si cette méthode sera implémentée au niveau du client iPhone, mais elle constitue quoi qu'il arrive un atout indéniable pour l'application.

3. Consommation du web service

Nous l'avons vu précédemment, l'application fera appel à un web service pour accéder aux données du serveur. Actuellement, il propose 3 fonctions.

```
Public Store[] findStoresByLocation(double latitude, double longitude)

Public Store[] findStoresByCityName(String City, String Name)

Public EcoResult computeEcoResult(String ean13Product, double latitude,
                                  double longitude)
```

La consommation d'un web service est constituée de 3 étapes:

- Envoyer le message
- Réceptionner la réponse
- Parser la réponse

Il s'agit donc ici de se documenter et d'effectuer quelques essais pour se familiariser avec l'utilisation de web services depuis l'iPhone.

Ensuite, nous pourrons nous attacher à récupérer le bilan écologique en utilisant la dernière fonction qui nous retourne le résultat après lui avoir passé le numéro d'identification du produit, la latitude et la longitude.

4. Affichage de l'étiquette énergétique

L'iPhone est très intéressant en ce qui concerne l'affichage des informations grâce à son grand écran et à la multitude de possibiltés qu'il offre. C'est pourquoi il convient de proposer l'étiquette énergétique d'un produit de manière claire et agréable.

L'utilisateur aura au minimum les vues suivantes à disposition.

- Détail d'un produit avec étiquette énergétique, description, informations sur le bilan écologique, etc…
- Liste des produits d'une même « famille » pour une comparaison de leur étiquette énergétique

Fonctionnalités supplémentaires

Le second objectif de l'application est d'apporter un plus par rapport à ces fonctionnalités de base. Voici ci-dessous les possibilités supplémentaires qu'il est prévu d'implémenter.

1. Affichage des informations sur la société

A l'image de l'application réalisée par le centre de compétence Icare (voir plus haut), il peut-être intéressant pour l'utilisateur de savoir quelle est l'entreprise qui fournit le produit concerné et de pouvoir accéder au site internet de cette société. C'est la raison pour laquelle, dans la mesure du possible, les informations sur le producteur seront aussi proposées à l'utilisateur, qui pourra ensuite atteindre directement le site grâce à un lien.

Affichage du parcours

Afin d'avoir une vision claire du chemin qu'un produit parcourt avant de se retrouver dans les rayons de notre magasin préféré, il est intéressant de pouvoir visualiser sur une carte son trajet entre le lieu de production et le lieu de consommation.

C'est cette fonctionnalité que nous souhaitons proposer, en utilisant l'application « Maps » intégrée à l'iPhone et qui permet d'afficher une carte interactive au sein de notre client.

2. Paramétrage

Il est possible que certains paramètres de l'application soient variables et à choix de l'utilisateur. Dans ce cas, il faut prévoir une interface permettant de les modifier.

Contraintes

Il est primordial de prendre en compte les contraintes liées au développement sur mobile, et en particulier sur celles spécifiques à l'iPhone pour éviter de rencontrer des problèmes de performance ou d'intégrité des données. Ces contraintes sont très bien expliqueés dans le premier chapitre du livre « Beginning iPhone Developpment » aux éditions Apress, Novembre 2008.

1. Une seule application

Contrairement au développement sur les système d'exploitation, une seule application peut être démarée à la fois. Cela pourrait évoluer dans le futur, mais actuellement il faut bien se rendre compte que notre application est la seule dont le code est executé lorsque elle est lancée, et que si l'utilisateur interagit avec une autre autre application, il est impossible de faire quoi que ce soit sur la notre.

2. Accès limité

Apple limite les données auquelles notre application a accès. Nous pouvons lire et écrire uniquement dans l'espace reservé pour l'application sur le système de fichier. Cet emplacement est nommé « sandbox ».

3. Temps de réponse limité

L'iPhone se doit d'être réactif, et en attend donc autant des applications installées. L'Application doit se charger rapidement et être opérationnelle aussi vite que possible. De même lorsque l'utilisateur quitte l'application, il faut sauvergarder les données et fermer l'application en moins de 5 secondes, sous peine de voir le processus être tuer et de s'exposer à des pertes de données.

4. Taille de l'écran limitée

Lorsque l'iPhone a été introduit, il disposait de la plus grande résolution disponible sur un appareil mobile. Mais il n'est finalement pas si grand, et il faut s'adapter à cette dimension de 480x320 pixels.

©HES-SO // Valais, juillet 2010

5. **Ressources limités**

Grâce à son interface graphique et les nombreuses possibilités offertes par l'iPhone, il est très facile d'utiliser la totalité de la mémoire à disposition. Les modèles actuels disposent de 128MB ou 256MB (3GS) de RAM, mais une partie est utilisée par les processus système. En général, il reste environ la moitié de cette mémoire pour notre application, il convient donc d'y être attentif.

Tâches du planning

Gestion de projet

- ✓ Planning et suivi du projet
- ✓ Cahier des charges
- ✓ Documentation

Acquisition de compétences et analyse

- ✓ Se familiariser avec l'existant Green-T
- ✓ Se familiariser avec le développement sur iPhone
- ✓ Utilisation de la géo localisation sur iPhone
- ✓ Méthodes d'identification du produit
- ✓ Consommation de web services

Modélisation et implémentation sur l'iPhone

Par fonctionnalités :

- ✓ Modélisation de l'interface du client
- ✓ Implémentation client
- ✓ Tests de la fonctionnalité

Fonctionnalités faisant partie du cahier des charges

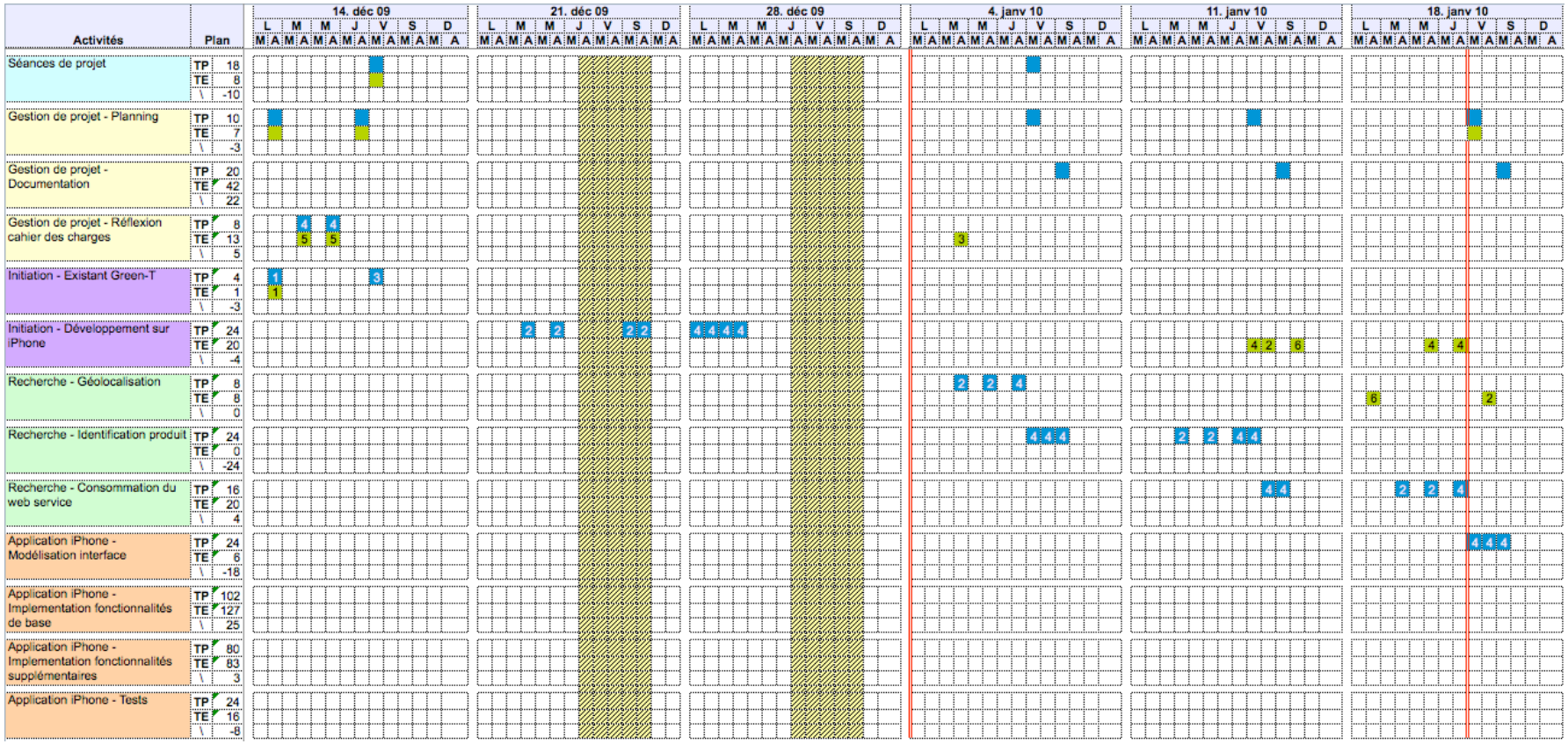
- ✓ Modélisation objet et API côté client pour la consommation des web services
- ✓ Identification du produit – mode texte
- ✓ Identification du produit – mode code-barre
- ✓ Géolocalisation – mode GPS

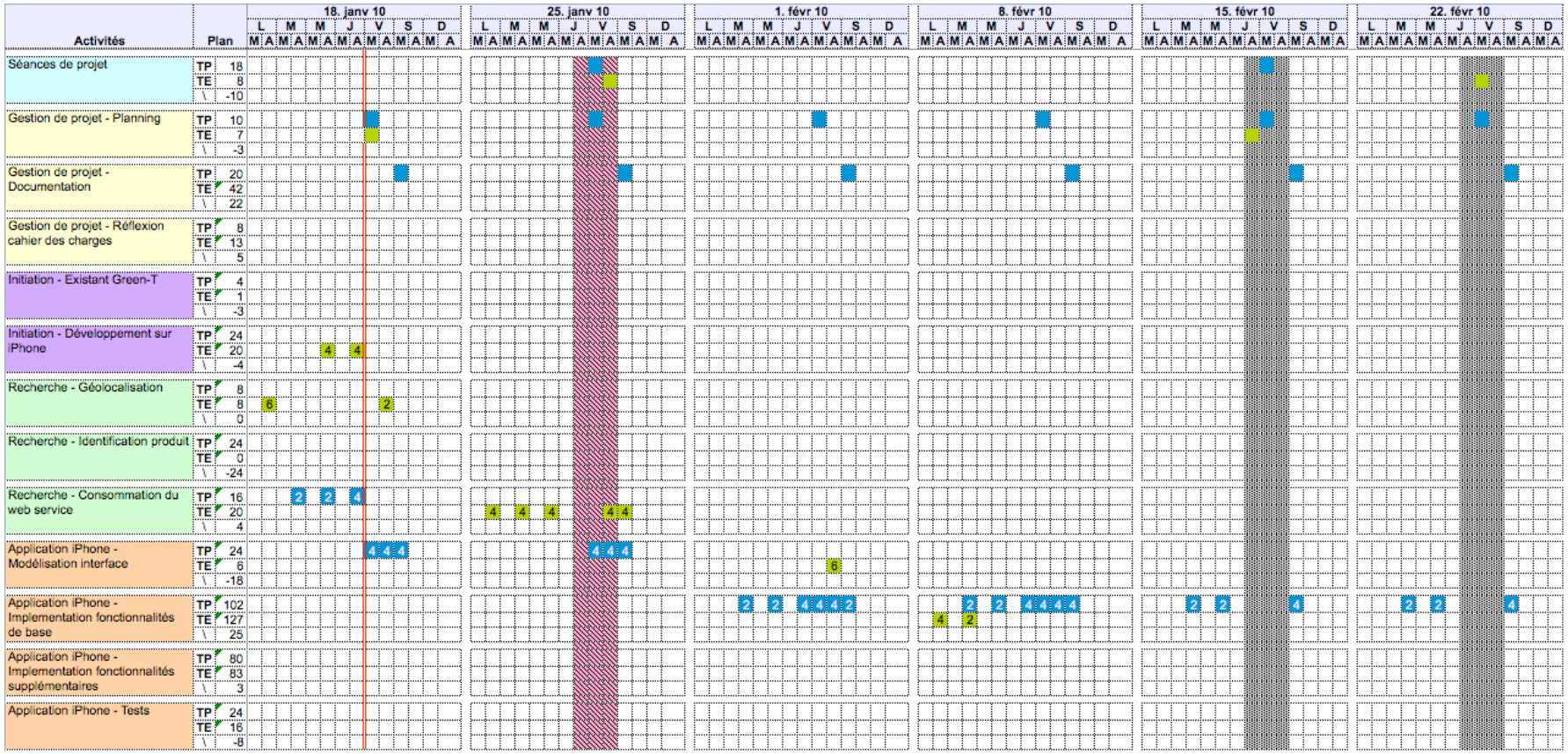
- ✓ Géolocalisation – mode texte
- ✓ Affichage du bilan environnemental pour un produit
- ✓ Affichage du bilan environnemental de chaque produit pour une famille de produit
- ✓ Historique des recherches

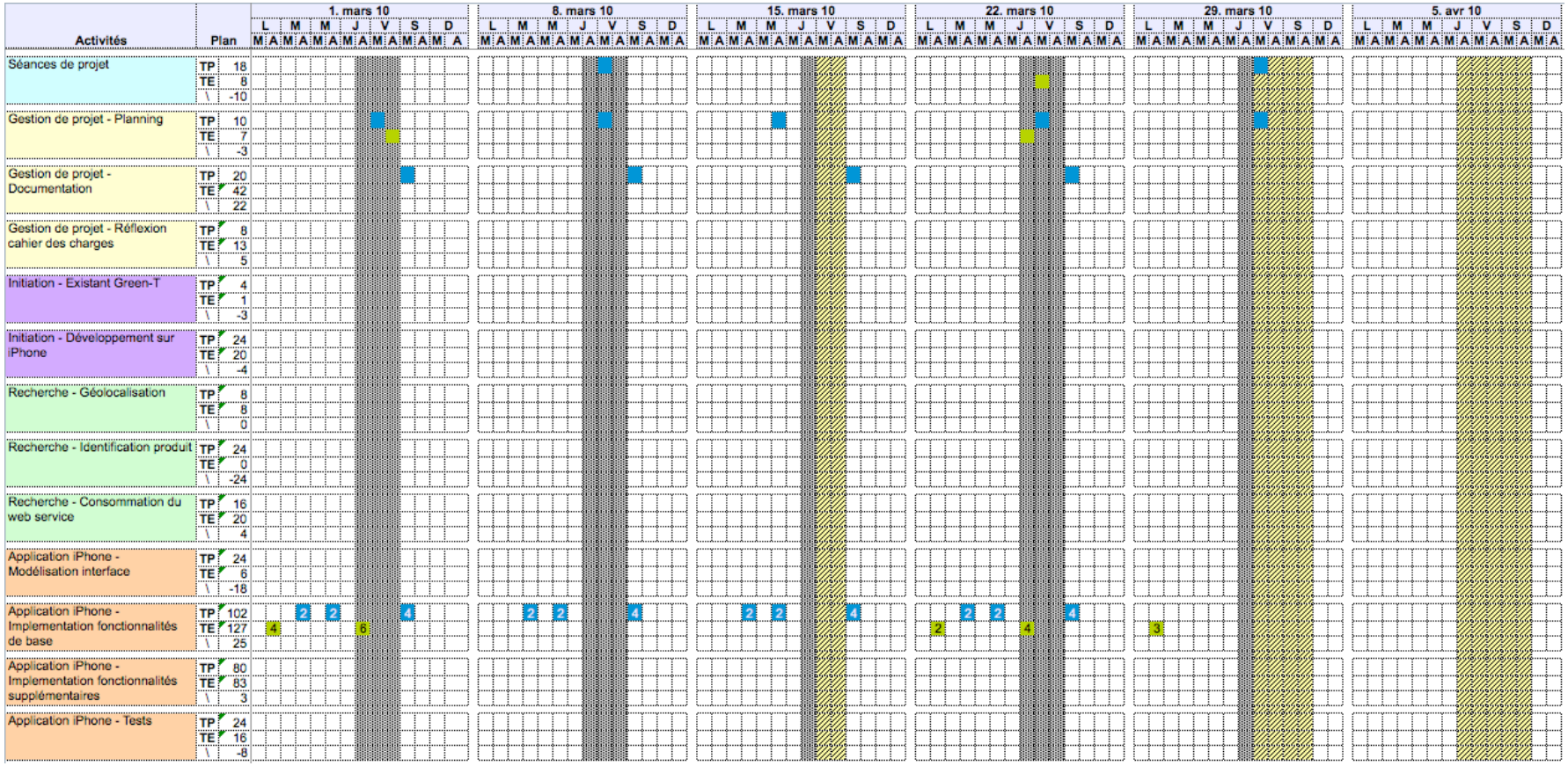
Fonctionnalités optionnelles

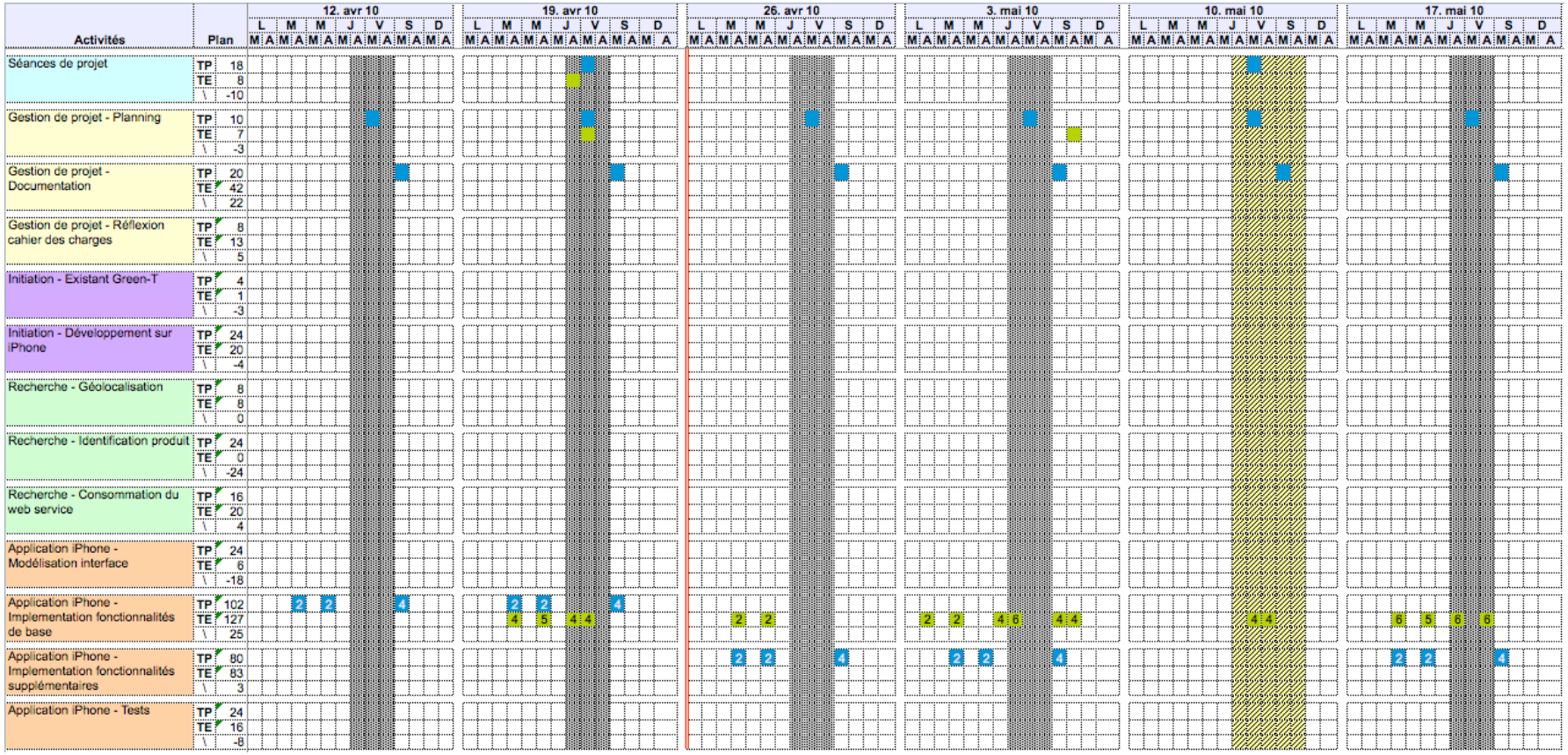
- ✓ Identification du produit – mode image
- ✓ Affichage des informations d'un produit et d'un producteur
- ✓ Partage d'une analyse avec un contact (envoi d'un email)
- ✓ Afficher un parcours producteur - consommateur sur une carte pour un produit donné

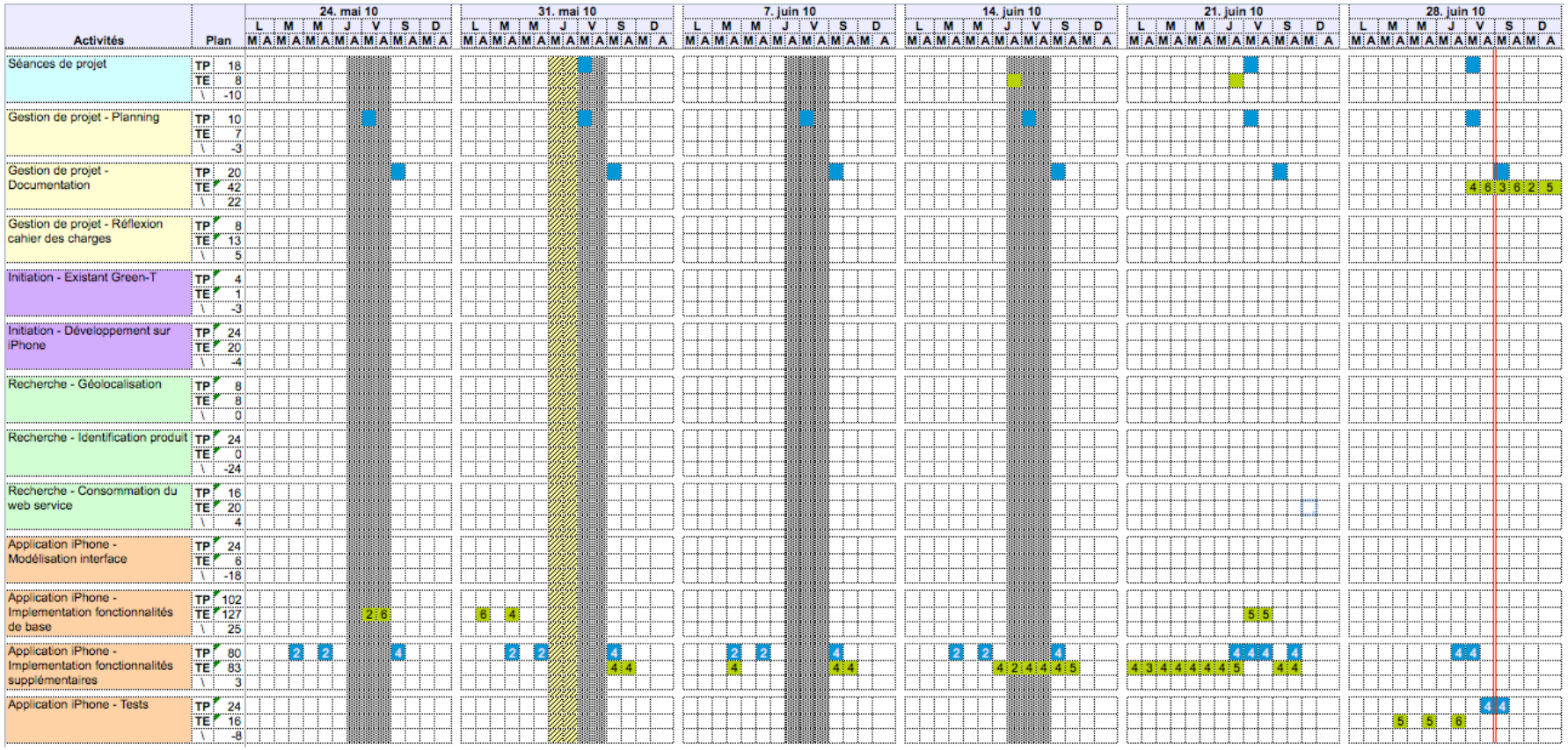
11.2 Annexe 2 - Planning




Activités
Plan
18. janv 10
25. janv 10
1. févr 10
8. févr 10
15. févr 10
22. févr 10
Séances de projet
Gestion de projet - Planning
Gestion de projet - Documentation
Gestion de projet - Réflexion cahier des charges
Initiation - Existant Green-T
Initiation - Développement sur iPhone
Recherche - Géolocalisation
Recherche - Identification produit
Recherche - Consommation du web service
Application iPhone - Modélisation interface
Application iPhone - Implementation fonctionnalités de base
Application iPhone - Implementation fonctionnalités supplémentaires
Application iPhone - Tests


Activités
Plan
1. mars 10
8. mars 10
15. mars 10
22. mars 10
29. mars 10
5. avr 10
L M M J V S D
M A M A M A M A M A M A
Séances de projet
TP 18
TE 8
\ -10
Gestion de projet - Planning
TP 10
TE 7
\ -3
Gestion de projet - Documentation
TP 20
TE 42
\ 22
Gestion de projet - Réflexion cahier des charges
TP 8
TE 13
\ 5
Initiation - Existant Green-T
TP 4
TE 1
\ -3
Initiation - Développement sur iPhone
TP 24
TE 20
\ -4
Recherche - Géolocalisation
TP 8
TE 8
\ 0
Recherche - Identification produit
TP 24
TE 0
\ -24
Recherche - Consommation du web service
TP 16
TE 20
\ 4
Application iPhone - Modélisation interface
TP 24
TE 6
\ -18
Application iPhone - Implementation fonctionnalités de base
TP 102
TE 127
\ 25
Application iPhone - Implementation fonctionnalités supplémentaires
TP 80
TE 83
\ 3
Application iPhone - Tests
TP 24
TE 16
\ -8


Activités
Plan
12. avr 10
19. avr 10
26. avr 10
3. mai 10
10. mai 10
17. mai 10
L M M J V S D
M A M A M A M A M A M A M
Séances de projet
TP 18
TE 8
\ -10
Gestion de projet - Planning
TP 10
TE 7
\ -3
Gestion de projet - Documentation
TP 20
TE 42
\ 22
Gestion de projet - Réflexion cahier des charges
TP 8
TE 13
\ 5
Initiation - Existant Green-T
TP 4
TE 1
\ -3
Initiation - Développement sur iPhone
TP 24
TE 20
\ -4
Recherche - Géolocalisation
TP 8
TE 8
\ 0
Recherche - Identification produit
TP 24
TE 0
\ -24
Recherche - Consommation du web service
TP 16
TE 20
\ 4
Application iPhone - Modélisation interface
TP 24
TE 6
\ -18
Application iPhone - Implementation fonctionnalités de base
TP 102
TE 127
\ 25
Application iPhone - Implementation fonctionnalités supplémentaires
TP 80
TE 83
\ 3
Application iPhone - Tests
TP 24
TE 16
\ -8


Activités
Plan
24. mai 10
31. mai 10
7. juin 10
14. juin 10
21. juin 10
28. juin 10
Séances de projet
Gestion de projet - Planning
Gestion de projet - Documentation
Gestion de projet - Réflexion cahier des charges
Initiation - Existant Green-T
Initiation - Développement sur iPhone
Recherche - Géolocalisation
Recherche - Identification produit
Recherche - Consommation du web service
Application iPhone - Modélisation interface
Application iPhone - Implementation fonctionnalités de base
Application iPhone - Implementation fonctionnalités supplémentaires
Application iPhone - Tests

Activités		Plan	5. juil 10												12. juil 10												19. juil 10											
			L		M		M		J		V		S		D		L		M		M		J		V		S		D		L		M		M		J		V		S		D	
			M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A
Séances de projet	TP	18																																										
	TE	8																																										
	\	-10																																										
Gestion de projet - Planning	TP	10																																										
	TE	7																																										
	\	-3																																										
Gestion de projet - Documentation	TP	20																																										
	TE	42	4		4		4		4																																			
	\	22																																										
Gestion de projet - Réflexion cahier des charges	TP	8																																										
	TE	13																																										
	\	5																																										
Initiation - Existant Green-T	TP	4																																										
	TE	1																																										
	\	-3																																										
Initiation - Développement sur iPhone	TP	24																																										
	TE	20																																										
	\	-4																																										
Recherche - Géolocalisation	TP	8																																										
	TE	8																																										
	\	0																																										
Recherche - Identification produit	TP	24																																										
	TE	0																																										
	\	-24																																										
Recherche - Consommation du web service	TP	16																																										
	TE	20																																										
	\	4																																										
Application iPhone - Modélisation interface	TP	24																																										
	TE	6																																										
	\	-18																																										
Application iPhone - Implementation fonctionnalités de base	TP	102																																										
	TE	127																																										
	\	25																																										
Application iPhone - Implementation fonctionnalités supplémentaires	TP	80																																										
	TE	83																																										
	\	3																																										
Application iPhone - Tests	TP	24							4	4	4	4																																
	TE	16																																										
	\	-8																																										